Asian Journal of Information Technology 16 (6): 503-510, 2017
ISSN: 1682-3915
© Medwell Journals, 2017

An Interface Maintainability Measure for Component-Based Software Systems

[1]N. Baskar, [1]A.V. Ramani and [2]C.Chandrasekar
Department of Computer Science, SRMV College of Arts and Science, 641020 Coimbatore, India
[2]Department of Computer Science, Government Arts College, 642 126 Udumalpet, India

Abstract: Now a days, component-based systems are being used increasingly which have made changes to build and maintain the systems. In the present technical researcher, each day several numbers of software are built up and offered in the market but software maintenance measuring is still a big challenge. Maintenance becomes very difficult if the components are integrated into the system due to source code may be partial or invisible completely. Several models and metrics are implemented for the software but they did not meet the need of the component-based software. Among those set of metrics, Response for a Class (RFC) is one of the metrics which is nothing but the number of methods that can be potentially executed in response to a message received by an object of a class. In RFC, each function call statement value is considered to be 1. The cognitive feature is not included in RFC metric which is felt as a major negative aspect of this metric. So, this researcher proposed a metric for measuring the interface maintenance for the component based software system. The proposed metric for the maintainability is Cognitive Weighted Response for a Class (CWRFC) metric. The proposed metric is applied to the computer classification and acquired better results which will help not only for low maintenance of the component based system but also to reduce the complexity efforts.

Key words: Component-based software systems, maintainability measure, cognitive weighted response for a class, interface, class, component

INTRODUCTION

Maintaining the software is the very expensive and resource requiring phase in the process of software development. The maintainability concept is the very important in the component based system and it is the ability of the software which can be modified. But the maintainability of the component system is a difficult task because the personnel of the maintenance will not have the access for the component source code which is to be modified. So, software maintainability research includes validating the maintainability predictors which is based on the measurable factors, it would have the behaviour on the activity of software maintenance.

Component Based Software Development (CBSD) (Marco *et al*., 2015) is a very important aspect in the current model which is expected to be the head for the recent researchers for constructing complex and large software systems. The main aim of this research is to reduce the effort; cost and time for development of using the reusable component and improving the productivity, quality and software maintainability. Already incorporated components of the software using reusable components will provide these advantages mainly. The component of the software is a self-contained software piece which will provide the functionality clearly besides that it also has open interfaces and provides the services of plug and play according to Thomas. It will represent an element of reusable software which are file, function, class, module or subsystem. Number of software metrics related to software maintainability, quality assurance and complexity which has been implemented in the past and are still being proposed.

Interfaces are the main tool for information hiding in software systems that have service contracts between users and providers of behavior. Because of this contract role, interfaces are different from classes. It should be more stable across the evolution of a software system to reduce the effort in order to understand and maintain a software system. Designing an interface is a sensitive task with a large influence on the rest of the system. Likewise, during the evolution of a software system, the design of interfaces must be assessed precisely to control the impact of any required change. However, as software remains changes in runtime with the modification, adding and deleting of new classes and services, the software gradually drifts and loses quality by Sreekumar and Sivabalan (2015).

To improve the quality of the software there has been recently an important progress in the area of automatic software refactoring and optimization of code quality. Most of the existing approaches in that field are mainly

Corresponding Author: N. Baskar, Department of Computer Science, SRMV College of Arts and Science, 641020 Coimbatore, India

related with code metrics such as metrics defined and predefined bad smells in source code. Due to this none of those approaches is taken into consideration, interfaces do not contain any logic such as method invocations, implementations or attributes. In literature, few recent works focused to address the particularities of interfaces. The well-known interface design principles such as dependency inversion "program to an interface not an implementation" or interface segregation "do not design fat interfaces" which are designed by Sreekumar and Sivabalan (2015). However, in existing design patterns, code smells and metrics revolve around classes without concentrating on the specifics of interfaces. This concept reveals that there are only few publications focused on the interface design quality. There are no tools that help estimate the interface design quality and detect design anomalies in software interfaces.

