Journal of Theoretical and Applied Information Technology
30th September 2017. Vol.95. No.18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REVISING PROGRAM'S INTERNAL DOCUMENTATION FOR DEVELOPERS SUSTAINING

[1]NOUH ALHINDAWI, [2]ZIAD SARAIREH, [3]OMAR MEQDADI, [4]OBAIDA M. AL-HAZAIMEH, [5]MOHAMMAD SUBHI AL-BATAH

[1] Department of Software Engineering, Jadara University, Jordan

[2] Department of Information Technology, Emirates College of Technology, Abu Dhabi

[3]Department of Software Engineering, Jordan University of Science and Technology, Jordan

[4] Department of Computer Science, Al- Balqa' Applied University, Jordan

[5] Department of Software Engineering, Jadara University, Jordan

E-mail: [1]hindawi@jadara.edu.jo, [2]ziad.saraireh@ect.ac.ae, [3]ommeqdadi@just.edu.jo, [4]dr_obaida@bau.edu.jo, [5]albatah@jadara.edu.jo

ABSTRACT

In software engineering, the developers in order to recognize which ingredients or fragments of any software code that put into practice a definite functionality, they utilize Information Retrieval (IR) methods to mechanically spot the code that implement them. The main contribution of this paper is to study and examine the effects of skipping some textual information, namely, the internal documentations from being integrated when performing source code indexing for locating changes process purposes. In this paper, we performed two experiments over three open systems namely Qt, HippoDraw, and KOFFICE. The first experiment is done with counting the internal documentations when preprocessing the software code for locating changes process, while the other one is when skipping it. We used the standard IR measurements, Recall and Precision, and we computed the Wilcoxon signed-rank test to compare and evaluate the results. The experiments results demonstrate that not all internal documentations should be always considered in the process of locating changes. Cases in point, for the Qt system, the results show that the internal documentations improved the results of locating changes while for HippoDraw and KOFFIC systems, the internal documentations negatively impacted it.

Keywords: *Software Engineering, Information Retrieval, HippoDraw, KOFFICE*

1. INTRODUCTION

When the developers attempt to maintain or fix any issue regarding the program's code, they usually start with understanding the existing code, and then change it. This requires a fully knowledge and accurate realizing of the intended program code. Therefore, the developers have to collect code artifacts as much as they can in order to analyze the code. For instance, information extraction, lexical analysis, and artifacts filtering are considered as basic pre-process when collecting and indexing the program code artifacts [1, 2]. The internal documentation of programs which includes writing comments or notes for code fragments is considered as one of the attributes of a perfect coding [3,4]. Well-documented software is easy to update and evolve. Moreover, the literature has shown that the successful use of well written internal documentations can extensively amplify a program's understanding [5, 6, 7].

In 8 the authors concluded that the internal documentations as well as the structure of the source code aid in program understanding and therefore reduce maintenance costs. Furthermore, program internal documentations have a very efficient and large variety of possible uses, for example, it adds worthy details for program, and it is used to create the external documentation when applying reverse engineering methodology [8, 9, 10, 11, 12]. Usually, commenting any program code is considered as important component of the programming mode in order to get an understandable code to the next person who arrives along or even for a afterward programmer's

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

treatment [13, 14, 15]. The developers usually try to make their code as much as clear, well arranged, and easy to follow [3, 16]. The Existing of program internal documentations add and allow for a wide stage of unpredictability and potentially any extra information inside the source code of any system. Sometimes, the internal documentation just simply doesn't denote or indicate any meaningful thing.

Information retrieval (IR) has proved itself in understanding and realizing software's code [17, 18]. However, not all of software artifacts can give benefits or provide the developers with meaningful information about the software. In this paper we investigate the effect of adding or deleting the software internal documentation to the process of analyzing the software. Moreover, we studied the effects of performing the stemming over the software internal documentation. In the literature, a significant amount of investigation has been done on the area of analyzing and assessing software internal documentations [3, 16]. In [19] the authors concluded that if the internal documentations are too small, then they are as well unknowable. On the other side, the ones that are too extensive may possibly contain extra, repetitive, and pointless information. When the developers extract and collects the program artifacts in order to accomplish a specific maintenance tasks, they consider the internal documentation, as shown in Figure 1, as elective linguistic information that can be pulled out 20. In [21] a client study on 48 qualified programmers was done, the study results showed and confirmed that writing code with well commenting style can be easily improved and updated by developers.

