SOFTWARE ARCHITECTURE UNDERSTANDABILITY IN OBJECT-ORIENTED SYSTEMS

By

TURKI F. ALSHAMMARY * MAMDOUH ALENEZI **

* College of Computer & Information Sciences, Prince Sultan University, Riyadh, Kingdom of Saudi Arabia.
** Chief Information and Technology Officer (CITO), Prince Sultan University, Riyadh, Kingdom of Saudi Arabia.

Date Received: 28/02/2018 Date Revised: 04/03/2018 Date Accepted: 09/04/2018

ABSTRACT

Software Architecture plays a vital role in the success or failure of software systems. Architecture understandability is a very important factor for managing and improving the system architecture. In this work, understandability of software architectures at the component-level will be explored. This study examines software structural properties of size, coupling, stability, and complexity against the effort spent by a developer to study a component. Number of software design metrics have been explored in the same context in the literature before, however, this work would explore a different combination of design metrics. A case study has been adopted from the literature that used an open source software system, which comprises of seven components. Analyses of Correlation, Collinearity, and Multivariate regression have been performed. The results of the statistical analyses indicate a correlation between most of the metrics used and the required effort needed to understand a component.

Keywords: Software Engineering, Software Architecture, Understandability, Object Oriented, Size, Coupling, Stability, Complexity, Software Design Metrics, Number of Methods, Loose Class Coupling, Tight Class Coupling, McCabe's Cyclomatic Complexity.

INTRODUCTION

In the object-oriented software systems, measuring architecture understandability is trending to be a major topic. Understandability is one of the most important characteristics of software quality because of the more difficult to understand a software system the more this would prevent or limit its reusability and maintainability and therefore can influence cost and/or reliability of software evolution (Stevanetic and Zdun, 2015). Software architecture is defined as, "the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them" (Bass et al., 1998). Understand ability is defined as the ease with which a model can be understood (Stevanetic and Zdun, 2014; Moody, 1998). It is a sub-characteristic of the usability characteristic of software quality as it is mentioned in ISO 9126-1 Quality Model (ISO, 2001). Hence, this work focuses on the area of software architecture understand ability metrics. It will extend exploration of some of the metrics that are believed to have an effect on the understandability of software architecture.

This work would extend a work done by (Stevanetic and Zdun, 2014) that explored the relationships between component-level metrics size, coupling and complexity and the required effort to understand a component. The design metrics that were used in (Stevanetic and Zdun, 2014) were Number of Classes as size metric, Number of incoming Dependencies and Number of Outgoing Dependencies as coupling metric and for complexity, Number of Internal Dependencies was used. On the other hand, in extending Stevanetic and Zdun work, this work aims to explore a different combination of design metrics that touches Size, Coupling, Complexity, and Instability. The design metrics are Number of Methods as a size metric, Loose Class Coupling (LCC), and Tight Class

Coupling (TCC) as coupling metrics, McCabe's Cyclomatic Complexity V(G) as a complexity metric and finally Instability and Distance as Instability metrics.

In this work, the first step is to develop and agree on research questions. The aim is to answer these research questions within the work. For this work, two research questions have been developed and agreed on and they:

(RQ1) What are the available metrics in the literature that measure design/ architecture properties understand ability of object-oriented software architecture?

(RQ2) What is the relationship of design metrics of size, coupling, complexity and stability, and the understandability of object-oriented software architecture?

1. Related Work

This section is designed to have a dedicated subsection for each metric. Some background information about what software architecture, understandability and the differences between package, component, and module is discussed. Tables A3 and A4 in the Appendices show a list of metrics and their information that was found in the literature and Information about the research papers, respectively.

1.1 Software Architecture

There are different definitions that define software architecture of a system. One of the famous ones that were repeatedly mentioned and used in the literature is "the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them" (Bass et al., 1998). IEEE 2000 standards define the software architecture as following, "Architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution" (IEEE, 2000). Another definition is used and it says, "software architecture is an abstraction of the run-time elements of a software system during some phase of its operation. A system may be composed of many levels of abstraction and many phases of operation, each with its own software architecture" (Shaw, 1990).

Gomaa (2011) has explained software architecture as, "a software architecture separates the overall structure of the system, in terms of components and their interconnections, from the internal details of the individual components."

Software architecture is used by architects to allow them focusing on the "big picture" of a system rather than the fine-grained details (Oreizy et al., 1999). In addition, software architecture is considered by Garlan (2000) to have a major role in 6 different sides of software development. These sides are understanding, reuse, construction, evolution, analysis, and management.

1.2 Package, Module, and Component

In the literature, one can notice that package, module and component levels are mentioned and that is because of the different levels of architecture that each literature is focusing on.

The package is used to represent a set of classes that might be hierarchically structured and to perform a series of related tasks (Gupta and Chhabra, 2009; Alenezi, 2016).

Niemeyer and Knudsen (2005) have defined a JAVA package as, "a group of classes that are related by purpose or by an application. Classes in the same package have special access privilege with respect to one another and may be designed to work together closely" (Niemeyer and Knudsen, 2005). A Package can contain sub-packages.

The module comprises of a large number of classes and "module is referred to as a package" (Alenezi, 2016). It provides information hiding for the module allowing a software engineer to see it as a black box (Hwa et al., 2009).

Alenezi (2016) has stated that, "A component in the context of object-oriented design is for organization purposes. The component contains a group of classes and other components as well." A component provides one or two similar system functionalities (Stevanetic and Zdun, 2014).

