Journal of Information Engineering and Applications www.iiste.org
ISSN 2224-5782 (print) ISSN 2225-0506 (online) DOI: 10.7176/JIEA
Vol.9, No.4, 2019

A Metrics-based Framework for Estimating the Maintainability of Object-Oriented Software

Elijah Migwi Macharia* Prof. Waweru Mwangi Dr. Michael Kimwele
School of Computing and IT, Jomo Kenyatta University of Agriculture and Technology, P O Box 62000-00200
Nairobi Kenya

Abstract
Time, effort and money required in maintaining software has always been considered greater than its development time. Also, its ambiguity in forecast at early stage of software development makes the process more complicated. The early estimation of maintainability will significantly help software designers to adjust the software product, if there is any fault, in early stages of designing. By doing this; time, effort and money required in maintaining software will be lessened. Although Object Oriented Software Development (OOSD) approach is equipped for enhancing software maintainability, a method for finding out if the required level of maintenance is being achieved amid the development process is required. This can be accomplished through measurement. This paper outlines the need and importance of maintainability at design phase and develops a Metrics-Based Maintainability Estimation Framework for Object-Oriented software(MEFOOS) that estimates the maintainability of object oriented software components in regard of their Understandability, Modifiability and Portability—which are the sub-attributes of maintainability.
Keywords: Software maintenance, Software Maintainability, maintainability model, Software Metrics, Software Component
DOI: 10.7176/JIEA/9-4-02
Publication date:June 30th 2019

1. Introduction

Maintainability as indicated by IEEE glossary of Software Engineering is defined as “the ease with which a software system or component can be modified to correct faults, get better performance or other attributes, or adapt to a change environment”, while Maintenance is defined, by the IEEE, as “the process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment”. Notwithstanding the fact that software maintenance is an expensive and challenging task, it is not well managed and frequently overlooked. One explanation behind this poor management is the absence of demonstrated measures for software maintainability (Pressman 2010). Given the fact that maintenance is (and will keep on being) the major resource consumer of the whole software development life cycle, maintainability has become one of the software product quality attribute that most software development organizations are more worried about. The mass of software organizations splurge 60% to 70% of resources for correcting, adopting and maintaining the existing software (Glass 2002; Bandiet al.2003). The 60% maintenance costs originate from making improvements, which is something that makes the systems give extra value (Lientz et al., 1978; Glass 2003). Subsequently, software companies must contrive different approaches to measure the ease of the maintenance process, not only to decrease the cost of maintainability but to as well ascertain whether maintenance of a specific software product is worthwhile or not.

As indicated by (Dubey and Rana, 2010), Object Oriented Software Development (OOSD) approach is capable of improving software maintainability, and has turned out to be mainstream in today's scenario of software development environment. However, a method of establishing if the required level of maintainability is being achieved amid the the Object Oriented Softaware Developement process is required. This can be accomplished through measurement. Measurement is required in software with a specific end goal to assess quality of software products as well as improvement of their performance (Chawla & Nath, 2013). Pressman (2010) underscores the significance of measuring software by expressing that; “if you do not measure, there is no genuine way of determining whether you are improving. And if you are not improving, you are lost” (p. 683). Measurement of software is accomplished by use of software metrics (Rawat, Mittal, & Dubey, 2012; Sharma & Dubey, 2012); and the task involved in maintainability measurement is; relating quantifiable maintainability characteristics with fitting metrics, and find out how these metrics collectively determine the maintainability of software components (Sandhu, Kaur, & Singh, 2009). At the end of the day, a framework that describes the maintainability of software and that structures appropriate metrics in a way that is easy to use is imperative in effective maintainability assessment (Hristov et al, 2012). This research concentrates on the maintainability of Object Oriented (OO) software components (classes).

2. Related Work

Research on software maintenance has been progressing for quite a while; with most works concentrating on the part of maintainability estimation —as a way of achieving effective maintenance. This has culminated into a number of metrics and frameworks for maintainability estimation. We review some of these works underneath.

2.1. Maintainability Measurement Frameworks

(Kiewkanya et al, 2004), introduced a maintainability model of class diagram utilizing three procedures viz. Discriminate technique, Weighted-Score-Level technique, Weighted-Predicted-Level technique and two sub-characteristics of maintainability: understandability and modifiability to assess maintainability model. These attributes are determined by factors, which can be straightforward or indirectly measured utilizing four software metrics, viz. McCabe's Cyclomatic Complexity, Halstead's Volume metrics, Regularity, and Reuse Frequency (Sandhu, Kaur, & Singh, 2009).

(Rizvi et al, 2010), built up a multivariate linear model ,Maintainability Estimation Model for Object-Oriented software in design stage (MEMOOD) ‟for evaluating the maintainability of UML class diagram in terms of understandability and modifiability''.

(Gautam et al, 2011), built up a a multivariate linear model Compound Maintainability Estimation Model for Object-Oriented software in design stage (Compound MEMOOD)‟ for estimatating the maintainability of class diagram in terms of understandability, modifiability, scalability and level of complexity.

(Tong Yi et al, 2014), correlated the advantages and disadvantage of class diagram complexity metrics based on statistics and entropy-distance in light of understandability, analyzability and maintainability. These attributes can be determined by various factors which can be straightforward or indirectly measured using appropriate metrics, which they propose.

2.2. Maintainability Metrics for OO software

Chidamber and Kemerer (CK) are the most referenced researchers in the area of OO metrics. They defined six metrics viz., Weighted Methods per Class (WMC), Response for a Class (RFC), Lack of Cohesion in Methods (LCOM), Coupling Between Object Classes (CBO), Depth of Inheritance Tree of a class (DIT) and Number of Children of a class (NOC) (Dubey & Rana, 2010).

Gill and Sikka (2011) proposed five inheritance hierarchy based metrics for measuring reuse and reusability in OO software development, which we arbitrarily chose to refer to as "the Gill and Sikka metrics". The metrics in question include; Breadth of Inheritance Tree (BIT), Method Reuse Per Inheritance Relation (MRPIR), Attribute Reuse Per Inheritance Relation (ARPIR), Generality of Class (GC) and Reuse Probability (RP). These metrics can be applied to different inheritance hierarchies at design time to help in selecting the best design, so that the development time and cost can be reduced.

