Mica: A Web-Search Tool for Finding API Components and Examples

Jeffrey Stylos, Brad A. Myers
Computer Science Department, Human-Computer Interaction Institute
Carnegie Mellon University, Pittsburgh PA 15213 USA
{ jsstylos, bam }@cs.cmu.edu
http://www.cs.cmu.edu/~mica

Abstract

Because software libraries are numerous and large, learning how to use them is a common and problematic task for experienced programmers and novices alike. Internet search engines such as Google have emerged as important resources to help programmers successfully use APIs. However, observations of programmers using web search have revealed problems and inefficiencies in their use. We present a new prototype search tool called Mica that augments standard web search results to help programmers find the right API classes and methods given a description of the desired functionality, and help programmers find examples when they already know which methods to use. Mica works by using the Google Web APIs to find relevant pages, and then analyzing the content of those pages to extract the most relevant programming terms and to classify the type of each result.

1. Introduction

Software libraries, frameworks and application programming interfaces (APIs) have grown larger and more complex, and applications have grown more dependent on them. For example, there are more than 34,000 classes and methods in the Java SDK, and more than 140,000 classes, methods properties and fields in Microsoft's .NET framework. In addition to their large size, these and other frameworks are changing and growing every year, making it impossible for even experienced developers to remember everything.

Because of their size and fluidity, it is no longer possible, for example, to get a complete printed reference for Microsoft's APIs (such a document would take tens of thousands of pages). Electronic documentation can be similarly unwieldy: a search for "time" on Microsoft's MSDN help returns more than 500 separate documentation pages.

Figure 1. The Mica web application. Mica includes a keyword sidebar on the left, which is generated from web search results shown on the right. Search result pages are categorized by their content; the Java icon indicates that those results contain code.

Learning how to use these APIs presents several barriers [11]: understanding how the APIs are structured, selecting the appropriated classes and methods, figuring out how to use the selected classes, and coordinating the use of different objects together all pose significant difficulties. Some of the difficulty of the selection barrier comes from the fundamental *vocabulary problem* [7]. A particular programming concept can be described in multiple ways and no one word will best describe it for all programmers. This is especially a problem when APIs are too large to easily browse.

In a formative study, the programmers we observed found web search especially useful for APIs, and were often able to use web search to overcome each type of learning barrier. They were able to do so because of the wide range of documents that the search engines indexed – documentation, forum discussions, code snip-

0-7695-2586-5/06 $20.00 © 2006 IEEE

pets and the source code for full programs – and also because of the effective ranking algorithms that made the most useful and relevant information in very large collections most likely to be found. This helped programmers overcome selection barriers by finding the right terminology from their naïve phrasing. This worked because the search engines indexed forums and other pages on which people had described a problem or solution using the same, incorrect, terminology.

However, while search engines were a popular and often a valuable tool for programming help, programmers encountered a number of problems and inefficiencies using them. One challenge was that the documents that programmers spent their time looking at were often not relevant. Programmers would get frustrated when they scanned long result pages only to find that they did not contain the source code they were looking for, or that they contained only source code and not the higher level documentation they needed. Sometimes the results had nothing to do with programming at all. Even when a search did yield some relevant results, if the first few documents programmers browsed did not seem relevant, they would often give up and try another path. The less familiar programmers were with the domain, the less successful they were at predicting how relevant a document would be.

The difficulties that programmers experienced are not surprising given that web search engines were not designed to specifically support the programming task. While there are several simple improvements that could aid programmers, such more control over searching of punctuation marks and other programming syntax, substantially improving programmers' experience requires a new type of tool that uses knowledge of programmers' behavior to provide more effective online programming support.

Mica (**M**aking **I**nterfaces **C**lear and **A**ccessible) is a prototype tool we designed to help programmers more effectively and efficiently use web search to learn how to use APIs (shown in Figure 1 and available at http://www.cs.cmu.edu/~mica). It does this by providing the following cues about the information contained in the result pages: (1) Mica displays relevant API methods, class and field names that are contained in the search results. (2) Mica orders the field names based on their frequency and correlation with the web results, and by the structural containment of classes and methods. (3) Mica displays icons that indicate whether a page contains source code and whether it is an official documentation page. (4) Mica provides keyword-relevant result summaries on demand.

The next section presents observations of programmers that motivated our work. Section 3 describes Mica in more detail, and discusses some of the implementation challenges and solutions. Section 4 presents early usage log analysis and Section 5 discusses related research. Section 6 looks at future research opportunities and Section 7 offers some concluding remarks.