In conclusion, it is noticeable that these terms explain the same thing; a container that is used to organize and hold a set of related software artifacts together. However, a component can mean a different artifact organization

when it is looked at in the context of a UML Diagram.

1.3 Understandability

Understandability in the context of software architecture simply means the degree of understanding system architecture by the average architect/developer. Understandability is defined in the literature as the ease with which a model can be understood (Stevanetic and Zdun, 2014; Moody, 1998). It is a sub-characteristic of the usability characteristic of software quality as it is mentioned in the ISO 9126-1 Quality Model (ISO, 2001). It is considered as one of the most important sub-characteristics since the difficulty of understanding the software system limits and might even prevent the software system reusability and maintainability. This drawback would lead eventually to influence the cost or reliability of software evolution (Stevanetic and Zdun, 2015). Software Assurance Technology Centre has developed a software quality model for quantitative measurement of quality of software. SATC has proposed five quality contributors/attributes for the code and design phase, such as efficiency, complexity, understandability, reusability, and testability/maintainability (Khaliq et al., 2011).

1.4 Metrics

Metric, in general, is defined as the following, "it is merely a measurement of an arbitrary standard" (Knudsen and Niemeyer, 2005). In the Software field, a metric is defined as, "a quantitative measure of the degree to which a software item possesses a given quality attributes" (Xu and Nicolaescu, 2015). In this case, understandability is a sub-characteristic that if an engineer would want to control, manage and then enhance then he/she needs to measure it first.

However, there is literature that has explored relationships of certain design metrics at a certain view-level of architecture and the understandability of the architecture to a developer. Each of these view-level design metrics is discussed separately in the next sub-sections.

1.4.1 Size

Size metric in software is related to the number of artifacts of a certain system part in a certain view level of an architecture.

Elish (2010) has used as the measurement of a package-level size the number of classes (NC). The NC metric for a package is defined as the number of concrete and abstract classes (and interfaces) in the package. Elish has found that there is a positive significant correlation between the Number of Classes and Package understandability. As the NC increases in a package, the more effort is required to understand it.

Almousa and Alenezi (2017) in their quest to measure software architecture stability have considered size to be the main metric. They have stated that, "These functionality will be represented at the end for each class in the system either by adding new line of codes or method or imported packages or adding new interfaces."

Stevanetic and Zdun (2014) have considered the number of classes too in a component-level as the metric to measure the size. They have concluded that when a number of class increases in a component, it increases the needed effort to understand it. The needed effort is measured by the time that a programmer/developer took to understand the component and answer the related questions of the component.

Hwa et al. (2009) in their proposed hierarchical quality model have found that there is a positive influence of the design size in the module and it is defined as a number of modules (MD) in the design on understandability. The larger the size the harder is to understand.

The Number of Methods (NOM) in a class is another size metric that represents the total number of all methods implemented within a class (Bansiya and Davis, 2002). It is commonly used in Weighted Methods per Class (WMC) metric. Rosenberg and Hyatt (1997) have mentioned that WMC is either the number of the methods implemented within a class or the sum of the complexities of the methods. Also Bansiya and Davis (2002) defined the number of methods as, "a count of all methods defined in a class". However, Chidamber and Kemerer (1994) have considered this WMC to be a design metric that measures a complexity attribute of a system. Table A1 in the Appendices shows a list of the literature that used size metrics and their view level along with the type of the

metric used.

For this work, Number of methods metric is selected. The reason behind this selection is that NC metric is already explored for the same variables of the Case Study (Stevanetic and Zdun, 2014). Another reason is to explore different size metric that the already explored NC metric in both (Elish, 2010; Stevanetic and Zdun, 2014).

1.4.2 Coupling

The Coupling metric is defined as "the manner and degree of interdependence between packages" (Gupta and Chhabra, 2009). Both terms of Coupling and Cohesion are usually occurring together and caused some confusion. Both terms were first introduced by Stevens et al. (1974). The aim of a software module is to be loosely coupled and has a high cohesion in order to reduce maintenance and modification cost. There are different ways of measuring coupling at a different level of architecture's views found in the literature. One is the afferent (Ca) (Almousa and Alenezi, 2017) also known as Number of Incoming Dependencies (NID). Another one is the efferent (Ce) (Almousa and Alenezi, 2017) –also known as Number of Outgoing Dependencies. Also, Loose Class Coupling (LCC), and Tight Class Coupling (TCC) are found which measure class coupling.

The Ca metric for a package is defined as, "the number of other packages that depend upon classes within the package" (Elish, 2010). Ca is used to measure the incoming dependencies (fan-in) for a package. The Ce metric for a package is defined as, "the number of other packages that the classes in the package depend upon" (Elish, 2010). Ce is used to measures the outgoing dependencies (fan-out) for a package. LCC is defined as a metric that calculates the low dependency between object-structure at run-time (Sharma and Chug, 2015). TCC is defined as the metric that calculates the high dependency between object-structure at run-time (Sharma and Chug, 2015). In addition to these Coupling metrics, Gupta and Chahbra (2009) have proposed new metrics based on Package level coupling and the formal definitions and properties of packages (Package, Sub-Package, and class).

Gupta and Chahbra's (2009) proposed metrics have been created based on the different types of connections in the packages: Class – Class Connection, Sub-Package – Sub-Package Connection, Sub-Package – Class Connection, and Class – Sub-Package Connection. Moreover, the proposed metrics have been empirically validated by evaluating two open source Java projects. They have concluded that there is a strong correlation between package coupling and effort required to understand the package.

