Impact of Refactoring on OO Metrics: A Study on the Extract Class, Extract Superclass, Encapsulate Field and Pull up Method

Iyad Alazzam, Belal Abuata, and Ghada Mhediat

Abstract—Refactoring is the key to improve software maintainability, reduce complexity and get clear code with the ability to understand and modify it in efficient way. In this paper we present four refactoring techniques which are: Extract Class, Extract Superclass, Encapsulate Field and Pull up Method to discover their effect on multi Object Oriented metrics and factors. The main objective of this study is to help the developers and maintenance engineers in choosing the best possible refactoring technique based on determined goals. The results demonstrate that the types of refactoring techniques have distinctive impact on the metrics values. The results show that all contemplated refactoring techniques increase the Cyclomatic number value (VG). However, the Encapsulate Field has no impact on: Coupling between Objects, Depth of Inheritance, Number of children and Outward Coupling metrics.

Index Terms—Refactoring techniques, encapsulate field, pull up method.

I. INTRODUCTION

With the enormous measure of improvement, particularly in software, the code becomes more multifaceted, hard to comprehend and hard to maintain. Henceforth, developers dependably look to make software with a great deal of highlights that permit utilizing it in a justifiable and clear way. One of the methods that can influence the software is refactoring. Refactoring is considered as a controllable process that can change the software structure without changing its conduct. The primary thought is to reallocate code components, for example, classes, techniques and others crosswise over class hierarchy to encourage alteration and expansion [1].

Software development tends to use refactoring in order to improve software quality which causes improving understandability, maintainability, flexibility and reusability. In any case, generally, there are some risks associated with using refactoring for example, it might create bugs that are hard to tackle or perhaps firm design will be excessively expensive, making it impossible to modify. Nonetheless,

refactoring is as yet extraordinary compared to other techniques to deal with numerous issues confronting the programmers.

One of the targets of developers testing code is the growth of software quality which relies upon testing suites which thus used to detect software faults. The most well-known methodology used is code coverage that empowers to test the code quality since it dependent on describe the statement coverage, branch coverage and path coverage of the software testing suites. Likewise there are numerous studies supported that there is a relationship between code coverage and fault detection [2].

Refactoring includes many techniques that are implemented on the code. The refactoring techniques we investigated are as follow:

- Extract class: if the class is very complicated then this refactoring technique can help to create new class and transfer some of the relevant methods from the old class to the new class. Hence the code will become more clear and understandable.
- Extract Superclass: helps to decrease complexity and code become more flexible to solve problems and future developments by building up a shared superclass and moving all similar methods from the two classes to it.
- Pull up method: move the identical methods or variables from the subclass into relevant superclass which disposes of duplicated code.
- Encapsulated Field: hiding object data and disallow other object to access and modify the data of your object without any check or access privilege. This can be done by encapsulated field which helps to convert public data members with getter and setter methods.
- Change method signature: complete change in method by changing its name, type, add or remove parameters and change the parameters order. This will automatically change all references to the identical method.

In this paper we evaluated the effect of refactoring methods (Extract Class, Extract Superclass, Encapsulate Field and Pullup Method) on Polymorphism Factor, Method inheritance factor, Number of children, Coupling between objects, Depth of inheritance tree, Directly called components, Cyclomatic number, Number of nested levels, Response for a class, Number of children and Outward coupling

This paper is organized as follows. In the first section we briefly discuss related work. In the second section we present the methodology used. We then present our approach to refactoring in Section III and also the tests and results

Manuscript received May 20, 2019; revised October 17, 2019.

Iyad Alazzam and Ghada Mhediat are with the Dept. of CIS at Yarmouk University, Jordan (e-mail: eyadh@yu.edu.jo, 2017930020@ses.yu.edu.jo).

Belal Abuata is with the Dept. of MIS at Yarmouk University, Jordan (e-mail: belalabuata@yu.edu.jo).

conducted, and in Section IV a conclusion is explained.

