Hindawi Publishing Corporation
ISRN Software Engineering
Volume 2013, Article ID 276105, 14 pages
http://dx.doi.org/10.1155/2013/276105

Research Article

Extension of Object-Oriented Metrics Suite for Software Maintenance

John Michura, Miriam A. M. Capretz, and Shuying Wang

Department of Electrical and Computer Engineering, Faculty of Engineering, The University of Western Ontario, London, ON, Canada N6A 5B9

Correspondence should be addressed to Miriam A. M. Capretz; mcapretz@uwo.ca

Received 2 December 2012; Accepted 22 January 2013

Academic Editors: Y. Malaiya, A. Rausch, and C. Rolland

Software developers require information to understand the characteristics of systems, such as complexity and maintainability. In order to further understand and determine characteristics of object-oriented (OO) systems, this paper describes research that identifies attributes that are valuable in determining the difficulty in implementing changes during maintenance, as well as the possible effects that such changes may produce. A set of metrics are proposed to quantify and measure these attributes. The proposed complexity metrics are used to determine the difficulty in implementing changes through the measurement of method complexity, method diversity, and complexity density. The paper establishes impact metrics to determine the potential effects of making changes to a class and dependence metrics that are used to measure the potential effects on a given class resulting from changes in other classes. The case study shows that the proposed metrics provide additional information not sufficiently provided by the related existing OO metrics. The metrics are also found to be useful in the investigation of large systems, correlating with project outcomes.

1. Introduction

Software metrics have been used to solve different problems such as predicting testing complexity [1], identifying errors [2], and promoting modularity [3]. The metrics proposed by Chidamber and Kemerer [4], now referred to as the CK metrics, have become well known and widely accepted by the software engineering community. The CK metrics can be used to measure some characteristics of OO systems such as classes, message passing, inheritance, and encapsulation. On the other hand, the software maintenance phase requires that changes are made to the existing system. Although the existing metrics, such as the CK metrics, can be used to predict outcomes during software maintenance, such as effort and defects, they do not provide sufficient information regarding the difficulty in implementing such changes, as well as the potential effects of those changes. It would also be beneficial if information is provided regarding the interaction of classes in an OO system in order to predict behavioral changes in those classes during maintenance. Therefore, it is necessary to develop new metrics for software maintainers to better understand the complexity of classes as well as the potential effects of changing classes.

In order to develop new metrics, it is necessary to determine the weaknesses present in the existing metrics. Weaknesses include the loss of information when two classes are said to be coupled and the simultaneous measurement of multiple attributes. The coupling relationship also does not disclose information regarding its direction or magnitude. It is therefore imperative to identify attributes that are relevant in determining the difficulty in implementing changes as well as the effects of such changes. However, only identifying such attributes is not sufficient. It is important that such attributes are quantified in order for them to be effective. As a result, a suite of metrics is required to measure the relevant class attributes.

In this paper, we define the influence relationship, identify class attributes, and develop a proposed suite of metrics used for maintaining OO systems. The key contributions of the research presented in this paper are as follows.

(i) The complexity of a class has been shown to predict the effort required to test and maintain it [4].

Consequently, the difficulty in implementing changes onto a class will be determined by its complexity. Three attributes will be identified and measured by the proposed set of complexity metrics: method complexity, method diversity, and complexity density. These attributes are associated with the difficulty in implementing changes onto a class.

(ii) In addition, the effect of changes will be determined by the use of instance variables and methods of a given class in other classes. A relationship between two classes, influence, will be defined. The influence relationship will be used by a set of proposed metrics to measure the effects of making changes to particular classes. Two attributes will be defined and measured by the proposed set of metrics: impact and dependence. Impact is used to determine the potential effects of changes in a given class on the behavior of the system, while dependence is used to determine the potential change in behavior in a given class that results from changes in other classes.

(iii) The metrics proposed in this research will be used in conjunction with the CK metrics to provide additional information regarding the maintenance of OO systems.

The remainder of this paper is organized as follows. In Section 2, a literature review outlines various established metrics. The CK metrics are discussed as well as research conducted to investigate them. Section 3 identifies and defines attributes associated with classes and presents a suite of metrics intended to complement the CK metrics discussed in Section 2. Results used to determine the effectiveness of the new metrics are presented in Section 4. Section 5 presents the conclusions and future work.

2. Literature Review

The goal of this section is to provide an overview of well-adopted works in software metrics. Although widely used metrics already existed, the arrival of the object-oriented (OO) approach required the development of new metrics. The majority of the established metrics are not designed for the OO paradigm [5]. As a result, they are not sufficient in measuring properties of OO systems. Some established metrics, such as cyclomatic complexity, are also thought to be insufficient in desirable measurement properties [6], lack theoretical basis [7], and are implementation dependent [8]. On the other hand, the CK metrics present a suite of metrics for OO systems that are claimed to be theoretically sound, contained desired measurement characteristics, and are empirically validated [4]. The suite comprises six metrics: Weighted Methods per Class (WMC), Coupling Between Objects (CBO), Depth of Inheritance Tree (DIT), Number of Children (NOC), Response For a Class (RFC), and Lack of Cohesion in Methods (LCOM).

There have been many studies conducted to investigate the value of the CK metrics. The metrics have often been associated with product results such as fault-proneness [9, 10], quality [11–15], complexity measurement [16], and project results such as maintenance effort [17, 18]. We summarize five improvements upon the CK Metrics as follows.

(1) Improving upon WMC. A criticism directed at WMC is that the metric is ambiguously defined [19]. It is the sum of the complexities of the methods in the class. However, the complexities can either be unity (equal to one) or some static complexity measure such as cyclomatic complexity as assumed in this research [18]. The value of WMC is dependent on how complexity is defined. Li and Henry [18] solved this problem by defining a metric that counts the number of methods declared in the class, called Number of Methods (NOM). Li later modified NOM to count only methods that are accessible by other classes [20]. This metric is called Number of Local Methods (NLM). Both metrics have their advantages and disadvantages. NOM is a better measure of the size of the class while NLM is a better measure of the potential effect the given class can have on other classes. Li also proposed another metric that would measure the overall complexity of the class [20]. Class Method Complexity (CMC) is equal to the sum of the complexities of the methods in the class. This differs from WMC in that the complexities used in calculating CMC cannot be defined as unity.

