Query Reformulation by Leveraging Crowd Wisdom for Scenario-based Software Search

Zhixing Li, Tao Wang, Yang Zhang, Yun Zhan, Gang Yin
National Laboratory for Parallel and Distributed Processing
National University of Defense Technology
starleelzx@163.com, {taowang2005, yangzhang15}@nudt.edu.cn, cloud_zhan@163.com, yingang@nudt.edu.cn

ABSTRACT

The Internet-scale open source software (OSS) production in various communities are generating abundant reusable resources for software developers. However, how to retrieve and reuse the desired and mature software from huge amounts of candidates is a great challenge: there are usually big gaps between the user application contexts (that often used as queries) and the OSS key words (that often used to match the queries). In this paper, we define the scenario-based query problem for OSS retrieval, and then we propose a novel approach to reformulate the raw query by leveraging the crowd wisdom from millions of developers to improve the retrieval results. We build a software-specific domain lexical database based on the knowledge in open source communities, by which we can expand and optimize the input queries. The experiment results show that, our approach can reformulate the initial query effectively and outperforms other existing search engines significantly at finding mature software.

CCS Concepts

•Software and its engineering → Software libraries and repositories; •Information systems → Query reformulation;

Keywords

software retrieval; crowd wisdom; query reformulation.

1. INTRODUCTION

Software reuse plays a very important role in improving software development quality and efficiency. With the quick development of open source movement, huge amounts of open source software are published over the internet [25]. For example, there are more than 460 thousand projects in SourceForge, and more than 30 million repositories in GitHub, and the number of projects in these communities are continuous growing dramatically every day. This on the one hand provides abundant reusable resources [8, 11], and on the other hand introduces great challenge for locating desired ones among so many candidate projects.

To help developers perform such tasks, many projects hosting sites, like SourceForge (sourceforge.net), and GitHub (github.com), have provided service for open source software search and users can launch a query on the base of software datasets they have indexed. General search engines, like Google, Bing are alternative choices because of their powerful ability for query process.

But, both of them are not fit for the scenarios that users are aware of only functionality requirement or application context,especially for the fresher who are lack of development experience and programming skills or the experienced developers who are stepping into a new domain. For example, they may search "android database" when actually meant to persist data for Android app, or "python orm" when programming with python and turn to some ORM engines to replace SQL statements. We call this kind of query as Scenario-based Query. Scenario-based queries are usually short and widely used. Project hosting sites usually match queries with the text contained in software metadata such as title, description, etc. But this strategy can't match user's intent perfectly [9]. Results returned by a general search engine cover a wide range of resource and usually need additional clicks and time to filter worthless information [5].

In order to solve the problem of Scenario-based Query, we introduce a novel method to take advantage of crowd wisdom and reformulate the initial query. The approaches to reformulate a query fall into two types: global methods and local methods [9, 15]. Global methods work fully automatically to find the query new terms that are related to its terms and independent of the results returned from it. Local methods make use of the documents that initially appear to match the query and usually rely on user's feedback or pseudo feedback to mark the top documents as relevant or not relevant. The marked documents are then used to adjust the initial query. However, relevance feedback has been little used in web search and most users tend to perform their search without no more interactions [15, 22].

In this paper, we implement the global approach by using domain knowledge that is obtained from a lexical database of software development which we constructed with the crowd wisdom from millions of developers on StackOverflow (stackoverflow.com). We firstly crawl all the tags created by users in a collaborative process in StackOverflow and then build the domain knowledge with those tags in which way attributes of tag, like count and co-occurrence, play an im-

Internetware '16, September 18 2016, Beijing, China
© 2016 ACM. ISBN 978-1-4503-4829-4/16/09...$15.00
DOI: http://dx.doi.org/10.1145/2993717.2993723

portant role. Given a query, after standard preprocess, our method execute synonymy substitution on each term in the initial query, for example "db" will be transformed into "database". Next, the query would be expanded using the related terms obtained from the lexical database. Finally, expanded queries are refined by a ranking model to search from project dataset. What's more, we conducted an empirical evaluation using 14 search scenarios with 35 voluntary developers. We combine measures Precision at k items and Mean Average Precision and use MAP@10 to measure the relevance performance of our method and other search service [15]. We also conduct a user study to assess the usability to help users find mature software.

In summary, our main contributions in this paper include:

1) We build a software-specific lexical database by leveraging crowd wisdom and effectively analyze the domain knowledge in StackOverflow.

2) We reformulate queries with software-specific lexical database to get user's real query intension and performs well in scenario-based queries.

3) Lots of experimental results illustrate that our method can benefit software development by helping users find mature software more efficiently.

The rest of paper is organized like this: Section 2 reviews briefly related work and Section 3 explains related concepts. Section 4 describes in detail our method through a prototype design. Section 5 presents our empirical experiment and evaluation on our method. Section 6 explains some threats to validity of our method. Finally section 7 conclude this paper with future work.

2. RELATED WORK

After reviewing a great deal of literature, we found that most studies in the area of open source software search focus on code search [12, 19, 20] while few researchers study on search of software project entities. Tegawende F. Bissyande proposed an integrated search engine Orion [3] which focuses on searching for project entities and provides a uniform interface with a declarative query language. They draw a conclusion that their system can help user find relevant projects faster and more accurately than traditional search engine. But they restricted the search language which results in additional burdens on users when they express their search intention and the usability goes down. Linstead, et al. [13] developed Sourcerer which takes advantage not only of the textual aspect of software, but also of its structural aspects, as well as any relevant metadata. This system processes query on the code level and is not fit for higher level search, but it inspires us to see software projects form different perspectives

Studies [15, 17] listed several techniques to reformulate a query which can be classified into two types: global methods and local methods. Global methods are independent of the results returned from it. Query expansion via a thesaurus is a widely used global method in which the thesaurus can be generated automatically or manually. In this paper, we use a thesaurus generated automatically. Local methods refine a query according to the documents that are retrieved in the first round to match the query. Relevance feedback has been shown to be effective local method to improve relevance of results, but just as Manning stated very few users used the relevance feedback option on the web [15]. Pseudo relevance feedback is another local method which treats the top k ranked items in the result list as relevant and do following work like relevance feedback does. The problem is that it may result in query drift sometimes. So reformulating queries via an atomically generated thesaurus is more practical for software search.

