SurfClipse: Context-Aware Meta Search in the IDE

Mohammad Masudur Rahman Chanchal K. Roy
Department of Computer Science, University of Saskatchewan, Canada
{masud.rahman, chanchal.roy}@usask.ca

***Abstract*—Despite various debugging supports of the existing IDEs for programming errors and exceptions, software developers often look at web for working solutions or any up-to-date information. Traditional web search does not consider the context of the problems that they search solutions for, and thus it often does not help much in problem solving. In this paper, we propose a context-aware meta search tool, *SurfClipse*, that analyzes an encountered exception and its context in the IDE, and recommends not only suitable search queries but also relevant web pages for the exception (and its context). The tool collects results from three popular search engines and a programming Q & A site against the exception in the IDE, refines the results for relevance against the context of the exception, and then ranks them before recommendation. It provides two working modes–*interactive* and *proactive* to meet the versatile needs of the developers, and one can browse the result pages using a customized embedded browser provided by the tool.**

Tool page: www.usask.ca/∼masud.rahman/surfclipse

***Index Terms*—Context-aware web search; meta search; context-relevance; errors and exceptions**

I. Introduction

Although existing IDEs (e.g., Eclipse, Net Beans, Visual Studio) are equipped with various debugging supports for programming errrors and exceptions, software developers often look into the web for working solutions and for any up-to-date information. According to the study of Brandt et al. [2], developers spend about 19% of their development time in web surfing. Traditional web search does not consider the *context* (i.e., surroundings, circumferences) of the programming problems, and involves the developers in *trial and error* based query selection and search, which are time-consuming and counter-productive. However, tool support through *relevant query suggestion* and *context-aware ranking* of search results can greatly benefit them in this regard, and this paper focuses on these two research problems.

There exist several studies [3, 5] that attempt to address similar research problems. Cordeiro et al. [3] propose an IDE-based recommendation system that recommends relevant StackOverflow posts for programming errors and exceptions. They extract a number of question and answer posts from StackOverflow data dump, and suggest those question posts that contain stack traces similar to that of an encountered exception in the IDE. Ponzanelli et al. [5] propose *Seahawk*, an Eclipse plugin, that analyzes the *context code* (i.e., code under development) of the current programming task and recommends relevant StackOverflow posts in the IDE. Although these approaches have their inherent strengths, they also suffer from several limitations. First, they consider only one source– StackOverflow Q & A site, for information, and thus the search scope is limited. Second, the developed corpus is static in nature and also subjected to the availability of the data dump. Third, they only consider either stack trace or source code under development as the *context* of a programming problem, which is partial (i.e., incomplete) and often does not help much. For example, the approach by Cordeiro et al. does not consider the code segment that triggers an exception and thus recommends solutions which might be non-applicable or even irrelevant to the code of interest given that the same exception could be triggered from different code context. Similarly, *Seahawk* [5] cannot recommend properly for the programming tasks associated with errors and exceptions as it does not analyze the stack traces reported by the IDE.

In this paper, we propose a *context-aware meta search* tool, *SurfClipse*, for the encountered programming errors and exceptions in the IDE that not only provides a complete web search solution but also addresses the concerns identified with the existing approaches [3, 5]. We package the solution as an Eclipse plugin [1], which collects search results from a remotely hosted web service [1] and displays them within the IDE. Technically, the tool works both as a search query recommender and a meta search engine. Once a developer encounters an exception in the IDE, the tool captures and analyzes the technical details (i.e., stack trace and context code) of the exception, and recommends a list of relevant search queries. The developer selects a query from the list, and the tool collects results from three reliable search engines– Google, Bing and Yahoo and a popular programming Q & A site, StackOverflow, against the query. It then analyzes, refines and ranks the results against not only the exception but also its context in the IDE before recommendation. To summarize, our tool provides the following features to support developers in problem solving:

(a) the proposed tool captures technical details of an encountered exception, and recommends a ranked list of suitable search queries that can be used both with *SurfClipse* and traditional web search,
(b) ranks the result pages adopting a *context-aware* approach so that the pages are relevant not only to the encountered exception but also to its context in the IDE,
(c) provides two working modes– *interactive* and *proactive*, to meet the versatile needs of different developers and different task scenarios.
(d) to ensure a broader search space, exploits the search and ranking algorithms of three popular search engines and a programming Q & A site through their API endpoints,
(e) exploits a dynamic source (e.g., API endpoints) compared to static source (e.g., data dump) by existing approaches [3, 5] for StackOverflow data, which makes the most

