International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 11, No.1 (Feb-2022)

http://dx.doi.org/10.12785/ijcds/110162

Identifying Duplicate Bug Records Using Word2Vec Prediction with Software Risk Analysis

Hussain Mahfoodh[1] **and Mustafa Hammad**[1]

[1]*Department of Computer Science, University of Bahrain, Sakheer, Bahrain*

Received 19 Apr. 2021, Revised 24 Aug. 2021, Accepted 18 Jan. 2022, Published 15 Feb. 2022

Abstract: Reporting duplicated bugs in bug reports have serious productivity consequences on software projects. The fewer reporting of duplicated bugs, the better software maturity processes are set between the internal software stakeholders. Automated identification of the duplicated category through bug reports could enhance risk identification approaches during the software life cycle. In this paper, we propose two different similarity measures to identify duplicated bugs using the word-embedding (Word2Vec) natural language processing technique through Tensorflow tool. We conduct a comparison experiment on two related bug records descriptions from eight different software components from the Mozilla Core dataset. We choose different sentence types through the duplicated bug category records to compare and discuss each component's accuracy results and identify whether the proposed module will be able to detect the related records. Using an earlier work, this paper calculates software risk values from duplication records and from bug-fix time prediction for the components that have not been identified as duplicated by the Word2Vec approach. The study results show maximum precision accuracy of 99.89% for the components that have been identified correctly as duplicated by the used approach. Additionally, we found that 66% of the software components that were excluded from the bug duplication proposed module showed an increase in software risk values.

Keywords: Bug reports, duplicated bugs, bug-fix time, software risk estimation, bug-fix time prediction, software risk management, word embedding, natural language processing, machine learning

1. Introduction

Early identification approaches of software risk can help in producing reliable software [1]. The identification allows developers to plan ahead and reduce bugs and errors in software projects by pre-detecting failures [2] that could threaten the software assets. Quantitative risk identification from the different bug reports attributes enables internal software stakeholders to manage project resources in advance before and after the software deployment and thus providing the highest levels of quality and assurance [3], [4] for the end-user.

Duplicated bug reporting records are also a software management issue that is common in many projects during the software life cycle. Despite its different causes, it provides discomfort and disturbance to the internal software stakeholders interested to resolve these issues [5] and looking to work on the software pending tasks.

No matter how advanced the tools used to manage bug-reporting [6], [7] processes, the duplication issue in a software project will still emerge. The reason is that is because bug reporting requires human input intervention and requires validation of existing bug records, which is difficult to track as the project continues to expand to fulfill the ongoing requirements. Another reason is that reported bug description is a natural language text that is written differently between each developer or quality tester even if the text represents the same bug meaning, the possibility of misunderstanding and poor interpretation might bring confusion and provides inefficient resolving operations.

The software maturity level could increase by providing an automated approach to discover duplicated bugs through the different bug report records. The word embedding (Word2Vec) approach can be used to analyze text similarity and helps to find the closest word-to-word among the description of the reported bug by excluding the similar words that are either same or marked as closely similar by the used approach. This can assist in reducing the appearance of duplicated bugs through bug reports and helps in better software project management.

Estimation of software risk levels is achievable through in-depth analysis of bug reports. By analyzing specific attributes such as the number of duplicated unfixed records, fix priority levels, and extraction of bug-fix time prediction from related previous fixed bugs. Those attributes can be used to provide insights to shift the focus of internal stakeholders to what needs to be addressed and resolved

E-mail address: 20073370@stu.uob.edu.bh, mhammad@uob.edu.bh

first through providing urgency levels estimation values of software risk through the different software components.

In this paper, we propose two text similarity measures using natural language processing through a Word2Vec approach among a list of chosen bugs description records for the aim to detect two related bugs records within and find word similarities through these selected sentences. We take different records with different common and uncommon actual valid words similarity samples to test the approach accuracy and check if the approach will be able to detect related duplicated bug records. We calculate the software risk values of the selected components and compare them with the software risk values after gathering the word similarity results. The purpose is to compare and analyze a Word2Vec approach similarity measure effect on the overall software risk levels. The experiment is performed on a *Mozilla Core* project bug report from Firefox online public repositories.

The rest of the paper is organized as follows. Section 2 provides related work. Section 3 describes the methodology. Sections 4, 5 presents the experimental results and discussion. Finally, Sections 6, 7 represents the limitations of the used approach and the conclusion.

2. Related Work

Previously reported existing bugs that are reported again by the internal software stakeholders can result in negative productivity to the software development project. The study by Cavalcanti, Y. C. *et al.* [8] discussed the negative impact of the duplication issue by studying nine different software projects. The study pointed different metrics that cause high bugs duplication similarities such as bug reports frequency, duplication similarity ratio, and duplication percentage. Another study by Bettenburg, N. *et al.* [9], explained that reporting duplicated bugs could occur because of incompetence users, technological unawareness of reporting platforms, errors, and accidental resubmission. The same study discussed that most developers will not spend much of their working time searching hard-to-find duplicated bugs in these bug reports. The study also stressed that high percentages of reported bugs are exact copies of each other and are wrongly identified as from the same duplicated bugs category.

Several studies to detect duplicated bugs records through bug reports have been conducted. The study by Lazar, A. *et al.* [10] used binary classification approaches to identify duplicate bug records using textual similarity. The study used different metrics such as precision-recall, and the area under the curve to calculate the results accuracies. The study by Tian, Y. *et al.* [11], extended the work by Jalbert, N., and Weimer, W. [12] and introduced new similarity detection measures to improve the accuracy of duplication identification by 160% where it was conducted on Mozilla open-source bug reports dataset.

Further NLP studies on bug reports have been used to detect and reduce redundant duplicated bugs records. The study by Wang, X. [13] *et al.* detected up to 93% of the duplicated bug category from a selected Firefox bug repository by using both Natural-Language-based and Execution-information-based similarities approaches. A different study by Runeson, P. *et al.* [14] reached up to 66% of accuracy for detecting duplicated bug records by using a specific NLP prototype. The same study used different detection techniques such as Stemming, Stop Words Removal, Tokenization, and Vector Space model.

