

An Accumulated Cognitive Approach to Measure Software Complexity

Mohammed A. Shehab, Yahya M. Tashtoush, Wegdan A. Hussien, Mohammed N. Alandoli, and Yaser Jararweh

Jordan University of Science and Technology, Irbid, Jordan

Email: {mashehab12, wahussien13, mnalandoli13}@cit.just.edu.jo, {yahya-t, yijararweh}@just.edu.jo

Abstract—The complexity of a program or software can create many difficulties during its lifetime. This complexity entails increased time and effort requirements maintain the code, or discover errors and defects. All of which will lead to an increase in the overall cost of the project. So that, software engineers and developers measure the complexity of program code before they start any project. This paper proposes a novel weighted complexity metric to measure code complexity by using six main attributes. Two of them are a mixture of Cyclomatic, Halsted, and Shao and Wangs metrics. The dataset of this research consists of 15 programs written in Java programming language, and collected from different websites. The programs were ranked by seven experts in Java programming language. Our metric was able to achieve 94% accuracy for results.

Index Terms—complexity metric, cyclomatic, halsted volume, Shao and Wangs metric, software engineering

I. INTRODUCTION

Software complexity is one of most important concerns in software lifetime development. If the code is complex, then the developers will likely face more problems while developing and maintaining it. Thus, most software companies focus on this issue prior to commencing any project. The lower the complexity of an application, the easier is it to measure its various other factors. In addition, errors and defects are easier to discover and repair, and the cost of many factors will be reduced. Today, software engineering needs to accurately predict the complexity of application to save millions in maintenance time and effort [1].

Many metrics are used to measure application complexity such as Cyclomatic Metric, Halsted Volume, and Shao and Wangs Cognitive Functional Size [1] [2]. Each one of these metrics focuses on some features in the source code. For example, Cyclomatic metric focuses on some key features, such as loops and condition controls by drawing a flow diagram for the program and calculating the number of nodes and edges [3]. Shao and Wangs Cognitive Functional Size uses the same

technique to calculate program complexity, but they give different weights for each control, depending on its complexity [2]. On the other hand, Halsted focuses more on number of operators and operands in program source code. Also line of code is one of the most famous metrics that used to calculate complexity of application by calculate its lines of code [4].

However, the aforementioned metrics still have certain drawbacks, so in this research we proposed a new metric to calculate complexity of programs by using some attributes from Cyclomatic Metric, Halstead Volume, and Shao and Wangs Cognitive Functional Size metrics, with new addition attributes in order to increase measurement accuracy and minimize the drawbacks of the traditional metrics.

The paper is ordered as follows: section 2 provides studies on measuring complexity in software. In section 3, we explain three traditional metrics, and in last subsection we explain our novel metric and its attributes. Then, we investigate the results for all four metrics, and analyze them in the Results Discussion section 4. Finally, conclusion and future work are outlined in Section 5.

II. RELATED WORKS

Before the 21st century, many metrics were proposed to measure software complexity. Among the most widely used were McCabe's Cyclomatic complexity [5], Halstead complexity [6], and Line of Code. On other hand, few studies were conducted to compare between different complexity metrics to determine which metric is more suitable to be considered in software engineering.

Graylin Jay *et al.* [4] made a comparison between two metrics to calculate the complexity of code. They used Cyclomatic Complexity (CC) and Line of code (LOC) to prove the stable linear relationship between these two techniques. They used five NASA projects with different programming languages, such as C, C++, and Java, to be the dataset for their research. Finally, they found that these two measurements are severely underestimated and CC does not have any different features from the LOC.

Min Zhang *et al.* [7] studied the performance of three complexity metrics. They selected McCabes Cyclomatic

Complexity (CC), Halsteads Complexity (HC), and Douces Spatial Complexity (DSP). Their experiment is based on four hypotheses using data from Eclipse JDT which is an open source application. As a result, they conclude that the three complexity metrics show different performance results during their hypotheses testing, and finally they recommended combining Cyclomatic and Halstead metrics for better judgments on software complexity.

At the beginning of 21th century, researchers started new studies to measure software complexity based on cognitive informatics. They believed that complexity metrics that are based on cognitive informatics would offer promising solutions to measure software complexity.