II. RELATED WORK

This section provides and summarizes the researches that have been carried out in the area of refactoring generally and the impact of refactoring on quality metrics specifically. In [2] they evaluated the relationship between test suite sizes, coverage and detect fault effectiveness applied on five large java projects. The main purpose was to discover if test suite coverage provides better fault detection when the test suite size is observed. The results showed that the high level of coverage don't mean the test suite is more effective. Another research done in [3] uses different open source projects such as Apache, MySQL Connector/J and JBoss Hibernate to measure the effect of refactoring quality on software. The shared results of those projects lead to two suggestions; the first, it is not necessary to improve software quality by refactoring and the second, developers have no effective way to use refactoring to get the best quality of software. Simon *et al.* [4] proposed a study that was carried out to assess the impact of refactoring on software quality. They used different refactoring techniques; Move method, Move attribute, Extract class and Inline method. They also used CASE tool to implement the methods, the result was positive and shows that refactoring can improves and increases software quality. In [5], [6] the authors compared between internal and external quality attributes to classify refactoring methods in order to help developers choose suitable refactoring techniques.

External measure has been selected (analyzability, changeability, time behavior and resource utilization) to discover the impact of refactoring techniques on software quality. Ten measures were used and refactoring techniques produced disaster results on each measure [7]. Chi & Lyu [8] discovered the relationship between code coverage and the ability of detect faults which is not steady according to different testing environments such as (functional testing and random testing). They used several types of code coverage; block coverage, decision coverage, C-use and P-use. They found that the ability of code coverage in fault detection is higher in functional testing rather than random testing. Bois, and Mens [9] tried to find metrics that help developers to provide best quality of software by using refactoring. But at the beginning, they studied the impact of refactoring on internal software quality using different refactoring techniques which are: Extract Method, Encapsulated field and Pull up method implemented on "Inellij IDEA" and "Eclipse". These methods were characterized on AST representation and used the results to analyze the impact of refactoring by object oriented program quality metrics. The results were positive and show feasibility of implementing these techniques. Chaparro *et al.* [10] used RIPE (Refactoring Impact PreEdicat) method to analyze the impact of refactoring on software by using different metrics to support the operation such as (Coupling Between Object) and others. This study was based on 12 refactoring methods and 11 metrics, the results were in the positive side since RIPE predicted 38% of 8,103 metrics values. This method is recommended and can supports developers when refactoring causes radical change in the code. Kataoka *et al.* [11] applied

quantitative technique to evaluate the maintainability on software after implementing refactoring. They described the impact of refactoring using coupling metrics and their experiment has shown that their method is very useful.

Fontana and Spinelli [12] studied many refactoring techniques in order to select the proper technique in detecting code smell. The process starts by identifying the metrics used to evaluate the system which are coupling, cohesion, number of methods, response for a class, tight class cohesion and weighted method per class, this will led to detect the code smell. Then after applying refactoring on the code, they recomputed the metrics on software then analyzed the result to determine the best refactoring techniques that affect the quality of analyzed system. The result shows that it is difficult to decide if refactoring removes code smells or it produces other unknown smells. Kumari and Saha [13] discovered the effect of using refactoring methods on specific software quality attribute. They used 14 refactoring techniques, and used IntelliJ Idea which is an IDE tool to determine the effect of refactoring. They also used nine software attributes which were computed after and before refactoring. They concluded that, refactoring could be a positive choice and sometime negative one because the result showed that refactoring is not definite to be a way to improve software quality and it depends on what the developer chooses as a refactoring method. Another research done in [14] studied the refactoring impact on the source code. The experiment started with using three refactoring methods (extract class, extract method and replace temp with query). The main idea here is to apply these refactoring techniques on six experiments with different order and compute the code before and after refactoring using several metrics and they used the metrics tool "Source Monitor". The results showed that applying refactoring methods in different order provide different effects on code. The main goal of the study in [15] is to assure or not assure if refactoring develop software quality and it was applied on the internal and external measures. The experiment applied on c# open source project and visual studio was the selected tool. They concluded that the result of external measures did not improve the quality of the code after refactoring. As for the internal measure; none of them make any improvement except maintainability which has positive effect in code quality. This positive effect was higher on the projects on which refactoring was applied than non-refactoring project.

Chávez *et al.* [16] assessed the impact of refactoring on the software quality. They used 23 open source projects to determine the effect of refactoring on internal software quality using 11 refactoring methods. The study was divided into two parts. The first one is to determine whether refactoring techniques can be applied on code with critical internal quality attributes and the second to figure the effect of refactoring on internal quality attributes. The results showed that 65% of internal quality attributes were positively affected and the rest 35% were unaffected. The study in [17] depends on discovering the impact of refactoring on maintainability applied on five open source software. The results were compared between the code after and before refactoring. They used five refactoring methods and the results were positive and showed that some refactoring methods can improve the software quality such as

consolidate conditional Expression and Extract Method. However, some other methods could be a dangerous tool that may destroy the code and decrease maintainability such as Extract class and Encapsulate field.