The word-by-word bug duplicates detecting through machine learning has been used in different researches. The study by Guthrie, D. *et al.* [15] worked on predicting and finding the closest related words for a given set of words using skip-grams. A different approach by Homma, Y. *et al.* [16] used Siamese networks with the use of deep learning to identify similar equivalent questions within the same language context. The study by Ma, L. and Zhang, Y. [17] evaluated word similarities using Word2Vec on corpus text. Another Word2Vec study by Jatnika, D. *et al.* [18] reached a result of 66% similarity accuracy by examining a different set of English words and finding its similarity rates equivalently. Further Word2Vec studied has been also used on different detection bug duplication studies by Lin, M. J. *et al.* [19], Deshmukh, J. *et al.* [20], Xie, Q. *et al.* [21], Kang, L. [22], Kukkar, A. *et al.* [23], and He, J. *et al.* [24].

Risk management has an important role in the software development cycle. Byron, J. *et al.* [25] emphasized the high need for early categorizing of software risk in software projects development phases. The study expressed the importance of categorization the risk to start the goal of risk prioritization. Different attributes need to be studied such as risk complexity, the project life cycle schedule, and the overall project size. Kaur, G., and Bahl [26] stated the difficulty of achieving higher software reliability and expressed that it can be improved by using better risk management processes, better configuration management, and development processes.

The appearance of duplicate bug reports causes negative incompetence [8], [9] to the software project's overall productivity. The detection of duplicate bugs using textual similarity [10], [11], [12], through NLP [13], [14], or through Word2Vec [17], [18], [19], [20], [21], [22], [23], [24] can be used as an automated method to enhance the early software risk for risk categorization and prioritization [25] and to increase the overall software reliability [26]. It is important to look for a correlation between software risk and duplicated bugs through the overall software project development to enhance its security and productivity. In addition, by providing new textual similarity measures, such a study could provide more aspects for researchers to identify duplicated bugs records within Word2Vec numeric close-indexed distance range results which were not the main focus in the related work. This study will focus on

using Tensorflow with Word2Vec word-to-vector closest technique to find sentence similarities of related duplicates bug reports by using Squared Euclidean distance. We propose two closest word-to-word similarity measures with the use of a neural network. Then, we calculate the software risk values before and after applying the similarity measures on the selected bug records to compare the approach's effect on the software risk values.

3. Methodology

In the following subsections, we discuss the bug reports dataset structure; then, we demonstrate the software risk retrieval approach used. Furthermore, the purposed Word2Vec similarity measures approaches are discussed, followed by an evaluation criteria explanation to evaluate whether the proposed module explained could determine whether the selected bug description text has any relation with the same duplicated bug record. Finally, we discuss the experimental results values and a comparison of risk values is illustrated among the excluded records from the bug duplication proposed module.

This research uses *Mozilla Core* [27] Firefox public open-source bug reports. Table I lists the dataset attributes definitions including their data types are also shown. The attribute *Days_to_fix_the_bug* is calculated from the difference between the attributes *Resolved_time* and *Created_time* respectively. This attribute then is converted into days' time format and which represents the input point for the proposed bug-fix prediction process that the risk module will use.

A non-linear regression prediction algorithm work [28] is implemented and only experimented on the fixed bugs records, where only *Fixed* category values from the *Resolution* attribute are considered. Table II shows the bug records total number, the *Fixed* bug records resolution type total number, the *Duplicate* resolution type total number, the bug fixing time average, and the date duration period for the used dataset.

We conduct the experiment on eight software components from the *Mozilla Core* bug report dataset. Table III shows the total number of bugs records, the number of duplicated bugs categorized by their date period.

A. Bug Reports Software Risk Estimation

We use an approach [29] to estimate Software risk values taken from predictions of bug-fix time and from unfixed bug duplicates records from the same bug report dataset.

Fig. 1 shows the overall used risk decision model. The used approach takes one input, which is the bug component name from the dataset. The bug records are then filtered to show only the entered component name records. To predict the upcoming bug bug-fix time, the bug-fix time prediction module is used on the filtered data. Then, the risk analyzer component takes the bug-fix time value prediction to process and calculate the risk estimation value. Furthermore, within the same filtered software component, the related count number of duplicated bugs is considered and also passed into the risk analyzer component module from the proposed approach. The reason to consider duplicated bugs is because of the probability of the same bug multi-occurrences among other bug records within the filtered software component and which could indicate an unmitigated risk level that the model must take into consideration. Finally, the predicted bug-fix time and the duplicates count will be passed to the risk analyzer module to calculate the overall final risk value.

This study uses non-linear regression with a neural network to predict the future bug-fix time value [28]. The

TABLE I. The Used Dataset Structure.

Attributes	Definition	Type
Issue_id	Bug record unique ID used.	Positive Integer
Priority	The Bug priority number assigned to the bug record.	String
Component	Software component name that belong to the bug.	String
Duplicated_issue	Contain duplicated bug ID to identify if the same issue reported before.	Positive Integer
Title	Title of the bug description.	String
Description	Detailed description for the bug.	String
Status	Assigned by the software project manager as: Verified, Resolved, or Closed bug.	String
Resolution	Specify whether the bug is (Fixed/ Invalid/will not be fixed/ already working or duplicated).	String
Version	The software version or branch repository the bug is located in.	String
Created_time	Date/time of the bug reported.	Date/Time
Resolved_time	Bug resolving Date and time.	Date/Time
Days_to_fix_the_bug	Difference between the attributes *Resolved_time* and *Created_time* calculated in days format.	Float

TABLE II. The Number of Bugs Records Selected and their Period From The Dataset.

