What Do Developers Search for in Source Code and Why

Oleksandr Panchenko Hasso Plattner Alexander Zeier

Hasso Plattner Institute for Software Systems Engineering
P.O.Box 900460, 14440 Potsdam, Germany
{panchenko, office-plattner, zeier}@hpi.uni-potsdam.de

ABSTRACT

Source code search is an important tool used by software engineers. However, until now relatively little is known about what developers search for in source code and why. This paper addresses this knowledge gap. We present the results of a log file analysis of a source code search engine. The data from the log file was analyzed together with the change history of four development and maintenance systems. The results show that most of the search targets were not changed after being downloaded, thus we concluded that the developers conducted searches to find reusable components, to obtain coding examples or to perform impact analysis. In contrast, maintainers often change the code they have downloaded. Moreover, we automatically categorized the search queries. The most popular categories were: method name, structural pattern, and keyword. The major search target was a statement. Although the selected data set was small, the deviations between the systems were negligible, therefore we conclude that our results are valid.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments; H.1.2 [**Information Systems**]: User/Machine Systems

General Terms

Human Factors, Verification

Keywords

Source code search, log file analysis, development activities

1. INTRODUCTION

Numerous tools have been developed to support source code search. Nevertheless, relatively little is known about how and why developers perform code search. Although some research has been done in this area (Section 4), several important questions remain unanswered. This paper addresses the following research questions:

SUITE'11, May 28, 2011, Waikiki, Honolulu, HI, USA

(1) What are commons and differences between source code search using a publicly available Internet-scale search engine and search in an internal development system?

(2) How does the type of tasks completed by software engineers influence the way they perform search and the way they work with the results?

(3) What are syntactic categories of search queries? By syntactic categories we mean the target of the query: for example, a method name, a regular expression, or a literal.

To answer these questions we analyzed a log file of a proprietary internally used source code search engine. We compared our findings with recent analysis of a publicly available Internet-scale search engine. Our log file originates from enterprise systems SAP Customer Relationship Management[1] (CRM) and SAP NetWeaver Business Warehouse[2] (BW). The language of these systems is ABAP[3], which is a fourth-generation programming language for data processing in commercial applications. In addition to language elements which are usual for programming languages, ABAP offers functions for background task scheduling, authority checks, operations with database tables, arrays, files, user interface elements, strings, date and time, etc., as language elements. Therefore, the language is very rich with more than 800 non-reserved keywords. In an ABAP system source code is organized into development objects (programs, classes, function modules, etc.), which consist of includes. These systems differ from usual software development in terms of handling of code because a relatively small amount of ABAP code is publicly available on the Internet, and thus, developers are constrained to use an internal search system or to relinquish the use of the search functionality.

To discover how the type of development tasks influences search behavior, we compared two pairs of systems: development and maintenance systems. We expect that maintenance systems feature less code modifications, but that the search targets are changed after being downloaded. In contrast, the development systems feature frequent code modifications and creating new code, but the search targets remain mostly unchanged. To prove this hypothesis we compared two pairs of systems: development (denoted as CRM-D, BW-D) and maintenance (CRM-M, BW-M). Note, that SAP software products are internally moved from the development department into the maintenance department. Thus, such a clear distinguishing of task types is possible.

[1] http://www.sap.com/solutions/business-suite/crm/
[2] http://www.sap.com/platform/netweaver/components/
[3] http://www.sdn.sap.com/irj/sdn/abap/

2. ANALYSIS

Following the introduction of some facts about the analyzed systems and the log file, this section describes the analysis of the log file and change history, and the results of the query categorization.

The search engine is based on the inverted index technique. Therefore, only text queries and regular expressions are supported. No other query interface is implemented. In total about 100 development and maintenance systems are indexed in this search engine. We selected four systems for our analysis and extracted a part of the log file corresponding to these systems. The length of the selected part was restricted to four months. To select the systems we chose the following criteria: (1) the systems should comprise of pairs (development and maintenance systems) of similar version; (2) the systems should have enough entries in the log file. The log file contains entries of two types: search activities and download activities. Each download activity corresponds to a click on the result list of the query and shows exactly one include on the screen.

Although the systems are large (the selected systems contain about 160 million statements) and have a substantial number of developers, the source code search engine was used by only about 100 developers. The reason for this is that the search engine is not integrated into the development environment. Nevertheless, the selected part of the log file contained 16k activities (search queries and downloads).

