Journal of Theoretical and Applied Information Technology
15th July 2020. Vol.98. No 13
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

PRACTICAL FRAMEWORK FOR SOFTWARE INTEGRITY MEASUREMENT BASED ON STATIC ANALYSIS

SHAREFA MURAD
Assistant Professor, Middle East University, Department of Computer Science, Amman, Jordan

E-mail: smurad@meu.edu.jo

ABSTRACT

Quality metrics technology is a disciplined approach used to evaluate, predict, and specify the software quality in terms of software integrity. The obtained metrics interprets the criteria influencing the implementation of software integrity. The primary goal of software integrity regarding security is ensuring the commendable and basic data around the organizations and makes it effectively achievable. In this study, we proposed two methods for finding the integrity bugs in software implementation. The first methods involve determining and consolidating the criteria of evaluation while in the second method we employed Chidamber and Kemerer Java Metrics (CKJM) to compute the software quality metrics. Furthermore, three open-source Java frameworks are utilized to analyze the effect of quality metrics on the implementation of software integrity. The proposed framework enabled to examine the most extensively employed metrics to test software integrity i.e the number of children, the coupling between objects, and response for class respectively. The empirical results suggested that the metrics calculated by the proposed framework have a significant effect on software integrity. The benefits of the proposed system to the absolute use of measurements at steady appearances of software products help in the early identification of software quality-related issues. Specific appraisal of quality levels gives better administration permeability and empowers auspicious dynamics.

Keywords: *Software Quality, Software Integrity, Correlation, ANOVA test, Quality metrics*

1. INTRODUCTION

Data security is the establishment of any conducive industry. Corporations can accomplish the objective of data security by having appropriate data security tools to ensure data against vulnerabilities. In a disseminated condition, it is a major issue that how to guarantee in two-way communication to make sure about data is protected [1, 2, 3]. In a System Engineering, non-functional requirements spout the patterns that can be utilized to assess the activity of the framework, which interprets as a simulate section of the framework not identified with its accomplishment but instead to its advancement after some time to make it increasingly distinct [4, 5, 6]. One notable non-functional requirement is considered as a security of specific software [7].

The primary objective of security is ensuring the commendable and basic data concerning the relationships between the data and makes it effectively reachable. Attackers utilize various strategies, apparatuses, and procedures to harm the frameworks and try to de-activate business tasks [8]. Accomplishing reliable software products becomes a challenge for the business experts; that is because it needed a profound comprehension of various viewpoints such as security categories, security policies, and security evaluations [9, 10].

Security testing targets approving evaluation criteria based on the framework requirements identified with security properties [11]. For empowering the direction of test recognizable proof, several security evaluation strategies are accessible for a long time, including the critical methodologies that take into consideration detail of experiments at a more elevated level of analysis [11]. Software metrics are frequently employed to evaluate the capacity of software to accomplish a pre-described objective [12]. It can be defined as a proportion to evaluate the characteristics of software in terms of security, integrity, and robustness. Furthermore, it can be utilized to quantify during different software development stages such as feasibility study, design, and developmental process [13].

Several researchers have used numerous models to ensure the security measure for the integrity of software development. This study depends on the principle Triple Security Model (TSP) likewise named the CIA model [14, 15]. Specifically, the CIA depends on three fundamental security ideas which include availability, integrity, and confidentiality. Each model relies on these three measures to structure the software. Figure 1 exhibits the CIA triangle.

Confidentiality is a terminology adopted to anticipate the revelation of data to unapproved people or frameworks [17]. Integrity implies that information cannot be altered without approval.

It implies keeping up and guaranteeing the precision and consistency of information over its whole life-cycle [17]. Lastly, the availability of the data must be accessible when it is required [17]. Various upgrades in the figuring condition and the wide affection of software technology in instructive foundations and associations; there are diverse new assaults routinely enforced on the Integrity of frameworks, the confidentiality of information, and the copyrights of media suppliers [18].

Security specialists have perceived that in the vast majority of the cases, security attacks are because of inadequately data storing methods and the evaluation metrics to investigate the quality score of any particular software. Hence there is a need to have such a software evaluation metric that helps to ensure software integrity in every aspect.

In this study, we have proposed a framework that defined standards to identify the integrity bugs into the implementation of the software and gives the experimental proof of associated correlation metrics (CK) with integrity. The proposed framework consists of two methods. The first methods involve determining and consolidating the criteria of evaluation while in the second method we used static tools analysis using the CKJM tool to compute the metrics.

Furthermore, we examine the impact on the integrity of software by the probability of CK metrics. Following hypothesis are considered for this research.

H0: (Null Hypothesis) No correlation between CK metrics and Integrity.

H1: (Substitute Hypothesis) Correlation exists between CK metrics and Integrity.

For the validation of the hypothesis, we used the CKJM tool [20] to consist of the open-source program to show the correlation.

A. Research Objectives

The primary objectives of the underlying results includes:

1. Establish standards and rules to examine software integrity for avoiding probable bugs.
2. Computing integrity score by defining quality metrics.
3. To determine if the computed metrics correlated with the integrity of software.

Availability
Integrity
Confidentiality

Figure 1: Basic CIA Triangle [16]

2. BACKGROUND

This section depicts the past studies related to estimating software integrity for plaining a model-based analysis set up for security.

Livshits et al. [21] have carried out a latent investigation method to distinguish numerous application vulnerabilities, for example, HTTP parting assaults, cross-site scripting, and SQL infusions respectively. A user-friendly framework was designed to tackle vulnerabilities that are consequently converted into static analyzers. Similarly in [22], the purpose of their examination is the exhibition of the prescient model which utilizes evaluation criteria to measure the integrity based on the level of code. The analysis dissects 13 datasets of the "NASA Metrics Data Program". By implementing a space of factual significant tests and demonstrating approaches, the proposed modal guaranteed superiority over other approaches that are based on design metrics.

