Proceedings of

2014 International Conference on Data and Software Engineering (ICODSE)

November 26th - 27th, 2014
Institut Teknologi Bandung
Bandung, Indonesia

IEEE Catalog Number : CFP14AWL-USB

ISBN : 978-1-4799-8177-9

Data and Software Engineering Research Group
School of Electrical Engineering and Informatics
Institut Teknologi Bandung

ISBN: 978-1-4799-8177-9

Proceedings of

2014 International Conference on Data and Software Engineering (ICODSE)

Institut Teknologi Bandung, Indonesia

November 26th - 27th, 2014

2014 International Conference on Data and Software Engineering (ICODSE)

IEEE Catalog Number CFP14AWL-USB
ISBN 978-1-4799-8177-9

Additional copies of this publication are available from

Curran Associates, Inc
57 Morehouse Lane
Red Hook, NY 12571 USA
+1 845 758 0400
+1 845 758 2633 (FAX)
email: curran@proceedings.com

General Chair's Message

Welcome to 2014 ICODSE.

It is a pleasure for us to host the 2014 International Conference on Data and Software Engineering (ICODSE) at Institut Teknologi Bandung (ITB) campus, Bandung, Indonesia. The conference is organized by Data and Software Engineering Research Group, School of Electrical Engineering and Informatics, Institut Teknologi Bandung. 2014 ICODSE is co-organized by Vienna University of Technology, Austria, and ASEA Uninet. 2014 ICODSE is also technically co-sponsored by IEEE Indonesia Section.

The 2014 ICODSE conference aims at uniting researchers and professionals in the domains of data and software engineering, presenting and discussing high-quality research results and outcomes in their fields. This year, the conference welcomes contributions for 2 tracks: Data Engineering and Software Engineering.

In total, we received 130 submissions of authors from 10 countries around the world. All submissions were peer-reviewed (blind) by at least three reviewers drawn from external reviewers and the committees, and as the result, 68 papers were accepted to be presented in this conference. These papers are in the proceedings of 2014 ICODSE.

Finally, as the General Chair of the Conference, I would like to express my deep appreciation to all members of the Steering Committee, Technical Programme Committee, Organizing Committee and Reviewers who have devoted their time and energy for the success of the event.

For all participants, I wish you an enjoyable conference in this beautiful city of Bandung.

Bayu Hendradjaya

General Chair of the ICoDSE2014

Committee

General Chair

Bayu Hendradjaya	Institut Teknologi Bandung - Indonesia

Steering Committee

Benhard Sitohang	Institut Teknologi Bandung - Indonesia
A Min Tjoa	Vienna University of Technology - Austria
Ford Lumban Gaol	IEEE - Indonesia
Iping Supriana	Institut Teknologi Bandung - Indonesia
Richard Lai	La Trobe Univesity - Australia

Technical Program Committee

Chair:

Wikan Danar Sunindyo	Institut Teknologi Bandung - Indonesia

Members:

Adila Alfa Krisnadhi	Wright State University - USA
Agung Trisetyarso	Universitas Telkom - Indonesia
Ahmad Ashari	Gadjah Mada University - Indonesia
Amin Anjomshoaa	Vienna University of Technology - Austria
Ary Setijadi Prihatmanto	IEEE Indonesia Computer Society Chapter
Arya Ardiansyah	Eindhoven University of Technology - the Netherlands
Ayu Purwarianti	IEEE Indonesia Education Activity Committee
Bayu Hendradjaya	Institut Teknologi Bandung - Indonesia
Bernardo Nugroho Yahya	Ulsan National Institute of Science & Technology - South Korea
Chitra Hapsari Ayuningtyas	Alpen-Adria-Universitat Klagenfurt - Austria
Dade Nurjanah	Telkom University - Indonesia
Dang Tran Khanh	Ho Chi Minh City University of Technology - Vietnam
David Taniar	Monash University - Australia
Dwi Hendratmo Widiyantoro	Institut Teknologi Bandung - Indonesia
Erich Neuhold	IFIP Technical Committee on Information Technology Application
Estefania Serral Asensio	KU Leuven, Belgium
Fathul Wahid	Universitas Islam Indonesia - Indonesia
Fazat Nur Azizah	Institut Teknologi Bandung - Indonesia
GA Putri Saptawati	Institut Teknologi Bandung - Indonesia
Gerald Quirchmayr	University of Vienna - Austria
Josef Küng	Johannes Kepler University of Linz - Austria

