Searching StackOverflow Questions with Multi-Faceted Categorization

Mingwei Liu*
School of Computer Science, Fudan University
Shanghai, China
17212010022@fudan.edu.cn

Xin Peng*
School of Computer Science, Fudan University
Shanghai, China
pengxin@fudan.edu.cn

Qingtao Jiang*
School of Computer Science, Fudan University
Shanghai, China
qtjiang14@fudan.edu.cn

Andrian Marcus
Department of Computer Science, The University of Texas at Dallas
Richardson, Texas, USA
amarcus@utdallas.edu

Junwen Yang*
School of Computer Science, Fudan University
Shanghai, China
15212010024@fudan.edu.cn

Wenyun Zhao*
School of Computer Science, Fudan University
Shanghai, China
wyzhao@fudan.edu.cn

ABSTRACT

StackOverflow provides answers for a huge number of software development questions that are frequently encountered by developers. However, searching relevant questions in StackOverflow is not always easy using the keyword based search engine provided by StackOverflow. A software development question can be characterized by multiple attributes, such as, its concern (*e.g.*,configuration problem, error handling, sample code, *etc.*), programming language, operating system, and involved middleware, framework, library and software technology. We propose a multi-faceted and interactive approach for searching StackOverflow questions (called MFISSO), which leverages these attributes of the questions. Our approach starts with an initial keyword-based query and extracts a multi-faceted categorization from all the candidate questions using natural language processing and data mining. It then allows developers to iteratively refine the search results through an interactive process. We evaluated an implementation of MFISSO in a controlled experiments with 20 computing students, solving ten software development tasks using StackOverflow. The experiment shows that MFISSO can help developers find relevant questions faster and with higher accuracy.

CCS CONCEPTS

• **Information systems** → **Information retrieval**; • **Software and its engineering** → *Software maintenance tools*;

KEYWORDS

Information Retrieval, StackOverflow, Multi-faceted Categorization

*Also with Shanghai Key Laboratory of Data Science, Fudan University

Internetware '18 , September 16, 2018, Beijing, China

ACM ISBN 978-1-4503-6590-1/18/09…$15.00
https://doi.org/10.1145/3275219.3275227

ACM Reference Format:
Mingwei Liu, Xin Peng, Qingtao Jiang, Andrian Marcus, Junwen Yang, and Wenyun Zhao. 2018. Searching StackOverflow Questions with Multi-Faceted Categorization. In *The Tenth Asia-Pacific Symposium on Internetware (Internetware '18), September 16, 2018, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3275219.3275227

1 INTRODUCTION

Question/answer sites, such as, StackOverflow[1] are more and more used by professional software developers and novices alike, for obtaining answers to their software development questions. StackOverflow provides answers for a huge number of software development questions that are frequently encountered by developers. The StackOverflow dump as of January 2016 contains 10,789,362 questions and 17,650,222 answers. Topics of these questions range widely from syntax of programming languages and application programming interfaces (APIs), to version control systems or error fixing.

Before posting a new question on StackOverflow, developers typically search for existing questions (and their answers), which approximate the problem they are facing. The common way to search for existing StackOverflow questions is by using the keyword based search engine provided by StackOverflow, which searches the titles, content, and tags of questions. Finding the best keywords to retrieve the most relevant questions is often a daunting task for developers, especially the novices, when exploring a new problem. They often need to spend a lot of time trying different sets of keywords and identifying the desired questions from a long list of candidate results. For example, if a developer wants to know how to implement the function that creates the Lucene index for text search, then she may search questions with the keywords "Lucene", "create", "index", "search". The query will return more than 1,093 results and the right question is ranked on the 53rd position among the results.

StackOverflow questions can be characterized by several attributes, expressed implicitly or explicitly in their title, content or tags. For example, most questions express a *concern* (*i.e.*,the

[1]http://stackoverflow.com

aim of the question), such as, finding sample code, system configuration method, solution of an encountered error, explanation of phenomenon, *etc.*In addition to its concern, a question can also be categorized based on other attributes, such as, the involved topics, programming language, operating system, database, development framework, *etc.*We argue that categorizing the results of a search based on these attributes will help users find their answers easier.

Based on this conjecture we propose a multi-faceted and interactive approach for searching StackOverflow questions. A facet corresponds to an attribute relevant to the targeted domain and the faceted search allows users to explore the desired information from search results organized according to multiple categories [9, 20]. For the purposes of this investigative research, we define eight facets for StackOverflow questions, *i.e.,Concern*, *Language*, *Operating System*, *Database*, *Development Tool*, *Middleware and Framework*, *Library and Software Technology*, and *Topic*. These facets cover the majority of StackOverflow questions. Our approach allows for the definition of additional facets and the extension of the existing ones. All the facets, except *Topic*, are generated by preprocessing all the existing questions using natural language processing (NLP) techniques (we call them static facets). The *Topic* facet is a kind of dynamic facets that is generated at the time of retrieval using data mining, based on candidate results (*i.e.,*questions).

Our multi-faceted search approach starts with an initial keyword-based query issued by a user when searching for StackOverflow questions. The retrieved questions are categorized based on the defined multi-facets. Leveraging the multi-faceted categorization, the developer iteratively refines the searching results via an interactive process. In each iteration, the developer chooses the desired categories in some facets and the candidate results, while the multi-faceted categorization is updated for the next iteration. For example, for the Lucene index creation problem described above, a developer can easily identify the right question by choosing the "sample code" category from the *concern* dimension, the "Lucene" category from the *library and software technology* dimension, and the "search, index" category from the *topic* dimension. The relevant question is ranked first in this final category, which makes it easier to locate.

We implemented the proposed approach as a Web-based tool called MFISSO (Multi-Faceted and Interactive Searching of StackOverflow) and conducted a controlled experiment to evaluate the approach and get feedback on the usability of the tool. Twenty computing students were asked to find relevant StackOverflow questions for ten development tasks, using MFISSO and the StackOverflow search, respectively. The results of the experiment indicate that MFISSO can help developers find the relevant questions quicker and more accurately than just using the StackOverflow search. In order to complement the results of the experiment, we replicated the searches performed by the subjects and simulated the user actions, and performed an analytical evaluation based on the theoretical usage models of MFISSO and the StackOverflow search. The results are consistent with the controlled experiment and indicate that one can find the relevant questions in fewer steps with MFISSO than with the StackOverflow search.

The rest of this paper is organized as follows. Section II presents related work on program exploration and information seeking, and on the use of StackOverflow. Section III describes the proposed approach, while Section IV describes MFISSO's implementation. Section V presents the controlled experiment and the analytical evaluation. Section VI concludes the paper and outlines future work.