D. I. De Silva *et al.* [1] study the applicability of three software complexity metrics: McCabe's Cyclomatic complexity (CC), Halstead's complexity (HC), and Shao and Wang's (SW) cognitive functional size (CFS). In their study, they used ten different programs of the same programming language (java) and determined which one of the three complexity metrics is most appropriate for software manufacturing. In addition to manually calculating the ranking of the ten programs complexity based on the three metrics, they applied Quota sampling method by selecting five big companies and randomly asked six programmers from each company to rank the complexity of the ten programs. Finally, the Spearman's rank correlation coefficient is conducted to compare between the ranks from the manual calculation of the three metrics and the questionnaire of the thirty programmers. As a result, they confirmed that Shao and Wang's (SW) cognitive functional size is the best metric to be used.

D. I. De Silva *et al.* [8] made a comparison to test relationship between three cognitive complexity metrics: Kushwaha and Misra (KM's) cognitive information complexity measure (CICM), Shao and Wang's (SW's) cognitive functional size (CFS), and Misra's cognitive weight complexity measure (CWCM). As in [?] they used the same ten java programs and asked thirty expert developers from five huge companies to rank the programs from least to highest complexity. Finally, the Spearman's rank correlation coefficient is conducted to compare between the ranks resulted from the manual calculation of the three metrics and thirty experts judgments. As a result, they confirmed that Shao and Wang's (SW) cognitive functional size is the best metric to be considered in real world.

III. METHODOLOGY

Methodology In this section, we discuss the three old complexity metrics: Cyclomatic, Halsted Volume, and Shao and Wang's Cognitive Functional Size. After that, we discuss the new complexity metric and how works to calculate complexity of program. The dataset of this work was collected from a number of different websites

[9] [10], this dataset is a collection of fifteen programs that are written in Java programming language. We choose one programming language to avoid any complexity differences in the syntax of different programming languages. Table I shows the statistical of dataset collection.

TABLE I. THE STATISTICS OF DATASET COLLECTIONS

	Min	Max
Line of code	18	677

A. Cyclomatic Metric

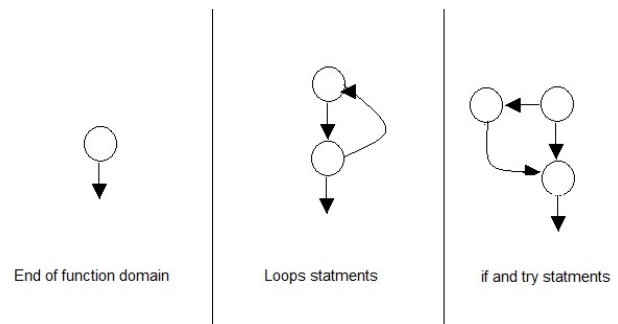


Figure 1. Cyclomatic direct graph

Cyclomatic is one of the most popular complexity metrics that uses a flow chart to represent application execution steps. There are many techniques to calculate program Cyclomatic complexity. The first one by a direct graph that has a number of nodes and edges. There are two types of nodes in this direct graph: normal nodes and predict nodes. The normal nodes are nodes that have one output, and those nodes are used to represent loops and end of function domain. On the other hand, the predict nodes are nodes which have two or more outputs and are used to represent a control keywords such as an IF statement, SWITCH, and TRY-CATCH statement [4]. Fig. 1 shows this representation. After drawing the direct graph of an application, the Cyclomatic complexity of application was calculated by (1).

$$\text{Cyclomatic complexity} = E - N + 2P \quad (1)$$

where:

E is the number of edges in direct graph.

N is the number of normal nodes.

P is the number of predict nodes

However, this is the traditional Cyclomatic calculation, and Microsoft uses another method to calculate it. The Microsoft method counts the number of IF, TRY, and loop keywords to get the Cyclomatic complexity [11]. In this paper, we use Microsoft method by build a simple compiler to detect and count those keywords automatically. Table II shows Cyclomatic complexity results for all dataset.

TABLE II. CYCLOMATIC COMPLEXITY OF ALL DATASETS

Program Name	Cyclomatic complexity	Rank
Bitmap	0	15
Factorial	1	14
md5 hashing algorithm	2	13
breadth first search algorithm	4	11.5
Date format	4	11.5
heap sort algorithm	5	10
depth first search algorithm	7	8.5
huffman codes	7	8.5
priority algorithm	8	7
Dijkstra's algorithm	10	6
A-Star	20	4.5
CSV to HTML translation	20	4.5
Self-organizing map (SOM)	23	3
Decision tree	30	2
Fuzzy Logic	76	1

