Software Source Code: Theoretical Analyzing and Practical Reviewing Model

Ayman Odeh
Al Ain University, College of Engineering, Al Ain/UAE (ORCID: 0000-0002-3892-6488)

Article History: Received: 10 November 2020; Revised 12 January 2021; Accepted: 27 January 2021; Published online: 5 April 2021

Abstract : The high-quality software product is one of the most important goals of software engineering in general, source coding or implementation phase is one of the main steps of the software development process; when this phase is going in a good way and according to the source code quality standards and rules, the final product will be a high-quality product. This research focuses on the issues that can affect source code quality, such as errors, faults, defects, and failures; a practical system model will be provided (proposed) to predict some possible types of errors and defects and suggests some guidance and recommendations on source code error detection and analysis.

Keywords — Source Code, Quality Assurance, Source Code Analyzing, Software Errors, Defects, Failures

1. Introduction

"Software engineering can be defined as a discipline that is concerned with all aspects of software production" (Sommerville 2016)(Lester 2008); it is a process of producing software products, and this product must satisfy customer requirements, and software standards of quality assurance factors and criteria(Maxim and Kessentini 2016)(Laporte and April 2017). Generally, the software process consists of phases starting from receiving the customer requirements, analysis these requirements, system and program design, implementation, testing, and delivery, at the end of each phase there is a product, this research is concerning on the analyzing implementation phase and its final product, which is source code in addition to the testing process.

Given a source code, how to be sure that it will do exactly performs its function or no? "This question is not only intellectually challenging but also of primary importance in practice."(Huang 2009), to answer this question many methods were implemented to proof of being the software source code are free from errors.

The analyzing of software errors includes methods and techniques applied to determine, test, data collection, rating and assessment errors and everything related to software errors. In perfect words and with extreme optimism, the software development process, especially source code implementation, should be error- free. Practically, error-free software is a dream, but reducing number of errors as much as possible is achievable goal.

When software developers do mistakes and errors in early phases in lifecycle development process, the cost and effort of error discovering and maintain, in the implementation phase, will be very high. Any mistake in the requirement phase can lead to an error in the design, and this error surely will cause an error or defect in the source code implementation as shown in Figure1, subsequently it will be very difficult to detect all errors in the testing phase, because it is not easy and possible to consider all possible test cases during the testing. So it is possible for some serious errors in the software to remain undetected, and converted to defects then may lead to a general software system failure.

1.1. Background of the Research:

The ability to understand software product error characteristics is totally as important as understanding the cost and effort of software development for the development team. The software error's nature consists of the rate at which errors occur, the effort and cost of discovering and eliminate (removing) errors, the riskiness of the errors, the popular reasons causing errors, and the most effective methods to identify and avoiding errors(Bassman, M. J., McGarry, F., & Pajerski 1994).

Source code as the final product of the implementation phase can contain different types of mistakes and errors, the developers of this phase trying to deliver the free-error product as much as they can. They can eliminate syntax errors by compiling the source code using programming language compilers, it's a good step

but not enough, many other types of errors, such logical, programming language's errors, and .... ect, can stay to be found in the testing phase.

In the testing phase, the testers are doing their best work to find and discover all errors to achieve the testing goal, but there are cases where some errors can hide other types of errors, and lead to unpredictable outputs; the developers, in this case, can never be sure that if the unexpected outputs caused by new errors or by the original errors (Sommerville 2016).

The process of identifying, fixing, and removing errors from source code (debugging and inspections) is one of the main goals of this research. To achieve this aim, the comprehensive analysis of different error types, defects, fault, and failure will be discussed.

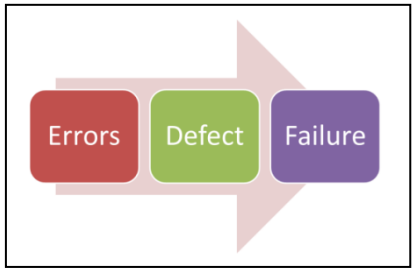

Figure 1: Error-Defect-Failure Relationship

1.2. Aims and Objective:

The purpose and the desired outcomes of this research is to provide an analytical and practical approach for understanding the nature of source code errors, and how to eliminate and remove them to produce high-quality source code through the following objectives:

- To understand the nature of source code errors, defect and failures, and identify the relationship among them.
- To analyze the effects of each error type.
- To identify, fixing, and removing errors and defects from source code.
- To reduce cost effort of error detection.
- Provide recommendation and guidelines for evaluating source code.

