ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 7, July 2016

Source Code Summarization of Context for Java Source Code

Suhas Mahakalkar, Prof. A. D. Gujar,

M. E Student, Dept. of Computer, BSCOER, Savitribai Phule Pune University, Pune, India

Asst. Professor, Dept. of Computer, BSCOER, Savitribai Phule Pune University, Pune, India

ABSTRACT: Programmers need software documentation. Consistent, correct and complete documentation of a software system is an important for the maintainer to gain its understanding, to ease its learning and/or relearning processes, and to make the system more maintainable. Programmers don't have the time and resources to write documentation. Therefore, automated solutions are desirable. Designers of automatic documentation tools are limited because there is not yet a clear understanding of what characteristics are important to achieving high quality summaries.

In this paper we propose source code summarization technique that writes English description of java methods by analysing how those methods are invoked.Analyzed and compare the summary with state-of-the-art automatic summarization tool.

KEYWORDS: Assisted Source code summarization, automatic documentation, program comprehension

I. INTRODUCTION

Programmers rely on good software documentation. [1], [2]. Software documentation exists to help software maintainers understand a system and its processes [3]. However, in practice, source code documentation is expensive to produce, resulting in often incomplete and low quality documentation [4]. Additionally, as software undergoes maintenance and updates, this high cost results in software documentation becoming outdated as software is update [5], [6]. Poor software documentation can lead to a software system rapidly degrading in quality and usability [3].

Writing documentation is difficult because it requires programmers to impart knowledge to other programmers. This difficulty creates a separation between the authors and readers of documentation. Authors are the initial developers of source code. Readers are any person attempting to understand source code for usage or maintenance after the fact. Authors typically write summaries of source code in order to communicate to readers. Authors often use document generation tools in order to communicate the readers in a standardized fashion. Tools such as Doxygen1 and JavaDoc2 have helped source code documentation by standardizing format and presentation of source code documentation. However, these tools still rely on manual author input. This means authors still face the cost of documenting source code if they wish to communicate to the readers.

Automatic source code summarization has begun to emerge as a means of helping authors expedite the documentation process [7], [8], [9], [10], [11], [12], [13]. These tools automatically provide a wide variety summarization techniques and outputs. For example, Haiduc et al. [7] and Rodeghero et al. [11] provide a list of keywords that describe the method. These keywords are acquired using a vectorspace model to determine importance. By contrast, work by Sridhara et al. [12] and my own work [8] generate natural language sentences that summarize methods. These tools use natural language generation templates to create human readable sentences summarizing methods.

In order for automatic summarization to mature, the field needs to better understand documentation quality. Developers of these tools must understand what makes a good summary. They must also understand what types of information are important. Additionally, developers need to understand if some information is irrelevant or misleading, to avoid including such information in their tools. In this paper, I propose new ideas and directions of research and present early findings into understanding how to improve automatic source code summarization.

.

II. RELATED WORK

The findings that result from this proposal would most directly affect automatic source code summarization. This paper cites several source code summarization tools. Haiduc et al. [7] used a vector space model to summarize methods. In this approach, methods were treated as documents, with keywords in the method being the terms. The importance of each keyword in each method was identified using term frequency/inverse document frequency (tf/idf). Eddy et al. [3] conducted a later study independently verifying that keywords selected by Haiduc et al.'s approach accurately summarized Java methods. Rodeghero et al. [11] modified Haiduc's algorithm by weighting tf/idf scores for keywords based on their immediate surroundings. Rodeghero et al based these weights on an eye-tracking study which found that method signature and method calls were more important to programmers, while control flow statements were less important.

Some source code summarization tools automatically generate natural language summaries. Sridhara et al. [12] developed an approach that selects important lines of code from a Java method and translates them into natural language summaries. The lines of code are translated by interpreting keywords using a Software Word-Usage Model (SWUM), developed by Hill et al. [5], [6], which can infer parts of speech for keywords in identifier names. Moreno et al. [10] used knowledge of method stereotypes to create natural language summaries for Java classes.

