Journal of Information Engineering and Applications
ISSN 2224-5782 (print) ISSN 2225-0506 (online)
Vol.4, No.4, 2014

A Metrics-based Framework for Measuring the Reusability of Object-Oriented Software Components

Sammy Olive Nyasente* Prof. Waweru Mwangi, Dr. Stephen Kimani
School of Computing and IT, Jomo Kenyatta University of Agriculture and Technology, P O Box 62000-00200 Nairobi Kenya
* E-mail of the corresponding author: samqolive@yahoo.com

Abstract

The critical role played by software in socioeconomic advancement, has seen a rapid demand for software; creating a large backlog in affordable and quality software that needs to be written. Although software reuse is capable of addressing this issue, effective reuse is seldom to come by, thus the issue still remains unresolved. In order to achieve effective reuse, practitioners need to focus on reusability: the property that makes software reusable. Although Object Oriented Software Development (OOSD) approach is capable of improving software reusability, a way of ascertaining if the required degree of reusability is being achieved during the OOSD process is required. This can be achieved through measurement. The task involved in measuring reusability of Object oriented (OO) software is to; determine major reusability attributes of reusable components, relate these characteristics with factors that influence them, link each factor with measurable OO design features that determines them, relate each feature with appropriate metrics, and find out how these metrics collectively determine the reusability of components. A novel framework for achieving this task is proposed in this paper.

Keywords: Software reuse, Software Reusability, Software Metrics, Software Component

1. Introduction

Software reuse is the use of existing artifacts to create new software (Gill & Sikka, 2011). The purpose of reusing software is to address issues related to software quality and productivity; as it is often pursued to produce quality and reliable software that are delivered on time and within budget (Frakes & Kang, 2005). Nonetheless, reuse is still facing numerous issues, hence lacking adoption from practitioners (Hristov, Hummel, Huq, & Janjic, 2012). The major hindrance to effective reuse is not the lack of components that can be reused (Hristov et al.), but rather most of the existing software components have little or no reusability (Sametinger, 1997). According to (Gill & Sikka, 2011), *Reusability* is the degree to which a given software component can be reused. In other words, *Reusability* is a property of a software component that indicates its probability of reuse (Frakes & Kang, 2005). This means that, if a component's reusability is low, then its potential for reuse becomes low as well. Therefore, efforts should converge at developing components with a high degree of reusability; if effective reuse is to be achieved.

According to (Dubey & Rana, 2010), Object Oriented Software Development (OOSD) approach is capable of improving software reusability, and has become popular in today's scenario of software development environment. Nonetheless, a way of ascertaining if the required degree of reusability is being achieved during the OOSD process is required. This can be achieved through measurement. Measurement is required in software engineering in order to assess quality of software products as well as improvement of their performance (Chawla & Nath, 2013). Pressman (2010) underscores the importance of measuring software by stating that; "if you do not measure, there is no real way of determining whether you are improving. And if you are not improving, you are lost" (p. 683). Measurement of software is achieved by use of software metrics (Rawat, Mittal, & Dubey, 2012; Sharma & Dubey, 2012); and the task involved in reusability measurement is; relating measurable reusability characteristics with appropriate metrics, and find out how these metrics collectively determine the reusability of components (Sandhu, Kaur, & Singh, 2009). In other words, a framework that describes the reusability of software and that structures appropriate metrics in a way that is easy to use is requisite in effective reusability assessment (Hristov et al, 2012). This research focuses on the reusability of Object Oriented (OO) components (classes).

2. Related Work

Research on software reuse has been ongoing for a long time; with most works focusing on the aspect of

reusability measurement—as a way of achieving effective reuse. This has culminated into a number of metrics and frameworks for reusability measurement. We review some of these works below.

2.1. Reusability Measurement Frameworks

Caldiera and Basili (1991) propose a framework for measuring the reusability of software components. They define a reusability attributes model, which attempts to characterize reusability, attributes directly through measures of an attribute, or indirectly through measures of evidence of an attribute's existence. The model consists of three attributes that are believed to influence the reusability of components; viz., reuse costs, functional usefulness, and quality of components. These attributes are determined by factors, which can be directly or indirectly measured using four software metrics, viz. McCabe's Cyclomatic Complexity, Halstead's Volume metrics, Regularity, and Reuse Frequency.

Another reusability measurement framework has been proposed by (Washizaki, Yamamoto, & Fukazawa, 2003). This framework targets the JavaBeans architecture (JavaBeans components). The authors propose a component reusability model, in which three major reusability attributes, i.e. understandability, adaptability and portability are considered. They further define five metrics, for measuring factors that influence the reusability attributes. The metrics include; Existence of Meta-Information (EMI), Rate of Component Observability (RCO), Rate of Component Customizability (RCC), Self-Completeness of Component's Return Value (SCCr), and Self-Completeness of Component's Parameter (SCCp).