(2) Improving upon coupling. CK's formulation of coupling has also been the subject of scrutiny. CK states that two classes are coupled if at least one method in one class uses a method or an instance variable of the other class. The metric they devised to measure coupling is CBO, a count of all classes the particular class is coupled to. Li and Henry's Message Passing Coupling (MPC) metric addresses both coupling strength and coupling direction [18]. The MPC of a class is equal to the number of messages sent out from the class. A possible shortcoming of MPC is that it assumes that instance variables are not accessed by other objects. Although the direct access of another class instance variables is considered questionable programming practices, it should not be ignored. It is often used to access constants defined within classes. As a result, it is possible that changes made to instance variables may affect other classes. Li's Coupling Through Message Passing (CTM) [20] also addresses both coupling strength and direction, but does not address instance variables in the count of message passing.

(3) Improving upon inheritance. Researchers have addressed the inheritance metrics. Li states that the DIT metric is flawed in that it does not consider cases where a class is inheriting from multiple roots [20]. She addresses this problem with the metric Number of Ancestor Classes (NAC). NAC is a count of all classes the given class inherits from. It is more accurate in determining the number of classes that can change the behavior of the given class. Li also states that CK's NOC is flawed by only considering classes that directly inherit from the given class. The

Number of Descendent Classes (NDC) is a count of all classes that inherit from the given class. Li's inheritance metrics are an accurate measure of a class involvement in inheritance. However, changes in ancestor classes can affect the behavior of classes not belonging to the same inheritance tree.

(4) Improving upon RFC (Response for a Class). The RFC metric can also be criticized in its measurement of multiple attributes [21] and is found to be highly correlated with CBO [22]. RFC intends to measure the number of different possible methods that can be executed by the class in response to a message. As a result, RFC is a function of the number of methods in the class as well as the number of different methods called within methods of the class. The criticism directed at the metric is resolved by instead using a combination of other metrics such as Li and Henry's NOM and MPC metrics.

(5) Improving upon LCOM (Lack of Cohesion in Methods). Hitz and Montazeri [23] have commented that values obtained from LCOM are difficult to explain as the value of the metric is influenced greatly by the number of methods the class contains. Chae et al. [24] also have commented that methods such as accessors (methods used to obtain the value of a class instance variables) and mutators (methods used to alter the value of a class instance variables) increase the value of the metric though they are considered sound programming practices. Many cohesion metrics have been developed by researchers to resolve the issues concerning LCOM [25, 26]. Such solutions include the exclusion of special methods [24] and changing the equation [14].

3. Software Metrics

The key contributions of this research are the definition of complexity and influence, the identification of class attributes related to maintenance, and the proposal of a suite of metrics used to measure such attributes.

3.1. Class Complexity and Influence Relationship. The focus of this research is on changes to an existing software during the maintenance phase of the software's life cycle. The attributes described in this research are aimed at providing an understanding of the difficulty, as well as the effects, of making changes to an existing system.

(i) *Complexity* is used to quantify the difficulty associated with implementing changes during the maintenance phase. Further, *complexity* is defined as the difficulty in understanding abstractions of software such as design documents and implementation code.

(a) *Class complexity* refers to the amount of effort required to understand a class. A class that is more complex will require more effort to maintain.

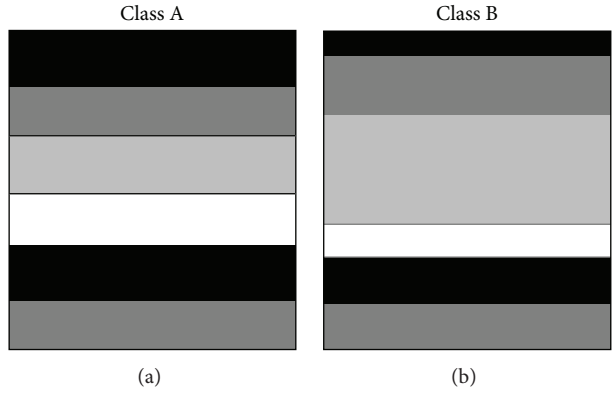

Figure 1: Method diversity.

(b) *Method complexity* refers to the amount of effort required to understand the method. The method complexity of a class refers to the amount of effort required to understand each of its methods.

(ii) *Complexity density* is defined as the proportion of the class complexity not resulting from methods with minimal complexity.

(iii) *Method diversity* is defined as the differences in complexities between methods within the same class. Classes with similar methods will have low method diversity while classes with very different methods will have higher method diversity. Figure 1 shows an example of two classes with different method diversities. The two classes, Class A and Class B, are represented by two rectangles. Methods are represented by subsections of the rectangles. Class complexities are represented by the areas of the rectangles. The complexities of methods are represented by the areas of the subsections. Class B is said to have more method diversity than Class A as its methods are less consistent with regard to complexity.

(iv) *Influence* is a unidirectional relationship between two classes. A class is said to have influence on the other if it contains methods or instance variables that are used in the other. The class containing the used instance variables or methods is referred to as the influencing class whereas the class using the methods or instance variables is referred to as the influenced class. Influence is measured by finding the number of instances that a method in the influenced class uses a method or instance variable declared in the influencing class. Therefore, the influence strength of a class is the average magnitude of the influences it has on other classes. The received influence strength of a class is the average magnitude of the influences other classes have on the given class. The sequence diagram in Figure 2(a) shows that the Dealership class uses two methods in the SalesAssociate class. This is shown in Figure 2(b) as the SalesAssociate class has an influence of two on the Dealership class.