Information flow metrics were proposed by Henry and Kafura, and they are sometimes referred to as Henry and Kafura's metrics (Singh et al., 2011). Information flow metrics can be used to determine the complexity of a system by measuring the flow of information among system modules (Laird & Brennan, 2006; Sharma & Dubey, 2012; Singh et al., 2011;). The underlying principle behind this is that, high information flow among system modules indicates lack of cohesion (i.e. a low degree of relationship between methods of a module), which causes higher complexity (Laird & Brennan, 2006). Information flow metrics use some combination of the number of local flows into a module (fan-in), the number of local flows out of a module (fan-out), and the length to compute a complexity number for a procedure. Fan-in, fan-out, and length are defined in a number of ways by different variations of the metric (Laird & Brennan, 2006).

3. Unresolved Issues in OO Maintainability Measurement

Although maintainability is undisputedly considered one of the fundamental quality attributes of software systems the research community has not yet produced a sound and accepted definition or even a common understanding what maintainability actually is. Even they are not sure about factors and still face difficulty in selecting influencing parameters of maintainability. This implies that, the research works have been conducted behind the backdrop of uncertainty as to which attributes should be used to measure maintainability (Hristov et al., 2012). In addition, OO metrics measure principle structures whose design affects quality attributes (Cho et al., 2001); therefore, a thorough understanding of OO concepts is requisite in order to use OO metrics (Dubey & Rana, 2010). This implies that, an effective OO maintainability framework should: (i) Estimate the maintainability of software at an early stage i.e. design phase of class diagrams in the software development life cycle, to significantly improve software quality and as well decrease overall cost, time and effort of rework, (ii) Clearly define a minimal set of maintainability factors (OO structures) for object oriented development system, which have optimistic impact on maintainability measurement, (iii) correlate each of the OO principle structures (concepts) with the maintainability attributes, and, (iv) Link various OO metrics with the maintainability factors (OO structures) that they measure, and structure them in a way that is easy to use. No such maintainability framework could be identified in literature; therefore, in spite of the existence of various OO metrics in literature, the challenges involved in assessing the maintainability of OO software effectively remain unresolved.

4. Proposed Framework

We propose a novel framework for measuring maintainability of OO classes in this paper. The framework correlates major maintainability attributes with different factors that influence them. These factors are determined by OO features that can be measured using appropriate metrics that exist in literature. The key elements of the framework include: the major maintainability attributes; factors influencing the maintainability attributes, measurable OO structures that determine each maintainability factor, metrics to measure the maintainability factors, and, a maintainability calculation framework. The framework is discussed in details in section 8.0 below.

5. Major Maintainability Characteristics for OO Software

Literature survey uncovers various attributes that are believed to impact maintainability of object oriented software. Such characteristics have been exhibited in, (Hristov et al., 2012; Caldiera & Basili, 1991; Washizaki et al., 2003; AL-badareen et al., 2010; Ilyas & Abbas, 2013). However, the problem is in figuring out which of these characteristics ought to be considered in assessing maintainability. Westfall (2005)) has remarked in this context by stating that; software entities possess many attributes that are quantifiable, and if all of these attributes are considered, then there are just too many measures, which may do more harm than good. An effective maintainability assessment framework ought to therefore, have as few attributes as possible, but at the same time adequate in assessing all aspects of maintainability. That is, overlapping and unimportant attributes should be excluded from such a framework. We discuss the major characteristics that we trust impact the maintainability of software components beneath.

5.1. Modifiability

The modifiability of a software system is the simplicity with which it can be modified to changes in the environment, requirements or functional (IEEE 1990). Thus as business processes change, organizations must modify their business applications to continue supporting the processes. Specifically, the modifiability of a business application determines the easiness of the application to be modified in response to changes caused by the environment, requirements or functional specification. . As indicated by (Ghezzi et al 2010) 50-70% of the cost in the lifecycle of a software system is dedicated to modifications after the initial development. Therefore, modifiability ought to be considered as a major maintainability attribute since its improvement is critical to reduce development costs and guarantee the success of a business application.

5.2. Understandability

Software systems have a tendency to withdraw more and more from the principle of simplicity and turn out to be progressively complex. The increase in size and complexity of software drastically influences several quality attributes, especially understandability and maintainability. Software developers and maintainers need to read and understand source programs and other documents of software. The significance of understandability is very evident that can be perceived as ‘If we can't learn something, we won't understand it. If we can't understand something, we can't use it - at least not well enough to avoid creating a money pit. Therefore, the beginning stage of maintainability a component is to understand its functionality, which requires high understandability (Washizaki et al., 2003). As indicated by Washizaki et al; "Understandability is defined based on the estimated effort needed by a user to recognize the concept behind a component and its applicability". Ghezzi, Jazayeri, and Mandrioli (2003) underscore the importance of understandability by stating that “Understandability is an internal product quality, and it helps in achieving many of the other qualities, such as evolvability and verifiability”.Therefore, understandability ought to be considered as a major maintainability attribute. Thus aforementioned facts reveal that understandability is a key factor to maintainability.

5.3. Reusability

Software reuse is the process of implementing or updating software systems using existing software components (Gill & Sikka, 2011). Thus issues related to software development such as; quality, productivity, cost of development etc, can be addressed by focussing and improving component reusability (AL-Badareen, Selamat, Jabar, Din, & Turaev, 2010; Budhija, et al, 2013; Babu & Srivatsa, 2009; Ilyas & Abbas, 2013; Mishra, Kushwaha, & Misra, 2009)). According to (Pressman, 2010), there is only one sure way of improving software quality, and that is through measurent. Therefore, developers must measure reusability of components if they need to improve software mainteinance in relation to maintainance cost management.

6. Design Features and Maintainability Factors

Factors that impact Understandability, Modifiability and Reusability are dictated by certain OO design features that can be measured. Thus, these factors can be directly be determined by measuring these features utilizing appropriate metrics. Fig.1. shows the relationship between the major maintainability attributes and the factors that influence them.

