Empir Software Eng (2012) 17:424–466
DOI 10.1007/s10664-010-9144-6

Analyzing and mining a code search engine usage log

Sushil Krishna Bajracharya · Cristina Videira Lopes

Published online: 18 September 2010

Editors: Mike Godfrey and Jim Whitehead

Abstract This paper presents an analysis of a year long usage log of Koders, the first commercially available Internet-Scale code search engine (http://www.koders.com). The usage log comprises about ten million activities from more than three million users. Analysis of the usage data shows that despite of attracting a large number of visitors, Koders has a very sparse usage and that it lacks regular usage from many of its users. When compared to Web search, search behavior in Koders showed many similar patterns. A topic modeling analysis of the usage data shows what topics users of Koders are looking for. Observations on the prevalence of these topics among the users, and observations on how search and download activities vary across topics, lead to the conclusion that users who find code search engines usable are those who already know to a high level of specificity what to look for. This paper also presents a general categorization of these topics that provides insights on the different ways code search engine users express their queries. It identifies various forms of queries in Koders's log and the kinds of results addressed by the queries. It also provides several suggestions for improvements in code search engines based on the analysis of usage, topics, and query forms. The work presented in this paper is the first of its kind that reveals several insights on the usage of an Internet-Scale code search engine.

Keywords Code search engine · Usage log analysis · Mining topics

S. K. Bajracharya · C. V. Lopes
Donald Bren School of Information and Computer Sciences, Department of Informatics,
University of California Irvine, Irvine, CA 92697-3440, USA

C. V. Lopes
e-mail: lopes@ics.uci.edu

S. K. Bajracharya (✉)
1230 Stanford, Irvine, CA 92612, USA
e-mail: sbajrach@ics.uci.edu

1 Introduction

Searching for source code constitutes a significant part of a software development activity (Singer et al. 1997; Murphy et al. 2006). Software developers use a variety of tools to search source code that range from conventional tools such as 'grep' to advanced search facilities in integrated development environments. With the explosion of source code available on the Web, it has become a routine practice among developers to search and reuse source code from the Web.

Recent research in software engineering has focused on understanding and supporting the search-driven nature of software development, and many such work have produced tools that aid various search needs of developers (Hoffmann et al. 2007; Bajracharya et al. 2006, 2009; Lemos et al. 2009; Holmes and Murphy 2005; Mandelin et al. 2005; Thummalapenta and Xie 2007; Hummel et al. 2008; Reiss 2009). On the other hand, there has been considerable commercial effort in the development of Internet-Scale code search engines that have powerful crawlers and large databases providing an index to large quantities of software available on the Web (Web site for Koders 2010; Web site for Krugle 2010; Web site for Google Code Search 2010).

Despite the progress in developing novel code search tools, there has been little work in understanding the actual usage of these tools. Recently, a questionnaire-based study obtained some insights on developers' practice of searching for source code on the Web (Umarji et al. 2008). Two other studies used search logs to investigate information needs and query styles of developers on Web search (Hoffmann et al. 2007; Brandt et al. 2009). These studies show that searching for source code on the Web has various purposes: learning existing APIs, finding code samples, implementations etc (Umarji et al. 2008; Hoffmann et al. 2007); and that developers express their information need using various forms of queries, ranging from natural language terms to names of code entities they are aware of (Brandt et al. 2009).

Code search engines are relatively new category of tools, and there are no existing studies that explain what users of these code search engines are looking for, and how they express what they are looking for. Despite of the apparent popularity of code search engines,[1] it is not clear whether they are effective in providing the information software developers need. We conducted an exploratory analysis of the usage log of Koders, the first commercial Internet-Scale code search engine, with a goal to answer three major research questions:

- *Usage:* What kind of usage behavior can we see in Koders?
- *Search Topics:* What are the users searching for?
- *Query Forms:* How are users expressing their information need in their queries?

There are four motivations behind these questions. First, we wanted to compare the usage patterns in Koders with known usage patterns in general purpose search and developers' search behavior on the Web. Second, we wanted to get a high-level summary of what developers are looking for in Koders. Third, by analyzing selected user interactions in detail we wanted to know if users have specific ways of expressing

[1] Koders mentions in its Web site (http://koders.com) that more than 30,000 developers use its search service everyday.

their queries, and whether some forms of expression were more effective than other. Finally, we wanted to gain some insights on what improvements could be made to Koders, and code search engines similar to it, based on our findings.

This paper is an extension of our work presented in Bajracharya and Lopes (2009) with new results on usage statistics, query forms, and more details on topic extraction. It describes the process and results that we extracted from the usage log to answer the three major questions listed above. To the best of our knowledge the work presented in this paper is the first of its kind with detailed analysis of results extracted from millions of entries in the usage log of an Internet-Scale code search engine. In the context of the prior work mentioned earlier, the contributions of this paper are as follows:

1. It provides a statistical characterization of search behavior in large scale code search by presenting an analysis of a year long usage data from Koders. It compares code search usage with Web search usage, and finds many similarities. It also reveals usage behavior that is unique to Koders.
2. Using topic modeling analysis on the Koders's usage data, it gives a solid empirical evidence of the range of topics that users of code search engines look for. It provides empirical data on the prevalence of these topics (popularity) among users and shows how search and download activities vary across the topics, supporting the conclusion that the most successful searches on Koders are those where the users already know what they are looking for. It also provides valuable insights at the different ways code search engine users express their queries.
3. With an analysis of 150 randomly selected search sessions across various topics (mined using the topic modeling), it identifies various lexical forms of the queries, and the kinds of results addressed by these queries that users gave to Koders.
4. It provides several suggestions and possible directions to improve code search engines based on the analysis of usage data, topic modeling results, and the analysis of query forms; something the next generation of code search engines should take into account.
5. It makes the Koders usage log, and associated software used to produce analysis results available to others; facilitating replication, extension, and improvement of the presented work.

The paper is organized as follows. Section 2 discusses the usage data we analyzed. Section 3 provides an analysis of the usage data at large, providing some general statistics on usage and its comparison with usage in Web search. Section 4 presents the LDA (Latent Dirichlet Allocation) topic modeling technique, describes how we applied it to the Koders's usage log, and presents our findings from topic modeling. Section 5 presents the results of analyzing 150 search sessions sampled from various topics where we identify five lexical forms and four result types users generally expressed in their queries. Section 5 also presents the form of queries that were effective in producing relevant results. Section 6 provides discussion on our interpretation and implication of the results we obtained from analyzing the general usage statistics, mining topics, and encoding various forms of queries. Section 7 discusses validity and limitations of our work. Section 8 presents related work, and we conclude in Section 9.

2 Usage Log Data

The data used for topic modeling consists of a year long user activity log obtained from Koders (Web site for Koders 2010). The usage log contained records of 5,207,758 search activities and 5,072,045 download activities from 3,187,969 unique users covering the period of 2007-01-01 to 2007-12-31.

The log data was recorded in a relational database. The portion of the log we used is represented as a set of tuples with the following fields; <*uid, act-type, term-or-file, ts, l*>, where:

1. *uid* = a unique user id assigned by Koders to each of its user based on the combination of the user's IP address and browser cookies.
2. *act-type* = activity type, that can be either *search* or *download*. A search activity constitutes a query consisting of several terms, whereas a download activity is an activity where the user interacts with one of the results shown in the hits either by selecting the code or downloading it. A download activity means the user showed interest in the code that was found in the search results and used it in some way. Therefore a download activity in Koders is equivalent to the result-click events in Web search.
3. *term-or-file* = denotes the collection of terms in the query when the activity is search, otherwise denotes a unique file identifier denoting the source code that was accessed during the download activity.
4. *ts* = the timestamp attached to each activity that denotes when that activity took place.
5. *l* = the programming language specified by the user for each query. If no language is specified the value is '*' denoting search in all languages. Other possible values are languages listed in the Koders's user interface such as Java, C, Python etc. This value exists only for search activities and not applicable to download activities.

The usage log studied in this paper is available from the UCI Source Code Data Sets Web site (Lopes et al. 2010). The software used to process the data is available as an open source project at Web page for Koders log analysis github repository (2010).

3 Analysis of Usage Data

In this section we look at several statistics to gain insights on usage behavior in Koders. We look at variables that are commonly used in analysis of query logs such as statistics on activities, search sessions, query types and query reformulations (Silverstein et al. 1999; Brandt et al. 2009). Being a code search engine, Koders offers some unique features not found in general purpose search engines. For example, query operators specific to source code, and facility to download (or browse) code after search. We focus more on variables pertaining to these features. These statistics not only reveal usage patterns that are unique to Koders, but also allow us to compare search behavior of users in Koders to those on the Web. Given below is a summary of all the variables we look at.

– *Routine usage:* First, we look at three variables to understand whether users are searching in Koders routinely and actively. We look at number of days that

users are active in Koders, number of search activities, and number of download activities among the users.

– *Analysis of sessions:* Second, we do an analysis on sessions of activities in the usage log. A session is considered to be a series of queries by a single user in a short interval that represents a single information need (Silverstein et al. 1999). We look at three variables in sessions: duration, activities, and page views. Duration is the length of session in minutes, activities are either search or download activities, and page views are count of consecutive repeating queries that are recorded in the log when a user navigates through multiple pages of search results for the same query.
– *Analysis of queries:* Third, we do an analysis of the queries in the log to understand how users are expressing their queries. We look at query length, common usage of terms in queries among users, types of queries user give, and the kind of query operators and reformulations in the queries.
– *Comparing with Web search:* Finally, we compare some of the results we obtained with existing results from analysis of logs in Web search.

3.1 Routine Usage

Koders mentions in its Web site that more than 30,000 developers use the search engine every day. However this number does not indicate how routinely users rely on the system. A code search engine might have many visitors, but if the visitors are not coming back, or using it routinely, its utility is questionable. Therefore, we seek to answer the following questions: How many activities users typically have in the system? Do users who use Koders once come back to it again later for their information needs?

Table 1 lists statistics on number of days the users used Koders, and the count of search and download activities for users. These statistics are computed for users who had at least one search activity. We can make the following observations based on the data in Table 1.

– *Users engage in very few activities:* A large percentage of the users had only few search activities. More than 85% of users had just three or less search activities; about 67% of users had only one search activity. More than half of the users

Table 1 Usage statistics

#Act	Days active		Download activities		Search activities	
	# Users	%	# Users	%	# Users	%
0	NA	NA	1,212,666	64.35	NA	NA
1	1,685,551	89.45	289,988	15.39	1,276,549	67.74
2	125,331	6.65	146,623	7.78	225,018	11.94
3	37,026	1.96	74,482	3.95	114,280	6.06
> 3	36,418	1.93	160,567	8.52	268,479	14.24
<= 3	1,847,908	98.06	1,723,759	91.48	1,615,847	85.75
3 < #Act <= 10	36,418	1.8	121,875	6.47	206,153	10.95
> 10	2,648	0.14	38,692	2.05	62,326	3.3

#Act = number of activities, % denote percentage with respect to total of the users with at least one search activity

do not download anything after search, and those who download have only few downloads. About 64% of the users who searched had no download activity, and 91% of users had three or less downloads.

– *Most of the users did not used Koders again after using it for a day:* About 90% of the users were active only for one day. Among others, 98% of users were active for less than or equal to three days. Only 0.14% of users were active for more than 10 days.

In summary, these statistics indicate that although Koders gets many visitors, the actual usage is quite sparse among the users.

3.2 Analysis of Sessions

In analysis of usage logs, it is often assumed that a series of activities by a single user within a small duration of time constitutes a session (Silverstein et al. 1999; Brandt et al. 2009). A session is considered to capture the interactions made by a particular user to fulfill a single information need. We divided the user activities in Koders into sessions, where a session is defined as a sequence of query and download events from a same user with no gaps longer than 6 min. This definition of session is common in query log analysis (Silverstein et al. 1999; Brandt et al. 2009).

We found out that among all the users in the Koders log, about 41% (1,303,643) did not have any search activity. They had only downloads. This initially seemed as an anomaly, as we were expecting that there needs to be a search in order to have a download. Later, upon discussing this issue with Koders, we realized there could be two reasons for this. First, a user could have reached a link to download a code in Koders directly by following a hyperlink from an external Web site. For example, someone can post a Web page that contains a link to a specific file on Koders. So, end users who initiate a search on a general search engine such as Google, can find such Web pages that provide links to files in Koders. When users follow such links and view code in Koders, such activity is recorded as a download activity in the log. Second, it could simply be that bots are crawling links of code downloads in Koders. Being unsure about what might have caused the log to contain a large number of users having no search activity, we excluded them while creating the sessions.

Breaking the users activities into sessions using a gap of more than six minutes, we obtained 2,804,432 sessions. These sessions had 5,207,758 search activities and 3,189,018 download activities in total. Table 2 lists the statistics on duration and

Table 2 Duration (in min) and activities count in sessions

Count (c)	% of sessions			
	c = duration	c = activities	c = searches	c = downloads
0	57.44	0.00	14.02	57.13
1	0.16	55.49	57.99	27.46
2	0.09	17.23	12.11	8.05
3	0.07	9.34	5.92	3.05
> 3	16.19	17.95	9.97	4.30
$<= 3$	83.81	82.05	90.03	95.70
$3 < c <= 10$	12.59	15.05	8.81	3.48
> 10	3.60	2.90	1.16	0.82

activities count in these sessions. Based on that statistics we can make the following observations about sessions:

- *Sessions are short:* About 57% of sessions had only one activity. About 84% of sessions had a duration of less than or equal to three minutes. Only 3.6% of sessions had a duration of more than 10 min.
- *More than half of the sessions had no downloads:* About 57% of sessions had no downloads in them. Sessions with lot of downloads are very rare; less than 1% had more than ten downloads.
- *There are few sessions with no search activities:* Table 2 shows that about 14% of sessions had no search activities. These sessions can be described as series of isolated download activities made by an user. Sessions have only few search activities in them. About 90% of sessions have less than or equal to three search activities.

