DOI: 10.21917/ijsc.2021.0342

INFORMATION RETRIEVAL BUG LOCALIZATION WITH WAVELETS ANALYSIS

S. Zouairi and M.K. Abdi

Department of Information and Technology, University Oran 1, Algeria

Abstract

Nowadays, maintaining an oversized and evolving computer code involves longer and value for the project team. In software maintenance, a bug report is employed to seek out a fault location. Once a bug report is received, it is suiTable.to automatically denote out the files that developers should change to repair the bug. This work develops a brand-new information retrieval (IR) system that allows representing textual data by signals. This new way of implementing gives us the chance to use various mathematical tools from the signal theory like Wavelets Transforms, unused nowadays within the field of IR. This paper proposes Wavelets Transforms for Bug Localization (WTBugLoc), a mathematical approach of IR-based bug localization using wavelet techniques. The results of the conducted experiments on the SWT (Standard Widget Toolkit) Eclipse project confirm the effectiveness of the proposed approach. The experiments also show that WTBugLoc outperforms method using the Vector Space Model (VSM).

Keywords:

Bug Fixing, Information Retrieval, Bug Report, Haar Transform, Software Maintenance

1. INTRODUCTION

During the 1960s and 1970s, the event of huge software systems leaded to several difficulties and since the term "software crisis" dates from that point. The crisis manifests in several ways including projects exceeding the estimated costs for development, the late delivery of software, and the poor quality of the delivered software. Currently, software continues to be a critical element in most large-scale systems and many companies must deal with a software crisis.

During the life cycle of software processing, the engineering team is overwhelmed with bugs reported. The major goal of the engineering team is to deliver a high-quality product, while keeping bugs down. Engineering teams often feel they are in an epic struggle to build the perfect product while dealing with ever-changing requirements, software complexity, framework changes, and evolving infrastructure. But, with the expanding size of programming and the restricted advancement assets, it is frequently unavoidable to deliver programming frameworks without bugs. A bug is an anomaly in the software product that causes the software to perform incorrectly or to behave in an unexpected way [1].

When a bug occur in a software, the engineering team reports it in a document called a bug report and logs it in a bug tracking system, such as Bugzilla [2]. Before validating a bug report, a various stages of quality checks are passed. This include, checking for duplicity, validity, and completeness. After that, the bug is addressed to a developer, who refers to the information in the bug report to identify the source files, which need to be altered, in order to solve the issue in the bug report. Bug reports typically consist of various fields which contain description of the bug, screen shots or snapshots of an error message, version, stack traces, etc., an example is shown in Table.2.

The developer, who is responsible for this bug, has to find the root cause of the bug in the source code and then fix the root cause. Doing this process manually can take 30%-40% of the total time needed to fix a problem [3]. This is a painful process, especially for big software projects with hundreds of thousands of source files. As a result, the bug fixing is time increasing, along with maintenance cost of the project. Although the bug fixing task constitutes sub-tasks such as understanding the bug, validating the bug, locating the cause of the bug, and finally fixing the bug, it is the process of finding the cause of the bug (bug localization) that consumes most of the time of the developer [4]. The need for automated tools or approaches, which perform bug localization, has become a requirement.

Recent bug localization techniques are often classified into two broad families–spectra based and information retrieval (IR) based [5]. While spectra-based techniques depend upon on execution traces of a package, IR-based techniques analyze shared vocabulary between a bug report (i.e., query) and therefore the project source for bug localization [5].

The basic idea in these IR approaches is that a bug report is treated as a query, and the source files to be scanned in a software product are the set of documents. IR techniques then rank documents by expected relevance and return a ranked list of candidate source files that would cause the bug.

Many of the existing IR-based bug localization methods are proposed to automatically look for relevant files with regard a bug report [6] - [16].

These IR methods are built around a similarity function. This function takes a query and a document as its arguments and generates a single score which represents the relevance of the query to the document [17].

Other techniques use vectors to represent documents and queries, where the vector space contains one dimension for each of the terms found in the document set. The contribution of each dimension in the document vector is generated by counting the appearances of the associated term in the document. The similarity function simply applies weights to the query and document vectors and compares them to generate a score based on their likelihood of relevance [17]. The process of converting the documents into vectors, take only into account the number of times each term appears in the documents and disregard the positional information.