1.3. Problem Statement

Producing high quality software product is highly affected by implementing high quality source code. The distinctive attributes and characteristics of source code are getting extremely influenced and touched by errors of the source code and by the internal errors of the programming language used in development the software product.

1.4. The Research Significance

This work has a very important impact as all software programmers and developers are encountering difficulties in defining and eliminating errors, defects, and failures in the development phase due to the nature and effects of source code errors, defect, and failures, and also the relationship among them, and its negative impact on the overall quality of software products.

2. Literature Review

This section includes the review of important works and concepts related to the source code errors, and software analyzing.

2.1. "Source Code Analysis – An Overview"(Kirkov and Agre 2010)

The "Source Code Analysis – An Overview"(Kirkov and Agre 2010) describes the algorithms, major structure, techniques, procedures, and experimental comparison of some tools used for software source code analysis. It illustrates the result of source code analysis using some tools to analyze program sample. The researchers in this work are suggesting using machine learning, and data mining for source code analyzing.

2.2. Source Code Size Effects on Errors

When the size of the software (source code) is increasing, the type and quantity of errors will be affected; more percentages of errors can be caused by previous software development phases like design and requirements as shown in Figure3 (McConnell 2004). Generally, the effort needed to code implementation and construction is about 65% for small software projects, and about 50% for medium software projects. The percentage of coding and construction errors, is increasing from 50% to 75% staring from small, to medium and large projects (McConnell 2004). The density and types of errors and defects will be changed according to size, the quality factors: maintainability, readability, testability also will be affected negatively.

2.3. Logical Error Detection using Object-Oriented Environment:

To avoid logic errors, the researchers in this work provided "an Object Behavior Environment (OBEnvironment)"(Samara 2017) that allow users to discover logic errors, it enforces the proper use of objects depending on their behaviors using software tools such "Xceed Component" that allow software developers to produce better codes, and writing correct statements and syntaxes by C# programming language.

2.4. A framework and methodology for studying the causes of software errors in programming systems (Kirkov and Agre 2010)

The research provides a framework that focuses on the identification and description of the error's cause in terms of constraints of cognitive breakdowns, this framework is built on new and old research of programming, and general studies on the techniques and mechanisms of human errors (Kirkov and Agre 2010).

2.5. Program Slicing

An increase in the number of output variables in the program increases the probability of error occurrence and decreases the performance of the program. Program slicing is one of important techniques used for source code analysis, it helps to break down source code; this issue makes the testing process easier and faster. There are different slice criteria used to identify a block of source code statements to test them separately without upsetting the rest parts of the program. The slicing increases the understandability, readability, and testability factors because it divides the source code into separate slices, then various testers can work on slices at the same time (Sasirekha, Robert, and Hemalatha 2011).

3. Methodology

This part includes some comprehensive discussion about the research methodology of analyzing source code errors, faults, defects, and failures, sampling and data collection.

3.1. Definitions

- Mistake: A user action that leads to producing invalid output.
- Software Error: The variation between software output and the true expected and corrected value (Ieee 1990). It can be produced by the developer mistake during the coding process.
- Defect: It can be called a silent (the killer in some cases, especially in real-time applications) error that causing to failure upon some conditions.
- Fault: it a wrong procedure or incorrect data type definition, also it called bug.
- Failure: It is an invalid result, or breaking and stopping software execution, it can happen if the system has a defect, or fault.
- Logical error: It is an error caused by incorrect logic steps (algorithm) used to solve problem.
- Run time error: It occurs during the run time of the program.
- Syntax error: It is incorrect using of the programming language syntax and it can appear during compilation or interpretation time.

3.2. Debugging

Debugging is very important series of actions and steps taken in order to fix errors that have been detected in testing process. Using source code testing results, programming experiences and knowledge, debuggers can discover the exact place or position of errors and their types, then repair them. Usually debuggers use interactive software tools to provide more information about source code running (Sommerville 2016).

3.3. Syntax and Syntax Errors

The syntax of programming languages is a set of rules and regulations used to write well-structured sentences to represent all parts of the source code including: variable and object declarations, expressions, commands, array, methods and procedures, and every single statement.