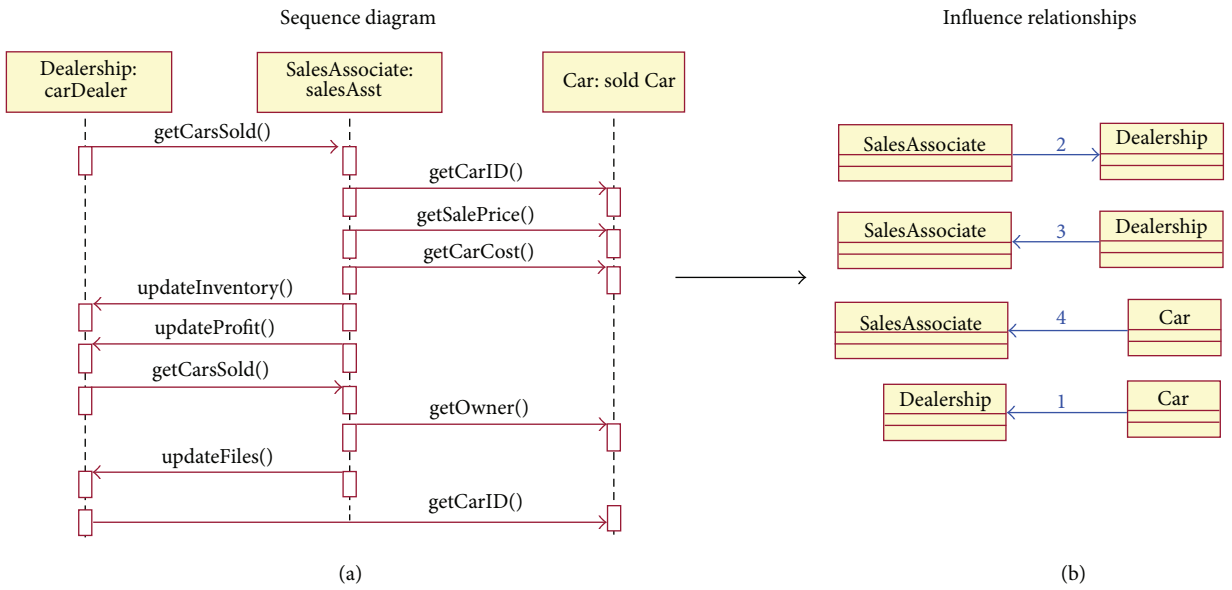

Figure 2: Influence relationship.

(v) *Impact* is defined as a class potential to alter the behavior of other classes when it is changed. Impact is a function of the class influence on other classes. Figure 3 shows the examples of impact of three classes: SalesAssociate, Dealership, and Car. The impact of a class can be measured by the summation of its influences. The SalesAssociate class has an impact of two since it has an influence on the Dealership class of two. The Dealership class has an impact of three since it has an influence of three on the SalesAssociate class. The Car class has an influence of four on the SalesAssociate class and one on the Dealership class. Therefore, its impact is equal to five.

(vi) *Dependence* is defined as a class potential to change in behavior when other classes are changed. It is used to quantify the possible effects associated with implementing changes during maintenance. The dependence of a class can be measured by the summation of influences on it. Figure 4 shows examples of dependence. Figure 4(b) shows the dependence of two classes: SalesAssociate and Dealership. The Dealership class has a dependence of three since the Car class has an influence on it equal to one and the SalesAssociate class has an influence on it equal to two. The SalesAssociate class has a dependence of seven since the Car class has an influence on it equal to four and the Dealership class has an influence on it equal to three. The Car class is said to have no dependence since it is not influenced by any other classes.

3.2. Proposed Metrics. In order to measure the attributes described in Section 3.1, a suite of metrics is developed to quantify the attributes. The suite is comprised of four subsets of metrics: complexity metrics, impact metrics, dependence metrics, and inheritance metrics.

3.2.1. Complexity Metrics. The purpose of the complexity metrics is to complement CK's WMC metric. Since WMC is a function of two different class attributes (number of methods and class complexity), other metrics may be required to fully understand its results. The proposed complexity metrics consist of three metrics: Mean Method Complexity (MMC), Standard Deviation Method Complexity (SDMC), and Proportion of Nontrivial Complexity (PNC).

Definition 1. Mean Method Complexity (MMC) is a measure of a class method complexity

$$\text{MMC} = \frac{\sum c_i}{n}, \tag{1}$$

where $1 \leqslant i \leqslant n$, c_i is the cyclomatic complexity of the class ith method, and n is equal to the number of methods in the class.

MMC serves as an indicator of the complexity of the class methods. MMC is similar to Etzkorn's [27] Average Method Complexity (AMC). The difference between the two metrics is their interpretation of c_i. AMC specifies c_i as the complexity of the method according to any static complexity measure whereas MMC specifies c_i as the method's cyclomatic complexity. The purpose of this metric is to determine if a class WMC value is a result of low-complexity methods or high-complexity methods. High MMC is an indicator of classes comprised of methods with high cyclomatic complexity.

Definition 2. Standard Deviation Method Complexity (SDMC) is to measure the method diversity of a class. Consider

$$\text{SDMC} = \sqrt{\frac{\sum (\text{MMC} - c_i)^2}{n-1}}, \tag{2}$$

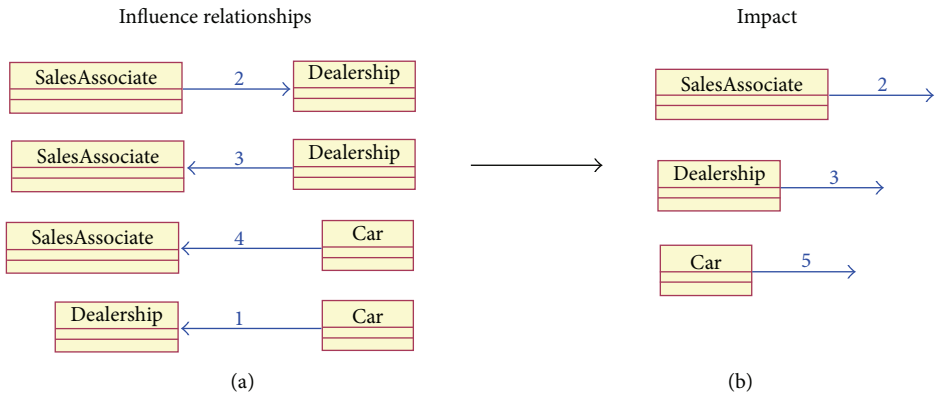

Figure 3: Influence relationships to impact.