AL-Badareen, Selamat, Jabar, Din, and Turaev (2010) propose another reusability assessment framework for systematic reuse. They categorize reusability characteristics into two main categories, viz., characteristics to assess components before they are stored in the reuse library, and, characteristics to assess components in order to build a new system. The first category considers the general characteristics that are required by any system, and they include: Software coexistence, adaptability/interoperability, generality, and compliance. The second category of characteristics considers specific characteristics that help in the system development, and they include; component suitability, documentation and modifiability. The authors propose that existence of these characteristics in a component should be determined by conducting tests on the component. For example, system and hardware independence, which are the two aspects of coexistence can be determined by running the software in different software and hardware environments respectively.

Hristov et al. (2012) propose a reusability assessment framework for ad-hoc reuse. Their framework structures existing reusability metrics for component-based software development. They propose eight attributes that should be considered in assessing the reusability of components in ad-hoc reuse scenario. The attributes include: availability, documentation, complexity, quality, maintainability, adaptability, reuse and price. These attributes can be determined by various factors which can be directly or indirectly measured using appropriate metrics, which they propose.

Another reusability assessment framework is proposed by (Ilyas & Abbas, 2013). The authors define three major attributes for determining reusability when extracting reusable components from existing components. The attributes are; Component Versatility, Reliability, and Understandability. Versatility, characterizes the multiplicity of tasks that can be simultaneously performed by a component, and it is determined by two factors, viz., Generality and Portability. These two factors can be determined by Generality and portability metrics respectively. Reliability on the other hand characterizes how well a software component has performed the required functionality. This can be determined by investigating the component's performance history in different circumstances for a given period of time—by recording a component's; failure history, Error rate, Error type, and Men Time to Failure. Lastly Understandability characterizes the ease with which a software component can be understood, and it can be determined by the readability metric.

2.2. Reusability Metrics for OO software

Chidamber and Kemerer (1994), propose six metrics for Object-Oriented design. The metrics include; 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). Literature indicates that five of the metrics can be used in assessing reusability, i.e. WMC, NOC, CBO (Chidamber & Kemerer 1994; Singh, Singh, & Singh, 2011), DIT (Chidamber & Kemerer 1994; Gill and Sikka, 2011), and LCOM (Cho, Kim, & Kim, 2001).

Cho et al. (2001) present metrics for measuring complexity, customizability and reusability of components. They proposed four metrics to measure different aspects of a Component's complexity, which include; component plain complexity (CPC), component static complexity (CSC), component dynamic complexity (CDC), and

component Cyclomatic complexity (CCC) metrics. The authors observe that, if a component does not provide customizable interfaces, its reusability becomes low, because reusable components are often modified to meet new requirements in different reuse contexts. Thus, they propose the Component variability (CV) metric to measure a Component's customizability. Lastly they propose two approaches for measuring the reusability of a component viz., how a component has reusability, and, how a component is reused in a particular application. Thus, they define the component itself reusability (CR) metric to measure the former, and the Component Reuse Level (CRL) metrics to measure the latter.

Gill and Sikka (2011) propose five inheritance hierarchy-based metrics for assessing Reuse and Reusability of OO software: 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). Two of the five metrics viz., GC and RP can be used to measure reusability, whilst the other three are for measuring Reuse.

3. Unresolved Issues in OO Reusability Measurement

Quality characteristics of OO software are dependent on OO-design features such as inheritance, coupling, cohesion etc., which can be measured using metrics; thus, quality assessment for OO software requires a thorough understanding of these features (Dubey & Rana, 2010). It then follows that; an effective OO reusability assessment framework should: (i) clearly define the attributes that affect the reusability of components (classes), (ii) define factors that influence each of the reusability attributes (iii) relate each of the OO design features with the factors that influence the reusability attributes, (iv) relate various OO metrics with the OO structures that they measure, and (v) determine how these metrics collectively determine reusability of classes. No such reusability assessment framework could be found in literature. Therefore, in spite of the existence of other reusability assessment frameworks, and various OO metrics in literature, the issue of effective reusability assessment of OO components remains unresolved.

4. Proposed Framework

We propose a novel framework for measuring reusability of OO classes in this paper. The framework correlates major reusability 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 reusability attributes; factors influencing the reusability attributes, measurable OO structures that determine each reusability factor, metrics to measure the reusability factors, and, a reusability calculation model.

5. Major Reusability Characteristics for Software Components

Literature survey reveals a number of characteristics that are believed to influence reusability of software components. Such characteristics have been presented 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 determining which of these attributes should be considered in assessing reusability. Westfall (2005) has commented in this context by stating that; software entities possess many attributes that are measurable, and if all of these attributes are considered, then there are just too many measures, which may do more harm than good. An effective reusability assessment framework should therefore, have as few attributes as possible, but at the same time sufficient in assessing all aspects of reusability. That is, overlapping and trivial attributes should be excluded from such a framework. We discuss the major characteristics that we believe influence the reusability of software components below.

5.1. Generality

Generality is defined in IEEE Standard 610.12, as "the degree to which a system or component performs a broad range of functions." Generality increases the reusability of a component (Gill & Sikka, 2011; Sommerville, 2011). That is, if generality of a component increases, then its probability to be reused increases. The hypothesis here is that: for any generalized component; the probability that the problem at hand may be restated as an instance of a problem solved by the component is high. Návrat and Filkorn (2005) also comment on the importance of generality with respect to reusability by stating that, things can only get reused if they are general and allow turning to specifics in a clear and straightforward manner.