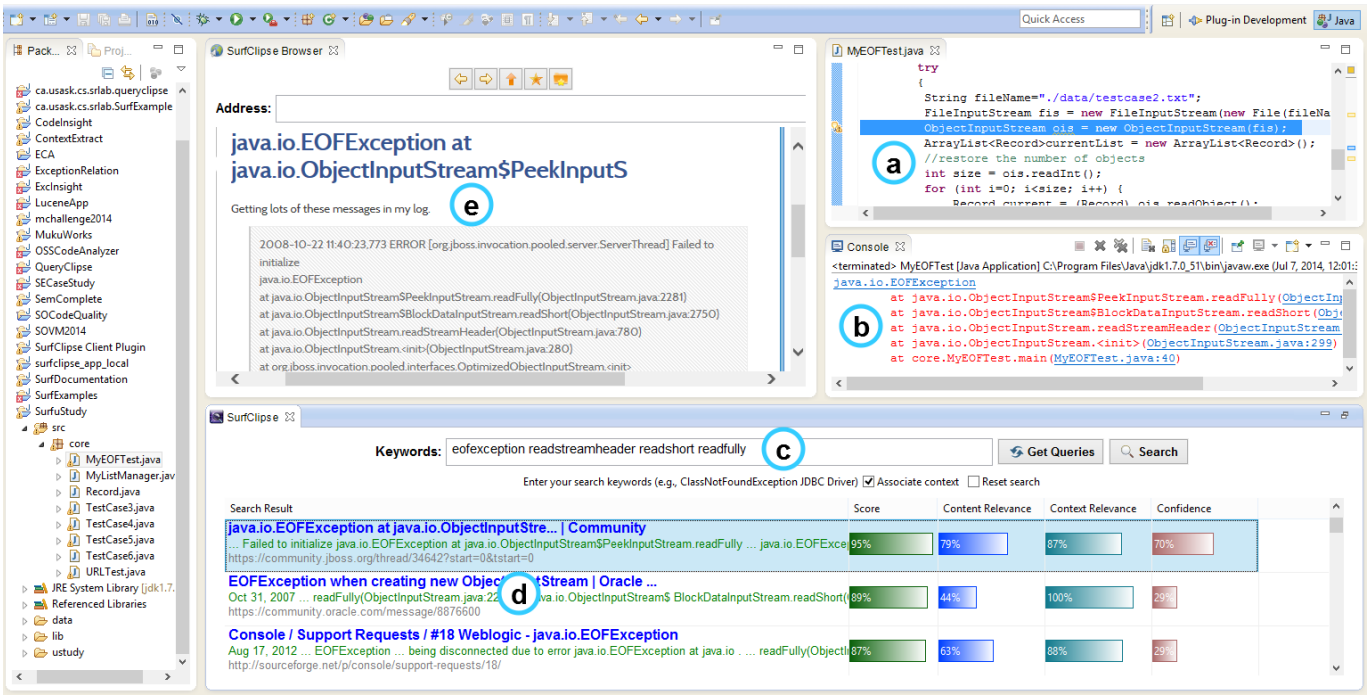

Fig. 1. SurfClipse User Interface

recent and relevant StackOverflow posts available for recommendation.

While this paper focuses on the tool aspect of our approach, we refer the readers to the original paper [6] for further details.

II. SurfClipse

Fig. 1 shows the user interface of *SurfClipse*, where we contribute in (c) search panel, (d) result panel and (e) browser panel of the interface. This section discusses different technical features provided by our tool.

(1) Working Modes: *SurfClipse* works in two modes–*interactive* and *proactive*. In case of *interactive* mode, a user (e.g., a developer) generally initiates the search by selecting an exception from *Console View* in the IDE or a search query (representing the exception and its context) from the recommendation list, whereas the tool itself initiates the search process in case of *proactive* mode. Once the tool is properly installed, it provides several main menu and context-menu based command options, which can be used to initiate the tool environment or to change the working modes.

(2) Automated Supports with Search Queries: Both the context code (e.g., Fig. 1-(a)) that triggers an exception, and the stack trace (e.g., Fig. 1-(b)) reported by the IDE contain overwhelming information, and developers often face difficulties in choosing a suitable search query from such information. *SurfClipse* provides automated supports in this regard, and helps them choosing queries from two options–*recommendation list* and *stack trace graph*.