Measurement and metrics: Now a days, software engineering plays a major role in the world. As computer software has grown, the software developers want to attempt continually to develop new technologies. In these developed technologies some of them focused on object-oriented technologies. In this researcher, class inheritances are differentiated with interface class diagrams, through maintainability measures. Without measurement it is impossible to measure the quality and the maintainability of software to detect problems before it is released. In this case, measurement is very important in managing the software projects. Metrics are used as a powerful tool in software research, maintenance and estimating cost, effort, maintenance, complexity, quality and to control, etc. Metrics serves as an early warning tool for potential problems happening in the software development. Each metric must be defined as a complete and well-designed quality improvement paradigm. Maintainability metrics of UML design:

- Association between a class and an interface
- Implementation of an interface by a class
- Dependency between a class and an interface
- Aggregation of an interface in a class (containers)

Figure 1 shows the layered approach of maintainability model for the object-oriented software systems. This approach includes three levels. The first layer is corresponding to the criterion the second layer is corresponding to sub-criteria and the third layer is corresponding to metrics impacting subcriteria. The layered approach deals with program information from bottom to top which starts from simple and measurable data and analyzes the quality of software maintainability.

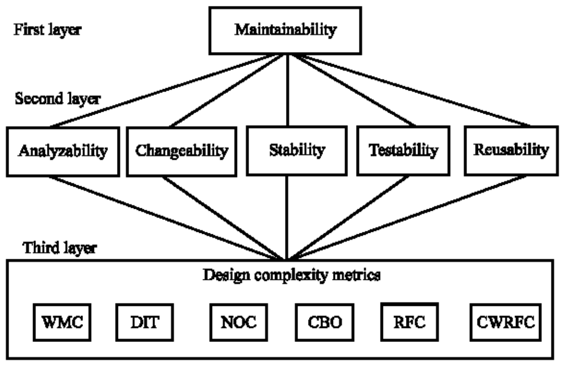

Fig. 1: Proposed maintainability hierarchy

In this research, interface maintenance metric is designed for computing the important feature of the maintainability of component-based system. The designed metric is Cognitive Weighted Response for a Class metric in CWRFC, the work proposed by Aloysius and Arockiam (2012). The cognitive weights are assigned to function call statements which are based on the effort required for understanding their function call type due to the message passed by the class object. The proposed metric has been proved to be a better measure of cognitive of class with function call statement through the case studies and experiments.

Literature review: By Sundararajan is implementing the method of model, driven for generating the specific code for platform of a user interface. Ulkuniemi *et al*. (2015) examined about the marketing practices of the component based system that is how a single buying company makes an attempt for shaping the market. Based on this, five types are identified for market shaping actions. Pham *et al*. (2015) provides a modeling scheme for reliability which are automatically transformed using the prediction tool of reliability into Markov modes for predicting the reliability and analysis of sensitivity. Wei and Guo (2015) implemented a method for predicting the QoS of the software system which will be affected due to the dynamic reconfiguration and showed the performance of the existing method using three aspects. Marco *et al*. (2015) developed a method by improving the architectural approach for detecting and recovering the incompatible interactions by manufacturing the exact coordinator.

Karambir and Suri (2015) aim at finding the existing component selection, characteristics, repository of components, testing and challenges in the science of CBSE. The systematic literature survey was based on 51 international journals collected from multiple-stage selection process. Fabian *et al*. (2015) offers a comparison of an in-depth and quantitative evaluation for the representative model transformations. The goal by Ganesh and Raj (2015) is to inventing new software metric for SQA for analyzing the brilliance of the software. Certifying the proposed metric as valid should also be considered as equal to the invention, since it describes how good the proposed metric is how it could be practically implemented and how useful it is for the technical fraternity and so on. The main aim by Tiwari and Chakraborty (2015) is to identify the different aspects of the quality of the software components. They also established the relationship among the components quality characteristics and sub-characteristics.

By using the reusable software product effectively there will be increase in the reliability, maintainability and productivity. But, there are some important challenges in reusing and determining the exact components in the software development process. To rectify these challenges, engineers should apply the efficient methods for identifying a high potential and quality reusable software component which is given by Sridhar (2015). Ana *et al*. (2015) review and critically analyze the developments in this domain by considering 26 of the most research papers addressing object-oriented coupling. Metrics are defined by Chhillar *et al.* (2015) to depict member access control mechanism and then employed in the class hierarchy. The proposed metrics provide a way to understand and implement these concepts in research and development of the software using object-oriented approach. Arbi (2015) proposed a new information theoretic measure of software complexity that, unlike previous measures, captures the volume of design information in software modules. Noor (2015) gives a brief review of cognitive computing and some of the cognitive engineering systems activities. A model clone is a set of similar or identical fragments in a model of the system. Aastha *et al*. (2015) provides about detecting clones utilizing the model architectures using CK metric suite.