6.1. Factors influencing Understandability

6.1.1. Cohesion and Coupling

Understandability of a software component is determined by coupling and cohesion; i.e. for a component to be understandable, it ought to have high cohesion and low coupling (Ghezzi et al., 2003). The authors state that, distinctive elements of a module cooperate to perform the functionality of a module, and thus they are grouped together for logical reasons and not by sheer chance. Absence of cohesion or low cohesion increases complexity, whilst high cohesion increases simplicity (understandability) (Cho et al., 2001). On the other hand, a high level of component independence enables components to be analyzed and understood separately (Ghezzi et al., 2003). That is, if a component is highly dependent on other components, reference to the components to which it is dependent on, is required in order to understand it. This reference is minimized if the degree of independence of the component is high, thus understanding it, winds up noticeably less demanding.

6.2. Factors influencing Modifiability

6.2.1. Inheritance and coupling

Modifiability is firmly identified to inheritance and coupling. These two OO design constructs can be measured using inheritance-based metrics and coupling metrics respectively. In this case, inheritance-based metrics should be used to determine if inheritance is used in proper ranges. The NOC metric can be used to accomplish this goal. High values of NOC are an indication of a likelihood of a probability of uncalled for (excessive) use of inheritance, which may cause maintenance to be difficult. Coupling metrics on the other hand can be used to measure the degree to which software components (classes) are coupled or decoupled with other components of the same system. The CBO metric can be utilized to decide coupling, with low values of CBO being desirable.

6.3. Factors influencing Reusability

6.3.1. Inheritance and coupling

Reusability is closely related to inheritance, since inheritance empowers the formation of reusable components. Inheritance increases reuse and improves similarity of implementation. On the otherhand, it also increases the complexity of software and the coupling between classes leading to increasing of the effort put in for maintenance. For this situation, the NOC metric which is an inheritance-based metrics should be used to determine if inheritance is used in proper ranges. Coupling metrics on the other hand can be used to measure the degree to which software components (classes) are coupled or decoupled with other components of the same system. The CBO metric can be utilized to decide coupling, with low values of CBO being desirable.

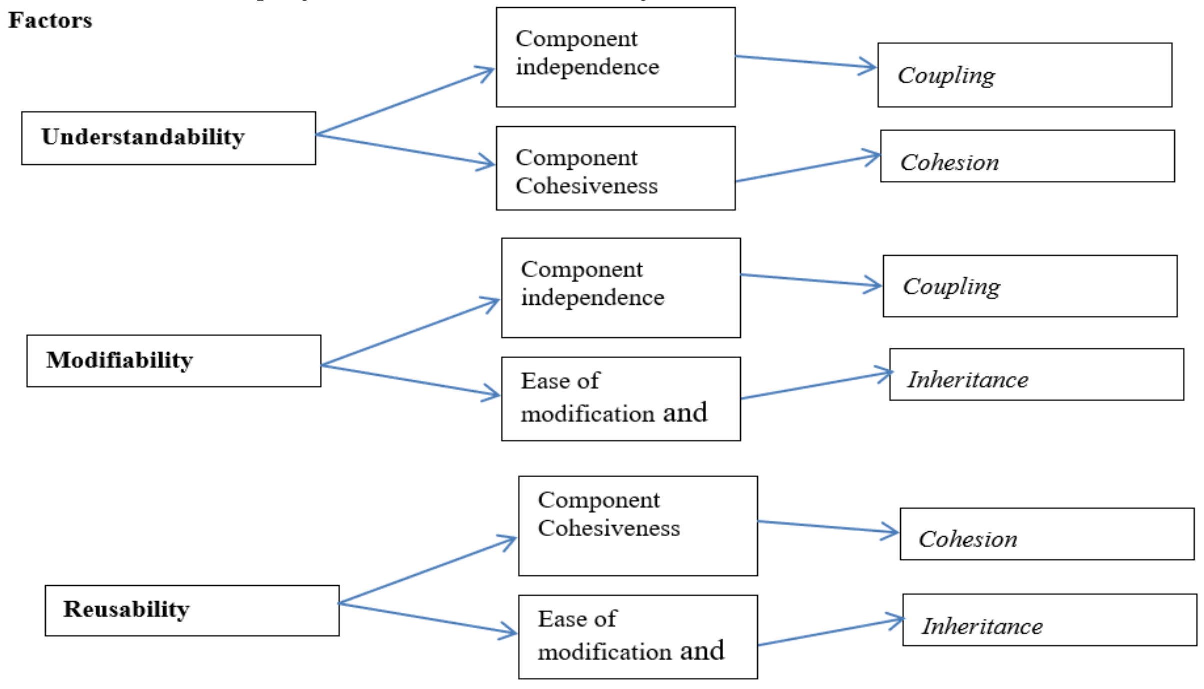

Fig. 1 Factors Influencing the Maintainability of OO Software

7. Candidate Metrics for the proposed Framework

Literature survey revealed some metrics that can be used to quantify the major maintainability attributes. Below is the initial suggestion of these metrics:

7.1. Measuring Modifiability

The two factors influencing modifiability viz., Inheritance and coupling. Thus, coupling metrics such as CBO can be used to determine component independence. Low values for CBO indicates high degree of independence. Ease of debugging and modification on the other hand, can be determined by inheritance metrics such as the NOC metric. Low values for NOC indicate a low degree of Component's complexity hence easy to modify and debug.

7.2. Measuring Understandability and Reusability

The two factors that influence Understandability i.e. Component independence and Component Cohesiveness are related to coupling and cohesion respectively. The CBO metric can be used to determine component independence, with Low CBO values being desirable. Cohesiveness on the other hand can be measured using cohesion metrics such as the LCOM metric. Low LCOM values (high cohesiveness) are desirable. Component reusability is determined by component cohesiveness and inheritance hierachy. Therefore, the NOC metric can be used to determine Reusability.