III. PROBLEM

The key problem facing automatic source code summarization is that there is no agreed upon or well-defined standard of "good" documentation. While research exploring measurement of documentation quality exists [9], [11], much of this research relies on assumptions that, in some cases, have not been verified in the literature empirically. It is unclear what a good source code summary entails. Further, it may be that a high quality manually written source code summary will be different than a high quality automatically generated summary. Developers of source code summarization tools need to understand what a "good" summary is before they can develop tools that create them. In this paper, I propose three specific questions related to understanding what "good" documentation looks like, both in manually written and automatically generated summarizations and automatic documentation.

By answering these questions, designers of documentation tools can better understand what the goals of automatic software documentation should be. As automatic summarization tools become more sophisticated, it will be important to direct research towards areas more likely to produce better summaries. While automatic source code summarization is still in its infancy, knowing what goals to strive for will direct early research more productively. Further, understanding what "good" documentation is with respect to automatic source code summarization will illuminate what manually written "good" documentation should include.

IV. IMPLEMENTATION DETAILS

A. Proposed System:

1. Software Word Usage Model

The Software Word Usage Model (SWUM) is a technique for representing program statements as sets of nouns, verbs, and prepositional phrases. SWUM works by making assumptions about different Java naming conventions, and using these assumptions to interpret different programs. SWUM first splits the identifier names using the typical Java convention of camel case. Next, it reads verbs from the method as the starting word from the method identifier.

2. Natural Language Generation Systems

The design of Natural Language Generation (NLG) systems typically follows an architecture described by Reiter and Dale.

Figure 1 illustrates this architecture. Conceptually, the architecture is not complicated: a "communicative goal" is translated from a series of facts into readable natural language sentences, known as "surface text." The NLG system has three main components, each of which is made up of several individual steps.

- Document Planner:
- Microplanner
- Surface Realizer

3. PageRank

PageRank is an algorithm for approximating the importance of the nodes in a graph. PageRank calculates importance based on the number of edges which point to a given node as well as the importance of the nodes from which those edges originate. PageRank is well-known for its usefulness in ranking web pages for web search engines. However, PageRank has seen growing relevance in its applications in software engineering. In particular, a body of work has shown how PageRank can highlight important functions or methods in a software program.

A common and effective strategy is to model a software program as a “call graph”: a graph in which the nodes are functions or methods, and the edges are call relationships among the methods. Methods that are called many times or that are called by other important methods are ranked as more important than methods which are called rarely, and thus have few edges in the call graph. We follow this model of using PageRank for this paper.

B. System Architecture:

This section describes the details of our approach, including each step of our natural language generation system. Generally speaking, our approach creates a summary of a given method in three steps: 1) use PageRank to discover the most important methods in the given method’s context, 2) use data from SWUM to extract keywords about the actions performed by those most important methods, and 3) use a custom NLG system to generate English sentences describing for what the given method is used.

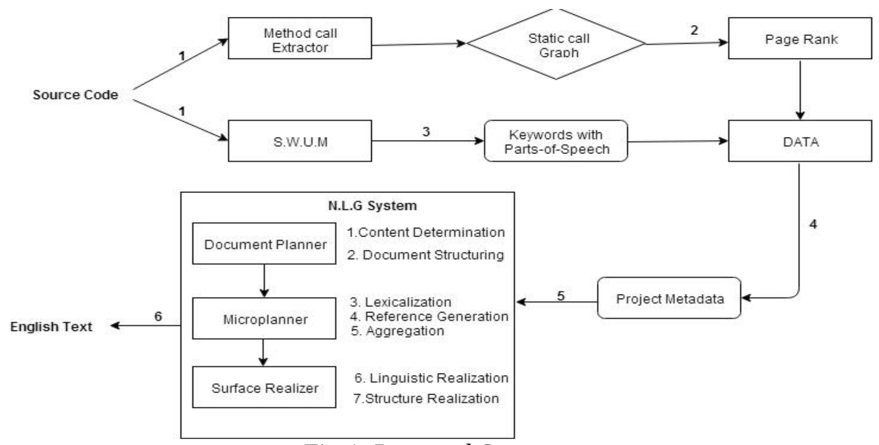


Fig 1: Proposed System.

We create five different types of “messages” that represent information about a method’s context. These messages are briefly described in Table 1.