Yet in many applications, there is a necessity to process data that's inherently of vector form. As an example, Speech and audio processing, Image and video processing, Biomedical imaging are usually represented as 2D or 3D vector fields in 2D or 3D space, while images with multiple spectral components may be considered to be 2D fields of multidimensional vectors.

Wavelets transform are functions providing certain mathematical purposes and representing data or other functions by vector (ie. signal). A Haar wavelet is the simplest sort of wavelet. In discrete form, Haar wavelets are associated with a mathematical process called the Haar transform [18]. Due to its low computing requirements, the Haar transform has been mainly used for pattern recognition and image processing [19]. In IR bug localization, wavelet techniques haven't been thoroughly puzzled out. Wavelet analysis has been found to be extremely useful for software code clone detection [20], clone detection in image processing [21], data reduction methods to handle potentially large and sophisticated nonstationary data curves [22] and in integral text information search [23], and in documents information retrieval [24].

Many reasons justify the usefulness of wavelets techniques. For the Computational Complexity viewpoint wavelet transform only needs $O(N)$ multiplications [25]. Applying wavelet transform to the source code project reduces the quantity of information to be treated by an element of two with each transform allowing us to use fewer resources.

- To the authors knowledge, wavelets transform have not been used for IR-based bug localization.
- The main aim of this study is to fill this research gap. To attain this objective, this study will try to answer the subsequent main research question:

RQ: what is the effectiveness of using wavelets transform for bug localization?

In this paper, we present the evaluation of the usefulness of wavelet transformations in bug localization. For this, we implement a tool called WTBugLoc a brand-new IR system that ranks buggy files and allows representing textual data by signals using Haar Transform. The remainder of the paper is organized as follows. Section 2 presents background and therefore the most relevant related works, section 3 includes an overview of wavelets and describes the proposed approach and also the followed methodology to hold out the study, results and discussions are presented in sections 4, Finally, section 5 concludes this work.

2. STATE OF THE ART

In general, automated bug localization approaches can be divided narrowly into two categories: dynamic and static approaches. In the dynamic approach, the semantics of the program and its execution information with test cases, i.e., pass/fail execution traces are used. These approaches are divided into Spectrum based fault localization and Model-based fault localization. Saha et al.[10] have proposed a model that uses this approach by using a tool named BLUiR (Bug Localization Using information Retrieval) which explores source code's structural information such as comments, names of classes, methods, and variables, to improve localization accuracy.

The static approach focuses only on the source code and the bug reports information. These static approaches can be further classified into two groups: program analysis based and IR-based.

To localize bugs the program analysis-based approach uses predefined bug patterns. A model called FindBugs used this approach was proposed by Hovemeyer et al. [26].

IR or Machine Learning techniques like TF-IDF, LSA, LDA, VSM, rVSM, and Naive Bayes [27], can be categorized in the second type of static approach. These approaches perform Learning-To-Rank IR problem for the bug localization problem.

Rao et al. [6] use the VSM methods for bug localization which explore the cosine similarity technique between document's terms. Youm et al. [14] propose a tool called BLIA (Bug Localization with Integrated Analysis), they utilize the content and stack traces in bug reports, structured information of source files, and source code change histories. In their work, they combined a method to integrate all analyzed data to increase localization accuracy. Zhou et al. [16] proposed BugLocator, a tool that can automatically search for relevant buggy files based on initial bug reports. This tool uses the revised Vector Space Model (rVSM) to rank all source code files based on an initial bug report.

This article proposes a tool called WTBugLoc a new IR system that ranks buggy files and allows representing textual data by signals. This new way of presentation will allow later, applying numerous mathematical tools from the theory of the signal such as Wavelets Transforms, rarely used nowadays in the field of IR.

3. BACKGROUND

We begin by giving the context of the study by presenting all concepts related to it (wavelet transform, file preprocessing and bug localization with Haar transform)

3.1 HAAR WAVELETS TRANSFORM

Haar functions are used from 1910 after the mathematician Alfred Haar introduced them. The Haar wavelets is one amongst the earliest samples of what's is known now as a compact, dyadic, orthonormal wavelet transform [28] A Haar wavelet is the simplest type of wavelet, for more detail on wavelets transform see [29]-[31]. In discrete form, Haar wavelets are associated with a mathematical process called the Haar transform. The Haar transform is a prototype for all other wavelet transforms.

