JAVA BASED SEARCH ENGINE

A Project Work Report

Submitted in the partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

CSE IBM BIG DATA ANALYTICS

Submitted by:

STUTI SRIVASTAVA

19BCS3834

Under the Supervision of:

Ms. JYOTI MEHRA

CHANDIGARH UNIVERSITY, GHARUAN, MOHALI - 140413, PUNJAB

December, 2020

STUTI SRIVASTAVA
Name and signature of student(s)

Ms. JYOTI MEHRA
Name and signature of Supervisor

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

PROJECT COMPLETION CERTIFICATE

JAVA BASED SEARCH ENGINE

This is to certify that STUTI SRIVASTAV has successfully completed the project work titled " JAVA BASED SEARCH ENGINE " *Submitted in the partial fulfilment for the award of the degree of* **BACHELOR OF ENGINEERING IN CSE IBM BIG DATA ANALYTICS**

This project is the record of authentic work carried out during the academic year

Ms. JYOTI MEHRA

Project Guide

Date: 08/12/2020

DECLARATION

I the undersigned solemnly declare that the project report is based on my own work carried out during the course of our study under the supervision of Ms. JYOTI MEHRA. I assert the statements made and conclusions drawn are an outcome of my work. I further certify that the work contained in the report is original and has been done by me under the general supervision of my supervisor.

II. The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or any other University of India or abroad.

III. We have followed the guidelines provided by the university in writing the report.

IV. Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and giving their details in the references.

STUTI SRIVASTAVA
(19BCS3834)

ACKNOWLEDGEMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to (Name of your Organization Guide) for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.
I would like to express my gratitude towards my parents and my department for their kind co-operation and encouragement which help me in completion of this project.

THANKS AGAIN TO ALL WHO HELPED

ABSTRACT

A fundamental problem of finding applications that are highly rel- evant to development tasks is the mismatch between the high-level intent reflected in the descriptions of these tasks and low-level implementation details of applications. To reduce this mismatch we created an approach called *Exemplar (EXEcutable exaMPLes ARchive)* for finding highly relevant software projects from large archives of applications. After a programmer enters a natural- language query that contains high-level concepts (e.g., MIME, data sets), Exemplar uses information retrieval and program analysis techniques to retrieve applications that implement these concepts. Our case study with 39 professional Java programmers shows that Exemplar is more effective than Sourceforge in helping program- mers to quickly find highly relevant applications

1. INTRODUCTION

Creating software from existing components is a fundamental challenge of software reuse. Naturally, when programmers develop software, they instinctively sense that there are fragments of code that other programmers wrote and these fragments can be reused. Reusing fragments of existing applications is beneficial because complete applications provide programmers with the contexts in which these fragments exist. Unfortunately, few major challenges make it difficult to locate existing applications that contain relevant code fragments.

A fundamental problem of finding relevant applications is the mismatch between the high-level intent reflected in the descriptions of these applications and low-level implementation details. This problem is known as the *concept assignment problem* [3]. Many search engines match keywords in queries to words in the descrip- tions of the applications, comments in their source code, and the names of program variables and types. If no match is found, then potentially relevant applications are never retrieved from reposito- ries. This situation is aggravated by the fact that many application repositories are polluted with poorly functioning projects [13]; a match between a keyword from the query with a word in the de- scription or in the source code of an application does not guarantee that this application is relevant to the query.

Objective:

Search engine will able to provide users required information at one particular place by using the words and patterns entered by the user during their search operation. All the information will be provided over the browser screen where users can select appropriate link filtered by the search query. Whatever the information presented to

the user can be in any form by default such as it may be in the form of web pages, pdf file, doc file etc. The search query will provide listing of web pages as per their occurrence during search operations.

This search mechanism will work on the concept of tags and meta tags which are used while writing the contents under the particular web pages. If the user's query don't matched with the tags and meta tags then it will go for summary section to match the given words in order to present the exact output or results. Upon going through web pages an index file will be created where listing of pages will be done by the system and present them as per their index number. Keyword density for a particular post under a particular web page will also help the system to set the priority for indexing the web page.

2. OUR APPROACH

In this section we describe the key ideas and give intuition about why and how our approach works.

2.1 The Problem

A straightforward approach for finding highly relevant applica- tions is to search through the source code of applications to match

[1] http://www.xemplar.org

keywords from queries to the names of program variables and types. The precision of this approach depends highly on programmers choosing meaningful names, but their compliance is generally dif- ficult to enforce [1].