Table 1: Message description

Message Type	Description
Quick Summary Message	Short sentence that describes method
Return Message	Notes the return type of the method
Importance Message	States how important a method is based on PageRank
Output Used Message	Describe at most 2 methods that call this method
Call Message	Describe at most 2 methods that this method calls
Use Message	Gives an example of how the message can be used.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 7, July 2016

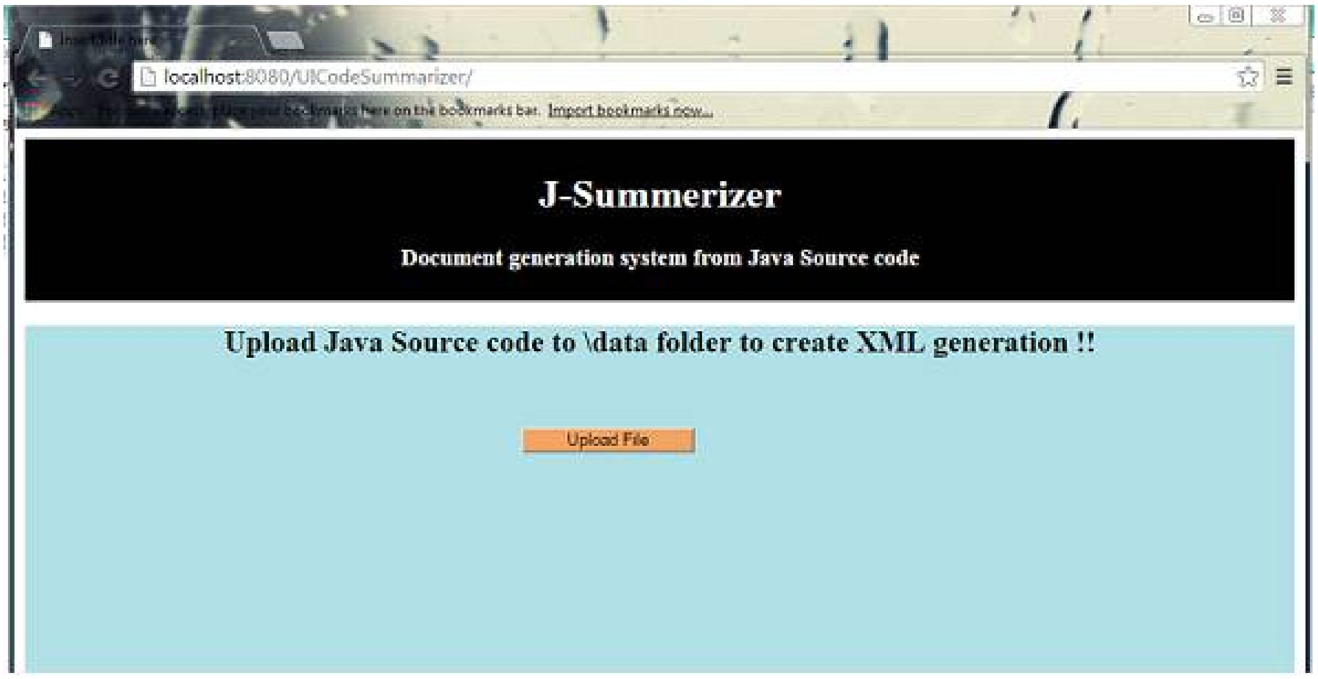
V. RESULTS AND DISCUSSION

In proposed work aim to generate source code summary based on Java source code, which will be easy to understand and illuminate what manually written document efficiency.

A. Practical Work:

1. Input: Java source Code

To show the effectiveness of proposed system some experiments are conducted on java based windows machine using Eclipse as IDE. After launching the tool, it will display



J-Summerizer

Document generation system from Java Source code

Upload Java Source code to \data folder to create XML generation !!