There are several approaches and metrics proposed for analyzing the maintainability of component based software system. But, there is no effective method for measuring the maintainability among the interfaces in the component based system. So, there is need for new metric for measuring the maintainability effectively which will reduce the effort, cost and time during the development process.

MATERIALS AND METHODS

Proposed metric for interface maintenabilty: Component-based software development is used in the industries as new efficient model of development. It highlights the software system construction and design using reusable components. It will minimize the cost of development and also will improve the entire system reliability using components. The main benefits are high quality solutions and simultaneously it can also be used for interfaces measuring. The cognitive weighted response for a class metric is used for maintainability. This metric is used to check the maintainability very efficiently and it is very simple. If the number of method is larger that can be invoked from class/interface through message. Then the maintainability of the class/interface is increased. Additionally, inherited methods are counted but overridden methods are not because only one method of a particular signature will always be available to an object of a given class. For possible response, a bad value will help in the allocating the appropriate testing time.

Two UML class inheritance diagrams are taken and all the above said metric is applied to measure complexity. The two diagrams are introduced with maximum possibility of interfaces and metrics which are used to measure the maintainability. Both inheritance and interface diagrams maintainability measures are compared. First UML class diagram has been taken as computer classification.

Response for a Class (RFC): The RFC is used to count all the methods which can be called in by the response to a message to class object or by other methods in that class. Member method in the class and member methods of other classes are equally counted. This metric will glance at the combination of the class complexity through the number of methods and communication amount with other classes. If the number of methods invoked is larger, then the class complexity will be high. Then the debugging and the testing of the class will also complicate because it requires a greater understanding level of the tester part.

The response set of a class is defined as set of methods that can be potentially executed in response to a message received by an object of that class:

$$\text{RFC} = \text{Number of elements in RS} \tag{1}$$

Where RS is the response set for the class. It can be expressed as, RS = Union of methods in the class and the inherited methods from in the class:

$$\text{RS} = \text{M U IM} \tag{2}$$

Where:
IM = The set of inherited methods
M = The set of methods in the class

In order to calculate response for a class, one must do the following:

- Classes are consulted at the first
- The first message is taken and the number of method is counted which can be executed in message response
- This number is recorded and the process is repeated for all methods
- After finishing, the largest number is entered

Cognitive weighted response for class and interface: A metric called Cognitive Weighted Response for a Class (CWRFC) is firstly implemented by Aloysius and Arockiam (2012). In CWRFC, the cognitive weights will be assigned for the function call statement which is based on the effort required for understanding their type of function calls due to message passed by an object of that class. CWRFC is used to calculate the maintainability of the class using the Response Set Complexity (RSC). If there are m numbers of response sets in a class, then the CWRFC of that class can be calculated by using the Eq. 3:

$$\text{CWRFC} = \sum_{j=1}^{m} \text{RSC}_j \tag{3}$$

where, RSC is the Response Set Complexity which can be calculated by adding the set of all Methods (M) in a class and set of methods (R) called by any of those methods as given in Eq. 4:

$$\text{RSC} = M_n + \forall_i R_i \tag{4}$$

The applications of cognitive informatics in cognitive science will cover a wide range of cognitive phenomena at the sensation, subconscious, Meta cognitive and higher cognitive function levels.

RESULTS AND DISCUSSION

In order to compute the maintainability of component-based systems through proposed measure, firstly class inheritance diagram are developed for computer classifications and then proposed metric is applied. The metric discussed above are applied for interface UML diagrams. The first study, considered is a computer classification using class inheritance diagram which is represented in Fig. 2.

The applicability of the proposed metric has been checked by applying it to an Object-oriented programming that its class hierarchy is given in Fig. 2. This example processes a computer database hierarchy. It has one main class computer and two subclasses, hardware and software. The class hardware has again two subclasses, desktop and notebook. We demonstrate how we can calculate the class and interface CWRFC for an object-oriented system. The CWRFC values corresponding to each class of Fig. 2 and 3 is summarized in Table 1.

The class inheritance Fig. 2 is introduced with possible number of interfaces and is represented in Fig. 3.

Response for a class: RFC is calculated for the class classification using class inheritance and interfaces as.