The authors in [4] considered security concerns due to the cloud being vulnerable. To ensure the collected information was unblemished since it was unfeasible to download the full information, a security component named reliability check was utilized. The integrity check was accomplished by probabilistic structures, for example, cuckoo channels and blossom channels. Transferring information documents on to one cloud server could be a state of failure, rather, the record could be disseminated over l servers.

Al-Badareen et al. [23] brought out a detailed comparison of the state of the art models designed for the evaluation of software quality such as Boehm Model [24], McCall Model [25] and IEC 9126 Model [26]. The outcome shows the shortcomings and quality of those principles in estimating security and other quality sides. In [13], the authors have given exact proof that non-cohesive, complex and, coupled software substances are commonly weak in terms of the security wall. The author investigated the absence of Cohesion, Coupling, and Complexity metrics at a factually critical level decidedly connected to the number of vulnerabilities. The relationship is on the p-value under 0.001 with normal 0.5.

The author proposed a strategy [5] for confirming the quality of integrity ranks by an electronic gadget, the technique includes getting to a module of the related frame, acquiring a mark (M), preparing a confirmation key (CK), related control key (COK) relating to a marking key (MSK), confirming whenever said signature (WSS) was determined by marking related frame module with said marking key (SK), by utilizing said confirmation key (VK), and building up a positive confirmation module. The work likewise gives a strategy for giving a check the integrity to play out the previously mentioned techniques.

Integrity Quantification Model" (IQM) based on the designed with object-oriented proposed by [27]. Their study consist of the evaluation of soft-ware security integrity, regarding unpredictability factors such as Coupling between Object (CBO), Total Supporting Services (TSS), Higher Level of Abstraction (HLA), and Coupling Function (CP). Schieferdecker et al. [28] carried out a novel security testing approach termed as"Model-based security testing". The suggested technique committed to the adequacy and the orderly detail of the security test of instances objects semi-robotized object. The MBST procedure incorporates security functional tests and model-based evaluation to prevent the framework from attackers with daily basis experiments.

A survey carried out by the authors in [29] on different techniques of security testing. Their study includes the most advance and recent advancements of security testing systems applied during the protected software improvement life cycle such as dynamic analysis, penetration testing, static analysis, code-based testing, and model-based security testing.

Lastly, the security testing procedures are represented by utilizing i.e. web-based applications with three-tiered architecture. The authors accompanied a systematic mapping study (SMS) in [30] supporting a thorough convention that was designed dependent on the best in class SMS and orderly survey rules. For data extraction from a large number of important studies, they deliberately recognized 34 essential MBSE4CPS. The empirical analysis for two on-going years (2014-2015) reveals that the quantity of essential MBSE4CPS examines has expanded altogether. Inside the essential investigations, the notoriety of utilizing Domain-Specific Languages (DSLs) is practically identical with the utilization of the UML demonstrating documentation.

MobSTer, an adaptable model-based security testing framework proposed in [31]. MobSTer considered as the superior framework in terms of filling the hole between these two security testing draws. The fundamental thought of the system is that the utilization of model-checking strategies can robotize the quest for conceivable section focuses on the web application, i.e., it allows an investigator to perform security testing without avoiding significant checks. Additionally, the system likewise considers reusability: the investigator can gather the evaluation results into the structure and (re)use it during future tests on conceivably unique web applications. The authors impersonate a study [32] of the ongoing exploration endeavors in coordinating learning-based models with testing. They recognized two coasts of literature review which include test-based demonstrating and learning-based testing. Furthermore, they managed the outcomes as far as their test object strategies, their hidden models, and their objective domains.

Hardware Security Modules (HSM) has carried out in the study [33] to be tested by a strong composite of three test determination standards to examine security prerequisites of segments. The proposed combination dependent on the utilization of static test determination rules, to be specific auxiliary model composition, supplemented by unique test criteria. Their methodology is executed in mechanical and adaptable MBT tools.

The author proposed another software quality measurement [14], for anticipating the unwavering quality of segments, in software that is based on the components. In their examination, a standard quality measurement like attachment, coupling, and cyclomatic unpredictability was distinguished, utilizing head part analysis (PCA). To approve the recently proposed measurements factually, the t-test was applied to it.

They assessed and efficiently employed it to three security applications based on the real world. Another information-based security testing technique is proposed in [34] by rational programming and the associated tool execution. The proposed strategy assists with defeating the contemporary pervasive center on functional requirements rather than the non-functional requirement just as the necessary elevated level of security information when performing non-functional security testing. The issues tackled by the suggested approach are: dealing with the vast measure of negative security experiments, non-functional requirements for testing, and non-functional requirement manipulation for the non-expert analyzers.

Simos et al. [35] carried out standard test criteria for TLS security protocols to prevent an application from the most recent attacks including BREACH, ROBOT, and DROWN respectively. They converged around cybernetic experiment period and performance concerning the TLS security protocol, wherever the point is to join arranging including combinatorial techniques for giving experiments likewise uncovering the previously hidden attacks. That is composed attainable by making proper information parameter models for various messages that can show up in a TLS communication system.

3. METHODOLOGY

In this study, we proposed two methods for finding the integrity bugs in software implementation. The first methods involve determining and consolidating the criteria of evaluation while in the second method we used static tools analysis using the CKJM tool to compute the metrics.

The evaluation metrics can be defined as :

NOC-expresses to the number of prompt subclasses (Children) subjected to the class (parent) in the family chain of command. NOC gauges what number of strategies or fields acquired legitimately through selections of a superclass. .Orders with countless children need to offer progressively nonexclusive support to all the children in different settings and ought to be increasingly adaptable, an imperative that can bring greater intricacy into the parent class.

CBO-Is the quantity of different sources that a class is linked to. This type of coupling is only feasible for object-oriented frameworks. For instance strategies for individual use techniques or occurrence factors of another, because a similar class has similar properties, the coupling between two classes is held when strategies pronounced in one class use techniques or case factors characterized by the different classes. Over the top coupling demonstrates shortcoming of class exemplification and may hinder re-use.