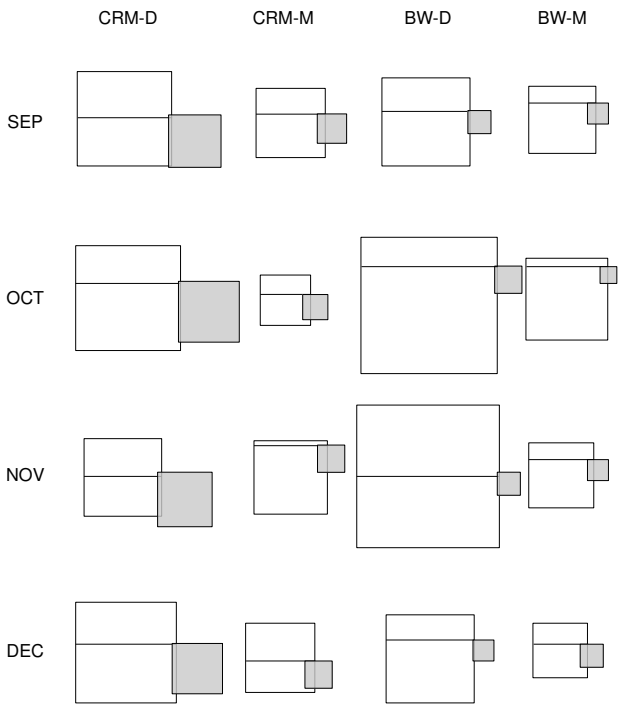

Figure 1: Change History and Download Activities

2.1 Results

In the first part of the analysis we investigated if there is any relation between search activities and code changes. In other words, if the search targets (in the technical sense these are includes that have been downloaded) were modified after they have been downloaded. Therefore, we extracted the change history and mapped download activities in the log file with modification activities in the change history. In principle, for each period of time (month in our case) we prepared a list of includes changed or created during this period and a list of includes downloaded during this period. We assume that includes belonging to both lists imply the relation between search and code modification activities.

Each include that appears in the log file has been counted only once per time period. Although some includes were downloaded several times, the majority of the includes were downloaded only once. Therefore we prepared the distinct lists of includes. In the same way we counted the changes of the includes. That is, if an include has been modified several times, it was counted only once. Automatically generated includes were excluded from the analysis.

Some statistics of the log file and the change history are presented in Table 1. The results of the analysis are presented in Figure 1. Each white square presents the list of modifications of the system. The upper part of the square represents created includes, while the lower part of the square represents changed includes. Each shadowed square presents downloads. Square size corresponds to the number of modifications/downloads. The overlap of the squares means that the same includes have been both modified and downloaded within a given period at least once. Each row in Figure 1 corresponds to a month (September to December). We conclude the following:

(a) There are no big fluctuations in the number of downloads over time. We found only a weak connection between the number of downloads and the number of modifications.

(b) There are almost no includes that are both created and downloaded. Based on this fact we conclude that the ability of a search engine to update the index immediately after a code has been created is of minor importance. Nevertheless, such a feature can increase the acceptance of a search engine by developers.

(c) There are a relatively small number of includes that are both changed and downloaded. Most of these includes have been found in the maintenance systems. Based on this fact we conclude that during development software engineers used search to find reusable components or code examples; during maintenance search targets were changed more often. This fact indirectly hints on the way how developers used search engines.

2.2 Query Categorization

The queries were automatically categorized into one of the categories presented in Figure 2. The categorization was done using naming conventions. For example, queries starting with "`GET_`", "`SHOW_`", "`CONFIRM_`", or other verbs followed by an underscore were assigned to the *Method name* category. The *Pattern* category includes queries where software engineers used symbols, such as "<", ">", "=", "==", "=>", "(", ")", to express the relations between terms of the query. Queries containing ABAP keywords were categorized into the *ABAP keyword* category. The category *Identifier* contains variables, parameters, table names, structures and other identifiers which could be assigned using naming conventions. For example, global variables should start with "`GV_`", importing parameters should start with "`IM_`", and

Table 1: Characteristics of the Systems

	CRM-D	CRM-M	BW-D	BW-M	Total
Number of search queries	4,839	898	1,077	902	7,716
Number of distinct queries	3,892	686	822	641	
Number of downloads	5,433	1,354	999	819	8,605
Number of changed includes	10,183	4,935	16,098	6,781	
Number of created includes	6,348	3,005	8,749	1,453	
Total number of includes	1,710k	1,484k	165k	123k	

local tables should start with "LT_". Moreover, queries consisting of one or several nouns separated with an underscore were categorized as identifiers too. Other categories were *Class name*, *Regular expression* and *Literal*. The category *Undefined* contains class, method and identifier names that could not be automatically categorized because of missing naming conventions. Remaining categories were identified accurately. Figure 2 shows that the most popular categories were: *Pattern*, *Method name* and *ABAP keyword*.