2. Motivating Programmer Observations

Mica's design was motivated by our observations of programmers using Internet resources. We studied programming projects in various stages of creating new functionality and observed how the programmers used Internet resources to support their programming. In three case studies we found common patterns of usage, and different situations where the effectiveness of the tools broke down.

While the information seeking literature includes some studies of how programmers [2] and others [3, 12] use internet resources, these mostly look at more formal documentation, with an emphasis on implications for documentation writers. We were interested in also looking at how programmers used informal resources, such as forum and Usenet posts and independent websites with sample applications.

The Information Foraging theory [14] explains people's search behavior in terms of "information scent," cues that indicate how useful a path will be. We were interested in finding out which cues were most helpful to programmers in finding relevant information.

We observed three small programming projects in Java by computer science graduate students and a collection of screen-captures of Java programmers collected for another study [10]. The projects involved creating a new GUI Java application, creating an Eclipse plug-in, and modifying an existing but unfamiliar open source application.

2.1. Observed steps of API learning

In observing the programmers, we noticed several different stages of API learning and transitions between these stages (shown in Figure 2).

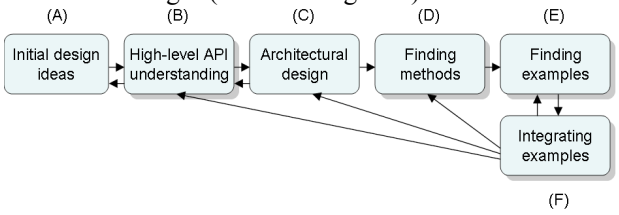

Figure 2. High-level programming activities observed in our study. Internet resources were used in steps (B), (D), (E) and (F); Mica is designed to help with these steps.

Each of the programmers started with an initial idea of what their application was to do (A). Only after get-

ting an overview of the structure of the APIs they would use (B) did the programmers begin to design how they would implement their application (C). These initial design ideas would sometimes require further high-level API understanding (B). Once they had a high-level design, they looked for specific methods that could accomplish their task (D). Once they found the name of a method in an API, they searched to find out exactly how to use it, using documentation and example code (E). They then integrated this example code into their own program and see if it accomplished what they wanted (F). If it did not, they would find new examples of how to use the same methods (E), look for new methods (D), redesign their architecture (C) or look for different APIs (B).

2.2. Internet resources used

Each of our observed programmers used Google as their primary resource for finding programming information on the Internet, with one also using Google Groups, Google's search engine for Usenet archives. The programmers found a variety of different types of resources using Google, including tutorial pages such as those on http://java.sun.com, documentation such as the Java SDK Javadoc pages, overviews and articles on software architectures, webpages with example programs, and forum posts with questions, answers and code snippets.

2.3. High-level API understanding (Step B)

Because the programmers we watched already had a good idea of what program they wanted to create (A), the first step we observed was that of getting an overview of which APIs they needed to use and those APIs' overall structure (B). Each programmer's first step in this task was to pose a general query to Google. For example, the programmer writing a Eclipse plug-in searched for "refactoring plugin eclipse". Google was effective at finding tutorials, high-level articles (such as those provided by IBM and Eclipse.com about Eclipse) and sample projects with source code and documentation that the programmers found useful.

One of the reasons that Google was effective at this task was because it was worked well even with the use of non-expert terminology. For example, the novice programmer creating a simple Java application was able to find information about creating windows and widgets using the search "creating a form in java," even though the word "form," which the programmer was familiar with from Visual Basic, is not often used in Java programming.

2.4. Discovering which methods to use (Step D)

When programmers had begun to understand the high-level elements of the APIs they were using (B), they then formed an idea of how they planned to implement their application (C). Their next action was to determine what specific classes and method calls they could use to accomplish specific tasks (D). Often this step was combined with the previous step (B) as the tutorial or article would include specific method references.

Programmers used different strategies in this step, including searching Google with a description of the desired functionality and browsing the list of classes in the JDK's Javadoc documentation. Because Google indexes many documentation sources, including the JDK's Javadocs, searching with Google could often double as a search of the official documentation. In addition, when programmers found a potentially useful method, they would often look up its official documentation to verify that it did what they thought.

When programmers performed a web search based on a description of the desired result, they would open and scan some of the resulting pages. They looked for code or words that seemed like they might be method or class names – looking for cues such as a fixed width font or programming punctuation. Often they spent time looking at pages that were not related to programming or which did not have a specific programming solution. Even when pages contained relevant information, programmers would often have a hard time finding it on the page, and sometimes falsely conclude that it was not there.