8. The Maintainability Estimation Framework for OO Software (MEFOOS)

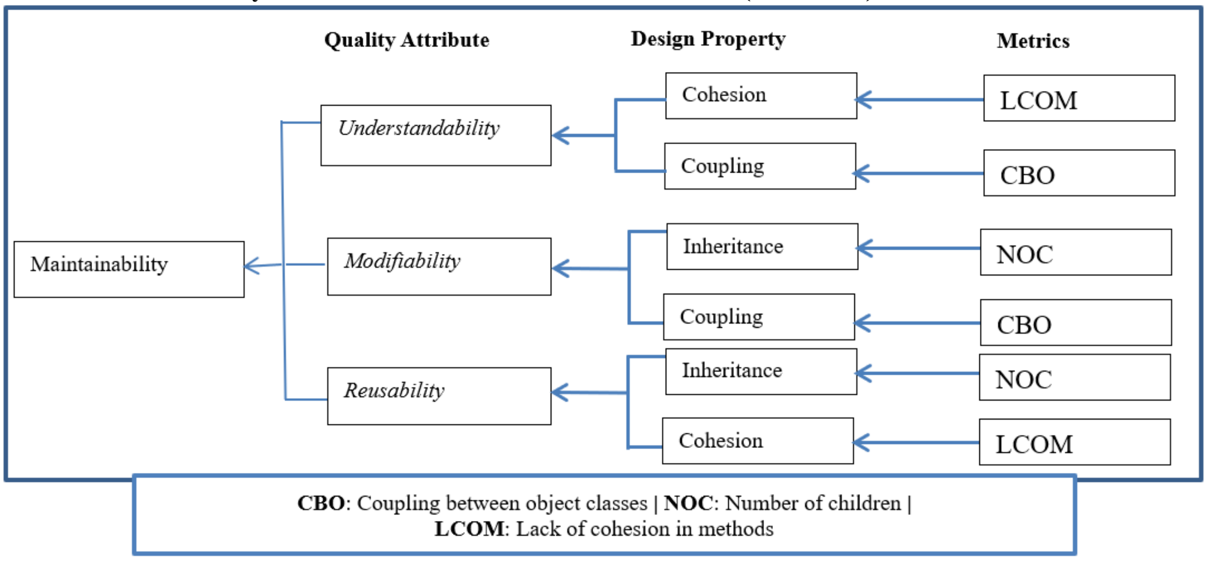

Fig. 2 Maintainability estimation framework for OO Software (MEFOOS)

A component's Maintainability (which we denote by Mc), can be calculated as:

$$M_c = \text{Understandability} + \text{Modifiability} + \text{Reusability} \qquad (1)$$

where values of the three attributes are obtained by measuring the attributes using appropriate metrics as proposed above. The three maintainability attributes are considered to be the most important ones, and are also of equal importance. The values of the three attributes are obtained by measuring the attributes using appropriate metrics (as indicated in fig. 2). These attributes are considered to important; hence, weighting values are assigned to them because some attributes are influenced by more factors than others. Therefore Maintainability of a software component can then be calculated using the expression:

$$M_c = w1\ .Und + w2\ .Mod + w3.Reus \qquad (2)$$

Where:

w1 to w3 are weighting values, and Und, Mod and Reus; are composite metrics for the maintainability attributes (i.e. Understandabiity, Modifiability and Reusability— respectively). The composite metrics values should be adjusted to a common scale to facilitate comparison of maintainability of different components in the same context (Hristov et al., 2012). Hristov et al. states that normalizing these values to the range of (0...1), is common in software metrics. The values of the weights; w1, w2 and w3 are 0.3,0.5 and 0.2 respectively. This is based on the fact that each maintainability attribute is determined by a varying number of factors, and there are a total of three factors in the maintainability attributes model. To obtain the Maintainability (Mc) of a software component, metrics values for each of the maintainability attributes should be obtained; by using appropriate metrics to measure the factors that affect each attribute—(with metric values for attributes that are determined by multiple factors being normalized to the range of (0...1)), then these composite metrics values should be aggregated into the maintainability calculation model (in equation 2).

9. Framework Experimentation

To demonstrate how the proposed model can be used to measure maintainability of OO components, we use it to calculate the maintainability of a sample (non-graphical user interface-based) for students loans application (obtained from Higher Education Loans Board). The application is web based developed using codeigniter framework which is PHP based. PHP as a tool is chosen since is object oriented and helps in building complex, reusable web applications. The UML block diagram of the application is shown in fig.3. The methods and instance variables in each of the classes are also listed in table 1 to 6 in next section.