Khabib Mustofa	Gadjah Mada University - Indonesia
Ladjel Bellatreche	Laboratorie d'Informatique Scientifique et Industrielle - France
Maguelonne Teisseire	University of Montpellier 2 - France
Michel Hassenforder	UHA - France
MM Inggriani	Institut Teknologi Bandung - Indonesia
Muhammad Asfand-e-yar	Masaryk University - Czech Republic
Muhammad Ilyas	University of Sargodha - Pakistan
Nguyen Duy Thanh	Ho Chi Minh City University of Technology - Vietnam
Robert P. Biuk-Aghai	University of Macau - PR China
RV Hari Ginardi	Institut Teknologi Sepuluh Nopember - Indonesia
Saiful Akbar	Institut Teknologi Bandung - Indonesia
Siti Rochimah	Institut Teknologi Sepuluh Nopember - Indonesia
Soleh Udin Al Ayubi	Boston Children's Hospital - USA
Stephane Bressan	National University of Singapore - Singapore
Vladimír Mařík	Czech Technical University in Prague - Czech Republic
Wenny Rahayu	La Trobe University - Australia
Wichian Chutimaskul	King Mongkut's University of Technology Thonburi - Thailand
Wikan Danar Sunindyo	Institut Teknologi Bandung - Indonesia
Yan Tang	European Space Agency - Netherland
Yudho Giri Sucahyo	University of Indonesia - Indonesia
Yudistira D. W. Asnar	Institut Teknologi Bandung - Indonesia
Zainal Arifin Hasibuan	University of Indonesia - Indonesia

Organizing Committee

Adi Mulyanto	Institut Teknologi Bandung - Indonesia
Christine Suryadi	Institut Teknologi Bandung - Indonesia
Hira Laksmiwati Soemitro	Institut Teknologi Bandung - Indonesia
Riza Satria Perdana	Institut Teknologi Bandung - Indonesia
Tricya E. Widagdo	Institut Teknologi Bandung - Indonesia
Yani Widyani	Institut Teknologi Bandung - Indonesia

Reviewers

A. Imam Kistijantoro	Institut Teknologi Bandung - Indonesia
Adila Krisnadhi	Wright State University - USA
Aditya Dakur	Hubvents VIT University - India
Agung Trisetyarso	Telkom University - Indonesia
Amin Anjomshoaa	Vienna University of Technology -Austria
Anto Nugroho	BPPT - Indonesia
Artem Lenskiy	Korea University of Technology and Education - South Korea
Ary Prihatmanto	Institut Teknologi Bandung - Indonesia
Arya Adriansyah	Eindhoven University of Technology - the Netherlands
Ayu Purwarianti	Institut Teknologi Bandung - Indonesia
Bayu Hendradjaya	Institut Teknologi Bandung - Indonesia
Benhard Sitohang	Institut Teknologi Bandung - Indonesia
Bernaridho Hutabarat	UPI YAI - Indonesia
Dade Nurjanah	Telkom University - Indonesia
David Taniar	Monash University - Australia
Dessi Puji Lestari	Institut Teknologi Bandung - Indonesia
Dicky Prima Satya	Institut Teknologi Bandung - Indonesia
Dwi Hendratmo Widyantoro	Institut Teknologi Bandung - Indonesia
Estefania Serral Asensio	KU Leuven - Belgium
Fathul Wahid	Universitas Islam Indonesia - Indonesia
Fazat Nur Azizah	Institut Teknologi Bandung - Indonesia
G. A. Putri Saptawati	Institut Teknologi Bandung - Indonesia
Hira Laksmiwati	Institut Teknologi Bandung - Indonesia
Indriana Hidayah	Gadjah Mada University - Indonesia
Iping Supriana Suwardi	Institut Teknologi Bandung - Indonesia
Jamaiah Yahaya	The National University of Malaysia - Malaysia
Ketut Wikantika	Institut Teknologi Bandung - Indonesia
Khabib Mustofa	Gadjah Mada University - Indonesia
Lukman Heryawan	Gadjah Mada University - Indonesia
Maguelonne Teisseire	University of Montpellier 2 - France
Mahmoud Neji	Sfax University - Tunisia
Maman Fathurrohman	Universitas Sultan Ageng Tirtayasa - Indonesia
Masayu Leylia Khodra	Institut Teknologi Bandung - Indonesia
Mewati Ayub	Maranatha Christian University - Indonesia
MM Inggriani Liem	Institut Teknologi Bandung - Indonesia
Muhammad Asfand-e-yar	Masaryk University - Czech Republic
Noor Maizura Mohamad Noor	Universiti Malaysia Terengganu - Malaysia
Nur Ulfa Maulidevi	Institut Teknologi Bandung - Indonesia