Elish has explored the relationships between a suite of five metrics and package understandability (Elish, 2010). Two of the five metrics are direct coupling metric and they are afferent couplings (Ca) and efferent couplings (Ce). Elish has found that there is a positive significant correlation between the Ca and package understandability and that is the higher number of "Incoming Dependencies" the more effort is required to understand the package. For the Ce, Elish found that there is a negative correlation between the Ce metric and the effort required to understand the package, which means simply high reusability of a package and that will lead to decreases the time needed to understand it.

Stevanetic and Zdun (2014) have used four types of metrics in their study to investigate the relationship of a number of component level metrics and the understandability of the architecture components. Two of these metrics are coupling metrics and they are Number of Incoming Dependencies (NID) and Number of Outgoing Dependencies (NOD). The authors found that when the NID in a component increases, the needed effort to understand the component increases as well. On the other hand, the NOD has shown no relationship between either the increase or decrease of the NOD in a component and the needed effort to understand the component.

Hwa et al. (2009) have proposed a hierarchical quality model (consist of 4 levels and 3 links to connect these levels) to assess the understandability of the modular design of an Object-Oriented software system. At the level 2 of their proposed model 6 design properties were identified that affect understandability of the modular design of a system. They have used for the coupling

metric the Direct Module Coupling (DMC) and that is the number of Modules which a module (MD) is directly related. They have found that the coupling property has a negative influence on understandability and that means the higher number of coupling the harder is to understand the system.

Sharma and Chug (2015) have carried out an empirical study to compare a list of dynamic software metrics (14 different Metrics) with a list of static software metrics (19 different Metrics) in the context of maintainability attribute. The authors have found that the dynamic metrics outperformed the static metrics. As the coupling property concerns, Sharma and Chug have identified Loose Class Coupling and Tight Class Coupling as dynamic software metrics. They measure the degree of how tightly or loosely a class is bounded with other classes (Kaur and Maini, 2016). LCC is defined as to, "calculate the low dependency between object-structure at run-time"(Kaur and Maini, 2016). For TCC, it is defined as to, "measure the high dependency between object-structure at runtime". Table A2 in the Appendices shows a list of the literature that used coupling metrics and their view level along with the type of the metric used (Kaur and Maini, 2016).

LCC and TCC are used as coupling metrics to explore the relationships between design metrics and component understandability. The reason behind this selection is that for this Case Study, NID, and NOD have been already explored with the relationship to the understandability (Stevanetic and Zdun, 2014; Elish, 2010). Second, the aim is to explore more metrics in the context of coupling and LCC and TCC are two metrics that were made available by the used Software Metric "Source Code Metric" (Warzocha, 2012) in this case study.

1.4. 3 Instability

Instability metric is defined as, "the ratio of efferent coupling (Ce) to total coupling (Ce+Ca) for the package" (Elish, 2010). The equation that describes the Instability it is as follows.

$$I = \frac{Ce}{Ce + Ca} \qquad (1)$$

This instability metric measures the resilience of a certain package to change (Elish, 2010). It has the range from zero to one. Zero measurement indicates to a totally stable package and one totally unstable package (Elish, 2010). Elish (2010) has found that there is a negative significant correlation between the Instability metric and the effort required to understand a package. Elish (2010) has interpreted that this metric against the understand ability as, "the easier to change a package the less effort is required to understand it". Therefore, Instability (I) metric will be included in the authors' exploration with other metrics.

1.4.4 Distance

Distance metric is an indicator of the package's balance between abstractness and stability (Elish, 2010). It is defined as, "the perpendicular distance of the package from the idealized line A+I = 1".

$$D = |A+I-1| \qquad (2)$$

In equation (2), A represents the percentage of the number of abstract classes to the total number of classes in a certain package. I represent the instability which is described in equation (1). Distance value ranges from zero to one. The zero number refers to a package that is coincident with the main sequence. On the other hand, the number one indicates that a package is at the furthest possible point from the main sequence (Elish, 2010).

Elish (2010) has found that when Distance metric increases, the effort needed to understand a package increases. He says that when there is unbalance of a package between abstracts and stability the need for more effort to understand that package increases. Therefore, Distance (D) metric will be included in their exploration with other metrics.

1.4.5 McCabe's Cyclomatic Complexity

McCabe's Cyclomatic Complexity is a metric that was introduced by McCabe in the 70s. In software programs, McCabe's Cyclomatic Complexity is considered the most used metric (Azim et al., 2008). It uses in its calculation the control flow graph and "measures the number of linearly-independent paths" (Azim et al., 2008). The result shows the degree of ease to understand and

modify the measured program. (McCabe, 1976) has defined his formula in his famous paper titled "A Complexity Measure" as "the cyclomatic number V(G) of a graph G with n vertices, e edges, and p connected components is"

$$V(G) = e-n+p \quad (3)$$

Second way is by the following equation

$$V(G) = \text{Total number of bounded areas} + 1 \quad (4)$$

As it has been mentioned in the beginning, 4 properties of size, coupling, instability, and complexity will be explored in this work. Therefore, McCabe's Cyclomatic complexity will be included in this case study.