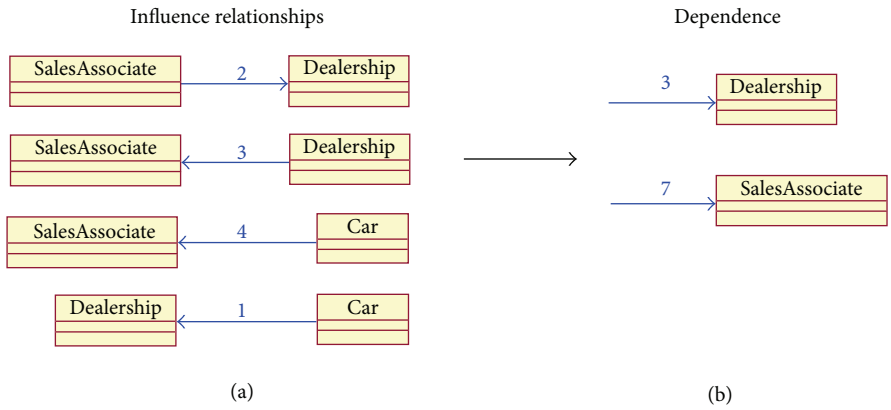

Figure 4: Influence relationships to dependence.

where $1 \leqslant i \leqslant n$, c_i is the cyclomatic complexity of the class ith method, and n is equal to the number of methods in the class.

A class with low SDMC implies similarly complex methods. A class with high SDMC indicates varying method complexities.

Definition 3. Proportion of nontrivial complexity (PNC) measures the class complexity density as

$$\text{PNC} = 1 - \frac{T}{\text{WMC}}, \tag{3}$$

where T is equal to the number of trivial methods in the class, and WMC is the CK's WMC metric that is the sum of complexities of local methods in the class. A method is said to be trivial if its cyclomatic complexity is equal to one. Examples of trivial methods are accessor and mutator methods, as well as empty methods.

A low PNC implies a large proportion of trivial methods. Trivial methods require less testing and maintenance as they have the lowest possible cyclomatic complexity. Classes with high PNC indicate a higher percentage of methods that require rigorous testing.

3.2.2. Impact Metrics. The purpose of the impact metrics is to complement CK's CBO metric. The CBO metric is a measure of both the number of classes the given class uses and the number of classes that use the given class. The impact metrics focus on classes that use the given class in order to determine and understand the potential effect a class may have on other classes. The proposed impact metrics consist of two metrics: Class Impact (CI) and Mean Nonzero Influence (MNI).

Definition 4. Class Impact (CI) measures the potential effect of making changes in the given class. Consider

$$\text{CI} = \sum I_i, \tag{4}$$

where $0 \leqslant i \leqslant N$, I_i is the class influence on class I, and N is equal to the number of other classes in the system.

CI serves as an indicator of the magnitude of the potential effect that a change in the given class will have on the entire system. Classes with high CI should be tested carefully as it implies a greater potential for affecting the behavior of other classes.

Definition 5. Mean Nonzero Influence (MNI) is a measure of influence strength. Consider

$$\text{MNI} = \frac{\text{CI}}{M}, \tag{5}$$

where M is the number of classes that the given class is influencing. If M is equal to zero, then MNI is valued at zero.

Using only CI may give an incomplete view of the impact of particular classes. The class with high CI may have a large influence on a small number of classes or a small influence on a large number of classes. MNI is an indication of which case a particular class belongs to. MNI serves as an indicator of the magnitude of the effect that a change in the given class will have on the classes it is influencing.

3.2.3. Dependence Metrics. Like the impact metrics, the purpose of the dependence metrics is to complement CK's CBO metric. Whereas the impact metrics focus on the classes that use the given class, the dependence metrics focus on the classes that are used by the given class. The purpose of the dependence metrics is to determine and understand the potential effect other classes may have on a given class. The dependence metrics consist of two metrics: Class Dependence (CD) and Mean Nonzero Dependence (MND).

Definition 6. Class Dependence (CD) is a measure of dependence. Consider

$$\mathrm{CD} = \sum D_i, \tag{6}$$

where $0 \leqslant i \leqslant N$, D_i is the class i's influence on the given class, and N is equal to the number of other classes in the system.

Class dependence measures the potential for a change in another class to create a change in behavior in the given class. CD serves as an indicator of the magnitude of the potential effect the system has on the given class. A class with high CD will likely need to be retested if changes are made to other classes.

Definition 7. Mean Nonzero Dependence (MND) is a measure of the influence strength on the class. Consider

$$\mathrm{MND} = \frac{\mathrm{CD}}{L}, \tag{7}$$

where L is the number of classes that have an influence on the given class. If L is equal to zero, then MND is equal to zero.

Like CI, CD does not provide a complete view of the influence of classes on each other. A class with a high CD may be heavily influenced by a few classes or lightly influenced by a large number of classes. MND solves this by determining the average magnitude of a class influence on the given class. MND serves as an indicator of the magnitude of the effect that a change an influencing class will have on the given class. Classes with high MND will require more testing when changes are made to classes that influence it.

3.2.4. Inheritance Metrics. The purpose of the inheritance metrics is to complement CK's DIT and NOC metrics. The goal of the inheritance metrics is to measure impact and dependence obtained through inheritance. The inheritance metrics consist of two metrics: Inherited Dependence (ID) and Weighted Descendent Impact (WDI).

Definition 8. Inherited Dependence (ID) is intended to measure how easily affected the given class is to changes to classes that influence its ancestors. ID is a measure of dependence obtained through inheritance and defined as

$$\mathrm{ID} = \sum \mathrm{CD}_i, \tag{8}$$

where $0 \leqslant i \leqslant N$, CD_i is the class dependence of the class ith ancestor, and N is equal to the number of ancestors of the given class.

A class with high ID will require more testing effort, as it is easily affected by changes made in other classes. The cause of change in behavior of this class may be more difficult to trace, as it is not found in the class itself. As a result, classes should not inherit from classes with high ID.

Definition 9. Weighted Descendent Impact (WDI) is a measure of a class impact obtained through inheritance. Consider

$$\mathrm{WDI} = \sum \mathrm{CI}_i, \tag{9}$$

where $0 \leqslant i \leqslant N$, CI_i is the class impact of the class ith descendent class, and N is equal to the number of descendents of the given class.

This metric measures the class potential effect on the other classes through inheritance. Changes to a class with high WDI should be avoided as such changes made in an ancestor class may propagate to the descendent classes and other classes.