Automatic query reformulation has been a widely used way to overcome inaccuracy of information retrieval systems (AQE). Carpineto, et al. [4] presents a unified view of a large number of approaches to AQE that leverage various data sources and employ very different principles and techniques. Gao, et al. [7] presents search logs as a labeled directed graph and expands queries with path-constrained random walk in which the probability of determining an expansion term for a term is computed by a learned combination of constrained random walks on the graph. Lu, et al. [14] identifies the part-of-speech of each item in the initial query firstly and then finds the synonyms of each item from WordNet [16]. Like Lu, et al., we also use corpus to expand initial query, but the difference is that the corpus we use is software domain specific that we build on open source software consumption communities inspired by Yin, et al. [25] who stated that data from software consumption communities is very important for the evaluation of open source software.

There are many available approaches to rank software. OpenBRR [23] makes use of source code, document and other data in software development process to do this job. Their method only consider the software itself and ignore the practical application. SourceForge and OpenHub take advantage of the popularity of a software to rank it. But the limitation of their methods is that their results sometimes have deviation from the actual situation because all user feedbacks they adopt come from their own platform. Fan, et al. [6] and Zhang, et al. [28] went further and use user feedbacks coming from consumption communities to assess and rank software. We share the same view of them and think it is more reasonable to make the best of crowd wisdom.

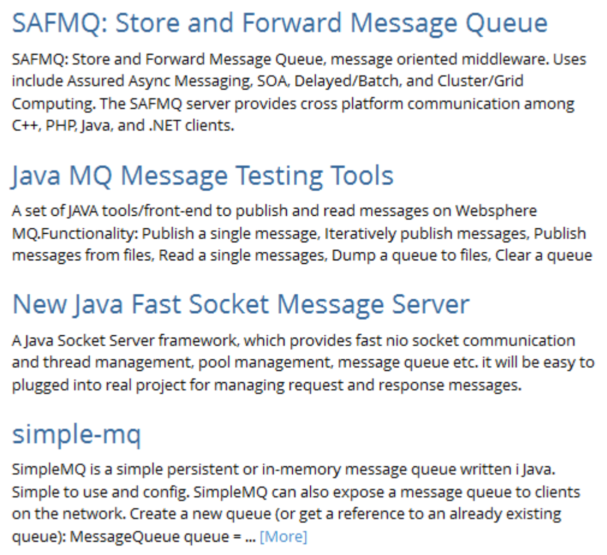
3. RELATED CONCEPTS

In this section, we briefly introduce the concept of scenario-based query and mature software.

3.1 Scenario-based Query

Scenario-based query is a type of queries that are usually short and generated by users to describe what application scenario a software project is about to be applied to or what functional feature a software project should provide. There are some concrete instances to explain scenario-based queries.

1) **A developer searches frameworks implementing specific functionality**. Suppose that a developer is required to build a distributed web crawler with Java language. To achieve this, he tends to use a message queue to dispatch the URLs of web page to be crawled, then the query "java message queue" will be searched. Figure 1 is the top result of what he got from OpenHub. From the figure, we find that the most common Java message queue, RabbitMq and ActiveMQ are not shown in the results. We can see that **current software project hosting platforms hardly return any popular and common software project**



SAFMQ: Store and Forward Message Queue

SAFMQ: Store and Forward Message Queue, message oriented middleware. Uses include Assured Async Messaging, SOA, Delayed/Batch, and Cluster/Grid Computing. The SAFMQ server provides cross platform communication among C++, PHP, Java, and .NET clients.

Java MQ Message Testing Tools

A set of JAVA tools/front-end to publish and read messages on Websphere MQ.Functionality: Publish a single message, Iteratively publish messages, Publish messages from files, Read a single messages, Dump a queue to files, Clear a queue

New Java Fast Socket Message Server

A Java Socket Server framework, which provides fast nio socket communication and thread management, pool management, message queue etc. it will be easy to plugged into real project for managing request and response messages.

simple-mq

SimpleMQ is a simple persistent or in-memory message queue written i Java. Simple to use and config. SimpleMQ can also expose a message queue to clients on the network. Create a new queue (or get a reference to an already existing queue): MessageQueue queue = ... [More]

Figure 1: Top-4 search result for "java message queue" returned by OpenHub

relevant to the query.

2) **A student looks for development tools**. A student who is only familiar with C++ is asked to write a Python program for some course requirement. Although the student does have programming experience, he is a fresher in Python. The first thing he wants to do could be to find a good IDE for Python. So he will fire up browser and search "Python ide" in Bing or Google and Figure 2 is Top-4 of the retrieval result returned by Bing. In the search result, there are some items linking to the home page of relevant software, but they are mixed with links that are not relevant to any target software or even though relevant to some but costing more time to figure out. We find that **regular search engines return search results containing scrambled information**.

