Recommending Forum Threads for Exception-related Bugs

Abstract—**Most modern programming languages implement their own exception handling mechanisms. When a bug triggers an exception, to better understand the bug, programmers often search online forums (*e.g.* Stackoverflow) for related threads. As a forum can have many threads and many issues discussed, it becomes difficult to search the right ones for a specific thrown exception. Our empirical study shows that issue trackers record many manual fixes that are related to various exceptions, while programmers constantly discuss thrown exceptions on forums. Based on the findings, in this paper, we propose an approach that builds the links between bug fixes and forum threads. With the support of such links, we further propose an approach that recommends related forum threads for a given thrown exception. We conduct an evaluation on forum threads from Stackoverflow and bug fixes from GitHub. Our results show that our approach achieves higher hit rate and average rank than existing web search engines and existing approaches.**

Keywords: recommendation, exception, bug, forum, Stackoverflow, GitHub

I. Introduction

An exception is an event, which occurs during the execution of program and disrupts the normal flow of the program's instructions. When an exception occurs within a method, the method creates an exception object and hands it off to the runtime system. Fig. 1 shows the call stack and the exception handler searching process. For many advanced programming languages like Java, C# and Python etc., most errors or bugs triggered during runtime will represent as the form of exception [1]. During software development process, exceptions are strong phenomenons of existence for bugs or bad smells in codes [2].

To deal with exception-related bugs, developers still highly reply on online resources [3], use search engines like Google or Bing for the solutions. However, there are gaps between the exceptions in IDE and the Web. First of all, useful information disperses widely in the Internet in variety of forms, such as online communities, forums, mailing list or Q&A sites [4], [5]. Searching in the Internet usually takes much efforts. A study shows that developers spends average 19% of their time on surfing the Web [6]. Furthermore, traditional searching engines help but far from enough. Google, Bing and other searching engines with query length limitation take few words thus may lose much information. It's also difficult to manually check large amounts of result pages in order to find the right one. Last but not least, web pages usually contain lots of useless and confusing information like advertisements [7]. Gibson et al. [8] estimated that about 40% to 50% of web data could be attributed as noise, and the ratio is constantly increasing due to explosive growth of the Internet.

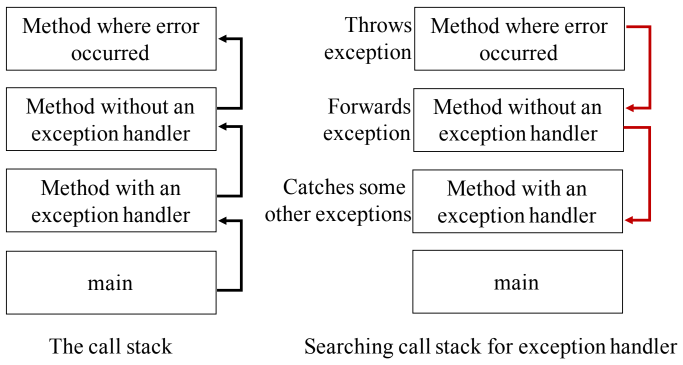
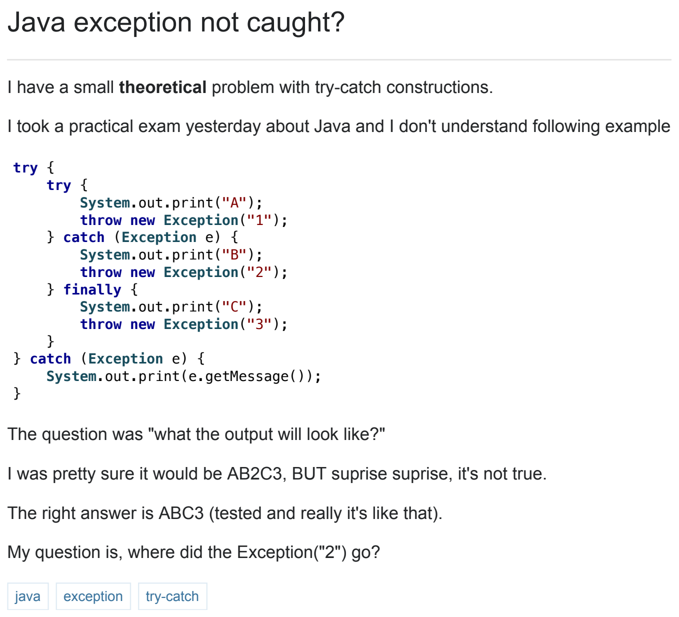

Method where error occurred
Method without an exception handler
Method with an exception handler
main
The call stack
Throws exception
Forwards exception
Catches some other exceptions
Method where error occurred
Method without an exception handler
Method with an exception handler
main
Searching call stack for exception handler

Fig. 1. Call stack and exception handler searching process

It's of great significance to help developers deal with exception-related bugs in a more efficiency way. One of the best solutions is automatic recommendation, which improves the efficiency of developers as well as the the reliability of software project. There have been a considerable amount of research works on automatic provide solutions or suggestions for programming errors or exceptions [9], [10].

Unfortunately, the results of these existing automatic recommendation approaches would probably lead to poor results when applied in exception solving for many reasons. One important reason is lacking of pertinence. Existing approaches are mainly built for general programming errors, like bugs or syntax errors [11], [12]. However, exception is full of specific structures which should be analyzed in a more detailed way. For example, *java.lang.NullPointerException*, a very common exception in Java. We can extract the name of exception *NullPointerException*, also *java* and *lang* refer to the related package or class. Besides, We can also easily extract more formatted information such as stacktrace as well as the codes which throws out the exception. Another reason is losing of context. Usually, number of reasons may lead to the occur of an exception. A same exception may totally differs in coding context, such as class type, function calling sequence or reference libraries. [13]–[15] take only parts of exception context information as query input which can not get complete information.

Stackoverflow is one of the most popular Q&A sites which owns more than 4 million registered users and 11 million questions [16]–[18]. Many developers also share and learn open-source projects in software repository platforms [19]. One of such platforms is GitHub, in which as of 2015, there were



Java exception not caught?

I have a small **theoretical** problem with try-catch constructions.

I took a practical exam yesterday about Java and I don't understand following example

```
try {
    try {
        System.out.print("A");
        throw new Exception("1");
    } catch (Exception e) {
        System.out.print("B");
        throw new Exception("2");
    } finally {
        System.out.print("C");
        throw new Exception("3");
    }
} catch (Exception e) {
    System.out.print(e.getMessage());
}
```

The question was "what the output will look like?"