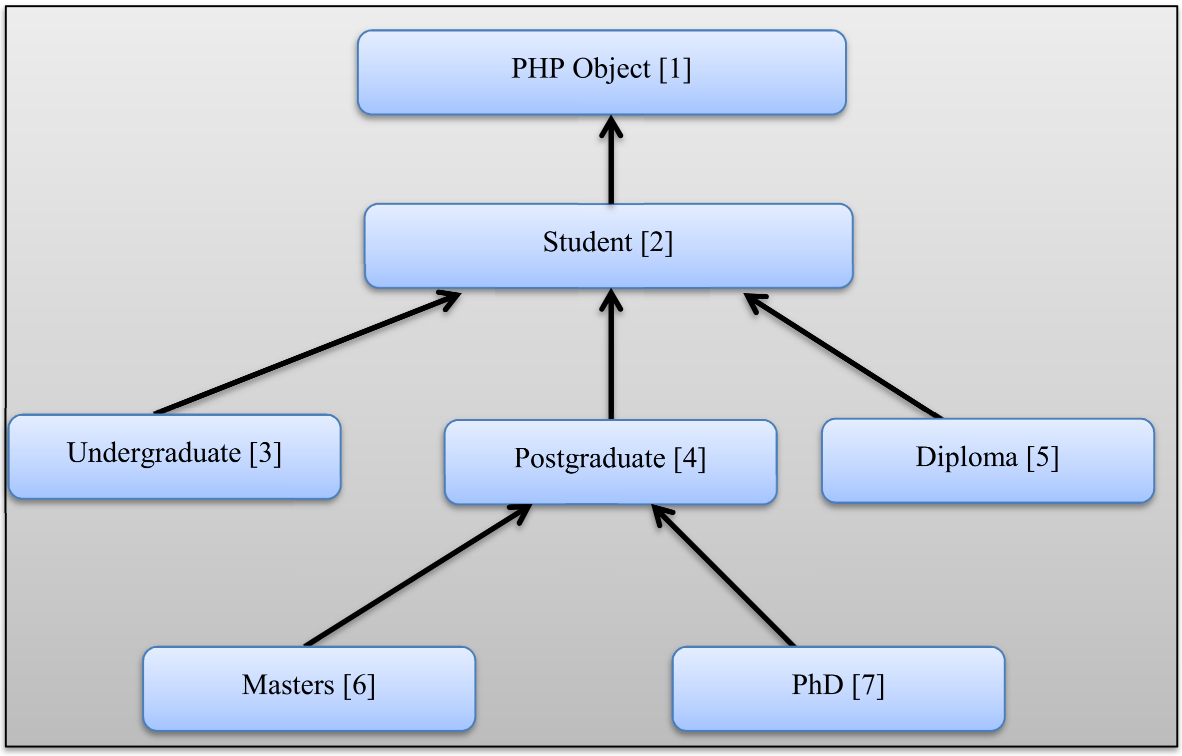

Figure 3: Class hierarchy for a sample OO component (adapted from Helb Disbursement System)

Table 1: Instance variables for the Student class

Method	**Instance Variables**
<<Constructor>> Student	firstName, lastName, nationalIDNumber
setFirstName	firstName
getFirstName	firstName
setLastName	lastName
getLastName	lastName
setNationalIDNumber	nationalNumber
getNationalIDNumber	nationalNumber
toString	firstName, lastName, nationalIDNumber
loans	ABSTRACT

Table 2: Instance variables for the Undergraduate Student subclass

Method	**Instance Variables**
UndergraduateStudent	firstName, lastName, IDnumber, loanType, Amount
setLoanType	loanType
getLoanType	loanType
setAmount	Amount
getAmount	Amount
loans	loanType, Amount
toDouble	Amount
toString	firstName, lastName, IDnumber, loanType

Table 3: Instance variables for the Diploma Student subclass

Method	**Instance Variables**
DiplomaStudent	firstName, lastName, IDnumber, loanType, Amount
setLoanType	loanType
getLoanType	loanType
setAmount	Amount
getAmount	Amount
loans	loanType, Amount
toDouble	Amount
toString	firstName, lastName, IDnumber, loanType

Table 4: Instance variables for the Postgraduate Student subclass

Method	**Instance Variables**
PostgraduateStudent	firstName, lastName, IDnumber, loanType, Amount
setLoanType	loanType
getLoanType	loanType
setAmount	Amount
getAmount	Amount
loans	loanType, Amount
toString	firstName, lastName, IDnumber, loanType

Table 5: Instance variables for the MastersPostgraduateStudent subclass

Method	**Instance Variables**
MastersPostgraduateStudent	firstName, lastName, IDnumber, loanType, Amount, Category
setLoanType	loanType
getLoanType	loanType
setCategory	Category
getCategory	Category
setAmount	Amount
getAmount	Amount
loans	loanType, Amount, Category
toDouble	Amount
toString	firstName, lastName, IDnumber, loanType, Category

Table 6: Instance variables for the PhDPostgraduateStudent subclass

Method	**Instance Variables**
PhDPostgraduateStudent	firstName, lastName, IDnumber, loanType, Amount, Category
setLoanType	loanType
getLoanType	loanType
setCategory	Category
getCategory	Category
setAmount	Amount
getAmount	Amount
loans	loanType, Amount, Category
toDouble	Amount
toString	firstName, lastName, IDnumber, loanType, Category

Our framework relates major maintainability attributes considered as important (understandability, modifiability and reusability) with factors that are determined by measurable OO principal constructs, and structures metrics for measuring the OO constructs in a way that is easy to use as shown in Fig 2 above.

9.1. Measuring Understandability

Understandability is closely related to cohesion and coupling. Coupling can be measured using the CBO metric, whilst cohesion can be measured using the LCOM metric. Understandability model is shown below.

$$\textbf{\textit{Understandability}} = \text{Und} = (1 - LCOM) + (1 - \frac{CBO}{n})$$

9.2. Measuring Modifiability

Modifiability is closely related to inheritance and coupling. Understandability model is shown below. These two

OO design constructs can be measured using inheritance-based metrics and coupling metrics respectively. The NOC metric can be used to achieve this objective. High values of NOC are an indication of a likelihood of improper (excessive) use of inheritance, which may cause maintenance to be difficult. The CBO metric can be used to determine coupling, with low values of CBO being desirable. Model is shown below.

$$\textbf{\textit{Modifiability}}\textbf{= \textit{Mod}} = \left(\mathbf{1}-\frac{\boldsymbol{CBO}}{\boldsymbol{n}}\right)+(\mathbf{1}-\frac{\boldsymbol{NOC}}{\boldsymbol{n}})$$

9.3. Measuring Reusability

Reusability is closely related to inheritance, since inheritance enables the creation of reusable components. In this case, The NOC metric which is an inheritance-based metrics should be used to determine if inheritance is used in proper ranges. The LCOM metric can be used to determine modules cohesiveness, with low values of LCOM being desirable. Model is hown below.

$$\textbf{\textit{Reusability}} = \textbf{\textit{Reus}} = (\mathbf{1}-\boldsymbol{LCOM})+\left(\mathbf{1}-\frac{\boldsymbol{NOC}}{\boldsymbol{n}}\right)$$

Variable names:

$\left(1-\frac{\text{CBO}}{\text{n}}\right)$ => Component_independence

$\left(1-\frac{\text{NOC}}{\text{n}}\right)$ => Ease_of_debugging_and_Modification

$(1-\text{LCOM})$ =>Lack_of_cohession_in_methods

Where n = number of classes for a given component.