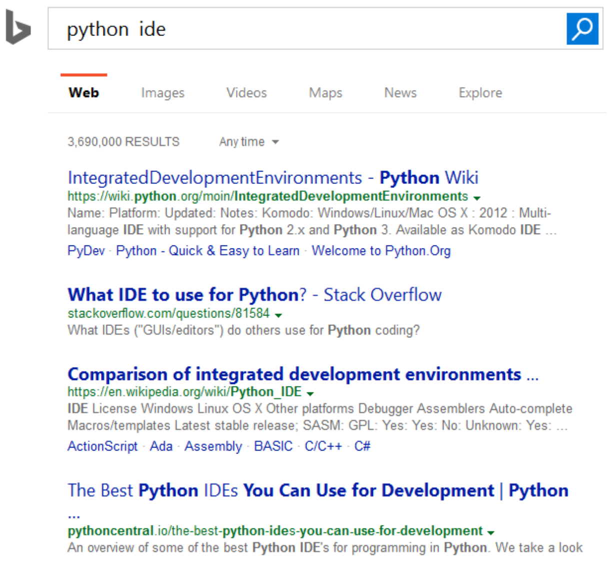
Based on the previous analysis, we conclude the challenges of scenario-based query as follows:

1) **Some terms in scenario-based queries are too general so that it will match so much irrelevant information**. In Figure 2, the retrieval result of "python ide" shows that several results containing term "python" or "ide" is not about an IDE for python in fact.

2) **Application information or functionality description of some software projects maybe lost in its metadata**. For example, Figure 3 shows the description of RabbitMQ in OpenHub from which we cannot find any word about "message queue" that RabbitMQ usually acts as or about "Java" which belongs to client programing languages that RabbitMQ supports for. So, it performs badly if a system only matches the words typed by user with the text contained in software metadata.

3.2 Mature software

With the rapid development of open source software, massive resources provide rich choices for developers, but it also brings great challenge in searching target software for reuse



python ide

3,690,000 RESULTS Any time

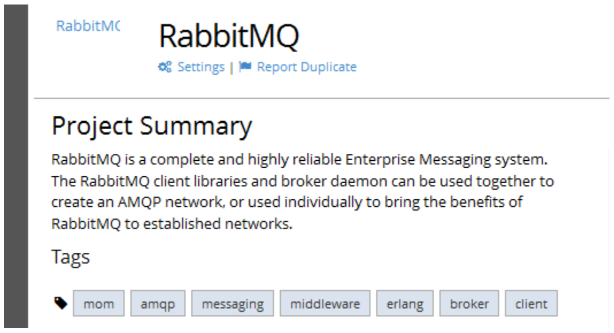
IntegratedDevelopmentEnvironments - **Python** Wiki
https://wiki.**python**.org/moin/**IntegratedDevelopmentEnvironments**
Name: Platform: Updated: Notes: Komodo: Windows/Linux/Mac OS X : 2012 : Multi-language **IDE** with support for **Python** 2.x and **Python** 3. Available as Komodo **IDE** ...
PyDev · Python - Quick & Easy to Learn · Welcome to Python.Org

What IDE to use for Python? - Stack Overflow
stackoverflow.com/questions/81584
What IDEs ("GUIs/editors") do others use for **Python** coding?

Comparison of integrated development environments ...
https://en.wikipedia.org/wiki/**Python_IDE**
IDE License Windows Linux OS X Other platforms Debugger Assemblers Auto-complete Macros/templates Latest stable release; SASM: GPL: Yes: Yes: No: Unknown: Yes: ...
ActionScript · Ada · Assembly · BASIC · C/C++ · C#

The Best **Python** IDEs **You Can Use for Development** | **Python** ...
pythoncentral.io/the-best-**python-ides-you-can-use-for-development**
An overview of some of the best **Python IDE**'s for programming in **Python**. We take a look

Figure 2: Top-4 result for "python ide" returned by Bing



RabbitMC

RabbitMQ

Project Summary

RabbitMQ is a complete and highly reliable Enterprise Messaging system. The RabbitMQ client libraries and broker daemon can be used together to create an AMQP network, or used individually to bring the benefits of RabbitMQ to established networks.

Tags

mom · amqp · messaging · middleware · erlang · broker · client

Figure 3: Description of RabbitMQ in OpenHub

from such large amounts of resources. There are many ways to evaluate a software, such as CapGemini maturity model [24], OpenBRR model [23] and SQO-OSS model [21], which make use of source code, document and other data in software development process. But we think that the points of Fan, et al. [6] and Zhang, et al. [28] are more reasonable. They take advantage of user feedbacks to assess software. They concluded that feedbacks of a software project in consumption communities provide a more effective evaluation on it. We share the same view of them, in our approach, we define the mature software as projects that remain in current and widespread use and receive extensive discussion.

4. CROWD-WISDOM BASED QUERY REFORMULATION FOR SOFTWARE SEARCH

In order to solve the two challenges as described in section 3.1. We propose a Crowd-wisdom based software search method. The overall process of our method is illustrated in Figure 4.

First of all, we build software-specific lexical database, and collect software metadata from Internet. And then, given an initial query, there are three main steps to be done: query

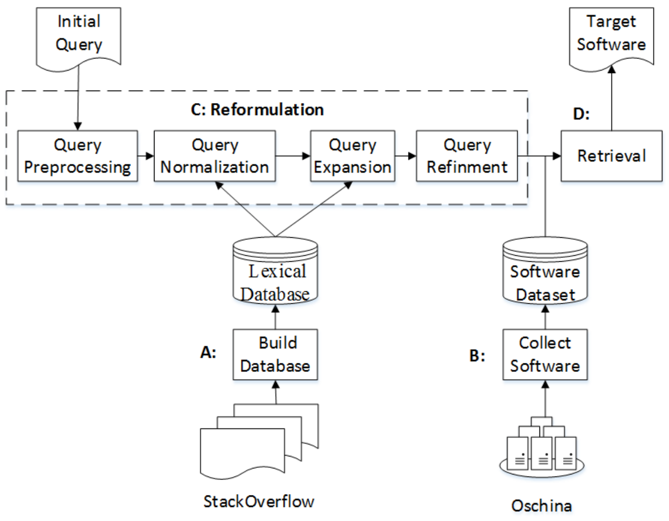


Figure 4: The overall process of our method

preprocessing and normalization, query expansion and refinement and target software retrieve. In the end, a list of target software that best meet developer's need will be produced. In the following subsections, we will explain each stage of our method in detail.