TABLE III. HALSTED VOLUME OF ALL DATASETS

Program Name	Halsted volume	Rank
Bitmap	240	15
Factorial	548	14
Date format	2274	13
md5 hashing algorithm	4123	12
huffman codes	5121	11
Dijkstra's algorithm	8447	10
breadth first search algorithm	10405	9
CSV to HTML translation	10569	8
depth first search algorithm	11050	7
heap sort algorithm	12714	6
Self-organizing map (SOM)	16238	5
Decision tree	20596	4
A-Star	26498	3
priority algorithm	28834	2
Fuzzy Logic	236828	1

B. Halsted Volume Metric

Halsted volume is a traditional metric to calculate program complexity. This metric focuses on operators and operands in application [1] [12]. Using these two attributes, Halsted volume can be used to calculate the complexity volume of program V as shown in (2) Where N can be calculated by (3).

$$V = N \log_2 (n_1 + n_2) \quad (2)$$

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2 \quad (3)$$

where:

- n_1 Number of distinct operators in application
- n_2 Number of distinct operands in application
- N_1 Total of operators in application
- N_2 Total of operands in application







In addition, test effort can be calculated by using (4) after calculating program level using (5) [12]. Nevertheless, Halsted is not quite optimally accurate, because there are many factors which can affect program complexity, like number of functions, number of loops, IF statement, and so on. We used number of operators to be one attribute in this research for the new metric. The Halsted volume results for each program in dataset shown in Table III.

$$effort = \frac{V}{PL} \quad (4)$$

$$PL = \frac{1}{\left(\frac{n_1}{2}\right) * \left(\frac{N_2}{n_2}\right)} \quad (5)$$

C. Shao and Wangs Cognitive Functional Size

TABLE IV. SHAO AND WANGS COGNITIVE FUNCTIONAL SIZE

Category	Control name	Flow diagram	Weight
Sequence	Sequence step		1
Branch	If – else control		2
	Switch control		Number of cases
Iteration	Loops controls		3
Embedded	Call functions		2
	Recursion function		3

This complexity metric is similar to Cyclomatic metric in which both metrics use a flow chart to represent program steps. As mentioned earlier, this flow chart is a direct graph that has number of nodes and edges, but Shao and Wangs metric has two different techniques to

calculate program complexity. The first is that Shao and Wangs do not have predicted nodes, unlike Cyclomatic. The second difference is that each control in the program code has a different weight. Table IV shows how Shao and Wangs give a different weight for each control [2].

These differences in control weights give Shao and Wangs metric better accuracy to calculate complexity of application code than Cyclomatic metric. This is because some controls are more complex than others, such as loops and recursion functions [1]. Table V shows Shao and Wangs Cognitive Functional Size results for all programs in dataset.

TABLE V. COGNITIVE FUNCTIONAL SIZE OF ALL DATASETS

Program Name	Shao and Wang's Cognitive Functional Size	Rank
Bitmap	3	15
Date format	4	13.5
Factorial	4	13.5
md5 hashing algorithm	7	12
breadth first search algorithm	16	10.5
heap sort algorithm	16	10.5
huffman codes	23	9
depth first search algorithm	24	8
Dijkstra's algorithm	31	7
CSV to HTML translation	34	6
A-Star	41	5
Self-organizing map (SOM)	64	4
Decision tree	68	3
priority algorithm	79	2
Fuzzy Logic	136	1

D. New Complexity Metric

In this section, we describe our new approach to calculate program complexity. We developed this metric by using some methods from Cyclomatic, Halsted Volume, and Shao Wangs metrics to cover any shortcomings in these metrics. The new metric has six attributes:

Flow chart

- If statement
- Switch statement
- Iteration statements
- Recursion function
- Number of operations in program code
- Number of external libraries and functions
- Number of function arguments
- Number of variables declaration

- _ Local variables
- _ Global variables

– Number of calling functions (only for local functions)

We use the same technique of creating a flow chart, like Cyclomatic, but with different weights for each control, similar to Shao and Wangs metric. We used the same weights that were used in Shao and Wangs metric such as: loops, IF statement, recursion functions, and number of cases for SWITCH statement. By using this additional technique, we avoid the flaw in Cyclomatic metric where it gives the same weight for all controls, since WHILE loops and recursion functions are more complex than other controls. Table VI shows the weights for all flow chart controls.

TABLE VI. WEIGHTS OF FLOW CHART ATTRIBUTE OF NEW APPROACH

Flow chart controls	Weight
If statement	2
Switch statement	Number of cases
Iteration statements "Loops"	3
Recursion function	3

The second attribute in the new metric is the number of operations in program code. We use this attribute from Halsted Volume, and calculate it using (6). This attribute also has I/O operation with logical and arithmetical operations [12].

$$\text{number of operations} = \log_2 N \quad (60)$$

where N is the total number of all operations of program code [12].