In summary, these statistics indicate that sessions are short, and usually have few activities in them.

Activities in Sessions Data in Table 2 indicates that we can classify sessions based on whether they contain search or download activities. One reason to do so is to see if there are any noticeable differences between sessions that do not have downloads with sessions that have both search and downloads. Since a download indicates that a user looked at a result that was considered relevant, knowing such differences might shed some light on what leads users to get relevant results. Table 3 lists the counts of activities in three different categories of sessions: sessions without downloads, sessions with both search and downloads, and sessions without search. We can see that sessions with both search and downloads tend to have more search activities in them. About 20% of sessions with both search and downloads have more than three search activities. Compared to this, only about 7% of sessions without downloads have more than three activities. This suggests that *users who look at relevant results have more search activities than those who do not.*

Table 3 also shows that sessions that only have downloads tend to have a larger percentage (82%) of sessions with only one download activities. Some of these downloads could be because that users arrived at Koders by following link found elsewhere (as discussed in Section 2). Table 4 provides some insight on what leads to a download in Koders. It shows that about 43% of all downloads in the session have a preceding download activity; about 29% of downloads follow search activities; and, about 27% of downloads have no other activity before them in the sessions. This

Table 3 Activity counts in different kinds of sessions

#Act	#Act = search		#Act = download
	% of Ses_{WOD}	% of Ses_{WSD}	% of Ses_{WOS}
1	76.95	48.62	82.23
2	10.90	20.37	10.24
3	5.01	10.60	2.88
> 3	7.14	20.40	4.65
<= 3	92.86	79.60	95.35
3 < #Act <= 10	6.47	17.72	3.14
> 10	0.67	2.68	1.51

Ses_{WOD} = sessions without downloads; Ses_{WSD} = sessions with both search and download; Ses_{WOS} = sessions without search activities

Table 4 What activities do downloads follow in a session

Previous activity	# downloads	% of downloads in session
Download	1,388,810	43.55
Search	938,440	29.43
None	861,768	27.02

suggests that most of the downloads in Koders are made because a user finds one result, and start browsing for other related code using its code browser; therefore a high percentage of downloads follow a download activity.

Screen Views (Repeating Queries) A series of consecutive repeating queries in a search log indicates a user going through the subsequent pages in the search result. This is because when a user requests for the next result page for a query, the same query is repeated. These repeating queries are counted as "Screen Views" in analysis of Web search logs (Silverstein et al. 1999). Screen views measure the tendency of users to navigate to low ranked search results for the same query.

Table 5 shows the statistics for screen views in the search sessions in Koders. We also compute this statistics for search request in sessions with both search and download, and sessions without download to observe any noticeable effect of screen views on downloads. Data in Table 5 indicate that most of the search requests have only one screen view, and there does not seem to be any noticeable difference between screen view statistics in sessions with both search and download, and sessions without download. About 85% of all downloads that followed a search activity were made after only one page view. This suggests that when users download code from Koders, most of the times they do it from the first result page itself. To summarize, we can say that *most of the users do not look beyond the first result page in Koders*.

3.3 Analysis of Queries

We looked at several variables related to the queries that users gave to understand how users express their information need in Koders.

Query Terms Each query in the log was broken down into its constituent terms by using whitespace as delimiter. This produced 913,325 distinct terms. Two different

Table 5 Screen views statistics

# Screen Views (SV)	% $Search_{ALL}$	% $Search_{WSD}$	% $Search_{WOD}$	% $Download_{F}$
1	86.35	85.17	87.47	85.68
2	9.53	10.50	8.62	10.16
3	2.30	2.47	2.13	2.38
> 3	1.82	1.86	1.78	1.78
$<= 3$	98.18	98.14	98.22	98.22
$3 < SV <= 10$	1.63	1.72	1.54	1.65
> 10	0.19	0.14	0.24	0.13

$Search_{ALL}$ = search requests in all sessions; $Search_{WSD}$ = search requests in sessions that have both search and download; $Search_{WOD}$ = search requests in sessions without download; $Download_{F}$ = download activities following screen views

Table 6 Number of terms (t) or query length among users and queries

# of max. terms in query	# of users	% of users	# of queries	% of queries
1	1,488,364	78.99	4,147,683	79.64
2	254,823	13.52	737,565	14.16
3	85,032	4.51	205,441	3.94
> 3	56,107	2.98	117,069	2.24
$<= 3$	1,828,219	97.02	5,090,689	97.75
$3 < t <= 10$	55,211	2.93	115,652	2.22
> 10	896	0.05	1,417	0.03

statistics were computed on these terms that revealed the following characteristics about term usage in the queries.

- *Queries are very short:* Table 6 shows that about 79% of the users had only one term in their query; 97% of the users had three or less terms. Only 0.05% of the users had more than ten terms in their queries. This statistic is similar when we look at number of terms across queries. More than 79% of queries had only one term in them, and more than 97% of queries had less than or equal to three terms.
- *Terms in queries are quite diverse:* Large percentage of the terms were unique among users. Table 7 shows that about 72% of the terms had only one user using them in queries, 89% the of terms had at most three users in common, and only 3% of all the terms were common among more than ten users. The top five of the most common terms were (showing number of users using them inside brackets): `md5` (46,433), `sort` (29,728), `file` (19,219), `code` (15,532), and `java` (15,092). Examples of (rare) terms that had only one user each are: `bjc_compress`, `stream_update`, `partitioning.h`, "`evaluate_nbr_bits`", and `ktportlet`. Koders maintains a list of most popular queries in its Web site, and the top terms listed above can be found in that list. It is possible that users tend to look at these popular examples and try those queries themselves, thus contributing to the popularity of already popular examples. All of the terms that were unique seem to be names of variables and files.

Users use very few terms in a query to express their search needs. It seems that they are using frequently used terms to try out popular queries listed in Koders, and rarely used terms are being used to find specific methods and functions within the code the users are familiar with.

Use of Operators Almost every search engine have options for query operators. Query operators provide various ways to restrict or expand search results by using

Table 7 Terms common among users

# of common users (u)	# of terms	%
1	659,401	72.19
2	107,881	11.81
3	48,029	5.25
> 3	98,014	10.73
$<= 3$	815,311	89.27
$3 < u <= 10$	67,001	7.33
> 10	31,013	3.40

special characters or terms in the query. Koders provides six operators that users can use to refine their queries: use of quotes to denote phrase search; using "cdef:", "mdef:", and "idef:" as prefix to a query term to find class, method and interface definitions; adding a "*" at the end of a term to denote stemming; and, adding a "−" before a term to denote exclusion of a term. We analyzed all query terms for these operators, and found that most of the queries do not have any operators at all. We also found rare instances where users used operators that are similar to but not available in Koders. Two such operators were: use of "mcall:" as term prefix, possibly to denote finding a method call; and, use of "+" as a term prefix possibly indicating inclusion (opposite of "−"). We found these two operators while doing a cursory scan of the queries in the log to see any observable use of operators and included them in our analysis.

Table 8 shows the statistics on use of operators. About 93% of the queries did not have any operators. The most popular operator seemed to be use of quotes, followed by operators to find definitions. It is interesting to note that two operators ("mcall:" and "+") that are not available in Koders, are more popular than some other operators that are available.

Koders offers two other operators in its user interface to refine the query results: Language and License type. The default value for these operators will return results in any language and results from code with any license. Looking at the use of these operators revealed an interesting information. About 62% of the queries had a language specified in them, whereas only about 1% of total queries had specified a license.

These observations on the use of operators indicate that in general users do not use query refinement operators, with an exception of the operator for a language. This probably means that users care about the language they want to search in, over any other refinements.

Query Types A recent study of search behavior of developers on the Web found that there are usually three forms of queries that developers write (Brandt et al. 2009): *Natural:* where all the terms in the queries are natural language words; *Code:* where none of the terms in the queries are natural language words, and *Hybrid:* where users mix natural and code terms in the query. We classified the queries in the Koders log into these three based on the types of terms they contained. First a query was split into terms using whitespace as a delimiter. For each term, all query

Table 8 Use of operators in queries

Operator	Description	# Queries	% Queries
	No operator	4,853,829	93.20
"	Use of quotes before/after a query term	178,014	3.42
cdef:	Finding class definitions	133,832	2.57
mdef:	Finding method definitions	89,107	1.71
idef:	Finding interface definitions	52,705	1.01
mcall:	Used but not defined by Koders	11,358	0.22
+…	Used but not defined by Koders	6,652	0.13
…*	* after a term, stemming operator	5,189	0.1
−…	− before a term, exclusion operator	4,761	0.09

operators contained in it were removed. Next, it was determined whether it is a *natural* term or a *code* term based on the following criteria:

- A term is a natural term if two conditions are met: it only contains the alphabets from the English language, and if the term is found in a dictionary of English words. We prepared this dictionary using an exhaustive list of words found in the *automatically generated inflection database* available at Web site for AGID word list (2010). The dictionary contained 252,379 unique English words.
- A term is a code term if it contains alphabets other than those of English language (such as numbers and symbols), or if the term is not found in the dictionary mentioned above.

With the above definitions for natural and code terms, a query is determined to be a *Natural* query if all of its terms are natural terms. A query is determined to be a *Code* query if all of its terms are code terms. A query is determined to be a *Hybrid* query if some of its terms are code and some are natural. Table 9 shows the statistics on these three query types we found in Koders log. We can make the following observations in Table 9:

- Code queries are the mostly used types of queries. Natural queries are less used than code queries. There are few queries that are Hybrid.
- Among the three types of queries, Code queries lead to the most of the downloads. About 21% of Code queries lead to a download. Compared to this, only about 12% Natural queries lead to a download. Hybrid queries are better than natural queries in terms of being followed by a download in a session.

In summary, it can be said that users seem to be more successful in getting relevant results with code queries (that do not contain any natural language term but only names and symbols used in code). Furthermore, it is quite rare that users combine natural terms with code like terms.

Query Reformulation Query reformulation measures how users are modifying their queries in search sessions. For our analysis of query reformulations, we use the definitions of modifications given in (Silverstein et al. 1999). Following that, we define a query to be totally new ("T") in a session if none of its terms matches the terms in a query that comes before it. Modifications to a query are classified into five kinds: Added Terms ("A"), meaning new terms are added to an existing query; Deleted Terms ("D"), meaning some terms are removed from an existing query; Operators modified ("O"), meaning either some operators are added or removed in a query with no other changes; and, Modified Otherwise ("M"), meaning changes

Table 9 Statistics on query types

Query type	# Queries	% Query	# $Downloads_F$	% Q2D
Code (C)	2,982,171	57.26	652,856	20.90
Natural (N)	1,756,080	33.72	215,871	12.29
Hybrid (H)	469,507	9.01	69,713	14.85

$Downloads_F$ = downloads that immediately follow a query type, % Q2D = percentage of the particular type of queries that lead to an immediate download

other than mentioned before. Table 10 shows the statistics on query reformulations in Koders based on these definitions of modifications.

Table 10 shows that the most widely used modification in the sessions is introducing a completely new query (more than 76% of all the queries that were modified). There are very few instances where users take an existing query and modify it by changing few terms or operators. In cases where users tend to modify an existing query, adding terms is the most common; next common one being combinations of modifications on terms and operators (identified by "M").

3.4 Comparison with Web search

Analysis of usage log of general purpose search engines is a widely research topic. It is not possible to compare every characteristics of Web search behavior with what we found in Koders. We provide comparison with results given in two studies of query logs. The first is the study of the query logs of the Altavista search engine that Silverstein et al. reported in Silverstein et al. (1999). Second one is a study of a query log of search engine in Adobe's developer portal done by Brandt et al. (2009).

Silverstein et al. reported that the query log they studied had a record of 993 million requests made in 43 days. Comparing that number with Koders, Koders has about 10 million requests (activities) over a period of one year. Naturally, being a code search engine, Koders is not that actively used as a general purpose search engine.

The average length of query terms per query is often reported to be 2.35 (Silverstein et al. 1999). The average length of queries in Koders is 1.31. It seems users in code search engines write even shorter queries compared to users in Web search. Silverstein et al. noted that there is very small duplication in queries, implying users searching for different things or using different words for same items. We noticed similar case in Koders, there were very few terms in the queries that were common among the users.

Sessions are generally found to be short and simple in Web search. Silverstein et al. noted that average number of queries in Web search is 2.02. Brandt et al. reported that average number of queries in sessions tend to be smaller among developers (1.45). In Koders, we found average number of queries (search activities) to be 2.16 (only including users that had search), that is higher than what both Silverstein and Brandt et al. found. About 63.7% of the sessions had only one activity in Web search. In Koders, 55.49% of sessions had only one activity. This indicates that sessions in code search are longer than sessions in Web search.

Table 10 Query reformulation (QR) statistics

QR	Description	% Q_{SES}	% Q_{MOD}
T	Totally changing the query	23.35	76.50
A	Adding terms	2.27	7.45
D	Deleting terms	1.61	5.28
O	Modifying operators only	1.31	4.31
M	Modifications other than those mentioned	1.97	6.46

MOD indicates if a query is modified, % Q_{SES} = percentage of all queries in sessions, % Q_{MOD} = percentage of those queries that are modified

The number of screen views in Koders seems to be quite similar to that seen in Web search. In Koders, about 86.35% of all search requests had a single page view. This is very similar to 85.2% of search request having a single screen view in Web search (Silverstein et al. 1999).