The study in [18] involves enhancing testability through obtaining better results in cohesion measure using automated refactoring. The experiment was carried out on java software and applied test cases to show the extent of complexity in typing it after and before refactoring. The results showed that refactoring can be away to improve testability, but it needs more of work and testing. The experiment in [19] is based on reducing faults and reaching high level of software quality through applying refactoring methods on several open source software. They find out that refactoring has a positively side on decreasing the number of faults in software system, and the percentage of errors that have been reduced are as follow: 20%, 16%, 4%, 14% and 5%, it's a promising ratio that can't be ignored.

I. Alazzam and M.S. Al-Amri [22] conducted an experiment employing refactoring technique "inline method" to study the impact of this technique on coverage of the test suite through harnessing impact analysis of test coverage. They did apply inline method refactoring on open source java project and did some experiments. Their results show the inline method refactoring has a main impact on decreasing complexity. Conversely, their results disclose that this refactoring technique "inline method" has a negative impact on coverage from testing perspective.

The research in [20] main objective was: Can refactoring improve maintainability? His study focused on applying refactoring methods on six large projects software. The results were as follows: the output of single refactoring on the comprehensive maintainability cannot be ascertained because refactoring may have negative effect on maintainability. However, in this experiment refactoring effect was useful on software maintainability because it was done by a whole period. The study in [21] is based on discovering the effect of refactoring on code quality and on legacy system particularly. The results showed that refactoring improved development in software maintainability in several quality metrics such as: cyclomatic complexity, line of code and class coupling, and that refers to the simplicity to affect the code and develop it after refactoring.

Stroggylos [23] did research on the source code version analysis of control system logs of selected common open source software system. Their results indicated an effect of refactoring on the software metrics to evaluate the impact of refactoring on quality. They found the increase in metric valued of LCOM, Ca and RFC which degrades the software quality. They concluded that refactoring does not always improve the software quality. Shrivastava [24] presented a case study to improve the quality of software by refactoring. They took open source and with the Eclipse refactoring tool produced three version of refactored code. The results found that the size and complexity of a software decreases with refactoring and hence maintainability increases. The study to find effect of refactoring on the software quality attributes has a wide scope. Fowler [25] has given 70 types of refactoring methods and each refactoring method can be linked to the various software quality attributes. So, our focus

is to find the effect of fourteen randomly chosen refactoring methods on the various object oriented metrics and hence on the external software quality attributes.

III. RESEARCH METHODOLOGY AND RESULTS

In order to evaluate and study the impact of refactoring techniques on object oriented metrics and factors, we carried out multi processes. In our experiment we applied four refactoring techniques on java project open source using eclipse platform. We have made four copies of the same project in order to apply each refactoring method individually on the whole project to verify the validity of the results. First, we have calculated the object oriented metrics using different selected tools according to their applicability. Table I shows the calculated metrics used along with the selected tool.

TABLE I: METRICS AND TOOL SUPPORT

Design Metric	Source (Tool)	Metric Name
Cyclomatic number (CComplexity)	Metrics 1.3.8	VG
Number of statements (Size)	Metrics 1.3.8	TLOC
Weighted methods per class (Complexity)	Metrics 1.3.8	WMC
Method inheritance factor (MethodInheri)	Understand	MFA
Average size of statements (avgSize)	Metrics 1.3.8	TLOC
Number of nested levels (Nesting)	Metrics 1.3.8	NBD
Coupling between objects (Coupling)	Understand	CBO
Lack of cohesion (LCohesion)	Metrics 1.3.8	LCOM
Depth of inheritance tree (Hierarchies)	Understand	DIT
Number of children (Subclasses)	Metrics 1.3.8	NSC
Directly called components (Communication)	Understand	RFC
Outward coupling (EffCoupling)	Metrics 1.3.8	CE

After that we calculated the following metrics for the java source code before applying any refactoring technique as shown in Table II.

Then we apply the following refactoring techniques:

A. Extract Class

We start with extract class where it is by nature small and can perform small or huge tasks according to its implementation. It can contain many methods related or unrelated to each other. This leads to difficult to read, modify, understand and maintain. In this case we can extract some of the relevant methods to a new created class as the example shown in Fig. 1.