4.1 Software domain lexical database building

We construct synonyms and relatedness lexical database based on data from StackOverflow. We study on StackOverflow dataset because it is a popular Q&A community focused on computer technology, which is used by more than a million developers to ask and answer questions related to computer programming [2]. It contains a great number of posts and associated tags of high quality. When users want to issue a question, they are required to specify at least one tag that is representative of the question or broadly describe the domain which their question belongs to.

According to StackOverflow, a tag is a keyword or label that categorizes questions and using the right tags makes it easier for others to search and answer. These tags are built by the community which is commonly known as a folksonomy. They did a bit of pre-seeding with a few dozen tags that are very obvious and clear, and most present tags we see were created by users in a collaborative process. What is important is that not everyone has the right to create a new tag if he or she doesn't reach the threshold of reputations which increased slowly in StackOverflow. For this reason, tags in StackOverflow are of high quality and strongly related to software engineering, which makes themselves the best choice to build a lexical database of software engineering. First of all, we crawled all the tags in StackOverflow on which the following works are based.

4.1.1 Synonyms thesaurus

Developers are usually preferred to use abbreviation to express some specialized vocabulary, such as “db” for “database” and “js” for “javascript”. However, this convenience results in serious problem in query process and usually leads to vocabulary mismatch [18]. To handle such situations, we convert these abbreviations, like “db” and “js”, to their corresponding master form, that is “database” and “javascript”. So, the synonyms thesaurus is built to help to do this job. We collect synonyms from StackOverflow rather than Wordnet [16] since StackOverflow has maintained a community driven tag synonym system. What's more, people who are willing to help improve the tag folksonomy can vote for or suggest new tags. This truly reflects what synonyms developers use in their daily development. For now, we have 2461 synonyms in total.

4.1.2 Relatedness thesaurus

With the API of StackOverflow we can get a one-way relatedness relationship between two tags. For each tag, StackOverflow return sixty related tags that most frequently cooccurred with it. However, this relationship will result in loss of information when used directly. Tag A is in the list of related tags of tag B doesn't means tag B also appears in A's. So, we decide to convert this one-way relationship to a two-way relationship and enrich the dataset. As a result, every tag is related to the tag that is related to it and this greatly enriched the relatedness thesaurus. For now, we have gathered more than 2.9 million relatedness records in total.

4.2 Open source software repository construction

As we were saying, there are lots of open source software, but few of them are of high quality [25, 26, 27, 29]. So we select those included by Oschina. Oschina founded in August 2008 is the largest open source community in China and has more than 2.2 million registered members. It provides multiple channels including news, forum, code sharing, blogs, and translation, etc. to help local developers study and utilize open source technology. For now, it has included more than 40,000 software projects that are relatively renowned and widely used in the word. The way it gather software is also a presentation of crowd wisdom. They look for valuable open source software in some sites at home and abroad. In addition, registered member is able to apply to help it find potential resource and it will examine whether the suggested software is worthy of including. Under this rule Oschina guarantees it can include as much excellent software as possible.

We designed a web crawler to collect the information of software project included by Oschina and it is illustrated in Figure 5. Oschina shows all the included projects in the form of list, so our crawler has two main steps: crawl list pages and crawl detail pages with the links in each list page. The process of the crawler and data flow is ordered by the number. First of all, model List_Page_URL_Generator use prefix of list page URL and index number to generate all the URLs of list page and put them into a queue. Then, Downloader fetches URLs from the queue in turn and downloads the html of corresponding list page. Next, URL_Extractor extracts URLs of detail page from the downloaded html files of list page according to extracting rules which are constructed manually. The extracted URLs of detail page are also put into a queue waiting to be fetched by Downloader and the downloaded html of detail page will be stored to database. Finally, lots of attributes of OSS, such as name, description, language and license, can be extracted from the final database. At present, we use the name of OSS for precise

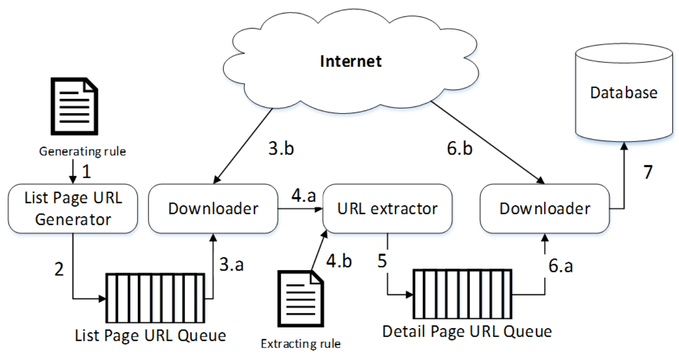


Figure 5: The process of our crawler for Oschina

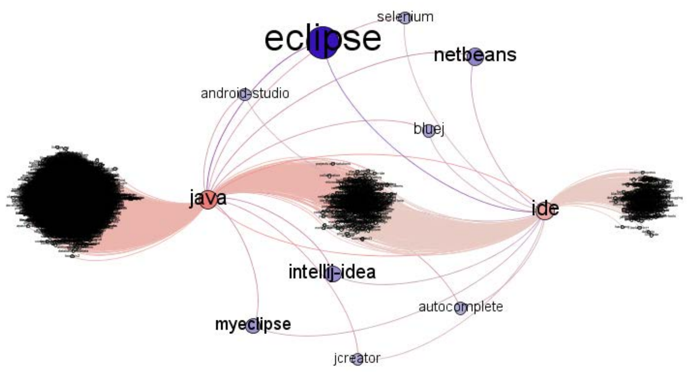


Figure 6: The visualization of reformulation for "java ide"

retrieval and treat it as the identifier for that software.

The above subsections are about offline work. In following steps, we pay attention to what is about to happen for a given query.