Every so often, the developers comprise some indication or information in their software code internal documentation with the purpose of using it later on throughout the development task as orientation guide (i.e. Date and Copyright). In [5] the authors conclude that the efficient exploit of well written internal documentation can radically augment the process of program understanding. However, the research that focus on the quality evaluation of in-line documentation is limited [22].

The in-line software documentation has been studied and an automated approach for assessing the quality of inline documentation was presented [23]. In [20] a study was presented about the usefulness of including the in-line documentations and about performing stemming over traceability links, one of their finds is that considering in-line documentation in the indexing process assists in improving the whole process of traceability link significantly.

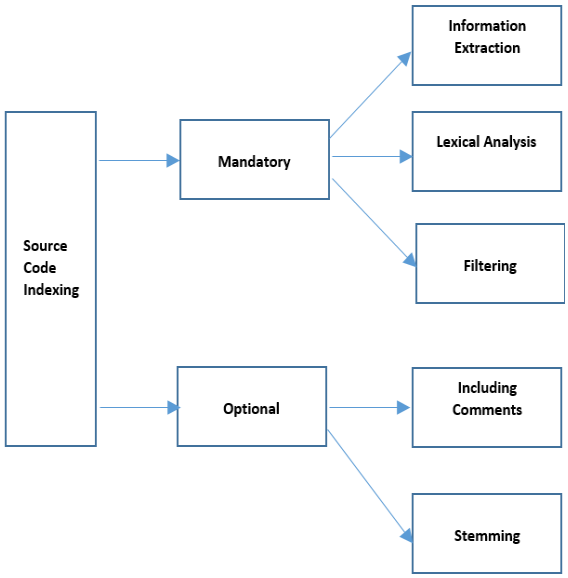

Figure 1: A feature diagram for code artifacts indexing

In [24] the software internal documentation was studied by the authors, and one of their advices was that in order to distinguish among software code and its internal documentation, an unambiguous detailed documentation or programming syntax has to be added. In [25] the practicability and the usefulness of automatically analyzing software internal documentations was studied, their aim was to spot software bugs and bad documents in the software code internal documentations. In [26] the authors concluded that not all of programmer's internal documentations are practical or helpful.

2. SOFTWARE INTERNAL DOCUMENTATIONS CLASSIFICATION

Generally, based on the aim of the internal documentations, there are three major modules for internal documentations; the documentary comments, functional comments, and descriptive comments. More details are presented on the following sub-sections about each module [11].

2.1 Functional Internal Documentations

When the developers want to add new functionality or new feature, this type of documenting is used always. The core goal of writing this documentation is only to describe the added functionality. In this type of documentation,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the developers do not explain the entire program. Such an examples of functional documentations are feature addition, bug description, and to do. To improve any software code understanding, the developers should add or assign this type of documentations in a standard and reasonably way to the fragments of code [26].

2.2 Documentary Internal Documentations

The name of this type of internal documentations is called documentary due to its usage in documenting the process of development any software project. Moreover, this type internal documentation holds important information about the project fragments as we see in Figure 2, for instance, version number, author's name, and program idea.

```
Revision: 1.1          PCM-DAS08 and -16S/12 are supported.
                       Sorting routine inserted.
                       Set-files are read in and card as well as
                       amplification factor are parsed.

         1.1.1         Standard deviation is calculated.

         1.1.2         Median is output. Modal value is output.

         1.1.4         Sign in Set-file is evaluated.
                       Individual values are no longer output.
                       (For tests with raw data use PCMRAW.EXE)

To do:                 outliers routine to be revised.
                       Statistics routines need reworking.
                       Existing Datafile is backed up.
```

Figure 2: An example about documentary internal documentations

Remaining the program in a shape that is easy to interrupt or update is consider as a main situation of usage this type of internal documentations. Moreover, this type of internal documentations can also hold a superior explanation for the equipment needed. Giving and supporting the new developers with a brief summary about the program before altering any fragment of software [25].