Oerip Santoso	Institut Teknologi Bandung - Indonesia
Ponmagal Rajendran	M G R Educational and Research Institute University - India
Rajesri Govindaraju	Institut Teknologi Bandung - Indonesia
Restya Astari	Institut Teknologi Bandung - Indonesia
Richard Lai	La Trobe University - Australia
Richki Hardi	University of Ahmad Dahlan - Indonesia
Rila Mandala	Institut Teknologi Bandung - Indonesia
Rinaldi Munir	Institut Teknologi Bandung - Indonesia
Robert P. Biuk-Aghai	University of Macau - PR China
Saiful Akbar	Institut Teknologi Bandung - Indonesia
Siti Rochimah	Institut Teknologi Sepuluh Nopember - Indonesia
Soleh Udin Al Ayubi	Boston Children's Hospital - USA
Soon Chung	Wright State University - USA
Suhardi Suhardi	Institut Teknologi Bandung - Indonesia
Thanh D. Nguyen	Ho Chi Minh City University of Technology - Vietnam
Thomas Basuki	Universitas Parahyangan - Indonesia
Tran Khanh Dang	Ho Chi Minh City University of Technology - Vietnam
Tricya Widagdo	Institut Teknologi Bandung - Indonesia
Wikan Danar Sunindyo	Institut Teknologi Bandung - Indonesia
Yani Widyani	Institut Teknologi Bandung - Indonesia
Yudistira Dwi Wardhana Asnar	Institut Teknologi Bandung - Indonesia

Contents

Data Engineering Track

Software Engineering Track

Java Archives Search Engine Using Byte Code as Information Source

Oscar Karnalim
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
23512012@std.stei.itb.ac.id

Rila Mandala
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
rila@stei.itb.ac.id

***Abstract*—Information from computer programs can be extracted from its source code, external documentation, and compiled code. Although compiled code is an assured information source which is always exists in published computer programs, it is seldom used by the existing search engines since some reverse engineering tasks are needed. In this research, a search engine for Java archives that uses byte code (compiled code for Java Archive) as its information source is developed. It enables user to search within a collection of Java Archives without relying with source code and external documentation. Compared with Penta and FindJar [2][7], A novel term extraction process beyond the file and class name is proposed, which includes field name, method name, string literal used in program, program flow weighting, and method expansion. Exclusive tokenization, stopping, and stemming are also implemented to improve effectiveness. Based on evaluation, it has a fairly good effectiveness although it may vary based on terms stored on index. Its effectiveness is higher than FindJar main features reimplementation which indicates that detailed compiled code has positive influences in computer programs search engine. Efficiency depends on how many terms stored on index and how many process used at certain step.**

Keywords—search engine; Java archive; information extraction; compiled code; byte code

I. INTRODUCTION

Computer programs could have many variations, such as executable files, libraries, or source codes. Information from computer program can be extracted from its source code [1], external documentation [1], and compiled code [2]. Compiled code is the most reliable feature since it always exist in published computer programs. Although the fact that it is the most reliable feature, it is seldom used since some reverse engineering tasks are needed on its information extraction step. Some existing search engine have applied compiled code as its information source but only limited to file name because of its simple extraction mechanism.