Fig 2: Upload Java Source Code

2. Summary generation from java source code in PDF format

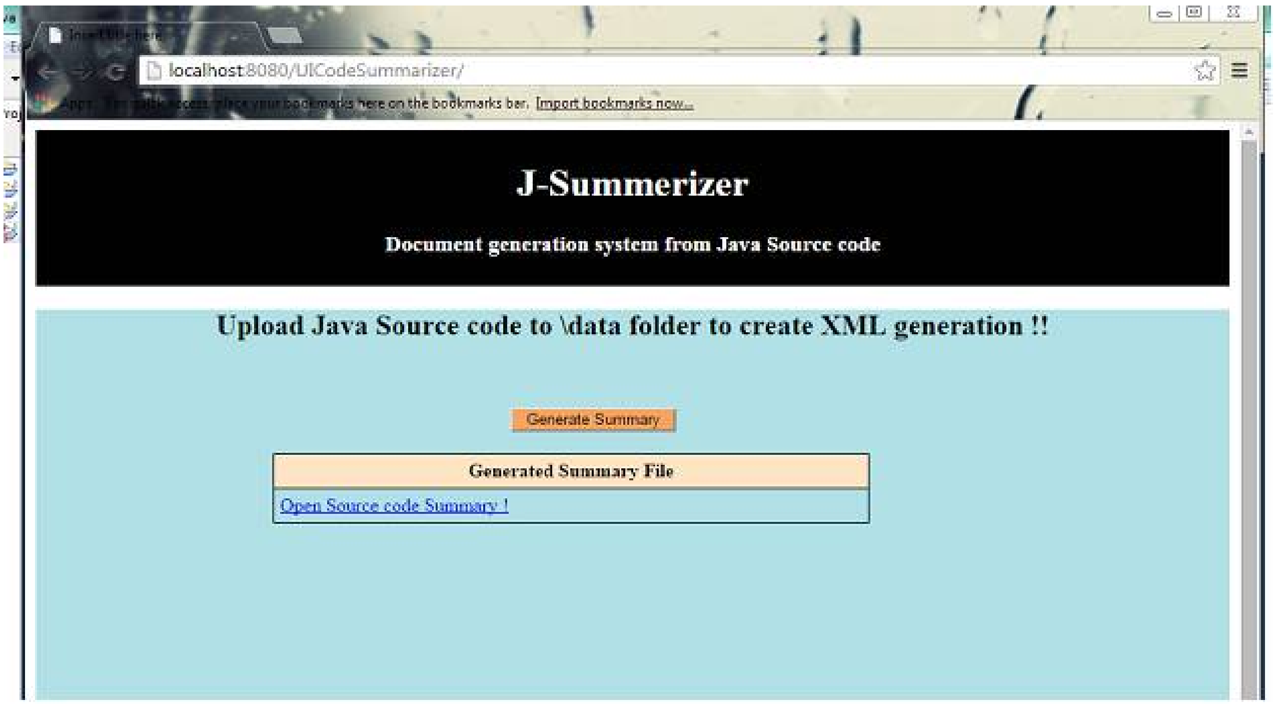
After uploading the Java source code, Summary document in PDF format got generated and the hyperlink for the same is displayed on screen.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 7, July 2016



J-Summerizer

Document generation system from Java Source code

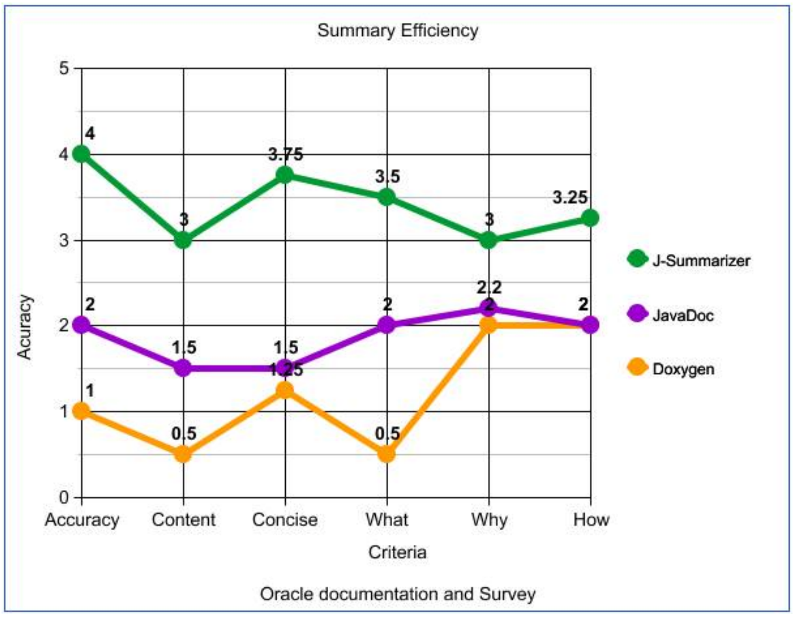
Upload Java Source code to \data folder to create XML generation !!