This section shows how the Haar transform are often used for bug localization, and introduces the essential notions related with the Haar transform, which is developed in section 3.3.

Throughout this work, a vector is mapped as a discrete signal. A discrete signal is a function of time with values occurring at discrete instants. Generally, a discrete signal is expressed in the form $(f=f_1,f_2,\ldots,f_N)$, where N is a positive even integer which is the length of F. Intuitively, f_1, $f_2,\ldots,f_N$ the values of F are real numbers. These values are typically measured values of an analog signal g, measured at the time values $t=t_1,t_2,\ldots,t_N,$. That is, the values of F are:

$$f_1=g(t_1),\ f_2=g(t_2),\ldots,\ f_N=g(t_N) \quad (1)$$

Like all wavelet transforms, the Haar transform decomposes a discrete signal into two subsignals of half its length. One subsignal is a running average or trend; the other subsignal is a running difference or fluctuation.

First of all, we begin by examining the trend subsignal. The first trend subsignal $a^1=(a_1,a_2,\ldots,a_{0.5N})$, for the signal F is computed by taking a running average in the following way:

Its first value a_1, is computed by taking the average of the first pair of values of $F = 0.5(f_1 + f_2)$, and then multiplying it by $\sqrt{2}$ that is $a_1 = \frac{(f_1 + f_2)}{\sqrt{2}}$. Similarly, its next value $a_2 = \frac{(f_3 + f_4)}{\sqrt{2}}$. Continuing in this way, all of the values of a^1 are produced by taking averages of successive pairs of values of F, and then multiplying these averages by $\sqrt{2}$.

A precise formula for the values of a^1 is

$$a_m = \frac{(f_{2m-1} + f_{2m})}{\sqrt{2}} \quad (2)$$

For $m = 1, 2, ..., 0.5N$

For example, suppose F is defined by eight values, say F=(4,6, 10,12,8,5,5); then its first trend subsignal is

$$a^1 = (5\sqrt{2}, 11\sqrt{2}, 7\sqrt{2}, 5\sqrt{2}) .$$

The other subsignal is called the first fluctuation (difference). The first fluctuation of the signal F, which is denoted by d^1=(d_1, d_2,…,$d_{0.5N}$), is computed by taking a running difference in the following way. Its first value d_1 is calculated by taking half the difference of the first pair of values of $F = 0.5(f_1 - f_2)$ and multiplying it by $\sqrt{2}$, that is $d_1 = \frac{(f_1 - f_2)}{\sqrt{2}}$. Likewise, its next value $d_2 = \frac{(f_3 - f_4)}{\sqrt{2}}$. So all of the values of d^1 are produced according to the following formula:

$$d_m = \frac{(f_{2m-1} - f_{2m})}{\sqrt{2}} \quad (3)$$

For $m = 1, 2, ..., 0.5N$

For example, for the signal F=(4,6,10,12,8,5,5) considered above, its first fluctuation d^1 is $(-\sqrt{2}, -\sqrt{2}, \sqrt{2}, 0)$.

3.1.1 Haar Transform Multiresolution:

The Haar transform is performed in several decompositions (stages, or levels). The first decomposition is the mapping H_1 defined by:

$$F \xrightarrow{H_1} (a^1 | d^1) \quad (4)$$

From a discrete signal F to its first average a^1 and its first difference d^1. Then the Haar transform 1-level for de signal F=(4,6,10,12,8,5,5) is like showed above:

$$(4,6,10,12,8,5,5) \xrightarrow{H_1} (5\sqrt{2}, 11\sqrt{2}, 7\sqrt{2}, 5\sqrt{2} | -\sqrt{2}, -\sqrt{2}, \sqrt{2}, 0)$$

By applying the Eq.(2) and Eq.(3) to each two successive values we get the following 2-level for de signal F which is $F' = (16,12 | -6,2)$. Recursively repeating this process, we get the full decomposition of F as shown in the Table.1

Table.1. Example of the Haar wavelet decomposition.