RFC-It can be characterized as a collection of strategies that can be conceivably affected because of an internal ping received through the object of the concerning class. This entire metric is also relevant to object-oriented systems. On the off chance, that an enormous number of strategies can be summoned in light of an internal ping, at that point, the testing and troubleshooting of the class turns out to be progressively muddled because it expects a more noteworthy degree of understanding required concerning the analyzer.

LOC- essentially checks the total number of lines a source code contains (can be defined as line break characters) of a specific software substance, this type of metric is straightforward yet ground-breaking to examine the intricacy of software techniques and elements.

The demonstration of the calculation of integrity is given in equation 1.

$$\text{Integrity} = 1 - \frac{\text{Errors}}{\text{Lines of Code}} \qquad (1)$$

Lastly, we analyze the relative correlation between the obtained metrics and integrity.

A. Proposed model 1

The following 5 phases involve are in the proposed method consist of defining and combining the evaluation criteria to identify the software integrity bugs in the implementation point of view.

1. Obtain the origin of warnings & bugs in source code.
2. Segregate the warnings & bugs into two different categories.
3. Choose the warnings & bugs with the strongest integrity relationship.
4. Exploring the bugs using the static analysis tools such as FindBugs, VC, and Jtest.
5. Implementation of static analysis tools for static bug analysis.

The flow chart of the proposed model based on combining the criteria is presented in figure 2.

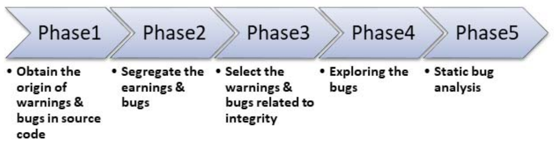

Phase1
Phase2
Phase3
Phase4
Phase5
• Obtain the origin of warnings & bugs in source code
• Segregate the earnings & bugs
• Select the warnings & bugs related to integrity
• Exploring the bugs
• Static bug analysis

Figure 2. Flow Chart Of The Proposed Model.

B. Proposed Model 2

After obtaining the activities, these projects are embedded in static analysis tools at the code level to find the Bugs that influence on integrity. The flow diagram of the proposed model based on static bug analysis is presented in figure 3.

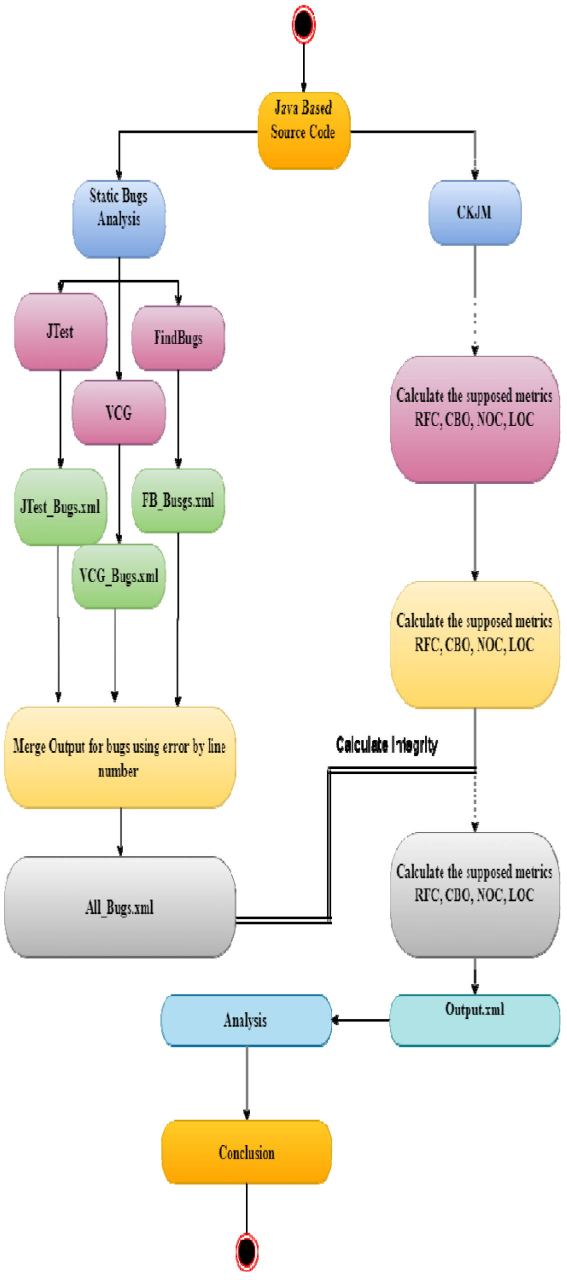

Java Based Source Code
Static Bugs Analysis
CKJM
JTest
FindBugs
VCG
Calculate the supposed metrics RFC, CBO, NOC, LOC
JTest_Bugs.xml
FB_Busgs.xml
VCG_Bugs.xml
Calculate the supposed metrics RFC, CBO, NOC, LOC
Merge Output for bugs using error by line number
Calculate Integrity
All_Bugs.xml
Calculate the supposed metrics RFC, CBO, NOC, LOC
Output.xml
Analysis
Conclusion

Figure 3. Flow diagram of the proposed model based on static bug analysis.

The following phases are contributed to form the proposed framework.

Step 1: The source files are parsed through the Jtest tool for finding the bugs in the source code along with the relationship among the metrics and integrity. The output obtained by the Jtest tool is given in figure 4.

For detecting the faults in the source file, the complete project is scanned through the VCG tools.

The output is given by the VCG framework is presented in figure 5.

Similarly, the layer of FindBugs takes the source file as input to find the bugs. The output achieved by the FindBugs toolbox is demonstrated in figure 6.

Likewise, the CKJM tool is employed for the software project's source files to be parsed to compute the intended metrics such as LOC, CBO, NOC, and RFC. The metric results obtained by CKJM is expressed in figure 7.