Brandt et al. report that the highest number of queries were Code queries (48%), second being Natural queries (38%), and third being mixed (Hybrid) queries (14%). In Koders, we found the same order of popularity among types, but there were more code queries (57.26%), and fewer Natural (33.72%) and Hybrid queries (9.01%). Since Koders is a code search engine, unlike the Adobe's developer's portal (whose query log Brandt et al. analyzed), it is quite plausible that users are using more Code queries compared to others. One commonality is that in both search engines, there are fewer instances where users mix natural terms with code terms in their queries.

Developers using the Web are reported to refine a query very rarely (Brandt et al. 2009). In Koders we noted this to be true for the case of modifications on an existing query; there were very few cases where a query was modified by adding or removing terms. In Koders, modifications to a query seem to be simpler than those found on the Web. Silverstein et al. report that the most common modification to a query on the Web involves complex modifications (not covered by simpler ones such as just adding or deleting terms). However, in Koders, the most common modification was changing the query entirely to a new query. More complex modifications only accounted for about 1.97% of the queries that were modified. Compared to 53.2% of complex modifications on the Web queries, this number is quite low. One thing that is noticeable in Koders is the use of operator for language. It seems, this particular feature is very widely used. The use of operators in Koders seems to be less than in Web search. In Koders about 93% of the queries did not have any operator, where as in Web search about 80% of queries did not have any operator (Silverstein et al. 1999).

To summarize, we can say that usage behavior is quite similar between Koders and search on the Web. Users in these search engines have short queries and simple sessions. There is very little chance that users refine their existing queries. Users in Koders also had few unique characteristics. For example, a lot of the queries had refinements for language type (probably because this is a unique feature applicable to source code not available in Web search engines), and fewer use of other operators compared to Web search. Furthermore, users in Koders seem to be using simpler query refinements than users in Web search engine.

4 Topic Modeling

In the previous section we looked at several variables that measured the usage behavior in Koders. In this section we seek an answer to the question: What are users searching for? Since our source of information to answer this question contains a large collection of queries given by users, we need a technique that allows us to get a high level summary of fine grained information stored in the queries. For this purpose we use a probabilistic topic modeling method named Latent Dirichlet Allocation (LDA) (Blei et al. 2003).

LDA is a popular topic modeling technique. The benefits of LDA over other topic modeling techniques is that it is an unsupervised method that requires no learning data. Topics emerge as sets of words that are probabilistically correlated in terms of

their co-existence in the same documents. LDA works with the following underlying model (adapted from Griffiths and Steyvers 2004):

1. A document can deal with multiple topics, and the words in the document reflect the particular set of topics it addresses, and
2. Each topic can be viewed as a probability distribution over words, and a document as a probabilistic mixture of these topics.

The mechanics of applying LDA has been shown in various kinds of corpora ranging from scientific papers to source code (Griffiths and Steyvers 2004; Baldi et al. 2008; Linstead et al. 2007a, b, 2009; Maskeri et al. 2008). Underlying details and its mathematical underpinnings are well described in Griffiths and Steyvers (2004) and Blei et al. (2003). We summarize the important points in the context of our log analysis.

In the LDA model for text, the data consist of a set of documents. The length of each document is known, and each document is treated as a bag of words. Applying LDA on a corpus requires three things as input: (i) A list of documents with corresponding bag of words (document X word matrix) , (ii) the number of fixed topics to extract, and (iii) parameters (hyperparameters over the topic-document and the word-topic distributions, α and β) for the LDA to tune the mining process to suit the nature of the corpus.

1. *Corpus:* We limit our corpus to a subset from the Koders's usage log. Our dataset consists of activities from users mostly searching in the Java programming language. This is primarily to address the fact that topic identification is a process of inference that needs some expertise in the domain. Selecting users searching mostly for Java code helps making better judgements in extracting the Java topics they were searching in. To get this subset of users searching in Java, first we select all the users who had at least one search activity with Java as the selected language. From this set, we then select those who have the largest number of search activities in Java among all their activities, or those who have the second largest number of search activities in Java provided that their largest number of search activities did not have any language specified. With this criteria our corpus for topic modeling consisted of 1,055,105 search and 755,588 download activities from 291,839 users.
2. *Document and words:* We model a document required for LDA as the collection of all queries made by a user. Thus we obtain a collection of 291,839 documents corresponding to all the users in our corpus. This document collection is assumed to contain a set of latent topics that we will discover as a result of the topic modeling. These topics can be considered as the topics that the users are searching for in the system.
3. *Number of Topics:* Non-parametric Bayesian and other methods exist to try to infer the suitable number of topics from the data. However results from such methods have been found to contradict the numbers predicted by human experts while applying LDA to software corpus (Maskeri et al. 2008). Thus, we manually set the numbers in our study.
4. *Hyperparameters:* Two hyperparameters are required to tune the distribution in the LDA model to match the nature of the corpus. In our study we fixed α at

0.5 and β at 0.1 as these values seem to result in the most meaningful assignment of words to topics. For further mathematical details on these hyperparameters please refer to Griffiths and Steyvers (2004).

For this study, we used the LDA topic modeling feature from the Dragon Toolkit (Zhou et al. 2007).

Data Processing for LDA The terms in individual queries are processed to extract more meaningful terms since the raw queries ranged from ambiguous symbols to code snippets. The words in a document are produced by first splitting all the queries into terms using whitespace as the delimiter, and, second splitting each of these terms further on non alphabetic characters (eg: '_', '–'). All the terms in the log were in lower case so we could not do code specific term extraction such as camel-case splitting. It should be noted that we do not use any stop word in our processing. After this step we have the document-word matrix ready to be fed into the topic modeling toolkit.

4.1 Results—Latent Topics

We applied LDA on our corpus with varying number of topics, starting with 50 and increasing this number to 100, 150 and 500. We found that increasing the number of topics results in more granular topics. For example, when the number of topics is set to 500, most of them are sub-topics of those that are found when the number of topics is 50.

Table 11 shows five sample topics that emerged when LDA was applied with 50 as the number of topics. A table with full list of 50 topics is given in the Appendix (Table 19). The topic description and the code for each topic were manually assigned after looking at (i) the top 20 most probable words assigned to the topic, and (ii) some randomly selected search activities from users who were assigned the topics. This process makes interpretation of the topic more reasonable when the top 20 words assigned to the topic are not enough to interpret it. We use the topic code as an easy mnemonic while presenting it throughout the paper.

Table 12 shows some sample topics that emerged when LDA was applied to mine 500 topics. It shows how the set of topics picked from the set of 500 can be seen as a fine grained breakdown of topics from the set of 50 shown in Table 11. 'Data Structure' is one of the topics identified when the number of topics is 50; when the number is 500, several variants of data structures can be identified, such as AVL tree and B-Tree. Table 20 in the Appendix shows another such example where the topic 'Network' breaks down into finer ones, such as http, ftp, packet sniffing etc. Many of these fine-grained topics are not visible when the number of topics is set to 50.

Table 11 Five sample topics mined from the query logs

Topic code	Description	Words
Audio	Working with audio and sound	Compare, control, encode, audio, decode
DataStr	Data structures	List, object, arraylist, map, vector
Network	Networking, FTP	Client, server, ftp, socket, iso
Files	Working with files	File, read, files, create, write
GUI	Swing and AWT GUI	Swing, jtable, applet, awt, window

Table 12 Sample topics related to data structures obtained when number of topics fixed at 500

Data structure topics	
AVL Tree	Tree, avl, minimum, spanning, avltree
B-Tree	Tree, b, btree, trie, suffix
Queue	Queue, priority, fifo, circular, priorityqueue
Lists	List, linked, linkedlist, sorted, lists
Heap	Max, heap, min, unix, chromaticity
Graph	Graph, vertex, dfs, edge, salvo

Two such examples are the topics 'initializing the modem' and 'listing usb devices'. This is also an expected behavior of LDA. In summary, this captures an important characteristic of LDA in identifying topics at various levels of granularity.

For the reminder of the paper, the discussion and results refer to the application of LDA with number of topics set to 50.

4.2 Topic Categories

We found that the 50 topics that we mined from the usage log can be generically placed under one of the following six categories.

1. *Applications:* These were topics where users were looking at specific applications. For example 'Calendar Scheduling', 'Multimedia' and 'Mobile Games'.
2. *Programming Tasks:* These topics mostly pertained to general programming tasks applicable to many domains and systems. Sample topics include; data structures, date time functionality, object relational mapping, document formats, working with files, string, xml etc.
3. *Frameworks:* This category represents topics that captured well known Java frameworks in use. Examples include JBoss, Eclipse, and Lucene.
4. *Java/JDK Libraries:* These topics represent common features available in the JDK. One prevalent topic under this category has the term 'java' as the most common word. Queries that matched this topic were either about the core JDK libraries such as the `java.lang.*` or users were simply adding java as a qualifier term along with other query terms. Other topics under this include GUI APIs such as Swing and AWT, and Database APIs such as JDBC.
5. *Form Centric:* This category of topics captured a different characteristics about the queries the user made compared to other topics. Rather than capturing what the users were searching for, topics under this category captured how they were expressing their queries. These topics pertained to the various forms of the queries users subscribe to while expressing their search need. Among these topics we found four distinct forms in which users express their queries:
   - Three of the 50 topics contained words (such as "How", "Source", "Code") that are often used in writing a verbose query. We looked at several queries that belonged to these three topics and found that they start with phrases such as; "How to use ..", "Source code for .." etc. Based on this information we interpreted these topics to be describing forms of queries where natural language expressions were used. LDA was able to detect these topics without any preprocessing. These topics appear with a prefix "NL." in their names in Table 19.

- Others topics related to the form of queries captured the use of query operators, “mdef:” and “cdef:”.
- We also found topics that seemed to be capturing the common terms used in the FQNs (fully qualified names) of Java entities. We include these topics under the category “Form Centric” with the interpretation that using FQNs is also a common technique to search for relevant source code.
- One of the topics (Topic ‘Jkw’ in Table 19) captured the use of the Java language keywords to express structure in the query. For example, a query such as ‘extends iactionlistener’ uses the keyword ‘extends’ where the user is possibly trying to find interfaces that extend the interface `IActionListener`.

6. *Unknown:* These were topics that we could not interpret easily. The most probable words assigned for these topics just seemed to be a combination of random words that did not seem to capture anything in particular. We group these topics into this category.

In the rest of this section we will look at several statistics on users, search activities and downloads associated with the topics. Our goal behind looking at these statistics is to get an idea about the popularity of topics based on the numbers of users and activities. Getting these statistics also leads us to understand the prevalence of users and activities in the topic categories.

While referring to the individual topics we will be using the mnemonics given to each topic that is listed in Table 19.

4.3 Users and Topics

Applying LDA results in a probabilistic assignment of the topics to each document in the corpus. In our case, each document represents the collection of words from all the queries a corresponding user makes. Consequently, the topics are assigned to the users. This results in a probability distribution of the topics assigned to each user. This parallels the fact that a user might be looking across more than one topic with varying degree of interest. To select the most likely topics a user searched in, we discard all the topics with the lowest probability for the user. This is based on our observation that when the topics assigned to a user are sorted in a descending order, based on the their probabilities assigned to a user, they follow a long tailed distribution. There are usually a larger number of least probable topics in this tail, all having the same probability value. This value varies among the users, but a common thing is that all users have such low probable topics assigned to them. Discarding these lower end topics from the distribution allows us to have a good approximation of the topics that a user searched in. Getting these assignments of topics to users allows us to look at two statistics: the number of users under a topic, and the number of topics per user. We discuss these statistics below.

Given the list of the most likely topics for each user in the corpus, the user count for a topic (say *tp*) is the sum of all unique users who has been assigned the topic *tp* as one of their most likely topics. In other words, if a topic has a user searching under it, its user count increases by one for each unique user. Getting this count gives us the first insight on the popularity of the topics among the users. Figure 1 shows this result; for each topic it shows the percentage of the total users who searched under the topic. Few topics were more popular than others. The top three being ‘JAVA’ (7.8% of

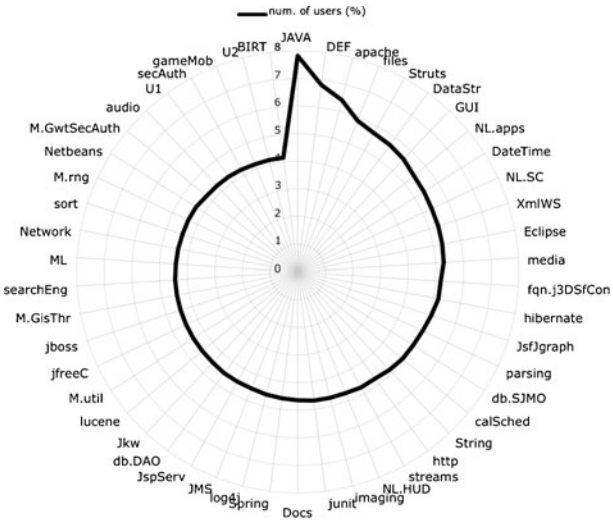

Fig. 1 Percentage of total users under different topics

users searched under this topic), 'DEF' (6.8% of users), and 'apache' (6.4%). The lowest three topics with least users were 'gameMob' (4.2% of users) , 'U2' (4.2% of users) and 'BIRT' (4.2% of users). These results indicate that users do not search in all of the topics. Each topic had users that ranged from 4% to 8% of the total users, and most of the topics had users that ranged from 4% to 5%.