2 RELATED WORK

There have been numerous empirical studies on StackOverflow focusing on different aspects. Barua et al. [2] use latent Dirichlet allocation (LDA) to automatically discover the main topics present in developer discussions of StackOverflow. Their results show that the topics range widely and the most popular topics are web development, mobile applications, Git, and MySQL. Wang et al. [21] also use LDA to find the various kinds of topics asked by developers and identify five topics (miscellaneous, web document, large code snippet, stack trace, and user interface). These works relate to our research in as much as we also identify topics from StackOverflow questions, but with different purpose. Also, as explain in the following section, we chose to use semantic clustering, rather than LDA for topic extraction.

Bazelli et al. [3] explore the personality traits of StackOverflow authors by categorizing them into different categories based on their reputation. They find that the top reputed authors are more sociable and interactive with others. Nasehi et al. [14] conduct a qualitative analysis of well-received StackOverflow answers to identify characteristics of effective code examples. They find that the explanations accompanying examples are as important as the examples themselves. Some researchers propose approaches that can recommend for developers useful information from StackOverflow. Ponzanelli et al. [15] develop an Eclipse plug-in called PROMPTER to recommend StackOverflow discussions that are relevant to the current context in the IDE (Integrated Development Environment). Wang et al. [22] propose an approach that uses social network analysis and topic mining to recommend StackOverflow posts that are relevant to API design-related issues to help API designers to better support developer needs. Guerrouj et al. [8] propose an approach that summarizes the use and purpose of code elements in StackOverflow posts based on the context that surrounds the code elements. These works focus only on code examples in StackOverflow questions/answers or questions that are relevant to specific APIs.

Other researchers proposed approaches for searching code and programming resources on the Web. Rahman et al. [19] proposed an Eclipse IDE-based web search solution that can recommend online resources relevant to encountered programming problems by combining web search engines and StackOverflow search engine. We envision that such approaches can enhance in the future with the multi-facet idea we are proposing and evaluating in this paper. Mcmillan et al. [12] develop a code search system called Portfolio that supports programmers in finding relevant functions and visualizing dependencies of the retrieved functions. Portfolio combines various NLP and indexing techniques with PageRank and spreading activation (SAN) algorithms. Li et al. [10] develop an Eclipse plug-in called amAssist for searching online programming resources that can monitor developerâĂŹs working context and integrate the context in the online search process. amAssist integrates Google custom search engine to search popular programming websites. Different from this work, we provide a multi-faceted categorization and support an interactive process for searching StackOverflow

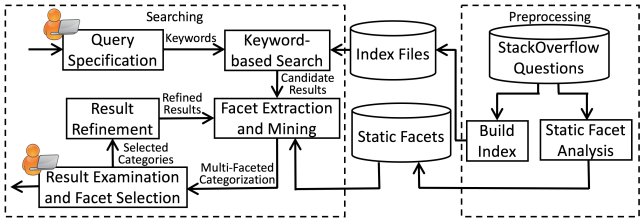

Figure 1: Approach Overview

questions. Common with some of these works is the use of NLP techniques to process the queries and result.

Related to our research is also the rich body of work on using information retrieval to support concept/feature/bug localization [11] [6]. In that body of work, developers issue keyword based queries and retrieve elements of the source code, just as we try to retrieve StackOverflow questions. Most related is the work of Poshyvanyk et al. [16, 18], which generates a labeled concept lattice based on the ranked results returned by IR-based search for developers to further determine the relevance of candidate results by examining their labels. In contrast, we categorize the results based on the multi-facets. The use of multi-faceted categorization is inspired by the work of Wang *et al.* [20], which proposed an interactive feature location approach that extracts and mines multiple facets from candidate program elements by source code analysis. In contrast, in this work we define a set of facets based on the textual content of StackOverflow questions and use NLP to generate the multi-faceted categorization.

3 APPROACH

Our approach includes two phases, *i.e.*,preprocessing and interactive searching. In this section, we first present an overview of the approach and then define the eight facets used in our approach. After that, we present the details of two key steps, *i.e.*,static facet analysis during preprocessing and facet extraction and mining during interactive searching.

3.1 Overview

Figure 1 presents an overview of our approach, showing the main steps of the two phases. The user icons indicate interaction with developers.

The purpose of preprocessing is to build an index for keyword-based searching and extract static facets for multi-faceted categorization. To build the index, our approach takes as input all the StackOverflow questions with their answers and uses a text-based search engine (*i.e.*,Apache Lucene[2] in our current implementation). To extract static facets, our approach analyzes the syntactic structures of the sentences of each question and its answers using the Stanford NLP toolkit. The syntactic structures are then matched against a set of syntactic patterns that we defined based on the analysis of StackOverflow questions. Based on the matching, the category of each question on each static facet is determined and stored.

The interactive searching process starts with an initial keyword-based query specified by the developer. Based on the query, an initial set of search results (*i.e.*,questions) is returned using keyword-based searching based on the index files. Then, our approach automatically extracts a multi-faceted categorization from all the candidate questions based on preprocessed static facets and the *Topic* facet that is dynamically generated at retrieval time. Based on the multi-faceted categorization, our approach sorts and groups the candidate questions. The developer examines the sorted and grouped results following the multi-faceted categorization. She can finish the search process if she finds the desired questions. Otherwise, she can choose to further refine the candidate questions by selecting categories on the facets of interest. The candidate questions are then refined based on the selected categories and a new iteration of searching is started based on the refined results.

Table 1: Facets and Facet Categories

Facet	Facet Categories
Concern	Configuration, Error, Sample Code, Explanations, Design, Algorithm, Implementation
Language	Java, C#, Ruby, CSS, Html, C++, C, JavaScript, PHP, Python, SQL, VB, Matlab
Operating System	Windows, Linux, iOS, OS X, Unix, Android, Ubuntu
Database	MySQL, Oracle, MongoDB, SQLite, PostgreSQL, SQL-Server
Development Tool	Eclipse, XCode, Git, Visual-Studio, Maven, SVN
Middleware and Framework	Tomcat, .NET, Django, AngularJS, Node.js, Spring, LINQ, WCF, Qt, CakePHP, Grails, Laravel, ExtJS, Java EE, MFC, Apache
Library and Software Technology	jQuery, Swing, GWT, Ajax, RegEx, JSON, XML, JSP, Http, URL, Dom, UML, Image, Video, Lucene
Topic	Labels generated from text clustering of candidate questions

3.2 Facets Definition

We manually analyzed 1,100 randomly selected questions from StackOverflow in order to identify the main attributes that describe them. Based on our analysis, we identified eight attributes, which define the facets used by our approach. While these facets describe a majority of StackOverflow questions, there are likely questions in StackOverflow that are not completely described by these facets. To mitigate such issues, our approach allows for extension of facets, as well for addition of new categories in each facet. As such, the approach can be easily updated, as the questions in StackOverflow evolve.