5.2. Understandability

A software component is more usable if it is can be easily understood (Hristov et al., 2012). More often than not, a developer will decide to reuse a component based on how well the component meets new requirements, which requires high understandability (Washizaki et al., 2003). According to Washizaki et al.*;* Understandability is defined based on the estimated effort required to recognize the concept behind a component and its applicability. Thus, understandability can be viewed as the property of a component "*not being complex*". Ghezzi, Jazayeri, & Mandrioli (2003) underscore the importance of understandability by stating that, understandability helps in achieving many of the other desirable qualities of software, such as evolvability and verifiability.

5.3. Portability

According to the IEEE Standard 610.12, portability is "the ease with which a system or component can be transferred from one hardware or software environment to another". Intuitively, the easier it is to transfer a component from one environment to another, the more the likelihood that the component will be reused in other applications. That is to say, that if a component has little or no portability then its chances of being reused reduce. For instance: if the effort required to modify a component in order for it to work a new environment equals the effort required to build the same component from scratch, then the component may as well be built from scratch. According to (Ghezzi et al., 2003), Portability is economically important because it helps amortize the investment in the software system across different environments and different generations of the same environment. This means, the payoff from reuse is higher for components that are environment independent.

5.4. Maintainability

The IEEE Standard 610.12 defines Maintainability as: “the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.” Ghezzi et al (2003), distinguish three categories of maintenance, viz. corrective, adaptive, and perfective. Corrective maintenance deals with the removal of residual errors that are present in the product when delivered, as well as errors introduced into the software during its maintenance. Perfective maintenance involves changing the software to improve some of its qualities. Here changes are due to the need to modify the functions offered by the application, add new functions, improve the performance of the application, make it easier to use etc. Adaptive maintenance on the other hand, involves adjusting the applications to changes in the environment. Maintenance is inevitable when a component is being reused in a new context, because developers may want to modify it for any of the reasons stated above. This should be achieved with a reasonable amount of work. Maintainability is an important characteristic because: (Ghezzi et al., 2003) there is evidence that maintenance costs exceed 60 percent of the total costs of software.

5.5. Documentation

Documentation is intended to make software components easier to understand (AL-badareen et al, 2010; Hristov et al., 2012). Proper documentation is of utmost importance because: (Vliet, 2000), software which is not sufficiently documented is bound to incur high costs later on. For example: (Ghezzi et al., 2003; Vliet, 2000) Maintenance is adversely affected by lack of proper documentation. Although documentation is largely a factor of understandability, it should be considered as a major attribute due to the fact that; Vliet, 2000) it gets the worst attention, which results to effects that counteract the objectives of reuse.

6. OO Design Features and Reusability Factors

Factors that influence maintainability, Portability, Understandability, and generality are determined by certain OO design features that can be measured. Thus, these factors can be indirectly determined by measuring these features using appropriate metrics. Documentation on the other hand is not related to any OO design features, but it can be determined as proposed by (Hristov et al., 2012): by use of four attributes viz., amount, quality, completeness, and, availability of legal terms and conditions. Fig.1. shows the relationship between the major reusability attributes and the factors that influence them.

6.1. Factors influencing Maintainability

6.1.1. Ease of Modification and Debugging

Maintainability involves two aspects viz., reparability and evolvability (Ghezzi et al., 2003). The former deals with correction of defects (debugging), whilst the latter involves modifying the software to satisfy new requirements. Software is said to be maintainable if these two aspects (i.e. debugging and modification) can be achieved with a reasonable amount of work. The difficulty of maintaining software is brought about by increased

software complexity (Laird & Brennan, 2006). In OO design, Complexity of software is increased if inheritance is not used in proper range; i.e. if it is overused or misused (Chawla & Nath, 2013). This means that; in order to achieve ease of debugging and modification, inheritance should be measured, to determine if it has been used properly (i.e. in proper range); and if not, the design should be reviewed and improved.

6.1.2. Component Independence

Coupling characterizes a module's relationship to other modules. It measures the interdependence of two modules; where modules that are dependent on each other heavily are said to have high coupling (Ghezzi et al., 2003). When classes of a system are highly dependent on each other, it is more likely that changing one class will affect other classes of the system (Sommerville, 2011). This means that, high interdependence between classes makes software modification difficult to perform. That is, high coupling will make it difficult to modify a component in order to fit new reuse contexts. Reusable classes should therefore exhibit a high degree of independence (i.e. low coupling). That is to say; that the degree of independence of a class can be easily determined by measuring coupling.

6.2. Factors influencing Portability

Low coupling (high degree of independence) according to (Ghezzi et al., 2003) enables a module to be reused separately. If a component is heavily dependent on other components, reusing it without the other components may not be possible, or it may require a lot of modifications for it to be reused separately. According to (Sametinger, 1997), low coupling is important with respect to component portability; as a component is also (indirectly) dependent on platforms of components with which it interacts. Washizaki et al. (2003) consider external dependency as one of the factors that affect portability, and according to them; external dependency indicates a component's degree of independence from other components of the software which originally used the component. This means that, external dependency and coupling are semantically equal in this context.