This problem is partially addressed by programmers who create meaningful descriptions of the applications that they deposit into software repositories. However, state-of-the-art search engines use exact matches between the keywords from queries, the words in the descriptions, and the source code of applications, making it difficult for users to guess exact keywords to find relevant applications. This is known as the vocabulary problem, which states that "no single word can be chosen to describe a programming concept in the best way" [7]. This problem is general to all search engines but some- what mitigated by the fact that different programmers who participate in the projects use their vocabularies to write code, comments, and descriptions of these projects.

Modern search engines do little to ensure that retrieved applications are highly relevant to tasks or requirements that are described using high-level queries. To ensure relevancy, a code mining system should take high-level queries and return executable applications whose functionality is described by high-level requirements thereby solving an instance of the concept assignment problem. We believe that the rich context provided by whole applications make it easier for programmers to reuse code fragments from these ap- plications.

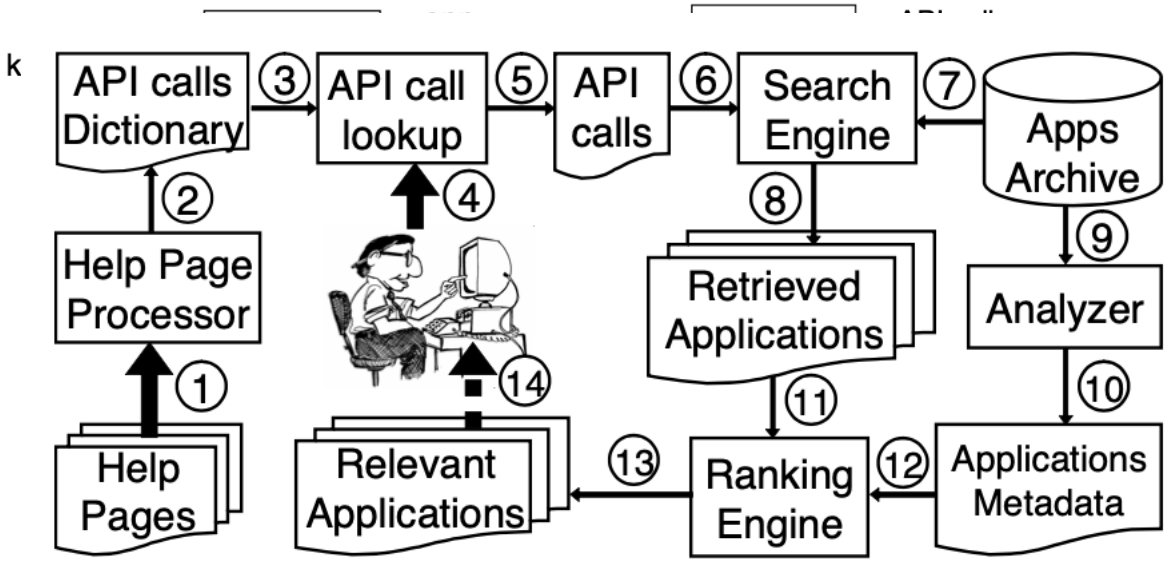
2.2 Key Ideas

Our goal is to automate parts of the human-driven procedure of searching for relevant applications. Suppose that requirements specify that a program should encrypt and compress data. When retrieving sample applications from Sourceforge[2] using the key- words encrypt and compress, programmers look at the source code to check to see if some API calls from third-party packages are used to encrypt and compress data. Even though the presence of these API calls does not guarantee that the applications are rele- vant, it is a good starting point for deciding whether to check these applications further.

2.3 Our Goal

Our goal is to develop a search engine that is most effective in the solution domain (i.e., the domain in which engineers use their ingenuity to solve problems [14, pages 87,109]). In the problem domain, requirements are expressed using vague objectives or wish lists. Conversely, in the solution domain engineers go into imple- mentation details. To realize requirements in the solution domain, engineers look for reusable abstractions that are often implemented using third-party API calls. Thus Exemplar should be most effective when keywords reflect the reality of the solution domain.

Consider a situation in which engineers develop a matchmaking application. Using keywords such as sweet and love to describe requirements from the problem domain, it is possible to find a variety of applications in Sourceforge with according descriptions. However, it is unlikely that these keywords are used to describe API calls in Java documentation, so it is up to engineers to investi- gate retrieved applications to determine if any code can be reused. Since Exemplar uses basic word matches in addition to locating API calls, it performs equally well in this situation when compared with existing search engines.