Thus; Maintainability (M_c) =understandability + modifiability + reusability

9.4. Code Snippet

```
<?php
/**
 * User: Elijah Macharia
 * Date: 7/25/2016
 * Time: 11:08 AM
 */

class Maintainability
{
   private $cbo;
   private $noc;
   private $lcom;
   private $n; //number of classes in an object
   private $k;
   private $results;

   public function _cbo($x){

      /*
       * Variable x is the number of other classes a class is coupled with in the component and comes as an array
object
       * cbo=x/(n-1)
      */
      $this->n=count($x); //Get the number of classes (n)
      $this->cbo=0; //Set cbo to zero

      foreach($x as $value){
         $this->results=$value/($this->n-1); //Calculates cbo for each class in component
         $this->cbo +=$this->results; //increment values
      }

      return $this->cbo;
   }
```

```
    public function _lcom($x,$y){
        /*
        * Input is number of disjoint methods x and number of non disjoint methods y
      */
        if($x > $y){
            $this->results +=($x-$y);
        }else{
            $this->results +=0;
        }

        return $this->lcom=$this->results;
    }

    public function _noc ($x){
        /*
        * Variable x is the number of subclasses for each Class component
        * and comes as an array object
        * noc=x/(n-1)
      */
        $this->n=count($x); //Get the number of classes (n)
        $this->noc=0; //Set noc to zero

        foreach($x as $value){
            $this->results=$value/($this->n-1); //Calculates noc for each class
            $this->noc +=$this->results; //increment values
        }

        return $this->noc;
    }
   public function  understandability(){
    ...

?>
```

9.5. Calculating Coupling between object classes (CBO)
The CBO metric of a class is the count of the number of other classes to which that class is coupled with.
Required user input:
The user should enter the number of other classes each class of a component is coupled with, then the system should compute the CBO of each class (relative CBO) using the formula:

X/n-1,

where, x is the number of other classes a class is coupled with, and n is the number of classes for the component. The CBO of the component is computed by adding the relative CBO of all classes. This value is stored in a text box.
Example1
To calculate CBO for a component with 3 classes

Metric	Values	Classes		
		1	**2**	**3**
CBO	x (the number of other classes a class is coupled)	2	1	2
	Computed CBO	1	0.5	1

Class 1=2/ (3-1) =1 class2=1/ (3-1) =0.5 class3=2/ (3-1) =1
CBO=1 + 0.5 + 1=2.5
Component CBO=2.5

9.6. Calculating Lack of Cohesion in Methods (LCOM) Metric
Required user input:
Number of disjoint method pairs and
Number of non-disjoint method pairs for each class.
The LCOM metric of a class (relative LCOM) is calculated as:
If Number of disjoint method pairs<Number of non-disjoint method pairs Then

(Number of disjoint method pairs minusNumber of non-disjoint method pairs)
Else If Number of disjoint method pairs>Number of non-disjoint method pairs Then
LCOM = 0
The system should calculate the component LCOM by adding the LCOM values of each class. This value should is stored in a text box.

Example 2

To calculate LCOM for a component with 3 classes

Class	No. of disjoint method	No. of non-disjoint	Computed (Normalized) LCOM value
1	3	5	0
2	4	6	0
3	1	7	0

LCOM=0

9.7. Calculating Number of children (NOC)

The NOC of a class is the number of immediate subclasses subordinated to it in the class hierarchy.

Required user input:

The user should input the number of subclasses for each class in the class hierarchy. The system should calculate the NOC of each class (relative NOC) using the formula x/n-1, where x is the number of subclasses a class has and n is the number of classes. The NOC of the component is computed by adding the relative NOC of each class. This value is stored in a text box.

Example3

To calculate NOC for a component with 3 classes

Metric	Values	Classes		
		1	**2**	**3**
NOC	x (number of subclasses for each class in the class hierarchy)	1	1	1
	Computed NOC	0.5	0.5	0.5

Class1=1/ (3-1) =0.5 class2=1/ (3-1) =0.5 class3=1/ (3-1) =0.5
NOC=0.5 + 0.5 + 0.5
Component NOC=1.5
Hence to get Component Maintainability=Understandability + Modifiability + Reusability, We apply the following formulas as shown below.

a) **Calculating Understandability**

Understandability = Und $=(1-\text{LCOM})+(1-\frac{\text{CBO}}{\text{n}})$

Und = (1-0) + (1-2.5/3) =1.17

b) **Calculating Modifiability**

Modifiability= Mod = $\left(1-\frac{\text{CBO}}{\text{n}}\right)+(1-\frac{\text{NOC}}{\text{n}})$

Mod= (1-2.5/3) + (1-1.5/3)=-0.17 + 0.5=0.33

c) **Calculating Reusability**

Reusability = Reus = $(1-\text{LCOM})+\left(1-\frac{\text{NOC}}{\text{n}}\right)$

Reus= (1-0) + (1-1.5/3)=1+0.5=1.5

These attributes are considered to important; hence, weighting values are assigned to them because some attributes are influenced by more factors than others.Since the composite metrics values for the three attributes are normalized to the range of (0...1), the value for M_c will always be between 0 and 1. Mc shows the maintainability level of a component—where high Mc values (values close to 1) indicate high maintainability. Low values for Mc indicate low maintainability —which is an indication of possible flaws in the system design. Therefore, components with low Mc values should be subjected to further review. Hence Maintainability of a software component will then be calculated using the expression:

$$M_c = w1 .Und + w2 .Mod + w3.Reus$$

w1=0.3;
w2=0.5;
w3=0.1;
M_c= (w1*und) + (w2* mod) + (w3 * reuse);

Therefore maintainability for the component;
mai=(0.3 * 1.17) + (0.5 * 0.33) + 0.1(1.5)
mai =0.35 + 0.17 + 0.15

$M_c=0.67$

10. Framework Results

10.1. Measuring OO Features for the Student Loans Application

We used the proposed metrics to measure different OO features of the loans application, and then, the obtained values are analysed to determine the values for the maintainability attributes; which are aggregated into the maintainability calculation framework.