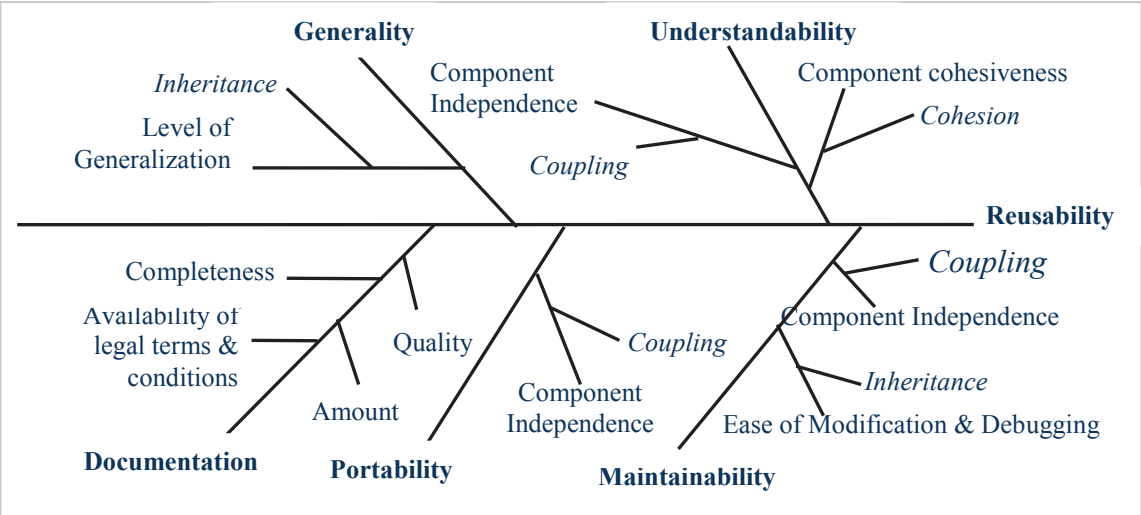

Fig. 1 Factors Influencing the Reusability of OO Components

6.3. Factors influencing Generality

Generality of OO software is achieved through generalization; i.e. by factoring out what is common to different components in the parent class, and then single out the variations subclasses. More often than not, all those features that are likely to be sufficiently general to be reused are factored out in the parent class (Ghezzi et al., 2003). Generalization is implemented using Inheritance mechanisms built into the OO languages; i.e. all reusable features that are factored out in the parent class are absorbed by heir classes, which are derived from the parent classes (Sommerville, 2011). According to (Gill & Sikka, 2011), the level of generalization of a class is determined by its relative abstraction level.

6.4. Factors influencing Understandability

6.4.1 Component cohesiveness and component independence

Understandability of a component is determined by coupling and cohesion; i.e. for a component to be understandable, it should have high cohesion and low coupling (Ghezzi et al., 2003). The authors state that, different 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. Lack 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, becomes easier.

7. Candidate Metrics for the proposed Framework

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

7.1. Measuring Maintainability

The two factors influencing maintainability viz., Component independence, and Ease of maintenance and debugging are determined by Coupling and inheritance respectively. 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

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.

7.3. Measuring portability

Component portability is determined by component independence. Therefore, the CBO metric can be used to determine portability.

7.4. Measuring Generality

Generality of a component is determined by a component's level of generalization, which is determined by its relative abstraction level. This concept is related to inheritance; therefore inheritance-hierarchy-based metrics such as the GC metric can be used to measure the generality of classes, where higher values of GC indicating high degree of generality.

7.5. Measuring Documentation

There are four factors used to determine documentation i.e. amount of documentation; quality; completeness; and, availability of legal terms and conditions. The amount of documentation can be measured through size, e.g. in kilobytes (kB) etc, whereas the existence of legal terms and conditions is a Boolean metric; i.e. either this information is provided or not (Hristov et al., 2012). Hristov et al. state that, Quality and Completeness are subjective measures, which should be measured on an ordinal scale based on advice of an expert. However, Quality can be determined by evaluating certain features for producing quality documentation, whereas completeness can be determined by evaluating if documentation possesses all the necessary parts.

Sommerville (2001) proposes three features that determine documentation quality viz., document structure, documentation standards and writing style. Document structure is the way in which the material in the document is organized. This has a major impact on readability and usability and it is important to design this carefully when creating documentation. It allows each part to be read as a single item and reduces problems of cross-referencing when changes have to be made. Documentation Standards on the other hand, ensure that produced documentation has a consistent appearance. According to Sommerville, documentation standards are dependent on the nature of the project, and therefore it is important that, appropriate standards that suit each project are chosen. In addition to the above, good documentation is fundamentally dependent on the writing style (writer's ability to construct clear and concise technical prose). That is; good documentation requires good writing.