Level	Averages	differences
1	$(5\sqrt{2}, 11\sqrt{2}, 7\sqrt{2}, 5\sqrt{2})$	$(-\sqrt{2}, -\sqrt{2}, \sqrt{2}, 0)$
2	(16,12)	(-6,2)
3	$(14\sqrt{2})$	$(2\sqrt{2})$

If the length of F is $N = 2^P$ (in the showed example $P = 3$), we can continue this process up to level P, at which there is one average and one detail coefficient. Finally the wavelet transform of F is $(14\sqrt{2}, 2\sqrt{2}, -6, 2, -\sqrt{2}, -\sqrt{2}, \sqrt{2}, 0)$, this process is called multiresolution analysis.

3.2 PROPOSED METHODOLOGY

To use the SWT Eclipse project for the empirical evaluation, some data preprocessing is needed for both the source code files and the bug reports, which consists of five steps: files assembling, corpus creation, indexing, query construction, and retrieval and ranking. The Fig.1 shows the overall structure of WTBugLoc for bug localization.

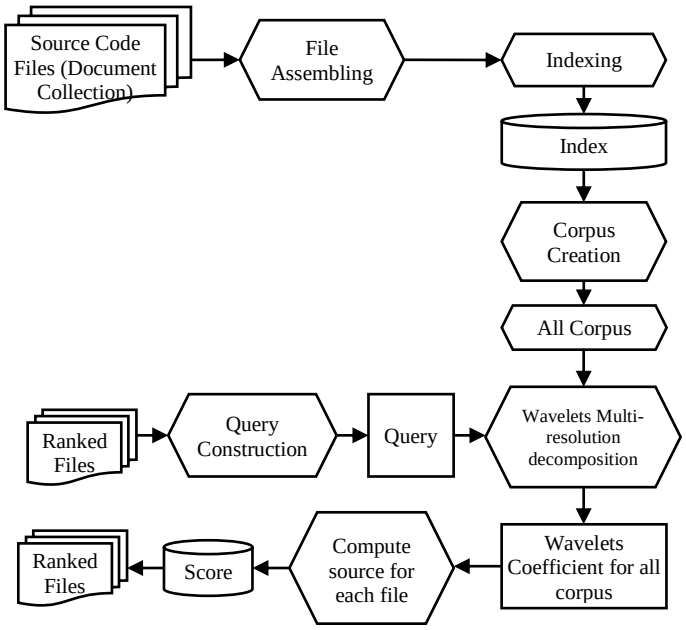

Fig.1. Structure of WTBugLoc

In this section, an example is given to illustrate the IR bug localization approach. The Table.2 shows a real bug report (ID: 102756) for SWT version 3.1. Once this report is received, the developer needs to locate relevant files among more than 2000 SWT source files to fix this bug. This bug report (including bug summary and description) contains many words such as GtkComboBox, GtkCombo and Combo. However, SWT source files contain a file called 'GTK.java' with a good match between the bug report and it source code.

Table.2. A bug report for bug id 102756

Bug Details	Report
Bug ID	102756
Summary	Use GtkComboBox instead of GtkCombo
Status	RESOLVED FIXED
Reported	2005-07-05 13:38 EDT, David Graham
Product	Platform

Component	SWT
Version	3.1
Description	Incorrect combo rendering Native combo rendering
Fixed	2006-06-29 13:48:22 EDT
Fixed files	Eclipse\git\eclipse.platform.swt\bundles\org.eclipse.swt\Eclipse SWT PI\gtk\org\eclipse\swt\internal\gtk\GTK.java

3.2.1 Module Description:

Files Assembling: to optimize the approach, the we looked for all files with extension *java* in tree source code and we copied them in a single directory (location), next step proceed for deleting all comments from each source code file, doing so will accelerate the bug finding approach.