The eight facets and their categories used in this paper are presented in Table 1. Each facet represents a dimension of categorization of StackOverflow questions and on each facet a question can be categorized into one or multiple categories. Each facet except *Topic* has a default category *Other* defined for questions that can not be categorized into any other categories. We describe the facets and categories bellow.

(1) Concern. This facet reflects the problems the question describes, such as, finding a software *Configuration*, fixing a software

[2]http://lucene.apache.org

Error, finding *Sample Code* for reference, considering a *Design* solution of a software, finding and understanding a specific *Algorithm*, *etc.*

(2) Language. This facet reflects the programming languages of interest, with the most popular ones on StackOverflow being: *Java, C#, Ruby, CSS, Html, C++, C, JavaScript, PHP, Python, SQL, VB, Matlab*, *etc.*

(3) Operating System. This facet reflects the operating systems of interest, with the most popular on StrackOverflow being: *Windows, Linux, iOS, OS X, Unix, Android, etc.*

(4) Database. This facet reflects the database of interest, if any, such as: *MySQL, Oracle, MongoDB, SQLite, PostgreSQL, SQL-Server, etc.*

(5) Development Tool. This facet reflects the development tools of interest, if any, such as: *Eclipse, XCode, Git, Visual-Studio, Maven, SVN, etc.*

(6) Middleware and Framework. This facet captures the middlewares and frameworks targeted in the question, such as: *Tomcat, .NET, Django, AngularJS, Node.js, Spring, LINQ, WCF, Qt, CakePHP, Grails, Laravel, ExtJS, Java EE, MFC, Apache*, *etc.*

(7) Library and Software Technology. This facet refers to the libraries and software technologies targeted by the question, such as: *jQuery, Swing, GWT, Ajax, RegEx, JSON, XML, JSP, Http, URL, Dom, UML, Image, Video, Lucene*, *etc.*

(8) Topic. This facet reflects the topics of the candidate questions and it is dynamically obtained by clustering, using the titles, tags, and the text of the questions.

The eight facets help us classify StackOverflow questions on three dimensions, *i.e.,Question Type*, *Question Environment* and *Question Topic*. The *Question Type* is determined based on the syntactic structure of the question and the *Concern* facet. For now, *Question Type* corresponds to the *Concern* categories, but we decided to define it as a separate dimension, as the types of question could include other attributes in the future. The *Question Environment* is determined based on the facet categories contained by the question, corresponding to the *Language*, *Operating System*, *Database*, *Development Tool*, *Middleware and Framework*, *Library and Software Technology* facets. The *Question Topics* are extracted from the *Topic* facet. It should be noted that these eight facets are not orthogonal. For example, *Topic* information may also be reflected on the *Language* facet.

All the categories for the first seven facets are determined during preprocessing and stored in XML files, which, as we explained before, can be updated at any time by the user.

3.3 Static Facet Analysis

Static facet analysis determines the categories of each question on each static facet in the preprocessing phase. To this end, we first apply linguistic analysis to parse the sentences of each question and then match their syntactic structures against a set of syntactic patterns to identify the categories of the question on each facet. In addition, we also analyze the tags and code fragments of each question to identify some of the categories.

3.3.1 Linguistic Analysis. We parse the text of each question using a natural language parser (*e.g.,*Stanford CoreNLP [3] used in our implementation) to annotate each of its sentences with POS (Part-Of-Speech) tags. Based on the POS tags of each sentence, we identify its syntactic structure using the reference structure bellow.

[*Subject Group, Predicate Group, Object Group*] [*Affirmative, Negative*][*Declarative, Question*]

A sentence is determined to be *negative* if it contains a negation word (*e.g.,*"not", "no", "none", *etc.*) or if it contains a negative verb (*e.g.,*"forbid", "prohibit", "fail", "die", "prevent". *etc.*) in its *Predicate Group*. A sentence is determined to be *question* if it contains a question word (*e.g.,*"why", "what", "how", *etc.*) or a question mark.

For example, the sentence "*I am new to Visual Studio, I would like to configure it.*" will be parsed to the two syntactic structures below. The *Subject Group* in both sentences is formed of "*I*", whereas "*am new to*" and "*would like to configure*" form the *Predicate Group*, "*Visual Studio*" and "*it*" form the *Object Group*.

1. [*I, am new to, Visual Studio*] [*Affirmative*][*Declarative*]
2. [*I, would like to configure, Visual Studio*] [*Affirmative*] [*Declarative*]

3.3.2 Syntactic Pattern Matching. Based on our initial analysis of the StackOverflow questions, we found that most of questions that belong to the same *Question Type* or *Question Environment* have similar syntactic patterns. For example, programming languages usually appear after prepositions (*e.g.,*"by Java", "with Java", *etc.*). So we can define syntactic patterns for the categories of different facets and determine the categories of a question by matching the syntactic structures of its sentences against the syntactic patterns.

Table 2 shows the 16 templates of syntactic patterns we use in this paper. Syntactic patterns instantiated from these templates can describe a majority of questions in StackOverflow. As mentioned before, these patterns can be expanded, as needed. The second, third, and fourth columns of the table describe the template, the facets and their categories (in the brackets) that the template applies to, and an example a of syntactic structure that conforms to the patterns, respectively. In the third column, an asterisk in the bracket after a facet means that the template applies to all the categories of the facet.

A template can be instantiated into a specific syntactic pattern for a category of a facet by replacing the parameters (*e.g.,*"$\#Term\#$") of the template with specific terms. In the examples of Table 2 (the fourth column), the bold Italic terms represent those that are used to replace corresponding parameters. For example, the first template can be instantiated into a specific syntactic pattern for *Language (Java)* by replacing "$\#Term\#$" with "Java"; the second template can be instantiated into a specific syntactic pattern for *Operating System (Linux)* by replacing "$\#Term_1\#$" with "use" and "$\#Term_2\#$" with "Linux".