Sommerville (2001) also gives a suggestion of seven parts that documentation for large systems that are developed to a customer's specification should include, and three parts that documentation for small systems that are developed as software products should have. Reusable components fall into the latter category, and documentation for such systems should include at least the following parts: specification of the system, an architectural design document, and, the program source code.

8. The Reusability Calculation Model for OO Components

In order to measure the reusability of classes, it is necessary to define a reusability calculation model. This model is based on the reusability attributes model shown in fig. 2. The relationship between the major reusability attributes, the factors that influence these attributes, and the metrics for measuring these factors are shown in this model. Theoretically, the reusability of a software component (denote by R_c), can be calculated using the expression:

$$R_C = Maintainability + Portability + Documentation + Generality + Understandability \quad (1)$$

The values of the five attributes are obtained by measuring the attributes using appropriate metrics (as indicated in fig. 2). These attributes are considered to be of equal importance; hence, weighting values are assigned to them, because some attributes are influence by more factors than others. Therefore Reusability of a software component can then be calculated using the expression:

$$R_C = w_1.Mai + w_2.Port + w_3.Doc + w_4 Gen + w_5 Und \quad (2)$$

Where:

w_1 to w_5 are weighting values, and *Mai*, *Port, Doc, Gen,* and *Und;* are composite metrics for the reusability attributes (i.e. Maintainability, portability, Documentation, generality, and understandability— respectively). The composite metrics values should be adjusted to a common scale to facilitate comparison of reusability 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; w_1, w_2, w_3, w_4 and w_5 are 0.2, 0.1, 0.4, 0.1, and 0.2 respectively. This is based on the fact that each reusability attribute is determined by a varying number of factors, and there are a total of ten factors in the reusability attributes model. To obtain the *Reusability* (R_c) of a software component, metrics values for each of the reusability 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 reusability calculation model (in equation 2).

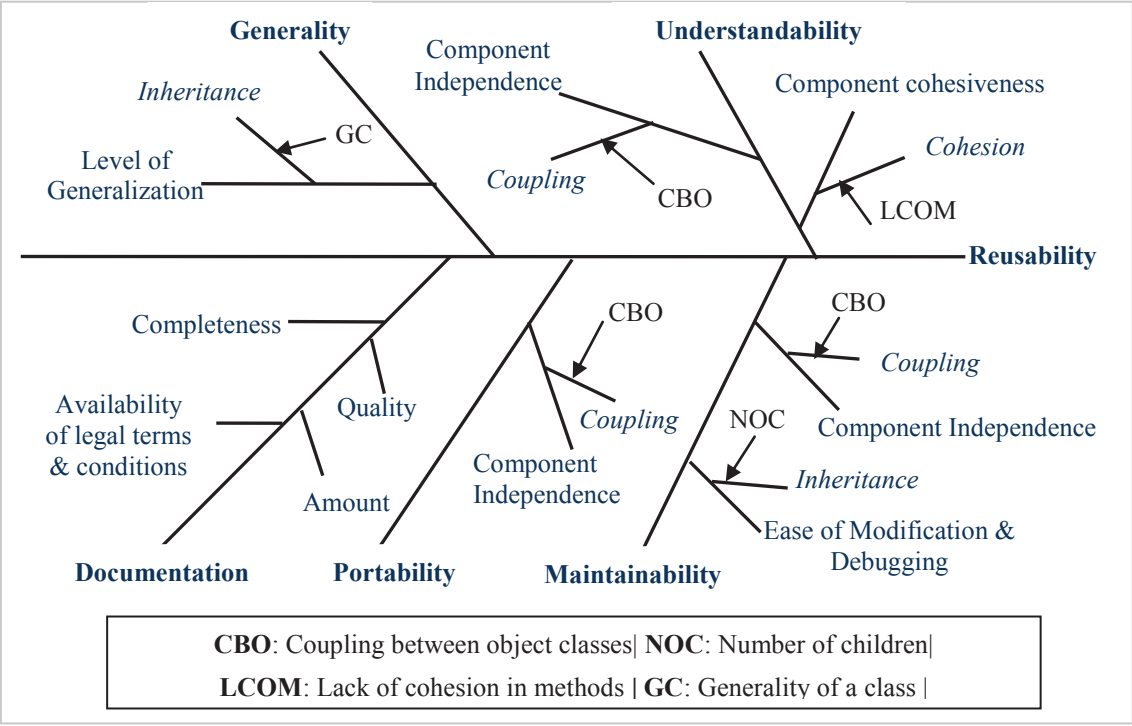

Fig. 2 Reusability Attributes model for OO Components

9. Experimentation and Results

To demonstrate how the proposed model can be used to measure reusability of OO components, we use it to calculate the reusability of a sample (non-graphical user interface-based) java payroll application (obtained from

(Deitel & Deitel, 2012)). The UML block diagram of the application is shown in fig.3. The Employee class is a subclass of Java.lang.Object, because: (Deitel & Deitel, 2012) all java classes inherit from class object. The methods and instance variables in each of the classes are also listed in table 1 to 5.