10.1.1. Coupling between Object Classes (CBO) Metric

Singh et al. (2011) define Coupling as, "the measure of strength of association established by a connection from one entity to another." The CBO metric is used to measure of how much coupling exists between classes (Sommerville, 2011). The CBO metric of a class is the count of the number of other classes to which that class is coupled with (Chidamber & Kemerer, 1994). CBO relates to the notion that an object is coupled to another object if methods of one object uses methods or instance variables of another (Chidamber & Kemerer, 1994). According to (Chidamber & Kemerer, 1991), any evidence of a method of one object using methods or instance variables of another object constitutes coupling.

10.1.2. Number of Children (NOC) Metric

The NOC of a class is the number of immediate subclasses subordinated to it in the class hierarchy (Chidamber & Kemerer, 1991, 1994).

10.1.3. Generality of Class (GC) Metric:

Generality of Class (GC) is the measure of its relative abstraction level, and it is obtained by dividing the abstraction level of the class by the total number of possible abstraction levels (Gill & Sikka, 2011).

10.1.4. Lack of Cohesion in Methods (LCOM) Metric

Cohesion can be defined as, the degree to which methods of a class are related to one another and work together to provide well bounded behavior (Singh et al., 2011). The LCOM metric is used to measure the cohesiveness of a class, by using instance variables to measure the degree of similarity of methods of a class, and it is defined as (Chidamber & Kemerer, 1994):

Consider a class C1 with n methods, M1, M2, ...,Mn. Let {Ij} = set of instance variables used by method Mi. There are n such sets {I1}, ... , {In}. Let P = {(Ii, Ij) | Ii ∩ Ij =Ø} and Q = {(Ii, Ij) | Ii ∩ Ij ≠ Ø}. If all n sets {I1}, ... , {In} are Ø then let P = Ø.

$$LCOM = |P| - |Q|, if\ |P| > |Q| \\ = 0 \quad otherwise \tag{3}$$

Example (Chidamber & Kemerer, 1994): Consider a class C with three methods M1, M2and M3. Let {I1} = {a, b, c, d, e} and {I2} = {a, b, e} and {I3} = {x, y, z}. {I1} ∩ {I2} is nonempty, (i.e. {I1} ∩ {I2} ≠ Ø), but {I1} ∩ {I3} and {I2} ∩ {I3} are null sets. LCOM is (the number of null intersections – number of nonempty intersections), which in this case is 1. According to (Chidamber & Kemerer, 1994), the LCOM value provides a measure of relative desperate nature of methods of a class.The values obtained after measuring different OO structures of the loans application are summarized in table 7 and 8. These values are analyzed and their interpretation given.

TABLE 7. Summary for NOC and CBO Measures for the Loans Application

Metric	Values	Classes							
		1	2	3	4	5	6	7	Total
NOC	Computed	0.2	0.5	0	0	0.3	0	0	1
	Maximum	1	1	1	1	1	1	1	7
CBO	Computed	0.2	0.8	0.2	0.2	0.5	0.2	0.2	2.3
	Maximum	1	1	1	1	1	1	1	7

NOC: table 7, shows that the computed value of NOC for the sample component is 1. This gives a normalized NOC value of 0.14. This value is obtained by dividing 1 by the maximum NOC value (i.e. 7). The lesser the value computed for NOC the lesser is the component‘s complexity, hence easy to debug and modify. The NOC value can be viewed as the difficulty of debugging and modification (Macharia et al., 2016a). Therefore ease of debugging and modification can then be obtained by subtracting the "difficulty of debugging and modification" from 1, since the highest possible value for ease of debugging and modification is 1. Thus, the value for ease of debugging and modification for the sample component is 0.86.

CBO: from table 7, the computed value for CBO is 2.3, compared to the maximum value of 7.When we normalize this value we obtain 0.32.Since the CBO value shows the, degree of interdependence between classes, then the degree of independence can be obtained by subtracting the degree of interdependence from 1; where 1 is the highest possible (normalized) value for the degree of independence. Therefore, the degree of independence for the sample component is 0.68. Although literature suggests that, CBO should be measured by counting the number of distinct non-inheritance related class hierarchies on which a class depends (Cho et al., 2001; Sharma & Dubey, 2012);

couplings due to inheritance are considered in the computation of CBO like in the case of (Chawla & Nath, 2013).
LCOM: The number of non-null intersections of instance variable pairs of methods (|Q|), for each of the classes of the sample component is greater than the number of null intersections (|P|), (i.e. |P| < |Q|); therefore, the LCOM value for all classes of the sample component is 0. This is based on the definition of LCOM. To find the cohesiveness of a class, its LCOM value is subtracted from 1—as the LCOM value measures the relative desperateness of a class, and the highest possible (normalized) value for cohesion is 1. Therefore, average cohesiveness for the component is 1. The computed LCOM and cohesion values for each class of the component are summarized in table 8.

Table 8: Summary of the LCOM measure for the sample component

Class	No. of Methods	Highest possible LCOM Value	Computed (Normalized) LCOM value	Cohesiveness
1				
2	6	28	0	1
3	5	10	0	1
4	7	21	0	1
5	7	21	0	1
6	5	10	0	1
7	7	21	0	1

The computed LCOM values are normalized with respect to the highest possible LCOM value, which is the total number of paired instance variables of methods in a class. That is to say that, a class has the highest value for LCOM if none of its paired methods have similar instance variables (i.e. when |Q| = 0).

10.3. Aggregating the Values into the Maintainability Calculation Model

To calculate the maintainability of the sample student loans component, the composite metrics viz., Und, Mod and Reus; for the three attributes are first calculated, and then aggregated into the maintainability calculation model:

$$M_c = w1.\ Und + w2.\ Mod + w3.\ Reus$$

w1 = 0.3 w2 = 0.5 w3 = 0.3
Und =0.5 (Component independence + Cohesiveness) => 0.5(0.68 + 1) =0.84
Mod = 0.5 (Component independence + Ease of modification and debugging)
=> 0.5(0.68 + 0.86) = 0.77
Rues = 0.5 (Cohesiveness + Ease of modification and debugging) => 0.5(1 + 0.86) =0.93
Therefore:
Component Maintainability (M_c) = 0.3 (0.84) + 0.5(0.77) + 0.2(0.93) = 0.82

10.4. Interpretation

The Maintainability for the sample loans application component is 0.82, compared to the maximum value of 1. Therefore it can be concluded that maintainability for the component is relatively high hence low maintenance cost (i.e. the maintainability of the component is at 82%).