- *Corpus Creation:* by performing word tokenization which is a helpful tool from Natural Language Processing- for further detail see [32], a lexical analysis is done for each source code file and create a vector of lexical tokens. Some tokens, such as keywords (e.g., int, double, char, etc.), separators, operators are common to all programs and are removed. English "stop words" (e.g., 'a', 'the', etc.), punctuations, whitespaces, tabulations and newline characters are also removed, which reduces the amount of converted data considerably.
- *Indexing:* After the corpus is created, all the files in the corpus are indexed by assigning to each file its length (number of tokens), then a Top-Down index ordering is applied to the whole collection of SWT source code files.
- *Query Construction:* Specific tokens are chosen from the bug report to construct the query vector, for using it to search for relevant files in the indexed source code corpus. The vector query is obtained by performing word tokenization from the bug title and description as same as in the corpus creation step.
- *Retrieval and Ranking:* Retrieval and ranking of relevant buggy files is based on wavelet transform of the query and each of the files in the corpus, then computing the score for each file in the corpus. A file in document collection, is viewed as the bag of words model [33], ordering of the terms in a document is ignored but the number of occurrences of each term is material. Applying Haar transform to a file lead to a vector with components corresponding to each term in the query. In this work, the step of computing the score is inspired from section 6.3.3 of [33]. The score of a file is the sum, over all vector components resulted from the Haar transform divided by the length of the vector.

3.3 BUG LOCALIZATION WITH HAAR WAVELET TRANSFORM

This section describes the algorithm to find the most relevant files according to the query generated by a bug report.

Pseudo Algorithm

Step 1: Create token vector for each file

For j=1, m compute $vect\_File_j[w_1, w_2,..., w_i,..., w_n]$, where w_i is a token in $File_j$

Step 2: Create pertinence vector for vector query's tokens for each file

For j=1,m

compute $vect\_ReqFile_j[Tfwq_1, Tfwq_2,..,Tfwq_i,..,Tfwq_n]$, where:

$$Tf_{wq_i} = \begin{cases} Tf & if\ w_i \in query\ tokens\ vector \\ 0 & else \end{cases}$$

where Tf - Term frequency of query token wq_i in $File_j$

Step 3: Apply multi-resolution Haar wavelet transform

For j=1, m

Compute $Haar\_Transform(vect\_ReqFile_j)$ using Eq.(2) and Eq.(3)

Step 4: Compute the score for each file

For j=1, m

$Score\_File_j = \sum|Haar\_Transform(Vect\_ReqFile_j)Components|$

In this paper, Python 3.6 is employed to create WTBugLoc. The language has many programming constructs that help in building both large scale and small-scale software. It has a broad range of ordinary library and toolkit which may be used for a spread of purpose. NLTK (Natural Language Toolkit), this toolkit is one in every of the foremost powerful NLP (Natural Language Processing) libraries, which contains packages to make machines understand human language and reply thereto with an appropriate response. Tokenization, Stemming, Lemmatization, Punctuation, Character count, word count is a number of these packages. PyWavelets is free and Open Source wavelet transform software for the Python programming language.

4. RESULTS AND DISCUSSION

This section describes the evaluation results of WTBugLoc while performing bug localization on the 1932 files of SWT eclipse project, Table.3 shows the percentage of Top N, with both the Haar Wavelets Transform (WT) and the VSM.

Table.3. Ranking relevant files based on WT and VSM on SWT eclipse project

Models	**Top 5 %**	**Top 10 %**	**Top 30 %**
WT	16,66	41,66	83,33
VSM	0	0	16,66

As shown in Table.3, with WTBugLoc, top 5 and top 10 are considerably higher than the VSM technique, therefore, in this approach a significant number of relevant files are highlighted in order to help the team project to find the buggy files and next to eliminate occurred bug.

As described in Table.2, the bug report ID 102756 concerns the file GTK.java. The latter is ranked in position one in the list of relevant files (Fig.2).

Moreover, with our tool WTBugLoc as we can see in Fig.3, the position of the relevant file according to each bug report treated is less than 50. This ensures a time and cost optimization in software maintaining project.