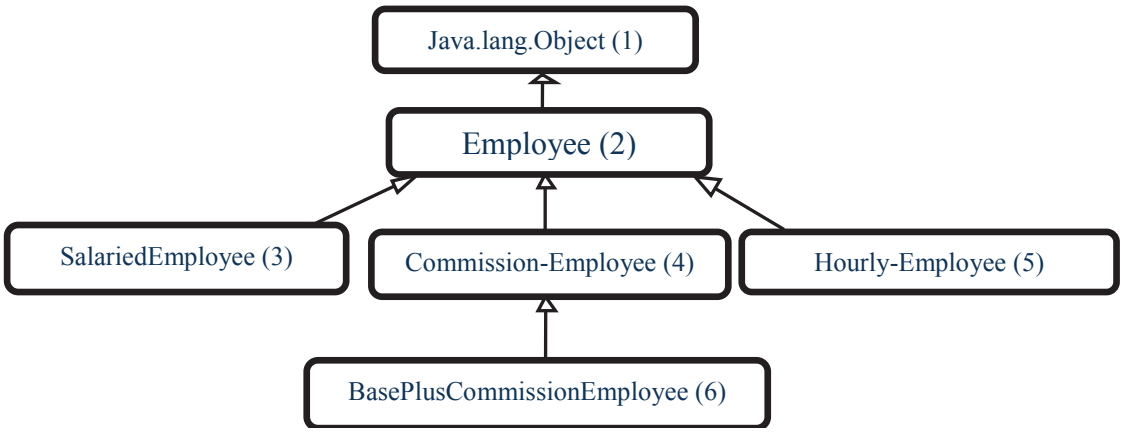

Fig. 2. The Class Hierarchy payroll Application

Table 1. Instance Variables in the Abstract Class Employee

Method	**Instance Variables**
<<Constructor>> Employee	firstName, lastName, socialSecurityNumber
setFirstName	firstName
getFirstName	firstName
setLastName	lastName
getLastName	lastName
setSocialSecurityNumber	socialSecurityNumber
getSocialSecurityNumber	socialSecurityNumber
toString	firstName, lastName, socialSecurityNumber
earnings	ABSTRACT

Table 2. Instance Variables in the Subclass SalariedEmployee

Method	**Instance Variables**
SalariedEmployee	firstName, lastName, socialSecurityNumber, weeklySalary
setWeeklySalary	weeklySalary
getWeeklySalary	weeklySalary
earnings	weeklySalary
toString	firstName, lastName, socialSecurityNumber, weeklySalary

Table 3. Instance Variables in the Subclass HourlyEmployee

Method	**Instance Variables**
HourlyEmployee	firstName, lastName, socialSecurityNumber, wage, hours
setWage	wage
getWage	wage
setHours	hours
getHours	hours
earnings	wage, hours
toString	firstName, lastName, socialSecurityNumber, wage, hours

Table 4. Instance Variables in the Subclass CommissionEmployee

Method	**Instance Variables**
CommissionEmployee	firstName, lastName, socialSecurityNumber, grossSales, commissionRate
setCommissionRate	commissionRate
getCommissionRate	commissionRate
setGrossSales	grossSales
getGrossSales	grossSales
earnings	commissionRate, grossSales
toString	firstName, lastName, socialSecurityNumber, grossSales, commissionRate

Table 5. Instance Variables In The Subclass BasePlusCommissionEmployee

Method	**Instance Variables**
BasePlusCommission-Employee	firstName, lastName, socialSecurityNumber, grossSales, commissionRate, baseSalary
setBaseSalary	baseSalary
getBaseSalary	baseSalary
earnings	baseSalary, commissionRate, grossSales
toString	firstName, lastName, socialSecurityNumber, grossSales, commissionRate, baseSalary

9.1. Measuring OO Features for the Payroll Application

We use the proposed metrics to measure different OO features of the payroll application, and then, the obtained values are analyzed to determine the values for the reusability attributes; which are aggregated into the reusability calculation model.

9.1.1. 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.

9.1.2. 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).

9.1.3. 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).

9.1.4. 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 C_1 with n methods, $M_1, M_2, ...,M_n$. Let $\{I_j\}$ = set of instance variables used by method M_i. There are n such sets $\{I_1\}, ... , \{I_n\}$. Let $P = \{(I_i, I_j) \mid I_i \cap I_j = \emptyset\}$ and $Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \emptyset\}$. If all n sets $\{I_1\}, ... , \{I_n\}$ are Ø then let $P = \emptyset$.

$$LCOM = |P| - |Q|, if\ |P| > |Q| \\ = 0 \quad otherwise \qquad (3)$$

Example (Chidamber & Kemerer, 1994): Consider a class C with three methods $M_1, M_2$and M_3. Let $\{I_1\} = \{a, b, c, d, e\}$ and $\{I_2\} = \{a, b, e\}$ and $\{I_3\} = \{x, y, z\}$. $\{I_1\} \cap \{I_2\}$ is nonempty, (i.e. $\{I_1\} \cap \{I_2\} \neq \emptyset$), but $\{I_1\} \cap \{I_3\}$ and $\{I_2\} \cap \{I_3\}$ 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 payroll application are summarized in table 6 and 7. These values are analyzed and their interpretation given.