We plot the number of users grouped by the number of topics to see how many topics users search in. Figure 2 shows this plot. We can see that this distribution is exponential. A very large number of users search in either one or two topics only, this is not surprising as we have seen that most of the search sessions are short in Koders. Almost all of the users searched in less than five topics. There was a very small number of users that searched almost in 40 topics (out of 50).

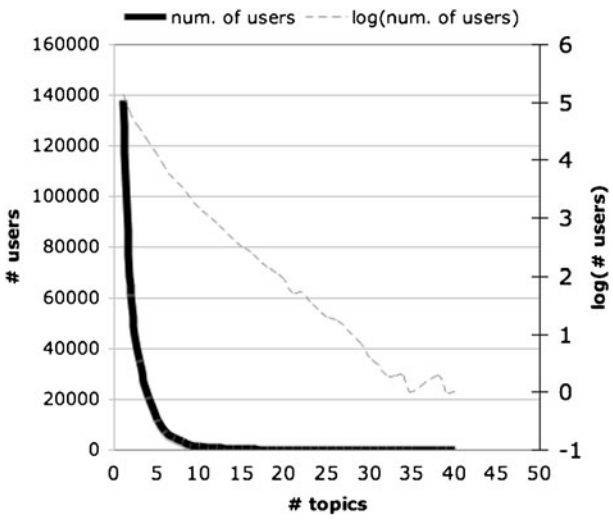

Fig. 2 Distribution of number of topics among users

4.4 Search, Downloads and Topics

In the previous section we matched users with the most likely topics they searched in. In this section we seek to rank the topics based on counts of search activities they have, and count of downloads that follow them. With this we will be able to see the topics that are popular among users, the topics that had the highest search activities, and the topics that led to most of the downloads.

Search Activities Under a Topic With each user being assigned to a list of topics, we can further match the queries from the user with the topics assigned to them. This assignment of topics to individual queries allows us to get the counts of search activities under each topic. The distribution of search counts across topics gives further insight on the prevalence of the topics among the users.

For each query from a user, we compute its similarity with topics that were assigned to the user. We then select the most similar topic (say tp) that the query matches with and count it as a search activity under the topic tp. We use a simple similarity metric defined as follows. For a query q composed of a list of terms $(t_1, t_2, .., t_n)$, issued by a user U who was been assigned a set of topics T, similarity of q to a topic tp from T is given as:

$$P(q|tp) = P(tp|U) \prod_{i=1}^{n} P(tr_i|tp) \tag{1}$$

where, $P(tr_i|tp)$ = probability of term (word) tr_i appearing in topic tp, and $P(tp|U)$ = probability of topic tp being assigned to user U. Both of these probabilities can be obtained from the output of the topic modeling.

This technique of associating queries with topics can result in different assignment of topics for exact same query for two different users. The matching depends on the topics that a user has searched in. Assignment of the query to a topic is limited to only the topics the users have been associated with.

Downloads that Follow a Topic We assume that an immediate download activity that follows a search activity is a download related to the preceding search activity. With this, we can associate a download with a topic by picking the topic that best matches the query (corresponding to the search activity) that precedes the download. This gives us the count of downloads that followed a topic. A higher count of downloads under a topic could suggest that users were successful in finding usable results under the topic and vice versa.

Results Figure 3 shows the distribution of the search and download counts among all topics. It shows that most of the topics get similar percentage of search activities, ranging from 1.5 to 2.5%, remaining top five topics had 2.5–4% of all search activities. Table 13 shows the top ten and lowermost ten of the topics based on how many users, search and downloads these topics get. The top ten topics with largest number of search activities are: JAVA, DEF, Eclipse, GUI, Network, Jkw, files, imaging, hibernate, and sort.

Looking at the counts of immediate downloads that followed a search in a topic (Fig. 3) shows a slightly different ranking of topics. The range is similar; from 1.5 to 3.6%. The top ten topics, with respect to number of immediate downloads that

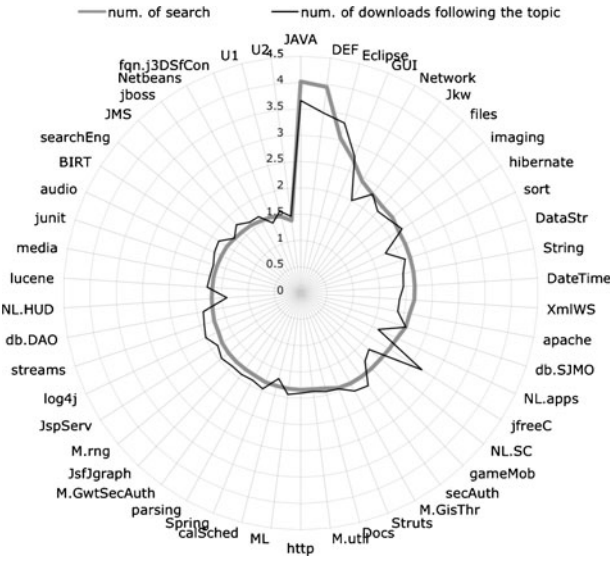

Fig. 3 Download and searches across topics

followed the search in those topics, are: JAVA, DEF, Eclipse, GUI, jfreeC, Jkw, hibernate, Imaging, secAuth, and files. Topics 'sort' and 'Network' disappeared from the top 10 list here, taken over by 'jfreeC' and 'secAuth'. Also the topics 'NL.apps', 'NL.SC', 'NL.HUD' are ranked lower compared to others with respect to the immediate count of downloads that follow them. This supports our observation in Section 3 that queries with natural terms lead to lower downloads.

Topics 'JAVA' and 'DEF' seem special since they are the top two topics with largest proportion of users, search and download activities. 'DEF' is a topic that captures two things: (i) use of query operators to find definitions in code, and (ii) the terms from this topic match very closely the examples and popular terms given in the Koders's Web site. These observations support our previous hypothesis about users trying popular queries that Koders lists in its Web site. The high rank for this topic could be explained by the fact that most users who come and use the search engine try some of the examples given in the Web site. 'JAVA' seems special as it relates to core JDK packages, and also the case where the term 'java' appears in

Table 13 Highest and lowest ranked topics

Top 10			Low 10		
Users	Search	Download	Users	Search	Download
JAVA	JAVA	JAVA	BIRT	U2	NL.HUD
DEF	DEF	DEF	U2	U1	fqn.j3DSfCon
apache	Eclipse	Eclipse	gameMob	fqn.j3DSfCon	U2
files	GUI	GUI	searchEng	Netbeans	U1
String	Network	jfreeC	U1	jboss	NL.apps
DataStr	Jkw	Jkw	audio	JMS	searchEng
GUI	files	hibernate	lucene	searchEng	Netbeans
Network	imaging	Imaging	ML	BIRT	jboss
DateTime	hibernate	secAuth	M.GwtSecAuth	audio	calSched
NL.HUD	sort	files	secAuth	junit	NL.SC

the queries. It is quite likely that both searches on the core Java packages and using the term 'java' along with other terms yields large number of search and download activities. This might also be a side effect of limiting our corpus to the users searching in the Java language.

Another interesting observation to make is that topics that were identified as unknowns ('U1', 'U2') are in the lower most part of the ranks. Also, among the lower ranked topics are specific topics such as 'Mobile Games and GUI' and 'Searching' (search engines).

4.5 Prevalence in Topic Categories

Table 14 shows all the above topic categories and how each of the 50 topics from Table 19 falls under these categories. We can see that 'Programming Tasks' and 'Frameworks' contain the majority of topics.

Figure 4 shows the prevalence of each of these categories based on the counts of activities (search and download) and users in all the topics that fall under a category. In a nutshell, it captures what users of Koders look at and get engaged in. The top two categories are 'Programming tasks' and 'Frameworks'. The rest are 'Java/JDK libraries' and 'Form centric' topics. 'Applications' category is the least prevalent. This means that there are not many activities and users searching for domain specific applications. Or, at least such topics are not visible when we look at a smaller number of topics. Conversely, all the frameworks represent solutions built for some specific domain. The prevalence of frameworks in the usage log leads us to conclude that there are two categories of users searching for specific application domains:

1. Those who know the projects to look at and use names from the projects in their search queries (users under topic 'Frameworks'), and
2. Those who are not yet familiar with the right projects to look at but have search needs pertaining to specific domains (users under topic 'Applications').

Besides providing these insights on the varying prevalence of topic categories among the users, the categorization of the topic under categories provides one important hint regarding the expression needs of code search engine users. The category 'Form Centric' captures some common ways in which users express their queries. Topics under these category illustrate the fact that users often qualify their queries with operators to find definitions (topic 'DEF' in Table 19) and keywords such as 'extends' and 'implements' to express structure. They also suggest that FQNs could have been used as quick mnemonics to retrieve code (Topic 'fqn.j3DSfCon' in Table 19) and terms such as "how to", "source code", "example" are often associated with verbose natural language like queries.

Table 14 Topic category statistics

Category	# of topics	Users	Searches	Downloads followed
Applications	4	51,723	74,828	20,109
Programming tasks	15	216,989	315,322	86,035
Form centric	6	93,695	142,321	35,011
Java/JDK libraries	5	80,406	133,898	36,744
Frameworks	15	205,630	299,281	89,632
Unknown	5	65,849	89,455	26,439

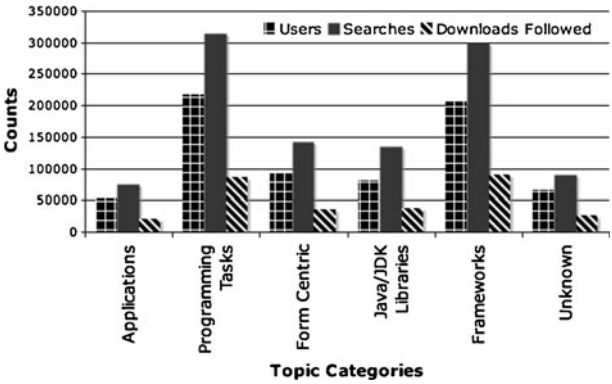

Fig. 4 Prevalence in topic categories

5 Query Forms

In Section 3 we saw that queries in Koders can be categorized as Code, Natural or Hybrid query based on their linguistic form. The topic category 'Form Centric' suggested that there are could be more idiomatic forms of queries that are used during code search, for example use of FQNs to retrieve code entities. To delve deeper inside this issue we sought answers to questions related to the form of code queries; in particular, what do they look like and what type of search result they find. The motivations behind these questions were to identify the ways in which users express themselves with queries while searching for code, and to see if there are certain query forms that are effective compared to others. For this purpose, we performed a qualitative analysis on 150 randomly selected sessions from the log.

We use a slightly different definition of session for our analysis in this section. Since we are interested in the query forms that lead to downloads effectively and efficiently, we focus on a set of search activities that precedes a download. We define a search-download session as a consecutive list of search activities that end with a download. We performed a qualitative analysis on 150 randomly selected search-download sessions from the query log.

We obtained the 150 sample sessions by randomly selecting a query from various topics mined from the usage log, and selecting the search-download session the query belonged to. Being able to sample queries across these topics ensures that the sample we consider is representative of the range of topics users search in. We selected ten queries from 15 different topics, resulting in 150 queries. For each of these 150 queries we looked at their corresponding search-download session. The 15 different topics span across various categories such as, Programming Tasks (Network Concepts, Working with Strings, Data Structures), Frameworks (Eclipse, Apache libraries), and those pertaining to ways users express queries (using natural prose to find systems and asking "how to" questions). The spreadsheet that contains the result of the analysis is available online at: http://wiki.github.com/sushil/koders-loganalysis/ese2010.

Table 15 shows a sample search-download session. It shows that the randomly selected query was the sixth activity in the session (starting with a $\rightarrow$). The second column shows the activities. It shows the query for a search activity and the downloaded code's unique identifier for a download activity (activity D_1). Topics that are assigned to the query are shown in the third column. In this session, we can see a user

Table 15 Example of a sample search-download session under investigation

	Activities	Topic assigned
S_1	"Date range"	Date time
S_2	How validate the number	Natural language
	Days data range	Query
S_3	"Date range validation"	Date time
S_4	"Date range"	Date time
S_5	Date range	Date time
$\rightarrow S_6$	Number days date range	Date time
D_1	Downloaded <file_id>	

was trying to find a feature that would give the number of days in a date range. The user issued six queries and then downloaded a result she found relevant. Looking at the file that was downloaded revealed that it was a class named `DateUtil` with a method named `getDataRange` that takes two dates and a flag, and returns a `long` value as the range. This demonstrates that looking at the activities in the session, the topic assigned to the queries in the session, and the file that was downloaded allows us to perform a reasonable qualitative analysis.

5.1 Lexical Structure

Observing all the queries in the sample revealed that the lexical structure of the terms in the queries followed five unique and recurring patterns. This allowed us to encode the queries into five basic forms:

1. **NL:** Query formulated as a verbose natural language phrase.
2. **Term:** Collection of few natural language words.
3. **Name:** Mostly a single word that resembles a name used in source code.
4. **Acronym:** short length terms denoting common acronyms such as "emf" standing for "Eclipse Modeling Framework".
5. **Code:** Query that appeared to be a direct copy of few lines of code with symbols and operators.

Table 16 shows some examples of these various lexical forms of queries. These lexical forms show the range of verbosity that queries can exhibit. They can be very terse as in the acronyms or very verbose as in natural language phrase.

