International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 8, No.5 (Sep-2019)
http://dx.doi.org/10.12785/ijcds/080503

Predicting Software Faults Based on K-Nearest Neighbors Classification

Mustafa Hammad[1], Abdulla Alqaddoumi[1], Hadeel Al-Obaidy[1] and Khalil Almseidein[2]

[1]*Department of Computer Science, University of Bahrain, Sakheer, Bahrain*
[2]*Department of Computer Science, Mutah University, Al-Karak, Jordan*

Received 8 Apr. 2019, Revised 15 Jul. 2019, Accepted 28 Aug. 2019, Published 1 Sep. 2019

Abstract: Software defect prediction is one of the most important task during the development of software systems in order to save developers' time and effort. Discovering defects in an early stage of software development will allow programmers and developers to take action and resolve these faulty parts in software before its launch. In this paper, the K- Nearest Neighbor (KNN) machine learning algorithm is used to predict faulty software projects. Experimental studies are conducted on five public datasets with different similarly measures. Results showed that KNN can be used to predict software faults with accuracy rate that can achieve up to 87.2%.

Keywords: Software Defect Prediction, Software Engineering, Software Faults, Mccabe, Halstead, KNN, Software Metrics.

1. INTRODUCTION

The use of software has increased substantially in the last two decades. Software has become in demand in all areas of life, such as education, banking, medicine and many different areas. The maintenance of software systems costs more than its establishment. This means that finding and correcting errors in software will be more expensive when discovered too late.

The software defects in the computer field is the occurrence of the slightest mistake in one of the phases of program analysis, design or implementation, resulting in mistakes or errors that may adversely affect the performance and correctness of the software system. It is worth mentioning the Y2K bug, the famous programmatic error that would cause disastrous effects if infected systems were not fixed. There are defects that do not show its effects on software. These defects may cause damage to humans, especially in the field of arms, medicine and banks. For example, intrusion systems depend on the existence of a loophole in software mainly.

This was reflected positively on the improved quality of software, to increase the ability of developers and to improve the software Reliability by reducing energy and raising efficiency [1]. Machine learning is used in testing repository software, such as defect, effort, forecast changes missing, reusing code, etc. Software consists of segments classified as correct or faulty. This classification is based on machine learning techniques. Coders and developers focus their attention on the faulty segments. After a fault is detected, they use available resources to check those segments, to propose solutions for those problems.

Recently, many of the studies have talked about estimating software defects. Artificial intelligence techniques were used to build models that identify defects through certain software features [2] which is considered one of the most important applications of machine learning that have been successfully applied to solve a lot of issues and problems in various areas such as classification, identification and processing.

There are many modern methods for error classification, such as K-Nearest Neighbor (KNN), Support Vector Machine (SVM), genetic algorithms, etc. These methods use the historical record of data to predict future data errors, this method works on the historical record of the data to build the scientific predictions about future data based on similarities existing between the data. Some use KNN as a classifier for defect prediction. KNN uses different types of distances for computing closeness such as Euclidean distance or Manhattan distance. In this paper, KNN with distance measure is used for defect prediction.

The remainder of this paper is organized as follows; Section 2 presents the literature review related to software

E-mail address: mhammad@uob.edu.bh, aqaddumi@uob.edu.bh, halobaidy@uob.edu.bh, Khalil426a@gmail.com

defect prediction followed by the proposed classification methodology, which is presented in Section 3. Section 4 discusses the used datasets and Section 5 presents the experimental results. Finally, Section 6 concludes the paper and presents possible future works.

2. Related Work

D'Ambros [3] made contributions to the field of software defect prediction by providing predictive data that has been applied to five open source systems with five periods. The file generation contains the results of examination for several versions of the systems and different periods per week. They adopted a 17-element metric of the famous performance metrics. Results reached to 90% after five applied to systems using Shannon entropy approved over the interval of the entropy (Shannon entropy) and reach the result that the best performance by techniques weighted Churn of source code metrics (WCHU) and Linearly Decayed Entropy of source code metrics (LDHH). But this method needs a large amount of data and calculations using the entropy.

Gray et al. [4] used the SVM metrics (MDP) that contains a set of 13 custom datasets and focused on PC2 working on major function to count the number of lines of code. The system scored better true positive forecasts than the false positive forecasts for groups of 13 NASA datasets.