Step 2: At this stage, the output achieved from FindBugs, VCG, CKJM, and Jtest is further added to form a single source code file. During the process, the entry appears as duplicated error or warning is removed from the file regardless of their parent bug. The output of the merging process is given in figure 8.

The integrity of the merged source code is then calculated using the CKJM. Figure 9 shows the CKJM results with integrity obtained by merging FindBugs, VCG, CKJM, and Jtest source files.

Step 3: In this step, we used the results from figure 9 to compute the relationship and the influence of CBO, RFC, and NOC metrics upon integrity. Furthermore, the proposed framework utilized Multiple Linear Regression for obtaining the relevant coefficients against each metric corresponds to the relationship with integrity. MLR is proposed in [36] for investigating software integrity during the developmental process. MLR attempts to shape a connection between at least two translations factors and the reaction variable utilizing the linear equation. The free factor x in each of the reaction variables is associated with an estimation variable y. MLR can be defined by the following equation.

$$Y = 0 + 1X1 + 2X2 + 3X3 + \ldots\ldots\ldots\ldots + nXn \quad (2)$$

Where Y: denoting the dependent variable which can be defined as what is being explained and $_0$: (Beta) expressed as the intercept point. Where $_1$: is the Slope for X.

4. EVALUATION AND RESULTS

A. Subject Evaluation

For the evaluation of the proposed modal, we used three open-source projects to check and compared in terms of the relationship between metrics and integrity. The obtained projects include which are Commercial, Payment, and ArgUML respectively. The Commercial and Payment projects are related to the security while ArgUML is related to OpenGL communication in the hyper networks. These programs were downloaded directly from Sourceforge and Github. The detailed description of the used projects is given in Table 1.

B. Regression Results

1. ArgoUML Project: Table 2 expressed the total number of variables entered and deleted when using the ArgoUML project.

SPSS permits users to enter factors based on the specific type of variable for regression analysis. This implies maintaining the co-relation between dependent and independent variables. Similarly, deleting variables that were expelled from the regression slope fr stepwise regression alignment. Choosing the Enter method of SPSS implies that every autonomous variable was entered in the standard style.

The results obtained by employing ArgoUml for the proposed model is presented in Table 3.

In Table 3, R representing the correlation precision relationship between the dependent variable and the metrics, where R-square can be defined as the square root of correlation based on the proportion of precision variance and the independent variable. Furthermore, a general proportion of the quality of affiliation is dependent on the degree to which a specific independent variable is correlated. Std referencing a root mean squared error can be obtained by taking the standard deviation of the error. The Pearson Correlation of 0.13 from Table 3, showing a relationship between RFC, NOC, CBO on integrity, nevertheless, R-square is 0.018 which symbolizes 1.8% of the variation of integrity.

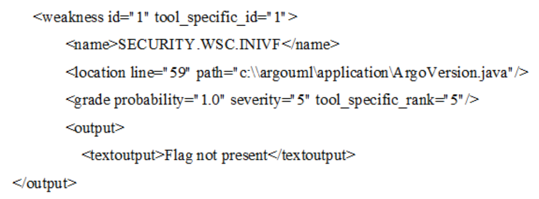
The ANOVA Test and Individual Regression Coefficients test are expressed in Table 4 and 5.

2. Shopizer-Ecommerce Project: Table 6 expressed the total number of variables entered and deleted when using the Shopizer-Ecommerce project.

The results obtained by employing ArgoUml for the proposed model is presented in Table 7. The ANOVA Test and Individual Regression Coefficients test are expressed in Tables 8 and 9.

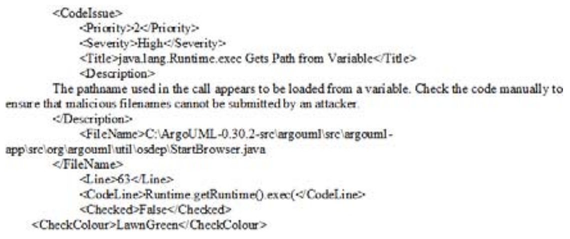
Table 8, shows that Sig= 0.942 over 5%, which implies the fluctuation of critical contrast between

CBO, RFC, NCO respectability. The previously mentioned table shows that Sig= 0.942 over 5%, which implies that there is a no measurably huge contrast between metrics and integrity and that implies a no factually noteworthy connection.



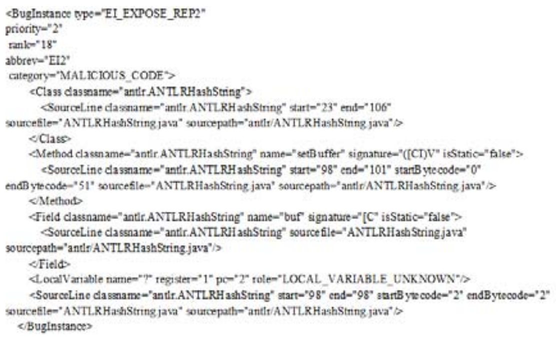
```
<weakness id="1" tool_specific_id="1">
    <name>SECURITY.WSC.INIVF</name>
    <location line="59" path="c:\\argouml\application\ArgoVersion.java"/>
    <grade probability="1.0" severity="5" tool_specific_rank="5"/>
    <output>
      <textoutput>Flag not present</textoutput>
</output>
```

Figure 4. Jtest results.



```
        <CodeIssue>
            <Priority>2</Priority>
            <Severity>High</Severity>
            <Title>java.lang.Runtime.exec Gets Path from Variable</Title>
            <Description>
        The pathname used in the call appears to be loaded from a variable. Check the code manually to
ensure that malicious filenames cannot be submitted by an attacker.
        </Description>
            <FileName>C:\ArgoUML-0.30.2-src\argouml\src\argouml-
app\src\org\argouml\util\osdep\StartBrowser.java
        </FileName>
            <Line>63</Line>
            <CodeLine>Runtime.getRuntime().exec(</CodeLine>
            <Checked>False</Checked>
    <CheckColour>LawnGreen</CheckColour>
```