2. Case Study

A case study from the literature is adopted where it is found to be applicable for the authors to use (Stevanetic and Zdun, 2014). This adoption would allow us to explore the relationships between Number of Methods (NOM), Loose Class Coupling (LCC), Tight Class Coupling (TCC), Instability (I), Distance (D) and McCabe's Cyclomatic Complexity (V(G)) components-level metrics and the effort needed to be measured in time that takes a programmer/developer to understand the component that was not covered by Stevanetic and Zdun's (2014) work. Stevanetic and Zdun (2014) have explored the relationships between "Number of Classes (NC), Number of Incoming Dependencies (NID), Number of Outgoing Dependencies (NOD) and Number of Internal Dependencies (NIntD) and the effort required to understand a component measured through the time spent on studying it". For this purpose, the already used open source software in (Stevanetic and Zdun, 2014) is used here and that is "Soomla Android store version 2.0". It contains "54 source code classes" allocated across "8 packages". The number of Line of Code is 3623 KLOC –not considering the comment and empty lines. However, it is important to mention that the software's packages are not considered in the broader sense for the Case Study in (Stevanetic and Zdun, 2014), but the components that are driven from the UML component diagram. The subject of the case study was 49 master students who were enrolled in the Advance Software Engineering course.

For the selected 6 metrics to be measured from the open source system "Soomla Android Store Version 2.0", Source Code Metric software is used (Warzocha, 2012) which is a NetBeans plug-in to generate the measurements.

Seven components were used for the component-level metrics after excluding the two external components Google Plays Server and SQL Lite Database. The description of the included seven components and their roles in the software is shown in Table 1.

The average time spent is based on the reading from the graph in Figure 1 that was found in (Stevanetic and Zdun, 2014).

The dependent variable is the Average Time Spent by participants to study the corresponding component. On the other hand, the independent variables that will be used in this case study are the 6 component level metrics mentioned earlier and they are Instability, Distance, Number of Methods, Loose Class Coupling, Tight Class Coupling, and McCabe's Cyclomatic Complexity. The source code metrics have been used to get the results of these metrics.

For NOM, the authors have summed up all the classes' methods in each component. For the LCC, TCC, and V(G), Source Code Metric software has calculated LCC

Component	Component's role
Security (C1)	Verifies the information during the purchasing process.
Crypt Decrypt (C2)	Provides encrypt/decrypt services to obfuscate the billing information and to encrypt/decrypt the data stored to or retrieved from the database.
Price Model (C3)	Describes the model that explains how the prices of virtual items are formed.
Google Play Billing (C4)	Simplifies in-app billing API which is a Google play service that lets you sell virtual goods from inside your applications.
Store Controller (C5)	Provides the runtime functionality of the Android store and contains up-to-date store information.
Database Services (C6)	Performs the initialization of the database and implement retrieve, add, and remove operations for store assets in the database.
Store Assets (C7)	Describes the virtual items used in the application (virtual currency, virtual goods, and their classification).

Table 1. Soomla Android Store Components and their Roles in the System (Stevanetic and Zdun, 2014)

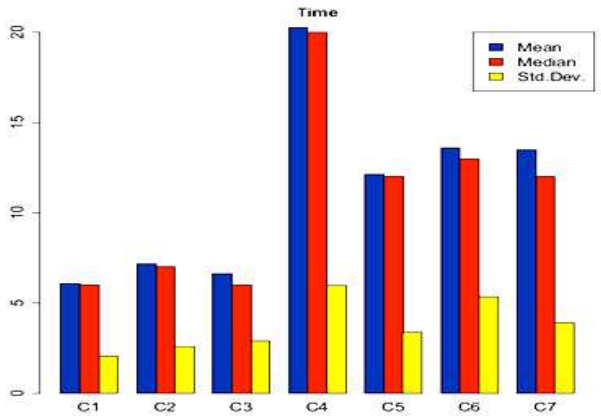

Figure 1. Mean, Median, and Standard Deviation Time Spent on each Component by Participants (Stevanetic and Zdun, 2014)

for each class in a component, hence, to get the LCC for each component, the average has been calculated. For I and D, they have calculated it by equations (1) and (2), respectively from the NIDs and NODs that were already measured by Stevanetic and Zduns (2014). The complete dependent and independent variables with their corresponding values that will be used in this case study are showing in Table 2.

2.1 Null Hypotheses

As it is mentioned previously, this work aims to explore the relationships of the previously selected design metrics with the understandability (measured in time spent by a developer to study and understand a component).

In this case study, the following null hypotheses are developed and will be tested:

Null Hypothesis I: I metric of a component does not have a significant correlation to the effort needed to be measured in time that takes a programmer/developer to understand the component.

Null Hypothesis II: D metric of a component does not have a significant correlation to the effort needed to be measured in time that takes a programmer/developer to understand the component.

Null Hypothesis III: NOM metric of a component does not have a significant correlation to the effort needed to be measured in time that takes a programmer/developer to understand the component.

Null Hypothesis IV: LCC metric of a component does not have a significant correlation to the effort needed to be measured in time that takes a programmer/developer to understand the component.

Null Hypothesis V: TCC metric of a component does not have a significant correlation to the effort needed to be measured in time that takes a programmer/developer to understand the component.

Null Hypothesis VI: V(G) metric of a component does not have a significant correlation to the effort needed to be measured in time that takes a programmer/developer to understand the component.

Component	Time Spent	I	D	NOM	LCC	TCC	V(G)
Security (C1)	6	0.57	0.43	7	0.29	0.29	3.00
Crypt Decrypt (C2)	7	0.00	1.00	37	0.19	0.15	1.20
Price Model (C3)	6.5	0.80	0.13	18	0.29	0.29	1.00
Google Play Billing (C4)	20	0.43	0.39	61	0.35	0.32	1.09
Store Controller (C5)	12	0.75	0.25	50	0.76	0.75	1.45
Database Services (C6)	14	0.50	0.50	43	0.29	0.28	2.33
Store Assets (C7)	13.75	0.25	0.60	61	0.54	0.41	0.86

Table 2. Dependent Variable (Average Time Spent on a Component) and the Independent Variables (I, D, NOM, LCC, TCC, and V(G))

3. Results and Discussion

For the results of the case study to be analyzed three types of analysis need to be performed. These analyses are Correlation Analysis, Multicollinearity Analysis, and Multivariate Regression Analysis.