Any deviation in the syntax of source code expressions and structure can cause a simple error known as a syntax error; this type of error can be easily discovered during the compilation time. Generally, the syntax error is a grammatical error or mistake. For example missing bracket, or semicolon, or using small character instead of a capital case in some languages (Java).this type errors also known as compilation Errors (Samara 2017).

3.4. Logical Errors:

The software source code written in any computer (programming) language is a formal script constructed according to strict rules, after compilation, it should be free of all syntax errors, however, some errors remain even after the compilation process has completed successfully. These errors are called logical errors and they have many types, so "Logical Errors are errors that remain after all syntax errors have been removed" (Samara 2017). Normally, the compilation process does not discover or identify the presence or existence of the logical errors, this issue leads to unexpected results by the execution of the source code or the software uses this code.

3.5. Run Time Errors

This type of error cannot be discovered by compilation or interpretation, they occur during program run-time when the program is trying to perform a function or an expression that not able to execute, for an example infinite loops and division by zero, the run time errors usually causing system failure (Samara 2017).

3.6. Arithmetic error

Sometimes an arithmetic calculation can produce inaccurate results, causing an error or failure. This type or error called arithmetic error. The possible cases of arithmetic errors must be specified in the requirement specification, and also the actions and error handling against each error should be identified (Sommerville 2016). The Output for the following example is: 0.999999999999999, which is not correct as shown in figure 3.

```
double sum=0.0;
for(int i=0;i<10;i++)
   { sum+=0.1;}
System.out.println(sum);
```

Figure.3: Floating Point Arithmetic Error

3.7. Algorithmic error

This type of error occurs in the following cases: Connecting the branches of the algorithm in incorrect places. Test wrong condition. Not establishing variables and loops with correct initial values. Neglecting or forgetting to test a certain condition, such as the case of division by zero. Comparing variables with inappropriate data types for their definitions. This type of error includes grammatical errors of the programming language used, and these errors usually appear during the linking and translation operations.

3.8. Comments

To increase readability, testing, and maintenance factors, software developers must adhere to the rules for writing source code in accordance with international standards, especially naming the variables, identifiers, and choosing the appropriate data type commensurate with their use. Using comments (inline, and inside, and at the beginning of blocks and loops) correctly and not excessive in the number of lines of comments, because this may lead to opposite results according to the proposed mathematical relationship for calculating the maintenance factor (Al-qutaish and Al-qutaish 2010), which shows the sinusoidal relationship between the average number of comment lines and the size of the source code itself.

$$MI1 = 171 - 3.42 * \ln(aHE) - 0{,}23 * cc \quad (1)$$

$$MI2 = 16.2 * \ln(aLOC) + 50 * \sin(\sqrt{2.4 * aCMp}) \quad (2)$$

$$MI = MI1 - MI2 \quad (3)$$

Equation no (1) shows calculation of Maintainability Index (MI) using Halsted's metrics(Bran 2010), equation no(2) using number of lines of code (LOC) and percentage of comments lines, where (aHE) is the average of software metrics (Halstead's) effort, cc is the cyclomatic complexity, (aLOC) the average of effective source lines of code, and (aCMp) is the average of percentage of comment line (Riaz, Mendes, and Tempero 2009)(Sharawat n.d.).

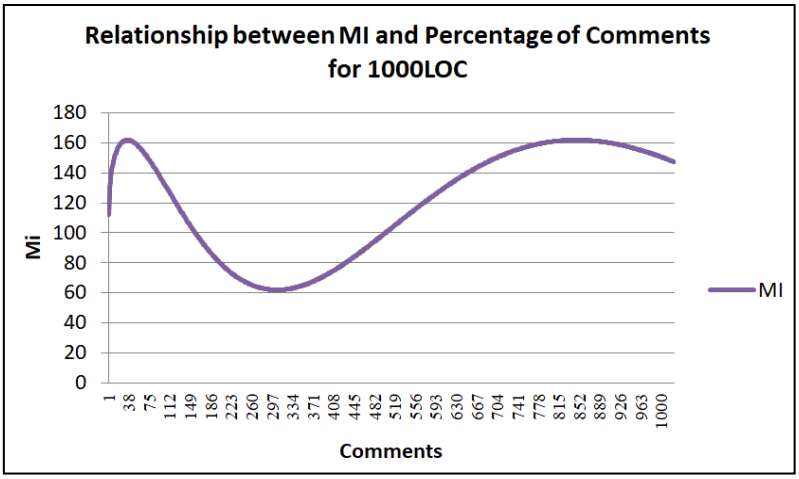

Figure 4: Relationship: MI and % of Comments for 1000LOC

We notice from the figure 4 that increasing the percentage of code documentation increases the value of the maintenance indicator MI, but this increase will stop at a certain limit (the 37% point for a 1000-line code) because it follows sinusoidal behavior. Cyclomatic complexity and Halsted effort are decreasing MI also (see equation no1), all of these issues can make detection errors more difficult.