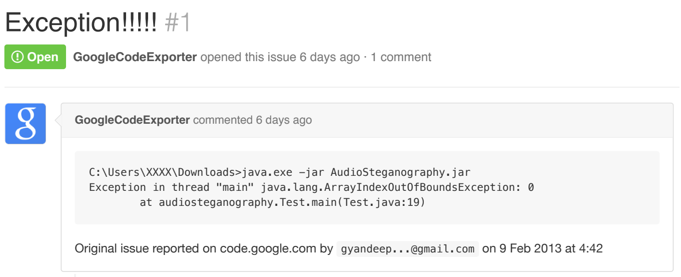
I was pretty sure it would be AB2C3, BUT suprise suprise, it's not true.

The right answer is ABC3 (tested and really it's like that).

My question is, where did the Exception("2") go?

java exception try-catch

(a) An exception-related Stackoverflow question



Exception!!!!! #1

Open GoogleCodeExporter opened this issue 6 days ago · 1 comment

GoogleCodeExporter commented 6 days ago

```
C:\Users\XXXX\Downloads>java.exe -jar AudioSteganography.jar
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at audiosteganography.Test.main(Test.java:19)
```

Original issue reported on code.google.com by `gyandeep...@gmail.com` on 9 Feb 2013 at 4:42

(b) An exception-related GitHub issue

Fig. 2. Examples of exception-related resources in Stackoverflow and GitHub

over 9 million users and 21.1 million repositories [20]. Denzil Correa et al. proposed an survey, and shows more than 60% bug-fixers do search and post questions on community-driven Q&A websites like Stackoverflow to leverage the archived knowledge and get inputs from experts when applicable [21]. As shown in Fig. 2, the large amount of Q&A or project issue resources in Stackoverflow as well as GitHub provide a promising way to build a cross-domain exception solver.

However, we will face some fundamental challenges for our approach. First, there are so many different kinds of information in Stackoverflow and GitHub which may make an influence for recommendation. It's necessary to make out what are the most helpful and important. Second, Stackoverflow and GitHub are two different domains. There are some totally different features or data structures between them. For example, the close state of a GitHub issue usually means the problem has been successfully solved. However, failing to solving a Stackoverflow question usually leads to the close state. We need to deal with the differences in the cross-domain problem. Third, the gap between online community resources and the exception context. As shown in Fig. 2, Most part of resources in Stackoverflow and GitHub are in the form of natural language such as English, while most parts of an exception are formal codes or exception stacktrace.

In order to solve the challenges mentioned, first we conducted an empirical study for Stackoverflow and GitHub resources. Three questions are raised in the study: (1) Why Stackoverflow and GitHub? (2) How suitable for Exception Solve? (3) Which are key features? Then, for cross-domain problem, we propose and build exception tree. An large-scale software programming taxonomy will be help in this case. From top to bottom, exception tree contains three layers, programming language layer, tags layer and exception context layer. All exception-related Stackoverflow Q&As and GitHub issues will link to the tree nodes. For the last challenge, we conduct four kinds of features to train our supervised machine learning model, including lexical features, program features, community based features and exception tree based features. Among them, exception tree based features plays an important role.

Our contributions mainly include:

- An empirical study of exception-related Stackoverflow and GitHub resources
- Propose and construction of exception tree to solve the cross-domain problem
- A semi-supervised learning model with four different features for recommendation

The rest of this paper is organised as follows. Section II describes the related work around exception solvers. Section III provides the approach overview. Section IV is an empirical study for Stackoverflow and GitHub resources. Section V describes the recommendation process which includes four main components. Section VI performs experiments to show the efficiency and accuracy of the approach. Section VII discusses our summary and future work.

II. Related Work

There have been a considerable amount of research works on automatically providing suggestions for programming errors or exceptions. These approaches can be divided into two kinds: retrieval filtering and content-based approaches. For the retrieval filtering approaches [7], [11], [22]–[24], they mainly focus on the way to construct a query from programming exception context, then filter out required results. Several existing search engines, such as general search engine Google or online communities internal search APIs, will accept the queries and return group of results. With the help of ranking models and filtering strategies, results with highest scores are recommended. Mohammad Masudur Rahman [13] uses APIs from Google to recommend relevant context from web pages about programming errors and exceptions. SurfClipse [14] is an context-aware meta searching with the support from Bing, Yahoo and Google APIs. There are advantages for these kinds of approaches. It's simple and reliable to construct search query, such as using term frequency or entropy. However, APIs of search engines usually only accept few keywords and do not support complex query settings. As a result, the exception context information cannot be fully expressed and transferred which leads to poor recommending quality.

TABLE I
EXCEPTIONS IN DIFFERENT PROGRAMMING LANGUAGES

Languages	Exception Keywords	Exception Paradigms	Exception Examples
Java	try, catch, throw, finally, exception	try...throw ***Exception***...catch...***Exception***...finally	java.lang.NullPointerException
Javascript(js)	try, catch, finally, throw, exception, error	try...throw ***Exception***...catch [***Exception***]...finally...	URIError
Ruby	raise, rescue, ensure, throw, catch, exception	raise [***Exception***]...rescue ***Exception***...ensure...	StandardException
C++(cpp)	try, catch, throw, exception	try...throw ***Exception***...catch ***Exception***...	DerivedException
C#(c-sharp)	try, catch, throw, finally, exception	try...throw ***Exception***...catch ***Exception***...finally...	System.DivideByZeroException
Python	try, raise, except, finally, exception	try...raise ***Exception***...except ***Exception***...finally...	ImportError
PHP	try, throw, catch, exception	try...throw ***Exception***...catch (***Exception***)...finally...	DBException
Objective-C	try, catch, throw, finally, exception	try...throw ***Exception***...catch ***Exception***...finally...	BoxUnderflowException

For the content-based approaches, they try to calculate the similarity between resources and programming errors in the IDE [12], [15], [25]–[27]. Seahawk [28], an Eclipse plugin that supports an integrated and largely automated approach to assist programmers using Stackoverflow. They import Stackoverflow documents from the public data dump and build documents index with the help of Apache Luence and it-idf. Joel Cordeiro et al. [12] developed a tool that integrates recommendation of question/answer web resources in Eclipse, according to the context of these exception stacktrace. The approach performs better than a simple keyword-based approach. Denzil Correa et al. [19] conduct approaches based on textual similarly analysis (content-based linguistic features) and contextual data analysis (exploited metadata such as tags associated to a S-tackoverflow question) to recommend Stackoverflow questions for an incoming programming bug. However, all the above methods mentioned take only parts of exception contextual data for similarity calculation. Except for exception name and stacktrace, there are some other valuable information can be made use of, such as related source codes or comments of the discussed exception. Furthermore, without relying on third party searching engines, there are not approaches trying to build a cross-domain exception solution recommender, because of the difficulty to deal with the differences in data and structures.