TABLE II: RESULT OF SOFTWARE QUALITY METRICS FOR JAVA PROJECT BEFORE REFACTORING

Design Metric	Result
VG	4.195
Σ TLOC	8563
WMC	28.744
MFA	0.2542
TLOC	4281.5
NBD	1.282
CBO	3.4545
LCOM	0.154
DIT	1.3295
NSC	0.314
RFC	6.4318
CE	8.875

We applied Extract class on 14 different classes from three

different packages. Extract class starts by choosing the appropriate class that contains several methods, after that we can choose which method will move to the new class. The result of each software quality attribute for the Extract class is shown in Table III.

TABLE III: RESULT OF SOFTWARE QUALITY METRICS FOR JAVA PROJECT APPLIED BY EXTRACT CLASS

Design Metric	Before	After
VG	4.195	4.034
Σ TLOC	8563	9078
WMC	28.744	25.173
MFA	0.2542	0.190
TLOC	4281.5	4539
NBD	1.282	1.219
CBO	3.4545	3.09433
LCOM	0.154	0.215
DIT	1.3295	1.2735
NSC	0.314	0.26
RFC	6.4318	6.71698
CE	8.875	9.75

Table III shows the results after applying extract class refactoring technique. The results prove that cyclomatic number, number of statement, average size of statement, coupling between object, lack of cohesion, number of children, direct called components and outward coupling (EFFCoupling) metrics increased while Weighted Method per class, Method Inheritance factor, Number of Nested Levels and Depth of Inheritance metrics decreased.

B. Extract Superclass

Another way to reduce complexity here, we have two classes that contain similar fields and methods. This technique involves about creating shared superclass and transfers all similar methods and fields to the shared class, the original class will be inherited from the superclass as the shown in the example of Fig. 2.

Fig. 1. Brief example on extract class.

Fig. 2. Brief example on extract superclass.

After applying extract superclass we get the results shown in Table IV.

TABLE IV: RESULTS OF SOFTWARE QUALITY METRICS FOR JAVA PROJECT APPLIED BY EXTRACT SUPERCLASS

Design Metric	Before	After
VG	4.195	4.722
Σ TLOC	8563	8724
WMC	28.744	24.252
MFA	0.2542	0.5627
TLOC	4281.5	4362
NBD	1.282	1.268
CBO	3.4545	3.3714
LCOM	0.154	0.133
DIT	1.3295	1.5809
NSC	0.314	0.427
RFC	6.4318	7.0857
CE	8.875	10.25

Table IV shows the results after applying extract Superclass refactoring technique. The results prove that cyclomatic number, number of statement, method Inheritance factor, average size of statement, and lack of cohesion, number of children, direct called components, outward coupling (EFFCoupling) and depth of inheritance metrics increased while coupling between object, weighted method per class, number of nested levels metrics decreased.

C. Encapsulate Field

Encapsulate field allows the user to protect the usable fields by making them private and create access methods (set and get) as described in the example shown in Fig. 3. After applying Encapsulate Field we get the results shown in Table V.

Fig. 3. Brief example on encapsulate field.

TABLE V: RESULT OF SOFTWARE QUALITY METRICS FOR JAVA PROJECT APPLIED BY ENCAPSULATE FIELD

Design Metric	Before	After
VG	4.195	4.442
Σ TLOC	8563	8770
WMC	28.744	29.547
MFA	0.2542	0.2692
TLOC	4281.5	4385
NBD	1.282	1.248
CBO	3.4545	3.4545
LCOM	0.154	0.181
DIT	1.3295	1.3295
NSC	0.314	0.314
RFC	6.4318	7.2613
CE	8.875	8.875

Table V shows the results after applying Encapsulate Field refactoring technique. The results prove that this type of refactoring do not have impact on the coupling between

object, depth of inheritance, number of children and outward coupling (EFFCoupling) metrics. However the values of cyclomatic number, number of statement, method inheritance factor, average size of statement, lack of cohesion, direct called components and weighted method per class metrics increased while number of nested levels metric decreased.

D. Pull up Method

Several subclasses contain common methods with identical behavior and we can move these methods into single methods in superclass. After applying change method signature technique we get the results shown in Table VI.

Table VI shows the results after applying Pull Up Method refactoring technique. The results prove that the Pull Up Method has no effect on two metrics number of children and Depth of Inheritance. On the other hand, cyclomatic number, Method Inheritance factor, coupling between object, direct called components and outward coupling (EFFCoupling) metrics increased while Number of statement, average size of statement, lack of cohesion, Weighted Method per class and Number of Nested Levels metrics decreased.

