An Empirical Analysis of the Relationship between Cohesion and Testing Effort

Waleed Albattah[1*] and Austin Melton[2]

[1] Department of Information Technology, Qassim University, Saudi Arabia.
[2] Department of Computer Science, Kent State University, Kent Ohio 44242.

* Corresponding author. Email: w.albattah@qu.edu.sa

Manuscript submitted February 1, 2017; accepted May 1, 2017.
doi: 10.17706/jsw.12.8.671-681

Abstract: Predicting software testability can reduce costs and efforts. Cohesion, as one of software quality metrics, found to be good indicator for software testability. Although there is a good interest in software testability on the class level in the literature, software testability on the package level has not received the same interest. The paper investigates the relationship between a newly proposed package cohesion metric and software testing effort. The empirical analysis used data collected from five Java open source software systems for which JUnit test classes are available. The results show that as good the package cohesion is, as the less testing effort is needed. The stability of the correlations allows us to draw optimistic conclusions about its use as an indicator.

Key words: Correlation, cohesion, package, software engineering, software measurements, software metrics, software testing, testability, testing effort.

1. Introduction

Software testability is known to be one of the software maintainability characteristics. During the software development process, the detection of faults and errors is always one of the main goals for the software development team. The early detection of errors and faults can reduce the maintenance effort and costs. Software testing is the process that offers this advantage to deliver a high quality software system. Software testability aims to facilitate the process of software testing. ISO [1] defines software testability as "attributes of software that bear on the effort needed to validate the software product." Another definition by ISO [2] is the degree of effectiveness and efficiency with which test criteria can be established for a system, product, or component, and tests can be performed to determine whether those criteria have been met. IEEE [1] defines it as the degree of the software that facilitates the establishment of test criteria and the performance of tests to determine whether criteria have been met. Software testability has a relation to testing effort reduction and software quality [3]. As stated by Gao *et al.* [3], the late detection of a lack of testability may be difficult and expensive to repair, and it can badly affect the testing and maintenance effort [4].

It has been argued that software testability, as one of the maintainability characteristics, should be considered as a key factor in software quality. Software quality measurements depend on software testability measures. Therefore, predicting software testability using software measurements is expected to give a chance to improve software quality.

Measuring software attributes that have an impact on testability after coding for the purpose of testability evaluation is later and more costly. However, predicting testability earlier, during the software design phase, may greatly reduce the cost and the time [5]. Software metrics can predict software quality characteristics [6]-[8]. Several metrics were proposed to predict quality attributes related to testability such as maintainability. Many studies have investigated the role of cohesion, which is one of the most important software quality internal attributes, in predicting software maintainability, and how cohesion can impact software maintainability in different abstraction levels. A cohesion metric can be a good predictor for software maintainability and software testability. Such predictions include, but are not limited to, fault prone-ness and defect density. In this paper, we empirically investigate the relationship between package cohesion and package testability of the software system. Our hypothesis is that a package with low cohesion is difficult to test.

This study has two folds. First, it investigates the relationship between package cohesion and software testing effort, which is presented in this paper. Second, it predicts software testing effort using package cohesion, which will be published in the near future. The first part of the study encouraged us to further exploration of how package cohesion can be used as a good testability predictor.

The rest of the paper is organized as follows: The related studies are briefly introduced in Section 2. Section 3 presents an overview of the studied package cohesion metrics. Section4 details the empirical study. Section 5 investigates and discusses the correlation between package cohesion and testing effort. Finally, Section 6 concludes the paper with future works.

2. Related Work

Finding a clear view of all the factors that can affect software testability is difficult because testability is an elusive concept [5]. Many testability approaches have been proposed to investigate the degree of software testability.