Specifically, we conduct an empirical study of Stackoverflow and GitHub resources. With the help of study, we design and propose a different approach and construct exception tree, a structured way to manage the relationship between exceptions in different programming languages, tags, libraries and classes. Based on the tree, we propose a unified model that incorporates exception contextual features to automatically recommendation. The recommendation problem is treated as a binary classification problem and solved by Support Vector Machines(SVM).

III. PROBLEM DEFINITION

In this section, we define the recommendation problem in a formal way. Let Stackoverflow questions and GitHub issues represented as unified resource set R. Every single exception-related question or issue $r \in R$ is a tuple $(i_r, b_r, T_r, CS_r, ST_r, EP_r, P_r)$, where i_r is the title, b_r is the question or issue body, T_r is the tag set which annotates the question or issue, CS_r and ST_r is the code fragments and stacktrace in the body, EP_r is the extracted exception keywords and P_r is the replies.

When an exception is thrown, we collect all possible exception-related information, called the context of an exception. Every exception context c can be represented as a four-tuple (l_c, ST_c, CS_c, ED_c), where l_c is the name of the exception, ST_c is the stacktrace, CS_c is the code fragment which throws the exception and ED_c is comments or description about the exception which can be gathered from codes comments or user input. Let all possible exception contexts represented as set C.

As the exception solution problem defined, for each of Stackoverflow question or GitHub issue $r \in R$, we measure the relationship between $(i_r, b_r, T_r, EP_r, P_r)$ with exception context $c \in C$. It is worth noting that we take the recommendation problem as a binary classification problem. So the goal is to find a function $f : R \times C \mapsto \{0, 1\}$, which means for pair $(r, c) \in R \times C$, if $f(r, c) = 1$, then r will recommended to c, others not.

IV. AN EMPIRICAL STUDY OF STACKOVERFLOW AND GITHUB

In this section, we explore whether the idea of cross-domain exception solution recommendation with the crowd knowledge from Stackoverflow and GitHub makes sense empirically. We propose three research questions and answer them based on real data extracted from Stackoverflow and GitHub.

A. Dataset

We get Stackoverflow data dump from Stackexchange[1] archive. The dump of Stackoverflow contains several data files in xml format. Among them, Q&A related data is stored in file posts.xml about 49GB. In order to quickly access, we transfer and index it to our mongoDB storage. Regards GitHub data, because there are 5,000 limitation for single IP address access per day of GitHub Developer API[2], we choose another great GitHub dataset name GHTorrent[3] [29] which creates a scalable offline mirror of data offered through the GitHub REST API.

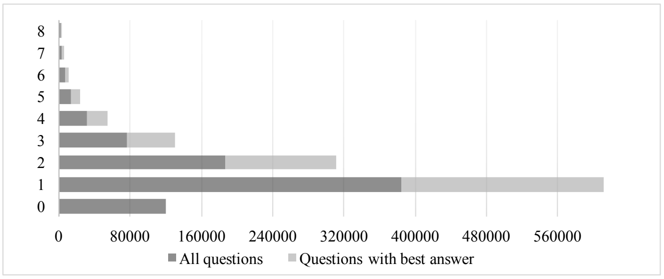
B. RQ1: Why Stackoverflow & GitHub?

Beyond the common understanding we have for Stackoverflow and GitHub, we would like to know to which

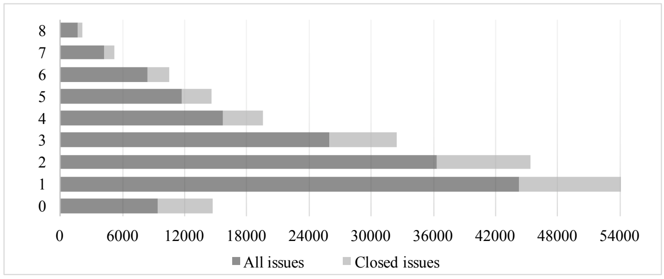
[1]https://archive.org/details/stackexchange
[2]https://developer.github.com/v3/
[3]http://ghtorrent.org/

TABLE II
BASIC DATA ANALYSIS FOR STACKOVERFLOW AND GITHUB DATASET

	Stackoverflow Q&As	GitHub Issues
Date Source	Stackexchange	GHTorrent
Total Quations/Issues	8,052,478	23,606,974
Exception Related Amount	826,185	738,898
Exception Related Rate	10.26%	3.13%



(a) Stackoverflow question answer number distribution



(b) GitHub issue reply number distribution

Fig. 3. Figures for RQ2

extent Stackoverflow and GitHub contains exception-related resources. If there are enough questions or issues related with exception on the two communities, it would make sense using this wealth of information to recommend.

It's usually difficult to detect whether a question or issue is exception-related. So we apply several keyword-based filters for quickly traversal in the large amount of data. As shown in Table I, we select 8 popular programming languages with complete exception mechanism. Exception related keywords also differ according to different programming languages. We use these keywords and exception paradigms to detect whether a question or issue is exception-related.

We can see in the Table II. The dump of Stackoverflow contains 8,052,478 questions. By applying the filters with regular expression, we obtain exception-related 826,185 Q&As, 10.26% are exception-related in Stackoverflow. At the same time, GitHub dataset contains 4,878,132 issues, 3.13% are exception-related. In our point of view, 826,185 and 738,898 are large numbers, validating our intuition that there is a wealth of information in exception on Stackoverflow and GitHub.

C. RQ2: How suitable for Exception Solve?

We would like to know whether the quality of data from Stackoverflow and GitHub is able to help solving exception bugs. We consider that the quality of data handles well when the post gets enough replies and concerns. At the same time, the *Problem Solving Rate* of Stackoverflow questions and GitHub issues is another important quality indicator. For Stackoverflow, a question is solved by choosing an *best answer* while in GitHub only *closing* of an issue matters.

TABLE III
DISTRIBUTION FOR PROBLEM SOLVING RATE ACCORDING TO PROGRAMMING LANGUAGES

	Stackoverflow		GitHub	
Language	Total	Rate	Total	Rate
Java	183,008	22.2%	79,801	10.8%
Javascript	57,184	6.9%	208,369	28.2%
Ruby	8,636	1.1%	73,151	9.9%
C++	25,801	3.1%	50,245	6.8%
C#	336,205	40.7%	28,817	3.9%
Python	42,159	5.1%	124,874	16.9%
PHP	48,615	5.9%	89,407	12.1%
Objective-C	10,591	1.3%	32,512	4.4%
others	113,986	13.8%	51,723	7.0%

In Fig. 3(a) and Fig. 3(b), we can see that most questions in Stackoverflow get more than 3 answers and are successfully solved. GitHub issues seem getting more replies but a lower close rate. One possible reason is that, average reply post length and amount of information of Stackoverflow answers is larger than GitHub issues. However, both the Stackoverflow and GitHub do provide resources with good quality.