Dataset name	Total number of bug records	Number of the Fixed bug samples	Number of the duplicated bug samples	Average bug fixing time (days)	Duration of Bug Reporting Period
Mozilla Core	205069	101500	44692	419.59	December 1995 – December 2013

TABLE III. Software Components Duplicated Bugs Count Categorized by their Duration.

Software component name	Bug records sample count	Duplicated bug samples count	Timeline period
JavaScript Engine	22073	3118	December 1997 – December 2013
Graphics	5867	945	December 1995 – December 2009
Build Config	4260	409	July 2000 - December 2013
Event Handling	2768	932	October 1997 – August 2013
DOM	7315	885	August 1998 – December 2013
Widget	488	44	March 2002 – December 2013
Layout	16402	4621	April 1993 – December 2009
XUL	9022	2409	April 1994 – December 2009

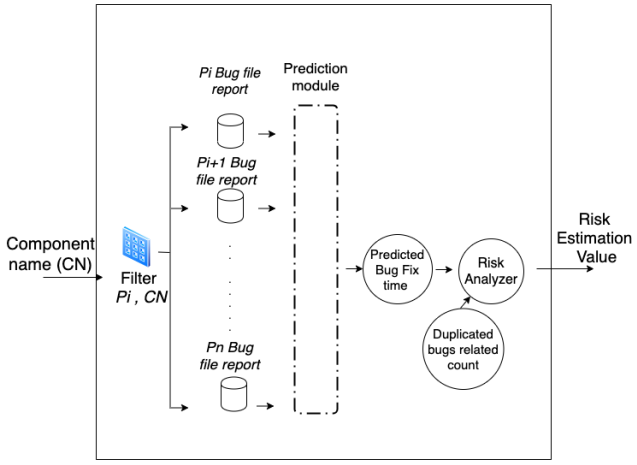

Figure 1. The Proposed Risk Estimation Process Model

formula will take into consideration the previous bug-fix time values to calculate the expected upcoming bug-fix time. Formula 1 shows the bug-fix time risk mathematical representation used.

$$RB = \frac{L}{a} * 100 \quad (1)$$

In Formula 1, *RB* is the risk percentage of the bug-fix time prediction. This value is calculated from the actual *Fixed* samples of the selected and filtered software component. We take the last predicted bug-fix time sample value which represents (*L*) and divides it over the average of the actual *Fixed* samples which represent (*a*).

The risk values from the duplicated bug category are calculated only from the duplicates bugs with *WONTFIX* resolution type. The reason is this category could contain a residual level of risk since these types of bugs records will not be fixed in the current project tasks pipeline. Formula 2 shows the duplicated bug records risk formula.

$$RD = \frac{DW}{T} * 100 \quad (2)$$

In Formula 2, *RD* represents the duplication bugs' final risk percentage. To calculate this values, *DW* is used for representing the duplicated bugs samples with the *WONTFIX* resolution type. The calculation will take into consideration the count of the related *WONTFIX* records for each of the other duplicated bug samples; *T* represents the total count of the duplicated bugs within the same software component even if there was no relationship with the *WONTFIX* resolution type bugs records.

The final values of risk will be calculated by converting values of (*RB*, *RD*) into a percentage of maximum 50% rate and adding them together. Formula 3 shows (*TR*) which is the total final risk value reaching up to 100% rate level.

$$TR = RB + RD \quad (3)$$

B. Word2Vec Duplicates Bug Determination Approach

Fig. 2 shows the process to determine whether two selected duplicated bug records from the bug report are related to each other. The proposed approach takes the bug file reports with records that include bugs description text. The words separator module then will separate each word and will be grouped into an array of array data type. The redundant words will be removed from this array to make sure only unique words are processed for the purpose of a more efficient machine-learning process. The predefined window size variable will restrict the combination of words in the array that the learning process will traverse through. Every two words pair from the combination array will be passed to the training module and a result of float vector value will be obtained for each of the given unique words. These vector values results will be handled to the duplication analyzer module along with the selected bug description to be compared with each other. The duplication analyzer output results are a boolean value that will determine if each pair of selected bugs descriptions are duplicated and related to each other or not. The following text is an example for demonstration of two bugs description text to be compared and to determine if there is any duplication relationship between them.

Bug 1: "*First bug description is listed here*".
Bug 2: "*The second issue description*".

The process will filter only understandable English valid words for the purpose of valid words training by the proposed module. Other invalid words such as alphanumeric strings or stop words (period, comma, and semicolons) will be omitted including any redundant words that are repeated or words with (pronouns, propositions) because of its insignificant to the duplication text identification module. The identification of those pronouns and propositions will

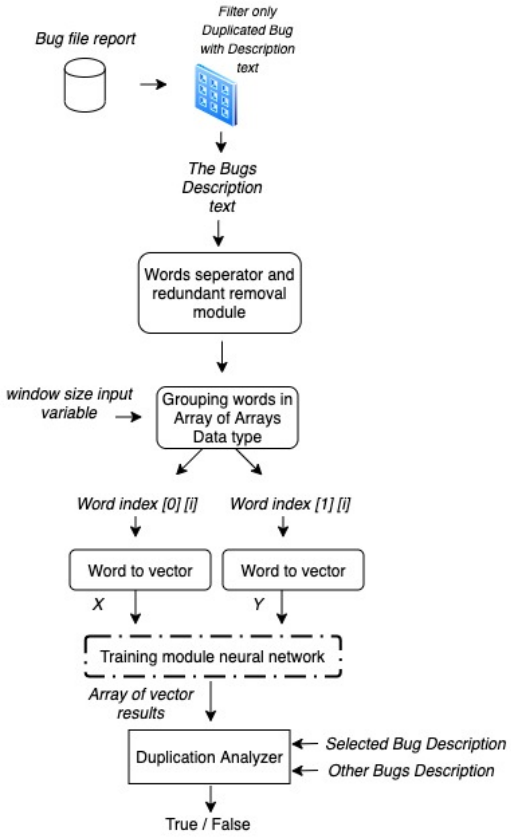

Figure 2. Bug Duplication Distinguish process.

be done through examining each word with a fixed common and known words list and omitting the word when there is a match. The word separator module will combine and separate the description text of these bugs into an array of words. Fig. 3 shows the words separated into one array variable.