Freedman [9] proposed testability metrics based on observability and controllability. The proposed testability measures examine the input and output domains. He meant by observability the ability of specific input to affect the output. Controllability is meant to be the ease of producing specific output from specific input. Fenton *et al.* [10] considers software testability as an external attribute that can be affected by internal attributes. Voas [11] states that the test case of a component will fail if it has a fault. Using the testability definition proposed by Voas [11], Khoshgoftaar *et al.* [12] modeled the relationship between static software measures and testability. They developed two distinct models, and classified the program modules as having low or high testability. Jungmayr [13] focuses on dependencies between software components and proposed the notion of “test-critical dependencies.” This new concept is used to estimate testability of software through integration testing. The reduction metric is used to calculate the effect of individual factors to find out the required testability metric [5]. Bruntink and Deursen [14] proposed some testability metrics to assess the testability of the classes of Java systems. They defined some testability factors based on source code metrics. One limitation of this study is the late detection of errors, which makes any repair expensive. Without empirical validation, Baudry *et al.* [15] aimed to detect the weaknesses of a UML class diagram to reduce the final testing effort. They addressed two configurations in a UML class diagram that can lead to code difficulties in testing. They proposed a testability measurement for a UML class diagram as well as solutions to improve the testability of the software design. Jianping and Minyan [16] proposed a request-oriented method of software testability measurement. The proposed method can select the appropriate elements from a self-contained software testability measurement framework to measure testability of all kinds of software. Their goal was to lower the difficulty and the cost of a software testability measurement as well as to accelerate the application and development of a software testability measurement.

Badri *et al*. [1], [4] aimed to empirically explore the relationship between a lack of cohesion and the testability of classes in object-oriented systems. Using two Java software systems that have JUnit test cases, they evaluated the capability of lack of cohesion metrics to predict testability. The results support the idea that there is a significant relationship between the (lack of) cohesion of classes and testability. In another work, Badri *et al.* [17] performed an empirical investigation to study the relationship between object-oriented design metrics and the testability of classes. Using logistic regression methods, they evaluated the individual and the combined effect of metrics on the unit testing effort of classes. The results indicated that complexity, size, cohesion, and (to some extent) coupling were found to be significant predictors of the testing effort of classes. Later, Badri *et al*. [18], [19] studied the effect of control flow of the unit testing effort of classes. They classified the classes into low and high according to the required testing effort.

Singh *et al.* [20] measured the testing effort in terms of lines of code added or changed during the life cycle of a defect. They predicted testing effort using object-oriented metrics and neural networks [20]. In another work, Singh *et al*. [21] performed a case study on Eclipse to predict testability at the package level. The results showed that there is a significant correlation between source code metrics and test metrics that obtained from JUnit test classes of test packages. They found that the low value of cohesion increases testing effort and decreases testability.

Although there is a good interest in software testability on the class level as seen from the above, software testability on the package level has not received the same interest. So this study focuses on the package testability and how it is affected by package cohesion. It uses our cohesion metric on the package level [22], proposed based on the well-known package cohesion principles, both theoretically and experimentally validated. Actual testing data of software have been used to investigate the relationship between the internal quality attribute, package cohesion, and the external quality attribute, package testability, using statistical analysis tests.

3. Package Cohesion Metrics

3.1. The Proposed Metric (CH)

In our previous work [22]-[24], which is motivated by Martin's package cohesion principles [25], we proposed two different cohesion metrics to measure two different cohesion concepts or types based on Martin's package cohesion principles in [25]. The first cohesion type, Common Reuse (CR), includes the factors that help in assessing CR cohesion. Similarly, the second cohesion type, Common Closure (CC), includes the factors that help in assessing CC cohesion. After each type of cohesion is measured by itself, the two values of CR and CC may be combined to one unified value of package cohesion, while still recognizing the two types.