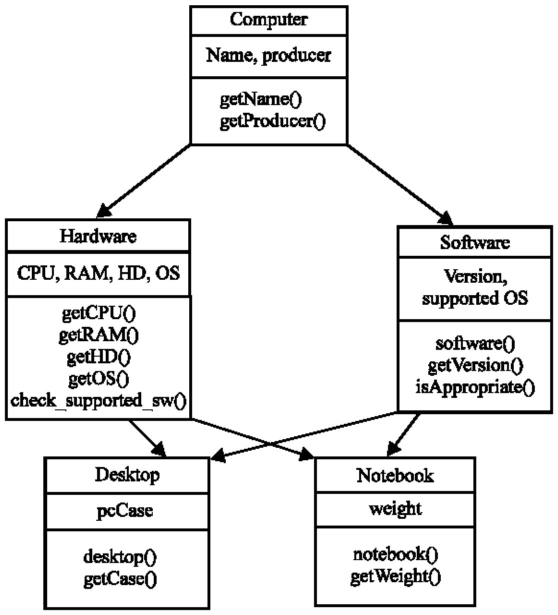

Fig. 2: Computer classification using class inheritance

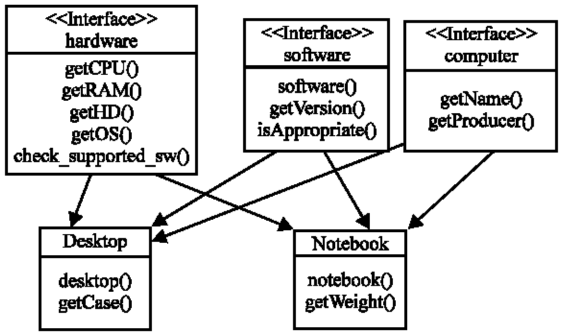

Fig. 3: Computer classification using interfaces

Table 1: Response for a class value

Metric	Computer	Hardware	Software	Desktop	Notebook
RFC for class	2	7	5	12	12
RFC for interface	2	5	3	12	12

Class inheritance

Case 1; computer: The computer has 2 methods so, RFC is calculated as:

$$\text{RFC} = 2\left[\text{getName(), getProducer()}\right]$$

Case 2; hardware: The hardware has 7 method so RFC is calculated as:

$$\text{RFC} = 7\begin{bmatrix}\text{getName( ), getProducer( ), getCPU( ),}\\ \text{getRAM(), getHD(), getOS( ),}\\ \text{check\_supported\_sw()}\end{bmatrix}$$

Case 3; software: The software has 5 method so RFC is calculated as:

$$\text{RFC} = 5\begin{bmatrix}\text{getName( ), getProducer( ), software( ),}\\ \text{getVersion( ) isAppropriate( )}\end{bmatrix}$$

Case 4; desktop: The desktop has 12 methods, so, RFC is calculated as:

$$\text{RFC} = 12\begin{bmatrix}\text{desktop( ), getCase( ), getName( ),}\\ \text{getProducer(), getCPU(), getRAM(),}\\ \text{getHD(), getOS(), check\_supported\_sw(),}\\ \text{software(), getVersion() isAppropriate( )}\end{bmatrix}$$

Case 5; notebook: The notebook has 12 methods so RFC is calculated as:

$$\text{RFC} = 12\begin{bmatrix}\text{notebook( ), getWeight( ), getName( ),}\\ \text{getProducer(), getCPU( ), getRAM( ),}\\ \text{getHD(), getOS(), check\_supported\_sw(),}\\ \text{software(), getVersion() isAppropriate( )}\end{bmatrix}$$

Interfaces

Case 1; computer: The computer has 2 methods, so, RFC is calculated as:

$$\text{RFC} = 2\left[\text{getName( ), getProducer( )}\right]$$

Case 2; hardware: The hardware has 5 methods, so, RFC is calculated as:

$$\text{RFC} = 5\begin{bmatrix}\text{getCPU( ), getRAM( ), getHD( ),}\\ \text{getOS(), check\_supported\_sw()}\end{bmatrix}$$

Case 3; software: The software has 3 methods, so, RFC is calculated as:

$$\text{RFC} = 3\left[\text{software(), getVersion() isAppropriate()}\right]$$

Case 4; desktop: The desktop has 12 methods so RFC is calculated as:

$$\text{RFC} = 12\begin{bmatrix}\text{desktop( ), getCase( ), getName( ),}\\ \text{getProducer(), getCPU( ), getRAM( ),}\\ \text{getHD( ), getOS( ), check\_supported\_sw( ),}\\ \text{software(), getVersion() isAppropriate()}\end{bmatrix}$$