Next, we add four new attributes for our new metric to improve its accuracy. We use the number of external libraries and functions, the number of function arguments, the number of variable declarations, and the number of calling functions. For variable declarations, we divided this attribute to two sub-attributes: local and global. This is because global variables are more complex than local variables and all functions can effect on their values, and thus tracking them requires more effort than local variables. The last attribute is for local functions only, because the external functions will be calculated by external libraries and functions attribute. Table VII shows the weights that we used for the new metric.

TABLE VII. ADDITION ATTRIBUTES IN THE NEW METRIC

Control	Weight
External library or function	1
Function arguments	Number of arguments + 1 for function declaration
Function call (for local functions only)	1
Variables declaration (local)	1
Variables declaration (global)	2

Then the total complexity of program is measured by getting the summation of all six attributes. Table VIII shows the results of the new metric.

TABLE VIII. NEW METRIC RESULTS FOR ALL DATASETS

Program Name	New Metric	Rank
Factorial	7	15
Date format	25	14
Bitmap	27	13
breadth first search algorithm	28	12
md5 hashing algorithm	30	11
CSV to HTML translation	49	10
heap sort algorithm	54	8.5
huffman codes	54	8.5
depth first search algorithm	63	6.5
priority algorithm	63	6.5
Dijkstra's algorithm	83	5
A-Star	98	4
Self-organizing map (SOM)	120	3
Decision tree	164	2
Fuzzy Logic	290	1

IV. EXPERIMENTS AND RESULTS

In this section, we discuss the experiment setup in the first subsection, after which we analyze the results of this work. In the experiment setup, we talk about the dataset, statistics, and how we rank programs by their complexity. After that, we study and analyze the results of code complexity for each metric.

A. Experiment Setup

This section discussed the experiment setup of the dataset in this research. First, we rank all programs in dataset from 1 to 15 by asking seven experts in Java programming language. After that, we calculate the average of all ranks for each program as shown on Table IX.

Next, we built a simple program to calculate the four complexity metrics: Cyclomatic complexity, Halsted volume, Shao and Wangs Cognitive Functional Size, and finally, our new metric. We built a simple compiler to calculate all the metric results values automatically. Then we calculate Spearman's rank correlation coefficient for each metric to measure the accuracy of metrics. We calculate a codes complexity using each metric, and organize them in ascending order, and give each program a rank. Finally, we calculate Spearman's rank correlation coefficient between each metric with the expert's rankings to get the closest ranking metric to expert's rankings.

TABLE IX. AVERAGE RANK OF DATASET PROGRAMS BY SEVEN EXPERTS IN JAVA PROGRAMING LANGUAGE

Program name	Average complexity	Spearman rank
Factorial	1.142857	15
Date format	2.428571	14
Bitmap	2.571429	13
CSV to HTML translation	4.571429	12
breadth first search algorithm	6	11
depth first search algorithm	6.428571	10
heap sort algorithm	6.714286	8.5
md5 hashing algorithm	6.714286	8.5
huffman codes	7.571429	7
Dijkstra's algorithm	7.857143	6
priority algorithm	8.714286	5
Decision tree	10.14286	3.5
A-Star	10.14286	3.5
Self-organizing map (SOM)	10.28571	2
Fuzzy Logic	12	1

B. Results Discussion

As shown in Table X, the worst ranking was for Halsted volume metric 81%. This is because Halsted metric does not mention many important controls such as loops. Because of that, Halsted cannot detect the real complexity for a program. Also, Cyclomatic metric got approximately 82% of accuracy. This result arose because Cyclomatic does not focus on weight or difficulties of code controls. There is a difference between iteration controls, branch controls, and recursion functions.

However, Shao and Wangs metric is more accurate than Cyclomatic and Halsted metrics. Shao and Wangs got about 85% accuracy. This is because Shao and Wangs use a weighted technique to calculate code complexity. This technique can cover any disadvantages in Cyclomatic and Halsted metrics, but Shao and Wangs do not give any weights for logical, input/output, and arithmetic operations.