Guoshun, and Wang [5] used multi-agents in the autonomy of the software in fault diagnosis by applying Spectrum-based Fault Localization (SFL) and setting performance standards on the source file and output the error and correct statistical.

Lessmann et al. [6] performed tests to predict the 22 works on the 10 datasets from NASA MDP warehouse and concluded that the use of the results for accuracy closest Naïve Bayes.

Kaur and Kaur [7] used the working principle of area under curve (AUC) using three methods: Bagging, Boosting, and Rotation Forest using 15 mechanisms for learning to predict errors and demonstrate results through operating characteristics curve after training in 9 groups of data and to adopt performance measures 20 by component and automated software defect prediction models (ASDPMS) and by (AUC) results show The proportion of NB with Poi, 89%. Reached the forest rotation is more beneficial to reduce the number of performance measures from other methods (Bagging, Boosting).

Zimmermann et al. [8] predicted software faults from a defect dataset of Eclipse to source code locations. Malhotra et al. [9] presented a new method using Artificial Neural Network (ANN) to predict software defects. The authors used text mining techniques, feature extraction and Radial Basis Function to estimate software defects. Many of studies proposed a Bell function based on Multilayer perceptron network, such as the one used in [10], where Gayathri et al. compared the results using various machine learning methods and achieved an accuracy of 98.2%.

Selvaraj et al. [11] used the Support Vector Machine for predicting software fault, they compared its performance of with Naïve Bayes model and Decision stumps. SVM performed better than Naïve Bayes model and Decision stumps. Also, Singh et al. [12] used SVM to predict the defects of software, but their method was used only on object-oriented large systems.

3. Classification Methodology

This section discusses the used classification algorithm, as well as, the used similarity measures.

A. K Nearest Neighbors (KNN)

KNN is a classification algorithm that depend on N vectors. Used datasets have to transform open source code into suitable representation for the learning algorithm [13]. They represented the code as a vector based on sets of criteria such as design complexity, essential complexity so on. To define a k-Nearest Neighbors classifier, the distance metric used to measure how close two vectors are to each other need to be defined [14].

Suppose that V is a vector that need to be classified. Then, the value of K determines the number of nearest neighbors to vector V. Fig. 1 shows an example when the value of K is 11 and 5. The classifier selects the k vectors in the training set that are closest to C, then it assigns whether the vector is a defect or not based on selected k vectors. Fig. 1 shows a classification example based on the KNN algorithm. In this example, when the value of k is equal to 5, the algorithm will investigate the nearest 5 neighbors for the vector V.

We opted for the KNN to classify each category in a tested class to defect or non- defect classes. The input consists of 22 attributes, for k =10, and two output 0 or 1, as illustrated in Algorithm 1.

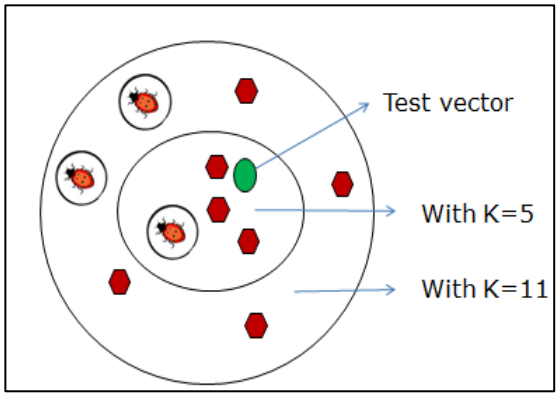

Figure 1. K-nearest Neighbors classification example

Algorithm 1: the proposed training algorithm.

Step 1: Read the text file from the training set folder of software defect dataset from PROMISE repository.

Step 2: Split the training dataset according to k-cross validation.

Step 3: Create test dataset with N features; N is the total number of features.

Step 4: Save each feature subset in new text file.

Step 5: Train KNN on the data saved in the previous text file.

Step 6: Calculate the accuracy for all classes.

Figure 2 shows the main steps of the proposed fault prediction model, which are mainly divided into six stages. First, read report of defects from open source stage. After that, the features extraction stage is executed, which includes basic complexity, independence of the program, complexity of design, and lines of code metrics. Then, the process of building the vectors is used to generate a training dataset for the prediction model. The model is trained for features extraction to classify the data. In the classifier stage, a KNN classifier is defined to calculate the distance based on Euclidean Distance, Manhattan Distance, Hausdorff Distance or Weighted Euclidean Distance. In the last stage, the decision on the classification of the class to be a defect or non-defect is made.