Generated Summary File
Open Source code Summary !

Fig 3: Summary generation

B] Result:

Graph shows comparative results of different tools. Results of our evaluations present in front of surveyor techies. Compare results with mostly used Java Doc and DoxyGen.



VI. CONCLUSION

We have presented a novel approach for automatically generating summaries of Java methods. Our approach differs from previous approaches in that we summarize the context surrounding a method, rather than details from the internals of the method. We use PageRank to locate the most important methods in that context, and SWUM to gather relevant keywords describing the behavior of those methods. Then, we designed a custom NLG system to create natural language text about this context. The output is a set of English sentences describing why the method exists in the program, and how to use the method.

REFERENCES

1. Paul W. McBurney and Collin McMillan, "Automatic Source Code Summarization of Context for Java Methods", IEEE Transactions on Software Engineering, 2015
2. Paige Rodeghero, Cheng Liu, Paul W. McBurney, and Collin McMillan, Member, IEEE," An Eye Tracking Study of Java Programmers and Application to Source Code Summarization", IEEE Transactions on Software Engineering, 2015
3. Latifa Guerrouj, David Bourque, and Peter C. Rigby, "Leveraging Informal Documentation to Summarize Classes and Methods in Context", 37th IEEE International Conference on Software Engineering, 2015
4. Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori Pollock, K. Vijay-Shanker, "Automatic Generation of Natural Language Summaries for Java Classes", IEEE Transactions on Software Engineering, vol. 32, pp. 971-987, 2014
5. Lori Pollock, "Summarizer: An Automatic Generator of NaturalLanguage Summaries for Java Classes", IEEE Transactions on Software Engineering (TSE), vol. 32, pp. 971-987, 2014
6. Emily Hill, Lori Pollock and K. Vijay-Shanker, "Automatically Capturing Source Code Context of NL-Queries for Software Maintenance and Reuse", IEEE Transactions on Software Engineering (TSE), vol. 39, pp. 521-757, 2014
7. Daniel S. Eisenberg, Jeffery Stylos, Brad A. Myers, "Apatite: A New Interface for Exploring APIs", CHI, Atlanta, GA, USA, 2010
8. D. S. Eisenberg, J. Stylos, and B. A. Myers. Apatite: a new interface for exploring apis. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '10, pages 1331–1334, New York, NY, USA, 2010. ACM.
9. B. Fluri, M. Wursch, and H. C. Gall. Do code and comments co-evolve- on the relation between source code and comment changes. In Proceedings of the 14th Working Conference on Reverse Engineering, WCRE '07, pages 70–79, Washington, DC, USA, 2007. IEEE Computer Society.
10. A. Forward and T. C. Lethbridge. The relevance of software documentation, tools and technologies: a survey. In Proceedings of the 2002 ACM symposium on Document
11. W. M. Ibrahim, N. Bettenburg, B. Adams, and A. E. Hassan. Controversy corner: On the relationship between comment update practices and software bugs. J. Syst. Softw., 85(10):2293–2304, Oct. 2012.
12. K. Inoue, R. Yokomori, H. Fujiwara, T. Yamamoto, M. Matsushita, and S. Kusumoto. Component rank: relative significance rank for software component search. In Proceedings of the 25th International Conference on Software Engineering, ICSE '03, pages 14–24, Washington, DC, USA, 2003. IEEE Computer Society.
13. M. Kajko-Mattsson. A survey of documentation practice within corrective maintenance. Empirical Softw. Engg., 10(1):31–55, Jan. 2005.

BIOGRAPHY

Suhas Mahakalkar is currently pursuing M.E (Computer) from Department of Computer Engineering, Bhivarabai Sawant College of Engineering and research, Pune, India. Savitribai Phule Pune University, Pune, Maharashtra, India -411041. He received B.E (Computer Engineering) Degree from RTTM, Nagpur University, Seven years of IT experience as a Software Developer.

Prof.A.D.Gujar received M.E Degree in Information Technology from Bharti Vidyapith University Pune, and is currently working as Assistant Professor at TSSMS BSCOER Savitribai Phule University, Pune, India