4. Validating Proposed Metrics

The goal of this section is to investigate and validate the proposed metrics discussed in Section 3. Section 4.1 discusses the case study used in this research. Section 4.2 discusses the validation plan used for this section, while Section 4.3 presents the results obtained from correlating the proposed metrics with existing metrics. The effectiveness of the proposed metrics is investigated in Section 4.4. Project data is correlated with the proposed metrics in Section 4.5. We also discuss the results in Section 4.6 and limitations of proposed metrics in Section 4.7.

4.1. Case Study. Company L provided three systems, written in Java, that required analysis. System 1 consists of 125 classes and utilizes inheritance in its design. The highest number of ancestors any given class inherited from is seven. System 2 consists of 34 classes and uses inheritance sparingly. No classes in System 2 inherit from more than one other class. System 3 consists of 12 classes and no classes in System 3 inherit from more than one other class.

4.1.1. Metrics Collection. The metrics are collected using a small software tool developed for this research. Values for the following existing metrics are collected:

(i) Weighted Methods per Class (WMC);

(ii) Number of Methods (NOM);

(iii) Number of Local Methods (NLM);

(iv) Coupling Between Objects (CBO);

(v) Message Passing Coupling (MPC);
(vi) Coupling Through Message Passing (CTM);
(vii) Depth in Inheritance Tree (DIT);
(viii) Number of Ancestor Classes (NAC);
(ix) Number of Children (NOC);
(x) Number of Descendent Classes (NDC).

Values for the following proposed metrics are collected:

(i) Mean Method Complexity (MMC);
(ii) Standard Deviation Method Complexity (SDMC);
(iii) Proportion of Nontrivial Complexity (PNC);
(iv) Class Impact (CI);
(v) Mean Nonzero Influence (MNI);
(vi) Class Dependence (CD);
(vii) Mean Nonzero Dependence (MND);
(viii) Weighted Descendent Impact (WDI);
(ix) Inherited Dependence (ID).

The proposed metrics are compared to values of the existing metrics. This allows for the effectiveness of the proposed metrics to be determined. It is also determined how the values of the proposed metrics correlated with project data provided by Company L in order to draw conclusions about the relationships that exist between various class attributes and project outcomes. The values obtained from the proposed metrics can also be used to obtain a better understanding regarding various characteristics of classes present in the systems.

4.1.2. Data Collection. The project data used in this research is collected using the Perforce software configuration management system (SCMS). The SCMS provides change histories along with the description of the changes, the classes involved in the changes, and the associated change ID number. The outputs of the SCMS provide the following project results for each class.

(i) Number of Revisions (REV): the number of revisions is investigated for multiple reasons as the number of revisions provides a clear and quantifiable measure of maintenance effort. It represents the number of times the class is changed.
(ii) Number of Defects (DEF): the number of defects is investigated for similar reasons. It is always beneficial to reduce the number of defects found in classes.
(iii) Number of Corrective Actions (COR): the number of corrections is investigated to determine the frequency to correct mistakes in the classes such as defects. Note that the number of corrections is not necessarily equal to the number of defects. It may take more than one correction to repair a defect as well as it may take only one correction to repair multiple defects. For this reason, it is determined that both defects and corrections are to be investigated. Also the number of revisions and the number of corrective actions differ in that it may require more than one revision to complete one corrective action.

4.2. Validation Plan. Pearson correlations are calculated using the Statistical Package for the Social Sciences (SPSS). The correlation values are an indication of how linearly related two variables are with the lowest and highest possible values being −1.0 and 1.0, respectively. A higher magnitude indicates a stronger linear relationship between two variables. A positive correlation indicates a simultaneous increase or decrease in the values of the two variables. A negative correlation indicates that as the values of one variable increase, the values for the other variable decreased and vice versa. Correlations are considered *significant* if there is a small probability that the correlation is found due to random error. A "*" indicates that the probability that the correlation is found due to random error is less than ten percent. The explanation of random error is beyond the scope of this research. Metrics are considered to be *highly correlated* if the magnitude of the correlation is greater than 0.8.

We perform three validation tasks for the proposed metrics as follows.

(i) The introduction of new metrics requires a series of steps to determine if they are necessary. Therefore it is important to investigate how the proposed metrics correlate with similar existing CK metrics. High correlations between proposed and existing metrics imply that the proposed metrics may not be required. These correlations are investigated in Section 4.3.
(ii) The proposed metrics must also provide additional information at an acceptable frequency. The frequency at which the proposed metrics provide additional information determines their effectiveness. Section 4.4 investigates the effectiveness of the proposed metrics. The values of the proposed metrics are compared to the values of the existing metrics to determine how frequently the proposed metrics are able to provide additional information.
(iii) Once the proposed metrics are shown to provide additional information, it must be shown that such information is useful in solving problems such as the reduction of maintenance effort, identification of defect-prone classes, and reduction of necessary corrective actions. In order to do this, the values obtained from the proposed metrics are correlated with project data. The results from this investigation are discussed in Section 4.5.

4.3. Correlating Metrics. The values extracted from the proposed metrics are correlated with the values extracted from the existing metrics. The purpose of finding these correlations is to provide the likelihood that the proposed metrics are providing additional information. A high correlation (< -0.8 or >0.8) between two metrics implies that only one of the metrics may be necessary in providing information concerning the attribute they are measuring. As a result,

high correlations between proposed and existing metrics need to be justified in order to consider the proposed metric necessary. Note that cells ("—") in Tables 1, 2, and 3 represent cases where a correlation could not be found.

4.3.1. Correlating Complexity Metrics. The proposed complexity metrics MMC, SDMC, and PNC are correlated with existing related metrics WMC, NOM, and NLM as presented in Table 1(a). It shows that high correlations (>0.8) are found between MMC and WMC in System 2 and System 3. However, MMC and NLM are only highly correlated (>0.8) in System 2. The system may have been designed such that classes with a large number of methods required such methods to be more complex. No high correlations (>0.8) are found for SDMC and PNC with the existing metrics.

4.3.2. Correlating Impact Metrics. The proposed impact metrics, CI and MNI, are correlated with CBO. Table 1(b) shows no high correlations between CI and CBO as well as between MNI and CBO.

4.3.3. Correlating Dependence Metrics. The values for MPC and CTM were equal in all three systems provided, as methods belonging to objects created in methods were not used. As a result, only correlations concerning MPC are discussed to avoid redundancy.