2.3 Descriptive Internal Documentations

Writing the internal documentations of any software code in a very good shape needs this type of internal documentations to shows up a lot. However, this internal documentation does not require to be found for each line of code or for each statement. Starting up fragment code and the standard expression are some examples where the explanatory internal documentations should be added. Figure 3, shows an example for this type 25.

```
r = new Regex("href          #This looks  for the string 'href'
\\s*=\\s                     #followed by whitespaces, '=', ws
(?:\"(?<1>[^\"]*)\"          #a ':', + a group in '"', no '"' in it
|                            #or
(?<1>\\S+))",                #a group followed by non-spaces
```

Figure 3: An example about descriptive internal documentations

2.4 Internal Documentations Samples

Here, we give some examples for internal documentations for the systems we investigated in this paper. We note that the internal documentations for HippoDraw system and Koffice system as we see in Table 2 and Table 3, has less standardized documenting style when compared with QT system.

Table1: Internal documentations for QT systems (4-Examples)

Function Name	Internal Documentations
setOpenFileName	"! options selectedFilter fileName openFileNameLabel selectedFilter options filename"
blendComponent	"! shadow gets a color inversely proportional to the alpha value then do standard blending"
findFiles	"! filePattern fileNameComboBox directory directoryComboBox allFiles directory matchingFiles file"
createLayout	"! fileLayout QHBoxLayout directoryLayout QHBoxLayout mainLayout QVBoxLayout “

For instance, as we see in the Table1 that the internal documentation, for function two (mousePressEvent) does not support the developers with any meaningful information and it is too tiny.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 2: Two examples for internal documentations for HIPPODRAW system

Function Name	Internal Documentations
setCutRange	"setCutRange projector * @bug @@@@@@ This needs fixing for two dimension functions"
mousePressEvent	"m_plotter"

Table3. Two examples for internal documentations for KOFFICE system

Function Name	Internal Documentations
createShape	"Factory shape factory path reset transformation that might come from the default shape / creates a shap from the given shape id"
saveImage	"Format NULL ret pixmap Save the image"

3. EXPERIMENTAL SETUP AND DATASET

As mentioned before, our main contribution in this paper is to come back with a solution for the following inquiry; should program internal documentations always have be considered when preprocessing program code for locating the needed changes locations?.

The hypothesis here that our experiments are built on top revolves around the factor hypothesis that not all of program internal documentations have to be included when preprocessing program's code when locating the needed changes locations for the developers. We conducted two experiments for changes allocating using IR technique, namely the Latent Semantic Analysis (LSA) [27, 28]. The first experiment is done with counting the internal documentations when locating changes and the other one done with skipping the internal documentations. Moreover, the stop-list removal and stemming were executed with the two experiments. In our evaluation, we used the Wilcoxon signed-rank by computing the p-value to inspect whether the diversity in terms of usefulness for the two approaches we have, which are counting the internal documentations and skipping the internal documentations from being preprocessed when locating any changes the developers need.

Wilcoxon signed-rank test (One-Tail) is non-parametric test and it takes as an input two ranked list created from the two different locating changes techniques, we assume that ranks unreservedly hold the entirety efforts required by developers when performing any maintenance activity. In our test, the significance level $\alpha = 0.05$ was chosen, and the production of the test is a p-value, which can be studied as follows.

If the p-value is less than α, then the difference in ranks formed by locating changes technique is statistically considerably lesser than the ranks formed by the other technique. Or else, if the p-value is bigger than α, then both of the two studied techniques produce approximately comparable results. The following are the null and alternative hypothesis that were formulated in order to test whether counting or adding the internal documentations has a higher effectiveness measure than when skipping them when locating the developers needed changes over the software artifacts or not.

H0: There is no statistical significant difference in the measure of effectiveness between Adding the internal documentations and when skipping them.

H1: Adding the internal documentations implied higher effectiveness than skipping them.

For the conducted experiments, we use the same Dataset we have in our previous work [17] and [29]; we have three open source systems, namely, QT, HippoDraw, and KOFFICE. As we see in Table 4 (HippoDraw and QT), for each system we used LSA to rank the relevant methods for 11 changes. The changes were selected based on the bug reports present in the online system documentation for all of three systems. The results analysis is shown in Figure 4.

All the below Figures, show that considering internal documentations in the locating changes

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

process has a significant effect on the recovery effectiveness for some systems.