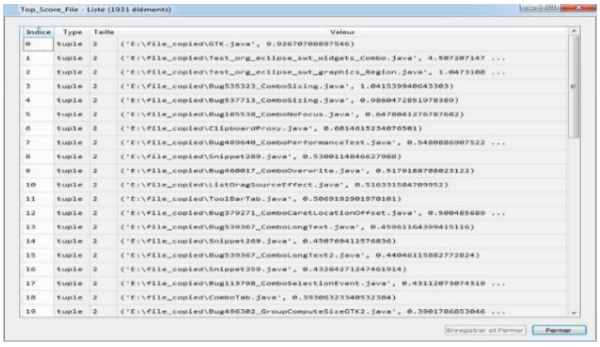

Fig.2. Top ranking scored relevant files based on bug report ID 102756

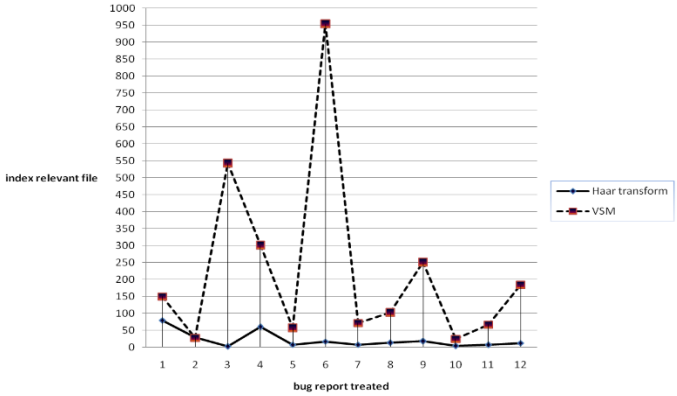

Fig.3. Top ranking scored relevant files beside each bug using WT and VSM

RQ: what is the effectiveness of using wavelets transform for bug localization?

The Table.3 shows the results of bug localization using the VSM and the proposed WT methods. The experimental results show that the proposed WT method outperforms the VSM method. For example, for Eclipse 3.1, we find that (16.66%) of top 5 and (41.66%) of top 10 and (83.33%) of top 30 whose relevant source file are returned using the proposed WT method. While in VSM method, those numbers are (0%), (0%) and (16.66%) respectively.

5. CONCLUSION

During the life cycle of software, many bugs appear which could be tedious and costly work. Once a new bug report comes, the team project needs to know which files should be modified to fix the bug. However, it requires much time to find the files to be altered manually based on the original bug reports, particularly for a large software project.

This contribution proposes an Information Retrieval method named WTBugLoc for locating relevant source code files based on initial bug reports. The tool uses Wavelets Transforms, especially the Haar Transform, a technique largely used in signal processing, well unknown nowadays in the field of the IR. The evaluation results on the SWT eclipse project shows that WTBugLoc can perform bug localization effectively and it shows that this tool outperforms other methods such as those based on VSM. In the future, further works will explore the usefulness of other Wavelets Transform in IR, to improve the performance of this theory in the field of data mining and bug localization.

REFERENCES

[1] E.J. Braude and M.E. Bernstein, "*Software Engineering: Modern Approaches*", Waveland Press, 2016.

[2] Eclipse, Available at https://bugs.eclipse.org/bugs/.

[3] S.S. Murtaza, "An Empirical Study on the use of Mutant Traces for Diagnosis of Faults in Deployed Systems", *Journal of Systems and Software*, Vol. 90, pp. 29-44, 2014.

[4] K.H. Chang, V. Bertacco and I.L. Markov, "Simulation-Based Bug Trace Minimization with BMC-based Refinement", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 26, No. 1, pp. 152-165, 2006.

[5] M.M. Rahman and C.K. Roy, "Improving IR-based Bug Localization with Context-Aware Query Reformulation", *Proceedings of ACM Joint Meeting on European Software Engineering*, pp. 621-632, 2019.

[6] S. Rao and A. Kak, "Retrieval from Software Libraries for Bug Localization: A Comparative Study of Generic and Composite Text Models", *Proceedings of 8th International Conference on Mining Software Repositories*, pp. 1-13, 2011.

[7] T.D.B. Le, R.J. Oentaryo and D. Lo, "Information Retrieval and Spectrum based Bug Localization: Better Together", *Proceedings of Joint Meeting on Foundations of Software Engineering*, pp. 23-29, 2015.

[8] B. Sisman and A.C. Kak, "Incorporating Version Histories in Information Retrieval based Bug Localization", *Proceedings of 9th IEEE Working Conference on Mining Software Repositories*, pp. 1-12, 2012.