Query Recommendation: In this case, the tool analyzes both stack trace and context code of the exception, and recommends a ranked list of five suitable search queries for the exception (Fig. 2).

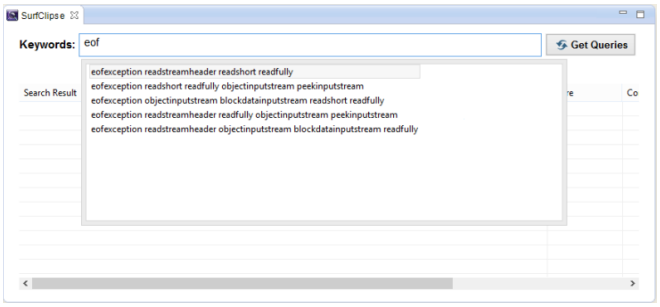

Fig. 2. Search Query Recommendation (Interactive Mode)

Stack Trace Token Graph: In this case, the tool extracts important tokens (e.g., class name, method name) from the stack trace (e.g., Fig. 1-(b)), and develops a *token graph* (e.g., Fig. 3). In the graph, tokens are represented as nodes, and the implied relationships (e.g., class to method static relations, method call sequences) among the tokens are represented as connecting edges. Thus the graph visualizes the relative importance of different tokens in terms of their connectivity, and the developers can get important hints about the useful query tokens in the trace information.

(3) Context-Aware Web Search: *SurfClipse* provides three options to conduct web search within the IDE–*proactive search*, *context-menu based search (i.e., interactive mode)* and *keyword search (i.e., interactive mode)*.

Proactive Search: When *SurfClipse* is set to *proactive mode*, it automatically detects an encountered exception in the IDE. In this mode, the tool constantly monitors the *Console View* for a stack trace using regular expressions. Upon detection, it collects other details– an auto-generated query and the context code of the exception, and initiates the search.

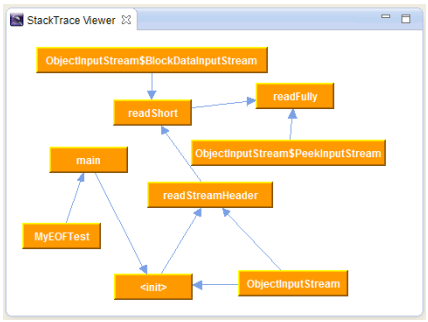

Fig. 3. Stack Trace Token Graph

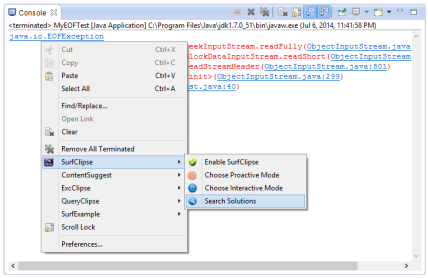

Fig. 4. Context-menu Based Search (Interactive Mode)

Context-Menu Based Search: The tool provides a context-menu based search option, and a developer can literally select any phrase in the IDE (from *Editor View* and *Console View*), and perform web search. More importantly, she can select the exception in the *Console View*, and conduct the search (e.g., Fig. 4). Once initiated, the tool captures necessary details from the IDE, performs the search, and collects the results.

Keyword Search: Given that a user might be interested in refining the auto-generated search query or in a more traditional way of search, the tool provides a keyword-based search feature (e.g., Fig. 1-(c)). The search is complemented with search query suggestion through auto-completion. The user can also configure whether the search should be a *keyword matching only* (i.e., does not refine the results against the *exception context*) or a *context-aware* one through *Associate context* option (e.g., Fig. 1-(c)).

(4) Search Results & Browsing: Once a search request is made for an exception, the tool collects results in a non-intrusive way (i.e., without freezing the IDE), and displays them within the IDE (e.g., Fig. 1-(d)). It also shows the metric details–*content relevance, context relevance* and *search engine confidence* of each result page through visualization, which helps one to choose the right (most relevant) page for browsing. One can select a page from the result panel, and browse it easily using a customized browser widget (e.g., Fig. 1-(e)) provided by the tool.

III. A Use Case Scenario

By means of a use case scenario, we attempt to explain how *SurfClipse* can help a software developer in solving problems related to programming errors and exceptions within the IDE.