3.9. Error Handling and Exceptions

When the error occurs during program execution, this moment must be determined, otherwise, the error will turn into a fault, or a failure, this issue can cause to fail the entire software system. All programming languages support what called error handling and Exceptions for example Java, C, and C# are using the try and catch structure. Using exceptions and error handler will give an error indicator which can be used to take the correct decision during program run-time. If the error handling is processing the error in the right way, the programming system (software) can avoid the negative impact on the software behavior. For example division by zero problem can lead to very serious issue especially in real-time software applications. Figure 5 shows an example of using error exception handling to test instance variable of class constructor.

```
public class A
{
  private int No1;
  private int No2;

 public A(){No1=1;No2=1;}
 public A(int No1,int No2){
 if(No1>=8 || No2>=8)
            throw new IllegalArgumentException("Data Input Error");
   this.No1=No1; this.No2=No2;}
 public int sum(){return(No1+No2);}
 public static void main(String[] args)
   {
      A a= new A(Integer.MAX_VALUE,Integer.MAX_VALUE);
      System.out.println(a.sum());
   }
}
```

Figure 5: Using Error Exceptions

3.10. Proposed Source Code Analyzing Model:

The structure of most used systems (automated or semi- automated) for analyzing the software source code could be classified into 4 compound blocks: "model construction, analysis, and pattern recognition algorithms, patterns knowledge, and result representation" (Kirkov and Agre 2010)(Al-qutaish and Al-qutaish

2010)(Binkley and Binkley 2007), and other system such as "Optimization of Software Testing for Discrete Test suite using Genetic Algorithm and Sampling Technique"(PrasadaTripathy and Kanhar 2013). The proposed model uses different method as shown in the figure no 6, and general algorithm depending on the analyzing of operators, operands and relationship between them and data type declaration section as shown below.

3.11. The General Algorithm model:

1) Read source code file
2) Extract Declaration section, and Expressions (Operators and operands): for variable in declaration prepare list of operands and data type used for each one.
3) Error detection:
   a. For each equal (=) operator, check the data type for its left and right operands, If the right side has :
      - a plus (+) or multiplication (*) operator: If all operands from the same data type, overflow error can be happened, it has a defect, the recommendation in this case change data type of left side to include the data type of the right side, or add some restriction on the input data to be included in the needed range, for an example and integer data type included in the Long integer type, or double and float.
      - a division (/) operator: In this case the division by zero problem must be processed.
   b. Calculate the number of output variables, it's recommended to be one for each function, if it exceeds one the possibility of decreasing performance and appearing defects will be high (Sasirekha et al. 2011).
4) Calculate the number of inline documentation.
5) Calculating Cyclomatic Complexity (CC) according to the number of loops and conditional statements.
6) Prepare recommendation and suggestion to correct source code: the recommendations will be varying depending on the test program; the suggestion and recommendation are saved in the data base.