5.2 Result Types

Just as the lexical nature of the query terms reveals the form of the queries, the meaning of the terms and interpretation of the terms together in a query allows us to make an estimate about the kinds of results users were expressing in their queries. We do this by looking at the individual 150 queries, the queries that surround them in their respective session, and downloads they ended with (in case where they existed). This allowed us to encode the kinds of results being sought for in the queries into four categories:

1. **System:** self contained applications or parts of applications.
2. **Feature:** Some functionality of a software system that achieves a particular task.
3. **Entity:** A smaller unit in a program such as a class or a method.

Table 16 Example of various forms of queries

Form	Examples (each query separated by a comma)
Acronym	emf, dao, crud
Code	Catch(sqlexception, substring(, byte[]
Name	Filewriter, tchat, javax.media.datasink
NL	Storing vector file format, example programs for datagram, read value from xml file
Term	Drag drop, string date, file reading

4. **Line:** A particular line or location in source code.

Table 17 shows some examples of these various kinds of results sought using queries. These various kinds of results show that users seek results at various levels of granularity ranging from an entire system to a particular line in a code.

Table 18 shows the cross-tabulation of the counts of the randomly selected 150 queries across the two categories we encoded for: the lexical form and result type of queries. The most common lexical form was *Name* (40% of all queries); second being *Term* (32% of all queries). The most common search was for finding an *Entity* defined in the code (48% of all queries); second being finding a *Feature* (32% of all queries) that implements some piece of functionality. Within the sample we collected it seems that the users mainly searched for entities using names and features using terms.

5.3 Form and Relevance

We looked into statistics on download activities for the various forms of queries (from the sample of 150 mentioned earlier) to gain an understanding on what seems to produce relevant and possibly usable results. For this purpose we devised the following definitions.

- *Relevant Results:* A search result for a query in a search session is relevant (and possibly usable) if a download activity follows the query in the session. We associate download with relevancy assuming that users download code only if they think it to be usable. This is quite similar to the assumption in general purpose search that click-through is a significant indicator of relevance (Joachims 2002; Joachims et al. 2007).
- *Efficient Query:* A query is efficient if it produces a download as the next immediate activity in a session.

Table 17 Example of various result types expressed in a query

Form	Examples (each query separated by a comma)
Entity	Hashtable, filewriter, cdef:jpegdecoder
Feature	Parse the url data, lucene search, pad
Line	Catch(sqlexception, byte[], //special values for whereclauses
System	Media player, example programs for datagram, apache torque

Table 18 Lexical form and result types of queries in code search

Form	Intent					
	Entity	Feature	Line	System	*S*	%
Acronym	2	2	0	8	12	8.00
Code	0	1	4	0	5	3.33
Name	55	6	0	0	61	40.67
NL	2	14	1	7	24	16.00
Term	13	25	0	10	48	32.00
S	72	48	5	25	150	
%	48	32.00	3.33	16.67		

– *Effective Query:* A query is effective if it produces a relevant download after the query in a session. We inspect the downloaded code to see if the download was indeed relevant to the query in the session. An efficient query might not be effective if it results in an immediate download that is not relevant to the query.

We measure effectiveness of a given type of a query (Q-type) by a metric D_r/S; where, D_r is the count of relevant downloads all Q-type queries produced in the search-download sessions, and S is the count of all Q-type queries in the sessions. Similarly, we measure efficiency by a metric D_i/S; where, D_i is the count of all immediate downloads all Q-type queries produced in our search sessions.

Figure 5 plots the efficiency and effectiveness for all of the lexical forms, and Fig. 6 for all of the result types. These figures also show the percentage of search activities across different lexical forms and result types. Name queries and queries asking for Entities are the most effective and efficient. Feature queries are ranked second in terms of effectiveness and efficiency. Term queries are second to Name queries in terms of being efficient, but much less effective. Term queries are ranked fourth in terms of effectiveness. These observations can be summarized as follows:

> Users mostly are looking for entities defined in programs, and features that implement some behavior. They mostly issue queries that look like names of the entities as they are defined in the code. In absence of the knowledge about the defined names, they issue short queries that consist of natural language terms. Users get to relevant results with much less effort when they use queries that include the names of code entities. While users seem to look into the results

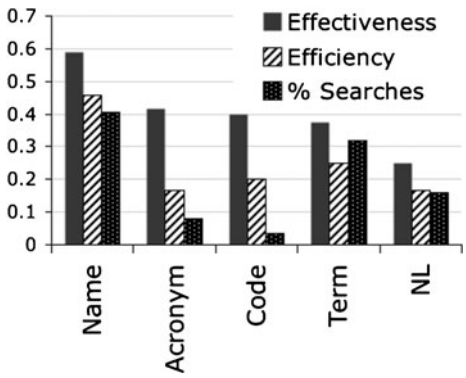

Fig. 5 Effectiveness and efficiency of queries with different lexical forms

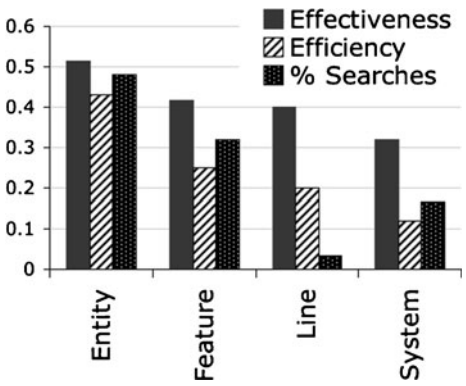

Fig. 6 Effectiveness and efficiency of queries for different result types

they get with queries having natural language terms, these queries mostly fail to yield relevant results compared to other forms of queries.

6 Discussion

In this section we reflect upon some of our findings on usage, topics and query forms, to provide some guidelines for the design of code search engines.

6.1 Usage

Supporting Users who Give up Quickly We noticed that most of the users in Koders do not look beyond the first result page. This could mean two things; they found the result they were looking for in the first page, or they tend not to browse through search results to find relevant results if they do not find one in the first result page. We believe it to be the second case because of reason that we explain next. If a relevant result was found in the first page itself it should have resulted in some download (note that download here means result view by clicking on the code found). Since more than 64% of the users had no downloads at all, this could mean that many users did not find relevant results in the first page. It could be argued that users could have fulfilled their information need by just looking at the search results without actually downloading results. However, we believe this is not the case either. In our own experience of using Koders, the result page often showed irrelevant parts from the code (mostly few lines from the top of the source code file). Therefore, there is an indication that users tend to abandon their search if they find irrelevant results. This implies that code search engines should have better ranking techniques and presentation of results to reduce abandonment. Source code being different both in content and structure, there could be techniques that are specific to code search engines that can improve efficiency. For example, code specific ranking heuristics (Linstead et al. 2009; Bajracharya et al. 2007) and improvements in user interface specific to code search engines (Hoffmann et al. 2007; Bajracharya et al. 2010b).

Support for Programming Language Specific Queries There is a strong indication in the analysis of the usage log that users care about what language they want to search

for. Most of the queries that users gave were Code queries. This implies that a code search engine should offer different set of features based on the language that users have selected. For example, searching for class definitions is not meaningful for users searching in C programming language. Koders could offer searching for 'Structs' or 'Unions' instead for users searching in C.

Support for Browsing Code After Search Most of the download activities that existed in the log followed another download, indicating users tend to browse code after they find some result. Users even expected the presence of operators to express relations that exist in source code, for example method calls. This implies that along with the search facility, a code search engine should also have strong support for browsing and finding code using relations in the code.

Leveraging Query Logs for Effective Retrieval Various forms of relevance feedback mechanisms (Grossman et al. 2004; Baeza-Yates and Ribeiro-Neto 1999) can be devised based on the information stored in a search engine's usage log. The underlying idea behind usage log based relevance feedback is that term associations mined from queries and successive click-through in sessions (downloads in case of Koders) can be applied to generate more relevant terms for query expansion and retrieval. Such techniques have proven to work in general Web search (Joachims 2002; Cui et al. 2003; Zhu and Gruenwald 2005), and could be applied to code search engines as well. One area of improvement could be in the user interface to support relevance feedback and retrieval by reformulation. These techniques have been known to be successful in past; both in the case of software and general information retrieval (Henninger 1997; Koenemann and Belkin 1996).

Support for Natural Queries Our results show that although users tend to use natural queries, and they are less effective in leading to a download. This could be because of the well known vocabulary problem in information retrieval (Furnas et al. 1987). This indicates that there should be better support for natural queries in code search engines. Automatic query expansion or semantic indexing techniques could help (Xu and Croft 1996; Zhu and Gruenwald 2005; Zazo et al. 2005; Bajracharya et al. 2010a).

6.2 Topic Modeling

Facilitating Search on Specific Topics The statistics on users and topics from Section 4.3 (Fig. 1) suggests that a large number of users look at few topics only. This is not surprising since we saw that users tend to have very short sessions with few queries. This observation implies that users look for specific topics in sessions. Therefore, a code search engine can facilitate searching in specific topics, or support filtering results by topics. Results from topic modeling on source code show that it is possible to extract meaningful topics from source code automatically (Baldi et al. 2008; Kuhn et al. 2007). Code search engines should leverage such information.

Supporting Users who do not Know what to Look for The top ten list in Table 13 shows that the ranked topics are almost the same between search and download counts but that they differ when it comes to users count. Many users searched in topics such as 'apache', 'strings', and 'DataStr'. These topics neither have the largest

number of searches nor downloads. In place of these we see that topics such as 'GUI', 'jFreeC', 'Jkw' are ranked high on both search and download. One interpretation of this observation is that many users look at very common programming idioms like data structures and strings manipulation, but these are not the ones that yield the largest number of the activities. There seems to be users who use the search engine extensively, and they do not look for these common programming idioms. Rather, they look at large frameworks like Eclipse, and get comparatively more downloads from specific projects (such as Eclipse, Jfree Chart). This could imply that users who stick to the system and use it more are looking at very specific projects, and not at general programming idioms as such. In other words, *users who find code search engines like Koders usable are those who already know where (or what) to look for*. This leads to the question that how can a code search engine support users who do not know what solutions exists for the problems they have, or those who do not know what exactly to search?

For example, the topic 'lucene' (a popular information retrieval engine for Java) could be the one that users searching in the topic 'searchEng' ought to look at. This suggests that there could be ways to assist users searching for specific application domains by giving them feedback so they could direct their queries to one of the frameworks that has solutions pertaining to their interest. Perhaps, suggesting the terms from 'lucene' to the users of 'searchEng' could result in better search performance among the users searching for search engines.

Support for Natural Language Expressions in the Queries All topics related to expressing queries in a more natural language form ('NL.HUD', 'NL.apps', and 'NL.SC') are ranked low in terms of Search and Download counts. However, they are not ranked low in terms of Users. In fact, 'NL.HUD' is one of the high ranked topics with respect to user counts. This observation can be interpreted as follows. *Large number of users who come and use the search engine try natural language queries (ranked high on user count) but they do not use it extensively (not ranked high on search activities) as they don't seem to lead to usable results (ranked low on downloads)*. An important implication of this observations is that certain linguistic feature in queries need to be treated specially. For example, when a query contains a phrase such as "How to …" a code search engine should understand that a user is interested in knowing about performing certain tasks, possibly trying to find an API to do something as opposed to find implementations.

6.3 Query Forms

Understanding the Underlying Structure in Code We encountered a few odd cases during the analysis of 150 sessions that were outliers from the general statistics. For example, name queries were found to be most effective and efficient. However, in one of the sessions a user had a name query 'xmlserializer' that followed an immediate download which was not relevant. Koders retrieved a source file that had the query term inside a comment but the code was not about xml serialization. This indicates that a code search engine needs to treat certain sections in code differently from others. A match of a query term in a comment could be less valuable than a match in a method name. Koders, in particular, seems to ignore this fact; many of the queries we tried in Koders gave results that matched source code portions that

we found to be irrelevant. For example a match in copyright information header on top of the file, or comments describing functionality that were no longer available in the code.

Supporting Code Abbreviations Use of acronyms in the queries implies that techniques to expand and contract abbreviations in code might be used to improve code search (Liu and Lethbridge 2001). Such techniques could be used as a part of a retrieval scheme where an abbreviation is expanded to match more terms, or commonly occurring words would be contracted to match abbreviations in code.

Generating the Right Code Snippet to Show in the Result Page In another session in our analysis of query forms, a name query 'org.apache.naming.config. xmlconfigurator' did not have any downloads. Upon running this query in Koders ourselves, we noticed that none of the results in the first page had any cue that suggested that the results were relevant to the query. However, clicking on the first result showed us that it was a class that imports `XmlConfigurator`, perhaps something the original user would have found useful. This suggests that lack of useful cues in the snippets shown in search result page will fail to make them useful to the users.

Support for Retrieving Various Result Types The observation that users seek results at various levels of granularity suggests that a single way of presenting results may not always work. For example, Koders and most other code search engines present the results at the level of a file. This result could be useful for someone interested in locating a file but not much of a use for a user seeking a feature. A feature usually spans multiple files and could have multiple dependencies on external libraries. A user seeking a feature would benefit if she can see the entire context of that feature (for example, related files and dependencies) in the result. Code search engines could use techniques for feature location that have been studied extensively in software engineering (Marcus et al. 2004; Poshyvanyk and Marcus 2007; Poshyvanyk et al. 2007).

7 Validity

In all three analyses (usage, topic modeling, and query forms analysis), we held certain assumptions and made some interpretations that could lead to validity threats. In this section we list the major issues that could pose some threats to the validity of our arguments and findings in the paper.

7.1 Usage Analysis

Accuracy of Identifying Users The analysis of usage we presented in Section 3 depends critically on the accuracy of identifying users in the log. When we contacted Koders during the time we got the query logs, they claimed their scheme for identifying users to be quite robust. We do not know much, besides the fact that Koders uses a combination of IP addresses and cookies stored in browsers to identify users.