The CR metric measures cohesion based only on the common reuse factors of the package. The elements of a package have different degrees of reachability. Reachability of a class in a package is the number of classes in the same package that can be reached directly or indirectly. The CR metric is defined as follows:

"Let c $\in$ C, and suppose there is an incoming relation to c from a class in a different package. Then c is called an in-interface class. The cardinality of the intersection of the hub sets of all the in-interface classes in C divided by the number of classes in C is the CR of P ".

$$CR= |\cap_{\text{In-interface class hub sets}}| \;/\; |C| \tag{1}$$

where

Hubness(c) = {d $\in$ C: if there is a path c $\rightarrow$d}

C: set of classes in package *P*

c and d: classes in C

The CC metric considers the package dependencies on other packages as well as the internal dependencies between classes of the package. The classes of the package should depend on the same set of packages and, thus, they will have the same reasons for a change. The CC metric is defined as follows:

"The cardinality of the intersection of the reachable sets divided by the cardinality of the union of the sets represents the CC of P ".

$$CC = \left(\left| \cap \text{ Reachable Package sets} \right| / \left| \cup \text{ Reachable Package sets} \right| \right) \tag{2}$$

The combined cohesion CH is defined as follows:

$$CH = \frac{\sqrt{2} - D}{\sqrt{2}} \tag{3}$$

$$D = \sqrt{(1 - CR)^2 + (1 - CC)^2} \tag{4}$$

3.2. Martin's Metric (H)

Martin proposed a rational cohesion metric for the package,

$$H=(R+1)/N \tag{5}$$

where R: number of relationships between classes in the package

N: number of classes in the package

Although Martin's cohesion principles [25] are well known and well accepted, H metric doesn't conform to them. H measures the ratio of the relationships between classes of the package. This simple concept doesn't measure the common reuse or the common closure of the package, but rather, in its best situation, it may measure the classes' extent of being connected. The H metric depends on the number of relations rather than how these relations are designed. In this case, a well-designed package and a badly designed package could have the same cohesion value. In our previous work [22], further discussions are presented.

4. Descriptive Statistics

This empirical study was conducted on five open-source Java software systems to discover the role of a package cohesion measure in predicting software testability. This section provides descriptions about the studied software systems and the testing data collection. Two package cohesion metrics are included in this study, Martin's cohesion metric (H) and our proposed package cohesion metric (CH)[22], which was developed based on Martin's package cohesion principles [25].

4.1. The Software Systems

Five open-source Java software systems were involved in the empirical study. All the five systems were selected based on the following criteria to allow results generality. They had: (1) to be implemented using Java programming language, (2) to have testing cases available, (3) to have a sufficient number of versions for each system that have been tested, (4) to be organized using packages, (5) to have different sizes ranging from very large to small systems in terms of number of packages and number of classes, (6) to be from different domains, and (7) to have positive reviews and to be mature. We expect these criteria will allow the generalization of the results obtained from the study. The first system, Camel [26], is rule-based and mediation engine to configure routing and mediation rules. The second system, Tomcat [27], is an open source webserver developed to implement Javaservlet and Java Server pages (JSP). Apache Tomcat is developed by the Apache Software Foundation. It has been developed and released under Apache License version 2. The third system, Hadoop [28], is open source framework software for large-scale processing of data sets on clusters of computers. It is licensed under the Apache License 2.0. The fourth system, Synapse [29], is a lightweight and high-performance open source Enterprise Service Bus (ESB). It provides exceptional support for XML and Web Services. It also supports several content interchange formats. Apache Synapse is licensed under the Apache Software License version 2.0. The fifth system, Ant [30], is a Java library and command-line tool used to build Java applications. It can be also used to build non-Java applications such as C or C++ applications. Table I provides details about the studied software systems.

Table 1. Details of the Studied Systems

	Release	#LOC	#Classes	#Packages	#TestPackages	#TestClasses	#TLOC
Camel	2.0.0	143732	5111	264	44	459	31494
Tomcat	7.0.6	170461	1725	113	37	323	10732
Hadoop	2.2.0	522903	4445	222	205	3351	181007
Synapse	2.1.0	82032	1115	117	47	288	13177
Ant	1.92	106300	1120	67	36	442	23170

4.2. Testing Data

The source of the testing data for this study is the test classes of the studied systems. Test classes are written for the purpose of software testing. JUnit, which is an open source framework, is designed for running tests in Java programming language [31]. JUnit has gained a lot of popularity [32][33]. It helps in testing a Java class by defining how to write the corresponding classes and provides the tool to run them [34]. JUnit gives testers support to write test classes for the system classes, in a convenient way, and then run them to output a report about the successful and failed methods of the class tested [4][31]. The source class and the test class can be kept in the same or different packages [34]. The more test classes that are written, the more the testing effort.