TABLE 6. Summary for NOC, CBO and GC Measures for the Payroll Application

Metric	**Values**	**Classes**						
		1	2	3	4	5	6	**Total**
NOC	Computed	0.2	0.6	0	0.2	0	0	**1**
	Maximum	**1**	**1**	**1**	**1**	**1**	**1**	**6**
CBO	Computed	0.2	0.8	0.2	0.4	0.2	0.2	2
	Maximum	**1**	**1**	**1**	**1**	**1**	**1**	**6**
GC	Computed	**1**	**0.75**	**0.5**	**0.5**	**0.5**	**0.25**	**3.5**
	Maximum	**1**	**1**	**1**	**1**	**1**	**1**	**6**

NOC: The computed value of NOC is 1, compared to the maximum value of 6 (i.e. the NOC value is **0.17**). The lesser value for NOC indicates that the component is not complex, hence easy to debug and modify. The NOC values can be seen as "*the difficulty of debugging and modification*"; and therefore ease of modification and debugging can 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* is **0.83**.

CBO: The computed value for CBO is 2 compared to the maximum value of 6. Thus, the degree of interdependence between classes is 0.33. To get the degree of independence, the degree of interdependence is subtracted from 1; where 1 is the highest possible (normalized) value for the degree of independence. Thus, the degree of independence for the payroll application is **0.67**. 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).

GC: The computed value for GC is 3.5, compared to the maximum value of 6. Therefore, the *level of generalization* for the payroll application is **0.58**.

LCOM: The number of non-null intersections of instance variable pairs of methods (|Q|), of each class of the payroll application is greater than the number of null intersections (|P|), (i.e. |P| < |Q|); therefore, the LCOM value for all classes of the payroll application is 0. To find the cohesiveness of a class, its LCOM value should be 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 payroll component is **1**. The computed LCOM and cohesion values for each class of the Payroll application are summarized in table VII. 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 of 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).

Table 7. Summary of the LCOM Measure for the Payroll Application

Class	**No. of Methods**	**Highest possible LCOM Value**	**Computed (Normalized) LCOM value**	**Cohesiveness**
1				
2	8	28	0	1
3	5	10	0	1
4	7	21	0	1
5	7	21	0	1
6	5	10	0	1

9.2. Measuring Documentation

9.2.1. Documentation Quality

The computed value for documentation quality is 15 out of a possible value of 15. Therefore the value for quality is **1**.The criteria used in calculating the value for quality is given in table 8.

Table 8. Measure for Documentation Quality

Factor	**Criterion**	**Comment**	**value on a scale of 1-5**	**Max value**
Quality	Document structure	-Well structured.	5	5
	Document standards	-Good programming practice adopted (e.g. Excellent use of comments). -Standard notation used (e.g. use of UML for class diagrams)	5	5
	Writing style	-Clear and concise technical prose used.	5	5
TOTAL			**15**	**15**

9.2.2. Completeness

Two out of the three documentation components are given, thus documentation can be said to be 67% complete. In other words, the degree of completeness of the documentation is 0.67.

Table 9. Measure for Completeness of Documentation

Documentation Component	**Provided**	**Not Provided**
System Specification	✓	
An architectural design document		✓
The program source code	✓	

9.2.3. Availability of legal terms and conditions

The legal terms and conditions for use of the entire text are available; therefore a value is **1** assigned.

9.2.4. Amount of documentation

Prima-facially the amount of documentation provided is small. For amount of documentation, a value 5 on a scale of 1 – 5 is assigned (with 5 representing the smallest possible amount of documentation). Therefore, the value for amount of documentation becomes **1**.

9.3. Aggregating The Values Into The Reusability Calculation Model

To calculate the reusability of the sample payroll component, the composite metrics viz., *Mai*, *Port, Doc, Gen,* and *Und*; for the five attributes are first calculated, and then aggregated into the reusability calculation model:

$R_C = w_1.Mai + w_2.Port + w_3.Doc + w_4Gen + w_5Und$

$w_1 = 0.2$ $w_2 = 0.1$ $w_3 = 0.4$ $w_4 = 0.1$ $w_5 = 0.2$

Mai = 0.5 (Component independence + Ease of modification and debugging) => 0.5(0.67 + 0.83) = 0.75

Port = Component independence => 0.67

Doc = 0.25(0.67 + 1 + 1 + 1) = 0.92

Gen = Level of generalization => 0.58

Und = 0.5 (Component independence + Cohesiveness) => 0.5(0.67 + 1) =0.84

Therefore:

$R_c = 0.2\ (0.75) + 0.1(0.67) + 0.4(0.92) + 0.1(0.58) + 0.2(0.84) =$ **0.811**

9.4. Interpretation

The *Reusability* for the sample payroll application is **0.811**, compared to the maximum value of 1. Therefore it can be concluded that reusability for the component is relatively high (i.e. the reusability of the component is at 81%).