Fig. 4 shows the combination of the pair words within a predefined variable *WINDOW_SIZE=2* which gives a total of 28 word combination. Increasing the value of the *WINDOW_SIZE* variable will lead to more words combination in the array. The more this value is increased the more words combination in the array will be covered to be trained by the training module.

A binary vector will be converted for each pair of words in the array that list every two words together. As an example, ['first', 'bug'] pair in the first array where the word 'first' will be converted to [0. 1. 0. 0. 0. 0. 0. 0. 0.] and will be added to X array. Similarly, the word 'bug' will be converted to [0. 0. 1. 0. 0. 0. 0. 0. 0.] and will be added to Y array. Thus, X will have the first-word index for each word pair in the array, and Y will have the second-word index for each word pair in the array. Fig. 5 shows both of the final arrays of (X and Y) to be given to the same mentioned words.

```
['first ', 'bug', 'description', 'is',
'listed', 'here', 'the', 'second', 'issue',
'description']
```

Figure 3. Bug Description sentences words separation array.

```
[['first', 'bug'],['first', 'description'],
['bug', 'first'], ['bug', 'description'],
['bug', 'is'], ['description', 'first'],
['description', 'bug'], ['description',
'is'], ['description', 'listed'], ['is',
'bug'], ['is', 'description'], ['is',
'listed'], ['is', 'here'], ['listed',
'description'], ['listed', 'is'], ['listed',
'here'], ['here', 'is'], ['here', 'listed'],
['the', 'second'], ['the', 'issue'],
['second', 'the'], ['second', 'issue'],
['second', 'description'], ['issue',
'the'], ['issue', 'second'], ['issue',
'description'], ['description', 'second'],
['description', 'issue']]
```

Figure 4. The words combination for the mentioned sentences when variable $WINDOW_SIZE = 2$.

```
X=
[[0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0.]

Y =
[[0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1.]
[1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 1. 0. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0.]]
```

Figure 5. The X and Y Binary Vector for each pair of the Array words.

Fig. 6 shows the Word2Vec neural network-training module conducted on this experiment. The neural network consists of 3 layers: one input, one hidden, and one output layer. Different weights *(Weight 1, Weight 2)* and different biases *(Bias 1, Bias 2)* are used for the input and hidden layers. By using Gradient descent optimization [30], error propagations are adjusted across the layers of the training module.

The prediction process happens at the hidden layer through the use of the Softmax function. The Softmax [31]

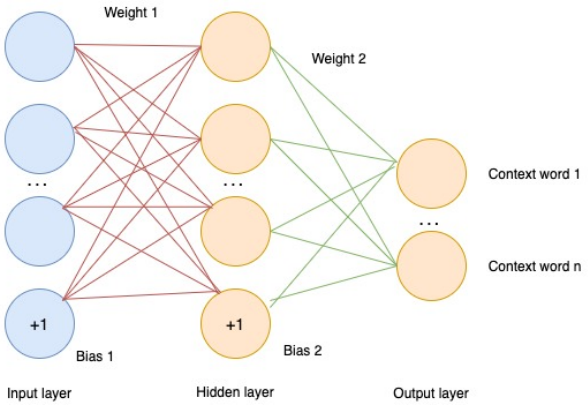

Figure 6. one input, one hidden, and one output layers of the Training Neural Network module.

function is used on non-normalized data output from the first input layer to provide a probability distribution for better prediction estimation. Formula 4 shows the Softmax function representation where $\sigma : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and $i, j = 1, ..., n$ with $z = (z_1, ..., z_n) \in \mathbb{R}^n$.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{4}$$

For each given unique word there will be an output of vector value from the training module. Fig. 7 shows the output vector numeric values for the same bug description example discussed earlier at the beginning of the second subsection of this section. For each given word in the input sentence, the float numeric output values are ranging from -0.99 to 0.99 as an output result from the training process.

```
[('issue', 0.21356754), ('listed',
-0.1225790), ('description', 0.43123457),
('first', -0.53720089), ('the', 0.7168123),
('bug', 0.885157), ('here', -0.1765210),
('second', -0.9964821), ('is', -0.3213419)]
```

Figure 7. Sample of the example for the array words final vector numeric output values results.

C. Duplication Analyzer and Similarity Measures

From the Word2Vec approach results, Squared Euclidean distance [32] is used on the trained vector result output to find the closest word from two given sentences set for comparisons. The following formula shows p, q which are two points to calculate the Euclidean distance between them in the n Euclidean space.

$$\sqrt{\sum_{i=1}^{n} (q_i - p_n)} \tag{5}$$

To determine if a duplication relation between two different given bug description texts, this paper uses two similarity measures for the comparison purpose. From the discussed Fig. 2, once each valid word in every sentence has a numeric vector result then the duplication analyzer module takes each two bugs description text and analyzes them word-by-word to determine if any duplication similarity exists. By following the same example discussed in subsection 2 of this section, each valid word in Bug #1 text will be compared with each valid word of Bug #2. Both similarity measures will iterate for every word in two different given bug descriptions. The following points will explain the two similarity measures in detail:

1) *First Similarity Measure:* This method will iterate between two given bug sentences and will find the closest valid word available through the float vector output results value using Squared Euclidean distance. The method will use a variable *counter* to identify how many words have been identified as similar to each other by the duplication analyzer module. Fig. 8 shows the first duplication similarity measure processes used in detail.

2) *Second Similarity Measure*: The same iteration approach method will be used in the first method discussed, however, it will look for the closest float value within a specific range (n) in the numeric float vector array output result. The variable *counter* is used to count the similar words if the word in two different given sentences has been found or identified as closely similar within the given range. Fig. 9 shows the second duplication similarity measure processes used in detail.