TABLE VI: RESULTS OF SOFTWARE QUALITY METRICS FOR JAVA PROJECT APPLIED BY PULL UP METHOD

Design Metric	Before	After
VG	4.195	4.374
Σ TLOC	8563	8127
WMC	28.744	26.14
MFA	0.2542	0.30744
TLOC	4281.5	4063.5
NBD	1.282	1.257
CBO	3.4545	3.5681
LCOM	0.154	0.148
DIT	1.3295	1.3295
NSC	0.314	0.314
RFC	6.4318	6.8863
CE	8.875	9

IV. CONCLUSION

In this paper we introduced different refactoring methods (Extract class, Extract Superclass, Pull up method and Encapsulate Field) in order to determine their impact on multi Object Oriented metrics and factors. The results reveal that the refactoring techniques have different impact on Object Oriented Metrics. Metrics values increased and decreased based on the refactoring technique. This will help developers and maintenance engineers to select the proper type of refactoring techniques based on the specific rules. In addition, the results showed that all studied refactoring techniques increased the value of cyclomatic number (VG). However, the Encapsulates Field and Pull Up methods have no effects on some metrics, specifically; the Encapsulate Field has no impact on: Coupling between Objects, Depth of Inheritance, Number of children and Outward Coupling metrics.

CONFLICT OF INTEREST

The authors declare no conflict of interest

AUTHOR CONTRIBUTIONS

Iyad Alazzam conceived of the presented idea. Belal Abuata wrote the paper with input from all authors. Ghada

Mhediat designed and performed the experiments, performed the calculations and analyzed the data with Iyad Alazzam all authors had approved the final version.

ACKNOWLEDGMENT

The authors also would like to acknowledge the financial support of the Deanship of Scientific Research, Yarmouk University.

REFERENCES

- [1] T. Mens and T. Tourwe, "A survey of software refactoring," *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126-139, 2004.
- [2] L. Inozemtseva and R. Homles, "Coverage is not strongly correlated with test suite effectiveness," in *Proc. the International Conference on Software Engineering (ICSE)*, 2014, pp. 435-445.
- [3] D. Spinellis and K. Stroggylos, "Refactoring – does it improve software quality?" in *Proc. Fifth International Workshop on Software Quality (WoSQ'07)*, 2007, pp. 10-16.
- [4] F. Simon, F. Steinbuekner, and C. Lewerentz, "Metrics based refactoring," in *Proc. the Fifth European Conference on Software Maintenance and Reengineering (CSMR '01)*, 2001, pp. 30-38.
- [5] M. Alshyab and K. O. Elish, "A classification of refactoring methods based on software quality attributes," *Arabian Journal for Science and Engineering*, vol. 36, no. 7, pp. 1253-1267, 2011.
- [6] S. Bashir, S. P. Lee, and C. C. Yung, "A methodology for impact evaluation of refactoring on external quality attributes of a software design," in *Proc. International Conference on Frontiers of Information Technology*, 2017, pp. 183-188.
- [7] W. M. J. I. Wijayanayake and S. H. Kannangara, "Impact of refactoring on external code quality improvement: An empirical evaluation," in *Proc. 2013 International Conference on Advances in ICT for Emerging Regions (ICTer)*, 2013, pp. 60-67.
- [8] X. Chi and M. R. Lyu, "The effect of code coverage on fault detection under different testing profiles," in *Proc. the 1st International Workshop on Advances in Model-Based Testing*, 2005, pp. 1-7.
- [9] B. D. Bois and T. Mens, "Describing the impact of refactoring on internal program quality," in *Proc. the International Workshop on Evolution of Large-scale Industrial Software Applications*, 2003, pp. 37-48.
- [10] O. Chaparro, G. Bavota, A. Marcus, and M. D. Penta, "On the impact of refactoring operations on code quality metrics," in *Proc. 30th IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 456-460.
- [11] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, "A quantitative evaluation of maintainability enhancement by refactoring," in *Proc. the International Conference on Software Maintenance*, 2002, pp. 576-585.
- [12] F. A. Fontana and S. Spinelli, "Impact of refactoring on quality code evaluation," in *Proc. the 4th Workshop on Refactoring Tools*, 2011, pp. 37-40.
- [13] N. Kumari and A. Saha, "Effect of refactoring on software quality," in *Proc. Fourth International Conference on Advances in Computing and Information Technology*, 2014, pp. 37-46.
- [14] Y. Khrishe and M. Alshayeb, "An empirical study on the effect of the order of applying software refactoring," in *Proc. 7th International Conference on Computer Science and Information Technology (CSIT)*, 2016, pp. 1-4.
- [15] S. H. Kannangara and W. M. J. I. Wijayanak, "An empirical evaluation of impact of refactoring on internal and external measures of code quality," *International Journal of Software Engineering & Applications*, vol. 6, no. 1, pp. 51-67, 2015.
- [16] A. Chávez, I. Ferreira, E. Fernandes, D. Cedrim, and A. Garcia, "How does refactoring affect internal quality attributes? A multi-project study," in *Proc. the 31st Brazilian Symposium on Software Engineering*, 2017, pp. 74-83.
- [17] R. Malhotra and A. Chug, "An empirical study to assess the effects of refactoring on software maintainability," in *Proc. International Conference on Advances in Computing, Communications and Informatics*, 2016, pp. 110-117.
- [18] M. Cinnéide, D. Boyle, and I. H. Moghadam, "Automated refactoring for testability," in *Proc. Fourth International Conference on Software Testing*, 2011, pp. 437-443.
- [19] A. Chug and M. Gupta, "A quality enhancement through defect reduction using refactoring operation," in *Proc. International*