To indicate the testing effort required for the software package, we consider a Testing Lines Of Code (TLOC) measure in which we count the number of LOC used for the purpose of testing the classes of the package. The written TLOC represents the effort spent to test a specific package. The more TLOC written, the more effort is spent and the lower the testability of the package. We consider this measure for two reasons. First, it seems to be a reasonable and a good measure for testing effort in terms of cost and time spent to test a package. Second, this measure is measurable using the freely available data.

Two computer science PhD students were dedicated to collecting the testing data. The data was collected manually from the systems' artifacts. The collected data were tested to check its validity. This process increased our confidence about the validity of the data collected.

For the purpose of listing all classes in each system and listing all packages, we have used the JHawk tool [35]. Then, each test class is assigned to its system class along with its LOC. Then, testing effort data are collected on the package level. The testing effort data were collected individually for the five studied systems. Table II summarizes testing data for the studied systems.

Table 2. Testing Data

	#LOC	Mean #LOC	#TLOC	Mean #TLOC

Camel	143732	544	31494	715
Tomcat	170461	1508	10732	290
Hadoop	522903	2355	181007	882
Synapse	82032	701	13177	280
Ant	106300	1586	23170	643

4.3. Package Cohesion Data

Package cohesion data is gathered from two package cohesion metrics. The first metric is our proposed package cohesion metric [22], CH. The second metric is Martin's cohesion metric, H. These two metrics have been used to investigate the correlation between package cohesion and testability. For the purpose of data gathering, we have developed our Java tool to measure the CH package cohesion metric. The tool has been extended to calculate Martin's package cohesion metric, H. For each system, a list of all the packages, number of classes in each package, and the associated cohesion values are generated.

5. Exploring the Relationship between Package Cohesion and Testability Using Correlation

In this section, we present the empirical study we performed to explore the relationship between package cohesion and package testability, in terms of testing effort. Correlation is one of the widely used statistical tests to investigate the relationship between internal quality attributes, e.g., cohesion, and external quality attributes, e.g., testability; and it has been used in different studies such as [1], [14], [18], [21], [36]. The correlation analysis aims to determine whether each individual package cohesion metric (CH and H) is significantly related to the testing measure, TLOC, of the package. For this purpose, we have performed Spearman's rank correlation due to the non-parametric nature of the metrics' data. We have used the well-known SPSS software for the correlation analysis of the empirical study. We have created and analyzed a correlation matrix for each software system in the study. Each correlation matrix has all the studied variables (cohesion and testing), a correlation coefficient (r), and significance level. For each pair of variables, the value of (r) can range between -1 and +1, where 1 represents a perfect positive correlation between the pair variables; -1 denotes a perfect negative correlation; and 0 indicates that there is no relationship between the variables. The magnitude of the coefficient determines the degree of the correlation. The ratings of correlation strength follow the adjectives developed by Cohen [37], Table 3.

Table 3. Ratings of Correlation

Correlation coefficient	Adjective rating
< 0.1	Trivial
0.1 to 0.3	Minor
0.3 to 0.5	Moderate
0.5 to 0.7	Large
0.7 to 0.9	Very large
0.9 to 1	Almost perfect

Besides the strength of the correlation, the relationship between any pair of variables should be assessed for its significance as well. The significance is assessed by the p-value, which corresponds to the probability that the found correlation might be due to purely random effects. The smaller the p-level, the more significant is the relationship between variables [38]. The significance of the correlation in this empirical study was tested at 95% confidence level (i.e., p-level $\leq$ 0.05). While the correlation can establish the relationship, it cannot establish a cause-effect relationship between the pair of variables [38].

5.1. Hypotheses

Our objective in this experiment is to explore empirically to what extent package cohesion is related to the package testability, in terms of testing effort. We evaluated cohesion at the package level and we counted the testing effort at the package level based on the test classes of the software system. The hypotheses of the empirical study are:

H_{01}: There is no significant correlation between package cohesion, CH, and the number of testing lines of code, TLOC.