To determine if two different given words from two different bug description sentences are identified as the same or closely similar by the duplication analyzer module, we use formula 6 for this purpose. p represents the similarity percentage calculated for the word similarity determination approach. c represents the counter discussed earlier, w represents the total number of words given from the second sentence to compare with the initial first bug sentence. This paper uses 50% threshold value to determine the bug duplication similarity determination between the bug description words. The bug will be identified as the same as the other compared bug when the value of p is more than or equal to the same threshold. The same threshold is extracted from a similar study on Mozilla dataset duplication category similarity identification measure [33].

$$p = \frac{c}{w} \times 100 \tag{6}$$

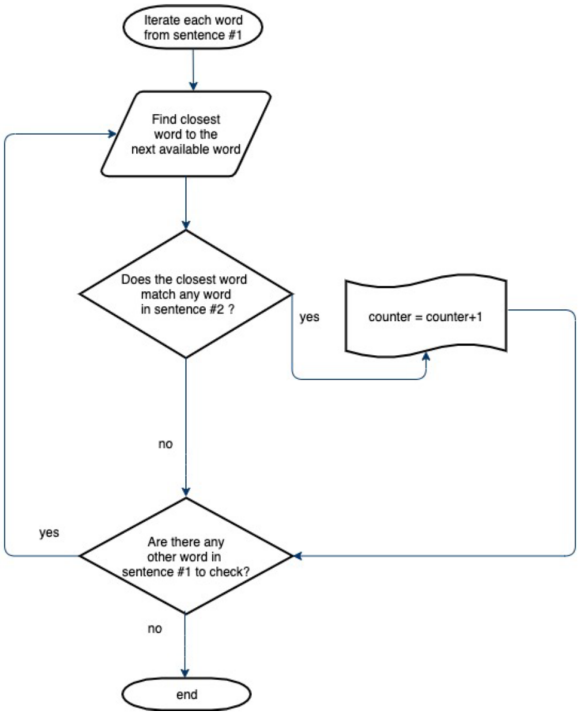

Figure 8. First Similarity bug duplication identification measure processes.

Figure 9. Second Similarity bug duplication identification measure processes.

D. Evaluation Criteria Accuracy

In order to calculate the approach accuracy towards the identification of bug duplication similarity, the percentage for True-Positive and False-Positive must be calculated. True-Positive are the values in which represent the actual sentences that are originally considered duplicated bugs and the duplication analyzer module successfully identifies these sentences as a true duplication bug. The False-Positive are the values that represent that the duplication analyzer module falsely identified a non-actual bug description as a true duplicate even though both sentences were not from the same category. Since the precision accuracy values are focused on the positive instances that were classified correctly, Formula 7 shows the precision formula from both values of True-Positive (TP) and False-Positive (FP) percentages that were used in this experiment to evaluate the duplication module accuracy.

$$Precision = \frac{TP}{(TP + FP)} \tag{7}$$

To evaluate the bug-fix time prediction results to assess the risk estimation approach accuracy, we have used four regression metrics on the bug-fix time predicted samples, which are: mean absolute error (MAE), mean squared error (MSE), R^2 (R squared), and median absolute error (MedAE). Considering E is the bug-fix time retrieved from the bug-fix time dataset selected samples, $\acute{E}$ is the bug-fix time predicted sample, $\hat{E}$ is the mean of E, and n is the total number of the bug records samples. Formulas 8, 9, 10, and 11 show the regression metrics error rate between the actual bug-fix time and the results of the predicted samples results.

$$MAE = \frac{\sum_{i=1}^{n} |E_i - E_i'|}{n} \tag{8}$$

$$MSE = \frac{\sum_{i=1}^{n} (E_i - E_i')^2}{n} \tag{9}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (E_i - E_i')^2}{\sum_{i=1}^{n} (E_i - \hat{E}_i)^2} \tag{10}$$

$$MedAE(E, E') = median(|E_1 - E_1'|, ...|E_n - E_n'|) \tag{11}$$

4. Experimental Results

The experiment is conducted using Tensorflow [34] version 1.9.0 and using Word2Vec word embedding libraries [35], which is a Tensorflow natural language processing module. The machine specification used in this experiment is a 2.2 GHz Intel Core i7 16 GB 1600 MHz DDR3 machine with 1536MB GPU.

Table IV shows the result of the experiment. The experi-

TABLE IV. Result of The Eight Selected Software Components for Bug Duplication Identification Experiment.

Software component name	*JavaScript Engine*	*Graphics*	*Build Config*	*Event Handling*	*DOM*	*Widget*	*Layout*	*XUL*
Number of actual similar words from both of text	14	1	15	0	0	0	0	3
Related bug record valid words Number	10	5	9	3	5	3	8	10
Related bug similar words number in the other record using similarity measure #1	10	0	4	0	1	4	0	2
Related bug similar words number in the other record using similarity measure #2	10	0	4	0	1	5	0	2
Related bug similar words number in the other record using similarity measure #3	10	0	4	0	1	7	0	2
Similar words Percentage using Similarity measure #1	100%	0%	44.4%	0%	20%	133.3%	0%	20%
Similar words Percentage using Similarity measure #2	100%	0%	44.4%	0%	20%	166.6%	%0	20%
Similar words Percentage using Similarity measure #3	100%	0%	44.4%	0%	20%	233.3%	0%	20%
TP	Method 1 TP: 100% Method 2 TP: 100% Method 3 TP: 100%	Method 1 TP: 0% Method 2 TP: 0% Method 3 TP: 0%	Method 1 TP: 0% Method 2 TP: 0% Method 3 TP: 0%	Method 1 TP: 0% Method 2 TP: 0% Method 3 TP: 0%	Method 1 TP: 0% Method 2 TP: 0% Method 3 TP: 0%	Method 1 TP: 100% Method 2 TP: 100% Method 3 TP: 100%	Method 1 TP: 0% Method 2 TP: 0% Method 3 TP: 0%	Method 1 TP: 0% Method 2 TP: 0% Method 3 TP: 0%
FP	Method 1 FP: 1.79% Method 2 FP: 2.82% Method 3 FP: 3.08%	Method 1 FP: 0.74% Method 2 FP: 0.848% Method 3 FP: 0.848%	Method 1 FP: 2.21% Method 2 FP: 2.45% Method 3 FP: 3.43%	Method 1 FP: 0.43% Method 2 FP: 1.39% Method 3 FP: 1.39%	Method 1 FP: 6.10% Method 2 FP: 7.12% Method 3 FP: 7.46%	Method 1 FP: 0.10% Method 2 FP: 0.95% Method 3 FP: 0.95%	Method 1 FP:0.51% Method 2 FP: 0.75% Method 3 FP: 0.75%	Method 1 FP: 0.04% Method 2 FP: 0.08% Method 3 FP: 0.12%
Precision from Similarity measure #1	98.23%	0%	0%	0%	0%	99.89%	0%	0%
Precision from Similarity measure #2	97.25%	0%	0%	0%	0%	99.05%	0%	0%
Precision from Similarity measure #3	97.011%	0%	0%	0%	0%	99.05%	0%	0%

