

Working with Search Results

Jamie Starke
University of Calgary
Calgary, Canada
jrstarke@ucalgary.ca

Chris Luce
University of Calgary
Calgary, Canada
cluce@ucalgary.ca

Jonathan Sillito
University of Calgary
Calgary, Canada
sillito@ucalgary.ca

Abstract

Source code search is an important activity for programmers working on a change task to a software system. We are at the early stages of a research program that is aiming to answer three research questions: (1) How effectively can programmers express (using today's tools) the information they are seeking? (2) How effectively can programmers determine which of the matches returned from their searches are relevant to their task? and (3) In what ways can tools be improved to support programmers in more effectively expressing their information needs and exploring the results of searches? To begin answering these questions we have conducted a study in which we gathered both qualitative and quantitative data about programmers' search activities. Our analysis of this data is still incomplete, however this paper presents several of our initial observations about how programmers interact with the results from their searches.

1. Introduction

Code search is an important activity for programmers working on a change task to a software system. Various tools exist for searching source code, including tools included with today's Integrated Development Environments (IDEs). Previous research studies have looked at various aspects of search as it relates to change task activities. We highlight just four such studies here. Ko et al. explored the strategies of programmers in finding, understanding, and using relevant information, in the context of a larger change task [3]. LaToza et al. focused on how experience effects work on changes tasks and found, for example, that more experienced programmers tend not to explore as many irrelevant elements [4]. Robillard et al. were interested in the differences between effective, and ineffective programmers and found that successful programmers often performed keyword and cross-reference type searches [5]. Finally, our own research that has been previously reported [6, 7] found that searches tend to produce many irrelevant

results and that time consuming exploration is required to determine what is relevant.

To build on this previous work we are at the early stages of a research program that is aiming to answer three research questions: (1) How effectively can programmers express (using today's tools) the information they are seeking? (2) How effectively can programmers determine which of their matches returned from their searches are relevant to their task? and (3) In what ways can tools be improved to support programmers in more effectively expressing their information needs and exploring the results of their searches?

To begin answering these questions we have conducted a study in which we observed programmers performing an assigned change task. As they performed the task we gathered qualitative and quantitative data about their search activities. In this way we have gathered information about 96 search episodes. Our analysis of this data is still incomplete, however this paper presents several of our initial observations about how programmers interact with the results from their searches.

2. Methodology

The study we conducted involved ten participants and ten sessions. In this paper we refer to our participants as P1...P10. Eight of our ten participants (P2, P4, P5, P6, P7, P8, P9, P10) had industrial programming experience. The remaining two (P1, P3) were graduate students. All participants had experience working with Java and the Eclipse development environment.

Each study session lasted 30 minutes. During the session the participant was asked to work on a change task to a large software system called Subclipse¹ (a popular open source plugin that provides support for Subversion² within the Eclipse IDE³). Subclipse contains approximately 70,000

¹<http://subclipse.tigris.org>

²<http://subversion.tigris.org>

³<http://www.eclipse.org>

lines of code and all of our participants were initially newcomers to the system. We did not expect that our participants would have sufficient time to complete the task, and we were primarily interested in their searching activities as they worked on the task.

For each session we randomly assigned one of two different change tasks. Both tasks were taken from the Subclipse project’s issue tracking system (specifically task one was based on issue 798 and task two was based on issue 801). These issues were addressed in revisions 3993 and 4012 of the Subclipse version control system, so we had our participants work with revision 3992 for task 1, and 4011 for task 2.

To facilitate data collection we had each participant work as a pair [9] with one of the researchers (the second author of this paper). The researcher was at the keyboard and the participant directed the work on the task by giving instructions to the researcher. We took this approach as a variation on the Think Aloud method [8]. The discussion between the participant and the researcher was recorded and a screen capture video was made for each session. Following the session a second researcher (the first author of the paper), who was also present during each session, conducted a short interview with the participant to further explore issues around their searching activities.

To carry out the change task, our participants used the Eclipse Java Development Environment⁴ and they were given a one page document describing the eight major kinds of searches supported by Eclipse. In the following we describe only the kinds of searches used by our participants.

Open Type: Supports searching for classes or interfaces based on a partial name or pattern.

File: Supports searching for text within all of the files in the Workspace.

Find in File: Supports searching for a piece of text within a specified file.

Java: For finding declarations, references and occurrences of Java elements (packages, types, methods and fields).

References: Performs a search for all references to a specified code element or elements matching a keyword.

Declaration: Performs a search for all declarations of a specified code element or elements matching a keyword.