Suppose a developer, Alice, is performing unit testing on a piece of code which searches for a specific item in the list and deletes the item when found (Listing 1). During testing,

```
20  List<String> myList = new ArrayList<String>();
21     String[] items={"apple","orange","banana",
22                     "mango","grape"};
23     for(String item:items){
24       myList.add(item);  }
25         //deleting a particular item from the list
26         Iterator<String> it = myList.iterator();
27         while(it.hasNext()){
28             String value = it.next();
29              if(value.equals("banana"))
30               myList.remove(value);  }
```

Listing 1. Working Code Example

```
1   Exception in thread "main" java.util.
        ConcurrentModificationException
2   at java.util.ArrayList$Itr.checkForComodification(
        Unknown Source)
3   at java.util.ArrayList$Itr.next(Unknown Source)
4   at core.MyListManager.main(MyListManager.java:28)
```

Listing 2. Stack trace of ConcurrentModificationException

she encounters a *ConcurrentModification* exception, and gets a stack trace (Listing 2) reported by the IDE. She is not pretty familiar with that exception; however, from the stack trace, she identifies the source line (i.e., Line 28, highlighted) triggering the exception, and she also assumes that the problem is something related to *ArrayList*.

She attempts to solve the issue, and at some point, she decides to perform web search for a solution or more helpful information. Now she faces several challenges–(1) How to develop an effective and appropriate search query for the current exception? (2) How to analyze and include the *context* of the encountered exception during search? and finally (3) How to choose a working solution for the current exception from the search results? The traditional web search does not help her to overcome those challenges; or if it does somehow, those are often not sufficient enough for problem solving.

Now let us assume that Alice has installed *SurfClipse* in her IDE, and she encounters the same exception during testing. Our tool provides her with several options such as *proactive search, context-menu based search* and *keyword search*. Suppose, she is interested in the third option– keyword search. During this search, she can easily choose a search query from the recommended list, and the tool returns the top 30 result pages with the rationale (i.e., metric details) behind their selection in the ranked list. Thus, in order to overcome the challenges that Alice faces with traditional search, our tool (1) helps her to develop a search query by either automatic suggestion (e.g., Fig. 2) or stack trace graph visualization (e.g., Fig. 3), (2) automatically captures the exception and its context (i.e., stack trace and context code) from the IDE, and associates them with the search, (3) returns results which are relevant not only to the exception but also to its context, and explains the rationale behind the selection of each result through metric details visualization (e.g., Fig. 1-(d)). Furthermore, the results are collected from four reliable and popular sources.

IV. Working Methodology

Fig. 5 shows the schematic diagram of the proposed tool. This section discusses the internal structures and working

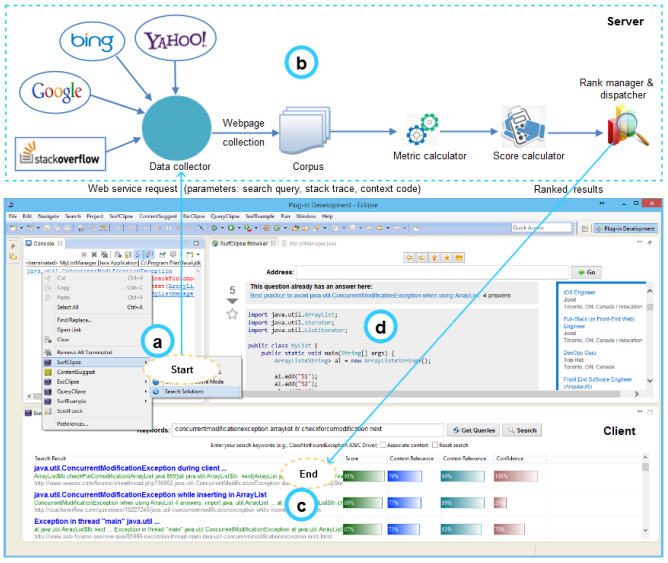

Fig. 5. Schematic Diagram of *SurfClipse*

methodologies of the tool in brief, while we refer the readers elsewhere [6] for details.