Without source code and external documentation, file name is not good enough to represent computer program as document in search engine since many file name contains product name and do not describe its functionality. This problem can be solved by extracting detailed compiled code features from a computer program and implementing exclusive tokenization, stopping, and stemming on its terms.

In this research, a search engine for computer programs that uses compiled code as its information source is proposed. Computer program used in this research is Java archive where each archive is treated as a document and each class included on it are treated as a part of document. Compiled code used in Java archive is called byte code which is parsed based on Java SE 7 specification and Java virtual machine compiler [3].

II. RELATED WORK

Information extraction of computer programs that is based on its source code [1], external documentation [1], and compiled code [2] has been applied in many researches. Information extraction based on source code is implemented by Maletic, Marcus, Kuhn, and Ohloh Code whereas external documentation is implemented by Maletic and Ohloh Code [1][4][5][6][7]. Maletic and Kuhn use source code comments and identifiers in their research. Maletic uses it to categorize software components automatically whereas Kuhn uses it to enrich information on reverse engineering process and source code topic identification [1][5][6]. Maletic, Kuhn, and Ohloh Code use source code as documents to search some source code fragments with information retrieval approach [4][7]. External documentation is also used by Maletic and Ohloh Code to improve its effectiveness. Although compiled code is seldom used, it also has been applied in some researches. Penta use file name and textual files in Java Archive to identify its licenses [2]. FindJar use file name as information source on Java Archive search engine [7].

Information source is a crucial task in search engine's indexing process since it determines all terms stored in index. Search engines for computer programs have been developed in some researches which some of them are Ohloh Code and FindJar. Ohloh Code uses source codes as search engine's documents whereas FindJar uses file name as its index terms [7][8].

978-1-4799-7996-7/14/$31.00 ©2014 IEEE

In this research, a search engine for computer programs that extract information based on compiled code is developed by expanding FindJar main features. Many detailed compiled code features like class name, field name, method name, and string literals used in programs are applied. It also uses exclusive tokenization, stopping, stemming, n-grams concatenation, program flow weighting, and method expansion mechanism to improve its efficiency. This research focuses on compiled code as information source because of its exact whereabouts and limited to Java Archive as its documents.

III. ANALYSIS AND DESIGN

The search engine applies vector space model with cosine correlation as its retrieval model and tf-idf weighting as its term weighting mechanism. Terms are taken by extracting file name, class name (including package name), method name, field name, and string literals from class file(s) on Java Archive. These textual informations are treated as string literals which are converted to terms by tokenizing, stopping, and stemming. Tokenization is applied in exclusive way because identifiers must be considered as set of terms. It is divided to three step which are:

a. Word tokenization, Input string are split using invalid identifier character as its separator.

b. Identifier tokenization, Terms which length is higher than 1 are split using Java naming rules and converted to lowercase format.

c. Term Concatenation, Terms are concatenated in n-gram format. This step is also repeated with lower positive n values. Concatenated terms which length is lower than 2 are ignored.

Stop words are classified to five category that are programming language keywords (Java and object oriented language), Java literals, developer terms, client terms, and English stop words. After stopping, each term are stemmed with Porter stemmer.

The example of term conversion for input string “Java ASTTree1988” with n = 2 can be seen in Figure 1. Input string is separated using invalid identifier character to “Java” and “ASTTree1988”. Each term are split using Java naming rules and converted to lowercase format which produces “Java”, “ast”, “tree”, and “1988” as terms. These terms are concatenated with n = 2 which produces unigram and bigram format (because it is repeated until n = 0). This step produces “Java”, “ast”, “tree”, “1988”, “Javaast”, “asttree”, and “tree1988” as terms. Each term that is not classified as stop words are stemmed using Porter stemmer (term “Java” is classified as stop word). So in this case, input string “Java ASTTree1988” are converted to “ast”, “tree”, “1988”, “Javaast”, “asttre”, and “tree1988”.