To match syntactic structures against syntactic patterns, we need to identify the synonyms of the terms used in the patterns. For example, in the third example of Table 2, if we only define the pattern for *Concern (Configuration)* with the term "configure", then we can not identify a question for configuration problems that uses the word "set" instead of "configure". Therefore, we created a domain

[3] http://nlp.stanford.edu

Table 2: Templates of Syntactic Patterns

	Syntactic Pattern Template	Facet Categories	Example of Syntactic Structure
1	#$Term$# appears after a preposition in the *Subject Group* or *Object Group*	Language (*), Operating System (*), Library and Software Technology (*), Database (*), Middleware and Framework (*), Development Tool (*), Concern (Algorithm, implementation)	Language (Java): [I, convert a String to, an int in ***Java***][Affirmative][Declarative]
2	#$Term_1$# appears in the *Predicate Group*, #$Term_2$# appears in the *Object Group*, and the sentence is affirmative and declarative	Language (*), Operating System (*), Library and Software Technology (*), Database (*), Middleware and Framework (*), Development Tool (*), Concern (Error, Sample Code)	Operating System (Linux): [I, want to ***use***, ***Linux*** commands in C source code][Affirmative][Declarative]
3	#$Term$# appears in the *Predicate Group*, and the sentence is a question	Concern (Configuration, Explanation)	Concern (Configuration): [How I, do ***configure***, tomcat][Affirmative][Question]
4	#$Term$# appears in the *Object Group*, and the sentence is a question	Concern (Error, implementation)	Concern (Error): [How I, can fix, the ***exception***][Affirmative][Question]
5	#$Term$# appears in the *Subject Group*, and the sentence is affirmative and declarative	Concern (Error)	Concern (Error): [The ***exception***, is catched in, the block][Affirmative][Declarative]
6	#$Term_1$# appears in the *Subject Group*, and modal #$Term_2$# appears in the *Predicate Group*, and the sentence is a question	Concern (Implementation)	Concern (Implementation): [***What*** I, ***can*** do to resolve, the connection issue][Affirmative][Question]
7	#$Term$# appears in the *Subject Group*, and the sentence is a question	Concern (Explanation)	Concern (Explanation): [***Why*** you, do create, a View in a database][Affirmative][Question]
8	#$Term$# appears in the *Predicate Group*, and the sentence is negative	Concern (Explanation)	Concern (Explanation): [I, don't ***understand***, Application Domains][Negative][Declarative]
9	#$Term$# appears in the *Object Group*, and the sentence is negative and declarative	Concern (Error)	Concern (Error): [log4net, is not, ***working***][Negative][Declarative]
10	#$Term_1$# appears in the *Subject Group*, the *Predicate Group* does not contain #$Term_2$#, and the sentence is a question	Concern (Implementation)	Concern (Implementation): [***How*** I, do remove, a submodule][Affirmative][Question]
11	#$Term$# appears before *to infinitive* in the *Predicate Group*, and the sentence is a question	Concern (Implementation)	Concern (Implementation): [it, is right ***way*** to upload, images][Affirmative][Question]
12	#$Term_1$# appears before *to infinitive* in the *Predicate Group*, the *Predicate Group* does not contain #$Term_2$#, and the sentence is affirmative	Concern (Implementation)	Concern (Implementation): [I, ***want*** to implement, Key Listener on jlabel][Affirmative][Question]
13	#$Term_1$# appears in the *Subject Group*, #$Term_2$# appears in the *Object Group*, and the sentence is a question	Concern (Error)	Concern (Error): [***What***, is, the ***error*** here][Affirmative][Question]
14	#$Term$# appears after *to infinitive* in the *Predicate Group*, and the sentence is affirmative	Concern (Configuration, Explanation), Database (*)	Concern (Explanation): [I, want to ***understand***, Android Log][Affirmative][Declarative]
15	#$Term_1$# appears before *to infinitive* in the *Predicate Group*, #$Term_2$# appears in the *Object Group*, and the sentence is affirmative	Language (*), Operating System (*), Library and Software Technology (*), Middleware and Framework (*), Development Tool (*)	Language (JavaScript): [I, am ***new*** to, ***javascript***][Affirmative][Declarative]
16	#$Term$# appears in the *Subject Group* or *Object Group*, and the sentence is affirmative	Concern (Algorithm, Design, Implementation)	Concern (Algorithm): [The ***algorithm***, is used to calculate, this checksum][Affirmative][Declarative]

Table 3: Examples of Semantic Classes

Facet and Category	Standardized Term	Synonyms
-	*SEARCH*	find, search, look
-	*USE*	use, apply, have, need, make, utilize
-	*PRODUCE*	produce, create, generate
-	*GET*	get, receive, obtain
Operating System (OS X)	*OSX*	Mac, OSX, OS X
Concern (Error)	*ERROR*	error, exception, bug, warn, defect

dictionary with a set of semantic classes, each of which includes a standardized term and its synonyms. For each parameter in a template shown in Table 2 and for each facet and category that the template applies to, we identified one or several most popular terms in StackOverflow that can be used to instantiate the template and created a semantic class with each term. For a semantic class, we used WordNet [13] to identify candidate synonyms of its standardized term and confirmed ones were added to the semantic class. When a new term was added, its synonyms were also identified using WordNet and considered for inclusion. Table 3 shows some examples of semantic classes. Some semantic classes belong to specific facets and categories. For example, the semantic class *ERROR* belongs to the *Error* category of the *Concern* facet. The other ones are generic, such as, *USE* and *PRODUCE*, which can be used for different facets and categories.

Based on the semantic classes, we instantiated the templates in Table 2 into syntactic patterns for specific facets and categories. For example, the syntactic patterns for *Concern (Error)* are shown in Table 4, which were instantiated from five different templates based on the semantic classes shown in Table 2. A complete list of semantic classes and instantiated syntactic patterns can be found in our replication package [1]. As with the facets and facet categories, these patterns can be easily modified and new ones can be added by the user.

Table 4: Syntactic Patterns for *Concern (Error)*

Template	Syntactic Pattern
2	*PRODUCE* or *GET* appears in the *Predicate Group*, *ERROR* appears in the *Object Group*, and the sentence is affirmative and declarative
4	*ERROR* appears in the *Object Group*, and the sentence is question
5	*ERROR* appears in the *Subject Group*, and the sentence is affirmative and declarative
9	*WORK* appears in the *Object Group*, and the sentence is negative and declarative
13	*WHAT* appears in the *Subject Group*, *ERROR* appears in the *Object Group*, and the sentence is question

The syntactic pattern matching is conducted in the following way. For a category c of a facet f, all the sentences of a question q are matched against c's syntactic patterns. If a sentence of q matches any of c's syntactic patterns, then q is categorized into c on f. Note that q can be categorized into multiple categories of f if q matches the syntactic patterns of multiple categories of f. On the other hand, if q does not match the syntactic patterns of any category of f, then q's category on f remains undetermined.

3.3.3 Tag and Code Analysis. In addition to the text of the questions, the tags and code fragments of StackOverflow questions are also used for category identification on different facets. For each category c of a facet f in *Question Type* or *Question Environment*, if any of the terms in the semantic classes that belong to c appears in the tags of a question q, then q is categorized into c on f. Code fragments embedded in a StackOverflow question and its answers are explicitly tagged in the StackOverflow dump. If a question has code fragments in its text or answers, then it is categorized into *Sample Code* on the *Concern* facet. If the code fragments of a question contain the keyword "exception", "error", or "warn", it is categorized into *Error* on the *Concern* facet.