2.4 Our Approach

We describe our approach using an illustration of differences be- tween the process for standard search engines shown in Figure 1(a) and the Exemplar process shown in Figure 1(b).

Consider the process for standard search engines (e.g., Source- forge, Google code search) shown in Figure 1(a). A keyword from the query is matched against words in the descriptions of the applications in some repository (Sourceforge, Krugle) or words in the entire corpus of source code (Google Code Search). When a match is found, applications app_1 to app_n are returned.

Consider the process for Exemplar shown in Figure 1(b). A key- word from the query is matched against the descriptions of different documents that describe API calls of widely used software pack- ages. When a match is found, the names of the API calls call_1 to call_k are returned. These names are matched against the names of the functions invoked in these applications. When a match is found, applications app_1 to app_n are returned.

A fundamental difference between these search schemes is that Exemplar uses help documents to obtain the names of the API calls in response to user queries. Doing so can be viewed as instances of the *query expansion* concept in information retrieval systems [2] and *concept location* [20]. The aim of query expansion is to reduce this query/document mismatch by expanding the query with con- cepts that have similar meanings to the set of relevant documents. Using help documents, the

initial query is expanded to include the names of the API calls whose semantics unequivocally reflects spe- cific behavior of the matched applications.

In addition to the keyword matching functionality of standard search engines, Exemplar matches keywords with the descriptions of the various API calls in help documents. Since a typical appli- cation invokes API calls from several different libraries, the help documents associated with these API calls are usually written by different people who use different vocabularies. The richness of these vocabularies makes it more likely to find matches, and pro- duce API calls API $call_1$ to API $call_k$. If some help docu- ment does not contain a desired match, some other document may yield a match. This is how we address the vocabulary problem [7].

As it is shown in Figure 1(b), API calls API $call_1$, API $call_2$, and API $call_3$ are invoked in the app_1. It is less probable that the search engine fails to find matches in help documents for all three API calls, and therefore the application app_1 will be retrieved from the repository.

4. RANKING

In this section we discuss our ranking algorithm and its compo- nents such as dataflow computations.

4.1 Components of Ranking

There are three components that compute different scores in the Exemplar ranking mechanism: a component that computes a score based on word occurrences (WOS), a component that computes a score based on the number of relevant API calls (RAS), and a component that computes a score based on dataflow connections between these calls (DCS). The total ranking score is the weighted sum of these three ranking scores. Each component pro- duces results of different perspectives (i.e., word matches, API calls, dataflow connections). Our goal is to produce a unified ranking by putting these different rankings together in a single score.

The purpose of WOS is to enable Exemplar to retrieve applica- tions based on matches between words in queries and words in the descriptions of applications in repositories. This is a baseline Ex- emplar search that should be as effective as the one of Sourceforge. This approach may be useful for a small subset of applications that do not use any third-party API calls to implement different func- tions. In this case, Exemplar's ranking engine should rely on word matches to return relevant applications.

5. IMPLEMENTATION

In this section we describe how we implemented Exemplar.

5.1 Crawlers

Exemplar consists of two crawlers: *Archiver* and *Walker*. *Archiver* populated Exemplar's repository by retrieving from Sourceforge more than 30,000 Java projects that contain close to 50,000 submit- ted archive files, which comprise the total of 414,357 files. *Walker* traverses Exemplar's repository, opens each project by extracting its source code from zipped archive, and applies a dataflow computation utility to the extracted source code. In addition, the Archiver regularly checks Sourceforge to see if there are new updates and it downloads these updates into the Exemplar repository.

To extract all occurrences of invocations of JDK API calls in all available Java projects, we ran 65 threads for over 50 hours on five servers and 25 workstations: three of these servers have two dual core 3.8Ghz EM64T Xeon processors with 8Gb RAM each, and two have four 3.0Ghz EM64T Xeon CPUs with 32Gb RAM each. The workstations uniformly had one 2.83Ghz quad-core CPU and 2Gb RAM. This job resulted in finding close to twelve million invocations of these API calls from JDK 1.5 across all projects. The next item was to compute dataflow connections between these calls in all Java applications in the Exemplar's repository.

5.3 Computing Rankings

We use the Lucene search engine[4] to implement the core retrieval based on keyword matches. We indexed descriptions and titles of Java applications, and independently we indexed Java API call doc- umentation by duplicating descriptions about the classes and pack- ages in each methods. Thus when users enter keywords, they are matched separately using the index for titles and descriptions and the index for API call documents. As a result, two lists are re- trieved: the list of applications and the list of API calls. Each entry in these lists are accompanied by a rank (i.e., conceptual similarity, *C*, a number between 0 and 1).