Table4: HIPPODRAW and QT systems features

HippoDraw Feature	QT Feature
Update Font Size	Change Font Settings
Update Font Style	Add New Font
Change Zoom Mode	Reset Font Size
Change Printer Settings	Change Password
Add Item	Change RGB
Remove Item	Create Menu
Modify Mouse Property	Delete Menu
Modify Cut Color	Create Action
Reset Representation	Delete Action
Add New Display	Search
Change Axis Modeling	Sketch Polygon

4. RESULTS AND DISCUSSION

Now, we will discuss the results for each system we experimented individually. For the first system (QT), as shown in Figure 4, counting and adding the internal documentations has enhanced the outcome. One of the most reasons at the rear that is the QT developers used a regular style when writing the internal documentations for QT code. As shown in the figure, the average for the two standard IR measurements we used (Recall and Precision) is higher when counting the internal documentations.

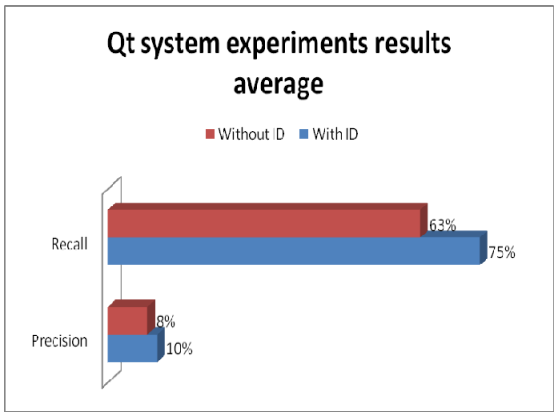

Figure 4: QT-System Experiments Results Average. ID Refers To Internal Documentations.

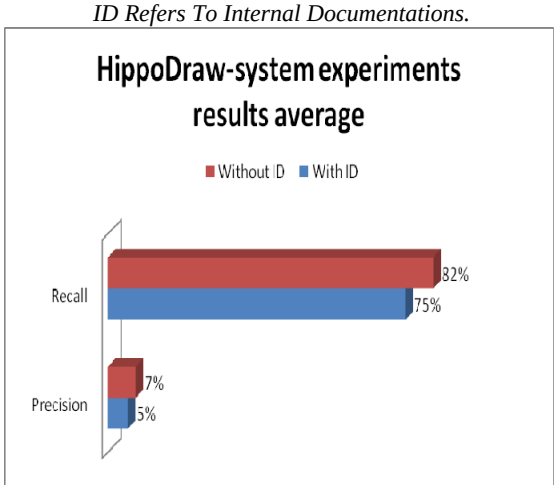

Figure 5: Hippodraw -System Experiments Results Average.

As shown in Figures 5 and 6, for HippoDraw and KOFFICE systems the results are opposite to QT results. In other words, the counting of internal documentations while preprocessing the software artifacts has negative impact when locating any needed changes. The main reason behind these results is the stuffing of internal documentations in both systems; the internal documentations for both HippoDraw and KOFFICE systems are poorly written and not reasonably added for most code fragments. Therefore, our findings go with the fact "a useful comment always follows some basic rules of style." [11]. That's it; counting or skipping the internal documentations depends on the contents of the internal documentations. For instance, the internal documentations that holds invaluable information such as author's copyright remark, yet also with removing the stop list words, a number of terms still reside indexed and unenthusiastically have an effect on locating changes process results for a number of software systems. Moreover, as we see in Table 5, for the three systems, we studied the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

distributions of internal documentations compared with lines of code of each system. In other words, the density of internal documentations over the code is computed for each system.

Table 5 shows that the developers of QT system used a well structured or a good standardized style when they wrote the internal documentations. This fact is due to the largest percentage of internal documentations of QT system as shown in Table 5, which means that the QT code is documented enough. As a result, this was reflected completely on the outcomes of locating changes process.

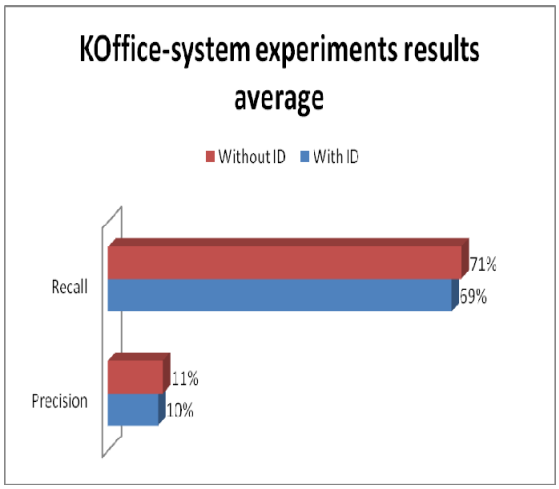

Figure 6. KOFFICE-System Experiments Results Average.