Figure 5. VCG Results.



```
<BugInstance type="EI_EXPOSE_REP2"
priority="2"
 rank="18"
abbrev="EI2"
 category="MALICIOUS_CODE">
    <Class classname="antlr.ANTLRHashString">
      <SourceLine classname="antlr.ANTLRHashString" start="23" end="106"
sourcefile="ANTLRHashString.java" sourcepath="antlr/ANTLRHashString.java"/>
    </Class>
    <Method classname="antlr.ANTLRHashString" name="setBuffer" signature="([CI)V" isStatic="false">
      <SourceLine classname="antlr.ANTLRHashString" start="98" end="101" startBytecode="0"
endBytecode="51" sourcefile="ANTLRHashString.java" sourcepath="antlr/ANTLRHashString.java"/>
    </Method>
    <Field classname="antlr.ANTLRHashString" name="buf" signature="[C" isStatic="false">
      <SourceLine classname="antlr.ANTLRHashString" sourcefile="ANTLRHashString.java"
sourcepath="antlr/ANTLRHashString.java"/>
    </Field>
    <LocalVariable name="?" register="1" pc="2" role="LOCAL_VARIABLE_UNKNOWN"/>
    <SourceLine classname="antlr.ANTLRHashString" start="98" end="98" startBytecode="2" endBytecode="2"
sourcefile="ANTLRHashString.java" sourcepath="antlr/ANTLRHashString.java"/>
  </BugInstance>
```

Figure 6. Find Bug Results.

Similarly, Table 9, expressing the significant effect of correlated metrics on integrity.

3. Payment4j Project: The results obtained by employing Payment4j for the proposed model is presented in Table 10.

Table 10 reveals that Correlation Pearson obtained is 0.0122, signifies the existence of a feeble connection between CBO, NOC, and RFC, on integrity, Wherever, R2 is 0.002 which demonstrates that 1.5% of the variety in a subordinate variable (respectability) can be clarified by the independent variables CBO, NOC, and RFC. The ANOVA test is expressed in Table 11.

5. ANALYSIS

From Table 4, the Sum of Squares related to the three well-springs of fluctuation Residual, regression, and Total. The Total fluctuation is divided into the difference which can be clarified by the autonomous variables (Regression) and the change which isn't clarified by the autonomous variables (Residual). Furthermore, F-measurement the p-value related to the Mean Square isolated by the Residual. The p-value is contrasted with some alpha level in testing the invalid speculation that the entirety of the model coefficients are 0. Table IV shows that Sig= 0.000 under 5%, where the factual relationship utilized is 5%, and it implies that there is a measurably critical distinction between CBO, RFC, and NOC.

Referred to the Table 5, Beta is the institutionalized coefficients that acquire on the off chance for institutionalizing the entirety of the variables in the regression, including the reliant and the entirety of the autonomous variables, and encompassed the reverse of the regression. By regulating the variables before covering the loss, the variables on a similar scale are passed through the user-defined entry which is dependent on the size of the coefficient.

Table 7, reveals that Correlation Pearson obtained is 0.021, signifies the existence of a feeble connection between CBO, NOC, and RFC, on integrity, Wherever, R2 is 0.005 which demonstrates that 0.5% of the variety in a subordinate variable (respectability) can be clarified by the independent variables CBO, NOC, and RFC. Table 11, shows that Sig= 0.04 less than 5%, which implies the fluctuation of critical contrast between CBO, RFC, NCO respectability.

In the Figure 10, the Cross correlation for Individual regression results obtained by employing argouml is presented. We can see that the Lag and CCF are significantly improved when comparing T and Std. error for the constant method as appose to the CBO, RFC, and NCO respectability. The achieved cross correlation ratio is -4.0 to 1.0.

From table 4, Taking a gander at the analysis of difference in the result variable, these are the classes we have computed Regression, Residual, and Total. The Total change is parceled into the fluctuation which can be clarified by the free

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

factors (Model) and the difference which is not clarified by the autonomous factors (Error).

Table 9 depicts the regression results which shows the model is identical to the necessity for dependability; the variety between them is that unwavering quality is concerned about all errors of the outcome, and reliability is concerned uniquely with the subset of errors that sway on the integrity. Marginal comparison for Shopizer-Ecommerce project in terms of Std error and B value is presented in the figure 11, which depicts the low marginal error with respect to the correlation and high B value with respect to the marginal significance. Kruskal-Wallis average ranks on 3 employed projects are presented in the Table 12 and shows significant ranks for CBO as compared to other models.

6. LIMITATIONS

We perceive that there are sure restrictions to the outcomes and effects of the underlying research.

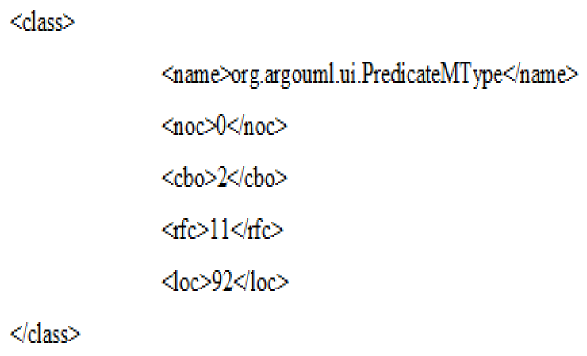
In the first place, our exploration depends on Bugs which have just been found and proclaimed. The bugs that have not been found or openly declared at this point are not utilized in our examination even though such data may add to progressively specific analysis.

The way that numerous different elements can influence quality metrics in terms of integrity. In this manner, we suggest that RFC, CBO, and NOC measurements ought to be the sole thought when to gauge integrity in the product lifecycle.