String literals from class files on Java Archive are extracted by reverse engineering string and method invocation on its byte code. Each method invocation are replaced by its invoked method content where each non-recursive method are expanded till no method invocation exist. This mechanism are used to minimalize the effect of instructions encapsulation variants (e.g. function and procedure encapsulation). The example of method expansion can be seen in Figure 2 and Figure 3. Figure 2 represents method content before expansion where Figure 3 represents method content after expansion. In this example, method *B* invocation at instruction 4 in method *A* are replaced by the content of method *B*.

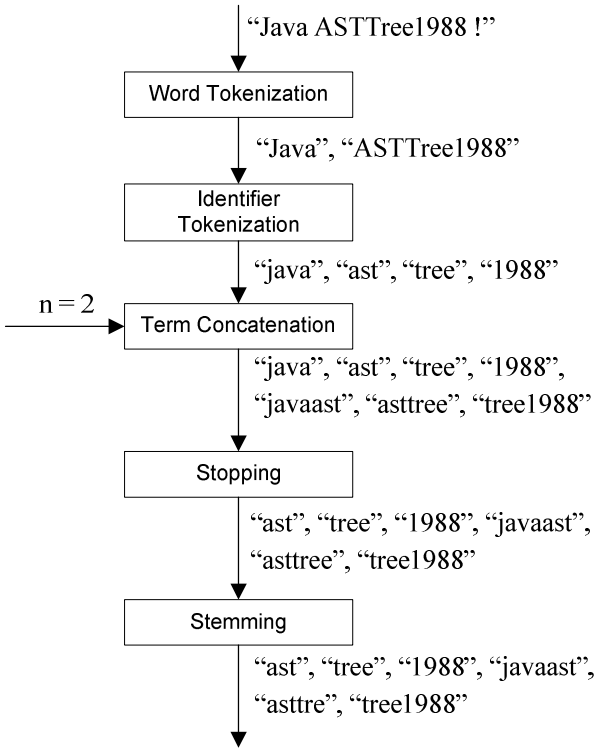

Figure 1. The example of term conversion with n = 2

Method A
1. Instruction 1
2. Instruction 2
3. String “precision” invocation
4. Method B invocation
5. Instruction 3
6. String “accuracy” invocation

Method B
1. Instruction 1
2. String “search” invocation
3. Instruction 2

Figure 2. Method A and B before expansion

Recursive methods are expanded by limiting its expansion at a certain numeric value. They are detected by selecting all strongly connected components from method relation graph.

Method relation graph is built by treating each method as a node where each method *B* invocation in method *A* is converted to an edge that points from node *A* to *B*. The example of a method relation graph can be seen in Figure 5 which is generated based on Figure 4. In this example, each method are converted to nodes where each method invocation are converted to edges (e.g. edge from A to B are built based on method invocation B in method A). Non-recursive methods are strongly connected components that has only one node without an edge that point itself. Strongly connected components are marked by rectangle in Figure 5.

Method A
1. Instruction 1
2. Instruction 2
3. String "precision" invocation
4.1. Instruction 1
4.2. String "search" invocation
4.3. Instruction 2
5. Instruction 3
6. String "accuracy" invocation

Method B
1. Instruction 1
2. String "search" invocation
3. Instruction 2

Figure 3. Method A and B after expansion

Method A
1. Method B invocation
2. Instruction 1

Method B
1. Instruction 1
2. Method A invocation
3. Method C invocation

Method C
1. Instruction 1

Method D
1. Method A invocation

Method E
1. Method E invocation

Figure 4. Sample methods for method graph relation example

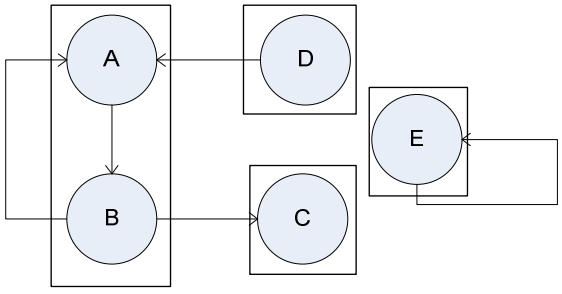

Figure 5. Method graph based on sample methods

String literals are taken from string invocation in each expanded methods and weighted based on how high its probability to be invoked when program runs. The probability is calculated based on program control flow such as looping, branching, and exception. Before weighting, program control flow is converted to graph where each *goto* represents an edge and each source or target instruction represents a node (Exceptions are also remodeled as branching and embedded in the graph). The probability of each node and edge is calculated as follows :

a. The first node's probability is 1 since it is always be invoked when program runs.

b. For each edge, its probability is calculated by dividing its source node's probability with the number of edges that is sourced from it.

c. For each node, its probability is calculated by sum up all the weight of edges that target itself (except back edges).