A weak technique to identify users, or the practice among users to regularly delete cookies could affect the results related to activity. For example, the measurement of active days will be severely affected. Unfortunately, it is not possible to find how accurate the information on users is in the Koders log.

Noise Introduced by Activities from Bots Some of the activities in Koders might have been because of bots crawling Koders. For example, a large number of users having no search activity, and few users having a large number of search and download activities could be because of bots. We detected two users who were routinely searching for only the example queries given in Koders Web site. We removed these users as outliers. There does not seem to be an easy way to detect activities due to bots in the system, and thus what effect they have on the results is unknown.

Limitations of Log Analysis It is well known that studying query logs alone is not enough to fully understand usage behavior of the search engines (Grimes et al. 2007; Aula and Nordhausen 2006). Results from the analysis we present in this paper complement those in Umarji et al. (2008), which used a completely different methodology. Since the findings in both of these studies point to the same general conclusion that current code search engines are not that effective, we believe the deeper analysis in this paper is not only valid, but provides valuable insights for what the problems may be and how to overcome them.

Generalizing to Other Search Engines It should be noted that although similar in functionality, Koders is quite different from other search engines in terms of its user interface and presentation of results. Because of this, it is also not possible to generalize our findings for other code search engines such as Krugle (Web site for Krugle 2010) and Google Code Search (Web site for Google Code Search 2010). More work is needed to understand the effect of user interface and presentation (of results) on usage behavior across various code search engines that are available today.

7.2 Topic Modeling

Choice of LDA Topic Modeling to Mine Topics We chose to use LDA topic modeling as a method to mine topics from the query log because of two major reasons. First, topic modeling using LDA has been applied to solve a wide range of problems in software engineering such as: statistical debugging (Andrzejewski et al. 2007), mining business topics (Maskeri et al. 2008), mining author-topic models (Linstead et al. 2007b), software traceability (Asuncion et al. 2010), software categorization (Tian et al. 2009), bug localization (Lukins et al. 2008) etc. In an earlier work we used LDA topic modeling to mine topics from large corpus of source code, and showed that topics that emerge often resemble widely known aspects or concerns in source code (Baldi et al. 2008). In that work we did an in-depth comparison of LDA with other well-known techniques for feature location, and found that LDA based technique can indeed find all the features that other tools find. Second, LDA is a more recent technique and has been shown to be superior to other alternatives such

as Latent Semantic Indexing (LSI) (Dumais 2004) in many information retrieval and machine learning applications (Blei et al. 2003). In addition to these two reasons we also found that LDA has been successfully applied to compare similarity of very short texts (Quan et al. 2009). Our corpus being a query log, contained many documents with very few words. Therefore, applicability of LDA to measure similarity in short documents (Quan et al. 2009) gave us some confidence that LDA would be an appropriate method for mining topics from the log.

In terms of functionality, both LDA and LSI seem to be equally relevant for various classification tasks in software. LSI is equally popular in software engineering research, in areas such as program comprehension (Maletic and Marcus 2001), automatic feature location (Marcus et al. 2004; Poshyvanyk and Marcus 2007; Poshyvanyk et al. 2007), software clustering and categorization (Maletic and Marcus 2000; Kuhn et al. 2007; Kawaguchi et al. 2006), clone detection (Marcus and Maletic 2001), recommender systems (McCarey et al. 2006; Ye and Fischer 2002) etc. Kuhn et al. report that LSI performs bit worse for analyzing software because of the scarcity of words in source code (Kuhn et al. 2007). Furthermore, LSI does not provide a solid probabilistic model to associate documents with topics and topics with words. Labeling clusters of similar documents (or topics) in LSI is often an additional step and requires a use of good similarity model. This could introduce an additional fuzziness in the accuracy of the method. Due to these issues with LSI, we believe that our choice of LDA for mining topics from the usage log is valid.

However, it should be noted that both LDA and LSI require proper tuning of parameters to get meaningful results. It is still an open question whether one approach is indeed superior to another for various classification tasks in software since there has not been a sound empirical comparison of these techniques for working with software artifacts.

Criteria for Selecting Users Searching in Java There were two options to select the user searching in Java. The first option was to simply select all the users who specified Java as a language in their query. Second was to select users who mostly searched for Java as defined by the criteria we presented in Section 4. When we looked at this list of users for all the languages that they searched in, we noticed that the users rarely searched in other languages than Java. Since this criteria covered maximum number of users who mostly searched for Java, we chose this option. Some noise could have been introduced if this criteria included non-Java queries in our corpus for topic modeling. In retrospect, we think that the choice of just selecting users searching in Java could have equally good too.

Choice of a Model for a Document For LDA, we modeled all collections of user queries as a single document. Other choices were to model a single query or a single session as a document. Since we observed that most of the queries had very few terms, and most of the sessions contained very few search activities (Section 3), these choices to model a document would have resulted in a document with mostly one word. That would have made our document collection even sparser. Therefore, we decided to model a document as a collection of all queries given by a user.

Data Processing for LDA A common practice in topic modeling is to carefully select the vocabulary by omitting some words that are considered noise (stop words).

We do not use any such stop words in our study. This was primarily because we wanted to see the results as it came from mining the queries as much in their original form as possible. Employing a list of stop words such as those commonly found in name definitions, query operators, Java keywords, and those that made up the natural language queries might result in a different set of topics. We plan to investigate this in future. Another limitation with the vocabulary was that the Koders log was case insensitive. A large number of the queries seemed to be names of entities in source code that users were trying to locate. It is possible that the users had included coding conventions such as camel case in their original query. Koders's log seems to have lost this information since all of the query terms in the log were in lower case. Had we been able to do a finer extraction based on code specific heuristic such as camel case splitting, we might have seen a different set of topics, perhaps easier to interpret linguistically.

Assumptions with LDA We have made several simplifying assumptions as we proceeded with our analysis. Here we discuss some limitations based on these assumptions.

Our choice of selecting the activities from users looking in Java to make interpreting the topics easy might pose some limitation in carrying over the conclusions to the entire users of the search engine. At this point, we cannot say for sure if there could be any perceivable difference in the results if we repeat our study with users in a different language or do the study on all the users combined.

Another assumption we made was with the decision of selecting only the best matching topic while matching queries with topics. We have tried getting the results with weighted counts, unlike the absolute count that considers only the best matching topic to query. The results did not differ much in that case. We also chose a simpler similarity metric to match queries with topics. Although there exists better models to do similarity matches of queries to topics, for example similarity metric used in relevance feedback based on the language model for information retrieval (Grossman et al. 2004; Baeza-Yates and Ribeiro-Neto 1999), we proceeded with our metric since it was computationally much cheaper to execute. We did some comparison on how the matching differed when we employed a bit more complicated scheme, but did not see any observable difference. Both of these are topics further to be explored.

The analysis of the results from LDA topic modeling requires subjective interpretation. For most of the topics, the interpretations were straightforward, few were difficult to interpret (eg; topics 'U1','U2'). There were topics that had relatively fewer descriptive words among the most probable words assigned to them (eg; topic 'JAVA'). We have included the full list of 50 topics in Table 19 to make our interpretations about the topics clear.

The value we used for α is smaller than the commonly used value of $50/K$ (where, K = number of topics), and the value we used for β is higher than what is chosen normally. In order to select an appropriate value for these hyperparamters we ran topic modeling with different values for these hyperparameters. We started with the standard values used in practice $\alpha = 1$ (for $K = 50$), and $\beta = 0.01$. We obtained topic modeling results for next two sets of values for these parameters with ($\alpha = 2$, $\beta = 0.005$), and ($\alpha = 0.5$, $\beta = 0.1$). Choice of ($\alpha = 0.5$, $\beta = 0.1$) seemed to result in

the best combination of words that described most of the topics in the set of 50. For example, when setting ($\alpha = 1$, $\beta = 0.01$) we noticed topics (showing 2 of 50) described by words as follows:

(class, database, public, object, main, ..)

(axis, mysql, stringutils, view, jasper, ..)

With $\alpha = 0.5$ and $\beta = 0.1$, we observed the following topics described by different combinations of some of the words found above:

(sql, jdbc, connection, database, mysql, ..)

(class, extends, public, key, implements, ..)

The second set of topics seem to be more meaningful combination of words describing topics related to "database" and "java keywords". Compared to the second set, the first set of topics mixes words among topics resulting in topics that seem less meaningful. Overall we noticed many such better word associations for vales ($\alpha = 0.5$, $\beta = 0.1$). Therefore, we decided to pick the topics generated with these values for the hyperparameters.

In LDA, smaller values for α yields finer distribution of documents into topics as it controls the division of documents into topics, and increasing β has an effect of producing coarser topics (Maskeri et al. 2008; Griffiths and Steyvers 2004). Two possible reasons for better results with our values for these hyperparameters might be: due to the nature of our corpus (it constitutes short-text documents made up of queries instead of natural language text), and mining smaller (50) number of topics.

Topic Categories The classification of topics into categories was done manually by the first author. After observing the top 20 terms for each of the topics and looking at some random sample of queries from the log that belonged to each topic, it seemed that all the topics could be classified into one of the six categories. The assignment of topics into the six categories was later verified by the second author to be a plausible classification. There is a chance that others could have come up with a different assignment of topics, if not an entirely different classification. However, both the authors have significant experience with programming in Java and are familiar with most of the libraries and programming concepts represented by the topics. Thus, it is believed that the assignment of topics to categories are quite representative of search topics in Java (see Table 21 in the Appendix).

Popularity of Topics The popularity ranking of topics indicate that certain frameworks such as Eclipse and JFreeChart have high download activity. This might indicate that there are certain frameworks that are popular. But, it could also be true that Koders is highly used by developers who are working with these frameworks, thus they have high downloads.

Choice of LDA Tools As mentioned earlier we used the Dragon toolkit (Zhou et al. 2007) as our tool for topic modeling. There are other tools available on the Web for LDA analysis; notably, "GibbsLDA++" (Web site for GibsLDA++ 2010),

"LDA-C" (Web site for LDA-C 2010), "Matlab Topic Modeling Toolbox" (Web site for Matlab Topic Modeling Toolbox 2010), "LDA-j" (Web site for LDA-J 2010), an implementation of LDA in the Mahout project (Web site for Apache Mahout 2010) etc. Our choice of using Dragon toolkit was more of a technical convenience. Dragon toolkit is implemented in Java, that made it easy to modify some parts of it to generate various probability distributions we used in our analysis. We believe other tools could have been equally suitable for our purpose since they seem to have similar functionality.

7.3 Form and Relevance

The numbers in Table 18 suggest that lexical forms could indicate the result type in a query. Running a Chi-square test fails to show that the lexical forms are independent of result types and supports this argument. However, some of the counts are too low in Table 18, that makes the statistical significance of this result questionable.

Relevance Judgements Making judgements about relevance was easier for certain forms compared to others. For example, in the case of name queries and queries finding entities we just had to check if the downloaded code contained the names that were used in the queries. This was quite easy compared to term queries and queries finding features where we had to analyze the downloaded code more carefully to see if it really contained an implementation of the desired feature that the query suggested. The relevance judgement was made by the first author by looking at other queries and download that followed in the search session. When it was difficult to make a good judgement based on this information, the session was discarded and another session was sampled from the log. Although, every effort was made to make a sound relevancy judgement, it is possible that some subjective bias could have been introduced. This could effect the results we obtained about effectiveness and efficiency of the different query forms. In the absence of feedback from the users, it is impossible to establish what information need they had while writing a particular query. Therefore to remove any subjective bias that might have been introduced in our analysis, our results can be cross validated with the results from a controlled experiment where it would be possible to define a more objective definition of relevance.

Analysis of Topics in Search-Download Sessions Our goal with analyzing the 150 search-download sessions was to get an overview of the forms of queries that could be considered effective and efficient. We realize there are many other variables related to topics, and activities that can be studied in the sessions. For example, change pattern in topics, and more complicated or accurate measures of effectiveness and efficiency. We plan to explore these topics in our future work.

8 Related Work

Search and information seeking behavior of software developers has been studied quite extensively in the past (Sillito et al. 2006; Ko et al. 2006, 2007; Singer et al. 1997;

Murphy et al. 2006; Sim et al. 1998). However, there have been very few studies on search behavior of developers in Internet-Scale code search engines. The two closest studies are those by Umarji et al. (2008) and Hoffmann et al. (2007). Umarji et al. (2008) identified three types of target sizes of results that users seek in Internet-Scale code search. Our identification of the four query types closely matches their target size descriptions. Hoffman et al. classify nine types of queries from 339 search sessions they retrieved from MSN search (a general purpose search engine). Some of their categories match some of the result types we have identified. For example, 'Implementation' resembles 'Entities', and 'APIs' resembles 'Features' (to some extent). Some of Hoffman et al.'s categories (such as 'Beginner Tutorials') do not match any of our categories. Such categories seem to be beyond the scope of code search engines. Both of these studies (Umarji et al. 2008; Hoffmann et al. 2007) are limited to identifying the motivations and intent behind search and differ from our study in scope and methods employed. In particular, they did not seek understanding of topics code search engines users look at.