4.3 Query reformulation

In this step, we apply some typical text preprocessing on initial query. Stop words that appear frequently but have little meaning, such as "for", "in", "to", "on", etc. will be removed firstly. Then we will convert all the words to lowercase and primary form, i.e. "JavaScript" to "javascript" and "databases" to "database". Finally, in order to deal with user's writing style, we combine neighbor items in initial query to try more possibility for matching the lexical database, for example "java message queue" can also be expressed as "java-message queue", "java message-queue" and "java-message-queue".

After preprocessing, we will normalize items which are found in database to corresponding master form. The synonym thesaurus is used to achieve this goal, and every synonym is replaced by its master form if it does have such one. Queries can be donated as $q = \{t_1, t_2, t_3, ..., t_k\}$, where k indicates query q has k valid terms after the above process, and $t_i(1 \leq i \leq k)$, is just one of them. To expand q, lexical database will return a list of its related tags for t_i together with their co-occur count. After expansion, every term in the query, if being a tag on StackOverflow, will be associated with such a list.

Refinement is the most important step and has a substantial influence on the final quality. In this stage, the term, or terms, used to be retrieved will be generated. We assume that terms appearing in each related tag list of query terms are very likely to be the real intent for the user who typed that query. We call that terms as mutual term. It is obvious that a query may have more than one mutual term. So, the problem is how to rank them, that is how to determine which one is more likely to satisfy the user's need. To achieve this, we proposed a ranking model for mutual terms. Every mutual term is assigned a value under this model and the value is computed by the following formula:

$$Rv(mt) = \prod_{i=1}^{k} \frac{co - occure(mt, term_i)^2}{count(mt) * count(term_i)} \tag{1}$$

In the above formula, mt indicates a mutual term of some query and *Rv(mt)* is its ranking value. Function *count(t)* returns the number of questions tagged by tag t, and $Co-occure(t_1, t_2)$ computes the number of questions tagged by tag t_1 and tag t_2. Each multiplier expresses the degree of relatedness between query term $term_i$ and mutual term mt. Numerator is the square of the number of questions that two tags tagged together. Denominator is the product of the numbers of questions that query item and mutual term tagging respectively. The degree of relatedness will be 1 if two terms always tag the same question, and 0 if they tag totally different questions. The mutual term will be ranked by Rv and the higher Rv is the more front the term will rank.

Figure 6 is the visualization of reformulation for *"java ide"*. There are two outbound nodes and the left one indicates *"java"* and the right one indicates *"ide"*. The nodes in the left side of node *"java"* are terms related to *"java"* but not related to *"ide"*, the nodes in the right side of node *"ide"* are terms related to *"ide"* but not related to *"java"*, and the nodes between node *"java"* and *"ide"* are items related to both *"java"* and *"ide"*. Except for nodes *"java"* and *"ide"*, the size of node corresponds to its Rv and the bigger its size is the more likely a node is the target software project.

4.4 Software Retrieval

In the end, the refinement result will be searched in the software dataset. Software's name will be checked in the retrieve process and if the mutual term is just some software's name, this software will be in the final result list. For performance purpose, we add a flag to each tag indicating whether the tag is the name of any software, so it will be easy to filter the final result list. The final step is to remove fundamental software which falls into two categories: operating systems such as *"Linux"*, *"Android"*, etc., and programing language such as *"java"*, *"php"*, *"python"*, etc. Those software projects are too fundamental to appear in the search result and it will be clearer without them.

5. EVALUATION

In this section, we conduct an evaluation of the prototype implementation of our method.

As discussed in section 4, we conduct our experiment on the data collected from StackOverflow and Oschina. Based on our statistics, there are 44,463 tags used to build lexical database and produce 2,461 synonyms to 1,550 of them and almost 2.9 million items of relatedness relationship. The software projects that we have collected is more than 39,000.

Table 1: Dataset of our experiment

Data source	Num. of items	Time period
StackOverflow	44,463	Jul. 2008 - Jun. 2016
Oschina	39,148	Aug. 2008 - Jun. 2016

Table 2: Search scenarios

ID	Key word		ID	Key word
1	Android ide		8	Android orm
2	Java ide		9	Android db
3	Python ide		10	Android rest
4	Java spider		11	Java message queue
5	Python web spider		12	Java logging
6	Python orm		13	CI
7	Ruby orm		14	python machine learning

We want to assess two aspects of our method: (1)relevance, whether our method helps the user to find relevant software and, in addition, stands out the search service provided by existing project hosting sites (SourForge, OpenHub, GitHub and Oschina) and general search engines (Google, Bing and Baidu) and (2) usability, based on the research [3], we use the usability to measure whether the relevant software returned by our method are mature and more likely to satisfy user's intent.

5.1 Relevance

We collect some typical search scenarios that belong to scenario-based queries from volunteers and use them to test our method and other search services. Table 2 shows all these search scenarios. These scenarios cover several domains of software engineering including development tools, network development, data persistence, data analyses, etc.

Many measures are available to assess the performance of search method. One of them is MAP [15] which is a common measure in ranked retrieval results. For a single query, Average Precision (AP) is the average of the precision value obtained for the of top k documents in retrieval results, and this value is then averaged over queries to get MAP. MAP is computed by the following formula.

$$MAP(Q) = \sum_{j=1}^{|Q|} \frac{1}{m} \sum_{k=1}^{m_j} Precision(R_{jk}) \tag{2}$$

In the formula, R_{jk} is the set of items in retrieve results from the top until meeting the K_{th} relevant item and m indicates the total number of relevant items for query Q_j. For a given query, it is important not just return relevant software but also rank the relevant software in the top of retrieval result. In web search, as researches [15, 1] stated, people tend to be interested in results that are on the first page or, at most, the first three pages. This leads to another measure Precision at k items (also known as P@k) that measures precision at fixed number of retrieved results, such as 10 or 20 items. So, we combine AP and P@k and use measure AP@k which is like AP but at fixed level of retrieval result rather than recall level. In our case, we adopt the advice from previous study [15] and set k to 10 which means "m" in the above formula is 10. That is we will compare the top10 results returned by each approach. As we have mentioned, we will test our method in two ways (1) our method vs. software project hosting sites and (2) our methods vs. general search engines