We recognize that one contextual analysis is not adequate to draw totally broad and solid outcomes. A few conclusions drawn from examining 3 trials may not have any significant bearing to another software integrity in various areas.

7. CONCLUSION

Software quality evaluation defined as the complicated procedure and partitioned into three angles; functional quality, structural quality, and process quality. It is critical to assess the software quality through the degree of code and the flow chart after execution. Consequently, there is a need to characterize criteria to decide integrity determined by the bugs in the actualized software. CK metrics are employed to evaluate the quality of software. In this research, we have carried out a static analysis tool for detecting the bugs to assure software quality and integrity. The recommended framework adopted JM to measure correlation metrics. Moreover, three open-source Java frameworks are employed to examine the effect of metrics. The empirical result confirmed that the independent variable RFC produces a notable influence on software integrity as compared to the NOC. Where the co-relation results obtained by CBO reinforce the zero effect with the lowest relationship between metrics and integrity.



```
<class>
        <name>org.argouml.ui.PredicateMType</name>
        <noc>0</noc>
        <cbo>2</cbo>
        <rfc>11</rfc>
        <loc>92</loc>
</class>
```

Figure 7. CKJM Results.

Table 1. Detail Description Of The Used Projects For Comparison

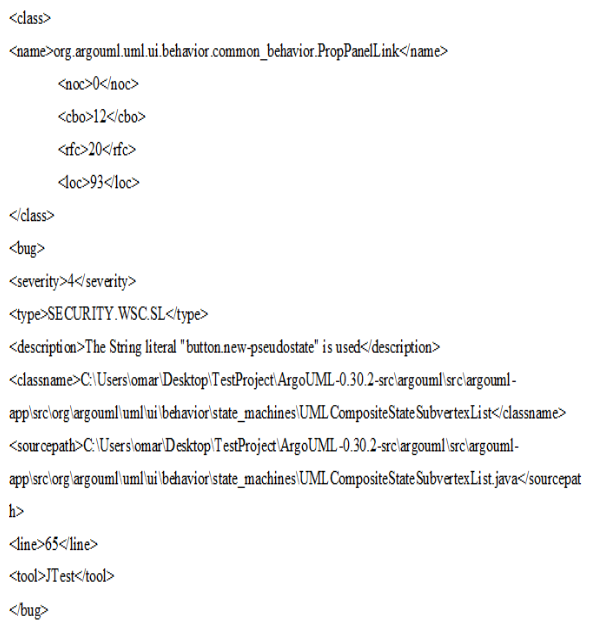
Name	Location	# of Class	Version
shopizer-ecommerce	https://github.com/shopizer-ecommerce/shopizer	861	2.0.0
payments4j	https://github.com/CarlosZ/payments4j	194	0.0
ArgoUML	http://argouml-downloads.tigris.org/source/browse/	1914	0.30.2

Table 2. Variables Entered And Deleted When Using Argouml Project.

Variables Entered/Deleted			
Model	Entered	Deleted	Method
	NOC, RFC, CBO	-	Enter

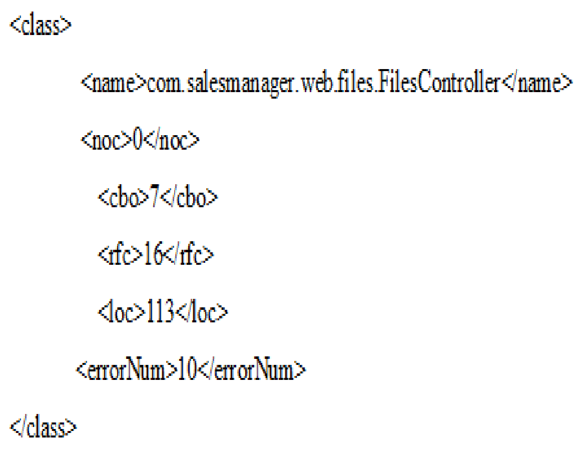
Table 3. Results Obtained By Employing Argouml For The Proposed Model

a.Predictors: NOC, RFC, CBO				
Model	std	Adjusted R Square	R Square	R
1	.09415	.016	.018	.134a



```
<class>
<name>org.argouml.uml.ui.behavior.common_behavior.PropPanelLink</name>
        <noc>0</noc>
        <cbo>12</cbo>
        <rfc>20</rfc>
        <loc>93</loc>
</class>
<bug>
<severity>4</severity>
<type>SECURITY.WSC.SL</type>
<description>The String literal "button.new-pseudostate" is used</description>
<classname>C:\Users\omar\Desktop\TestProject\ArgoUML-0.30.2-src\argouml\src\argouml-
app\src\org\argouml\uml\ui\behavior\state_machines\UMLCompositeStateSubvertexList</classname>
<sourcepath>C:\Users\omar\Desktop\TestProject\ArgoUML-0.30.2-src\argouml\src\argouml-
app\src\org\argouml\uml\ui\behavior\state_machines\UMLCompositeStateSubvertexList.java</sourcepat
h>
<line>65</line>
<tool>JTest</tool>
</bug>
```

Figure 8. The output of the merging process.



```
<class>
        <name>com.salesmanager.web.files.FilesController</name>
        <noc>0</noc>
          <cbo>7</cbo>
          <rfc>16</rfc>
          <loc>113</loc>
        <errorNum>10</errorNum>
</class>
```

Figure 9. CKJM Results With Integrity Obtained By Merging Findbugs, VCG, CKJM, And Jtest Source File

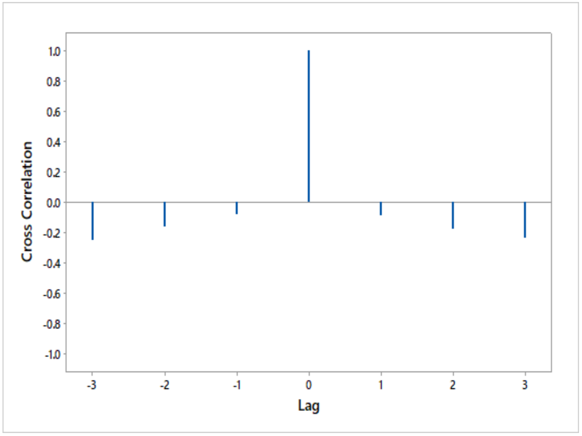


Figure 10. Cross Correlation For Individual Regression Results Obtained By Employing Argouml