With respect to the results from topic modeling presented in this paper, Umarji et al.'s result (Umarji et al. 2008) complements ours as they provide some basic understanding about what developers want to find in code search engines. Our results in this paper provides a deeper understanding about the topics code search engine users look at compared to what has been found in Hoffmann et al. (2007). First, our results come from a larger corpus compared to the one used in Hoffmann et al. (2007) (1,055,105 search activities vs. 339 query sessions). Second, our results come from a code search engine as opposed to a general purpose one (Koders vs. MSN). Four out of nine query categories (APIs, Implementations, Algorithms, Language Syntax and Semantics) identified in Hoffmann et al. (2007) seem to constitute most of the topic categories we have discovered. This indicates users limit their queries to limited kinds while searching in code search engines compared to a general purpose search engine. This highlights the fact that a code search engine cannot provide all kinds of information programmers need. There are other sources of information where users seek answers related to their code; such as, tutorials, troubleshooting and development tools (Hoffmann et al. 2007). Solutions that could integrate these kind of information along with the source code might better serve the various search needs of developers.

A more recent study analyzed the Web search log of the Community Search portal on Adobe's Developer Network Web site (Brandt et al. 2009). The authors in Brandt et al. (2009) analyzed 300 search sessions from the log and computed three properties about search sessions: query type, query refinement method, and types of Web pages visited. We have compared some or our results with the results presented in Brandt et al. (2009) in Section 3. The study in Brandt et al. (2009) also showed that query type predicts types of page visited. This finding is similar to our result that result types seem to be related with particular lexical forms. For example, name queries mostly finding entities, and term queries mostly finding features.

Analyzing general purpose search engine logs is a well established research topic. Numerous techniques have been employed, resulting in wide range of results, that had implications from understanding the search behavior of users to getting insights into improving performance using query expansion by leveraging the information in query logs (Silverstein et al. 1999; Whittle et al. 2007; Zhu and Gruenwald 2005). As mentioned earlier in Section 3, some of our results indicate similar usage behavior

among code and general purpose search engine users. Typical search engine users searching the Web use very few terms per query, have short sessions, and only look at the first result page (Silverstein et al. 1999; Jansen and Spink 2006). All these were found to be similar with users in Koders too. However, we cannot make strong claims about these two types of search engines being similar in any other aspect. The similarity we have seen might just be because of the large numbers of users both these search engines have, and the similarity in the user interface Koders has with general purpose search engines.

We are not aware of a work where topic modeling has been used in analyzing general purpose search engine logs. Just as the various applications of mining usage logs from general search engines might be applicable to code search engine, the method we employed in this paper of using topic modeling could be applicable in mining usage log from general purpose search engines too.

9 Conclusion

Software development is a process of both information creation and information seeking. Developers are constantly seeking information in (and about) the code they write, and the code they use (that others have written). As vast amount of source code becomes available in code repositories, the advent of tools that seem capable of mining information from large number of such repositories holds much promise in addressing the various information needs of developers. Koders, being the first Internet-Scale code search engine that became commercially available, is an ideal candidate that holds such a promise. In our analysis, we have learnt that such a promise is far from being fulfilled entirely. Koders attracts a large number of potential users but seems to satisfy only few of them. The interactions of the users with Koders have accumulated large usage information that holds some valuable information about the nature of large scale code search. In this paper we have demonstrated that a detailed analysis of such usage information can provide important insights that suggest improvements in the current system to make it more usable. We believe that extensions to our work will elucidate these issues concerning the *Whats* and the *Hows* of Internet-Scale code search. Hopefully, in near future, an Internet-Scale code search engine such as Koders would fulfill the promises it holds in addressing the search needs of developers - both effectively and efficiently.

Acknowledgements We thank Darren Rush (former CEO, Koders) for providing us with the Koders usage log without which this work would not have been possible. We thank Black Duck Software (current company that owns Koders) for granting us the permission to make the Koders usage log accessible to the public. We thank Andi Zink (VP of Engineering, Black Duck Software) for his efforts in making the Koders usage log public. We thank the anonymous reviewers of this paper for their valuable suggestions that helped to improve the contents of the paper. The work presented in this paper has been supported in part by the National Science Foundation's grant CCF-0347902.

Appendix: List of Topics Mined from the Usage Log

Table 19 Full list of 50 topics mined from the query logs

Topic code	Description	Words
apache	Terms from Apache projects	org, apache, commons, ant, axis, stringutils, poi, lang, tools, impl, beanutils, catalina
audio	Working with audio and sound	compare, control, encode, audio, decode, uuid, diff, sound, record, barcode, rectangle
BIRT	BIRT project from Eclipse	birt, ibm, matrix, page, report, group, scanner, init, ws, a, width, decimalformat
calSched	Calendar and scheduling	calendar, password, excel, quartz, sorting, codehaus, xfire, none, gregoriancalendar
DataStr	Data strutures	list, object, arraylist, map, vector, hashmap, array, set, add, iterator, hashtable, collection
DateTime	Date and Time	tree, date, time, format, btree, binary, simpledateformat, b, timestamp, avl
db.DAO	Data Access Objects, Object Relational Mapping	table, name, dao, base, getproperty, stringtokenizer, column, persistence, ibatis
db.SJMO	Database	sql, jdbc, connection, database, mysql, oracle, datasource, update, tag, select, pool
DEF	Queries with operators, examples in Koders	cdef, mdef, idef, insert, tree, mcall, compute, account, getuserid, jme, ruby, controller, dir
Docs	Various document formats	text, pdf, dom, document, print, node, element, xpath, lowagie, w3c, jdom, itext
Eclipse	Eclipse project	eclipse, org, ui, swt, internal, core, jface, jdt, resources, widgets, wizard, osgi, gef
files	Working with files	file, read, files, create, write, upload, csv, directory, copy, input, reader, open, save
fqn.j3DSfCon	Mixed FQN fragments from sun, sourceforge, bea	com, net, sun, util, sf, utils, tools, j3d, sourceforge, internal, concurrent, bea, edu
gameMob	Mobile games and GUI	game, j2me, bluetooth, mobile, sms, microedition, midlet, chess, canvas, polish
GUI	Swing and AWT GUI	swing, jtable, applet, awt, window, menu, event, jtree, snmp, frame, button, dialog
hibernate	Hibernate project	hibernate, session, address, property, query, transaction, id, mapping, lookup, expression
http	Http and URLs	http, download, url, proxy, httpclient, protocol, line, httpurlconnection, rss, ping
imaging	Image processing, image file formats	image, color, jpeg, compression, tiff, processing, jai, bufferedimage, fft, paint
JAVA	java.lang.io, using 'java' in query terms	java, lang, io, koders, jav, noclassdeffounderror, doctor, soapclient
jboss	Jboss project	org, jboss, ejb, cache, resource, loader, j2ee, spi, sudoku, jmx, container, weblogic
jfreeC	JFreeChart charting library	html, jfreechart, chart, jfree, dataset, data, wicket, telnet, link, renderer, demo, dynamic
Jkw	Java Keywords	class, extends, public, key, implements, gnu, crypto, cipher, des, rsa, interface, void
JMS	Java Message Service	message, jms, listener, tostring, integer, resourcebundle, mcall, locale, xsd, datatype
JsfJgraph	JSF and Jgraph	javax, graph, faces, component, model, jsf, wsdl, event, portlet, jgraph, myfaces, help

Table 19 (continued)

Topic code	Description	Words
JspServ	JSP and Servlets	servlet, jsp, request, path, javax, httpservletrequest, tomcat, multipart
junit	Junit	test, exception, junit, framework, queue, testcase, font, start, assert, tapestry, antlr
log4j	Logging with log4j	log, logger, log4j, configuration, logging, config, common, logfactory, info, debug, util
lucene	Lucene, Information Retrieval	lucene, main, hash, javascript, index, field, count, function, displaytag, mozilla, analysis
M.GisThr	Mixed topics: Unknown	thread, ldap, geotools, manager, wrapper, cvs, monitor, point, draw, polygon, lock, jndi
M.GwtSecAuth	Mixed topics: possibly login/authentication	google, login, user, gwt, example, authentication, agent, grid, checkbox
M.rng	Random numbers and timing	random, number, calculator, zip, timer, generator, port, edu, serial, clock, packet, cs
M.util	Mixed topics: possibly utilities	util, new, properties, pattern, load, enum, regex, bufferedreader, constants, stringutil
media	Multimedia	type, error, video, media, player, mp3, template, serializable, change, classloader
ML	WEKA, Machine Learning	m, math, huffman, weka, attribute, instance, call, language, linear, fibonacci, x, entry
Netbeans	Netbeans	org, netbeans, editor, api, command, modules, windows, lib, openide, console
Network	Networking, FTP	client, server, ftp, socket, iso, 3166, smtp, chat, tcp, jxta, udp, ftpclient, nio
NL.apps	Natural language queries for projects/apps	system, project, application, management, koders, c, sip, h, projects, library, out, online
NL.HUD	Natural language, "How to" type queries	and, using, from, data, the, get, how, with, program, value, find, for, display, check, not
NL.SC	Natural language queries for "source code"	code, source, for, mail, algorithm, network, email, send, sample, puyo, internet, codes
parsing	Parsers and compilers	parser, parse, base64, compiler, parsing, htmlparser, funambol, xmlparser, utility
searchEng	Searching	search, engine, rmi, browser, method, first, status, remote, reflect, desktop, invoke
secAuth	Security and authentication	security, jar, service, jasper, ssl, auth, engine, jasperreports, export, dori, signature
sort	Sorting, Koders's example terms	sort, md5, card, merge, quicksort, shell, messenger, multi, poker, hello, selection, join
Spring	Spring framework	org, web, springframework, spring, context, factory, support, ajax, view, beans, services
streams	Working with streams	default, inputstream, process, size, content, runtime, length, buffer, response
String	Working with strings	string, convert, 1, byte, int, array, replace, 0, split, converter, 2, character, stringbuffer
Struts	Struts project	struts, filter, action, form, validator, validation, validate, opensymphony, bean
U1	Random words, unknown topic	import, word, bouncycastle, header, ioexception, provider, velocity, utils, location
U2	Unknown, possibly FQN fragments	package, com, gui, plugin, task, all, v2, xstream, basic, mchange, impl, c3p0
XmlWS	XML and web services	xml, shopping, cart, javax, soap, rpc, sax, bind, transform, webservice, schema, stream

Table 20 Sample topics related to networking obtained when number of topics fixed at 500

Network topics	
Sockets	socket, serversocket, programming, accept, sockets, socketserver, socketimpl, closed, socketpool
Multithreaded client/server	server, smtp, client, webserver, multithreaded, smtpclient, multithread, smtpconnection, smtpserver
xmpp	org, jivesoftware, smack, packet, xmppconnection, spark, openfile, jive, wildfire
UDP	udp, char, multicast, broadcast, disable, printf, const, sin, datagram
Telnet	telnet, settings, terminal, ioutil, telnetclient, preference, step, ansi, socketclient
tcp/ip	socket, tcp, ip, external, protocol, tcpclient, keepalive, tcpserver, alive
snmp	snmp, opennms, snmp4j, mib, trap, agent, pdu, oid, jmx
smpp, wireles	segmentation, smpp, azureus, latitude, adaptive, wireless, tile, longitude, connected
packet sniffing	lookup, packet, dns, jpcap, sniffer, env, router, arp, ctx
messaging, lan	messenger, initialize, lan, messaging, db4o, instant, intranet, webmail, self
jabber, dhcp	id, chain, jabber, sequencegenerator, dhcp, isnumber, maze, identifiergenerator, tablegenerator
http, apache	httpclient, commons, apache, methods, jakarta, getmethod, postmethod, httpconnection, executemethod
http	http, post, httpconnection, httpurlconnection, www, request, protocol, httpserver, httpresponse
ftp	client, ftp, ftpclient, ftpserver, serveur, enterprisedt, transfert, ftpexception, ftpconnection
file transfer, sharing	file, transfer, sharing, semap, textfile, handling, filetransfer, lister, fileobject
addresses, mac, ip	address, mac, book, ipaddress, fingerprint, addressbook, automatic, machine, networkinterface
Routing protocols	protocol, routing, dwr, simulator, aglets, aglet, networks, route, uk

Table 21 Topic assignment to categories

Category	Topics under category
Applications	calSched, gameMob, media, searchEng
Programming tasks	audio, DataStr, DateTime, db.DAO, Docs, files, http, M.rng, Network, parsing, secAuth, sort, streams, String, XmlWS
Form centric	DEF, fqn.j3DSfCon, Jkw, NL.apps, NL.HUD, NL.SC
Java/JDK libraries	db.SJMO, GUI, imaging, JAVA, JMS
Frameworks	apache, BIRT, Eclipse, hibernate, jboss, jfreeC, JsfJgraph, JspServ, junit, log4j, lucene, ML, Netbeans, Spring, Struts
Unknown	M.GisThr, M.GwtSecAuth, M.util, U1, U2

References

Andrzejewski D, Mulhern A, Liblit B, Zhu X (2007) Statistical debugging using latent topic models. In: Proceedings of the 18th European conference on machine learning. Springer, Warsaw, Poland, pp 6–17

Asuncion H, Asuncion A, Taylor R (2010) Software traceability with topic modeling. In: 32nd international conference on software engineering

Aula A, Nordhausen K (2006) Modeling successful performance in web searching. J Am Soc Inf Sci Technol 57(12):1678–1693

Baeza-Yates RA, Ribeiro-Neto BA (1999) Modern information retrieval. ACM Press/Addison-Wesley

Bajracharya S, Lopes C (2009) Mining search topics from a code search engine usage log. In: 6th IEEE international working conference on mining software repositories, 2009. MSR '09, pp 111–120. doi:10.1109/MSR.2009.5069489

Bajracharya S, Ngo T, Linstead E, Dou Y, Rigor P, Baldi P, Lopes C (2006) Sourcerer: a search engine for open source code supporting structure-based search. ACM Press, New York, pp 681–682. doi:10.1145/1176617.1176671

Bajracharya S, Ngo T, Linstead E, Rigor P, Dou Y, Baldi P, Lopes C (2007) A study of ranking schemes in internet-scale code search. Tech. Rep. UCI-ISR-07-8, UCI ISR

Bajracharya S, Kuhn A, Ye Y (2009) SUITE 2009: first international workshop on search-driven development—users, infrastructure, tools and evaluation. In: 31st international conference on software engineering–companion volume, 2009. ICSE-Companion 2009, pp 445–446

Bajracharya S, Ossher J, Lopes C (2010a) Leveraging usage similarity for effective retrieval of examples in code repositories. In: 18th international symposium on the foundations of software engineering

Bajracharya S, Ossher J, Lopes C (2010b) Searching API usage examples in code repositories with sourcerer API search. In: Proceedings of 2010 ICSE workshop on search-driven development: users, infrastructure, tools and evaluation. ACM, Cape Town, South Africa, pp 5–8. doi:10.1145/1809175.1809177

Baldi PF, Lopes CV, Linstead EJ, Bajracharya SK (2008) A theory of aspects as latent topics. In: Proceedings of the 23rd ACM SIGPLAN conference on object oriented programming systems languages and applications. ACM, Nashville, TN, USA, pp 543–562. doi:10.1145/1449764.1449807

Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. J Mach Learn Res 3:993–1022

Brandt J, Guo PJ, Lewenstein J, Dontcheva M, Klemmer SR (2009) Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In: Proceedings of the 27th international conference on human factors in computing systems. ACM, Boston, MA, USA, pp 1589–1598. doi:10.1145/1518701.1518944.