B. Similarity Measures

There are many similarity measures that can be used int the KNN algorithm. These metrics measure the degree of closeness between vectors.

There is no ideal measure for all kind of vectors and problems. This is due to the nature of the vectors. So, selecting an appropriate similarity measure is essential. There are several ways to compare between vectors and measure the proportion of similarities between them. In this paper, Euclidean, Manhattan and Weighted Euclidean, and Hausdorff distances were used as such measures. The following subsections review these measures.

1) Euclidean Distance Metric

Euclidean distance is one of the most commonly used metric in software defect prediction problem. In this metric, the vectors are represented as a single point in the D-dimensions space, then the distance between them is calculated using the following equation:

$$ED(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \qquad (1)$$

where x and y are vectors.

2) Manhattan Distance Metric

Manhattan distance metric, also called Taxicab geometry [15,16,17], is calculated by finding the length between vectors that represented in the X and Y axes. The most common example of this method is a taxi between cities, so the distance traveled by car will not differ, in case it is advancing towards other city [18]. The value of Manhattan Distance MD between two vectors x and y can be calculated by the following equation:

$$MD\ (x, y) = \sum_{i=1}^{n}|x_i - y_i| \qquad (2)$$

3) Weighted Euclidean Distance Metric

It is a common similarity measure used to calculate similarity between vectors. In this distance, the vector is represented as a single point in the D-dimensions space, then calculate the distance between them hit weight for point by the square root of the sum total of the difference between those two points box. The value of weighted Euclidean distance metric between vectors a and b can be calculated as follows:

$$D(a, b) = \sqrt{\sum_{i=0}^{n} W_i(x_i - y_i)^2} \qquad (3)$$

where $0 < W_i < 1$.

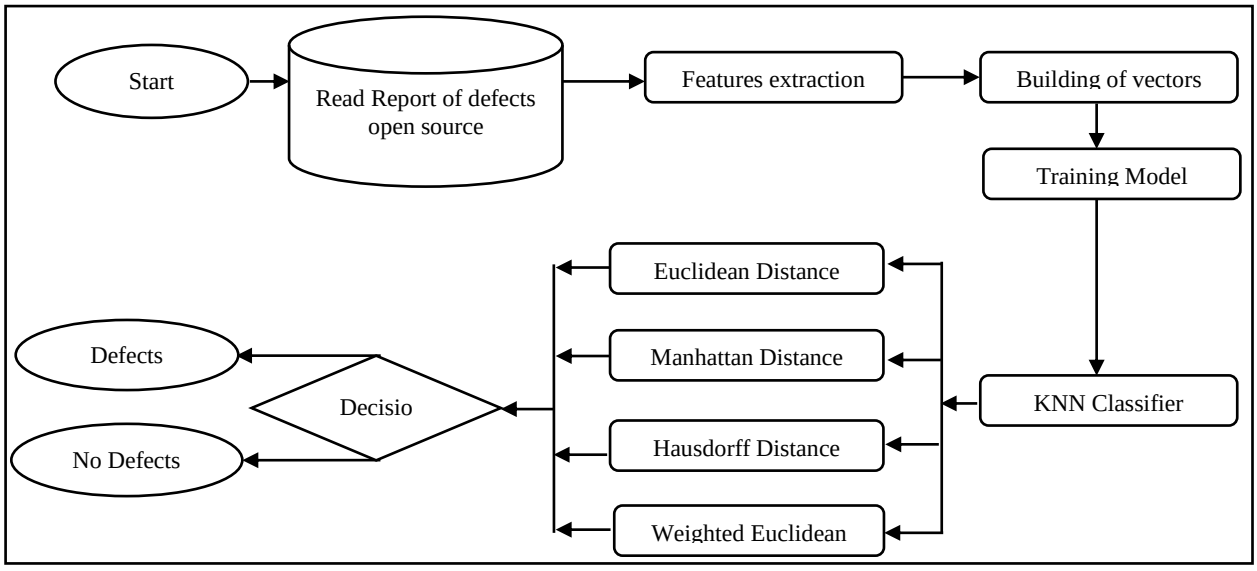

Figure 2. Main steps of the proposed classification

4) Hausdorff Distance Metric

One of metrics to calculate the distance between two points according to the following equation:

$$D(X, Y) = \text{Max}\,(D(x, y), D(y, x)) \qquad (4)$$

where x and y are points of the sets X and Y respectively, and D(x, y) is any distance metric between these points. For simplicity, Euclidian Distance will be used to calculate D(x, y) as the distance between x and y [19].