The next step is to locate retrieved API calls in the retrieved ap- plications. To improve the performance we configure Exemplar to use the positions of the top two hundred API calls in the retrieved list. These API calls are crosschecked against API calls invoked in the retrieved applications, and the combined ranking score is com- puted for each application. The list of applications is sorted using the computed ranks, and returned to the user.

7.1 Case Study Results

We use one-way ANOVA, t-tests for paired two sample for means, and χ^2 to evaluate the hypotheses that we stated in Section 6.3.

7.1.1 Variables

A main independent variable is the search engine (SF, EWD, END) that participants use to find relevant Java applications. The other independent variable is participants ' Java experience. Depen- dent variables are the values of confidence level, *C*, and precision, *P*. We report these variables in this section. The effect of other variables (task description length, prior knowledge) is minimized by the design of this case study.

7.1.2 Testing the Null Hypothesis

We used ANOVA to evaluate the null hypothesis H_0 that the variation in an experiment is no greater than that due to normal variation of individuals ' characteristics and error in their measure- ment. The results of ANOVA confirm that there are large differ-

ences between the groups for *C* with $F = 129 > F_{crit} = 3$ with $p \approx 6.4 \cdot 10^{-55}$ which is strongly statistically significant. The mean *C* for the SF approach is 1.83 with the variance 1.02, which is smaller than the mean *C* for END, 2.47 with the variance 1.27, and it is smaller than the mean *C* for EWD, 2.35 with the variance 1.19. Also, the results of ANOVA confirm that there are large dif- ferences between the groups for *P* with $F = 14 > F_{crit} = 3.1$ with $p \approx 4 \cdot 10^{-6}$ which is strongly statistically significant. The mean *P* for the SF approach is 0.27 with the variance 0.03, which is smaller than the mean *P* for END, 0.47 with the variance 0.03, and it is smaller than the mean *P* for EWD, 0.41 with the variance 0.026. Based on these results we reject the null hypothesis and we accept

the alternative hypothesis H_1.

A statistical summary of the results of the case study for *C* and

T (median, quartiles, range and extreme values) are shown as box- and-whisker plots in Figure 3(a) and Figure 3(b) correspondingly with 95% confidence interval for the mean.

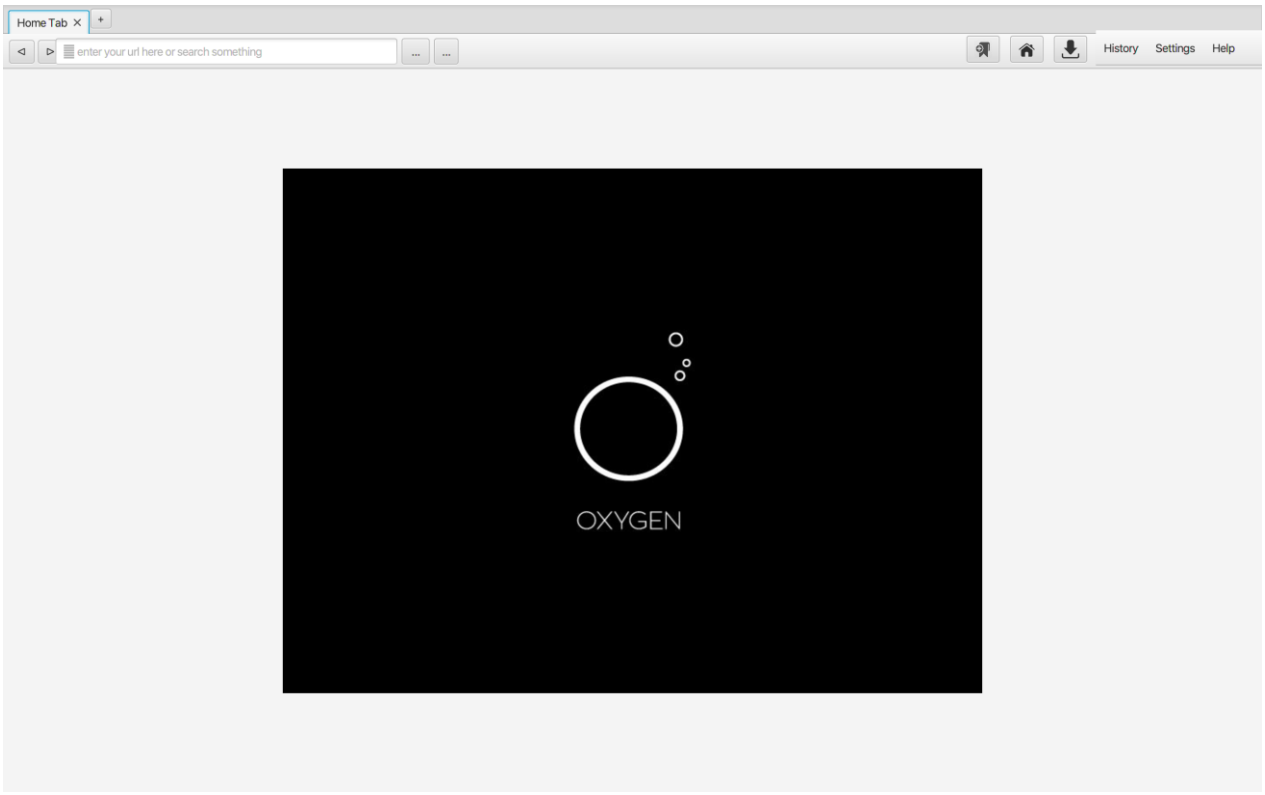
Proposed System:

Some special mechanism has been used to work on the concept of web crawling which can be integrated with this search engine. As to work under real time situation, it will enable the system, to go through the various web pages by using the keyword and prepare the list of it based on the search pattern. Before accessing the data to the users, it will first encrypt the index information and saves them to the database for their use.

Acknowledgments

We warmly thank anonymous reviewers for their comments and suggestions that helped us to improve the quality of this paper. We are especially grateful to Dr. Kishore Swaminathan, the Chief Sci- entist and Director of Research for his invaluable support. This work is supported by NSF CCF-0916139, CCF-0916260, Accen- ture, and United States AFOSR grant number FA9550-07-1-0030. Any opinions, findings and conclusions expressed herein are the authors ’and do not necessarily reflect those of the sponsors.

Screenshots:-



10. REFERENCES

- [1] N. Anquetil and T. C. Lethbridge. Assessing the relevance of identifier names in a legacy software system. In *CASCON*, page 4, 1998.

- [2] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

- [3] T. J. Biggerstaff, B. G. Mitbander, and D. E. Webster. Program understanding and the concept assigment problem. *Commun. ACM*, 37(5):72–82, 1994.

- [4] S. Chatterjee, S. Juvekar, and K. Sen. Sniff: A search engine for java using free-form queries. In *FASE*, pages 385–400, 2009.

- [5] D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *IEEE Trans. Software Eng.*, 31(6):446–465, 2005.

- [6] U. Dekel and J. D. Herbsleb. Improving api documentation usability with knowledge pushing. In *ICSE*, pages 320–330, 2009.

- [7] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Commun. ACM*, 30(11):964–971, 1987.

- [8] M. Grechanik, K. M. Conroy, and K. Probst. Finding relevant applications for prototyping. In *MSR*, page 12, 2007.

- [9] S. Henninger. Supporting the construction and evolution of component repositories. In *ICSE*, pages 279–288, 1996.

- [10] R. Hill and J. Rideout. Automatic method completion. In *ASE*, pages 228–235, 2004.

- [11] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *ICSE*, pages 117–125, 2005.

- [12] R. Holmes, R. J. Walker, and G. C. Murphy. Strathcona example recommendation tool. In *ESEC/SIGSOFT FSE*, pages 237–240, 2005.

- [13] J. Howison and K. Crowston. The perils and pitfalls of mining Sourceforge. In *MSR*, 2004.

- [14] E. Hull, K. Jackson, and J. Dick. *Requirements Engineering*. SpringerVerlag, 2004.

- [15] K. Inoue, R. Yokomori, H. Fujiwara, T. Yamamoto, M. Matsushita, and S. Kusumoto. Component rank: Relative significance rank for software component search. In *ICSE*, pages 14–24, 2003.

- [16] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto. Ranking significance of software components based on use relations. *IEEE Trans. Softw. Eng.*, 31(3):213–225, 2005.

- [17] C. W. Krueger. Software reuse. *ACM Comput. Surv.*, 24(2):131–183, 1992.

- [18] O. A. L. Lemos, S. K. Bajracharya, J. Ossher, R. S. Morla, P. C. Masiero, P. Baldi, and C. V. Lopes. Codegenie: using test-cases to search and reuse source code. In *ASE 07*, pages 525–526, New York, NY, USA, 2007. ACM.