[9] S. Wang and D. Lo, "Version History, Similar Report, and Structure: Putting them Together for Improved Bug Localization", *Proceedings of International Conference on Program Comprehension*, pp. 1-5, 2014.

[10] R.K. Saha, "Improving Bug Localization using Structured Information Retrieval", *Proceedings of IEEE/ACM International Conference on Automated Software Engineering*, pp. 1-12, 2013.

[11] Z. Shi, "Comparing Learning to Rank Techniques in Hybrid Bug Localization", *Applied Soft Computing*, Vol. 62, pp. 636-648, 2018.

[12] C.P. Wong, "Boosting Bug-Report-Oriented Fault Localization with Segmentation and Stack-Trace Analysis", *Proceedings of IEEE International Conference on Software Maintenance and Evolution*, pp. 1-12, 2014.

[13] Y. Yang, "An Empirical Study on Dependence Clusters for Effort-Aware Fault-Proneness Prediction", *Proceedings of IEEE/ACM International Conference on Automated Software Engineering*, pp. 1-8, 2016.

[14] K.C. Youm, "Bug Localization based on Code Change Histories and Bug Reports", *Proceedings of Asia-Pacific Conference on Software Engineering*, pp. 1-13, 2015.

[15] S. Zhang and C. Zhang, "Software Bug Localization with Markov Logic", *Proceedings of International Conference on Software Engineering*, pp. 321-334, 2014.

[16] J. Zhou, H. Zhang and D. Lo, "Where Should the Bugs be Fixed? more Accurate Information Retrieval-based Bug

Localization based on Bug Reports", *Proceedings of International Conference on Software Engineering*, pp. 225-229, 2012.

[17] K. Ramamohanarao and M. Palaniswami, "A Novel Document Retrieval method using the Discrete Wavelet Transform", *ACM Transactions on Information Systems*, Vol. 23, No. 3, pp. 267-298, 2005.

[18] J.S. Walker, "*A Primer on Wavelets and Their Scientific Applications*", CRC Press, 2008.

[19] S.F. Stankovic Radomir and J. Bogdan, "The Haar Wavelet Transform: its Status and Achievements", *Computers and Electrical Engineering*, Vol. 29, pp. 25-44, 2003.

[20] S. Karus and K. Kilgi, "Code Clone Detection using Wavelets", *Proceedings of International Conference on Software Clones*, pp. 1-14, 2015.

[21] A. Krishnan, K.B. Li and P. Issac, "Rapid Detection of Conserved Regions in Protein Sequences using Wavelets", *In Silico Biology*, Vol. 4, No. 2, pp. 133-148, 2004.

[22] M.K. Jeong, "Wavelet-Based Data Reduction Techniques for Process Fault Detection", *Technometrics*, Vol. 48, pp. 26-48, 2006.

[23] M.Y. Dahab, M. Kamel and S. Alnofaie, "An Empirical Study of Documents Information Retrieval using DWT", *Proceedings of International Conference on Intelligent Natural Language Processing: Trends and Applications*, pp. 251-264, 2018.

[24] O. Maimon and L. Rokach, "*Data Mining and Knowledge Discovery*", Springer, 2005.

[25] D. Hovemeyer and W. Pugh, "Finding Bugs is Easy", *ACM SIGPLAN Notices*, Vol. 39, No. 12, pp. 92-106, 2004.

[26] R. Khoury and C. Drummond, "*Advances in Artificial Intelligence*", Springer, 2016.

[27] D.F Walnut, "*An Introduction to Wavelet Analysis*", Springer, 2013.

[28] Z. Abba and P. Rain, "A Study on Applications of Wavelets to Data Mining", *International Journal of Applied Engineering Research*, Vol. 13, No. 12, pp. 10886-10896, 2018.

[29] A. Graps, "An Introduction to Wavelets", *IEEE Computing in Science and Engineering*, Vol. 2, No. 2, pp. 50-61, 1995.

[30] C.D. Manning and H. Schutze, "*Foundations of Statistical Natural Language Processing*", MIT Press, 1999.

[31] C. Manning, P. Raghavan and H. Schutze, "*Introduction to Information Retrieval*", Cambridge University Press, 2008.