The statistical analysis process is done with the help of an analysis software tool. The analysis software tool that has been used during the analyses is 'XLSTAT Version 2016.05.35252' (2016).

3.1 Correlation Analysis

Correlation Analysis is a statistical method used to evaluate if there are possible connections/relationships between two variables. If a correlation is found to be present between the two variables this means that when one variable changes, there is also a positive or negative change in the other variable. The range of the correlation coefficient is between +1 and -1. When a correlation coefficient is equal to +1 or -1 this indicates to the strongest possible positive or negative correlation

respectively. However, when the coefficient gets closer to the zero the weaker the correlation is and when it is equal to zero this means there is no correlation found in the relationship (Correlation Analysis-Market Research, 2016).

In this Correlation Analysis, Spearman's rank correlation test is performed with the level of significance $\alpha = 0.05$. This means that the level of confidence is 95%. The goal of this type of analysis is to determine if each component-level metric is related to the average Time Spent to understand the component.

The critical value of the spearman's ranking correlation coefficient is determined based on the significance level and the degree of freedom (df) which is the size of the sample data subtracted by 2. In this case study, the sample size was 49 different number of pairs of observations and the significance level is selected to be $\alpha=0.05$ (Stevanetic and Zdun, 2014). Based on Zar (1984), the critical value in this experiment for the Spearman's ranking correlation equals to 0.243.

3.1.1 Discussion

The results of Spearman's coefficient and the corresponding p-values between the average time that had been spent by participants on studying the components and component level metrics are put together in Table 3.

Based on the results in Table 3 and keeping in mind that the critical value = 0.243, the null hypotheses I, III, IV, V, VI are rejected and II is accepted.

I metric of a component does have a significant negative correlation to the effort needed to be measured in time that takes a programmer/developer to understand the component. It indicates that the decrease in the instability of a component leads to the increase of the needed effort to understand the component measured in time.

Metric	Coefficient r	p-value from Table	p-value from Equation
I	-0.393	P>0.10	0.383
D	0.143	P>0.10	0.760
NOM	0.865	0.025>P>0.01	0.012
LCC	0.571	0.1>P>0.05	0.180
TCC	0.250	P>0.10	0.589
V(G)	-0.250	P>0.10	0.589

Table 3. The Spearman Correlation Coefficients and Corresponding p-values between the Average Time Spent on studying a Component and the Selected Metrics

D metric of a component does not have a significant correlation to the effort needed to be measured in time that takes a programmer/developer to understand the component.

NOM metric of a component does have a significant positive correlation to the effort needed to be measured in time that takes a programmer/developer to understand the component.

LCC metric of a component does have a significant correlation to the effort needed to be measured in time that takes a programmer/developer to understand the component.

TCC metric of a component does have a significant correlation to the effort needed to be measured in time that takes a programmer/developer to understand the component.

V(G) metric of a component does have a significant correlation to the effort needed to be measured in time that takes a programmer/developer to understand the component.

3.2 Multicollinearity Analysis

The Multicollinearity analysis is a statistical condition where the independent variables are highly correlated (Hart, n.d.). The problem of the collinearity when it is present is that it causes the inflation of at least one estimated regression coefficient. To identify the collinearity, the Variance Inflation Factor (VIF) values need to be found.

- If the VIFs >10, this indicates to a serious collinearity (Elish, 2010; Stevanetic and Zdun, 2014).

According to Hart (n.d.), there are ways of dealing with Multicollinearity, we are getting rid of the "redundant" variables.

3.2.1 Correlation Matrix (Spearman)

The first step in the Multicollinearity analysis is finding the highest correlation's coefficient between the predictors. As shown in Table 4, we can see that there are strong positive correlations between (LCC and TCC) by the amount of 0.893 and between (I and D). So, the next step

is to exclude the predictor of the 4 predictors which has the highest VIF (Variance Inflation Number).

According to Table 5, TCC has a very high VIF (86.709) and as stated previously when VIF is greater than 10 this indicates to a serious collinearity. Hence, TCC will be eliminated from the list of variables.

After getting rid of the TCC variable from the list, VIF is recalculated between the remaining variables (NOM, LCC, V(G), I and D). As a result, we can see in Table 6 that I and D have very high VIF and one of them need to be eliminated to get rid of the redundancy. I metric has been chosen since it has the higher VIF = 33.104.

After getting rid of the I variable from the list, VIF is recalculated between the remaining variables (NOM, LCC, V(G) and D). Finally, all the remaining variables (NOM, LCC, V(G) and D) are way below 10 as shown in Table 7.

3.3 Multivariate Regression Analysis

The multivariate analysis is performed to build different multivariate linear regression models for predicting the time spent to understand a component (Elish, 2010). Based on the conclusion from the Multicollinearity Analysis in the previous section, there are four independent variables left after excluding Tight Class Coupling (TCC) and Instability (I) variables because of their high VIF – much greater than 10. The remaining variables are Number of Methods (NOM), Loose Class Coupling (LCC), McCabe's Cyclomatic Complexity (V(G)), and Distance (D).