Each time a participant used one of these searches we call this a search episode. Our data set consists of 96 completed episodes and in our ongoing analysis we are using these episodes as the basic unit of analysis.

⁴<http://eclipse.org>

Episodes by type		Size of results (explored)	
File	46	570 (1)	
Java	16	91 (1)	
Find in File	16	2 (1)	
References	13	3 (1)	
Open Type	4	4 (0)	
Declaration	1	12 (0)	

Figure 1. The number of times our participants conducted each type of search. For each type of search the average number of results is shown. The average number of results explored is shown in brackets.

3. Findings

3.1. Searches and Results

Figure 1 shows the number of episodes for each kind of search that our participants performed. Also shown in the figure are the average number of results and the average number of results that were explored. Eclipse’s File search was by far the most popular amongst our participants, accounting for approximately half of all search episodes. File search is the most inclusive search and as compared to a Java search has fewer options for scoping the search, so it is unsurprising that on average it returns the most results (570).

Some of our data suggests that participants use File search when they have very little information to base their exploration. For example, P1 began with File searches because “I don’t feel like there’s a starting point other than the keyword.” Similarly: “when I don’t know what I’m looking for, then usually I start with a text search” (P2). Participant P7 performed two similar File searches consecutively (the first search was for the keyword “image”; the second was for the keyword “icon”) as he explored which term was used for this concept in the system. These cases suggest that if our participants were not newcomers to the code base or if our sessions were longer we may have had proportionately fewer File searches in our data set.

The average number of results in a result set was 290. We found that the 96 search episodes could be divided into three nearly equal categories, based on the size of the result sets:

- **No results.** 30 of the 96 searches performed by our participants returned no matches.
- **1 – 9 results.** 32 of the searches performed by our participants returned between 1 and 9 matches.

Table 1. The number of episodes, the average number of elements explored and the average time taken for an episode for three different categories of result sizes.

Matches	Episodes	Explored	Time
0	30	0	27s
1-9	32	1.2	66s
10+	34	1.0	107s

- **10 or more results.** Finally, the remaining 34 searches returned 10 or more matches. The largest result set returned had 4770 matches.

Some quantitative data, organized around these three “bins” is summarized in Table 1.

This means that a full one third of the time our participants did a search they got back no results. At times this was surprising or frustrating for our participants, and in some cases it meant that they were off track and needed to consider a different approach. For example, “it should be contained there” and “ why the **** is it not finding anything then?” (P8). A very large number of matches was also difficult for our participants to deal with. We discuss this more in the next two sections.

3.2. Time Spent

To analyze the amount of time spent on each search episode we have given each of the 96 complete episodes in our data set a start and an end time. We consider a search episode to be started as soon as a participant starts to describe a search that they would like to perform. We considered this episode ended when they last explore one of the results in the result set. If no results are explored, we considered the episode time to be complete at the latter of when the search returned the last result into the result set, or the participant last comments on the results.

The longest amount of time spent in a single search episode was about five minutes and the shortest amount of time for a particular search episode was 5 seconds. The mean amount of time spent in a single search episode was 69 seconds and the median was 39 seconds. Table 1 shows the average amount of time spent on episodes in each of the bins described above (27 seconds, 66 seconds and 107 seconds). On average more time was spent on episodes where ten or more results were returned than on episodes in the other bins. However, in our data there is not a clear trend that would suggest that more results means more time spent.

Table 2. The number of episodes in which a particular number of elements were explored.

Explored	0	1	2	3	4	5
Episodes	17	38	5	3	2	1

In some cases, a large result set seemed to deter the participants from spending much time going through the results—or in even looking at any of the results. When one of participant P4’s searches returned 4770 matches he said “that doesn’t seem to be especially helpful” and moved on to another search. Later he performed a search which returned 2584 matches and said “ok, so that didn’t really work out” before abandoning the search.

3.3. Exploration Activity

For each search episode we have also tracked the number of elements in the search result that the participant chose to explore. For our analysis of exploration activity we are omitting episodes in which no matches were returned. This leaves us with 66 episodes in which there was potential for exploration activity.

As has been noted in previous research, we found that our participants were generally exploring only a small number of matches. The mean number of results that were explored within a single search episode was 1.06 results, and the median was 1. Table 1 shows the average number explored by bin. Note that the average number explored goes down slightly as the number of results increases.

As shown in Table 2, the most common behavior exhibited by our participants was to explore exactly one result from a result set, regardless of the size of the result set. Interestingly, another quite common behavior was to explore none of the results.