Case 5; notebook: The notebook has 12 methods so RFC is calculated as:

$$\text{RFC} = 12\begin{bmatrix}\text{notebook( ), getWeight( ), getName( ),}\\ \text{getProducer( ), getCPU( ), getRAM( ),}\\ \text{getHD(), getOS(), check\_supported\_sw(),}\\ \text{software(), getVersion( ) is appropriate()}\end{bmatrix}$$

Based on the values obtained, RFC is shown in Table 1. Figure 2 and 3 showed the metrics are measured and tabulated in Table 1. Response for class correlates with the defect densities. However, the research of CK is not without criticisms. Object-oriented designs are relatively richer in information. Therefore, metrics if properly defined can take advantage of that information available at any early stage in the life cycle. Unfortunately, most of the prior researchers do not exploit this additional information. In order to overcome the above-said drawbacks of object-oriented design metrics, it is proposed to opt for cognitive complexity metric suite.

Cognitive weighted response for a class: In this research, CWRFC is calculated for the computer classification using class and interfaces as.

Class inheritance

Case 1; computer: The computer interface has 2 methods, so CWRFC can be calculated as:

$$\text{CWRFC} = \sum_{j=1}^{2} \text{RSC}_j = \text{RSC}_1 + \text{RSC}_2 = (M_n + R_i) + (M_n + R_i)$$

Case 2; hardware: The hardware interface has 7 methods, so, CWRFC is calculated as:

$$\begin{aligned}\text{CWRFC} = \sum_{j=1}^{7} \text{RSC}_j &= \text{RSC}_1 + \text{RSC}_2 + \text{RSC}_3 + \text{RSC}_4 + \\ &\quad \text{RSC}_5 + \text{RSC}_6 + \text{RSC}_7 \\ &= (M_n + R_i) + (M_n + R_i) + (M_n + R_i) + \\ &\quad (M_n + R_i) + (M_n + R_i) + (M_n + R_i) + (M_n + R_i) \\ &= (7+6) + (7+6) + (7+6) + (7+6) + (7+6) + \\ &\quad (7+6) + (7+6) = 91\end{aligned}$$

Case 3; software: The software interface has 5 methods, so, CWRFC is calculated as:

$$\begin{aligned}\text{CWRFC} &= \sum_{j=1}^{5}\text{RSC}_j = \text{RSC}_1+\text{RSC}_2+\text{RSC}_3+\text{RSC}_4+\text{RSC}_5\\ &= (M_n+R_i)+(M_n+R_i)+(M_n+R_i)+\\ &\quad (M_n+R_i)+(M_n+R_i)\\ &= (5+4)+(5+4)+(5+4)+(5+4)+(5+4) = 45\end{aligned}$$

Case 4; desktop: The desktop has 12 methods, so CWRFC is calculated as:

$$\begin{aligned}\text{CWRFC} &= \sum_{j=1}^{12}\text{RSC}_j = \text{RSC}_1+\text{RSC}_2+\text{RSC}_3+\text{RSC}_4+\text{RSC}_5+\text{RSC}_6+\\ &\quad \text{RSC}_7+\text{RSC}_8+\text{RSC}_9+\text{RSC}_{10}+\text{RSC}_{11}+\text{RSC}_{12}\\ &= (M_n+R_i)+(M_n+R_i)+(M_n+R_i)+(M_n+R_i)+\\ &\quad (M_n+R_i)+(M_n+R_i)+(M_n+R_i)+(M_n+R_i)+\\ &\quad (M_n+R_i)+(M_n+R_i)+(M_n+R_i)+(M_n+R_i)\\ &= (12+11)+(12+11)+(12+11)+(12+11)+\\ &\quad (12+11)+(12+11)+(12+11)+(12+11)+(12+11)+\\ &\quad (12+11)+(12+11)+(12+11) = 276\end{aligned}$$

Case 5; notebook: The notebook has 12 methods so CWRFC is calculated as:

$$\begin{aligned}\text{CWRFC} &= \sum_{j=1}^{12}\text{RSC}_j = \text{RSC}_1+\text{RSC}_2+\text{RSC}_3+\text{RSC}_4+\text{RSC}_5+\text{RSC}_6+\\ &\quad \text{RSC}_7+\text{RSC}_8+\text{RSC}_9+\text{RSC}_{10}+\text{RSC}_{11}+\text{RSC}_{12}\\ &= (M_n+R_i)+(M_n+R_i)+(M_n+R_i)+(M_n+R_i)+\\ &\quad (M_n+R_i)+(M_n+R_i)+(M_n+R_i)+(M_n+R_i)+\\ &\quad (M_n+R_i)+(M_n+R_i)+(M_n+R_i)+(M_n+R_i)\\ &= (12+11)+(12+11)+(12+11)+(12+11)+\\ &\quad (12+11)+(12+11)+(12+11)+(12+11)+(12+11)+\\ &\quad (12+11)+(12+11)+(12+11) = 276\end{aligned}$$