H_{02}: There is no significant correlation between Martin's package cohesion, H, and the number of testing lines of code, TLOC.

In this experiment, rejecting the null hypothesis indicates that there is a statistically significant relationship between the pair of variables (significance level $\alpha = 0.05$).

5.2. Statistical Analysis

The number of testing lines of code (TLOC) of the software package assesses the software package testability. A smaller number of testing lines of code during the software-testing phase indicates that less effort is needed to test the software, i.e., the software is highly testable.

Table IV provides descriptive statistics (mean and standard deviation) for the variables used in analyzing software testability across the five systems, Camel, Tomcat, Hadoop, Synapse, and Ant. We included Martin's package cohesion metric (H) in the list of variables for the purpose of comparison.

Table 4. Means and Standard Deviations of the Variables Used in Testability Analysis

Variable	Camel N=264		Tomcat N=113		Hadoop N=222		Synapse N=117		Ant N=67	
	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.
H	.636	.361	.817	.524	.674	.456	.738	.423	.851	.486
CH	.530	.388	.358	.374	.281	.326	.539	.409	.557	.345
#Classes	13.700	29.637	16.17	23.062	20.02	34.879	9.53	16.836	16.72	32.973
TLOC	119.295	689.072	76.60	206.907	815.35	2527.927	112.62	242.194	345.82	1056.753

5.3. Results and Discussion

Spearman Rho correlation is the appropriate measure of a bivariate relationship when normality and linearity conditions for the Pearson's product moment correlation do not hold. For this study, the Spearman Rho correlation provides a measure of association between the proposed measure of package cohesion CH, the Martin's package cohesion metric H, package size (#Classes), and the measure of package testability, the number of testing lines of code (TLOC), within each of the five data sets. Table V provides the list of these correlations for the five sets of data.

Table V reveals that the new proposed measure of package cohesion CH, consistently has a negative moderate correlation with the measure of package testability, the number of testing lines of code (TLOC), across all the five data sets except for the Synapse system where the correlation is minor. The correlation values between package cohesion CH and the number of testing lines of code (TLOC) across the five data sets ranges from -0.199 (for the Synapse system data set) to -0.488 (for the Hadoop system data set). The statistically significant correlations confirm the expectation that a highly cohesive software package requires less effort to be tested. That is high values of the proposed measure of package cohesion are associated with lower number of testing lines of code.

Correlations between Martin’s package cohesion metric H and the package testability measure, the number of testing lines of code (TLOC), tend to be not as strong as the ones with the newly proposed measure of package cohesion CH. These correlations are consistently weak and statistically insignificant across all the five data sets, except for the correlation with the testing lines of code (TLOC) for Synapse system’s data. The value of the correlation is -.310, which is statistically significant at the .001 level. The correlation values between Martin’s package cohesion H and the number of testing lines of code (TLOC) across the five data sets is never more than 0.135 except for Synapse system data set (-0.310).

Table VI summarizes the results of the examined null hypotheses. In this experiment, rejecting the null hypothesis indicates that there is a statistically significant relationship between the pair of variables (significance level α = 0.05).

Table 5. Spearman's Rho Correlations for Testability Analysis

Data Set		H	CH	#Classes
Camel N=264	CH	.281**		
	#Classes	-.350**	-.655**	
	TLOC	-.086	-.329**	.334**
Tomcat N=113	CH	.169		
	#Classes	-.069	-.736**	
	TLOC	-.123	-.394**	.549**
Hadoop N=222	CH	.157		
	#Classes	.063	-.688**	
	TLOC	-.033	-.488**	.615**
Synapse N=117	CH	-.038		
	#Classes	-.084	-.490**	
	TLOC	-.310**	-.199*	.465**
Ant N=67	CH	.227		
	#Classes	.078	-.527**	
	TLOC	-.135	-.385**	.450**

** Correlation is significant at the .001 level
* Correlation is significant at the .05 level

Table 6. The Results of the Null Hypotheses

	Camel	Tomcat	Hadoop	Synapse	Ant
H_{01}	Rejected	Rejected	Rejected	Rejected	Rejected
H_{02}	Accepted	Accepted	Accepted	Rejected	Accepted

6. Conclusion

In this paper, we investigated empirically the relationship between Package cohesion metrics (H and CH) and the testability of software packages in terms of required testing effort. We performed an empirical analysis using data collected from five Java open source software systems for which JUnit test classes are available. To measure the testability of packages, we used testing lines of code (TLOC) to quantify the corresponding testing effort.