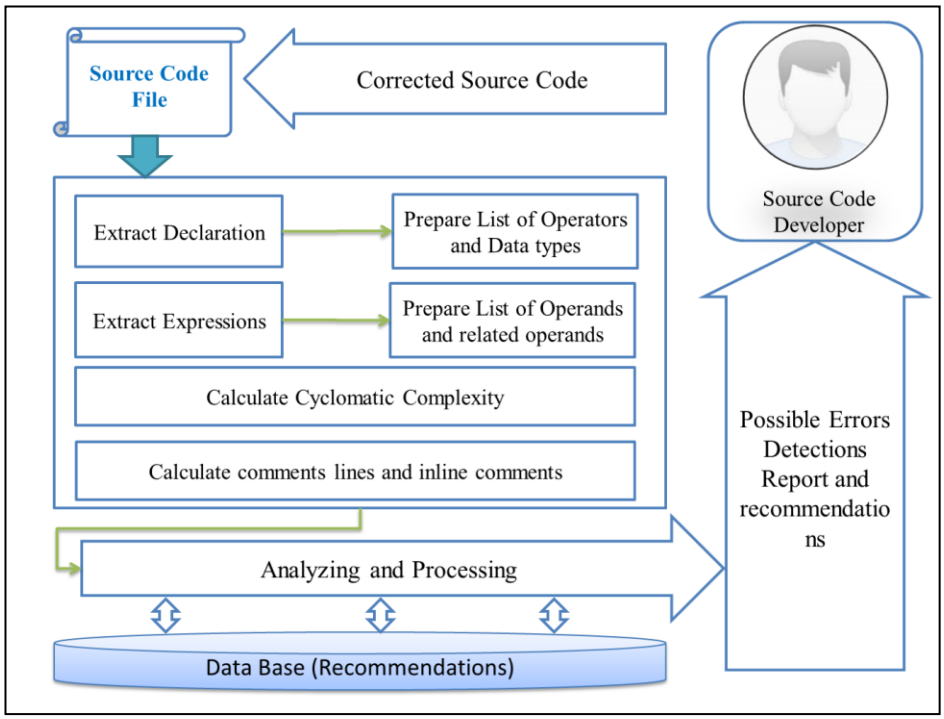

Figure 6: A General Structure of Analyzing Model

4. Results

In this section the simple example shown in Figure 7 will be checked manually using Test Cases listed in Table 1, and it will be tested using the proposed model.

```
public class Test_A{
public static void main(String[] args)
{
   int No1,No2,Result; /* Declaration Line */
   /* Assigning integer numbers to the variable No1, and No2 */
   No1=9;
   No2=5;
   Result= No1+No2; /* Calcuation of two integer Numbers */
   System.out.println(Result); /* Dispaying result */
   }
 }
```

Figure7: Java simple example

4.1. Manual Testing:

The result of manual testing is shown in the table1

Table 1: Test Cases and result of manual testing

No	Input Data		Output (Result)		remarks
	No1	No2	Expected	Calculated	
1	1	1	2	2	Correct
2	10	7	15	15	Correct
3	010	5	15	13	First parameter (a) in the form 010: was counted as Octal, causing to result in Octal also.
4	10	09	19	error	Compilation error: integer number too large: 09
5	010	010	20	16	Octal
6	010	-5	5	3	Octal
7	IMV	IMV	-1	-1	correct
8	0	0	0	0	correct
9	3	5	8	8	correct
10	10	-7	3	3	correct
11	-15	8	-7	-7	correct
12	IMV	0	2147483647	2147483647	correct
13	IMV	1	-2147483648	2147483648	X
14	IMV	IMV	-2	4294967294	X
15	IMV	0	-2147483648	-2147483648	correct
16	IMV	1	-2147483648	-2147483647	correct
17	IMV	IMV	0	0	correct

Where the Integer.MAX_VALUE= 2147483647, it will be used as IMV in the table no 2, and Integer.MIN_VALUE= -2147483648.

4.2. Automated testing

Extract Declaration section, and Expressions (Operators and operands): for variable in declaration prepare list of operands and data type used for each one.

Table 2: Variables and data type

Variable Name	Data type
No1	Integer
No2	Integer
Result	Integer

Table 3: Operators and related operands

Operator	Operands
=	Result, No1, and No2
+	No1, No1

By comparing the data type for the output operand (Result) in the left side of the expression (Result= No1 + No2), we found that the left side is consisting of two operands and relation operator plus (+), so the balance of the data type must be in the integer range, in this case the list of recommendations will be generated to direct the user of this program to avoid expected errors including overflow error.

Table 4: List of expected errors and recommendations

Expected Error Type	Recommendation and suggestions
Overflow	Add direction message to inform user keeping data in the integer range. Enter data during run-time using Scanner class. Avoid entering data for both variables (No1, and No2) greater than absolute of half maximum of integer. Finally, try to use function with two parameters of double data type test accepted inputs.
Non dismal calculations (Octal: an example)	Avoid using (0) at beginning of entered numbers, by adding an expression to that. In case you are using your program to calculate Octal data numbering, try to enter numbers greater than 07.
Accuracy	In case of using loops with non-integer (double or float) counters, tack care of final result it will be not correct.
Lines of code	It's good in your program but the must be less than total number of the effective lines of code according to the quality standards.
Summary	It's better to use Object oriented programming to solve the above mentioned notes.