Table5. Internal documentations-Density (%) Line of Code (LOC) for the three systems

Systems	Internal documentations-Density (%) Line of Code (LOC)
QT	18
KOFFICE	12
HIPPODRAW	11

Moreover, we compared the results of locating all of the needed changes for the three systems. As shown in Figures 7 and 8, for the QT system, 90% of the inquiries were best retrieved when counting the internal documentations and the rest 10% provided the same ranks for the related functions, either when counting or skipping the internal documentations from the system corpus. However, for the HippoDraw system, the outcomes are diverse than QT outcomes; 70% of the inquiries were not affected by counting the internal documentations. The internal documentations did not affect the retrieving process. While the rest 30% of the inquiries have been recovered more correctly when counting the internal documentations. As a conclusion, and based on the manual inspection of HippoDraw system and on the results, the developers of HippoDraw poorly documented the code fragments. Moreover, the internal documentations of HippoDraw itself stated previously, doesn't have a lot of significant information that considered as important for location process.

The KOFFICE system results are slightly diverse than the other two system. As shown in Figures 7 and 8, skipping the internal documentations enhanced 50% of the inquiries while counting them only enhanced 20%. Conversely, 30% of the inquiries relevant methods ranks were not affected when counting or skipping the internal documentations. That's mean that the developers poorly documented the related methods of the take inquiries which reflected negatively on the outcomes of inquiring the system for locating purposes. Moreover, as shown in Figures 8 and 9, we have calculated the recall and precision for all the systems.

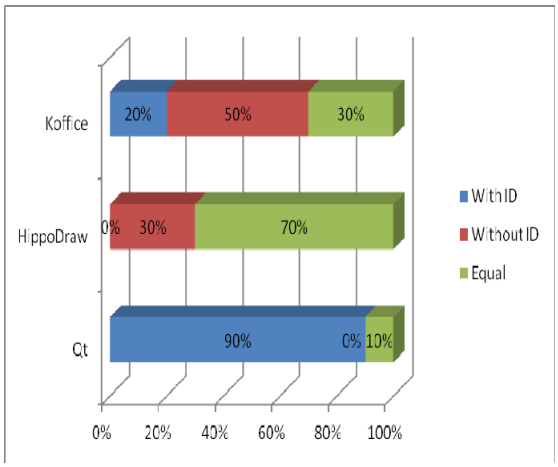

Figure 7: Ranking Comparison For All Relevant Methods Of All Taken Systems Queries

As shown in Figure 7, three cases taken, the red color shows the percentage of relevant methods that best answered when including the internal documentations, the yellow color shows the percentage when excluding the internal documentations, and finally the blue color shows

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the percentage when including and excluding the internal documentations do the same.

As shown in figure 8, three cases taken, one with including all internal documentations, and one without including any internal documentation, and the final one, is when including the internal documentations except the bug internal documentations.

As shown in Figure 9, three cases taken, one with including all internal documentations, and one without including any internal documentation, and the final one, is when including the internal documentations except the bug internal documentations.