Cui H, Wen J, Nie J, Ma W (2003) Query expansion by mining user logs. IEEE Trans Knowl Data Eng 15(4):829–839

Dumais ST (2004) Latent semantic analysis. Annu Rev Inf Sci Technol 38(1):188–230. doi:10.1002/aris.1440380105

Furnas GW, Landauer TK, Gomez LM, Dumais ST (1987) The vocabulary problem in human-system communication. Commun ACM 30:964–971. doi:10.1145/32206.32212

Griffiths TL, Steyvers M (2004) Finding scientific topics. Proc Natl Acad Sci USA 101(Suppl 1):5228–5235. doi:10.1073/pnas.0307752101

Grimes C, Tang D, Russell D, Amitay E, Murray C, Teevan J (2007) Query logs alone are not enough. In: Query log analysis: social and technological challenges. A workshop at the 16th international World Wide Web conference (WWW 2007)

Grossman DA, Frieder O, Frieder O (2004) Information retrieval algorithms and heuristics, 2nd edn. Springer

Henninger S (1997) An evolutionary approach to constructing effective software reuse repositories. ACM Trans Softw Eng Methodol 6(2):111–140. doi:10.1145/248233.248242

Hoffmann R, Fogarty J, Weld DS (2007) Assieme: finding and leveraging implicit references in a web search interface for programmers. In: Proceedings of the 20th annual ACM symposium on User interface software and technology. ACM, Newport, Rhode Island, USA, pp 13–22. doi:10.1145/1294211.1294216

Holmes R, Murphy GC (2005) Using structural context to recommend source code examples. ACM Press, New York, pp 117–125. doi:10.1145/1062455.1062491

Hummel O, Janjic W, Atkinson C (2008) Code conjurer: pulling reusable software out of thin air. IEEE Softw 25(5):45–52

Jansen BJ, Spink A (2006) How are we searching the world wide web? A comparison of nine search engine transaction logs. Inf Process Manag 42(1):248–263. doi:10.1016/j.ipm.2004.10.007

Joachims T (2002) Optimizing search engines using clickthrough data. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, Edmonton, Alberta, Canada, pp 133–142. doi:10.1145/775047.775067

Joachims T, Granka L, Pan B, Hembrooke H, Radlinski F, Gay G (2007) Evaluating the accuracy of implicit feedback from clicks and query reformulations in Web search. ACM Trans Inf Syst 25(2):7. doi:10.1145/1229179.1229181

Kawaguchi S, Garg PK, Matsushita M, Inoue K (2006) MUDABlue: an automatic categorization system for open source repositories. J Syst Softw 79(7):939–953

Ko AJ, Coblenz MJ, Aung HH (2006) An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. IEEE Trans Softw Eng 32(12): 971–987

Ko AJ, DeLine R, Venolia G (2007) Information needs in collocated software development teams. In: Proceedings of the 29th international conference on software engineering. IEEE Computer Society, pp 344–353

Koenemann J, Belkin NJ (1996) A case for interaction: a study of interactive information retrieval behavior and effectiveness. In: Proceedings of the SIGCHI conference on human factors in computing systems: common ground. ACM, Vancouver, British Columbia, Canada, pp 205–212. doi:10.1145/238386.238487

Kuhn A, Ducasse S, Gírba T (2007) Semantic clustering: identifying topics in source code. Inf Softw Technol 49(3):230–243

Lemos O, Bajracharya SK, Ossher J, Masiero PC, Lopes C (2009) Applying test-driven code search to the reuse of auxiliary functionality. In: 24th annual ACM symposium on applied computing (SAC 2009)

Linstead E, Rigor P, Bajracharya S, Lopes C, Baldi P (2007a) Mining concepts from code with probabilistic topic models. In: Proceedings of the twenty-second IEEE/ACM international conference on automated software engineering. ACM, Atlanta, Georgia, USA, pp 461–464. doi:10.1145/1321631.1321709

Linstead E, Rigor P, Bajracharya S, Lopes C, Baldi P (2007b) Mining eclipse developer contributions via author–topic models. In: Proceedings of the fourth international workshop on mining software repositories. IEEE Computer Society, p 30

Linstead E, Bajracharya S, Ngo T, Rigor P, Lopes C, Baldi P (2009) Sourcerer: mining and searching internet-scale software repositories. Data Mining and Knowledge Discovery 18(2):300–336. doi:10.1007/s10618-008-0118-x

Liu H, Lethbridge TC (2001) Intelligent search techniques for large software systems. In: Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative Research. IBM Press, Toronto, Ontario, Canada, p 10

Lopes CV, Bajracharya SK, Ossher J, Baldi PF (2010) UCI source code data sets. University of California, Bren School of Information and Computer Sciences, Irvine. http://www.ics.uci.edu/~lopes/datasets/

Lukins SK, Kraft NA, Etzkorn LH (2008) Source code retrieval for bug localization using latent dirichlet allocation. In: 15th Working Conference on reverse engineering, 2008. WCRE'08, pp 155–164

Maletic JI, Marcus A (2000) Using latent semantic analysis to identify similarities in source code to support program understanding. In: 12th IEEE international conference on tools with artificial intelligence (ICTAI'00), pp 46–53

Maletic JI, Marcus A (2001) Supporting program comprehension using semantic and structural information. In: Proceedings of the 23rd international conference on software engineering. IEEE Computer Society, Toronto, Ontario, Canada, pp 103–112

Mandelin D, Xu L, Bodik R, Kimelman D (2005) Jungloid mining: helping to navigate the api jungle. ACM Press, New York, pp 48–61. doi:10.1145/1065010.1065018

Marcus A, Maletic JI (2001) Identification of high-level concept clones in source code. In: Proceedings of the 16th IEEE international conference on automated software engineering, p 107

Marcus A, Sergeyev A, Rajlich V, Maletic J (2004) An information retrieval approach to concept location in source code. In: Proceedings of the 11th working conference on reverse engineering (WCRE 2004), pp 214–223

Maskeri G, Sarkar S, Heafield K (2008) Mining business topics in source code using latent dirichlet allocation. In: Proceedings of the 1st conference on India software engineering conference. ACM, Hyderabad, India, pp 113–120. doi:10.1145/1342211.1342234

McCarey F, Cinneide MO, Kushmerick N (2006) Recommending library methods: an evaluation of the vector space model (vsm) and latent semantic indexing (lsi). Lect Notes Comput Sci 4039:217

Murphy GC, Kersten M, Findlater L (2006) How are java software developers using the eclipse IDE? IEEE Softw 23(4):76–83

Poshyvanyk D, Marcus A (2007) Combining formal concept analysis with information retrieval for concept location in source code. In: Proceedings of the 15th IEEE international conference on program comprehension. IEEE Computer Society, pp 37–48

Poshyvanyk D, Gueheneuc Y, Marcus A, Antoniol G, Rajlich V (2007) Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. IEEE Trans Softw Eng 33(6):420–432

Quan X, Liu G, Lu Z, Ni X, Wenyin L (2009) Short text similarity based on probabilistic topics. Knowl Inf Syst 1–19. http://www.springerlink.com/content/75r0k316w8600684/

Reiss SP (2009) Semantics-based code search. In: Proceedings of the 2009 IEEE 31st international conference on software engineering. IEEE Computer Society, pp 243–253. doi:10.1109/ICSE.2009.5070525

Sillito J, Murphy GC, Volder KD (2006) Questions programmers ask during software evolution tasks. ACM, Portland, Oregon, USA, pp 23–34. doi:10.1145/1181775.1181779

Silverstein C, Marais H, Henzinger M, Moricz M (1999) Analysis of a very large web search engine query log. SIGIR Forum 33(1):6–12. doi:10.1145/331403.331405

Sim SE, Clarke CLA, Holt RC (1998) Archetypal source code searches: a survey of software developers and maintainers, p 180

Singer J, Lethbridge T, Vinson N, Anquetil N (1997) An examination of software engineering work practices. IBM Press, p 21

Thummalapenta S, Xie T (2007) Parseweb: a programmer assistant for reusing open source code on the web. In: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering. ACM, Atlanta, Georgia, USA, pp 204–213. doi:10.1145/1321631.1321663

Tian K, Revelle M, Poshyvanyk D (2009) Using latent dirichlet allocation for automatic categorization of software. In: 6th IEEE international working conference on mining software repositories

Umarji M, Sim S, Lopes C (2008) Archetypal internet-scale source code searching. In: Open source development, communities and quality. IFIP International Federation for Information Processing, vol 275/2008. Springer, Boston, pp 257–263

Web page for Koders log analysis github repository (2010) http://www.github.com/sushil/koders-loganalysis. Accessed 8 August 2010

Web site for AGID word list (2010) http://wordlist.sourceforge.net/. Accessed 8 August 2010

Web site for Apache Mahout (2010) http://lucene.apache.org/mahout/. Accessed 8 August 2010

Web site for GibsLDA++ (2010) http://gibbslda.sourceforge.net/. Accessed 8 August 2010

Web site for Google Code Search (2010) http://www.google.com/codesearch. Accessed 8 August 2010

Web site for Koders (2010) http://www.koders.com. Accessed 8 August 2010

Web site for Krugle (2010) http://www.krugle.com. Accessed 8 August 2010

Web site for LDA-C (2010) http://www.cs.princeton.edu/ blei/lda-c/index.html. Accessed 8 August 2010

Web site for LDA-J (2010) http://www.arbylon.net/projects/. Accessed 8 August 2010

Web site for Matlab Topic Modeling Toolbox (2010) http://psiexp.ss.uci.edu/research/programs_data/toolbox.htm. Accessed 8 August 2010

Whittle M, Eaglestone B, Ford N, Gillet VJ, Madden A (2007) Data mining of search engine logs. J Am Soc Inf Sci Technol 58:2382–2400

Xu J, Croft WB (1996) Query expansion using local and global document analysis. In: Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval. ACM, Zurich, Switzerland, pp 4–11. doi:10.1145/243199.243202

Ye Y, Fischer G (2002) Supporting reuse by delivering task-relevant and personalized information. In: Proceedings of the 24th international conference on software engineering. ACM, Orlando, Florida, pp 513–523. doi:10.1145/581339.581402

Zazo F, Figuerola CG, Berrocal JLA, Rodriguez E (2005) Reformulation of queries using similarity thesauri. Inf Process Manag 41(5):1163–1173

Zhou X, Zhang X, Hu X (2007) Dragon toolkit: incorporating auto-learned semantic knowledge into large-scale text retrieval and mining. In: 19th IEEE international conference on tools with artificial intelligence, ICTAI 2007, vol 2, pp 197–201. doi:10.1109/ICTAI.2007.117

Zhu Y, Gruenwald L (2005) Query expansion using web access log files. In: Database and expert systems applications. Lecture notes in computer science, vol 3588/2005. Springer, Berlin, pp 686–695

Sushil Krishna Bajracharya obtained his Ph.D. in Information and Computer Science from the Department of Informatics in the School of Information and Computer Sciences at University of California Irvine. His Ph.D. research focused on Internet-Scale Code Retrieval that contributed several novel techniques to retrieve relevant source code from large code repositories on the Internet. His current research lies at the intersection of Software Engineering and Information Retrieval. More information about his research and professional activities are available on his web site http://sushil.me.

Cristina Videira Lopes is an Associate Professor in the School of Information and Computer Sciences at the University of California, Irvine. Prior to being in Academia, she worked at the Xerox Palo Alto Research Center (1995–2001). She is most known as co-inventor of AOP (Aspect Oriented Programming). She also made contributions to Ubiquitous Computing research with her work on lightweight software acoustic modems that encode small amounts of data in socially-acceptable sounds. Recently, she has been conducting research at the intersection of software engineering and information retrieval. Besides research work, she is also a contributor to several open source software projects. She is the recipient of several NSF grants, including a prestigious CAREER Award. Dr. Lopes has a PhD from Northeastern University, and MS and BS degrees from Instituto Superior Tecnico in Portugal.