A detailed description provides useful information about the metric. If the example in Fig. 2 is considered, we can find that desktop and notebook classes are on the same level and inherit the property from hardware and Software. Hardware and software will inherit the property from computer. So, the desktop and notebook will inherit the methods from computer, hardware and software. The CWRFC value will be calculated based on the inheritance and obtained as 276 for both desktop and notebook.

Hardware and software are subclasses of computer and inherits the properties from computer therefore, CWRFC values can be calculated based on the computer and obtains 91 and 45. It is because of the hardware and software classes are on the same level and both inherit from the class computer. This example shows the usage of inheritance property of the classes in calculations.

Interfaces

Case 1; computer: The computer interface has 2 methods so CWRFC can be calculated as:

$$\text{CWRFC} = \sum_{j=1}^{2}\text{RSC}_j = \text{RSC}_1+\text{RSC}_2 = (2+1)+(2+1) = 6$$

Case 2; hardware: The hardware interface has 5 methods so CWRFC is calculated as:

$$\begin{aligned}\text{CWRFC} &= \sum_{j=1}^{5}\text{RSC}_j = \text{RSC}_1+\text{RSC}_2+\text{RSC}_3+\text{RSC}_4+\text{RSC}_5\\ &= (5+4)+(5+4)+(5+4)+(5+4)+(5+4) = 45\end{aligned}$$

Case 3; software: The software interface has 3 methods, so, CWRFC is calculated as:

$$\begin{aligned}\text{CWRFC} &= \sum_{j=1}^{3}\text{RSC}_j = \text{RSC}_1+\text{RSC}_2+\text{RSC}_3\\ &= (3+2)+(3+2)+(3+2) = 15\end{aligned}$$

Case 4; desktop: The desktop has 12 methods so CWRFC is calculated as:

$$\begin{aligned}\text{CWRFC} &= \sum_{j=1}^{12}\text{RSC}_j = \text{RSC}_1+\text{RSC}_2+\text{RSC}_3+\text{RSC}_4+\text{RSC}_5+\text{RSC}_6+\\ &\quad \text{RSC}_7+\text{RSC}_8+\text{RSC}_9+\text{RSC}_{10}+\text{RSC}_{11}+\text{RSC}_{12}\\ &= (12+6)+(12+6)+(12+6)+(12+6)+(12+6)+\\ &\quad (12+4)+(12+4)+(12+4)+(12+3)+(12+3)+\\ &\quad (12+11)+(12+11) = 214\end{aligned}$$

Case 5; notebook: The notebook has 12 methods so CWRFC is calculated as:

$$\begin{aligned}\text{CWRFC} &= \sum_{j=1}^{12}\text{RSC}_j = \text{RSC}_1+\text{RSC}_2+\text{RSC}_3+\text{RSC}_4+\text{RSC}_5+\text{RSC}_6+\\ &\quad \text{RSC}_7+\text{RSC}_8+\text{RSC}_9+\text{RSC}_{10}+\text{RSC}_{11}+\text{RSC}_{12}\\ &= (12+6)+(12+6)+(12+6)+(12+6)+(12+6)+\\ &\quad (12+4)+(12+4)+(12+4)+(12+3)+(12+3)+\\ &\quad (12+11)+(12+11) = 214\end{aligned}$$

Table 2: Cognitive weighted response for a class value

Metric	Computer	Hardware	Software	Desktop	Notebook
CWRFC for class	6	91	45	276	276
CWRFC for interface	6	45	15	214	214

Based on the values obtained before, CWRFC is tabulated as Table 2 as follows. Figure 2 and 3 shows the, metrics are measured and tabulated in Table 2. CWRFC value for interface program obtained value for class attributes is shown. And it is more efficient and accurate one. The results show that the effect of this parameter on maintainability of a component-based system is quite significant. The computer hardware software desktop notebook class is partitioned into two sub classes and their corresponding CWRFC values are 6, 45, 15, 214 and 214 as show in Table 2. Introduction of interfaces in object-oriented programming in possible places is better for good quality and high reliable software.