ment is applied on the two similarity approaches mentioned earlier. We have chosen 3 similarity measures where the first similarity measure follows the Squared Euclidean distance to find the closest valid word available through the float vector output results value. The other two remainings are based on the second similarity measure mentioned earlier and from the final vector results by finding the closest words within 2 and 3 range limits respectively. From the different software components, we take different bug description sample text to provide multiple comparisons analysis. For example, we have taken 14 similar word bugs descriptions between the pair of the text with the existence of 10 valid words identified for the first component *JavaScript Engine*. For the other components, we take the text with similar words range from 0 to 15 and with valid words from 3 to 10 words. For each selected component, we take one fixed bug records sample with other duplicated records that contain one sample that has a relation with the fixed record using bug ID mapping mentioned in the same bug report. As shown in Table IV, the precision for the experiment results ranged from 0 to 99.89% for all the selected software components and all the chosen similarity measures. The *Widget* software component shows the highest similarity percentage with 133.3% for the first measure, 166.6% for the second measure, and 233.3% for the third measure. *Graphics*, *Event Handling*, and *Layout* show the similarity percentage with the lowest for all of the similarity measures with 0%.

Table V shows the count of all the duplicated bug records used on the experiment categorized by their similarity percentage, similarity type, and categorized by their software components. Table VI and VII shows the actual risk values of the used software components on the experiment. These two tables also shows the software component risk values after conducting the bug duplication identification experiment and after excluding the excluded related bug record by the used approach. This table only uses the software component that marked by the proposed module as not related with true-positive and equal to 0% since the risk values of the 100% true-positive will be the same as the actual risk values. Some of the excluded bug records are not read by the module because of the same hardware limitation discussed on the used bug-fix time prediction study [28] and

TABLE V. Count of Bug Duplicated Records Similarity Percentage for the Three Similarity Measures From the Proposed Model.

	Count of similarity percentage >=0% and <25%			Count of similarity percentage >=25% and <50%			Count of similarity percentage >=50% and <75%			Count of similarity percentage >=75% and <100%			Count of similarity percentage >=100%		
Software component name	**SM1**	**SM2**	**SM3**	**SM1**	**SM2**	**SM3**	**SM1**	**SM2**	**SM3**	**SM1**	**SM2**	**SM3**	**SM1**	**SM2**	**SM3**
JavaScript Engine	2935	2843	2820	126	186	201	44	76	81	0	0	0	13	13	16
Graphics	923	926	923	15	11	14	7	8	8	0	0	0	0	0	0
Build Config	365	358	339	35	41	56	9	10	13	0	0	0	0	0	1
Event Handling	881	844	828	46	74	90	4	12	10	0	1	3	0	0	0
DOM	786	751	728	45	71	91	13	18	21	0	0	0	41	45	45
Widget	41	34	31	1	0	3	1	5	3	0	2	1	1	3	6
Layout	4484	4445	4438	113	141	148	22	33	33	0	0	0	2	2	2
XUL	2384	2357	2349	24	50	57	1	2	3	0	0	0	0	0	0

TABLE VI. Risk Results For Software Components: *Graphics, Build Config, and Event Handling* That was marked as Excluded From The Bug Duplication Proposed Model.

Priority types	*Graphics* (Actual)	*Graphics* (After Excluding the related bug record)	*Build Config* (Actual)	*Build Config* (After Excluding the related bug record)	*Event Handling* (Actual)	*Event Handling* (After Excluding the related bug record)
No. of Fixed samples and acceptable by the machine	13373	3373	2056	2056	943	943
Duplicates categorized to *WONTFIX* records number	208	208	128	128	287	287
***WONTFIX* records number**	363	363	162	162	861	861
Number of Duplicates samples	945	944	409	408	932	931
Last sample predicted of bug-fix time value (days)	0.0296475	0.05211727	0.01509095	0.03829417	0.04175371	0.03113707
Bug-fix time average value (days)	0.08338276	0.08338276	0.08341444	0.08341444	0.08351026	0.08351026
RD	60.42%	60.48%	70.90%	71.07%	123.17%	123.30%
RB	35.55%	162.50%	18.09%	45.90%	49.99%	37.28%
TR	47.98%	61.49%	44.49%	58.49%	86.58%	80.29%
Evaluation criteria accuracy (Bug fix time prediction)	MAE: 0.0605 MSE: 0.0050 R^2: 0.1000 MedAE: 0.055	MAE: 0.0608 MSE: 0.0051 R^2: 0.0861 MedAE: 0.0527	MAE: 0.0625 MSE: 0.0054 R^2: 0.0338 MedAE:0.0609	MAE: 0.0622 MSE: 0.0053 R^2: 0.0481 MedAE: 0.0596	MAE: 0.0616 MSE: 0.0052 R^2: 0.0685 MedAE: 0.0566	MAE: 0.0616 MSE: 0.0052 R^2: 0.0662 MedAE: 0.0579

TABLE VII. Risk Results For Software Components: *DOM, Layout, and XUL* That was marked as Excluded From The Bug Duplication Proposed Model.