3.3.4 Example. The categories of the static facets of StackOverflow Question 31766870 "Connect to MySQL Database Remote" are shown in Table 5, as an example. The third column gives the pattern that supports the categorization, which can be a syntactic pattern instantiated from a template, or based on tags or code fragment. The fourth column shows the instances of the corresponding pattern, which can be sentences or code fragments in the text of the question. It can be seen that a question can be categorized into more than one categories on some facets (*e.g.,Concern*) or none of the categories on some facets (categorized into *Other*, *e.g.,Development Tool*).

3.4 Facet Extraction and Mining

The multi-faceted categorization during the interactive search is dynamically generated based on candidate results (*i.e.,*questions that are returned as results to a query). It includes two parts, *i.e.,*the static facets and the *Topic* facet.

The categories of static facets are generated based on the categorization determined in preprocessing. For each static facet, all the categories of it that at least one candidate result belongs to are extracted and shown on the multi-faceted user interface.

The *Topic* facet is dynamically generated by clustering candidate results and generating labels for each cluster. To this end, we treat the title, text, and tags of each question in candidate results as a document and normalize all the documents by removing stop words and stemming. We then cluster the documents and each cluster is treated as a category. The topic of each question is described by the labels generated for the cluster they belong to. Our current implementation uses the *Lingo* algorithm provided by *Carrot*2 [4] (an open source search results clustering engine), which can generate readable labels to help users better choose facet categories. There are many other techniques available for topic extraction, such as, using Latent Dirichlet Allocation (LDA) [4], or Formal Concept Analysis (FCA) [7]. Investigating such alternative approaches is subject of future work.

The facets are used to filter and organize the candidate questions, returned as results to a user query. The questions are grouped based on their *Type*, *Environment*, and *Topic*. The idea is that the users will only explore questions in one category, or at the intersection of a few categories. When multiple categories of a facet are chosen, the candidate questions that belong to any of them will be selected. When the categories of multiple facets are chosen, the candidate questions that conform to the chosen categories of all the facets will be selected.

Within each category the questions are sorted according to relevance to the search keywords. We use the sorting mechanism provided by the integrated search engine (e.g., Apache Lucene) with the following setting of weights on search fields: question title has the highest weight (*i.e.,*3), tags have the second highest weight (*i.e.,*2), and question text has the lowest weight (*i.e.,*1).

4 IMPLEMENTATION

We implemented a web-based StackOverflow question searching tool based on the proposed MFISSO approach. The current implementation imported the data of the StackOverflow dump released in January 2016, which contains 10,789,362 questions and 17,650,222 answers. It uses Apache Lucene 4.6 for indexing and keyword-based searching, and Stanford CoreNLP V1.1.0 for natural language processing.

In keyword-based searching, MFISSO filters out all the stop words in the keyword-based query given by the user and then uses all the other keywords for searching with Lucene's searching APIs. Based on the results returned by Lucene, the tool filters out the questions that have no answers or whose scores are lower than 0. Then the tool chooses a number of top-ranked candidate questions for facet extraction and mining. For example, in our experimental study we chose to return 200 top-ranked questions for facet extraction and mining.

The MFISSO tool has a typical multi-facet search UI, as shown in Figure 2. The three main areas in the UI are *Search Panel*, *Result List*, and *Facet Panel*.

To search StackOverflow questions, a developer can input an initial set of keywords in the *Search Panel* and then click the *Search* button to start the search process. The returned candidate questions are sorted and displayed in the *Result List*. The title, text, tags, and three top-rated answers of each question are shown on the UI. The keywords in the query are highlighted in the text of each question. Embedded code fragments are shrunk and can be shown by clicking the *Code* buttons in the text or answers of a question. Clicking the title of a question will open the page of the question on StackOverflow site.

The *Facet panel* shows all the facets and categories generated for the current candidate questions. Categories of a facet are displayed as a tree with descriptive labels. The number after the label of each category shows the number of candidate questions involved in the category. The developer can select or unselect categories of different facets by clicking the checkboxes next to the category's label. Each time after the selected categories are changed, the *Result List* is updated with candidate questions that conform to the selected categories for preview. If the developer decides to refine the results with the selected categories, she can click the *Update* button below the facets. After that, the candidate questions are refined and the categories of all the facets are updated based on the refined candidate questions.

We provide a feedback mechanism for the developers using our tool. During or after a searching, the developer can click the *Suggest* button in the *Search Panel* and provide her feedback of the tool.

5 EVALUATION

The aim of our approach is improving StackOverflow search using multi-facets. In order to assess whether our approach achieves its goal, we implemented MFISSO, as described in the previous section and conducted an empirical evaluation, for answering the following research question:

RQ *Does MFISSO improve the search in StackOverflow?*

We performed two related empirical studies in order to answer our research question: (1) a controlled experiment where subjects solved a set of programming tasks using MFISSO and the StackOverflow search feature; and (2) an analytical comparison of the retrieval performance of MFISSO and the StackOverflow search.

[4]http://project.carrot2.org

Table 5: Categories of Static Facets of StackOverflow Question 31766870 "Connect to MySQL Database Remote"

Facet	Category	Pattern	Instance
Concern	Implementation	Template 6	***What*** other things do I ***need*** to do to connect?
	Sample Code	Code Fragment	Code Fragment
Language	Java	Template 1	Then in the ***Java*** application (that utilizes the MySQL database) I did jdbc:mysql://public_ip:3306/database_name
Operating System	Windows	Template 2	I also commented out bind-address in the my.ini file (I ***have windows***)
Database	MySQL	Template 2, Tag	1. I ***have*** a ***MySQL*** database on another computer 2. that ***utilizes*** the ***MySQL*** database
Development Tool	Other	-	-
Middleware and Framework	Other	-	-
Library and Software Technology	URL	Template 1	I used my public ip address in the jdbc ***url***