CONCLUSION

A cognitive maintenance metric for the component-based software systems has been introduced in this research. The main motive behind introducing a new metric is to calculate the cognitive maintainability for the internal architecture by considering the unique feature of the program. This method can also be used to evaluate the design efficiency and therefore it can be applied at the initial phase of software development process. A better design will decrease the efforts of maintainability in the final stage. So, our proposed metric will provide the effective information about the software system maintainability. This metric is evaluated for class and interface program for simple computer classification in this work. It is clear that the proposed metric works efficiently for the interfaces in the component-based software. The total values are reduced for both examples of object-oriented interfaces compared to object-oriented class inheritance concepts. Interface concept has shown better performance compared to inheritance concept in object-oriented programming. Software reliability will increase with lower software maintainability.

REFERENCES

Aastha, S., V. Sharma and J.M.I.T. Radaur, 2015. Detecting model clones using CK metrics suite. Intl. J. Eng. Res. Gen. Sci., 3: 1605-1612.

Aloysius, A. and L. Arockiam, 2012. Cognitive weighted response for a class: A new metric for measuring cognitive complexity of OO systems. Intl. J. Adv. Res. Comput. Sci., Vol. 3.

Ana, N., H. Lichter and Y. Xu, 2015. Evolution of object oriented coupling metrics: A sampling of 25 years of research. Proceedings of the 2nd International Workshop on Software Architecture and Metrics, May 16-24, 2015, IEEE, Florence, Italy, pp: 48-54.

Arbi, G., 2015. A theory of software complexity. Proceedings of the 4th SEMAT Workshop on General Theory of Software Engineering, May 18-18, 2015, IEEE Press, Florence, Italy, ISBN:978-1-4673-7053-0, pp: 29-32.

Chhillar, R.S., P. Kajla, U. Chhillar and N. Kumar, 2015. An access control metric suite for class hierarchy of object-oriented software systems. Intl. J. Comput. Commun. Eng., 4: 61-65.

Fabian, B., P. Meier, S. Becker, A. Koziolek and H. Koziolek *et al*., 2015. Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures. IEEE. Trans. Software Eng., 41: 157-175.

Ganesh, S.H. and H.V. Raj, 2015. Performance based analysis on MALCOM: A software metric. Proceedings of the International Conference on Circuit, Power and Computing Technologies (ICCPCT), March 19-20, 2015, IEEE, Nagercoil, India, ISBN:978-1-4799-7075-9, pp: 1-5.

Karambir, S. and P.K. Suri, 2015. Technical review: Inheritance of component based software engineering. Intl. J. Adv. Res. Comput. Sci., Vol. 6.

Marco, A., P. Inverardi and M. Tivoli, 2015. Synthesis of correct adaptors for protocol enhancement in component-based systems. Master Thesis, Cornell University, Ithaca, New York.

Noor, A.K., 2015. Potential of cognitive computing and cognitive systems. Open Eng., 5: 75-88.

Pham, T.T., X. Defago and Q.T. Huynh, 2015. Reliability prediction for component based software systems: Dealing with concurrent and propagating errors. Sci. Comput. Program., 97: 426-457.

Sreekumar, R.A. and R.V. Sivabalan, 2015. A survival study of object oriented principles on software project development. Proceedings of the Global Conference on Communication Technologies (GCCT), April 23-24, 2015, IEEE, Thuckalay, India, ISBN:978-1-4799-8554-8, pp: 307-310.

Sridhar, S., 2015. A review on reuse of software components for sustainable solutions in development process. Intl. J. Innov. Technol. Res., 3: 1998-2001.

Tiwari, A. and P.S. Chakraborty, 2015. Software component quality characteristics model for component based software engineering. Proceedings of the IEEE International Conference on the Computational Intelligence & Communication Technology (CICT), Feburary 13-14, 2015, IEEE, Ghaziabad, India, ISBN:978-1 4799-6024-8, pp: 47-51.

Ulkuniemi, P., L. Araujo and J. Tahtinen, 2015. Purchasing as market-shaping: The case of component-based software engineering. Ind. Marketing Manage., 44: 54-62.

Wei, L. and W. Guo, 2015. QoS prediction for dynamic reconfiguration of component based software systems. J. Syst. Software, 102: 12-34.