Variable	NOM	LCC	TCC	V(G)	I	D
NOM	1					
LCC	**0.78**	1				
TCC	0.60	**0.89**	1			
V(G)	-0.51	-0.29	-0.4	1		
I	-0.41	0.18	0.32	0.21	1	
D	0.13	-0.32	-0.50	0.04	**-0.89**	1

Table 4. Values in bold are different from 0 with a Significance Level Alpha=0.05

	R^2	Tolerance	VIF
NOM	0.777	0.223	4.491
LCC	0.985	0.015	67.493
TCC	0.988	0.012	86.709
V(G)	0.454	0.546	1.831
I	0.988	0.012	83.618
D	0.981	0.019	51.811

Table 5. Multicollinearity Statistics

	R^2	Tolerance	VIF
NOM	0.734	0.266	3.755
LCC	0.620	0.380	2.631
V(G)	0.454	0.546	1.831
I	0.970	0.030	33.104
D	0.962	0.038	26.18

Table 6. Multicollinearity Statistics without TCC

15 prediction models have been designed that involve all the possible combination of the selected variables as shown in Table 8. Since four variables are present than 2^4–1 (the subtraction of the combination that excludes the variables) which equals to 15 Models. For the evaluation and accuracy comparison of the 15 developed prediction models, first, the procedure of Leave-One-Out Cross-Validation (LOOCV) is used (Refaeilzadeh et al., 2009). Then the accuracy of the 15 models is determined based on de facto standard and commonly used measures of the Mean Magnitude Relative Error (MMRE) and Prediction at level 0.25 (Pred(0.25)) (Kitchenham et al., 2001).

As shown in Table 8, all the different 15 models with their corresponding MMRE and Pred(0.25) are listed.

3.3.1 Discussion of Multicollinearity and Multivariate Regression Analyses

In the Multicollinearity Analysis, TCC and I were found to have very high VIFs which were much greater than 10 due to the high correlation with LCC and D, respectively. After eliminating TCC and I variables from the list, VIF is recalculated between the remaining variables (NOM, LCC, V(G) and D). Finally, all the remaining variables (NOM, LCC, V(G) and D) are way below 10 as shown in

	R^2	Tolerance	VIF
NOM	0.576	0.424	2.358
LCC	0.486	0.514	1.947
V(G)	0.346	0.654	1.529
D	0.338	0.662	1.51

Table 7. Multicollinearity Statistics without TCC and I

Model	NOM	LCC	V(G)	D	MMRE	Pred(0.25)
1	X				0.811	28.57%
2		X			1.127	0.00%
3			X		0.607	14.29%
4				X	1.258	0.00%
5	X	X			0.533	42.86%
6	X		X		1.118	14.29%
7	X			X	0.556	42.86%
8		X	X		0.788	0.00%
9		X		X	1.541	0.00%
10			X	X	0.919	0.00%
11	X	X	X		0.619	42.86%
12	X	X		X	0.590	14.29%
13	X		X	X	0.617	28.57%
14		X	X	X	1.202	0.00%
15	X	X	X	X	0.325	71.43%

Table 8. Evaluation of the Prediction Models with MMRE and Pred(0.25)

Table 7. Hence, the Multicollinearity is not strong between these independent variables.

From Table 8, it is noticed that Model 15 has achieved the best MMRE value of 0.32. In terms of Pred(0.25), the best model is Model 15 as well; achieved a value of 71.43%.

Conclusion

This work has explored the relationships between six component-level metrics and the average time that took a programmer/developer to study a component in order to understand it. The metrics were Number of Methods (NOM), Loose Class Coupling (LCC), Tight Class Coupling (TCC), Instability (I), Distance (D) and McCabe's Cyclomatic Complexity. These metrics measure different structural properties of a component, such as size, coupling, stability, and complexity. Next, a case study was performed using open source software. The subjects of the experiment were 49 master students. Three types of analysis were performed and they are Correlation, Collinearity, and Multivariate Regression. The results from the correlation analysis have shown that all of the metrics except (Distance) have significant correlations to the effort needed to be measured in time that takes a programmer/developer to understand the component. In Multivariate Regression, the model that included all the metrics that measure the different structural properties is better than the ones that did not.

In the case study, there are limitations that are common and can be found in the literature. The case study has used a dataset that is recently published (Stevanetic and Zdun, 2014). This adopted dataset is small in size since it comprises of 7 components. However, three types of analysis were performed.

For the future work, it would involve exploring more metrics of the different structural properties. For example, Size metric, in this work the NOM metric has been explored and Number of classes metric has been explored in (Elish, 2010; Stevanetic and Zdun, 2014), yet the LOC as a size metric has not been explored. Same applies to Coupling, Complexity, and Instability. Moreover, more software systems which have large numbers of components need to be used in the exploration in the quest to measure and study the software architecture understandability.

References

[1]. Alenezi, M. (2016). Software Architecture Quality Measurement Stability and Understandability. *International Journal of Advanced Computer Science and Applications (IJACSA),* 7(7), 550-559.

[2]. Almousa, H., & Alenezi, M. (2017). Measuring Software Architecture Stability Evolution in Object-Oriented Open Source Systems. *Journal of Engineering and Applied Sciences,* 12(2), 353-362.

[3]. Architecture Working Group of the Software Engineering Committee. (2000). Recommended Practice for Architectural Description of Software Intensive Systems. *IEEE Standards Department,* Piscataway, New Jersey, USA.

[4]. Azim, A., Ghani, A., Koh Tieng, W. G., Muketha, M., & Wen, W. P. (2008). Complexity metrics for measuring the understandability and maintainability of business process models using Goal-Question-Metric. *International Journal of Computer Science and Network Security,* 8(5), 219-225.

[5]. Bansiya, J., & Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering,* 28(1), 4-17.

[6]. Bass, L., Clements, P., & Kazman, R. (1998). *Software Architecture in Practice.* Addisson.