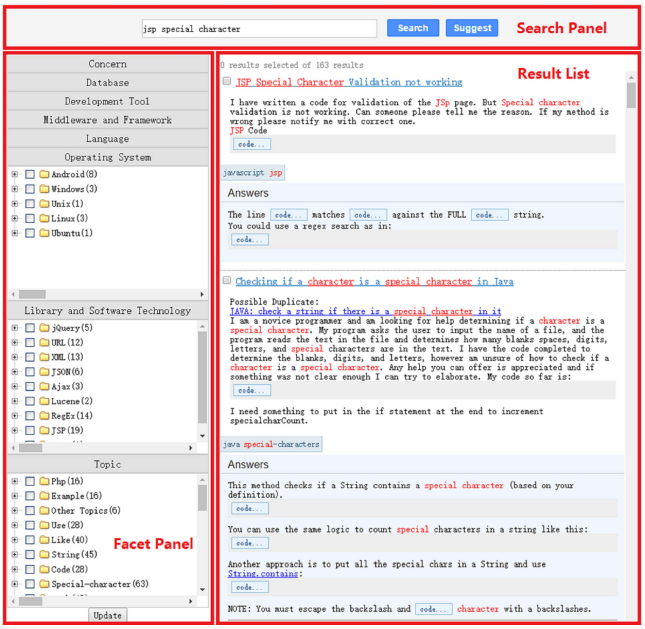

Figure 2: Layout of MFISSO Tool

5.1 Controlled Experiment

5.1.1 Experiment Design. The *object* of the study was to solve 10 programming tasks by finding the relevant solutions on StackOverflow. The tasks were defined by two of the authors and range in difficulty. For example, one of the easier tasks is "*How to display a tree structure using JSP.*", whereas one of the more difficult tasks is "*I am using Eclipse to develop a web project in Linux. How to configure Tomcat in Eclipse.*" The ten tasks are of different type, such as, debugging task, tool configuration task, algorithm task, implementation task, *etc.*All ten tasks are described in our replication package [1].

The *subjects* of the study are 20 students from the School of Computer Science at Fudan University - six undergraduates, eleven masters, and three PhD level. The subjects received a USB-memory stick or a mug, upon participation in the experiment.

The *dependent variable* is the tool usage, which has two levels: MFISSO and StackOverflow search.

The independent variables are: *correctness* (*i.e.*,a measure of effectiveness) and *completion time* (*i.e.*,a measure of efficiency).

The *controlled variables* are the *task difficulty* and *subject experience*. *Task difficulty* has four levels: very easy, easy, hard, very hard. These values are relative to each other and were assigned to the ten tasks by two of the authors, based on their experience. The ten tasks were divided into two groups P1 and P2, such that each group had one very easy task, two easy tasks, one hard task, and one very hard task.

Prior to the selection of the subjects, we asked them to complete a pre-experiment survey about their experience. The survey and the answers are available in our replication package [1]. In order to control the subject experience, we divided the subjects in two groups of ten subjects each, S1 and S2, using a randomized block design, with the *experience* as the blocking factor. Group S1 had eight subjects with the *Experienced* level and two subjects with the *Novice* level. Group S2 had one subject with the *Expert* level, six subjects with the *Experienced* level, and three subjects with the *Novice* level.

We chose a *within-subject design* and used *counterbalancing* not to confound the order in which a task is performed with the experimental treatment. So we defined four treatments as follows: T1 - MFISSO+P1; T2 - MFISSO+P2; T3 - SO+P1; T4 - SO+P2. Then we assigned S1 to T1 and T4 and S2 to T2 and T3, meaning that each group of subjects performed all tasks (P1+P2) and used MFISSO for half the tasks (P1 or P2) and the StackOverflow search for the other half. Since the ten tasks are independent, we did not anticipate any learning effect, which allowed us to employ a within-subject design. Since each subject used both search tools, we were able to ask them to compare the use of the tools via a post-experiment survey.

A pilot study was done with two participants, which helped us in better assessing the difficulty of the ten tasks and the approximate time it takes to answer them. The actual study was carried out during two days. During day one, the subjects participated in a 30 minutes tutorial on how to use MFISSO. They were given time after

the tutorial to use the tool freely. The second day, the subjects were shown a 10 minutes demo of MFISSO to remind them how to use it. After that, they answered the tasks, one by one. They were given maximum 15 minutes per question, for a total not to exceed 150 minutes, in order to eliminate fatigue. Each subject had to answer all five questions in P1 or P2, before answering the other five, using the tools assigned to them via the treatment design. Two of the authors were present during the session in order to answer any questions of the subjects and to observe that they are using the proper tool to answer the proper questions.

In order to answer our research question, we studied the effect of the dependent variable (*i.e.,tool usage*) on the independent variables (*i.e.,correctness* and *completion time*). We formulated two null hypothesis and their alternatives, for each variable, see Table 6.

5.1.2 Data Collection. During the session each subject ran a screen capture software on their workstation. The screen captures were later analyzed to determine the exact time it took each subject to answer each question, as well as to record the queries they issued and the facets they selected when using MFISSO. Time was collected and reported in seconds. For each task, the subjects spend some time on understanding the task and then they started searching for StackOverflow questions. The time it took them to solve a task was considered since the moment they started searching until they write the final answer. The time it took them to understand the tasks was not considered.

For each task, the subjects reported the StackOverflow question and answer, which has the best solution for the task. The set of correct solutions was identified based on the pilot study. Note that for some of the tasks, the complete solution required combining elements from answers to more than one question. As with any programming task, there are always more than one correct solution. So, in the cases where the subject's answer did not match one of the previously identified correct solutions, two of the authors assessed the proposed solution for correctness. These are the two authors that formulated the tasks and are experts in solving each task. For each reported solution answer the subjects received a score of '0' if their answer was incorrect or '1' if their answer was correct. If the answer was not correct but can help to solve the problem, it was considered partially correct (*i.e.,*scored '0.5'). In the cases where the solution was formed from several elements (two in our tasks), the answer was also considered partially correct (*i.e.,*scored '0.5') if only one element of the solution was correctly identified.

In the end, each participant solved ten tasks, five with each tool, hence each task was solved by two subjects, each using a different tool (*i.e.,*SO and MFISSO).

At the end of the session, each subject completed a survey, to provide feedback about the perceived difficulty of their tasks, challenges or difficulties in the process, their experience with the tools, and suggestions for the improvement of MFISSO.

5.1.3 Results. Our choice of single-factor, within-subject design for our experiment and the formulation of the null hypothesis two-tailed alternatives, suggested that the suitable parametric test for hypothesis testing is a two-sample t-test. We minimize the chance of type-I errors, as we compare only two groups (*i.e.,*using MFISSO vs StackOverflow).

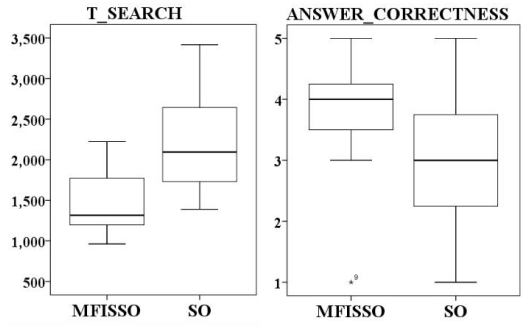

Figure 3: Performance Comparison of SO and MFISSO

As a precondition for the test we checked for the normality of the dependent variables, completion time and correctness, with respect to the two level of tool, using the k-s test for normality. The test indicated that the normality assumption is met.

We also tested our data for homogeneity of both correctness and completion time, using ANOVA test and in both cases the assumption was met.