Remarkably, the number of literals used in queries in the maintenance systems is higher. We guess that the reason for it is that maintainers often put the text of the error message or its code into the search query. The number of queries based on ABAP keywords if higher in development systems. From this fact we conclude that during development the language elements were more important. Moreover, the percentage of class names in queries ran against the development systems was higher. We guess that developers perform search for reuse more often than maintainers.

The search engine provides the possibility of searching over *all* indexed systems. The analysis of the complete log file showed that about 13% of queries were run against all systems simultaneously. Figure 3 compares the category distribution of search queries issued in *all* systems with the distribution of queries ran against a *single* system. The complete log file contained 440k search activities. As expected, developers that searched in all systems had different queries compared to developers that searched in a single system. We grouped the categories into two classes: coarse-grained entities (class and method names) and fine-grained entities (patterns, regular expressions, ABAP keywords, identifiers and literals). These classes are marked in Figure 3 using different filling patterns. From the differences in the query distribution we conclude that developers searching in all systems were more interested in coarse-grained entities. In this sense, the search over all systems is similar to the Internet-based source code search.

3. THREATS TO VALIDITY

Our analysis has several threats to its validity. Firstly, not all needed data could be collected. For example, we could not explicitly connect a download activity to the modification activity of the corresponding piece of source code. For mapping of those events we supposed that if they occur within the same time frame they are connected. One possibility of controlling this would be reducing the size of the time frame. However, the modification activities in the change history we obtained were already aggregated to months. Thus, the accurate mapping was not possible.

Secondly, we used ABAP, which has a unique development process, which influences the way developers work with search engines.

Thirdly, the search engine used in this experiment is capable of text-based search only. This source code search engine has been known as a pure text-based search engine. This preconception has influenced the way developers have formulated the search queries. Recently developed search engines for code include many more important features, for example a special source code query language.

Finally, the size of the log file is not representative enough, especially given the large size of the system.

All in all, we hope that in spite of all these restrictions the study delivers a complementary view on source code search.

4. RELATED WORK

Numerous source code search engines exist, for example Codase[4], Google Code Search[5], Koders[6], Krugle[7], Merobase[8]. Several previous studies have investigated typical search targets and developers' motivations [3, 4, 5, 2]. Some efforts in analysis of logs of a source code search engine have been undertaken. Bajracharya and Lopes presented an extensive analysis of a log file of an Internet-scale source code search engine [1]. Although the previous study features a larger data set, the analysis covers only a small set of all possible scenarios in which code search can be used. It focuses on Internet-based search and reuse. Although our data set is smaller, the contribution of this paper is threefold: it gives another perspective of search in a proprietary development landscape; it focuses on the syntax of queries rather than on semantics; and it connects search activities to change history.

An important aspect to mention is the interaction between the developer and the search engine. The previous study reported that 64% of users have no downloads at all [1]. In our interpretation this means that the majority of the users did not have a concrete idea in mind and just tried to play with the tool. In our case the situation is opposite. The majority of the users downloaded several includes and on average had longer search sessions. On average, each developer performed 160 search and download activities. Users of the Internet-scale code search engine performed only 3 activities on average. This is no surprise given the difference in responsibility between developers working with an industrial software system and users of an Internet search

[4]http://www.codase.com
[5]http://www.google.com/codesearch
[6]http://www.koders.com
[7]http://www.krugle.com
[8]http://www.merobase.com

engine. Moreover, industrial programmers have a possibility to influence the development of an internal source code search engine, e.g., request new features. Users of Internet-based search engines have only limited possibility to give their feedback to the developers of those engines.