[7]. Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on*

Software Engineering, 20(6), 476-493.

[8]. Correlation Analysis - Market Research. (n.d.). Retrieved from http://www.djsresearch.co.uk/glossary/item/Correlation-Analysis-Market-Research

[9]. Elish, M. O. (2010, June). Exploring the relationships between design metrics and package understandability: A case study. In *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on* (pp. 144-147). IEEE.

[10]. Garlan, D. (2000, May). Software architecture: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 91-101). ACM.

[11]. Gomaa, H. (2011). *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures.* Cambridge University Press.

[12]. Gupta, V., & Chhabra, J. K. (2009). Package coupling measurement in object-oriented software. *Journal of Computer Science and Technology,* 24(2), 273-283.

[13]. Hart, J. D. (n.d.). *Collinearity of independent variables.* Retrieved from http://www.stat.tamu.edu/~hart/652/collinear.pdf on October 21 2016.

[14]. Hwa, J., Lee, S., & Kwon, Y. R. (2009, December). Hierarchical understandability assessment model for large-scale OO system. In *Software Engineering Conference, 2009. APSEC'09. Asia-Pacific* (pp. 11-18). IEEE.

[15]. International Organization for Standardization, & International Electrotechnical Commission. (2001). *Software Engineering--Product Quality: Quality Model* (Vol. 1). ISO/IEC.

[16]. Kaur, S., & Maini, R. (2016). Analysis of Various Software Metrics used to Detect Bad Smells. *Int J Eng Sci (IJES),* 5(6), 14-20.

[17]. Khaliq, M., Khan, R. A., & Khan, M. H. (2011). Significance of Design Properties in Object Oriented Software Product Quality Assessment. *TECHNIA – International Journal of Computing Science and Communication Technologies,* 3(2), 622-625.

[18]. Kitchenham, B. A., Pickard, L. M., MacDonell, S. G., & Shepperd, M. J. (2001). What accuracy statistics really measure [software estimation]. In *IEE Proceedings - Software* (Vol. 148, No. 3, pp. 81-85). doi: 10.1049/ip-sen:20010506

[19]. McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering,* SE-2(4), 308-320.

[20]. Moody, D. L. (1998, November). Metrics for evaluating the quality of entity relationship models. In *International Conference on Conceptual Modeling* (pp. 211-225). Springer, Berlin, Heidelberg.

[21]. Niemeyer, P., & Knudsen, J. (2005). *Learning Java.* O'Reilly Media, Inc.

[22]. Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimhigner, D., Johnson, G., Medvidovic, N., ... & Wolf, A. L. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and their Applications,* 14(3), 54-62.

[23]. Refaeilzadeh, P., Tang, L., & Liu, H. (2009). Cross-validation. In *Encyclopedia of Database Systems* (pp. 532-538). Springer US.

[24]. Rosenberg, L. H., & Hyatt, L. E. (1997). Software quality metrics for object-oriented environments. *Crosstalk Journal,* 10(4), 1-6.

[25]. Sharma, H., & Chug, A. (2015, September). Dynamic metrics are superior than static metrics in maintainability prediction: An empirical case study. In *Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), 2015 4th International Conference on* (pp. 1-6). IEEE.

[26]. Shaw, M. (1990). Toward higher-level abstractions for software systems. *Data & Knowledge Engineering,* 5(2), 119-128.

[27]. Statistical software and data analysis add-on for Excel | XLSTAT. (n.d.). Retrieved October 21, 2016, from https://www.xlstat.com/en/

[28]. Stevanetic, S., & Zdun, U. (2014, May). Exploring the relationships between the understandability of components in architectural component models and component level metrics. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (p. 32). ACM.

[29]. Stevanetic, S., & Zdun, U. (2014, October). Empirical Study on the Effect of a Software Architecture

Representation's Abstraction Level on the Architecture-Level Software Understanding. In *Quality Software (QSIC), 2014 14th International Conference on* (pp. 359-364). IEEE.

[30]. Stevanetic, S., & Zdun, U. (2015, April). Software metrics for measuring the understandability of architectural structures: a systematic mapping study. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering* (p. 21). ACM.

[31]. Stevanetic, S., & Zdun, U. (2016, April). Exploring the Understandability of Components in Architectural Component Models Using Component Level Metrics and Participants' Experience. In *Component-Based Software Engineering (CBSE), 2016 19th International ACM SIGSOFT Symposium on* (pp. 1-6). IEEE.

[32]. Stevanetic, S., Javed, M. A., & Zdun, U. (2015, September). The impact of hierarchies on the architecture -level software understandability-a controlled experiment. In *Software Engineering Conference (ASWEC), 2015 24th Australasian* (pp. 98-107). IEEE.

[33]. Stevens, W. P., Myers, G. J., & Constantine, L. L. (1974). Structured design. *IBM Systems Journal,* 13(2), 115-139.

[34]. Warzocha, K. (2012, May 14). Source Code Metrics - NetBeans Plugin detail. Retrieved October 21, 2016, from http://plugins.netbeans.org/plugin/42970/sourcecodemetrics

[35]. Xu, Y., & Nicolaescu, A. (2015). Evolution of Object Oriented Software Coupling Metrics. *Full-scale Software Engineering,* 79.

[36]. Zar, J. H. (1984). Comparing simple linear regression equations. *Biostatistical Analysis,* 2, 292-305.