How exactly programmers choose which items to explore is still an open question. When asked about their choices most of our participants said that they guess that something is relevant based on the package and element name displayed in the results view. Participant P3 said that in making this decision he “... looked at the result that looked most promising”. This participant when asked how they decide what looks promising stated that it was “Totally based off name” and their understanding of a similar system. Many of our participants seemed to look for any element that seemed relevant, rather than looking at all of the results and selecting the most relevant element.

We also found that making such a decision simply based on name is not always reliable. Participant P2 performed

a search that returned four matches, one of which was arguably the element most directly related to the assigned change task. He looked over the names of the elements and decided to try a different type of search: “I’m not sure this is useful, maybe you should do a text search for the same string”.

The result sets that were returned by Eclipse were displayed by default in a tree format, with the exception of the Open Type which was in a list with no context, and Find in File which would jump to the elements one at a time. For the File search the top level contained the base directory of the project, followed by deepening levels based on the directories as far as the files. For the other tree results, the top level of the tree contains packages and the next two levels contains types and method or field names. If either of the trees are expanded further, Eclipse will also show one line of context which shows the match in context. Participant P6 was our only participant that made use of this contextual information.

4. Limitations

Our study involved a relatively small number of programmers, performing two specific software change tasks. Also, our analysis of the collected data is incomplete. As a result, care should be taken in drawing general conclusions based on our results. In the following we discuss a few specific limitations.

Our participants were newcomers to the system we asked them to change. In our experience this is a common scenario for programmers in industry and is convenient for conducting controlled studies. However, it may also mean that our results provide limited insights into the search behavior of programmers performing change tasks to systems with which they are familiar.

Our study setup involved one of the researchers being the “driver” during the session, similar to a pair programming experience. This was effective as it encouraged the participant to vocalize their intentions and allowed them to perform searches without necessarily knowing how to perform it in Eclipse. While this approach was helpful, it is an open question what effect the pairing had on the search behavior.

5. Conclusion

The analysis of our data set is not yet complete so we are not yet in a position to draw strong conclusions. However, we want to highlight two key observations that have relevance to the design of source code search tools.

First, it is clear that the way that search results are presented to programmers in IDE’s such as Eclipse provides

little support for programmers. Specifically, it appears that element names are not sufficient and we believe that more contextual information may be helpful, though it is not yet clear what form that should take. Programmers can obviously get significantly more information about search results by exploring the source code of an element, however we found that when they do this, they tend not to return to consider other results.

Second, along with only exploring a small number of matches (often zero or one) a large set of matches is not first examined before selecting which element(s) to explore. Based on this we would argue that the sorting of results should be given more attention, possibly based on confidence values such as those used by the Hipikat system [1]. We have found that programmers are put off by large result sets, but if the results were ranked in a meaningful way, it is possible that programmers would be able to make use of such results. How best to rank results is an open question, however we believe that a successful ranking would likely need to be context aware, possibly based on contextual information maintained by tools such as Mylyn [2].

References

- [1] D. Cubranic and G. Murphy. Hipikat: recommending pertinent software development artifacts. In *Proceedings of the International Conference on Software Engineering*, pages 408–418, 2003.
- [2] M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for IDEs. In *Proceedings of the International Conference on Aspect-Oriented Software Development*. ACM, 2005.
- [3] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung. An exploratory study of how developers seek, relate and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering*, 32(10):971–987, 2006.
- [4] T. D. LaToza, D. Garlan, J. D. Herbsleb, and B. A. Myers. Program comprehension as fact finding. pages 361–370. Association for Computing Machinery, 2007.
- [5] M. P. Robillard, W. Coelho, and G. C. Murphy. How effective developers investigate source code: An exploratory study. *IEEE Transactions on Software Engineering*, 30(12):889–903, 2004.
- [6] J. Sillito, G. C. Murphy, and K. D. Volder. Questions programmers ask during software evolution tasks. In *Proceedings of the SIGSOFT Foundations of Software Engineering Conference (FSE)*, 2006.
- [7] J. Sillito, G. C. Murphy, and K. D. Volder. *IEEE Transactions on Software Engineering*, 34(4):434–451, 2008.
- [8] M. W. van Someren, Y. F. Barnard, and J. A. Sandberg. *The Think Aloud Method; A Practical Guide to Modelling Cognitive Processes*. Academic Press, 1994.
- [9] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries. Strengthening the case for pair-programming. *IEEE Software*, 17(4):19–25, 2000.