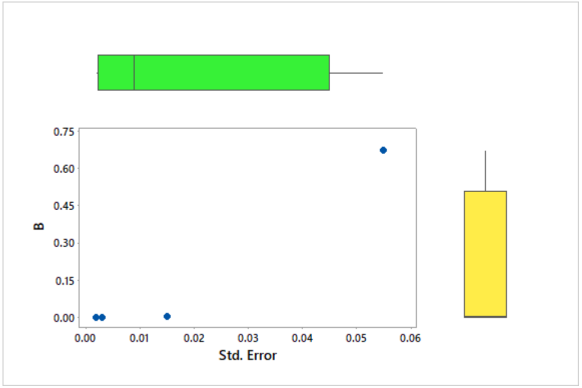


Figure 11. Marginal Comparison For Shopizer-Ecommerce Project In Terms Of Std Error And B Value

Table 4. Anova Results Obtained By Employing Argouml For The Proposed Model

ANOVAb						
Model		Mean Square	Df	Sum of Square	Sig	F
1	Residual	.009	1911	16.939		
	Regression	.104	3	.311	11.677	000a
	Total		1914	17.249		
a. Predictors: NOC, RFC, CBO b. Integrity.						

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 5. Individual Regression Results Obtained By Employing Argouml

Coefficientsa						
Model		Unstandardized Coefficients		Standardized Coefficients	Sig.	T
		Beta	Std. Error	B		
1	(Constant)	0.134	0	0.002	0	5.328
	NO C	-0.001	0	-3.85E-06	0.977	-0.029
	CB O	0.004	0	8.60E-05	0.853	0.185
	RF C		0.003	0.943	0	338.469
a.Integrity						

Table 6. Variables Entered And Deleted When Using Shopizer-Ecommerce Project.

Variables Entered/Deleted			
Model	Entered	Deleted	Method
	NOC, RFC, CBO	-	Enter

Table 7. Results Obtained By Employing Shopizer-Ecommerce For The Proposed Model

a.Predictors: NOC, RFC, CBO				
Model	Std	Adjusted R Square	R Square	R
1	1.24428	-0.003	0.005	0.21a

Table 8 Results Obtained By Employing Shopizer-Ecommerce For The Proposed Model
.

ANOVAb						
Model		Mean Square	Df	Sum of Square	Sig	F
1	Residual	.202	3	.606		
	Regression	.104	3	.311	130	942a
	Total		1914	17.249		
a. Predictors: NOC, RFC, CBO b. Integrity.						

Table 9 Regression Results Obtained By Employing Shopizer-Ecommerce For The Proposed Model
.

Coefficients a						
Model		Unstandardized Coefficients		Standardized Coefficients	Sig.	T
1		Beta	Std. Error	B		
	RFC	.018	.002	.001	.643	.464
	CBO	-.004	.003	.001	.911	-.112
	NOC	.015	.015	.006	.672	.423
	(Constant)		.055	.671	.000	12.246
a.Integrity						

Table 10 Results obtained by employing Payment4j for the proposed model

a.Predictors: NOC, RFC, CBO				
Model	Std	Adjusted R Square	R Square	R
1	.22804	0.000	.122a	.122a

Table 11 Anova results obtained by employing Payment4j for the proposed model

ANOVAb						
Model		df	Sum of Squares	F	Mean Square	Sig.
1	Regression	3	.151	0.968	0.05	.04a
	Residual	191	9.93		0.52	
	Total	194	10.08			

a. Predictors: NOC, RFC, CBO

Table 12. Kruskal-Wallis Avergaeg Ranks On 3 Employed Projects

Methods	Shopizer-Ecommerce	Payment4j project	ArgoUml project
Constant	4.0	4.0	4.0
NOC	3.0	3.0	2.0
CBO	1.5	1.0	1.0
RFC	1.5	2.0	3.0

ACKNOWLEDGEMENT
The author thanks the Middle East University, Amman, Jordan for their financial support of this research and their continuous encouragement.

REFRENCES:

[1] A. Nikishova, T. Omelchenko, and Y. Umnitsyn, “Software integrity control,” *2019 International Russian Automation Conference (RusAutoCon)*. IEEE, 2019, pp. 1–5.
[2] M. Ahmadvand, A. Pretschner, and F. Kelbert, “A taxonomy of software integrity protection techniques,” in *Advances in Computers*. Elsevier, 2019, vol. 112, pp. 413–486.
[3] S. Murad, I. Passero, and R. Francese, “Metric pictures: Source code images for visualization, analysis and elaboration,” in *Information Technology and Innovation Trends in Organizations*. Springer, 2011, pp. 279–287.
[4] H. S. Asl, B. M. Tazehkand, and M. J. Museviniya, “Distributed scfmbf based protocol for integrity in cloud storage system (dpics),” *SN Applied Sciences*, vol. 2, no. 2, pp. 1–18, 2020.
[5] N. Asokan, J. Mantyla, and R. Serafat, “Method and device for verifying the integrity of platform software of an electronic device,” Jan. 30 2020, uS Patent App. 16/591,192.
[6] R. Francese, S. Murad, I. Passero, and G. Tortora, “Metric pictures: The approach and applications,” *The 2010 International Conference on Computer Engineering & Systems*. IEEE, 2010, pp. 320–325.
[7] J. Son, S. Koo, J. Choi, S.-j. Choi, S. Baek, G. Jeon, J.-H. Park, and H. Kim, “Quantitative analysis of measurement overhead for integrity verification,” in *Proceedings of the Symposium on Applied Computing*, 2017, pp. 1528–1533.
[8] C. A. Sennewald and C. Baillie, Effective security management. Butterworth-Heinemann, 2020.
[9] S.-F. Wen and B. Katt, “Toward a context-based approach for software security learning,” *Journal of Applied Security Research*, vol. 14, no. 3, pp. 288–307, 2019.
[10] A. Al-Far, A. Qusef, and S. Almajali, “Measuring impact score on confidentiality, integrity, and availability using code metrics,” in *2018 International Arab Conference on Information Technology (ACIT)*. IEEE, 2018, pp. 1–9.
[11] O. N. Basheer, “Practical measurements frame-work for software integrity,” Ph.D. dissertation, Middle East University, 2015.
[12] R. Z. Frantz, M. H. Rehbein, R. Berlezi, and F. Roos-Frantz, “Ranking open source application integration frameworks based on maintainability metrics: A review of five-year evolution,” *Software: Practice and Experience*, vol. 49, no. 10, pp. 1531–1549, 2019.