2.5. Finding examples (Step E)

Once programmers had found a method they thought might be useful (D), they then looked for specific code examples of how to call the method (E). This was usually done by searching Google, or Google Groups, with the name of the specific method, but was also sometimes combined with the previous step (D), or the previous two steps (B, D) when the search results already included code samples.

The Javadoc documentation usually did not provide examples of code use. Examples were used to answer such questions as: "How do I instantiate an instance of this method's class?", "How do I get variables of the appropriate types to pass as arguments?" and "At what point in my code should I call this method?"

The examples the programmers found were occasionally in the form of complete programs with a few lines of interesting code, but more often small code snippets without an accompanying full project.

2.6. Observation conclusions

The popularity and effectiveness of search engines like Google helped convince us that these are important sources of information for programming help tools. The search engines were effective often because of the wide variety of material they indexed, allowing for the successful use incorrect terminology to find correct answers. Because of this, we chose to build Mica on top of Google rather than have it search its own index as several other tools do [6, 13].

Where Google failed was in providing appropriate relevance cues for the information that programmers needed. Programmers wanted to find examples or documentation and placed heavy emphasis on the specific API terms that appeared, but Google's presentation of its results did not provide this information.

3. Mica

We designed Mica to provide the cues that the programmers in our observations needed to more effectively use web search. Mica identifies specific relevant methods and class names by loading and analyzing the result pages of a search, identifying the code-related terms, using frequency-based heuristics to determine which names are likely to be the most relevant to the programmer's specific search, and displaying the results in a sidebar of the webpage that dynamically updates as result pages are loaded and processed on the server. When pages are loaded by the server, they are classified as official documentation or as containing source code, and results of these types are given appropriate icons to guide the programmer. Mica currently finds keywords contained in the Java APIs, but is designed for a range of languages and APIs, for which we plan to add support in the future.

3.1. Search results

Mica uses the Google Web APIs [8] to generate its web search results. Doing so allows the same search options as Google, such as quoted and negated search terms, and the same quality of results. These search results are used both for generating the web search portion of the result page (right part of figure 1) and for obtaining the addresses of the result pages that are loaded and processed by Mica to generate the keyword sidebar.

Mica also uses the spell-check portion of the Google Web APIs to detect likely misspellings and suggest corrections (shown in Figure 3). While a programming specific spelling correction mechanism might offer some advantages, Google's system is based on its users' searches and word commonality and usually performs well for programming searches.

Figure 3. Mica uses the Google API's spelling correction to catch misspelled words.

3.2. Keyword sidebar

3.2.1. Keyword selection. The keywords Mica shows (in the left part of figure 1) are selected from all the words related to programming that are contained in any of the ten web search result pages. We chose to examine only the first ten result pages so that we could show programmers *where* each of the words came from, and because in prototypes of Mica we found that looking at more pages did not significantly improve the quality of the results. Currently, keywords are considered to be related to programming if they match any method, class and interface name from the Java SDK libraries. The list of Java programming names was generated in advance from JavaDoc annotations in the Java source code. However, we plan to expand our approach to support more languages and APIs, and to implement keyword selection using other clues (such as fixed width fonts and other page formatting) to avoid needing a precomputed complete list of known keywords, and to avoid falsely recognizing keywords that are also common words when they are used outside of a code context.

3.2.2. Keyword ranking. Because the search result pages typically contain hundreds of programming keywords, one key contribution of Mica is how it ranks the keywords to determine which to display, and in what order.

Intuitively, keywords that occur frequently in the search results but infrequently globally (across the whole Internet) are the ones most relevant to a programmer's specific search. We measure *search frequency* by the number of unique search result pages that contain the keyword, and *global frequency* by using a precomputed table of how many pages in Google's index of more than 8 billion pages contain each keyword. This table was precomputed for all of the Java API keywords using Google's "Google Suggest" feature. However, a simple ranking metric of *search frequency* divided by *global frequency* did not yield good results in our experimentation. Globally

uncommon keywords that happened to occur in one result would often rank highest. On the other hand, ranking metrics that more heavily weighted search result frequency tended to rank highest the common methods that were not specific to the search.