The example of calculating node and edge probability can be seen in Figure 6. Node's probability are summed from the weight of all non-back edges that point itself (0.25+0.25+0.25=0.75) whereas each edge's probability are source node's probability divided by the number of edges that is sourced from it (0.75 / 3 = 0.25).

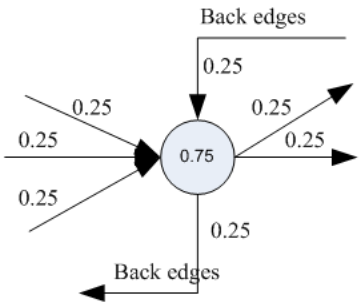

Figure 6. The example of branch weighting

Loops are detected by adopting Miecznikowski's algorithm which recognizes loop as *while* loop, *do-while* loop, and unconditional loop (Miecznikowski's algorithm can be seen in [10]). Nested loops are detected by removing some conditional nodes whereas string literals are affected by multiplying its probability with the number of loops. To avoid zero probability problem on multiplication step, the number of loops is always added by 1.

After method expansion and program control flow weighting, weight of string literal is weighted by tf-idf weighting and treated as term weight. Each term and its weight are stored in serialized index which is used as data source on its retrieval step. Retrieval step are conducted by tokenizing, stopping, and stemming input query and retrieve all relevant Java Archives based on input query terms.

IV. EVALUATION AND RESULT

The search engine developed in this research is evaluated by measuring two main factors which are efficiency and effectiveness. Efficiency is measured based on index size, average indexing time, and mean average query latency whereas effectiveness is measured using mean average precision (MAP) and recall. Dataset used in this evaluation are FindJar's sub-dataset which queries are determined from manual judgement based on its description. Dataset are taken manually since FindJar did not provide API to retrieve all data on its dataset at once. The statistics of dataset used in this research can be seen in Table I.

TABLE I. DATASET STATISTICS

Statistic Variable	Values
Number of Java archives in dataset	552 Java archives
Number of queries in dataset	1860 queries
Number of manual judge	8 judge
Shortest query	DJ
Longest query	org.apache.sling.jcr.jackrabbit.usermanager-2.0.2-incubator
Average number of characters in queries	17.332 characters
Average number of words in queries	1.161 words
Query with the highest number of relevant java archives	Spring (21 Java archives)
Average number of relevant Java archive for each query	2.298 Java archives
The largest Java archive size in dataset	7580792 bytes
The smallest Java archive size in dataset	1764 bytes
Dataset size	146 megabytes

In this research, Influence of each feature is measured in term of its efficiency and effectiveness which can be seen in Table II. They are measured in percentage unit which is obtained by comparing cases where the selected feature exists or absences (percentage value is based on case where the selected feature absences). Evaluation are conducted in Windows 7 Ultimate 32-bit with 4.00 GB RAM, and Intel(IR) Core(TM) i7-3770 CPU @ 3.40 GHz 3.90 GHz as its processor.

Method content is the most influential feature in term of efficiency. It greatly affects index size and mean average query latency since many string literals used in program are contained on it. Average indexing time is also greatly affected because many process needed on its extraction step. Method expansion and program flow weighting do not affect index size because they only modify the weight of each index terms.

File name is the most influential feature in term of recall since most users know which Java Archive they are looking for. Java Archive is usually named as its product name which is frequently used as a query. It ensures the selected Java Archive to be retrieved although in low position. File name does not affects mean average precision greatly since some product name consists of common terms which make it more difficult to be put in high position. Class name greatly affects mean average precision since many of them are used in the queries and are adequately represent Java Archive's functionality.