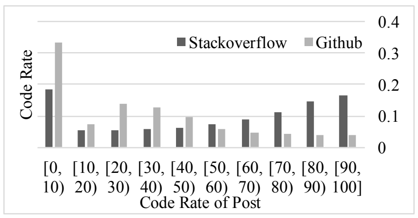
D. RQ3: Which are key features?

We show that Stackoverflow and GitHub are suitable for solving exception bugs. However, in the data of Stackoverflow and GitHub, there are lots of different information, such as title, tags, labels, questions body, comments and scores etc. Let us now discuss discovering the hidden rules and effects of different information. We try to make a study of following aspects:

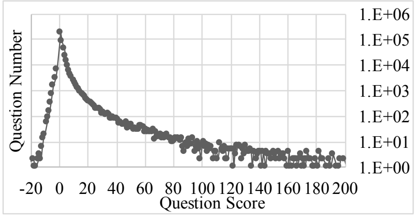
- Programming languages distribution
- Proportion of distribution for natural language and codes
- Score of questions and answers
- Number of tags or labels

Results in Table III shows that, for exception-related data, both GitHub and Stackoverflow have good coverage for selected 8 popular programming languages. There is only 13.8% and 7.0% other kinds languages and not involved. It is also worth noting that the method to recognize a programming language from Stackoverflow Q&A and GitHub issue will be explained in the following Section V-B about exception tree construction.

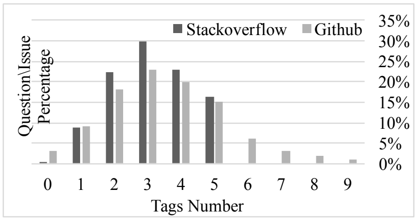
Most parts of Stackoverflow Q&As and GitHub issues are written in natural language like English. However, code fragments are also used to explain the exception or as a reference. We consider both codes fragments and stacktrace as program codes, and then investigate their percentage in post body. As shown in Fig. 4(a), most questions or issues have more than 10% code rate in the body. It's of great importance to take consideration the program features in our recommendation.



(a) Proportion of distribution for code rate of posts



(b) Score distribution of Stackoverflow questions



(c) Distribution of tags/labels number

Fig. 4. Figures for RQ3

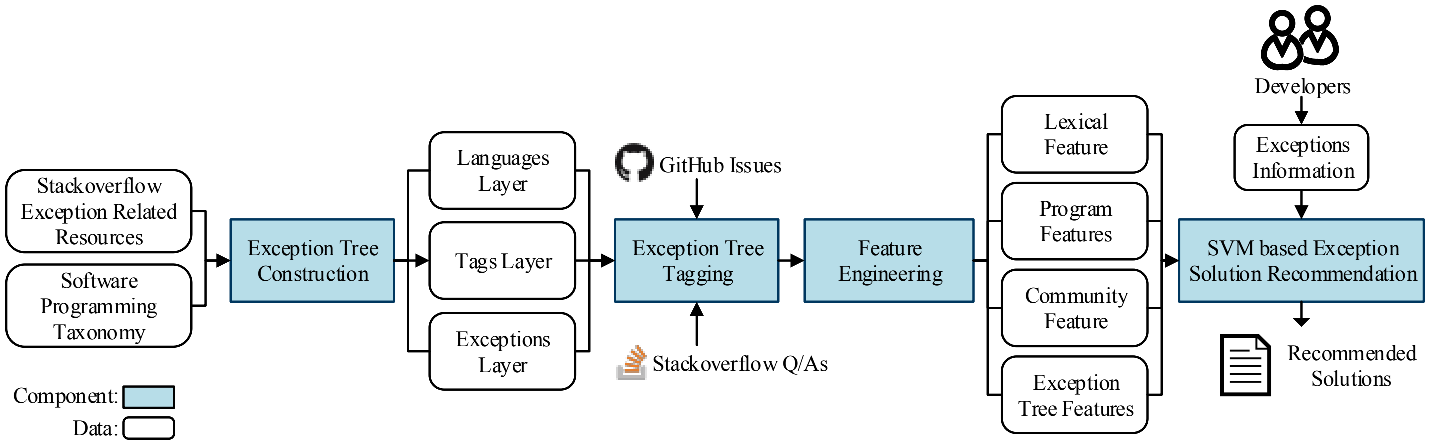


Fig. 5. Workflow of the recommendation approach

Fig. 4(b) and Fig. 4(c) study the question score and distribution of tags number. Only Stackoverflow questions have scores while GitHub issues not. Most questions take scores in the range from -10 to 50 which helps quality evaluation. Both for Stackoverflow and GitHub, there are obvious distribution laws of tags or labels number. Most resources are tagged round 3 labels. Enough number of tags or labels can help our approach to identify the linkage between exception contexts.

V. Approach

In this section, we will describe the recommendation approach in detail. There are four main components, namely Exception Tree Construction, Exception Tree Tagging, Feature Engineering, and SVM based Exception Solution Recommendation.

A. Approach Overview

We now provide a workflow to explain our whole process, and the interaction way between different components. For the recommendation process, as shown in Fig. 5, there are four main components, namely *Exception Tree Construction*, *Exception Tree Tagging*, *Feature Engineering*, and *SVM based Exception Solution Recommendation*. The input of *Exception Tree Construction* is exception-related questions or issues collected from Stackoverflow and GitHub. The method to identify an exception-related questions is mentioned in Section IV-B. *Exception Tree Construction* tries to build an exception tree with three different layers, including programming languages layer, community tags layer as well as exceptions layer. *Exception Tree Tagging* aims to build linkages between exception tree and Q&As/issues from Stackoverflow and GitHub.

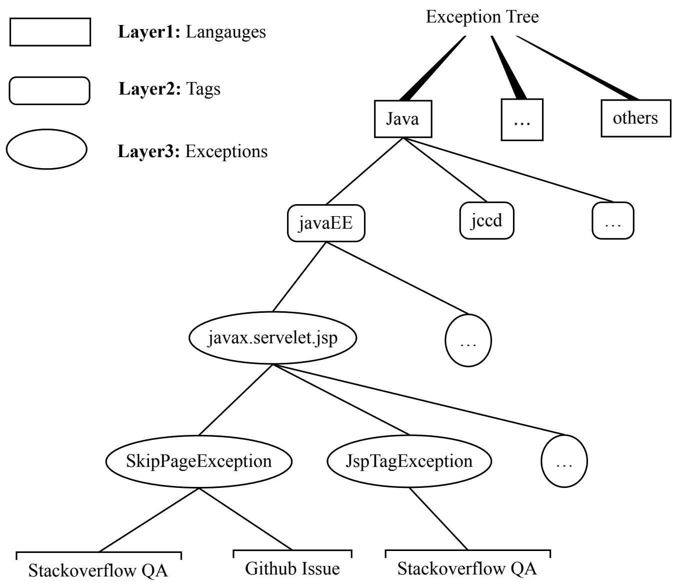


Fig. 6. Parts of exception tree

After tagging process, the crowd resources will be tagged to exception tree as leaf nodes. With the help of exception tree and tagging process, we can construct and aggregate exception-related resources from Stackoverflow and GitHub in a more effective way, thus will guarantee the scalability and efficiency of our approach. Next component is *Feature Engineering*. Based on the results of survey for Stackoverflow and GitHub in Section IV, we extract features like lexical feature, program features, community feature and the exception tree based features. Finally, the exception solution recommendation is considered as a binary classification problem. Based on the selected features, we train a supervised learning model with SVM and solve the recommendation problem.