To compromise between these, we experimented with a threshold that filters out globally common keywords, and settled on a threshold value of 250,000. Words that occur on more than 250,000 of Google's more than 8 billion pages indexed are considered *globally common*, and are never suggested as answers in Mica's keyword sidebar. Using this threshold we found that ranking based on result frequency first and global infrequency second (to break ties) worked well. While a threshold has the disadvantage that common keywords are never suggested, we have found that to a surprising extent, the specific questions from programmers relate to tangible input or output and are answered by specific, relatively uncommon keywords. The very common keywords tend to be helper methods that are used across many different tasks, and specific task tend not to contain only common helper methods.

3.2.3. Keyword structure. In addition to ranking them based on frequency, the keywords that are in the same class are also grouped and indented beneath the enclosing class if the class is included in the results. For example, "GraphicsDevice" and "GraphicsEnvironment" are enclosing classes in Figure 1. To avoid hiding relevance, the higher of the rankings of a method and its enclosing class is used to determine the ranking of the aggregate result that contains the class and method(s).

3.2.4. Highlighting. When moused-over, the keywords in Mica's sidebar show programmers which results contain the keywords by highlighting their background (as shown in the bottom two web search results in figure 1). Similarly, mousing-over search results highlights in the sidebar keywords contained in that result page. The highlighting of keywords can be used to help more quickly reveal which of the displayed keywords are relevant given a search result that seems relevant. The highlighting of search results can also be used to quickly observe clusters of results that contain related solutions and conversely results that contain different solutions.

In addition to mouse-based highlighting, search results are displayed with a light grey background until they are processed by the server. Doing so gives the programmer feedback on the progress of the server and also helps reveal dead or unresponsive result pages that may not be worth clicking.

3.2.5. Keyword links. When a keyword is clicked, links for that term appear underneath it (the links under "setFullScreenWindow" in Figure 1).

The three links currently provided are: a new Mica query on that term, a new Mica query restricted to only Java source code files, and a link to the Javadoc definition of that class. These links are based on three commonly observed uses of keywords: formulating a new query with that term, looking for examples of that term, and looking up the official documentation for that keyword.

3.3. Summary generation

When a keyword is clicked, in addition to the appearance of new links, Mica generates new summaries for the search results that contain that keyword. The new summaries are displayed without affecting the rest of the page.

We designed this feature so that programmers could compare the different uses of a particular method without having to open up the result pages. A design challenge is selecting the context that makes a short summary most useful for a particular keyword. We currently begin a keyword-specific summary at the line with the first occurrence of the keyword on a page, but plan to explore other heuristics for choosing which instance of a keyword to show and choosing how much text before the keyword to include.

3.4. Icons for result attributes

Mica displays icons next to search results to represent the type of content some pages contain. For results that contain source code, it displays a Java icon, and for official documentation, it displays a Javadoc icon. The motivation for this feature came from observing programmers' use of Google.

Figure 4. Result icons for code and documentation.

When programmers wanted to find source code examples in our study (stage E in Figure 2), we observed them spending time opening and scanning result pages looking for code. For many searches, only two or three of the first ten results would contain code, and so this process was relatively slow and did not add to programmers' understanding.

To decide if a page contains source code, Mica currently uses a heuristic that if it contains two or more of the eight code keywords shown in the sidebar, it contains code. While we plan to use a more sophisticated algorithm, such as the robust code recognition algorithm in [15], this heuristic has worked surprisingly well. One reason for this is that because of the keyword ranking, the keywords are known to be globally uncommon, and so this helps avoid the detection of Java keywords that are also English words.

When programmers wanted to refer to the official documentation, we observed that they would often use Google to find the specific Javadoc reference page, even when the root documentation page was already bookmarked or open.

Mica decides if a page should be marked with the icon for official documentation by comparing the URL to Sun's API documentation site.

While Mica currently recognizes and displays an icon only for documentation and source code, we plan to extend it to recognize tutorials and forum discussions. Forum questions and answers might also be distinguished, but we feel it would be more useful combine these into one "discussion" category because in our observations, the questions themselves were often as useful as answers, and labeling them differently might unnecessarily discourage programmers from exploring them.

3.5. Implementation challenges

Mica is implemented as a collection of Java servlets that use the Google Web-APIs.

While creating a local index of content would have allowed much faster processing, we use the Google search results because of their high degree of relevance. Especially when helping programmers solve vocabulary problems, the effectiveness of any analysis will be limited by the quality of the initial rankings, and so dealing with a non-local index, as we do here, or trying to replicate the quality of Google's rankings is an important and difficult tradeoff for a programming web-search engine.