Table 7 and Figure 3 present the descriptive statistics of the aggregated correctness and completion time results for each level of the tool variable.

The effect of MFISSO on search time. MFISSO has significantly impact on the completion time, p=0.001. In average, MFISSO allowed subjects to solve the tasks 35% faster (1,474.5 seconds vs. 2,256.75 seconds). Remember that the subjects solved five tasks with each tool.

The effect of MFISSO on answer correctness. As in the case of completion time, MFISSO has significantly impact on the accuracy, p=0.028. In average, when using MFISSO subjects accurately solved 3.85 questions compared to 2.95 when using StackOverflow. Remember that the subjects solved five tasks with each tool.

We expected somewhat higher correctness, so we investigated the correctness results deeper. The easier questions were answered typically well by the subjects, regardless whether they used MFISSO or StackOverflow. For example, all 20 participants answered correctly the first question in group P1. However, for some difficult questions, such as the fifth question in group P1 and the fourth question in group P2, participants who use MFISSO were able to perform better than those using StackOverflow. For half of the participants these questions were answered towards the end of the experiment, regardless of the tool they used. As it happens, using MFISSO to answer these questions yielded a substantial gain in time, over StackOverflow. Hence, we believe that, given the level of fatigue towards the end of the experiment, MFISSO's assistance in saving time, helped the subjects answer better these answers. When it comes to the third question of P2, participants performed weak no matter which tool they used. Participants using MFISSO only have the correctness at 4.5 out of 10, and those using StackOverflow had the correctness at 5/10. We concluded that this situation occurred because some of the subjects are not familiar with how to parse Java code although they know well about Java. One participant stood out, as he answered only one correct answer when dealing with questions in group P1 and had three correct answers when dealing with questions in group P2, using MFISSO. When investigating his recording, we observed that he barely used the

Table 6: Null and Alternative Hypotheses

	Null hypothesis		Alternative hypothesis
HT0:	The tool usage does not impact the time required to complete the tasks.	*HT1*:	The tool usage impacts the time required to complete the tasks.
HC0:	The tool usage does not impact the correctness of the solutions to the tasks.	*HC1*:	The tool usage impacts the correctness of the solutions to the tasks.

Table 7: Results Statistics

		avg	min	max	median	standard dev.	T-Test	
							T	p
Completion time	MFISSO	1,474.5	962	2,224	1316	378.38	-4.061	0.001
	StackOverflow	2,256.75	1,390	3,417	2,095	677.27		
Answer correctness	MFISSO	3.8	1	5	4	0.90	2.37	0.028
	StackOverflow	2.95	1	5	3	1.05		

facets. Still, we decided to consider his answers valid, rather than accidental outliers.

5.1.4 User Experience. The post-study survey included questions about the experience of the subject with MFISSO. Eighteen (90%) participants strongly agree (10) or agree (8) that they will use MFISSO frequently in the future. Nineteen (95%) participants strongly agree (9) or agree (10) that the tool is easy to use. An example of comment in the free form answers is: "*This tool is easy enough to use with clear and simple classification*". Only one participant was not sure on how easy the tool is to use and said that "*Facets are a little bit too many*". All of them disagree when asked whether one needs to learn a lot before one could effectively use this search tool. When asked why, one of them said that "*The tool is easy to use since its similar to the filter tool of online shopping websites*" and another commented that "*Anyone who is familiar with a search engine should get started quickly, because it is designed in a same way*". Most importantly, they also thought MFISSO helped them more than just StackOverflow (10 strongly agreed and 10 agreed). One participant said that "*StackOverflow search engine is not very good, I can get better result with less time with the tool*", while another one said that "*The search engine of SO is really bad, I used to use Google to retrieve related questions on SO instead*". These answers support our initial motivation for developing MFISSO. The participants also gave some suggestions on improving MFISSO. For example, someone wanted to "*add more facets in order to be utilized by diverse kinds of developers*" and someone hoped that a "*more beautiful UI could make MFISSO become more popular among users*". Overall, the participants were pleased with the use and performance of MFISSO.

5.2 Analytical Study

To complement the controlled experiment, we performed an additional analytical evaluation, comparing the retrieval effectiveness of MFISSO and StackOverflow. This type of evaluation is common in applications of information retrieval in software engineering [5].

The objects of the study are the queries formulated by the subjects in order to solve the ten tasks during the controlled experiments. We extracted the first query issued by each subject for each task, using the captured videos from the experiment. Some subjects used identical queries for the same tasks. After removing the duplicate queries, we had 134 queries left.

In order to answer the research question, we measure the effectiveness of the retrieval [17] (*i.e.*,the location of the first relevant question in the set of retrieved results). This measure is commonly used in IR-based feature location applications [11]. One of the authors ran all the 134 queries to compute the effectiveness. In the case of StackOverflow, the effectiveness was measured by locating the relevant question in the list of results returned by the StackOverflow search engine. In the case of MFISSO, the location was retrieved within the facet where the relevant question was categorized.

Table 8: The Effectiveness Analysis for The 134 Queries

Queries	Effectiveness SO					Effectiveness MFISSO			
	Retrieved	Min	Med	Avg	Max	Retrieved	Min	Med	Avg
134	77	1	5	12.33	98	105	1	1	1.33

Table 8 shows the results of the effectiveness analysis for the 134 queries. Some queries did not retrieve the relevant question, 57 (42.5%) for StackOverflow and 29 (21.6%) for MFISSO. The average/median effectiveness for MFISSO are much lower than for StackOverflow (1.33/1 vs. 12.33/5), which means that using MFISSO the relevant question is, on average, in the top 1.33 results in the facet (1 median), whereas for StackOverflow, the relevant question is on position 12.33, on average (5 median).

These results support the findings of the controlled experiment with respect to time, as we can expect that relevant questions located closer to the top of the results list should be found faster.

5.3 Threats to Validity

5.3.1 Internal validity. We conducted the pre-experiment survey to ensure that the subjects have enough knowledge to complete the ten tasks. In addition, we used their self-assessed expertise information to balance the two groups, although our choice of within-subject design mitigates partially the effect of the expertise. In order to ensure that the subjects are familiar enough with using MFISSO and StackOverflow, we provided the tutorial and training on using the tools on day 1 before the experiment, and provided a ten minutes demo before starting the experiment. While we did not expect any learning effect between solving the ten tasks, we counterbalanced the treatments to mitigate the effect of fatigue. We expected that the subject will be more tired while answering the final few questions, hence potentially impacting their performance. While the subjects received small gifts for participation, there was little incentive for them to perform, except for the fact that they were observed by two of the authors.