10. 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 Reusability is inevitable; if effective reuse is to be achieved. We reviewed some research works on reusability measurement, in order to understand the current state of research on OO software reusability measurement: where we reviewed and presented a number of reusability 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 reusability. We also noted in this paper that, in OO software, factors that influence the reusability attributes are related with several OO design features, like inheritance, coupling etc, which can be measured using OO metrics. Thus, OO reusability measurement requires a thorough understanding of how various OO design features influence the reusability factors.

In this paper, we have proposed a novel reusability measurement framework for OO software that considers three measurable features (i.e. Inheritance, coupling and cohesion) as the determinants for OO component reusability. However there might be other several design features like, polymorphism, information hiding etc. that may influence components reusability. Future research works should examine how other features influence reusability, and present metrics for quantifying these features. Moreover, automating every aspect of the proposed model should be the focus of future research works, in order to reduce the intellectual effort required to use the model, and facilitate its implementation in practice.

References

AL-Badareen, A. B., Selamat, M. H., Jabar, M. A., Din, J., & Turaev, S. (2010). Reusable Software Components Framework. Advances. *in Proc Communications, Computers, Systems, Circuits and Devices*, (pp. 126-130). Puerto De La Cruz, Tenerife.

Budhija, N., Singh, B., & Ahuja, S. P. (2013). *Detection of Reusable Components in object Oriented Programming Using Quality Metrics*. 3 (1).

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

Chawla, S., & Nath, R. (2013). *Evaluating Inheritance and Coupling Metrics*. 4 (7).

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

Chidamber, S. R., & Kemerer, C. F. (1991). Towards a Metrics Suite for Object Oriented Design. *ACM Conference. OOPSLA* (pp. 197-211). Phoenix: ACM.

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.

Deitel, P., & Deitel, H. (2012). *Java: How to Program* (Vol. 9th ed). Prentice Hall.

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.

Frakes, W. B., & Kang, K. (2005). Software Reuse Research: Status and Future. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING , 31 (7), 529-536.

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

Gill, N. S., & Sikka, S. (2011). Inheritance Hierarchy Based Reuse & Reusability Metrics in OOSD. *International Journal on Computer Science and Engineering (IJCSE)*, 3 (6), 2300-2309.

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

Ilyas, M., & Abbas, M. (2013). Role of Formalism in Software Reusability's Effectiveness. *International Journal of Database Theory and Application* , 6 (4), 119-130.

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

Návrat, P., & Filkorn, R. (2005). A Note on the Role of Abstraction and Generality in Software Development. *Journal of Computer Science* , 1 (1), 98-102.

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

Rawat, M. S., Mittal, A., & Dubey, S. K. (2012). Survey on Impact of Software Metrics on Software Quality. *International Journal of Advanced Computer Science and Applications* (IJACSA) , 3 (1), 137-141.

Sametinger, J. (1997). *Software Engineering with Reusable Components*. Berlin: Springer-Verlag.

Sandhu, P. S., Kaur, H., & Singh, A. (2009). Modeling of Reusability of Object Oriented Software System. *World Academy of Science, Engineering and Technology* , 162-165.

Sharma, A., & Dubey, K. S. (2012). Comparison of Software Quality Metrics for Object-Oriented System. *International Journal of Computer Science & Management Studies (IJCSMS)* , 12 (Special Issue), 12-24.

Singh, G., Singh, D., & Singh, V. (2011). A Study of Software Metrics. *International Journal of Computational*

Engineering & Management , 22-27.

Sommerville (2001). *Software Documentation*. Retrieved from: http://www.literateprogramming.com/documentation.pdf

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

Vliet, H. V. (2000). *Software Engineering Principles and Practice* (2nd Editions ed.). Wiley and Sons.

Washizaki, H., Yamamoto, H., & Fukazawa, Y. (2003). A Metrics Suite for Measuring Reusability of Software Components. *International Software Metrics Symposium*, (pp. 211-223).

Westfall, L. (2005). 12 Steps to Useful Software Metrics, *the Westfall team*: Retrieved from: http://www.westfallteam.com/Papers/12_steps_paper.pdf

The IISTE is a pioneer in the Open-Access hosting service and academic event management. The aim of the firm is Accelerating Global Knowledge Sharing.

More information about the firm can be found on the homepage:
http://www.iiste.org

CALL FOR JOURNAL PAPERS

There are more than 30 peer-reviewed academic journals hosted under the hosting platform.

Prospective authors of journals can find the submission instruction on the following page: http://www.iiste.org/journals/ All the journals articles are available online to the readers all over the world without financial, legal, or technical barriers other than those inseparable from gaining access to the internet itself. Paper version of the journals is also available upon request of readers and authors.

MORE RESOURCES

Book publication information: http://www.iiste.org/book/

Recent conferences: http://www.iiste.org/conference/

IISTE Knowledge Sharing Partners

EBSCO, Index Copernicus, Ulrich's Periodicals Directory, JournalTOCS, PKP Open Archives Harvester, Bielefeld Academic Search Engine, Elektronische Zeitschriftenbibliothek EZB, Open J-Gate, OCLC WorldCat, Universe Digtial Library , NewJour, Google Scholar