TABLE II. INFLUENCE OF FEATURES

Features	Efficiency			Effectiveness	
	Index size	*Average indexing time*	*Mean average query latency*	*Mean average precision*	*Recall*
File name	0.151 %	0.253 %	1.046 %	3.428 %	4,939 %
Class name	3.026 %	0.311 %	16.105 %	12.052 %	1.579 %
Field name	12.902 %	1.336 %	24.806 %	- 0.484 %	0.115 %
Method name	11.782 %	0.228 %	21.969 %	- 0.866 %	0.346 %
Method content	56.959 %	729.01 %	99.586 %	0.573 %	2.425 %
Method expansion	0 %	83.858 %	5.228 %	- 0.027 %	0 %
Program control flow weighting	0%	1.077 %	4.772 %	1.205 %	0 %

Some features negatively affect mean average precision since they contain a lot of common terms which are not explicitly related with its Java Archive (e.g. method name may contains some common terms like "build" and "close"). Method expansion and program control flow weighting do not affect recall since it only modify index term's weight.

Recursive methods are expanded by limiting its expansion at a certain numeric value which is also evaluated in this research. Influence of each recursive method expansion constant can be seen in Table III. They are measured by comparing it with default case (constant value 0).

TABLE III. INFLUENCE OF RECURSIVE METHOD EXPANSION CONSTANT

Recursive method expansion constant	Efficiency			Effectiveness	
	Index size	*Average indexing time*	*Mean average query latency*	*Mean average precision*	*Recall*
1	0 %	0. 378 %	0.207 %	-0.027 %	0 %
2	0 %	6.992 %	- 0.207 %	-0.028 %	0 %
3	0 %	10.404 %	0.414 %	-0.047 %	0 %

Changes in the value of recursive method expansion constant do not have much impact on index size, mean average query latency, and recall since it only modify index term's weight. Indexing time is proportionally increased to recursive method expansion constant since expanding recursive method takes a

considerable amount of process. Mean average precision is inversely proportional to recursive method expansion constant since some terms in recursive method are not explicitly related with its Java Archive.

Loops that are detected in this research are considered as constant loops by multiplying its amount with an integer value called loop constant. The influence of loop constant can be seen in Table IV which is compared with loop constant 1 as its default case. Loop constant does not greatly affect its efficiency since it only adds multiplication process on extraction step. Some terms in loops are not explicitly relevant to its Java archive, so the greater loop constant will lower its mean average precision.

TABLE IV. INFLUENCE OF LOOP CONSTANT

Loop constant	Efficiency			Effectiveness	
	Index size	*Average indexing time*	*Mean average query latency*	*Mean average precision*	*Recall*
2	0 %	0.441 %	0.207 %	- 0.116 %	0 %
3	0 %	0.765 %	0 %	- 0.288 %	0 %
4	0 %	0.683 %	0.207 %	- 0.372 %	0 %
5	0 %	0.566 %	0.207 %	- 0.714 %	0 %

Tokenization step concatenates result token in n-gram format which initial n values may be vary and also evaluated in this research. The influence of initial n-gram can be seen in Table V which result is compared with unigram case (n=1). Index size, average indexing time, and mean average query latency are getting larger as n value grows because index terms are also increasingly varied. Based on dataset, recall is only improved as n go up from 1 to 2 which means significant terms in Java Archive are consist of 1 or 2 terms. Mean average precision yields the highest percentage at n=3 since many unigram, bigram, and trigram terms improves rellevant Java Archive ranking position.