Because waiting for the result pages to download takes as long as thirty seconds, an architectural challenge was how to display keywords and result-type analysis dynamically, as each individual page is downloaded and processed.

The first implementation strategy that we tried was to use the `XMLHttpRequest` JavaScript object used by many recent dynamic webpages such as Google Maps and many different web-mail sites. In our currently implementation however, we instead use a servlet that continuously appends JavaScript code to the HTML page that overwrites earlier results as the page loads. The two main advantages of this approach are that it is compatible with more browsers, including those that do not yet fully support the `XMLHttpRequest` object, and that because the end result is a single regular HTML page, browsers are better able to cache it. This is particularly an issue when using the back button to revisit a Mica query in between result pages.

4. Usage logging and analysis

We have made Mica available for public use and have advertised its presence on several Java-related newsgroups. We have used the logs of its usage to help guide future directions of the tool's implementation.

Mica logs the queries as well as out going result clicks and uses of the sidebar. To log outgoing links, which do not usually involve any communication with the server of the source page and so are not trivially loggable, we use JavaScript to trap all click events and record the ones that correspond to outgoing links.

So far Mica has logged some two hundred queries from a hundred unique IP addresses.

4.1. Query types

One early finding was that while about half of the queries submitted to Mica appeared to involve a general topic or vocabulary search – for example, "load dll jar" or "date arithmetic," most of the remaining searches were for a specific known Java method or class name, such as "JSpinner" or "URLEncoder".

While we had observed programmers using known method names to search for documentation and examples, we had not expected the percentage of such searches to be so high. The usage of Mica in this way helped motivate the documentation recognition and links that provide further exploration from keywords. Figure 4 shows a handful of the searches that external programmers have posed to Mica since it has been available.

We would like to know how helpful Mica was for the programmers who issued the queries, and provided feedback links (shown in Figure 1), however these were used very rarely, and so this question may only be answerable by a lab study.

Specific	General
toUpperCase	concatenating strings
thread	weak references
WeakHashMap	lazy loading and caching
urlencoder	awt events
DeferredOutputStream	regular expressions
JTable	date arithmetic
JSpinner	load dll jar
class.forname	iterate array

Figure 4. A sample of the queries that programmers have posed to Mica.

4.2. Sidebar usage

Users clicked the sidebar in roughly half of all searches. Since we expected much of the sidebar's usefulness to be been in the learning the terms themselves, which does not necessarily require explicit user action, these numbers are not surprising. We have since expanded the functionality offered by clicking the keywords in the sidebar, making it easier to use them as a basis for new queries and providing a direct link to the documentation for each term.

5. Related Work

The Strathcona system [9] finds and recommends source code examples based on an IDE's current context. While it does not find examples from the Internet or allow explicit queries, it addresses the problem of discovering how to use APIs to perform a desired task from examples. Because its example search is implicit and based on the currently written code statements, it is unable to help in situations where programmers do not have any starting point. Mica helps programmers with the stage of finding the first few methods from which they could then make use of a tool like Strathcona.

Other related IDE tools such as Team Tracks [5] help programmers with how to use the internal APIs of large projects based on other programmers' IDE usage. While the learning task that this tool addresses is similar to Mica's, it is more suited to learning private code, about which there might be no information on the Internet, while Mica is more suited to public APIs or open source projects large enough to have Internet discussion sites.

The observation of programmers by Steven Clarke and the Visual Studio User Experience group at Microsoft has yielded several interesting and relevant results, including three different programming personas [4]. These personas capture the different learning styles and intents of programmers. For example, "opportunistic" programmers are much more likely to look for example code to work from, while "systematic" programmers are more likely to want to read documentation first. These personas help motivate Mica's differentiation of different types of information so that each persona can avoid unnecessary browsing of results.

The techniques Mica uses for finding keywords that are correlated with the top results of a query are similar to techniques used for query expansion [1] in information retrieval systems. While this is often used only for document retrieval, in interactive search systems, the expanded terms may be shown to the user. Mica's sidebar differs from these systems in that it filters based on programming relevance and the primary use of the terms is user understanding.

There are several search engines that search specifically for code [6, 13]. However, these search only a limited repository of known good code. They are unsuitable for solving the vocabulary problem because they do not search the informal forums and other pages that help programmers use naïve terminology to find the correct terminology. They are also limited in their use in finding examples even when programmers have a method name to start from, because they have such small repositories and because the repositories lack the textual descriptions that let programmers find methods used with a particular intent.