Appendices

Literature's Authors	Level	Metric
Stevanetic and Zdun (2014)	Component	Number of Classes
Elish (2010)	Package	Number of Classes
Hwa et al. (2009)	Module	Number of Classes
Rosenberg and Hyatt (1997)	Class	Number of Methods
Bansiya and Davis (2002)	Class	Number of Methods

Table A1. View-level and the used Size Metric

Literature's Authors	Level	Metric
Gupta and Chahbra (2009)	Package	General Coupling • Class–Class Connection • Sub-Package–Su-Package Connection • Sub-Package–Class Connection • Class–Sub-Package Connection
Stevanetic and Zdun (2014)	Component	(NID) Number of incoming dependencies (Fan-in) (NOD) Number of outgoing dependencies (Fan-out)
Elish (2010)	Package	(Ca) Afferent Couplings (Fan-in) (Ce) Efferent Couplings (Fan-out)
Hwa, Lee, and Kwon (2009)	Module	General Coupling
Sharma and Chug (2015)	Class	Loose Class Coupling (LCC) Tight Class Coupling (TCC)

Table A2. Coupling Metric that is used at a certain Level

Metric Name	Abbreviation	Definition	Validation
Number of Classes	NC	The number of concrete and abstract classes (and interfaces) in the package	Correlation, MultiCollinearity and multivariate regression analysis
Module Size in Classes	MSC	Total number of classes in a module	Empirical
Package Coupling Metric	PCM	Summation of coupling of the package with all other packages present at the hierarchical level	Theoretical and Empirical
Afferent / Number of Incoming Dependencies	Ca / NID	Total number of dependencies between the classes outside of a component and the classes inside a component that are used by those outside classes.	Correlation, Multicollinearity and multivariate regression analysis
Efferent/ Number of Outgoing Dependencies	Ce/ NOD	Total number of dependencies between the classes inside a component and the classes outside of a component that are used by those outside classes.	Correlation, Multicollinearity and multivariate regression analysis
Distance	D	It indicates to the package's balance between abstractness and stability	Correlation, Collinearity and multivariate regression analysis
Instability	I	It measures the resilience of a certain package to a change.	Correlation, Multicollinearity and multivariate regression analysis
Number of Internal Dependencies	NIntD	Total number of dependencies between the classes within a component	Correlation, Multicollinearity and multivariate regression analysis
Number of Methods	NOM	Number of method	Empirical
Loose Class Coupling	LCC	It measures the degree of how loosely a class is bounded with other classes.	Empirical
Tight Class Coupling	TCC	It measures the degree of how tightly a class is bounded with other classes.	Empirical
McCabe's Cyclomatic Complexity	V(G)	It measures of control flow complexity.	Testing methodology
Abstraction Level	--	It is the level where it is sufficient to adequately map the system's relevant functionalities to the corresponding architectural components.	Empirical
Hierarchal Abstraction	--	Architectural representation where architectural components at all abstraction levels in the hierarchy are shown.	Empirical

Table A3. List of Metrics and their Information that were found in the Literatures

References	Goal	Research Methodology	Systems Used
Gupta and Chhabra (2009)	To propose new metrics for measurement of package level coupling.	Theoretical and Empirical	• XGen Source Code Generator • Jakarta Element Construction Set (ECS)
Khaliq et al. (2011)	To explore the relationships between five package-level metric (Size, Afferent, Efferent, Instability and Distance) and the average effort required to understand a package in O.O. design.	Empirical	• Xgen Source Code Generator • Jakarta Element Construction Set (ECS)
Alenezi (2016)	Systematic mapping study on software metrics related to the understandability concept of such higher-level software structures with regard to their relations to the system implementation	Systematic Review	NA
Hwa et al. (2011)	To propose a hierarchical model to assess understandability of modularization in large-scale O.O. software.	Empirical	JFreeChart
Bass (1998)	To examine the relationships between the efforts required to understand a component, measured through the time that participant spent on studying a component and component level metrics that describe component's size, complexity and coupling.	Experimental	Soomla Android store Version 2.0
IEEE (2000)	To examine the effect of the level of abstraction of the software architecture representation (3 levels) on the architecture-level understandability of a software system.	Experimental	Soomla Android store Version 2.0
Stevens et al. (1974)	To examine the impact of hierarchies on architectural-level software understandability.	Empirical	WebWork version 2.2
Warzocha (2012)	To extend their previous studies (Stevanetic, Javed and Zdun, 2015) the impact of personal factors of participants like experience and expertise and the combinations of both personal factors and the metrics.	Empirical	Soomla Android store Version 2.0
McCabe (1976)	Software quality metrics for object-oriented environments.	NA	NA
Oreizy (1999)	Dynamic metrics are superior to static metrics in maintainability prediction: An empirical case study.	Empirical	Hodoku 1.1 and Hodoku 2.2.
Shaw (1990)	A complexity measure	Empirical	Various Fortran Programs

Table A4. Information about the Research Papers

ABOUT THE AUTHORS

Turki Alshammary is currently a Senior System Analyst at the Ministry of Foreign Affairs in the Kingdom of Saudi Arabia. He received BS in Computer Engineering from the Western Sydney University and his MS in Software Engineering from Prince Sultan University. He is a certified Business Analyst (CBAP), (PMP) and (ITIL) and a member of the Saudi Council of Engineers. His research interests, include Software Engineering and Big Data.

Dr. Mamdouh Alenezi is currently the Chief Information and Technology Officer (CITO) at Prince Sultan University. He received MS and Ph.D. Degrees from DePaul University and North Dakota State University in 2011 and 2014, respectively. He is a member of the Institute of Electrical and Electronic Engineers (IEEE). His research interests, include Software Engineering, Open Source software, Software Security, and Data Mining. He teaches mainly software engineering courses.