5. Conclusion

This research is completely about analyzing source code, the nature and difference between logical and syntax errors were discussed. All types of errors in the implementation phase can lead to big complicated problems such cracks and loopholes, so it is very important for the software system to check and validate inputs and reject the problematic one; the attackers can use weaknesses to exploit it for accessing the system. This research offers the developers extra and new validation techniques and activity to achieve their goals in reducing number of errors and defects. More Line of Codes (LOC) means error density, more difficulty and complexity, it is better to keep your code modular and break down big source code procedures into smaller module to achieve required standards (50 LOC/module (Al-qutaish and Al-qutaish 2010)). Increasing Cyclomatic Complexity, Halsead's effort and LOC lead to more difficulty and more non-expected defects and errors.

References

[1]. Al-qutaish, Rafa E., and Rafa E. Al-qutaish. 2010. "Quality Models in Software Engineering Literature: An Analytical and Comparative Study." *Journal of American Science* 6(3):166–75.

[2]. Bassman, M. J., McGarry, F., & Pajerski, R. 1994. *Software Measurement Guidebook*. National Aeronautics and Space Administration, Goddard Space Flight Center ; National Technical Information Service, distributor,.

[3]. Binkley, David, and David Binkley. 2007. "Source Code Analysis: A Road Map." *IN FUTURE OF SOFTWARE ENGINEERING* 104--119.

[4]. Bran, Alain. 2010. "Halstead's Metrics: Analysis of Their Designs." Pp. 145–59 in *Software Metrics and Software Metrology*. Hoboken, NJ, USA: John Wiley & Sons, Inc.

[5]. Huang, J. C. 2009. *Software Error Detection through Testing and Analysis*. Hoboken, NJ, USA: John Wiley & Sons, Inc.

[6]. Ieee. 1990. *IEEE Standard Glossary of Software Engineering Terminology*. Vol. 121990.

[7]. Kirkov, Radoslav, and Gennady Agre. 2010. *Source Code Analysis-An Overview*. Vol. 10.

[8]. Laporte, Claude Y., and Alain April. 2017. "Software Quality Assurance: Laporte, Claude Y., April, Alain: 9781118501825." *The IEEE Computer Society*. Retrieved March 23, 2021

[9]. Lester, Cynthia Y. 2008. "An Innovative Approach to Teaching an Undergraduate Software Engineering Course." Pp. 301–6 in *Proceedings - The 3rd International Conference on Software Engineering Advances, ICSEA 2008, Includes ENTISY 2008: International Workshop on Enterprise Information Systems*. IEEE.

[10]. Maxim, Bruce R., and Marouane Kessentini. 2016. "An Introduction to Modern Software Quality

Assurance." Pp. 19–46 in *Software Quality Assurance: In Large Scale and Complex Software-intensive Systems*. Elsevier Inc.

[11]. McConnell, S. 2004. "Code Complete: A Practical Handbook of Software Construction, Second Edition: McConnell, Steve: 0790145196705: Amazon.Com: Books." *Microsoft Press*. Retrieved March 23, 2021.

[12]. PrasadaTripathy, Siba, and Devanand Kanhar. 2013. "Optimization of Software Testing for Discrete Testsuite Using Genetic Algorithm and Sampling Technique." *International Journal of Computer Applications* 63(7):1–5.

[13]. Riaz, Mehwish, Emilia Mendes, and Ewan Tempero. 2009. "A Systematic Review of Software Maintainability Prediction and Metrics." Pp. 367–77 in *2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009*. IEEE.

[14]. Samara, G. 2017. "A Practical Approach for Detecting Logical Error in Object Oriented Environment." *World of Computer Science and Information Technology Journal* 7(2):10–19.

[15]. Sasirekha, N., A. Edwin Robert, and M. Hemalatha. 2011. "PROGRAM SLICING TECHNIQUES AND ITS APPLICATIONS." *International Journal of Software Engineering & Applications (IJSEA)* 2(3).

[16]. Sharawat, Mr Sandeep. n.d. *Software Maintainability Prediction Using Neural Networks*. Vol. 2.

[17]. Sommerville, I. 2016. *Software Engineering*. 10th ed. Harlow: Pearson Education.