Our new metric gives the best accuracy with 94%, and that means an increase in accuracy of 9%. This is due to our new metric having more attributes to cover the shortcomings of previous methods, and its ability to detect complex code inside program. We mix the attributes of two main metrics: Cyclomatic, and Shao and Wangs. We use the flow-chart technique more specifically, Microsoft technique to detect Cyclomatic [11] with Shao and Wangs weights as shown in Table VI. Thus, our new technique gains the advantages of both metrics. In addition, we use Halsted volume by adding a number of operations as attribute to avoid the disadvantage of Cyclomatic and Shao and Wangs metrics.

Finally, the new metric can detect the external functions and libraries. Also, we give different weights for variables declaration, since global variables are more complex than local ones. Function calling for local functions, as well as the number of function arguments, can give the developer an approximate number for function complexity. Altogether, the six attributes give this new metric higher accuracy than the three metrics separately.

TABLE X. SPEARMAN'S CORRELATION COEFFICIENT FOR ALL COMPLEXITY METRICS

Program name	Experts	Cyclomatic	Halsted volume	Shao Wang's	New metric
Factorial	15	14	14	13.5	15
Date format	14	11.5	13	13.5	14
Bitmap	13	15	15	15	13
CSV to HTML translation	12	4.5	8	6	10
breadth first search algorithm	11	11.5	9	10.5	12
depth first search algorithm	10	8.5	7	8	6.5
heap sort algorithm	8.5	10	6	10.5	8.5
md5 hashing algorithm	8.5	13	12	12	11
huffman codes	7	8.5	11	9	8.5
Dijkstra's algorithm	6	6	10	7	5
priority algorithm	5	7	2	2	6.5
Decision tree	3.5	2	4	3	2
A-Star	3.5	4.5	3	5	4
Self-organizing map (SOM)	2	3	5	4	3
Fuzzy Logic	1	1	1	1	1
Spearman's correlation coefficient	1	0.816071	0.814286	0.850893	0.941964

Fig. 2 shows the results of Spearman's correlation coefficient of the four metrics.

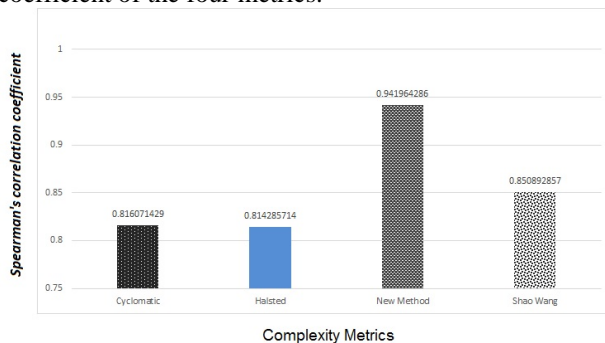


Figure 2. Spearman's correlation coefficient to the four metrics

V. CONCLUSION AND FUTURE WORK

Software complexity is one of most important factors in software engineering. Complexity of a program can be affected by many other factors in software engineering and software lifetime. If any program has a high complexity in its code, then developers prefer to stop supporting, or they risk losing a lot more time and effort for maintaining, testing, re-engineering etc. To reduce risks related to software complexity, most software engineers measure the complexity of program as a feasibility study before start any project.

In this paper, we proposed a new metric to calculate code complexity by utilizing and combining a number of older metrics: Cyclomatic metric, Halsted Volume, and Shao and Wang's Cognitive Functional Size. This new metric has six main attributes: flow chart with different weights for each control, similar to Shao and Wang's metric; number of operations, similar to Halsted Volume, but with the addition of I/O operations; number of function arguments; number of external functions and libraries; number of functions calling for local functions only; and finally, number of variables declaration with different weights for local and global variables. We programed a tool to calculate all four metrics results automatically. The dataset of this work was 15 programs were written in Java programming language.

We collected them from several different websites, and gave them to be ranked by 7 experts in Java programming language. The results for this paper show that the new metric has the highest accuracy among the 4 metrics tested, with an accuracy of 94%. Shao and Wang had better results than

Cyclomatic and Halsted volume, with an accuracy of 85%. While Cyclomatic and Halsted volume got 82% and 81% respectively. For future work, we may wish to consider adding new attributes for OO programming complexity.