Priority types	DOM (Actual)	*DOM* (After Excluding the related bug record)	*Layout* (Actual)	*Layout* (After Excluding the related bug record)	*XUL* (Actual)	*XUL* (After Excluding the related bug record)
No. of Fixed samples and acceptable by the machine	601	601	10	10	2017	2017
Duplicates categorized to *WONTFIX* records number	131	131	0	0	600	600
***WONTFIX* records number**	216	216	0	0	1083	1083
Number of Duplicates samples	601	601	10	10	932	931
Last sample predicted of bug-fix time value (days)	0.05022792	0.01656917	-0.02291683	-0.00983587	0.0657157	0.09041004
Bug-fix time average value (days)	0.08361111	0.08361111	0.10185185	0.10185185	0.08341601	0.08341601
RD	57.73%	57.73% (Excluded Bug is not within range)	0%	0% (Excluded Bug is not within range)	83.44%	83.44%
RB	60.07%	119.81%	-22.50%	-9.65%	78.78%	108.38%
TR	58.90%	38.77%	-11.25%	-4.82%	81.11%%	95.91%
Evaluation criteria accuracy (Bug fix time prediction)	MAE: 0.0586 MSE: 0.0049 R^2: 0.1208 MedAE: 0.0542	MAE: 0.0575 MSE: 0.0050 R^2: 0.1090 MedAE: 0.0545	MAE: 0.0594 MSE: 0.0052 R^2: 0.3550 MedAE: 0.0485	MAE: 0.0597 MSE: 0.0053 R^2: 0.3386 MedAE: 0.0493	MAE: 0.0652 MSE: 0.0054 R^2: 0.0293 MedAE: 0.0655	MAE: 0.0661 MSE: 0.0056 R^2: -0.0052 MedAE: 0.0683

also means that the excluded bug record was not in range to be included in the calculation.

5. Discussions

From Table IV, the proposed module gave low False-positive values from 0.04% to 7.46% for all the three similarity measures. The module succeeded to identify the bug duplication record for the *Javascript Engine* for a maximum of 98.2% precision percentage and the *Widget* component for a maximum of 99.8%. The similarity percentage measure of the *Widget* component gave better results than the *Javascript Engine* reaching more than 100% and giving up to 233.3%. The *Build Config* software component gave the highest of the lowest results with a 44.4% similarity percentage across all of the three measures. The *Graphics*, *Layout*, and *Event Handling* software components gave the lowest similarity percentage and precision with 0% for both.

From Table V, the most of duplication values similarity percentages are within 0% and 25% for all the software components. Generally, the count will decrease as we go for more percentage reaching up to 100% and more with the exception of *Javascript Engine*, *DOM*, *Widget*, and *Layout* software components. By showing that the count decreased for a higher percentage, it means the module is able to exclude a variety of false category types at below 50% similarity percentage.

From Table VI and VII, the numbers of duplicated bug records along with software risk results are illustrated. The most values of total software risk among the software components tend to increase since the total of duplicated samples is decreased by the excluded bug record identified by the module.

The *TR* value can reach up to 95.9% for the *XUL* software component. The lowest *TR* value given is for the Layout component reaching -4.82% because of minus *RB* values predicted by the bug-fix time prediction module. The risk values are only getting decreased from the actual risk values for the components *Event Handling* and *DOM*.

6. Threats To Validity

Invalid words contained in some bug duplication records such as technical expressions, alphanumeric variables, or some module names are eliminated by the approach used since they are not considered valid words. If these words were considered, the accuracy results could be improved. Furthermore, we have used a fixed Window size variable in this work and by manipulating this value then more training words can be taken and improve the results accuracy more. In addition, the samples taken are selected to fit specific

text comparison criteria, and by choosing different bug text records, then accuracy rates could get improved [36]. Finally, the hardware limitation on the used bug-fix time prediction study [28] has eliminated some records from the calculation and could affect the final risk results obtained with its bug-fix time value accuracy.

7. Conclusions

Two similarity measures have been proposed in this study to identify bug duplication records using the Word2Vec embedding approach with maximum precision accuracy of 99.89%; compared to the work by Deshmukh, J. *et al.* [20] to give a maximum of 90% accuracy. Furthermore, we have used the data of bug-fix time, and bug duplication to estimate the total software risk for specific software components and analyze their relationship with proposed automated bug duplication identification. We have calculated software risk values per software component where it shows 4 components of the selected dataset components showed an increase in software risk giving 66% percentage value when removing the duplicated related bug from the dataset that includes the *RD* calculation. This similarity detection proposed approach could be used by other researchers to provide different techniques and insights on Word2Vec text similarity methods to identify bug duplication issues across bug reports.

References

[1] S. R. Dalal, J. R. Horgan, and J. R. Kettenring, "Reliable software and communication: software quality, reliability, and safety," in *Proceedings of 1993 15th International Conference on Software Engineering*. IEEE, 1993, pp. 425–435.

[2] R. R. Lutz, "Analyzing software requirements errors in safety-critical, embedded systems," in *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*. IEEE, 1993, pp. 126–133.

[3] B. Kitchenham and S. L. Pfleeger, "Software quality: the elusive target [special issues section]," *IEEE software*, vol. 13, no. 1, pp. 12–21, 1996.

[4] G. McGraw, "Software assurance for security," *Computer*, vol. 32, no. 4, pp. 103–105, 1999.