Table 3: AP@10 of our method and project hosting sites

Query id	Search Approach				
	OM	OS	OS	SF	GH
1	0.55	0.05	0.14	0.00	0.1
2	0.72	0.34	0.36	0.01	0.1
3	0.55	0.41	0.57	0.40	0.4
4	0.37	0.05	0.41	0.53	0.74
5	0.53	0.00	0.59	0.01	0.79
6	0.41	0.76	0.68	0.33	0.42
7	0.30	0.63	0.23	0.00	0.19
8	0.33	0.87	0.34	0.03	0.45
9	0.14	0.00	0.05	0.00	0.10
10	0.35	0.10	0.26	0.01	0.60
11	0.70	0.33	0.05	0.01	0.01
12	0.30	0.47	0.19	0.00	0.23
13	0.77	0.23	0.27	0.02	0.44
14	0.79	0.00	0.19	0.12	0.05

5.1.1 Our method vs. software project hosting site

In the compare between our method(OM) and project hosting sites, GitHub(GH), SourceForge(SF), OpenHub(OP) and Oschina(OS) are chosen which are the most famous of them. Table 3 shows the compare result of AP@10 for each search scenario.

A two-valued judgment system was used and the two values are "not relevant" and "relevant". Every item in the retrieval results has been upon a careful scrutiny to be determined whether it is relevant to the query or not. Software projects which have title or description are relevant to the query but don't have any code at all are viewed as "not relevant". Each row in table 3 shows the AP@10 of every method for a search scenario and the cell in which the number is in bold points out the method that performs best for this scenario. For query 14#, that is "python machine learning", 8 out of 10 retrieval results of our method are relevant and the corresponding AP@10 is 0.79, while other approaches find quite a few relevant software projects resulting in low performance. In the retrieval of 14 search scenarios, our method,Oschina, OpenHub, SourceForge, and Github do the best in six, four, one, none and three of them respectively.

Take the search scenario "java message queue" as an example, table 4 presents the result of each method. From the result we can see that search service provided by project hosting sites usually retrieval by matching the textual query and text contained in project title or description.

This tends to result in **superficial match where textual query is contained, or partly contained, in the metadata of retrieved project but the relevance between them is very low**. What's more, most project hosting sites don't even check whether the returned projects are valid that the code is not empty. All these factors lead to worse performance for these project hosting sites.

5.1.2 Our methods vs. general search engines

To compare our method and general search engines, we select Google, Bing and Baidu to do the contrast analysis.

Table 5 shows the compare result of AP@10 for each search

Table 4: Retrieval result for "java message queue"

Search Method	Retrieval result
Our method	Activemq, rabbitmq, websphere-mq, spring, amqp, apache-kafka, apache-camel, hornet, spring-amqp, zeromq
Oschina	SUN Java System Message Queue, HQueue, Akka, Appserver.io, mJMS, Open Message Queue, Android package android-ActionQueue, Jetlang, NoHttp
OpenHub	Java MQ Message Testing Tools, SAFMQ, New Java Fast Socket Message Server, simple-mq, metis-jms, notify4j, myqueue, ProMVC, Java_Examples, OpenSource .NET & Java Messaging Service
SourceForge	Qmhandle, bacnet for java, mxa, weblogic mq, Java SMPP Client, jlibmodbus, gmail api for java, facebook auto group poster, beecrypt cryptography library, activemqbrowser, java application framework for all
GitHub	mongo-queue-java, rmq, amazon-sqs-java-messaging-lib, softlayer-message-queue-java, javascript-message-queue, storage-queue-java-getting-started, message-queue, message-queue, burrow-java, javamessagequeue

scenario. Results returned by general search engines are a list of web page's URL, and we treat a URL as relevant if it links to the official site or hosting site of some relevant software. What is interesting is that Google and Bing which are the outstanding search engines in the world worked very poor. For these general search engines, it's not their advantage in vertical search for software and they hardly return an official site of a particular software but many posts related to users' query. Maybe user can dig valuable information about related software after clicking and going through these posts, but it required more number of click and took more time. It is a bad idea to force users to filter useful information when the system can do it for them.

Finally, Figure 7 shows the MAP@10 on all these scenarios of each approach. In this measure, our method do the best and Baidu do the worst which hardly return any relevant software projects.

In average, vertical search service provided by project hosting sites do better than general search engine while SourceForge lags behind Google and Bing.

Discussion: In the compare between our method and project hosting sites, our method fall behind others on some search scenarios. For "android orm", the results returned by our method are shown in the table 6. The software project relevant to "android orm" is a kind of software that is ORM engine and can be applied to Android context at the same time. Obviously, "greenDAO", "ORMLite", "ActiveAndroid", "SugarORM" and "dbflow" are target projects, but "SQLite", "MySQL" and the others shouldn't appear in the retrieval result. They have been returned because of their co-occurrence with terms "android" and "orm" On this

Table 5: AP@10 of our method and general search engines

Query id	Search Approach			
	OM	Google	Bing	Baidu
1	0.55	0.20	0.20	0.12
2	0.72	0.34	0.32	0.03
3	0.55	0.07	0.06	0.03
4	0.37	0.06	0.20	0.03
5	0.53	0.10	0.11	0.00
6	0.41	0.19	0.36	0.00
7	0.30	0.10	0.04	0.00
8	0.33	0.13	0.14	0.00
9	0.14	0.00	0.00	0.00
10	0.35	0.19	0.04	0.00
11	0.70	0.18	0.15	0.00
12	0.30	0.16	0.00	0.00
13	0.77	0.00	0.05	0.00
14	0.79	0.07	0.30	0.00