Table 1(c) shows that CD was highly correlated (>0.8) with both MPC and CTM. MND was also correlated with both MPC and CTM. This is not unexpected, as CD differs from MPC and CTM mostly from its treatment of instance variables. CD counts the use of instance variables while MPC and CTM do not. Table 1(c) also shows that CD and MPC are significantly correlated (*) in all three systems. The two metrics are also highly correlated (>0.8) in all three systems as CD and MPC only differ in the use of instance variables. CD counts the use of other classes' instance variables as well as methods while MPC only counts the use of methods. The direct use of instance variables is considered as questionable programming practices and therefore does not occur often. This results in the high correlation between the two metrics. Finally, Table 1(c) shows that MND and MPC are significantly correlated (*) in all three systems. As the definitions of CD and MND show that they are proportional to each other, if CD is highly correlated to MPC, it would follow that MND is highly correlated to MPC.

4.3.4. Correlating Inheritance Metrics. From Table 1(d), ID is not highly correlated (>0.8) with either DIT or NAC. This implies that there is a substantial difference between the ID and the existing metrics. Multiple inheritance has not been utilized in the first two systems and therefore the values of DIT are equal to those of NAC. Correlations for the third system could not be found, as inheritance has not been used in the development of the system. Also, Table 1(d) shows that WDI and NDC are significantly correlated (*) in System 1. That system produces a high correlation (>0.8) between the two metrics. As the number of descendents increase, so will the sum of the descendents' impacts. Also note that correlations for the third system could not be found, as inheritance has not been used in the development of the system.

4.4. Effectiveness of Proposed Metrics. The effectiveness of a metric is determined by the frequency at which it provides additional information when compared to other metrics. The additional information can be detected when a case is found where an existing metric cannot correctly measure some class attributes. Therefore, the proposed metric is able to show that the existing metric leads to incorrect conclusions about the attribute. An example is using WMC to measure method complexity because more complex classes generally have more complex methods. The effectiveness of MMC is determined by the frequency at which it can show that the conclusions drawn using WMC are false. Therefore the frequency of disagreement determines the effectiveness of a proposed metric.

4.4.1. Effectiveness of Proposed Complexity Metrics. Table 2(a) shows the results of investigating the effectiveness of MMC, SDMC, and PNC when compared to the existing metrics WMC, NOM, and NLM.

(i) *Effectiveness of MMC compared to WMC, NOM, and NLM—Method Complexity*. Table 2(a) shows that WMC is inaccurate in identifying which of two classes has higher method complexity in 19.1, 10.5, and 9.1 percent of cases in Systems 1, 2, and 3, respectively. It also shows that NOM is inaccurate in identifying which of two classes has higher method complexity in 28.6, 16.2, and 22.7 percent of cases in Systems 1, 2, and 3, respectively. Furthermore, Table 2(a) indicates that NLM is inaccurate in identifying which of two classes has higher method complexity in 31.2, 16.8, and 36.4 percent of cases in Systems 1, 2, and 3, respectively. Although MMC was shown to be correlated with the existing complexity metrics discussed, the results show that those metrics cannot accurately draw conclusions regarding method complexity. Therefore MMC is shown to provide additional information.

(ii) *Effectiveness of SDMC compared to WMC, NOM, and NLM—Method Diversity*. Table 2(a) shows that WMC is inaccurate in identifying which of two classes has higher method diversity in 29.6, 18.9, and 24.2 percent of cases in Systems 1, 2, and 3, respectively. Also, it is observed that NOM is inaccurate in identifying which of two classes has higher method diversity in 38.3, 22.3, and 37.9 percent of cases in Systems 1, 2, and 3, respectively. In addition, NLM is inaccurate in identifying which of two classes has higher method diversity in 40.2, 20.3, and 42.4 percent of cases in systems 1, 2, and 3 respectively. Although SDMC was correlated with the existing complexity metrics in most cases, the results show that those metrics cannot accurately draw conclusions regarding method diversity. Therefore SDMC is shown to provide additional information.

Table 1: Correlations between metrics.

(a) Correlations between complexity metrics

Metric	MMC			SDMC			PNC		
	System 1	System 2	System 3	System 1	System 2	System 3	System 1	System 2	System 3
WMC	0.794*	**0.902***	**0.873***	0.490*	0.436*	0.647*	0.417*	0.500*	0.620*
NOM	0.422*	0.796*	0.615*	0.113	0.429*	0.315	0.348*	0.545*	0.448
NLM	0.312*	**0.888***	0.326	0.037	0.568*	0.129	0.291*	0.531*	0.232

(b) Correlations between impact metrics

Metric	CI			MNI		
	System 1	System 2	System 3	System 1	System 2	System 3
CBO	0.693*	0.660*	0.481	0.277*	0.713*	0.245

(c) Correlations between dependence metrics

Metric	CD			MND		
	System 1	System 2	System 3	System 1	System 2	System 3
CBO	0.346*	0.547*	0.442	0.257*	0.494*	0.435
MPC	**0.992***	**0.939***	**0.995***	0.797*	**0.857***	**0.974***
CTM	**0.992***	**0.939***	**0.995***	0.797*	**0.857***	**0.974***

(d) Correlations between inheritance metrics

Metric	ID			WDI		
	System 1	System 2	System 3	System 1	System 2	System 3
DIT	0.550*	0.579*	—	—	—	—
NAC	0.550*	0.579*	—	—	—	—
NOC	—	—	—	0.336*	0.186	—
NDC	—	—	—	**0.886***	0.186	—

(iii) *Effectiveness of PNC compared to WMC, NOM, and NLM—Method Density*. It is shown in Table 2(a) that WMC is inaccurate in identifying which of two classes has higher method density in 15.5, 18.8, and 23.6 percent of cases in Systems 1, 2, and 3, respectively. Table 2(a) also shows that NOM is inaccurate in identifying which of two classes has higher method density in 27.8, 21.3, and 17.9 percent of cases in Systems 1, 2, and 3, respectively. It indicates that NLM is inaccurate in identifying which of two classes has higher method density in 31.0, 16.3, and 27.4 percent of cases in Systems 1, 2, and 3 respectively. Although PNC was shown to be correlated with the existing complexity metrics discussed in all but two cases, the results show that those metrics cannot accurately draw conclusions regarding complexity density. Therefore PNC is shown to provide additional information.