[5] M. S. Rakha, W. Shang, and A. E. Hassan, "Studying the needed effort for identifying duplicate issues," *Empirical Software Engineering*, vol. 21, no. 5, pp. 1960–1989, 2016.

[6] J. Fisher, D. Koning, and A. Ludwigsen, "Utilizing atlassian jira for large-scale software development management," Citeseer, Tech. Rep., 2013.

[7] N. Serrano and I. Ciordia, "Bugzilla, itracker, and other bug trackers," *IEEE software*, vol. 22, no. 2, pp. 11–13, 2005.

[8] Y. C. Cavalcanti, E. S. de Almeida, C. E. A. da Cunha, D. Lucrédio, and S. R. de Lemos Meira, "An initial study on the bug report duplication problem," in *2010 14th European Conference on Software Maintenance and Reengineering*. IEEE, 2010, pp. 264–267.

[9] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful... really?" in *2008 IEEE International Conference on Software Maintenance*. IEEE, 2008, pp. 337–345.

[10] A. Lazar, S. Ritchey, and B. Sharif, "Improving the accuracy of duplicate bug report detection using textual similarity measures," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 308–311.

[11] Y. Tian, C. Sun, and D. Lo, "Improved duplicate bug report identification," in *2012 16th European conference on software maintenance and reengineering*. IEEE, 2012, pp. 385–390.

[12] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 2008, pp. 52–61.

[13] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 461–470.

[14] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 2007, pp. 499–510.

[15] D. Guthrie, B. Allison, W. Liu, L. Guthrie, and Y. Wilks, "A closer look at skip-gram modelling." in *LREC*, vol. 6. Citeseer, 2006, pp. 1222–1225.

[16] Y. Homma, S. Sy, and C. Yeh, "Detecting duplicate questions with deep learning," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS*, 2016.

[17] L. Ma and Y. Zhang, "Using word2vec to process big text data," in *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 2015, pp. 2895–2897.

[18] D. Jatnika, M. A. Bijaksana, and A. A. Suryani, "Word2vec model analysis for semantic similarities in english words," *Procedia Computer Science*, vol. 157, pp. 160–167, 2019.

[19] M.-J. Lin, C.-Z. Yang, C.-Y. Lee, and C.-C. Chen, "Enhancements for duplication detection in bug reports with manifold correlation features," *Journal of Systems and Software*, vol. 121, pp. 223–233, 2016.

[20] J. Deshmukh, K. Annervaz, S. Podder, S. Sengupta, and N. Dubash, "Towards accurate duplicate bug retrieval using deep learning techniques," in *2017 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 2017, pp. 115–124.

[21] Q. Xie, Z. Wen, J. Zhu, C. Gao, and Z. Zheng, "Detecting duplicate bug reports with convolutional neural networks," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2018, pp. 416–425.

[22] L. Kang, "Automated duplicate bug reports detection-an experiment at axis communication ab," 2017.

[23] A. Kukkar, R. Mohana, Y. Kumar, A. Nayyar, M. Bilal, and K.-S. Kwak, "Duplicate bug report detection and classification system based on deep learning technique," *IEEE Access*, vol. 8, pp. 200 749–200 763, 2020.

[24] J. He, L. Xu, M. Yan, X. Xia, and Y. Lei, "Duplicate bug report detection using dual-channel convolutional neural networks," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 117–127.

[25] B. J. Williams, J. C. Carver, and R. B. Vaughn, "Change risk assessment: Understanding risks involved in changing software requirements." in *Software Engineering Research and Practice*. Citeseer, 2006, pp. 966–971.

[26] G. Kaur and K. Bahl, "Software reliability, metrics, reliability improvement using agile process," *International Journal of Innovative Science, Engineering & Technology*, vol. 1, no. 3, pp. 143–147, 2014.

[27] J. Zhu, *BugRepo*, July 2018 (accessed November, 2019), https://github.com/logpai/bugrepo.

[28] H. Mahfoodh and M. Hammad, "Bug-fix time prediction using non-linear regression through neural network," in *3rd Smart Cities Symposium (SCS 2020)*, vol. 2020. IET, 2020, pp. 622–627.

[29] H. Mahfoodh and Q. Obediat, "Software risk estimation through bug reports analysis and bug-fix time predictions," in *2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)*. IEEE, 2020, pp. 1–6.

[30] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[31] B. Gao and L. Pavel, "On the properties of the softmax function with application in game theory and reinforcement learning," *arXiv preprint arXiv:1704.00805*, 2017.

[32] J. Dattorro, *Convex optimization & Euclidean distance geometry*. Lulu. com, 2010.

[33] L. Hiew, "Assisted detection of duplicate bug reports," Ph.D. dissertation, University of British Columbia, 2006.

[34] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[35] A. Joshi, *Word2Vec by implementing it in tensorflow*, July 9, 2017 (accessed February, 2020), https://towardsdatascience.com/learn-word2vec-by-implementing-it-in-tensorflow-45641adaf2ac.

[36] S. Gandrabur, G. Foster, and G. Lapalme, "Confidence estimation for nlp applications," *ACM Transactions on Speech and Language Processing (TSLP)*, vol. 3, no. 3, pp. 1–29, 2006.

Hussain Mahfoodh is a M.Sc. Software Engineering student in the college of IT at the University of Bahrain. He received his B.Sc. in Computer Engineering from University of Bahrain. He has more than 10 years experience working in software development and technical operations in different business fields (Software, Government, Telecommunication, Retail, and multiple Startups). His research interest includes Artificial intelligence, software engineering, and computer security.

Mustafa Hammad is an Associate Professor in the Department of Computer Science at the University of Bahrain. He received his Ph.D. in Computer Science from New Mexico State University, USA in 2010. He received his Masters degree in Computer Science from Al-Balqa Applied University, Jordan in 2005 and his B.Sc. in Computer Science from The Hashemite University, Jordan in 2002. His research interests include machine learning, software engineering with focus on software analysis and evolution.