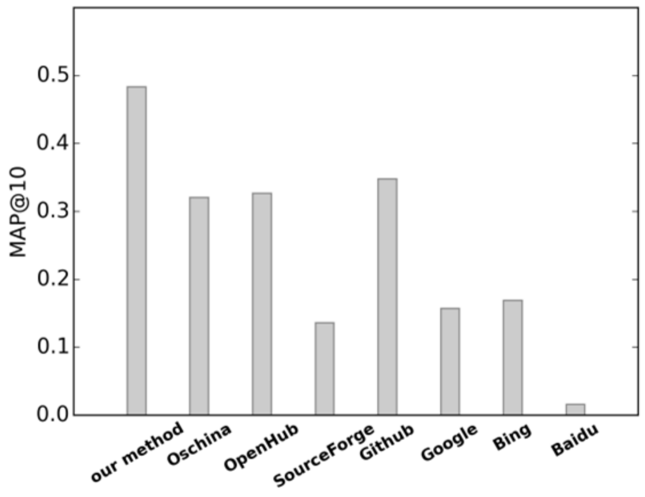


Figure 7: MAP@10 of each approach

point, they really should be in the result list. But their co-occurrence means differently from the relevant software's. The "greentDAO" co-occurs with "orm" because it is a kind of ORM, while "SQLite" co-occurs with "orm" because it is usually used together with an ORM project which results in drift from original search intention and decreases the relevance performance. We treat this problem as mismatch of intention. It is a meaningful work to solve this problem for improving the relevance performance in the future work.

From the tables and figures we can see that **for most search scenarios the AP@10 of our method exceed other approaches' and our MAP@10 is the most outstanding one which means our method is more likely to find users relevant software projects than others**.

5.2 Usability

The above only think of the relevance between query and retrieval result and the software's quality is out of consideration. Software of high usage and attention should be in the front of the retrieval result because people tend to choose software that has been tested and applied widely. We call

Table 6: Top10 retrieval results for "android orm" returned by our method

Rank	Result	Rank	Result
1	SQLite	6	MySQL
2	greenDAO	7	design-patterns
3	ORMLite	8	dbflow
4	ActiveAndroid	9	Realm
5	SugarORM	10	Node.js

Table 7: Likert scale response categories for user evaluation

Scale	Response category
5	Most of them are mature
4	Some of them are mature
3	Not sure
2	Few of them is mature
1	All of them are worthless

these software mature software. The ability to return more mature software rather than no-mature software, which we called worthless software, is an important quality for search service. And this ability is just the usability that we are going to study. In order to do this, we perform a survey on users' attitude towards the search results returned by each approach.

Users are asked to evaluate the relevant items in the search result of each query returned by every approach. The evaluation is a Likert-type scale with more detailed expression for each choice [10]. The respondents need to choose one of five candidate response items to claim how they think about these relevant software in the search result (i.e. our evaluation process is a 5-point Likert scale). Table 7 describes these five candidates.

To do the evaluation 35 individuals with different backgrounds were invited to assess the result. Among them, 12 are master students, 8 are PhD students and 15 engineers with at least 3 years software development experience.

From Figure 8 we can see that, for our method, Q1 (25th percentile) is 4, median, Q2 (75th percentile), and the maximum are all five, and the median is 4.8 which indicates that our method is more likely to find user a mature software with stable performance and does better than other methods. In the last experiment, general search engines lag behind project hosting sites in relevance evaluation. But in usability assessment, general search engines are apparently better than project hosting sites which means that general search engines tend to return few relevant but mature software while project hosting sites return relatively more relevant software but most of which is worthless.

We test and confirm that the performance of all these search approaches are significantly different using the statistical test ($p \ll 0.005, cs = 929.3, df = 6$). This indicates that **it is important to prefer an appropriate and reliable approach to search for software project**.

Discussion: After evaluating each search approach. We can see that our method gets outstanding performance in both relevance evaluation and usability evaluation because our method is crowd-wisdom based which reflects the actual state of software domain and can better satisfy user information need.

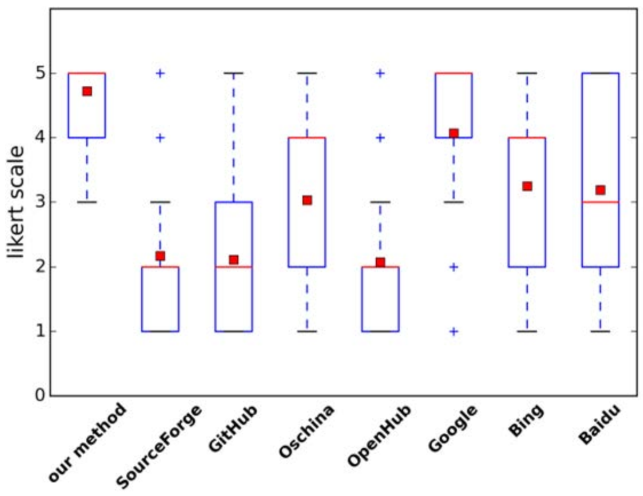


Figure 8: User evaluation on usablity

6. THREATS TO VALIDITY

6.1 Internal validity

Our software domain lexical database is built only on tags in StackOverflow which are of high quality maintained by a strict and excellent mechanism. There still exists many other valuable information that we can extract such as comments, post contents, etc. In the future, we plan to analyze the contents and comments of posts in StackOverflow and rich the lexical database.

6.2 External validity

Interfering software is a kind of software that is too fundamental to be considered as potential candidate for scenario-based queries, such as "Windows", "Ruby" and so on. Although we have filtered out such software from our datasets, the datasets still contain some interfering software. This may impact the overall experience.