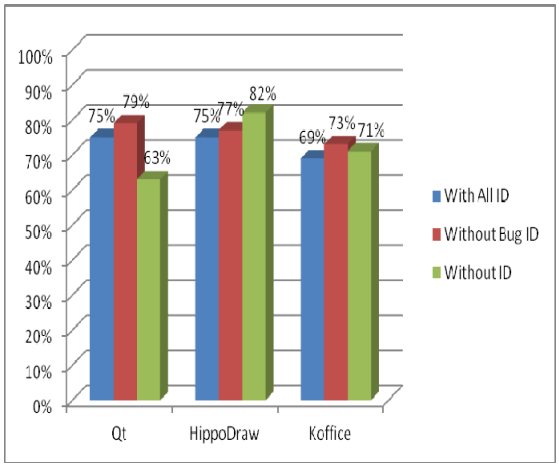

Figure 8: Recall Results For The Relevant Methods Of All Taken Systems Querie

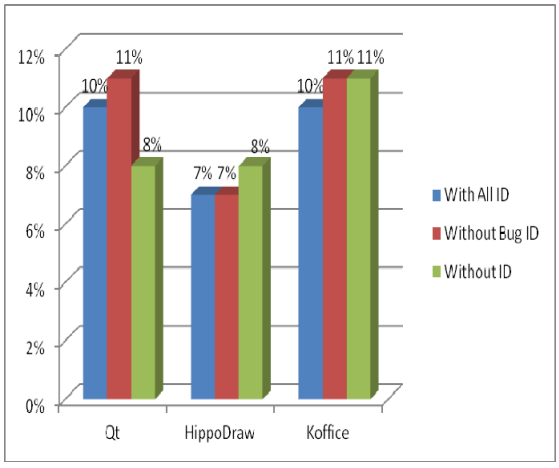

Figure 9: Precision Results For The Relevant Methods Of All Taken Systems Queries.

Moreover, we computed the Wilcoxon signed-rank test to examine if the differentiation in terms of effectiveness for the two approaches is statistically significant. We computed it based on the total effort measure (Σ EM) dependent variable. The null hypothesis is that there is no statistical significant difference in the measure of effectiveness between Adding the internal documentations and when skipping them.

As stated before, the alternative hypothesis is that adding the internal documentations implied higher effectiveness than skipping them. Our results were found to be statistically significant. The p-value is lower than $\alpha = 0.05$, it was actually less than 0.0001. This permits for declining and rejecting the null hypothesis we have.

5. CONCLUSION

Examining the related artifacts or fragments of software source code that related to particular changes is considered as a crucial phase during any software maintenance process. Moreover, the internal documentations of program code play a major role in guiding the developers while locating the related code fragments that need to be altered. In this paper we present an empirical study to come back with answer for the inquiry, should internal documentations of source code must be always considered when stuffing the software source code for locating changes or not?.

To answer this inquiry, we have performed two experiments over three open source systems, namely QT, HippoDraw, and KOFFICE. The first experiment is done with counting the internal documentations while the other one with skipping it. The results show that not all internal documentations should be included or considered. For instance, for the QT system, the results show that the internal documentations are granted in a more consistent and standardized way when compared to those for HippoDraw and KOFFIC systems.

Additionally, the results show that for HippoDraw system, the internal documentations have an insignificant role in enhancing the results locating changes.

Therefore, counting or skipping the internal documentations while staffing any software code for maintenance issues, is mainly dependent on how much the internal documentations of any system are written in a standard and regular way, whether the it is up to date or not, and if it is

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

contains a meaningful and helpful information. As a future work, we plan to study more systems from different domains.

Moreover, we plan to build a tool that can automatically describe and translate any form of internal documentations (i.e., UML, Logos, diagrams, and flowcharts) into a standardized human language form.

A limitation for our research is the number of taken systems; we plan to conduct more experiments with different programming language like Java and Python. In future work, we plan to compare between the comment's lengths between different programming g languages.

REFERENCES

[1] McMillan C. Portfolio: finding relevant functions and their usage. in 33rd International Conference on Software Engineering (ICSE). Waikiki, Honolulu, HI, USA: ACM. 2011 May, pp .111-20.

[2] Dragan N, Collard M L, Maletic J I. Reverse Engineering Method Stereotypes. in 22nd IEEE International Conference on Software Maintenance. 2006 Sept, pp .24-34.

[3] BinkleyD, Lawrie D, Information Retrieval Applications in Software Development, in Encyclopedia of Software Engineering: Taylor & Francis LLC. 2010.

[4] Brooks R, Towards a theory of the comprehension of computer programs. International Journal of Man-Machine Studies, 1983 June, 18(6), pp. 543-54.

[5] Dit B, et al, Feature location in source code: a taxonomy and survey. Journal of Software Maintenance and Evolution: Research and Practice, 2011 Jan, 25(1), pp. 53 - 95.

[6] HindleA, German D M. SCQL: A Formal Model and a Query Language for Source Control Repositories. in 2nd International Workshop on Mining Software Repositories (MSR).. St. Louis, Missouri ACM Press: New York NY. 2005 J?uly, 30(4), pp .1-5

[7] Poshyvanyk D. Using information retrieval to support software maintenance tasks. in IEEE International Conference on Software Maintenance (ICSM).2009 Sep, pp .453-56

[8] Elshoff JL. Marcotty M, Improving computer program readability to aid modification. Commununication of the ACM, 1982 Aug, 25(8), pp. 512-21.