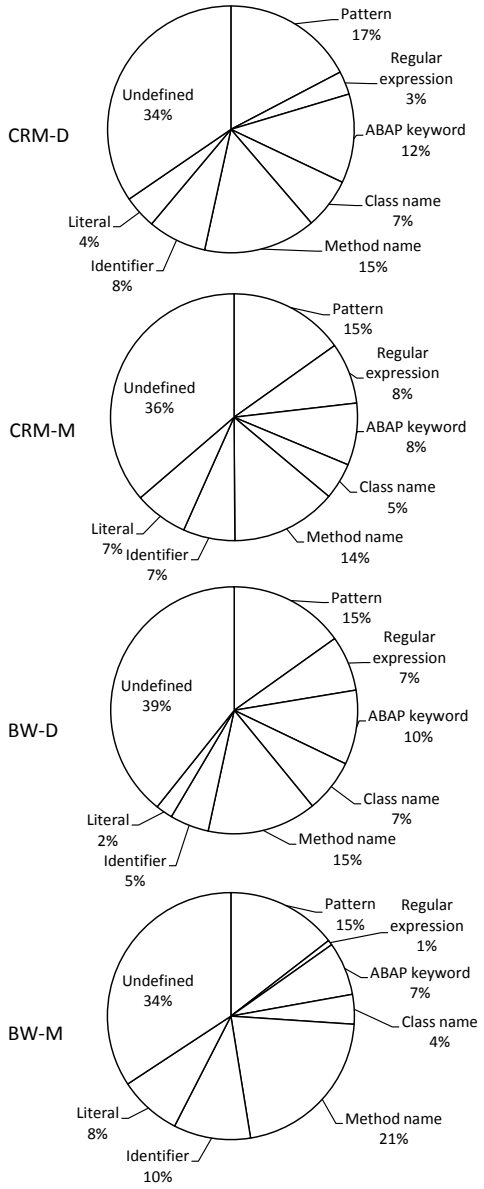

Figure 2: Search Categories

5. CONCLUSION AND FURTHER WORK

This paper presents the results of a log file analysis of a text-based source code search engine. It unveils some knowledge about the use of search tools in software engineering. The results presented in this paper can serve as additional requirements for source code search engines and for designing a benchmark for search tools. The following conclusions were made: (1) software engineers work with internal source code search engines in a different way comparing to Internet-based ones; (2) developers and maintainers have different motivations while searching code; (3) software engineers often use fine-grained code entities (e.g., identifiers, keywords) in queries. Although the dataset used in this analysis cannot be shared, we hope that the ideas and conclusions of this paper will aid designing of a benchmark. Further work includes the validation of the findings with developers using interviews.

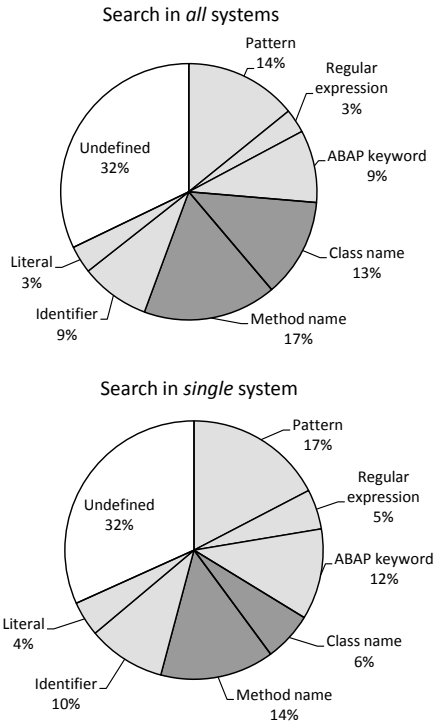

Figure 3: Search in All Systems vs. in Single System

6. REFERENCES

[1] S. Bajracharya and C. Lopes. Analyzing and Mining a Code Search Engine Usage Log. *Empirical Software Engineering*, pages 1–43, 2010.

[2] R. Hoffmann, J. Fogarty, and D. S. Weld. Assieme: Finding and Leveraging Implicit References in a Web Search Interface for Programmers. In *Proceedings of the Symposium on User Interface Software and Technology*, pages 13–22, New York, NY, USA, 2007. ACM.

[3] S. E. Sim, C. L. A. Clarke, and R. C. Holt. Archetypal Source Code Searches: A Survey of Software Developers and Maintainers. In *Proceedings of the 6th International Workshop on Program Comprehension*, pages 180–187. IEEE Computer Society, 1998.

[4] S. E. Sim, M. Umarji, S. Ratanotayanon, and C. V. Lopes. How Well Do Search Engines Support Code Retrieval on the Web? *ACM Transactions on Software Engineering and Methodology*, pages 1–30, 2011.

[5] M. Umarji, S. Sim, and C. Lopes. Archetypal Internet-Scale Source Code Searching. *International Federation for Information Processing*, 275:257–263, 2008.