B. Exception Tree Construction

As shown in the Fig 6, there are three layers of the exception tree.

1) Layer1 Languages: In the Exception Tree, nodes of programming languages layer connect with the root node. Based on the survey in Section IV-D, we selected 8 widely used programming languages in Stackoverflow and GitHub as our language layer nodes. Every language refers to one language node. Besides them, we also add an ***others*** node, where all the unrecognized questions or issues will link to.

2) Layer2 Tags: According to the survey in Section IV-D, almost every question in Stackoverflow and issue in GitHub will be labeled with several keywords named tags or labels. Through these artificial tags, resources can be quickly classified and indexed. However, it's difficult to analysis associations between the loose tags. We use our previous work, a taxonomy about software programming constructed from Stackoverflow [30] The taxonomy captures hierarchical semantic structures of tags in Stackoverflow. Based on the taxonomy, we try to construct the tag layer subtree belonging to every language layer node. The key parts is the way to identify children nodes from the taxonomy. As shown in Algorithm 1, we build a recursive function with depth first traversal.

In the algorithm, we assume that there is a set of n tags in the taxonomy $T = \{t_1, \dots, t_n\}$. The hyponym information is summarized in an $n \times n$ matrix H, where $H_{ij} = 1$ means t_i is a hyponym of t_j, $Hij = 0$ otherwise. Nodes in Exception Tree are donated by node set $N_T = \{n_1, \dots, n_n\}$, and edges set E_T. $e_{ij} \in E_T$ means n_i is the children node of n_j. We build the tags layer of exception tree from some selected tags in the taxonomy, donated by T^*. And the layer1 and layer 2 nodes in the tree are donated by N^*. We can eailsy match tags and nodes by means of the same keywords or synonyms[4]. We define the matching function $\zeta : T^* \mapsto N_T^*$ and $\eta : N_T^* \mapsto T^*$.

3) Layer3 Exceptions: The third layer of exception tree consists of exceptions extracted from exception-related keywords Stackoverflow Q&As and GitHub issues. As shown in Table I, based on the different exception paradigms for different languages, we construct regular expressions and extract the exception keywords from selected question or issue body .

[4]The synonyms relations are in http://stackoverflow.com/tags/synonyms

Algorithm 1 Depth First Traversal for Identity Layer 2 Nodes

```
Input:
    Exception Tree Node $n_j$
1: $t_k = \eta(n_j)$
2: Let hyponym tags of $t_k$ donated by $T_h$
3: $T_h = \{t_i \in T | H_{ik} = 1\}$
4: while $t \in T_h$ do
5:     $n_j = \zeta(t)$
6:     Add $n_j$ to $N_T$
7:     Add $e_{ji}$ to $E$
8:     $Alghorithm\ 1(n_j)$
9: end while
```

Extracted exceptions will be added to exception tree as layer 3 nodes, and link to the layer 2 nodes which the tags of selected question or issue match with function ζ. For exception with the format like *java.lang.NullPointerException*, we extract three keywords *java*, *lang* and *NullPointerException*. Every keyword would be added as one layer 3 nodes in the exception tree as well as their relationship.

We donate function τ which is used to generate layer 3 exception nodes from exception keywords. Function ξ is used to extract the nodes relationships among layer 3 nodes or relationship between layer 2 and layer 3 nodes. Algorithm 2 shows the construction process.

However, one question or issue may extract more than one exception keywords which have same prefix. For example, exceptions *jsp.JspTagException* and *jsp.SkipPageException* get same prefix *jsp*. For this situation, we let corresponding exception tree nodes *JspTagException* and *SkipPageException* share same parent node *jsp* through a combination process.

C. Exception Tree Tagging

After the construction of exception tree, it's easy to link all exception-related questions and issues to exception tree nodes. Based on the exception keywords extracted from the body, we can built the linkage between resources from Stackoverflow & GitHub to exception tree. It is also worth noting that we can achieve the link process during the construction of exception tree layer 3. As shown in the last line of Algorithm 2, for every input resource r, we can link r with the exception nodes just added in exception tree $E_{EP_r^*}$.

However, there are two problems. (1) One question or issue may link to more than one exception tree nodes. We will only link the question or issue to the common subnodes of all related exception tree nodes. For exception tree nodes like *java* and *javaEE*, we only link to *javaEE*, because *javaEE* is one of the child nodes of *java*. (2) There are questions or issues which cannot link to corresponding layer 2 nodes. For the exceptions nodes extracted of these kind of questions or issues, we link them to the ***others*** node in the layer 1.

D. Feature Engineering

We first extract four kinds of features, then apply a machine learning algorithm, SVM, to train a model from a ground-truth data set based on these features. The purpose of feature

Algorithm 2 Algorithm for Identity Layer 3 Nodes

```
Input:
    A Stackoverflow question or GitHub issue $r$
    Exception tree nodes $N_T$ and edges $E$
Output:
    Updated tree nodes $N_T$ and edges $E$
 1: Get tags $T_r$ for $r$
 2: Get exceptions $EP_r$ for $r$
 3: Declare set of nodes to be linked $N_r^* \leftarrow \emptyset$
 4: while $t \in T_r$ do
 5:     $n = \zeta(t)$
 6:     if $n$ is exists then
 7:         $N_r^* \leftarrow N_r^* + n$
 8:     end if
 9: end while
10: if $N_r^*$ is $empty$ then
11:     $N_r^* \leftarrow N_r^* + n_{others}$
12: end if
13: while $n \in N_r^*$ do
14:     if $n$ is one of the parent of $N_r^* - n$ then
15:         $N_r^* \leftarrow N_r^* - n$
16:     end if
17: end while
18: Get combined exceptions $EP_r^*$
19: $N_{EP_r^*} = \tau(EP_r^*)$
20: $E_{EP_r^*} = \xi(N_{EP_r^*}) + \xi(N_{EP_r^*}, N_r^*)$
21: $N_T \leftarrow N_T \cup N_r^* \cup N_{EP_r^*}$
22: $E \leftarrow E \cup E_{EP_r^*}$
23: Link $r$ with $E_{EP_r^*}$
```

engineering is to quantitatively characterize the similarities or relatedness between r and c. We define six features to characterize the relations. The details of these features are as follows:

Lexical Feature(LF)

1) Textual Similarity: We measure the textual similarity between resource r from Stackoverflow and GitHub with exception context c. All possible parts of r and c, like title, tags or body etc., will be used to calculate *tf-idf* and *cosine* similarity. $\vec{r}$ and $\vec{c}$ refers to the corresponding term vector of r and c:

$$S_{TS} = \frac{\vec{r} \cdot \vec{c}}{\|\vec{r}\| \cdot \|\vec{c}\|} \tag{1}$$

Program Features(PF)

2) Stacktrace and Code Similarity: During exceptions, the IDE or runtime generally issues notifications from a fixed set of formatted messages. As a result, there is a great chance that exception-related questions or issues contain stacktrace or codes fragments about an exception. Study shows that simhash is good at measuring the format resouces [31]. We select the minimum value from the hamming distance between stacktrace and code fragments as the similarity:

$$S_{SS} = min\left(ham\left(s_{CS_r}, s_{CS_c}\right), ham\left(s_{ST_r}, s_{ST_c}\right)\right) \tag{2}$$

s_{CS_r} and s_{CS_c} are the code fragments extracted from community resources and exception context. s_{ST_r} and s_{ST_c} are the corresponding exception stacktraces.

3) API Similarity: Generally, it's difficulty to make a depth analysis of code similarity without program recognition technology. Methods like call graph or grammar tree analysis are very time-consuming and complex. However, the higher usage of the same types in both community resources and exception context also shows more potential usefulness of the resources. With the help of regular expression, we extract the APIs types P_r, P_c and calculate the Jaccard Similiarty:

$$S_{AS} = \frac{|P_r \cap P_c|}{|P_r \cup p_c|} \tag{3}$$

Community Feature(CF)

4) Q&A Score: As shown in the previous study, the score of the question also represents quality of resources. We measure the resource quality for resources from Stackoverflow:

$$S_{QS} = \alpha QuestionScore + \beta \frac{\sum AnswerScore}{|Answers|} \tag{4}$$

Generally, the scores from the question and the answers both reflect the resource quality. We use α and β as parameters to distinguish between questions and answers scores proportion.

Exception Tree Features(ETF)

5) Average Exception Tree Path Score: In the graph, two high correlation points generally have a short path and the path nodes between two points also have a smaller degrees. Let the path from node n_j to n_k in exception tree as $P = \{n_j, \cdots, n_k\}$. We define the weight of the path:

$$\Gamma(P) = \sigma(n_k) \prod_{i=j}^{k-1} \frac{\sigma(n_i)}{|degree(n_i)|^{\sigma}} \tag{5}$$

The weight for exception node n:

$$\sigma(n) = \begin{cases} \alpha, n \in N_{layer1} \\ \beta, n \in N_{layer2} \\ \sigma, n \in N_{layer3} \end{cases} \tag{6}$$

α, β and σ are parameters describing influences of three different layer nodes. N_r is the set of exception tree nodes which resource r links to, and N_c is the set of nodes existing in the context c. So we get average exception tree path score between N_r and N_c:

$$S_{TPS} = \frac{\sum_{n_i \in Nr} \sum_{n_j \in Nc} \Gamma(P_{ij})}{\sum_{n_i \in Nr} \sum_{n_j \in Nc} (1)} \tag{7}$$

6) Average Exception Tree Altitude difference: The height difference of the nodes may also have an effect on the result. Two nodes with a smaller height difference usually share more in common. We measure Average Exception Altitude difference between N_r and N_c:

$$S_{TAD} = \frac{\sum_{n_i \in Nr} \sum_{n_j \in Nc} HeightDiff\left(n_i, n_j\right)}{\sum_{n_i \in Nr} \sum_{n_j \in Nc} (1)} \quad (8)$$

Function $HeightDiff$ is used to calculate the altitude difference between two tree nodes.

E. SVM based Exception Solution Recommendation

We treat the recommendation problem as a binary-class classification problem. And we use SVM algorithm to train the binary classifier, which is known as one of the best single classifiers [32]. Generally, classification is kind of supervised learning process, which requires labeled data for training. The performance of classification process highly relies on the quality and distribution of labeled training data.

However, it's usually difficulty to identify whether a recommended result is truly the right one without strict standards. Human checking of training data is also very time-consuming. Luckily, we found an interesting phenomenon in our data sets. Some successfully closed GitHub issues contains links to specific solved Stackoverflow questions. We consider that if one closed GitHub issue or solved Stackoverflow question have at least one link pointing to the other, then they are exactly the same solution for one exception context.

Therefore, we can generate the positive training data in this way. As in our statistics, while there are small amount of closed issues containing links to solved Stackoverflow questions, it do provide enough amount of data for training and testing. Finally, we totally select 3,000 Stackoverflow question and GitHub issue pairs as our positive training set. For negative training data, we randomly generated 5,000 data pairs with lexical similarity less than 0.05. Finally, among the generated data pairs, we manually selected 2,000 data pairs which are not related for one exception context. These set are not only used to boost the learning process but also treated as ground truth to evaluate our approach in Section VI.

VI. Experiments

In this section, we first present our experimental settings and then analyze the experiment results. We totally conduct two experiments to answer following two questions: (1) How much does three kinds of features contribute to the model? (2) How does our cross-domain method perform compared to existing methods?

A. Experiment Setup

We select two famous software communities, Stackoverflow and GitHub, to conduct the experiments. As shown in Section IV-B and Table II, we filter out the exception-related resources from origin data sets. In total, there are 8,052,478 Stackoverflow questions, 23,606,974 GitHub issues. Among these resources, 826,185 Stackoverflow questions and 738,898 GitHub issues are exception-related. We totally select 5,000 Stackoverflow question and GitHub issue pairs as our experiments set. Among them, 3,000 are positive where the Stackoverflow question and GitHub issue in one pair share the same solution for one exception context. Other 2,000 pairs are negative.

B. Evaluation Metrics

Given that our proposed approach is aligned with the research areas of recommendation system and information retrieval, we use flowing list of performance metrics from those areas.

1) Precision: Precision denotes the fraction of retrieved results that are relevant to the query. We use *Precision* to measure the basic contribution and importance of selected features.

2) Recall: Recall is the fraction of the results that are relevant to the query that are successfully retrieved. In our experiments, we consider *recall* as the percentage of the exception contexts for which the solutions has been correctly recommended.