Search Query Formulation: Once an exception occurs, we analyze both stack trace and context code of the exception in order to extract suitable query tokens. We develop a token graph (e.g., Fig. 3) from the stack trace, where the connectivity of a token is based on its implied relationships (e.g., class to method static relations) with other tokens. We consider the connectivity as a measure of token's importance, and exploit a graph-based term weighting algorithm (i.e., a variation of PageRank algorithm) in order to determine the *weight* of each token. We estimate the *degree of interest* [3] of each token, and also calculate the *frequency* of the token in the context code. We then normalize each of the three metrics, and calculate the final score (i.e., importance) of each token in the graph. Finally, we choose the top scored five tokens, and combine each three of them to formulate a list of search queries. Each of the queries essentially gets a score based on its token scores, and then the queries are also ranked for recommendation.

Data Collection & Context-Aware Ranking: The proposed tool follows a client-server architecture and it has two major entities– Eclipse plugin (client) (Fig. 5-(a, c, d)) and web service provider (server) (Fig. 5-(b)). Once a developer selects an encountered exception from *Console View* in the IDE, the client plugin collects associated context– *stack trace* and *context code*, and generates a web search request to the service provider [1] (e.g., Fig. 5-(b)). The provider module collects results from three search engines– *Google*, *Bing* and *Yahoo* and *StackOverflow* Q & A site against the client provided search query, and develops a dynamic corpus containing 100-120 result pages. It then analyzes the content of the pages, and checks if they discuss the exception of interest and the discussed exception belongs to a programming context similar to the one in the IDE and so on. We use title and textual content of the page to determine its content level relevance, whereas we exploit the content of <*code*>,<*pre*> and <*blockquote*> tags in the page to determine its context relevance. Those tags generally contain the *context* (i.e., stack traces and code segments) associated with the discussed exception in the page. Finally, each of the result pages is ranked based on its *content relevance*, *context relevance*, *search engine confidence* and *popularity* (metric details can be found elsewhere [6]). The service provider module then returns the top 30 results, and the client plugin displays them in the IDE with corresponding metric details (Fig. 5-(c)).

V. Performance

In order to evaluate the recommended queries by *SurfClipse*, we conducted a user study with five participants (graduate students) using five problem solving scenarios. Each of the participants solves the five exceptions, and we collect the queries they use for web search for each of the exceptions. We then compare the recommended queries by our tool with those queries for the same exception using *pyramid score* [4]. The metric determines if an auto-generated query resembles with a set of manually prepared search queries for the same exception, and we got an average *pyramid score* of 0.88. The finding indicates that the recommended queries are promising and comparable to the queries of expert users. We also conducted experiments using 75 programming errors and exceptions, and compared our results against existing approaches [3, 5], and three traditional web search engines (e.g., Google, Bing, Yahoo) and StackOverflow [6]. The proposed tool outperforms the existing approaches both in *precision* and *recall*. Among the search engines, Google performs the best. While our tool provides slightly less precise results than Google, it performs significantly better than Google in terms of *recall*. Detailed results can be found elsewhere [1, 6].

VI. Conclusion & Future Works

To summarize, we propose a context-aware meta search solution, *SurfClipse*, to the programming errors and exceptions encountered by software developers. The tool works both as a search query recommender and a meta search engine, and helps the developers in solving their programming problems especially associated with programming errors and exceptions. In future, we plan to conduct a more exhausted user study with prospective participants. We also plan for the recommendation of more sophisticated items such as relevant sections from a selected web page so that developers can easily locate the solutions and can solve the problems with reduced efforts.

References

[1] SurfClipse Web Portal. URL http://www.usask.ca/~mor543/surfclipse.
[2] J. Brandt, P.J. Guo, J. Lewenstein, M. Dontcheva, and S.R. Klemmer. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In *Proc. SIGCHI*, pages 1589–1598, 2009.
[3] J. Cordeiro, B. Antunes, and P. Gomes. Context-Based Recommendation to Support Problem Solving in Software Development. In *Proc. RSSE*, pages 85 –89, 2012.
[4] S. Haiduc, J. Aponte, and A. Marcus. Supporting Program Comprehension with Source Code Summarization. In *Proc. ICSE*, pages 223–226, 2010.
[5] L. Ponzanelli, A. Bacchelli, and M. Lanza. Seahawk: Stack Overflow in the IDE. In *Proc. ICSE*, pages 1295–1298, 2013.
[6] M. M. Rahman, S. Yeasmin, and C. Roy. Towards a Context-Aware Meta Search Engine for IDE-Based Recommendation about Programming Errors and Exceptions. In *Proc. CSMR-WCRE*, pages 194–203, 2014.