7. CONCLUSION & FUTURE WORK

This paper proposed a novel method for scenario-based software search. We build a software-specific domain lexical database based on the knowledge in open source communities and then to reformulate the initial query. The evaluation on an empirical experiment shows that our crowd-wisdom-based method significantly outperforms other search services. Compared with other search services, our method can find user more mature software projects that are more likely to be helpful for user's development.

One of our future work is to apply our method to OSSEAN [25] which is a search engine for open source software and strengthen its ability for various type of queries. Anyway, there are still some problems to be resolved before successful integration. For example, how to identify different type of queries and apply corresponding process method.

8. ACKNOWLEDGEMENT

The research is supported by the National Natural Science Foundation of China (Grant No.61432020, 61472430, 61502512) and National Grand R&D Plan (Grant No. 2016-YFB1000805).

9. REFERENCES

[1] A. Aula, and P. Majaranta. Eye-tracking reveals the personal styles for search result evaluation. In *In Proceedings of INTERACT' 05*, pages 1058–1061, 2005.

[2] V. Bhat, A. Gokhale, R. Jadhav, J. Pudipeddi, and L. Akoglu. Min(e)d your tags: Analysis of question response time in stackoverflow. In *Ieee/acm International Conference on Advances in Social Networks Analysis and Mining*, pages 328–335, 2014.

[3] T. F. Bissyande, F. Thung, D. Lo, L. Jiang, and L. Reveillere. Orion: A software project search engine with integrated diverse software artifacts. In *International Conference on Engineering of Complex Computer Systems*, pages 242–245, 2013.

[4] C. Carpineto and G. Romano. A survey of automatic query expansion in information retrieval. *Acm Computing Surveys*, 44(1):159–170, 2012.

[5] M. Chau and H. Chen. Comparison of three vertical search spiders. *Computer*, 36(5):56–62, 2003.

[6] Q. Fan, H. Wang, G. Yin, and T. Wang. Ranking open source software based on crowd wisdom. In *Software Engineering and Service Science (ICSESS), 2015 6th IEEE International Conference on*, pages 966–972, Sept 2015.

[7] J. Gao, G. Xu, and J. Xu. Query expansion using path-constrained random walks. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 563–572, 2013.

[8] M. D. Ghezzi C, Jazayeri M. *Fundamentals of software engineering*. Prentice Hall PTR, 2002.

[9] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies. Automatic query reformulations for text retrieval in software engineering. In *International Conference on Software Engineering*, pages 842–851, 2013.

[10] S. Jamieson. Likert scales: how to (ab)use them. *Medical Education*, 38(12):1217-1218, 2004.

[11] C. W. Krueger. Software reuse. *Acm Computing Surveys*, 24(2):131–183, 1992.

[12] O. A. L. Lemos, S. K. Bajracharya, J. Ossher, R. S. Morla, P. C. Masiero, P. Baldi, and C. V. Lopes. Codegenie: using test-cases to search and reuse source code. In *Ieee/acm International Conference on Automated Software Engineering*, pages 525–526, 2007.

[13] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi. Sourcerer: mining and searching internet-scale software repositories. *Data Mining & Knowledge Discovery*, 18(2):300–336, 2009.

[14] M. Lu, X. Sun, S. Wang, and D. Lo. Query expansion via wordnet for effective code search. In *IEEE International Conference on Software Analysis, Evolution and Reengineering*, pages 545–549, 2015.

[15] C. D. Manning, P. Raghavan, and H. Tze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[16] G. A. Miller. Wordnet: a lexical database for english. *Communications of the Acm*, 38(11):39–41, 1995.

[17] J. Ooi, X. Ma, H. Qin, and S. C. Liew. A survey of query expansion, query suggestion and query refinement techniques. In *International Conference on Software Engineering and Computer Systems*, 2015.

[18] D. Pal, M. Mitra, and S. Bhattacharya. Exploring query categorisation for query expansion: A study. *Computer Science*, 2015.

[19] S. Paul and A. Prakash. Framework for source code search using program patterns. *IEEE Transactions on Software Engineering*, 20(6):463–475, 1994.

[20] S. P. Reiss. Semantics-based code search. In *International Conference on Software Engineering*, pages 243–253, 2009.

[21] I. Samoladas, G. Gousios, D. Spinellis, and I. Stamelos. *The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation*. Springer US, 2008.

[22] A. Spink, B. J. Jansen, and H. C. Ozmultu. Use of query reformulation and relevance feedback by excite users. *Internet Research Electronic Networking Applications & Policy*, 10(4):pags. 317–328, 2000.

[23] A. Wasserman, M. Pal, and C. Chan. The business readiness rating model: an evaluation framework for open source. In *Proceedings of the EFOSS Workshop, Como, Italy*, 2006.

[24] C. Widdows and F. Duijnhouwer. Open source maturity model. *Cap Gemini Ernst & Young. New York NY*, 2003.

[25] G. Yin, T. Wang, H. Wang, and Q. Fan. Ossean: Mining crowd wisdom in open source communities. In *Service-Oriented System Engineering*, pages 367–371, 2015.

[26] Y. Yu, H. Wang, G. Yin, and T. Wang. Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? *Information & Software Technology*, 74(C):204–218, 2016.

[27] Y. Yu, G. Yin, T. Wang, C. Yang, and H. Wang. Determinants of pull-based development in the context of continuous integration. *Science China Information Sciences*, 59(8):1–14, 2016.

[28] Y. Zhang, G. Yin, T. Wang, Y. Yu, and H. Wang. Evaluating bug severity using crowd-based knowledge: An exploratory study. In *The Seventh Asia-Pacific Symposium on Internetware*, 2015.

[29] H. Zhong, Y. Yang, and J. Keung. Assessing the representativeness of open source projects in empirical software engineering studies. In *Asia-Pacific Software Engineering Conference*, pages 808–817, 2012.