REFERENCES

- [1] D. I. D. Silva, N. Koadagoda, and H. Perera, "Applicability of three complexity metrics," in *Proc. International Conference on Advances in ICT for Emerging Regions*, 2012.
- [2] J. Shao and Y. Wang, "A new measure of software complexity based on cognitive weights," *Canadian Journal of Electrical and Computer Engineering*, no. 0840-8688, p. 6, 2003.
- [3] U. Tiwar and S. Kumar, "Cyclomatic complexity metric for component based software," *ACM*, vol. 39, no. 0004-5411, p. 6, 2014.
- [4] G. Jay, J. E. Hale, R. K. Smith, D. Hale, N. A. Kraf, and C. Ward, "Cyclomatic complexity and lines of code: Empirical evidence of a stable linear relationship," *J. Software Engineering & Applications*, p. 7, 2009.
- [5] T. J. McCabe, "A complexity measure," *IEEE Computer Society*, vol. SE-2, no. 4, p. 12, 1976.
- [6] M. Halstead, "Elements of software science," *Elsevier North-Holland*, 1997.
- [7] M. Zhang and N. Baddoo, "Performance comparison of software complexity metrics in an open source project," in *Proc. 14th European Conference Software Process Improvement*, Potsdam, Germany, 2007.
- [8] D. D. Silva, N. Weerawarna, K. Kuruppu, N. Ellepola, and N. Kodagoda, "Applicability of three cognitive complexity metrics," in *Proc. 2013 8th International Conference on Computer Science&Education*, Colombo, 2013.

- [9] Rosetta code. (2014). [Online]. Available: [http://rosettacode.org/wiki/Rosetta Code](http://rosettacode.org/wiki/Rosetta_Code)
- [10] Github.com. (2014). [Online]. Available: <https://github.com>
- [11] Z. Naboulsi. (2014). MSDN. Microsoft. [Online]. Available: [http://blogs.msdn.com/b/zainnab/archive/2011/05/17/code-metrics-cyclomaticcomplexity.aspx?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+zainnab+\(Visual+Studio+Tips+and+Tricks\)](http://blogs.msdn.com/b/zainnab/archive/2011/05/17/code-metrics-cyclomaticcomplexity.aspx?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+zainnab+(Visual+Studio+Tips+and+Tricks))
- [12] R. Pressman, "Metrics for source code," in *Software Engineering Seven Edition*, McGraw-Hill, 2010, pp. 638-639-640.
- [13] Source forge. Sourceforge. (2014). [Online]. Available: <http://sourceforge.net>
- [14] Code project. (2014). [Online]. Available: <http://www.codeproject.com>
- [15] Compiler. (2014). [Online]. Available: <https://compilr.com/>



Mohammed Shehab is a Research Associate at Computer Science Department at Jordan University of Science and Technology. His M.Sc. degree has been received from at Jordan University of Science and Technology in Computer Science and his B.Sc. degree has been received from Muta'h University also in Computer Science. His main research interests include Data Mining, Software engineering, Image processing and Machine learning.



Yahya M. Tashtoush is Associate Professor at the College of Computer and Information Technology, Jordan University of Science and Technology, Irbid, Jordan. He received his B.Sc. and M.Sc. degrees in Electrical Engineering from Jordan University of Science and Technology, Irbid, Jordan in 1995 and 1999. He received his Ph.D. degree in Computer Engineering from the University of Alabama in Huntsville and the University of Alabama at Birmingham, AL, USA in 2006. His current research interests are Software Engineering, Artificial Intelligence, Robotics, and Wireless Sensor Networks.



Yaser Jararweh (Jordan University of Science and Technology, Jordan) received his Ph.D. in Computer Engineering from University of Arizona in 2010. He is currently an assistant professor of computer sciences at Jordan University of Science and Technology, Jordan. He has co-authored about fifty technical papers in established journals and conferences in fields related to cloud computing, HPC, SDN and Big Data. He was one of the TPC Co-Chair, IEEE Globecom 2013 International Workshop on Cloud Computing Systems, and Networks, and Applications (CCSNA). He is a steering committee member for CCSNA 2014 and CCSNA 2015 with ICC. He is the General Co-Chair in IEEE International Workshop on Software Defined Systems SDS - 2014 and SDS 2015. He is also chairing many IEEE events such as ICICS, SNAMS, BDSN, IoTSMS and many others. Dr. Jararweh served as a guest editor for many special issues in different established journals. Also, he is the steering committee chair of the IBM Cloud Academy Conference.



Mohammed Al-andoli is a master student at Computer Science Department at Jordan University of Science and Technology. His B.Sc. degree has been received from Muta'h University in Computer Information Systems. His main research interests include natural language processing, data mining and machine learning.



Wegdan Abdulqader Hussien is a master student at Computer Science Department at Jordan University of Science and Technology. His B.Sc. degree has been received from Aden University in Computer Science and Engineering. His main research interests include Wireless Sensor Networks, Computer Security and Data Mining.