4. USED DATASET

The systems used five datasets: cm1, jm1, kc1, kc2 and pc1. They are collected from different software projects in NASA. These datasets are available for research purposes at (http://promise.site.uottawa.ca/SERepository/datasets-page.html). Table 1 summarizes the characteristics of the datasets used.

TABLE I. CHARACTERISTICS OF USED DATASETS

Dataset	Programming Language	Kilo- Line of Code (KLOC)	No. of modules	No. of defected modules
CM1	C language	20	505	48
KC1	C++ language	43	2107	325
PC1	C language	40	1107	76
KC2	C++ language	18	522	105
JM1	C language	315	10878	2102

The datasets contain static code measures, which include Halstead, McCabe, and LOC metrics along with the defect rate. The datasets CM1, KC1, KC2, PC1 and JM1 are publicly available in the NASA repository by NASA Metrics Data Programmer and are the most commonly used datasets for defect prediction. They are used in 60% of the selected primary studies used for software development, which includes four criteria for arbitration: Basic complexity, the independence of the program, the complexity of the design and Lines of Code (LOC).

To ensure the quality of the software and get the best productivity, the software needs to be tested before marketing it through the study of the complexity of the software. Nevertheless, there are many features of software that cannot be tested in the same scale. There are many studies made in the field of software metrics. The most common metrics are McCabe, Halstead, Line Count, Operator, and Branch Count. In [20], McCabe presented a new approach, McCabe measure, for software metrics that discussed the complexity of software. The McCabe metrics consists of the following four measurements: essential complexity, cyclomatic complexity, design complexity and Lines of Code (LOC). In [21], Maurice presented a new model measure, Halstead, to test software without the implementation of the code. It included three measurements: the base measures, the derived measures, and lines of code measures. Nguyen et al [22] presented an algorithm for Source lines of code (SLOC). This algorithm is used to measure the size of software by calculating the number of lines in the source code.

Table 2 presents the 21 field structure of the used dataset. These field are a set software measures, which depend on the software product, complexity, and size of the vocabulary. The attribute of number of lines in code, indicate the lines of code that are executable, the lines of comments, lines that contain all the code, and comments. For complexity measurement, it includes cyclomatic, complexity, essential complexity, and module design complexity. Twelve of the remaining measure are used to measure vocabulary. These matrices consist of the following measures: Halstead programming effort, number of unique operators, Halstead level, Halstead intelligent content, Alstead difficulty, Halstead length, total operators, Halstead error estimate, Halstead programming time, Halstead volume, number of unique operands, and total operands. Table 2 summarizes the class attribute for each data set refers to the defect classification, the last attribute represents classify category where it will wither false or true we expressed 0 or 1.

5. EXPIREMENTAL RESULTS

The accuracy of a defect predication model is usually measured in terms of its effectiveness. The process of correct classification is the key factor for any classification model. In this paper, we used the accuracy measure to evaluate the KNN defect predication model. Accuracy rate is calculated as follows:

$$\text{Accuracy Rate: } = \frac{|TN|+|TP|}{|TN|+|TP|+|FN|+|FP|} \qquad (5)$$

where, as shown in Table 3, True positive (TP) is the number of vectors that is should be retrieved as related to a class and retrieved. True Negative (TN) is the number of vectors that is should not retrieved as related to a class and not retrieved. False Positive (FP) is the number of vectors that is not related to a class but retrieved, and finally, False Negative (FN) is the number of classes that is not retrieved as related to a category but should be. Table 3 shows the structure of the confusion matrix.