One strength of this study is the number of the studied systems and the relatively large sample used in the analysis. The proposed package cohesion metric (CH) is found to be correlated with package testing effort, measured by testing lines of code (TLOC). The stability of the correlations allows us to draw optimistic conclusions about its use as an indicator. However, the results of this study should be viewed as exploratory and indicative rather than conclusive.

In future, the second fold of the study will be presented. Other statistical analyses, simple and multiple regression analyses, will be conducted to predict the testing effort of the package using the proposed package cohesion metric. Regression model can evaluate the effect of the proposed package cohesion metric (CH) on the testing effort of packages.

We hope these findings will help lead to a better understanding of the relationship between package cohesion and package testability. In future, this study will be extended to include more testability measures, such as the number of test classes in packages. We also plan, in the future, to investigate the combined ability of multiple factors (i.e., cohesion, coupling, size) in predicting the testability of packages. Additionally, open source systems developed in other languages (such as C++) can be investigated, since this study has focused on Java open source systems.

References

[1] Linda, B., Mourad, B., & Fadel, T. (2010). Exploring empirically the relationship between lack of cohesion and testability in object-oriented systems. *Advances in Software Engineering.* Springer Berlin Heidelberg, 78-92.

[2] ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models

[3] Gao, J., & Shih, M. C. (2005). A component testability model for verification and measurement. *Proceedings of 29th Annual International Computer Software and Applications Conference*.

[4] Badri, L., Mourad, B., & Fadel, T. (2011). An empirical analysis of lack of cohesion metrics for predicting testability of classes. *International Journal of Software Engineering and Its Applications, 5(2)*, 69-85.

[5] Abdullah, R. S., & Khan, M. H. (August, 2013,). Testability estimation of object oriented design: a revisit. *International Journal of Advanced Research in Computer and Communication Engineering, 2(8)*.

[6] Bansiya, J., & Davis, C. G. (January, 2002). A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering, 28(1)*.

[7] Basili, V. R., Lionel, B. C., & Walcelio, M. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering,* 22.10, 751-761.

[8] Briand, L. C. *et al.* (2000). Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of Systems and Software, 51(3),* 245-273.

[9] Freedman, R. (1991). Testability of software components.. *IEEE Transactions on Software Engineering*.

[10] Fenton, N. E., & Shari, L. P. (1998). *Software Metrics: A Rigorous and Practical Approach*.

[11] Voas, J. (August, 1992). A dynamic failure-based technique. *IEEE Transactions on Software Engineering, 18(8),* 717–727.

[12] Khoshgoftaar, T. M., Szabo R. M., & Voas, J. M. (1995). Detecting program modules with low testability. *International Conference on Software Maintenance Proceedings,*.

[13] Jungmayr, S. (October, 2002). Identifying test-critical dependencies. *Proceedings of the International Conference on Software Maintenance* (pp. 404–413). IEEE Computer Society.

[14] Bruntink, M., & Deursen A. V. (2004). Predicting class testability using object-oriented metrics. *Proceedings of Fourth IEEE International Workshop on Source Code Analysis and Manipulation,* IEEE.

[15] Baudry, B., & Traon Y. L. (2005). Measuring design testability of a UML class diagram. *Information and Software Technology, 47(13),* 859-879.

[16] Fu, J., & Lu M. (2009). Request-oriented method of software testability measurement. *International Conference on Information Technology and Computer Science*.

[17] Badri, M., & Toure, F. (2012). Empirical analysis of object-oriented design metrics for predicting unit

testing effort of classes. *Journal of Software Engineering and Applications*, *5(7),* 513-526.