The ten tasks were designed by the authors and, thus, there could exist some bias towards MFISSO because this tool was also designed by the authors. To mitigate this potential bias, we selected tasks that ranged in type and difficulty. 80% of the subjects rated the tasks overall as "moderate" in the post-experiment survey and only one thought they were "Hard", which indicates that the tasks were neither trivial nor too difficult. The difficulty level was assessed by two of the authors, so it is somewhat subjective. Yet, we do not have means to asses the difficulty more objectively for such tasks. We defined the two group of question, P1 and P2, to balance the

difficulty. We aggregated and analyzed the time and correctness for each group of tasks. It took, in average, 2,077 seconds for the subjects to solve the tasks in P1 and 1,642 seconds to solve the tasks in P2. At the same time, the subjects answered correctly 2.9 question from P1, in average, and 3.9 answers from P2, in average. This analysis indicates that the questions in P2 were somewhat easier than the ones in P1, despite the fact that they were rated the same (as a group) by the authors.

5.3.2 External validity. The twenty participants were all students, including six undergraduates, eleven master students and three doctoral students. While their experience and background are somewhat varied, they are hardly a representative sample for the software developers community. However, we have no reason to believe that the results with professional programmers would be any different. In future work, we will include professional developers as well, in subsequent experiments.

MFISSO's implementation only considered the eight facets mentioned in the paper with a limited number of categories. Clearly not all type of questions can be categorized by the facets we used. However, we believe that including additional facets and categories would make MFISSO even more efficient and effective to use.

The ten tasks certainly do not cover all types of tasks that developers use StackOverflow for, but they were developed based on the authors' past experiences in using StackOverflow.

6 CONCLUSIONS AND FUTURE WORK

We hypothesized that categorizing the StackOverflow question based on their attributes will help developer retrieve relevant questions and answers more effectively and efficiently. Our approach, MFISSO, extracts the concern, environment, and topics of questions, and provides a multi-faceted categorization for StackOverflow questions that allows interactive and iterative refinement of search results. A controlled experiment with 20 subjects solving ten programming tasks revealed that they solved the tasks faster and more accurately than using the StackOverflow search. An analytical evaluation of 134 user queries also indicated that relevant questions can be retrieved faster with MFISSO than with the StackOverflow search.

The improved retrieval is achieved with only the use of eight facets. We expect that extending the number of facets and categories in each facet will yield even better retrieval. Part of our future work is to extend the categories and facets. Further refinement of the syntactic patterns is also envisioned.

While we used tasks of several types in the controlled experiment, we did not study the impact of the task type on the effectiveness and efficiency of retrieval with or without MFISSO. A different study would be needed for that and we plan to do it in the future.

Our approach can be instantiated to be used with other Question-Answer sites, by redefining the facets and categories, and some of the syntactic patterns. Future work will expand to applications beyond StackOverflow.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China under Grant No. 2016YFB1000801 and the Shanghai Science and Technology Development Fund (16JC1400801).

REFERENCES

[1] 2018. MFISSO Replication Package. http://www.se.fudan.edu.cn/research/SOSearch
[2] Anton Barua, Stephen W. Thomas, and Ahmed E. Hassan. 2014. What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empirical Software Engineering* 19, 3 (2014), 619–654.
[3] Blerina Bazelli, Abram Hindle, and Eleni Stroulia. 2013. On the Personality Traits of StackOverflow Users. In *2013 IEEE International Conference on Software Maintenance*. 460–463.
[4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research* 3 (2003), 993–1022.
[5] Jane Cleland-Huang, Olly Gotel, and Andrea Zisman (Eds.). 2012. *Software and Systems Traceability*.
[6] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2013. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process* 25, 1 (2013), 53–95.
[7] Bernhard Ganter, Gerd Stumme, and Rudolf Wille. 2005. *Formal concept analysis: foundations and applications*. Vol. 3626.
[8] Latifa Guerrouj, David Bourque, and Peter C. Rigby. 2015. Leveraging Informal Documentation to Summarize Classes and Methods in Context. In *37th IEEE/ACM International Conference on Software Engineering*. 639–642.
[9] Marti A. Hearst. 2006. Clustering versus faceted categories for information exploration. *Commun. ACM* 49, 4 (2006), 59–61.
[10] Hongwei Li, Xuejiao Zhao, Zhenchang Xing, Lingfeng Bao, Xin Peng, Dongjing Gao, and Wenyun Zhao. 2015. amAssist: In-IDE ambient search of online programming resources. In *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering*. 390–398.
[11] Andrian Marcus and Sonia Haiduc. 2011. Text Retrieval Approaches for Concept Location in Source Code. In *Software Engineering - International Summer Schools*. 126–158.
[12] Collin McMillan, Denys Poshyvanyk, Mark Grechanik, Qing Xie, and Chen Fu. 2013. Portfolio: Searching for relevant functions and their usages in millions of lines of code. *ACM Trans. Softw. Eng. Methodol.* 22, 4 (2013), 37:1–37:30.
[13] George Miller. 1998. *WordNet: An electronic lexical database*. MIT press.
[14] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. 2012. What makes a good code example?: A study of programming Q&A in StackOverflow. In *28th IEEE International Conference on Software Maintenance*. 25–34.
[15] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. 2014. Mining StackOverflow to turn the IDE into a self-confident programming prompter. In *11th Working Conference on Mining Software Repositories*. 102–111.
[16] Denys Poshyvanyk, Malcom Gethers, and Andrian Marcus. 2012. Concept location using formal concept analysis and information retrieval. *ACM Trans. Softw. Eng. Methodol.* 21, 4 (2012), 23:1–23:34.
[17] Denys Poshyvanyk, Yann-Gaël Guéhéneuc, Andrian Marcus, Giuliano Antoniol, and Václav Rajlich. 2007. Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval. *IEEE Trans. Software Eng.* 33, 6 (2007), 420–432.
[18] Denys Poshyvanyk and Andrian Marcus. 2007. Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code. In *15th International Conference on Program Comprehension*. 37–48.
[19] Mohammad Masudur Rahman, Shamima Yeasmin, and Chanchal K. Roy. 2014. Towards a context-aware IDE-based meta search engine for recommendation about programming errors and exceptions. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering*. 194–203.
[20] Jinshui Wang, Xin Peng, Zhenchang Xing, and Wenyun Zhao. 2013. Improving feature location practice with multi-faceted interactive exploration. In *35th International Conference on Software Engineering*. 762–771.
[21] Shaowei Wang, David Lo, and Lingxiao Jiang. 2013. An empirical study on developer interactions in StackOverflow. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. 1019–1024.
[22] Wei Wang, Haroon Malik, and Michael W. Godfrey. 2015. Recommending Posts concerning API Issues in Developer Q&A Sites. In *12th IEEE/ACM Working Conference on Mining Software Repositories*. 224–234.