4.4.2. Effectiveness of the Proposed Impact Metrics. In Table 2(b), the effectiveness of CI when compared to CBO is investigated. CBO is a measure of the number of classes that the given class is coupled to.

The percentages shown in Table 2(b) represent the percentage of classes in each system where coupling is due to the sending of messages (measured by CD) as opposed to the receiving of messages (measured by CI). The CBO metric is not able to distinguish between the sending and receiving of messages therefore drawing false conclusions regarding a class impact. CBO was shown to inaccurately determine that a class has at least some impact and thus CI provides additional information. Table 2(b) shows that CBO falsely identified a class as having at least some amount of impact in 36.0 percent of classes in System 1, 20.5 percent of classes in System 2, and 33.3 percent of classes in System 3.

In Table 2(b), MNI is used to complement CI in order to provide additional information. The definitions of CI and MNI show that MNI is more similar to CI than to CBO. Consequently, MNI is investigated with respect to CI as opposed to CBO. Table 2(b) shows that CI is inaccurate in identifying which of two classes has higher influence strength in 14.5 percent of cases in System 1, 10.6 percent of cases in System 2, and 30.0 percent of cases in System 3. The results show that CI is not sufficient in measuring influence strength. Therefore, MNI is shown to provide additional information.

4.4.3. Effectiveness of Proposed Dependence Metrics. Table 2(c) shows the results from investigating the effectiveness of CD when compared to CBO, MPC, and CTM. CBO measures the number of classes that the given class is coupled to. It also shows CBO can falsely conclude that the class has at least some class dependence. Note that for three investigated systems, MPC and CTM values are equal. This is not necessarily true for all systems as the systems we

Table 2: Effectiveness of metrics.

(a) Effectiveness of complexity metrics

Metric	MMC			SDMC			PNC		
	System 1	System 2	System 3	System 1	System 2	System 3	System 1	System 2	System 3
WMC	19.1	10.5	9.1	29.6	18.9	24.2	15.5	18.8	23.6
NOM	28.6	16.2	22.7	38.3	22.3	37.9	27.8	21.3	17.9
NLM	31.2	16.8	36.4	40.2	20.3	42.4	31.0	16.3	27.4

(b) Effectiveness of impact metrics

Metric	CI			MNI		
	System 1	System 2	System 3	System 1	System 2	System 3
CBO	36.0	20.5	33.3	—	—	—
CI	—	—	—	14.5	10.6	30.0

(c) Effectiveness of dependence metrics

Metric	CD			MND		
	System 1	System 2	System 3	System 1	System 2	System 3
CBO	14.4	14.7	25.0	—	—	—
MPC	18.4	32.4	8.3	—	—	—
CTM	18.4	32.4	8.3	—	—	—
CD	—	—	—	15.8	11.0	0.0

(d) Effectiveness of inheritance metrics

Metric	ID			WDI		
	System 1	System 2	System 3	System 1	System 2	System 3
DIT	5.6	8.8	25.0	—	—	—
NAC	5.6	8.8	25.0	—	—	—
NOC	—	—	—	5.6	11.8	8.3
NDC	—	—	—	5.6	11.8	8.3

investigated do not use methods of objects declared within methods. We also compare MND with CD. MND is used to complement CD in order to provide additional information.

4.4.4. Effectiveness of Proposed Inheritance Metrics. Table 2(d) shows that DIT, NAC, NOC, and NDC falsely identified a class as having at least some amount of ancestor or descendent dependence in all three systems. In addition, although ID and WDI are shown to be correlated with the related existing inheritance metrics in System 1 and System 2, the results show that those metrics cannot accurately draw conclusions regarding ancestor or descendent dependence.

4.5. Correlating with Project Data. In order to determine if the additional information is useful, the results from the proposed metrics are correlated with the project data described in Section 4.1. This will provide an understanding of the relationships that exist between the class attributes measured by the proposed metrics and the following project outcomes: the number of revisions (REV), number of defects (DEF), and number of corrective actions (COR).

4.5.1. Correlating Proposed Complexity Metrics with Project Data. The relationship between complexity and project outcomes is determined by correlating the results of the proposed complexity metrics with project data. In System 1, Table 3(a) shows that MMC, SDMC, and PNC are significantly correlated (*) to REV, DEF, and COR. This implies that, in System 1, classes with higher method diversity and complexity density are likely to require more revisions, contain more defects, and require more corrective actions. A possible explanation for the correlations is that complex methods, classes with methods that are significantly different from each other lead to mistakes which cause revisions, defects, and corrective actions.

In Systems 2 and 3, Table 3(a) shows that MMC, SDMC, and PNC are not significantly correlated (*) to REV, DEF, or COR. This implies that, in both systems, there is no relationship between complexity metrics and number of revisions, defects found, as well as required corrective actions. This could be due to the size of the systems. Systems that consist of fewer classes are less likely to require revisions than larger systems.

Table 3: Correlating metrics with project data.

(a) Correlating proposed complexity metrics with project data

Metric	MMC			SDMC			PNC		
	System 1	System 2	System 3	System 1	System 2	System 3	System 1	System 2	System 3
REV	0.444*	0.218	−0.315	0.190*	0.141	−0.369	0.307*	0.154	0.085
DEF	0.573*	0.218	—	0.394*	0.141	—	0.230*	0.154	—
COR	0.515*	0.218	—	0.307*	0.141	—	0.243*	0.154	—

(b) Correlating proposed impact metrics with project data

Metric	CI			MNI		
	System 1	System 2	System 3	System 1	System 2	System 3
REV	0.229*	0.001	−0.176	0.437*	0.097	−0.207
DEF	0.174	0.001	—	0.436*	0.097	—
COR	0.200*	0.001	—	0.491*	0.097	—

(c) Correlating proposed dependence metrics with project data

Metric	CD			MND		
	System 1	System 2	System 3	System 1	System 2	System 3
REV	0.624*	0.031	−0.038	0.495*	0.098	0.070
DEF	0.745*	0.031	—	0.487*	0.098	—
COR	0.703*	0.031	—	0.477*	0.098	—

(d) Correlating proposed inheritance metrics with project data

Metric	ID			WDI		
	System 1	System 2	System 3	System 1	System 2	System 3
REV	0.376*	−0.047	—	−0.025	0.032	—
DEF	0.249*	−0.047	—	−0.025	0.032	—
COR	0.256*	−0.047	—	0.032	0.032	—

4.5.2. Correlating Proposed Impact Metrics with Project Data. Table 3(b) shows the relationship between impact and project outcomes. In System 1, CI and MNI are significantly correlated (*) to REV and COR. This implies that, in System 1, classes with higher impact or influence are likely to require more revisions and require more corrective actions. A possible explanation is that classes with high impact undergo revision and corrective action in order to repair the defect in classes they influence. This would also explain why there is no significant correlation between CI and DEF.