6. Future work

We plan to refine and revise Mica based on programmer observations and feedback. While it is often useful now, we hope to better understand and recognize the cases in which it is *not* useful.

For example, when the search results fail to find *any* relevant pages, Mica does not add any relevant information. Automatically recognizing this situation might allow it to quickly motivate programmers to try new search terms rather than wasting time browsing the current results. Programmers would also be aided by a tool that could recognize when there is *no* solution available for their desired search terms, i.e. when the task is known to be unsolvable with the current APIs, which is often mentioned in discussion forums.

Using a local index of relevant portions of the web would enable much faster results and allow other types of processing. For example, the source code in each of the pages could be preprocessed to do name resolution, allowing specific keyword name searches to work better when the names are also English words. A local index could still use Google's page-rank algorithm, or could use new ranking algorithms of its own. Such ranking algorithms could use usage log statistics from other programmers to help find useful programming documents.

Finding the right APIs and examples is only part of the programming cycle, as shown in figure 2; integrating examples is also an important task. We plan to explore tools that will help programmers copy and paste code found from web searches into an IDE and successfully integrate it into their project. Such a tool could also automatically record the source URL of pasted code, making it easier for other programmers to refer to the original code for understanding or debugging.

Finally, such IDE tools could be extended to feed back into the online environment. For example, tools could make it easier to post examples to forums, could record the customization that was necessary to adapt a particular copied example, and could manually or automatically link related problems posted by others to an eventually found solution.

7. Conclusions

Motivated by our observations of how programmers use web searches to find API information, Mica is a tool provides better information cues for programmers by extracting relevant information from web results and guiding programmers toward the results that will be most helpful for their current task.

By focusing on programmers' needs and behaviors, Mica shows that tools can offer practical web search improvements for programmers.

8. Acknowledgements

We would like to thank Andrew Ko for his ideas and comments on this paper. This work was partially supported under NSF grant IIS-0329090 and by the EUSES Consortium via NSF grant ITR-0325273. Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the NSF.

9. References

[1] Baeza-Yeats, R., Ribeiro-Neto, B. *Modern Information Retrieval*. Addison-Wesley, Reading, MA, 1999.

[2] Berglund, E. *Library Communication Among Programmers Worldwide*. PhD Thesis, Linköping University, 2002.

[3] Choo, CW., Detlor, B., and Turnbull, D. *Information Seeking on the Web - An integrated model of browsing and searching*. ASIS, Washington DC, 1999.

[4] Clarke, S. Weblog. http://blogs.msdn.com/stevencl/.

[5] Deline, R., Czerwinski, M., and Robertson, G. Easing Program Comprehension by Sharing Navigation Data. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '05)* (Dallas, TX, USA, Sept. 20-24, 2005). 241-248.

[6] Eakle, P. JExamples. http://www.jexamples.com/.

[7] Furnas, G.W., Gomez, T.K.L.L.M., and Dumais, S.T. The Vocabulary Problem in Human-System Communication. *Communications of the ACM*, 1987. 30(11): pp. 964-971.

[8] Google. The Google Web APIs. http://www.google.com/apis/.

[9] Holmes, R., Murphy, G. C. Using structural context to recommend source code examples. *In Proceedings of the 27th international conference on Software engineering (ICSE '05)* (St. Louis, MO, USA, Sept. 15-21, 2005). ACM Press, New York, 2005, 117-125.

[10] Ko, A. J., Aung, H., Myers, B. A. Eliciting Design Requirements for Maintenance-Oriented IDEs: A Detailed Study of Corrective and Perfective Maintenance Tasks. *International Conference on Software Engineering* (St. Louis, MI, May 15-21, 2005). 126-135.

[11] Ko, A. J., Myers, B. A., and Aung, H. Six Learning Barriers in End-User Programming Systems. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'04)* (Rome, Italy, Sept. 26-29, 2004). 199-206.

[12] Marchionini, G. *Information Seeking in Electronic Environments*. Cambridge University Press, New York, NY, 1995.

[13] Mitchell, N. Hoogle. http://www-users.cs.york.ac.uk/~ndm/hoogle/.

[14] Pirolli, P., Card, S. Information Foraging in Information Access Environments. *In Proceedings of the Conference on Human Factors in Computing (CHI '95)* (Denver, Colorado, USA). ACM Press, New York, 1995, 51-58.

[15] Rha, P. Detecting and Parsing Embedded Lightweight Structures. Masters thesis, Massachusetts Institute of Technology. 2005.