Conference on Advances in Computing, Communications and Informatics (ICACCI), 2017, pp. 1869- 1876.

- [20] G. Szőke, G. Antal, C. Nagy, R. Ferenc, and T. Gyimóthy, "Empirical study on refactoring large-scale industrial systems and its effects on maintainability," *Journal of Systems and Software*, vol. 129, pp. 107-126, 2016.
- [21] A. Jonsson, "The impact of refactoring legacy systems on code quality metrics," Master Thesis, Uppsala University, 2017.
- [22] I. Alazzam and M.S. Al-Amri, "Test coverage and impact analysis for detecting refactoring faults: a study on the inline method" *Int. J. Business Information Systems*, vol. 32, no. 2, 2019.
- [23] S. Konstantinos and D. Spinellis, 2007, "Refactoring--does it improve software quality?" presented at the 5th International Workshop on Software Quality, IEEE Computer Society.
- [24] S. V. Shrivastava and V. Shrivastava, 2008, "Impact of metrics based refactoring on the software quality: A case study," presented at the TENCON 2008 IEEE Region 10 Conference, IEEE.
- [25] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*, Addison Wesley, 1999.

Copyright © 2020 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

Iyad M Alazzam, is an associate professor in the Department of Computer Information Systems at Yarmouk University in Jordan. Previously he worked as a full-time lecturer in the Department of Management Information Systems at Yarmouk university during 2008 to 2009, Irbid, Jordan. Prior to that he worked as a research and teaching assistant at the Faculty of Prince Al-Hussein bin Abdullah II of Information Technology, Zarqa, Jordan during 2006 to 2008. He has received his Ph.D degree in software engineering from NDSU, USA in 2012. His master was got from LMU (UK) (2006) in electronic Commerce and his

B.Sc in computer science and information systems from Jordan University of Science and Technology in Jordan 2004. His research interests lay in software engineering, software testing and information retrieval.

Belal M. AbuAta obtained his BSc. in computer science from Yarmouk University, Jordan and MSc in computer science from the National University of Malaysia in. Also, he obtained his PhD degree from National University of Malaysia in the area of multilingual information retrieval. He worked as a lecturer, assistant Prof. and head of the Dept. in different public and private universities in Malaysia between 1997 and 2001. After that he worked an assistant Prof. at Philadelphia University, Jordan for one year. Then for seven years he worked as an assistant Prof. in Computer Science Dept., University of Bahrain. He is presently working as an associate professor in the Department of Information Technology at the faculty of Information Technology and Computer Sciences at Yarmouk University, Jordan. He is having a total of more than 20 years' teaching experience and research in various IT colleges in Malaysia, Bahrain and Jordan. His research interest includes information retrieval, web searching and evaluation, multimedia systems.

Ghada Mhediat received her bachelor degree in computer science from Al- Balqa' Applied University, Jordan in 2015. Now she is doing MSc in Computer Information Systems, Faculty of Information Technology and Computer Sciences at Yarmouk University, Irbid, Jordan. Her research interests include software engineering and web information retrieval.