11. Conclusion and Future Work

Software measurement is a key element in software Engineering; as it is used in evaluating quality of software, hence finding ways of improvement. Thus, measuring Maintainability is inevitable; if effective software maintenance is to be achieved in order to control or at least reduce future maintenance cost. We reviewed some research works on maintainability measurement, in order to understand the current state of research on OO software maintainability measurement: where we reviewed and presented a number of maintainability measurement frameworks and metrics that exist in literature. We also observed in this paper that software entities possess several measurable attributes, and trying to measure all of these attributes may be counterproductive. Thus, an effective software measurement framework should exclude trivial as well as overlapping attributes. We also discussed major attributes that we believed influence maintainability. We also noted in this paper that, in OO software, factors that influence the maintainability attributes are related with several OO design features, like inheritance, coupling etc., which can be measured using OO metrics. Thus, OO maintainability measurement requires a thorough understanding of how various OO design features influence the maintainability factors.

In this paper, we have proposed a novel maintainability measurement framework for OO software that considers three measurable features (i.e. Inheritance, coupling and cohesion) as the determinants for OO software maintainability that have direct impact to understandability, modifiability and reusability sub-characteristic of maintainability. The values of understandability, modifiability and reusability are of immediate use in the software development process and may help software designer to review the design and take appropriate corrective measures, early in the development life cycle, in order to control or at least reduce future maintenance cost. The maintenance team may also use this information to know, on what module to focus during maintenance. More sophisticated maintainability estimation model can be developed in future, by conducting a larger scale study with a variety of industrial projects across diverse domains. The maintainability estimation framework for OO software developed in the paper focuses on object-oriented paradigm, but in future more generalized maintainability estimation model can be developed. The future research may also focus on measuring other quality factors proposed in the ISO 9126, such as reliability, portability, testability etc. Beside these, a maintainability index can also be developed that may help software industry in project ranking.

References

AL-Badareen, A. B., Selamat, H. M., Jabar, M. A., Din, J., & Turaev, S. (2010). Reusable Software Components Framework. Advances in Communications, Computers, Systems, Circuits

Babu, G. N., & Srivatsa, S. K. (2009). Nanalysis and Measures of Software Reusabilit y. International Journal of Reviews in Computing, 41-46.

Budhija, N., Singh, B., & Ahuja, P. S. (2013, January). International Journal of Advanced Research

Caldiera, G., & Basili, V. R. (1991). Identifying and Qualifying Software Components. 24.

Chawla, S., & Nath, R. (2013, July). Evaluating Inheritance and Coupling Metrics. International Journal of Engineering Trends and Technology (IJETT), 4(7), 2903-2908.

Chidamber, S. R., & Kemerer, C. F. (1994, June). A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, 20(6), 476–493.

Cho, S. E., Kim, M. S., & Kim, D. S. (2001). Component Metrics to Measure Component Quality. Asia-Pacific Software Engineering Conference (APSEC'01). Eighth. IEEE.

Dhankhar, P., Mittal, H. M., & Amita. (2011, Jan). MAINTAINABILITY PREDICTION FOR OBJECT ORIENTED SOFTWARE. International Journal of Advances in Engineering Sciences, 1(1), 8-11.

Dubey, S. K., & Rana, A. (2010). A Comprehensive Assessment of Object-Oriented Software Systems Using Metrics Approach. 2(8).

Dubey, S. K., & Rana, A. (2010). A Comprehensive Assessment of Object-Oriented Software Systems Using Metrics Approach. International Journal on Computer Science and Engineering (IJCSE), 2(8), 2726-2730.

Farooq, S. U., Quadri, S. M., & Ahmad, N. (2011, January). SOFTWARE MEASUREMENTS AND METRICS: ROLE IN EFFECTIVE SOFTWARE TESTING. International Journal of Engineering Science and Technology (IJEST), 3(1), 671-680.

Frakes, W. B., & Kang, K. (2005). Software Reuse Research: Status and Future. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING. 31. IEEE Computer Society.

Ghezzi, C., Jazayeri, M., & Mandrioli, D. (2003). Fundamentals of Software Engineering (2nd Edition ed.). New Jersey: Prentice-Hall.

Gill, N. S., & Sikka, S. (2011, June). Inheritance Hierarchy Based Reuse & Reusability Metrics in OOSD. Inheritance Hierarchy Based Reuse & Reusability Metrics in OOSD, 3(6), 2300-2309.

Hristov, D., Hummel, O., Huq, M., & Janjic, W. (2012). Structuring Software Reusability Metrics. International Conference on Software Engineering Advances. IARIA.

Laird, L. M., & Brennan, M. C. (2006). Software Measurement and Estimation A Practical Approach. Hoboken, New Jersey: John Wiley & Sons, Inc.

Mishra, S. K., Kushwaha, D. S., & Misra, A. K. (2009). Creating Reusable Software Component from Object-Oriented Legacy System through Reverse Engineering. Journal of Object

Orenyi, B. A., Basri, S., & Jung, L. T. (2013, September). ISO/IEC9126-based Analyzability Estimation Model for OO Software. International Journal of Advancements in Computing Technology(IJACT), 5(13), 71-79.

Pressman, R. S. (2000). Software Engineering: A practitioner's Approach (5th ed.). McGraw-Hill.

Pressman, R. S. (2010). Software Engineering: Apractitioner's Approach (7th Edition ed.). New York: McGraw-Hill.

Riaz, M., Mendes, E., & Tempero, E. (2009). A Systematic Review of Software Maintainability Prediction and Metrics. Third International Symposium on Empirical Software Engineering and Measurement, (pp. 367-377).

Rizvi, S. W., & Khan, R. A. (2010, APRIL). Maintainability Estimation Model for Object-Oriented Software in Design Phase (MEMOOD). JOURNAL OF COMPUTING, 2(4), 26-32.

Sommerville, I. (2011). Software Engineering (9th Edition ed.). Boston: Pearson Education.