TABLE V. INFLUENCE OF INITIAL N-GRAM

n	Efficiency			Effectiveness	
	Index size	*Average indexing time*	*Mean average query latency*	*Mean average precision*	*Recall*
2	424.6 %	2.632 %	603.3 %	6.798 %	1.831 %
3	953.2 %	5.895 %	1396.6 %	7.224 %	1.831 %
4	1419.2 %	9.252 %	2069.1 %	7.118 %	1.831 %
5	1825.4 %	9.497 %	2522.5 %	6.995 %	1.831 %

Based on evaluation conducted in this research, efficiency are mostly depends on how many terms stored on index Indexing time is not only affected by how many index terms but also how many process used at indexing step. Mean average precision and recall may be vary based on terms stored on index whereas mean average precision is also affected by term weight. Features that enhance its effectiveness are also combined and compared with FindJar main features reimplementation. It yields greater effectiveness (MAP : 69.282 %, recall : 94.369 %) than FindJar main features reimplementation (MAP : 54.626 %, recall : 60.875 %). The combined features are file name, class name, method content, program flow weighting with loop constant 1, and n=3 as its initial n-gram value. Its mean average precision is still low since Java Archives are mostly composed of common terms.

V. CONCLUSION

In this research, a Java Archive search engine that uses bytecode as its information source has been developed. It enables user to search within a collection of Java Archives without relying with source code and external documentation. Based on evaluation result, detailed byte code features have positive influences in search engine for Java Archives. File name, class name, method content, program control flow weighting with loop constant 1, and n=3 as initial n-gram value yields greater effectiveness than FindJar main feature reimplementation. Its mean average precision is still low since many Java Archives mostly consists of common terms whereas its recall is considerably high since many queries consist of index terms.

Efficiency depends on how many terms stored on index and how many process used at certain step. Recall depends on the number of Java Archive significant terms which are stored on index. Mean average precision is affected based on index terms and its weight. File name, class name, method content, and program control flow weighting enhance mean average precision although method name, field name, and method expansion are lowering it. Loop constant yields the best mean average precision at 1 where greater value will lower its mean average precision. Initial n-gram value yields the best effectiveness at 3. Its effectiveness will be reduced if it is reduced or added from 3. The highest effectiveness enhancement is gained through transition between 1 to 2.

VI. FUTURE WORK

Certainly, this research has many aspects that needs improvements which are :

1. Features are still considered equal in terms of weight although some features may affect more than others.
2. Object oriented techniques such as overriding and polymorphism are still not handled.
3. Programs used in this research are still limited to Java archive which can be expanded to any other program files.
4. Terms are stored in single index file without using any compression method which is not scalable enough to handle large dataset.

REFERENCES

[1] Maletic, J. L., & Valluri, N. (1999). Automatic Software Clustering via Latent Semantic Analysis. *Automated Software Engineering, 1999. 14th IEEE International Conference* (page 251 - 254). Cocoa Beach: IEEE.

[2] Penta, M. D., German, D. M., & Antoniol, G. (2010). Identifying Licensing of Jar Archives using a Code-Search Approach. *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference* (page 151 - 160). Cape Town: IEEE.

[3] Gosling, J., Joy, B., Steele, G., & Bracha, G. (2005). *Java Language Spesification Third Edition.* California: Sun Microsystems, Inc.

[4] Marcus, A., Sergejev, A., Rajlich, V., & Maletic, J. L. (2004). An Information Retrieval Approach to Concept Location in Source Code. *Reverse Engineering, 2004. 11th Working Conference* (page 214 - 223). Delft: IEEE.

[5] Kuhn, A., Ducasse, S., & Girba, T. (2005). Enriching Reverse Engineering with Semantic Clustering. *Reverse Engineering, 12th Working Conference* (page 1095-1350). Pittsburgh: IEEE.

[6] Kuhn, A., Duccase, S., & Girba, T. (2007). Semantic Clustering: Identifying Topics in Source Code. *Information and Software Technology Volume 49 Issue 3* , 230–243.

[7] FindJar. (2008). Retrieved December 12, 2013, from Jar Search - FindJar.com: http://findjar.com/index.x

[8] Ohloh. (2013). Retrieved December 22, 2013, from Ohloh Code Search: http://code.ohloh.net/

[9] Gosling, J., Joy, B., Steele, G., & Bracha, G. (2005). *Java Language Spesification Third Edition.* California: Sun Microsystems, Inc.

[10] Miecznikowski, J. (2003). *New Algorithms for a Java Decompiler and Their Implementation in Soot.* Montreal: Master Thesis, Computer Science, McGill University.