In this study, the accuracy percentages are calculated based for different classifiers on the five datasets with k=10. Based on many experiments, the highest accuracy rate was achieved when the classifier parameter K is set to equal 10. Table 4 shows the average, maximum, minimum, and the slandered deviation predication accuracies over the five datasets for different KNN classifiers. As shown in the table, the highest accuracy rate was achieved by using the KNN classifiers with ED similarity measure with 93.9% as a maximum accuracy and 87.2% as an overall average accuracy rate. The lowest rate was when we use the Hausdorff similarity measure with 58.8% overall average accuracy. We believe this is because the nature structure of the dataset attributes, which may affect the training process significantly based on the used similarity measure.

TABLE II. THE USED DATASET ATTRIBUTES

#	Metric Type	Definition	Attribute	Type
1	McCabe's	line count of code	loc	numeric
2	McCabe	cyclomatic complexity	v(g)	numeric
3	McCabe	essential complexity	ev(g)	numeric
4	McCabe	design complexity	iv(g)	numeric
5	Halstead	total operators + operands	N	numeric
6	Halstead	Volume	v	numeric
7	Halstead	program length	L	numeric
8	Halstead	Difficulty	D	numeric
9	Halstead	Intelligence	I	numeric
10	Halstead	Effort	E	numeric
11	Halstead	Effort Estimate	b	numeric
12	Halstead's	time estimator	T	numeric
13	Halstead's	line count	lOCode	numeric
14	Halstead's	count of lines of comments	lOComment	numeric
15	Halstead's	count of blank lines	lOBlank	numeric
16	LOC	LOC	lOCodeAndComment	numeric
17	Operator	unique operators	uniq_Op	numeric
18	operands	unique operands	uniq_Opnd	numeric
19	Operator	total operators	total_Op	numeric
20	operands	total operands	total_Opnd	numeric
21	branch count	of the flow graph	branchCount	numeric
22	*Name of class*	*defect / no defect*	*True / false*	*Boolean*

TABLE III. THE CONFUSION MATRIX

	Observed True	Observed False
Predicted True	True Positive (TP)	False Positive (FP)
Predicted False	True Negative (TN)	False Negative (FN)

To have a closer look to the collected accuracy rate for each dataset. Figure 3 presents the accuracy rate for each classifier on the used datasets.

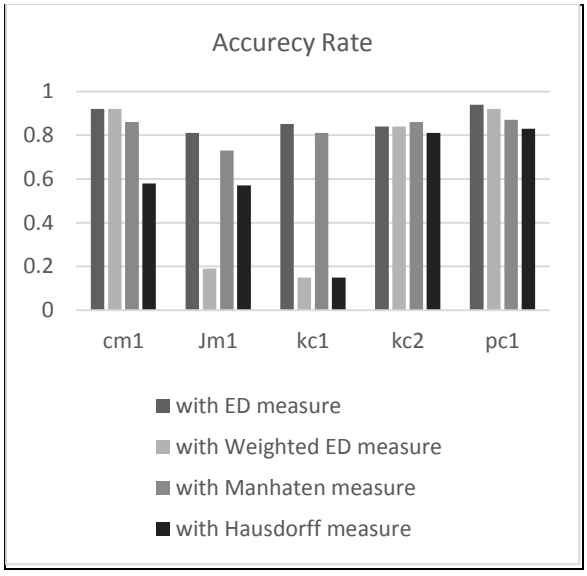

Figure 3. Accuracy rate for KNN algorithm with different similarity measures for the five datasets

6. CONCLUSION

In this paper, the K-nearest neighbor machine learning algorithm is used to predict software defects based on software metrics. Four different similarity measures are used to build different classifiers. These measures are Euclidian distance, weighted ED, Manhattan distance, and Hausdorff distance measures. The generated prediction models are evaluated based on five public datasets. The results showed that the fault prediction model is dependable and the KNN algorithm can be used to classify software faults. The highest average accuracy rate was 87.2%, which achieved when the Euclidian distance measure is used to build the KNN algorithm. Using more similarity measures, datasets, and classifiers is one way to extend this work. Moreover, finding the impact of each software metrics in the model can increase the prediction accuracy rate.

REFERENCES

[1] Koru, A. Gunes, and Hongfang Liu. "Building effective defect-prediction models in practice." Software, IEEE 22.6 (2005): 23-29.

[2] Al-Jamimi, Hamdi A., and Lahouari Ghouti. "Efficient prediction of software fault proneness modules using support vector machines and probabilistic neural networks." Software Engineering (MySEC), 2011 5th Malaysian Conference in. IEEE, 2011.