3) F-Score: Often, there is an inverse relationship between precision and recall. We use *F-Score* to measure the total influence of precision and recall.

4) TopN Hits: In the recommendation process, we call a hit when the correct solution occurs in the top-n recommendation results. *TopN Hits* is the total number of hits for all the queries in the experiment query set.

5) Average Hit Rank: Hit Rank is the rank of hit for an recommendation query. *Average Hit Rank* is a statistical measure that averages the *Hit Rank* for all the queries.

C. Feature Contribution Analysis

We totally use four kinds of features to train the model, including lexical feature, abbreviated by *LF*, program features, abbreviated by *PF*, community feature, abbreviated by *CF*, and exception tree features, abbreviated by ETF. The first experiment will show how much the four kinds of different features donate to classification results. We use the labeled data generated in Section V-E as the ground truth and selected 5,000 data pairs to validate our method. Among the pairs, 4,000 pairs are selected as training data and the rest 1,000 as testing data. As shown in Fig. 7, we train eight SVM classifiers with different features or feature combinations by SVM algorithms and show the results. A 5-fold cross validation is applied to train the classifiers. Precision, recall, and F-score are used for effectiveness study.

Fig. 7 illustrates how each feature or feature combination affects the model. According to the results, model trained with all features performs great. That is to say, all these features are useful in predicting new subsumption relations. When only one kind of feature is concerned, the model trained with exception tree features reaches the best, with the precision of 56%, better than those trained with lexical feature or with program features. When two kinds of features are considered, model

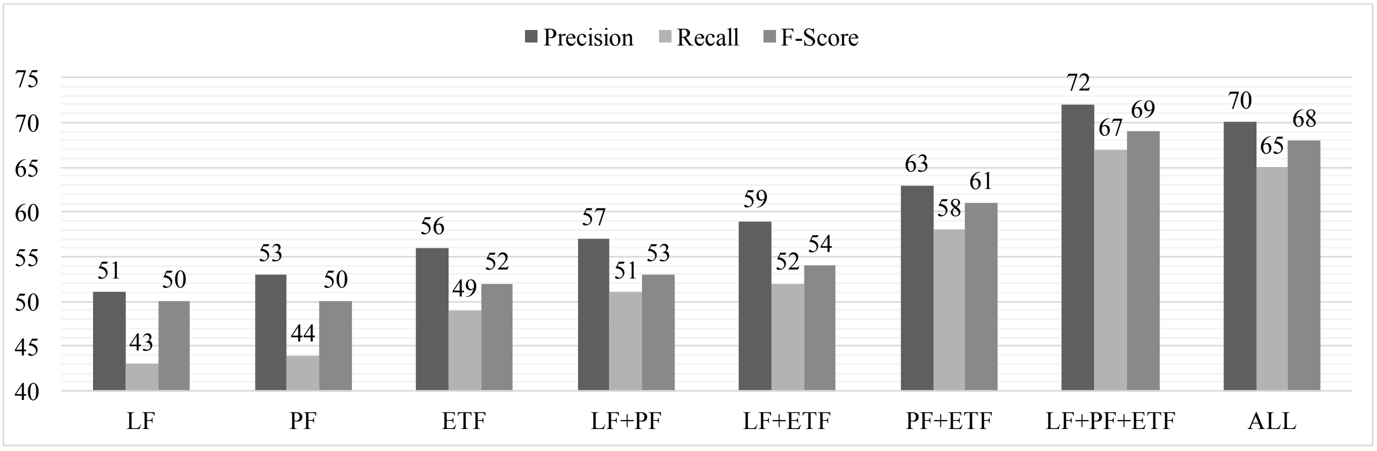


Fig. 7. Experiment results for different features

trained with exception tree features (*LF+ETF* or *PF+EFT*) is also better than that trained without exception tree features (*LF+PF*).

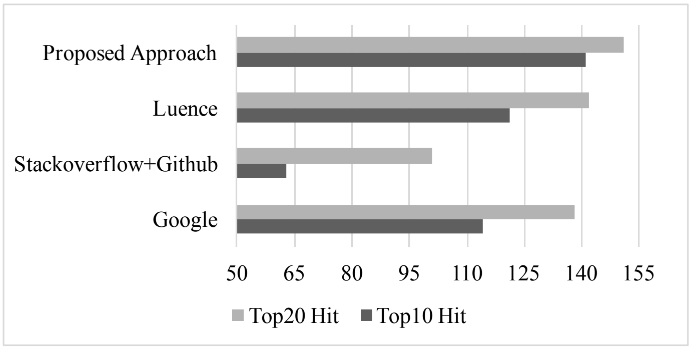
From the results, compared with total text similarity calculated in lexical feature, program features performs better. This is because an exception context contains more program related information which describes exception more accurate like code fragments or stacktrace than natural language description. It's also easy to find that exception tree features are important, and lexical and program features also have positive contribution to final model. We consider this is mainly because the information structured in the exception tree is accurate for and good at describing and locating an exception. Programming languages, tags, and exceptions play a more important role in the recommendation than general text description and programming codes.

However, it's remarkable that community feature seems not have an improvement for the results. This is because community feature including Q&A score concerns more about quality of the resources rather than the of degree of correlation with exception context.

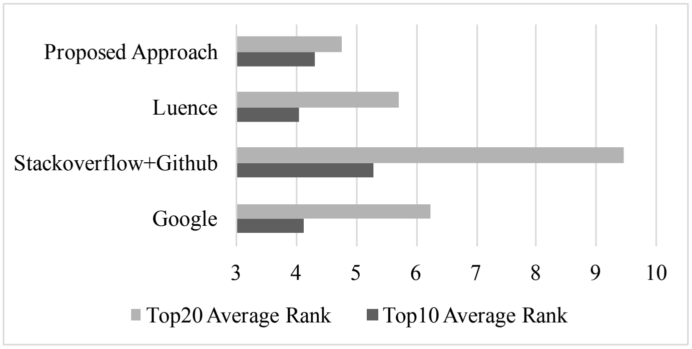
D. Comparison with Other Methods

In this experiment, we compare our method with other three state-of-the-art methods, the Google-based general searching engine method, the internal searching method in Stackoverflow and GitHub and the luence-based retrieval method, to solve the recommendation problem.

The experiment data are selected from pairs of Stackoverflow question and GitHub issue with related links in Section V-E. Among them, we manually choose 200 pairs for the experiment. Every pair refers to the solution for one exception-related problem. For every pair, there are two sides including Stackoverflow question or GitHub issue. We are able to select one side from the pair as exception context for experiment input, another side as corresponding correct answer. To be fair, pairs with each side takes a half. In Google-based general searching engine method and internal searching method in Stackoverflow and GitHub, we generate queries based on the



(a) Total hits



(b) Average rank

Fig. 8. Experiments results for comparison with other methods

following pattern. Every query is started with *ExceptionName* and follows keywords with high frequency keywords. For Google-based method, we take specific searching techniques[5] to limit the results to domain of Stackoverflow or GitHub.