[18] Badri, M., & Touré, F. (July 2011). Empirical analysis for investigating the effect of control flow dependencies on testability of classes. *Proceedings of 23rd International Conference on Software Engineering and Knowledge Engineering*.

[19] Badri, M., & Touré, F. (2012). Evaluating the effect of control flow on the unit testing effort of classes: An empirical analysis. *Advances in Software Engineering*.

[20] Singh, Y., Kaur, A., & Malhota, R. (October, 2008). Predicting testing effort using artificial neural network. *Proceedings of the World Congress on Engineering and Computer Science,* San Francisco, 22-24.

[21] Singh, Y., & Saha, A. (2010). Predicting testability of eclipse: A case study. *Journal of Software Engineering*, *4(2),* 122-136.

[22] Albattah, W., & Austin M. (2014). Package cohesion classification. *Proceedings of 2014 5th IEEE* International *Conference on Software Engineering and Service Science (ICSESS)*, IEEE, 2014.

[23] Albattah, W. & Alsuhibany, S. (2015) Revisiting Package-level cohesion approaches. ICSEA 2015, *Proceedings of the Tenth International Conference on Software Engineering Advances.*

[24] Albattah, W. An empirical investigation of the correlation between Package-level cohesion and maintenance effort. *International Journal of Advanced Computer Science and Applications (IJACSA)*, (in press).

[25] Martin, Robert Cecil. (2003). *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR.

[26] Apache. Retrieved March 2012, http://camel.apache.org

[27] Apache. Retrieved March 2013, from http://tomcat.apache.org)

[28] Hadoop. Retrieved March 2014, from http://hadoop.apache.org

[29] Synapse. Retrieved March 2014, from http://synapse.apache.org

[30] Apache. Retrieved March 2014, from http://ant.apache.org)

[31] Junit. Retrieved April 2014, from http://junit.org

[32] Beck, K. (2003). *Test-Driven Development: By Example*. Addison-Wesley Professional.

[33] Cockburn, A. (2000). Agile software development. *Cockburn Highsmith Series Editor.*

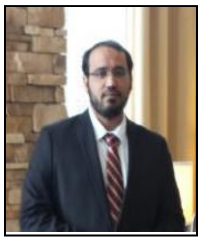
[34] Mahmoud O. E., Al-Yafei A. H., & Al-Mulhem M. (2011). Empirical comparison of three metrics suites for fault prediction in packages of object-oriented systems: A case study of Eclipse. *Advances in Engineering Software, 42(10),* 852-859.

[35] Virtualmachinery. (2001). Retrieved from http://www.virtualmachinery.com/jhawkprod.htm

[36] Aymen K., Toure F., & Badri M. (2011). An empirical analysis of a testability model for object-oriented programs. *ACM SIGSOFT Software Engineering Notes, 36(4),* 1-5.

[37] Cohen, J. (1988). Statistical power analysis for the behavioral sciences. 2nd edn. Lawrence Erlbaum, Mahwah.

[38] Gupta, V., & Chhabra J. K. (2012). Package level cohesion measurement in object-oriented software. *Journal of the Brazilian Computer Society*, *18(3),* 251-266.



Waleed Albattah received his Ph.D. from Kent State University, Ohio, USA. Dr. Albattah is a faculty member at Information Technology Department, Qassim University, Saudi Arabia. His research interests are software engineering, software measurements, software design and agile software development, and software quality. Recently, he has been working in Big data and cloud computing security projects.

He is the dean of College of Computer at Qassim University, and he is a member in ACM Society SIGSOFT.

Austin Melton received his Ph.D. from Kansas State University in 1980. Dr. Melton is a professor for the Departments of Computer Science and Mathematical Sciences. His research areas include formal semantics, the semantic web, software measurement theory, software engineering, security, lattice theory, fuzzy mathematics, Galois connections, and category theory. Lately, he has co-organized the Symposia on Fuzzy Mathematics, hosted by Kent State University and Youngstown State University.