[3] D'Ambros, Marco, Michele Lanza, and Romain Robbes. "An extensive comparison of defect prediction approaches." In Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on, pp. 31-41. IEEE, 2010.

[4] Gray, David, David Bowes, Neil Davey, Yi Sun, and Bruce Christianson. "Software defect prediction using static code metrics underestimates defect-proneness." In Neural Networks (IJCNN), The 2010 International Joint Conference on, pp. 1-7. IEEE, 2010.

[5] Chen, Guoshun, and Gefang Wang. "Software Fault Diagnosing System Based on Multi-agent." In Intelligent Systems (GCIS), 2012 Third Global Congress on, pp. 327-329. IEEE, 2012.

[6] Lessmann, Stefan, Bart Baesens, Christophe Mues, and Swantje Pietsch. "Benchmarking classification models for software defect prediction: A proposed framework and novel findings." Software Engineering, IEEE Transactions on34, no. 4 (2008): 485-496.

[7] Kaur, Amardeep, and Kaur, Kanwalpreet. "Performance analysis of ensemble learning for predicting defects in open source software." In Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on, pp. 219-225. IEEE, 2014.

[8] Zimmermann T, Premraj R, Zeller A. Predicting defects for eclipse. In Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on 2007 May 20 (pp. 9-9). IEEE.

[9] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures." in Proceedings of the International Conference on Software Engineering (ICSE 2006), Shanghai, China, 2006.

[10] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches." IEEE Trans. Software Eng., vol. 22, pp. 886-894, 1996.

[11] Selvaraj, P, Thangaraj, P, "Support Vector Machine for Software Defect Prediction" International Journal of Engineering & Technology Research, Vol. 1, Issue 2, pp. 68-76, 2013.

[12] Y.Singh, A.Kaur, R.Malhotra, "Software Fault Proneness Prediction Using Support Vector Machines," In Proceedings of the World Congress on Engineering, Vol. 1, 2009

[13] J. Han and M. Kamber, "Data mining: concepts and techniques", 2nd ed. The Morgan Kaufmann Series, (2006).

[14] T. Divya and A. Sonali, "A survey on Data Mining approaches for Healthcare", International Journal of BioScience and Bio-Technology, vol. 5, no. 5, (2013), pp. 241-266

[15] Dickau, R. M. "Shortest-Path Diagrams. " http://mathforum.org/advanced/robertd/manhattan.html.

[16] Krause, E. F. Taxicab Geometry: An Adventure in Non-Euclidean Geometry. New York: Dover, 1986.

[17] Skiena, S. Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Reading, MA: Addison-Wesley, pp. 172 and 227, 1990.

[18] Willard, S. General Topology. Reading, MA: Addison-Wesley, p. 16, 1970.

[19] http://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/98/normand/main.html

[20] McCabe (December 1976). "A Complexity Measure". IEEE Transactions on Software Engineering: 308–320. doi:10.1109/tse.1976.233837.

[21] Halstead, Maurice H. (1977). Elements of Software Science. Amsterdam: Elsevier North-Holland, Inc. ISBN 0-444-00205-7.

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.

Abdulla Alqaddoumi received his B.Sc. in Computer Science from University of Bahrain, Kingdom of Bahrain in 2002. He completed his M.Sc. in Advanced Computing Science at University of Manchester, UK in 2005. He completed his Ph.D. from New Mexico State University in 2016. Dr. Abdulla Alqaddoumi is currently Assistant Professor in the Department of Computer Science at the University of Bahrain. His research interest includes Knowledge Representation, Optimization, Parallel Processing, Programming Languages and Graph Theory.

Hadeel Alobaidy is an Assistant Professor at University of Bahrain, College of Information Technology, Department of Computer Science, specializing in Software Engineering and web engineering. Her current research interests Human Computer interaction, software Engineering, Web engineering, Ontology, Semantic Web and Data Mining.

Khalil Al-Mseidein received his B.Sc. in Computer Science from the Al-Hussein Bin Talal University – Jordan in 2007. He completed his M.Sc. in Computer Science at Mutah University in 2017. He is currently the head of the Technical Support Center at the Ministry of Justice – Jordan. His research interests include artificial intelligence, digital image processing, and software engineering.