Fig. 8 illustrates the results of comparison. We take care of the hit number and average rank for top 10 and top 20 recommendation results. In the results, our approach achieve better than other three methods. In top 10 results, we achieve

[5]Query like ***site:stackoverflow.com keywords*** could limit Google searching results in Stackoverflow domain

high hit rate 65.5%, while others are below 60%. In top 20 results, while other three methods perform better than top 5, they are still cannot catch up our approach in hit rate and average result rank. It's interesting that, the results from internal searching of Stackoverflow and GitHub performs lower than Google. We think this may due to the poor searching and matching mechanism.

VII. Conclusion And Future Work

In this work, we proposed a cross-domain exception solution recommendation approach. Two famous communities, Stackoverflow and GitHub, are selected to validate the feasibility of our method. Differ to the previous methods, based on the empirical study results, we propose and construct an exception tree to link cross-domain resources. Exception tree based features are considered in our SVM training model to help improve the results. The experiments show the high quality of our approach.

As for future work, we will try to extract much detail information about exception into our exception tree, such as versions of programming languages or external libraries. Moreover, it would be interesting to enrich our approach with resources from some other online communities or domains. User ability and interest analysis in social network could also help our recommendation.

References

[1] B. Cabral and P. Marques, "Exception handling: A field study in java and. net," in *ECOOP 2007–Object-Oriented Programming*, pp. 151–175, Springer, 2007.

[2] M. V. Mäntylä, J. Vanhanen, and C. Lassenius, "Bad smells-humans as code critics," in *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*, pp. 399–408, IEEE, 2004.

[3] M. Umarji, S. E. Sim, and C. Lopes, "Archetypal internet-scale source code searching," in *Open source development, communities and quality*, pp. 257–263, Springer, 2008.

[4] A. Bacchelli, T. Dal Sasso, M. D'Ambros, and M. Lanza, "Content classification of development emails," in *Proceedings of the 34th International Conference on Software Engineering*, pp. 375–385, IEEE Press, 2012.

[5] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," in *Proceedings of the 28th international conference on Software engineering*, pp. 361–370, ACM, 2006.

[6] M. Goldman and R. C. Miller, "Codetrail: Connecting source code and web resources," *Journal of Visual Languages & Computing*, vol. 20, no. 4, pp. 223–235, 2009.

[7] M. M. Rahman and C. K. Roy, "Recommending main & relevant content from web pages about programming errors and exceptions,"

[8] D. Gibson, K. Punera, and A. Tomkins, "The volume and evolution of web page templates," in *Special interest tracks and posters of the 14th international conference on World Wide Web*, pp. 830–839, ACM, 2005.

[9] B. Ashok, J. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V. Vangala, "Debugadvisor: a recommender system for debugging," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 373–382, ACM, 2009.

[10] D. Čubranic and G. C. Murphy, "Hipikat: Recommending pertinent software development artifacts," in *Software Engineering, 2003. Proceedings. 25th International Conference on*, pp. 408–418, IEEE, 2003.

[11] M. Monperrus and A. Maia, "Debugging with the crowd: A debug recommendation system based on stackoverflow," 2014.

[12] J. Cordeiro, B. Antunes, and P. Gomes, "Context-based recommendation to support problem solving in software development," in *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on*, pp. 85–89, IEEE, 2012.

[13] M. M. Rahman, S. Yeasmin, and C. K. Roy, "An ide-based context-aware meta search engine," in *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pp. 467–471, IEEE, 2013.

[14] M. M. Rahman and C. K. Roy, "Surfclipse: Context-aware meta-search in the ide," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pp. 617–620, IEEE, 2014.

[15] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Mining stackoverflow to turn the ide into a self-confident programming prompter," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 102–111, ACM, 2014.

[16] B. Vasilescu, A. Serebrenik, P. Devanbu, and V. Filkov, "How social q&a sites are changing knowledge sharing in open source software communities," in *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, pp. 342–354, ACM, 2014.

[17] C. Gomez, B. Cleary, and L. Singer, "A study of innovation diffusion through link sharing on stack overflow," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, pp. 81–84, IEEE, 2013.

[18] M. Allamanis and C. Sutton, "Why, when, and what: analyzing stack overflow questions by topic, type, and code," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 53–56, IEEE Press, 2013.

[19] D. Correa and A. Sureka, "Integrating issue tracking systems with community-based question and answering websites," in *Software Engineering Conference (ASWEC), 2013 22nd Australian*, pp. 88–96, IEEE, 2013.

[20] E. C. Campos, L. B. de Souza, and M. d. A. Maia, "Nuggets miner: Assisting developers by harnessing the stackoverflow crowd knowledge and the github traceability,"

[21] C. Denzil, "Content quality in web 2 0 services analysis, detection, systems and enhancement," 2015.

[22] A. Bacchelli, L. Ponzanelli, and M. Lanza, "Harnessing stack overflow for the ide," in *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*, pp. 26–30, IEEE Press, 2012.

[23] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: integrating web search into the development environment," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 513–522, ACM, 2010.

[24] D. Poshyvanyk, M. Petrenko, and A. Marcus, "Integrating cots search engines into eclipse: Google desktop case study," in *Proceedings of the Second International Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques*, p. 6, IEEE Computer Society, 2007.

[25] M. M. Rahman, S. Yeasmin, and C. K. Roy, "Towards a context-aware ide-based meta search engine for recommendation about programming errors and exceptions," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pp. 194–203, IEEE, 2014.

[26] N. Sawadsky, G. C. Murphy, and R. Jiresal, "Reverb: Recommending code-related web pages," in *Proceedings of the 2013 International Conference on Software Engineering*, pp. 812–821, IEEE Press, 2013.

[27] B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer, "What would other programmers do: suggesting solutions to error messages," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1019–1028, ACM, 2010.

[28] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Seahawk: Stack overflow in the ide," in *Proceedings of the 2013 International Conference on Software Engineering*, pp. 1295–1298, IEEE Press, 2013.

[29] G. Gousios, "The ghtorrent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, (Piscataway, NJ, USA), pp. 233–236, IEEE Press, 2013.

[30] Anonymous, "Anonymized for double-blind review,"

[31] M. S. Uddin, C. K. Roy, K. Schneider, A. Hindle, *et al.*, "On the effectiveness of simhash for detecting near-miss clones in large scale software systems," in *Reverse Engineering (WCRE), 2011 18th Working Conference on*, pp. 13–22, IEEE, 2011.

[32] T. Joachims, *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998.