[13] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, 2011.

[14] R. Sehgal, D. Mehrotra, and R. Nagpal, "New metrics for predicting the reliability of individual component based on software design metrics," in *International Conference on Distributed Computing and Internet Technology*. Springer, 2019, pp. 79–94.

[15] E. Dias Canedo, K. Valenca, and G. A. Santos, "An analysis of measurement and metrics tools: A systematic literature review," in *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 2019, pp. 9–20.

[16] S. Samonas and D. Coss, "The cia strikes back: Redefining confidentiality, integrity and availability in security." *Journal of Information System Security*, vol. 10, no. 3, 2014.

[17] Y. Sattarova Feruza and T. Kim, "It security review: Privacy, protection, access control, assurance and system security," *International journal of multimedia and ubiquitous engineering*, vol. 2, no. 2, pp. 17–32, 2007.

[18] S. Chandra and R. A. Khan, "An empirical validation of integrity risk factor metric: An object-oriented design perspective," *International Journal of Software Engineering and Knowledge Engineering*, vol. 3, no. 8, 2013.

[19] R. Bashir and A. G. Dunn, "Software engineering principles address current problems in the systematic review ecosystem," *Journal of clinical epidemiology*, vol. 109, pp. 136–141, 2019.

[20] J. Singer, S. Marion, G. D. Brown, R. E. Jones, M. Lujan,´ C. Ryder, and I. Watson, "An information theoretic evaluation of software metrics for object lifetime prediction," in *2nd Workshop on Statistical and Machine learning approaches to Architectures and compilation (SMART'08)*, 2008, pp. 1–14.

[21] V. B. Livshits and M. S. Lam, "Finding security vulnerabilities in java applications with static analysis." in *USENIX Security Symposium*, vol. 14, 2005, pp. 18–18.

[22] Y. Jiang, B. Cuki, T. Menzies, and N. Bartlow, "Comparing design and code metrics for software quality prediction," in *Proceedings of the 4th international workshop on Predictor models in software engineering*, 2008, pp. 11– 18.

[23] A. B. Al-Badareen, M. H. Selamat, M. A. Jabar, J. Din, and S. Turaev, "Software quality models: A comparative study," in *International Conference on Software Engineering and Computer Systems*. Springer, 2011, pp. 46–55.

[24] B. Boehm, J. Brown, H. Kaspar, and M. M. Lipow, "Fj & merritt, mj (1978). Characteristics of software quality. trw series of software technologies 1."

[25] J. A. McCall, "Quality factors," Encyclopedia of software engineering, 2002.

[26] R. G. Dromey, "A model for software product quality," *IEEE Transactions on software engineering*, vol. 21, no. 2, pp. 146–162, 1995.

[27] S. A. Khan and R. A. Khan, "Integrity quantification model for object oriented design," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 2, pp. 1–3, 2012.

[28] I. Schieferdecker, J. Grossmann, and M. Schneider, "Model-based security testing," arXiv preprint arXiv:1202.6118, 2012.

[29] M. Felderer, M. Buchler,¨ M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, "Security testing: A survey," in *Advances in Computers*. Elsevier, 2016, vol. 101, pp. 1–51.

[30] P. H. Nguyen, S. Ali, and T. Yue, "Model-based security engineering for cyber-physical systems: A systematic mapping study," *Information and Software Technology*, vol. 83, pp. 116–135, 2017.

[31] M. Peroli, F. De Meo, L. Vigano,` and D. Guar-dini, "Mobster: A model-based security testing framework for web applications," *Software Testing, Verification and Reliability*, vol. 28, no. 8, p. e1685, 2018.

[32] B. K. Aichernig, W. Mostowski, M. R. Mousavi, M. Tappler, and M. Taromirad, "Model learning and model-based testing," in *Machine Learning for Dynamic Software Analysis: Potentials and Limits*. Springer, 2018, pp. 74–100.

[33] J. Botella, J.-F. Capuron, F. Dadeau, E. Fourneret, B. Legeard, and F. Schadle, "Complementary test selection criteria for model-based testing of security components," *International Journal on Software Tools for Technology Transfer*, vol. 21, no. 4, pp. 425–448, 2019.

[34] P. Zech, M. Felderer, and R. Breu, "Knowledge-based security testing of web applications by logic programming," *International Journal on Software Tools for Technology Transfer*, vol. 21, no. 2, pp. 221–246, 2019.

[35] D. E. Simos, J. Bozic, B. Garn, M. Leithner, F. Duan, K. Kleine, Y. Lei, and F. Wotawa, "Testing tls using planning-based combinatorial methods and execution framework," *Software quality journal*, vol. 27, no. 2, pp. 703–729, 2019.

[36] O. Fedotova, L. Teixeira, H. Alvelos et al., "Software effort estimation with multiple linear regression: Review and practical application," *Journal of Information Science and Engineering,* vol. 29, no. 5, pp. 925–945, 2013.