[9] Antoniol G, et al, Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering,Oct 2002, 28 (10), pp. 970-83.

[10] Maletic JI, Kagdi H. Expressiveness and effectiveness of program comprehension: Thoughts on future research directions. in Frontiers of Software Maintenance (FoSM). 2008.

[11] Spuida B, The fine Art of Commenting, in Tech Notes, general Series, S.W. Wrangler, Editor 2002.

[12] Holmes R, Murphy G C, Using structural context to recommend source code examples, in 27th international conference on Software engineering (ICSE), ACM: St. Louis, MO, USA. 2005 pp. 117-25.

[13] Haiduc S, Aponte J, Marcus A. Supporting program comprehension with source code summarization. in 32nd ACM/IEEE International Conference on Software Engineering (ICSE).. Cape Town, South Africa: ACM.2010, pp .223-26

[14] Hill E,. Pollock L, Vijay-Shanker K. Improving source code search with natural language phrasal representations of method signatures. in 26th IEEE/ACM International Conference on Automated Software Engineering.. IEEE Computer Society. 2011, pp .524-27

[15] MaleticJI. Marcus A. Using latent semantic analysis to identify similarities in source code to support program understanding. in 12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI) 2000, pp .46-53.

[16] Maletic JI, Marcus A. Support for Software Maintenance Using Latent Semantic Analysis. in 4th Annual IASTED International Conference on Software Engineering and Applications (SEA). 2000.

[17] Alhindawi N, Improving Feature Location by Enhancing Source Code with Stereotypes. in 29th IEEE International Conference on Software Maintenance (ICSM). Eindhoven, The Netherlands.2013.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[18] Maarek YS, Berry D M, Kaiser G E, An Information Retrieval Approach for Automatically Constructing Software Libraries. IEEE Transactions on Software Engineering, 1991 Aug, 17(8), pp. 800-13.

[19] Cleary B, , An empirical analysis of information retrieval based concept location techniques in software comprehension. Empirical Software Engineering, 2009 Feb, 14(1), pp. 93-30.

[20] Mahmoud A, Niu N. Source code indexing for automated tracing. in 6th International Workshop on Traceability in Emerging Forms of Software Engineering.. Waikiki, Honolulu, HI, USA: ACM.2011.

[21] Woodfield SN, Dunsmore H E, Shen V Y. The effect of modularization and comments on program comprehension. in 5th IEEE International Conference on Software Engineering (ICSE).. San Diego, California, USA: IEEE Press.1981, pp .215-23.

[22] Padioleau Y, Lin T, Yuanyuan Z. Listening to programmers-Taxonomies and characteristics of comments in operating system code. in 31st IEEE International Conference on Software Engineering (ICSE) 2009 May, pp .331-41.

[23] Khamis N, Witte R, Rilling J. Automatic quality assessment of source code comments: the JavadocMiner. in 15th international conference on Applications of natural language to information systems. Cardiff, UK: Springer-Verlag.2010 ,6177 pp .68-79.

[24] SchreckD, Dallmeier V, Zimmermann T, How documentation evolves over time, in Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting, ACM: Dubrovnik, Croatia. 2007, pp. 4-10.

[25] Tan L, Yuan D, Zhou Y. Hotcomments: how to make program comments more useful? in 11th USENIX workshop on Hot topics in operating systems.. San Diego, CA: USENIX Association.2007

[26] Howden WE, Comments Analysis and Programming Errors. IEEE Transition on Software Engineering, 1990 Jan, 16(1), pp. 72-81.

[27] Deerwester S, Indexing by Latent Semantic Analysis. Journal of the American Society of Information Science, 1990, 41(6), pp. 391-07.

[28] Suchithra M, A Survey on Different Web Service Discovery Techniques, Indian Journal of Science and Technology, 2015 July, 8 (15), pp. 1-5.

[29] Alhindawi, Nouh, Obaida M. Al-Hazaimeh, Rami Malkawi, and Jamal Alsakran. "A Topic Modeling Based Solution for Confirming Software Documentation Quality." INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS 7, no. 2 (2016): 200-206.