In addition, in Systems 2 and 3, CI and MNI are not significantly correlated (*) to REV, DEF or COR. This implies that, in both systems, there is no relationship between impact and number of revisions, defects found, and required corrective actions. This could also be due to system size.

4.5.3. Correlating Proposed Dependence Metrics with Project Data. Table 3(c) shows that CD and MND are significantly correlated (*) to REV, DEF, and COR in System 1. This implies that, in System 1, classes with higher dependence and influence strength are likely to require more revisions, contain more defects, and require more corrective actions. A possible explanation is that classes that heavily rely on other classes, as indicated by high dependence, are more difficult to comprehend, leading to mistakes which cause revisions, defects, and corrective actions. This also supports the conclusion offered in Section 4.5.2 regarding CI and MNI. If high dependence classes are likelier to contain defects, they are likely candidates for revisions and corrective actions. Although the classes that are influencing the high dependence classes may not contain defects, they may require revisions, as well as corrective actions, to repair the defects in the high dependence classes. This would account for the significant correlation between CI and REV as well as CI and COR. It would also explain why CI and DEF do not correlate. It is also possible that changes in other classes are introducing errors into high dependence classes. Such errors may be the cause of revisions, defects, and corrective actions.

The low correlation in Systems 2 and 3 implies that in both systems there is no relationship between dependence and number of revisions, defects found, and required corrective actions.

4.5.4. Correlating Proposed Inheritance Metrics with Project Data. Table 3(d) shows the dependence metrics of ID and WDI. In System 1, ID is significantly correlated (*) to REV, DEF, and COR. This implies that, in System 1, classes with higher dependence are likely to require more revisions, contain more defects, and require more corrective actions. It is possible that changes in other classes are introducing errors into high dependence classes. The errors are then inherited by subclasses leading to revisions, defects, and corrective

actions. In System 2, ID is not significantly correlated (*) to REV, DEF, or COR. This implies that, in System 2, there is no relationship between inherited dependence and number of revisions, defects found, or required corrective actions. This could also be due to system size as well as a lack of inheritance in the system. There are no cases of impact or dependence through inheritance in System 3.

There was no significant correlation between WDI and REV, DEF, or COR. The lack of significant correlations may be due to system size and a lack of inheritance in System 2. However, this is not the case for System 1. System 1 consists of 125 classes and utilizes inheritance in many of its classes. This implies that there is no relationship between the impact of a class descendents and project outcomes.

4.6. Discussion. The investigation of the metrics in this section has two key objectives: to investigate the validity of the proposed metrics and to assess the maintainability of the three systems.

Section 4.3 discussed the correlations between the proposed metrics and related existing metrics. Although the proposed metrics are shown to be correlated with the related existing metrics, only a small subset of correlations, with the exception of the dependence metrics, are sufficiently high to raise concerns. High correlations (>0.8) raised concerns as they serve as indications that two metrics may be providing similar information. An explanation is required in cases where two metrics are highly correlated to help ensure that the proposed metrics are providing additional information. In addition, with regard to the dependence metrics, the high correlations are expected as the proposed metric CD mostly differs from MPC and CTM just in their treatment of direct access of other classes' instance variables. It was therefore necessary to investigate the frequency in which instance variables are accessed by other classes.

Section 4.4 revealed that the existing metrics are not sufficient in measuring the attributes that the proposed metrics are designed to measure. The existing metrics are inaccurate in their assessment of such attributes that it can be concluded that the proposed metrics do in fact provide additional information not otherwise available. The results from Section 4.4 show that the frequency at which CD produces different results from MPC and CTM is not negligible. This shows that instance variables are in fact accessed by other classes, although such a practice is not advised. The correlation between CD and MPC can be used as a measure to determine the level of encapsulation in the class. A higher correlation would imply that classes are being encapsulated properly with regards to hiding instance variables.

The results from Section 4.5 show that, for larger systems, the attributes measured by the proposed metrics are in fact related to the project outcomes investigated. The results supported the notion that, for large systems, method complexity, method diversity, and complexity density are related to number of revisions, defects found, and required corrective actions. The results from Section 4.5 also suggested that, in large systems, it is important to pay close attention to the complexity, impact, and dependence of classes as they can lead to revisions, defects found, and required corrective actions. The dependence of ancestor classes is shown to be related to revisions, defects found, and required corrective actions. One point of interest is that the impact of a class descendent classes is not shown to be related to the project outcomes investigated. However, this does not mean that the metric used to measure this attribute should be discarded. It may be valuable to determine which classes have impact through inheritance. Such information may affect decisions concerning changes made during maintenance.

Another important note is that the results obtained from the values of the proposed metrics can be used to understand various attributes of classes present in the system. The values obtained from the proposed complexity metrics will allow the understanding of method complexities, method diversities, and complexity densities of the classes in systems. The values from the proposed impact and dependence metrics will allow better understanding of message passing in systems. The inheritance metrics will show how the classes interact beyond direct message passing and inheritance.

4.7. Limitations. We consider the limitations of proposed metrics as follows.

(i) The attributes identified and the metrics proposed are specific in its purpose: to determine the difficulty and the effects of change. The metrics are designed to provide very specific information. The values for the metrics can only be extracted using source code or very highly detailed design documents. As a result, the attributes may not be available in early phases of a system's life cycle.

(ii) The size of the system may be important in determining the significance of the proposed metrics [28]. The results in Section 4 imply that small systems do not benefit as readily from the metrics as larger systems. It seems advantageous to use system size in conjunction with the proposed metrics. However, the size of the system is often not determinable until implementation therefore restricting the use of metrics to the implementation phase [29]. The project data is also a determining factor in the value of the proposed metrics. A lack in project data may provide poor results and little information may be gained from using the metrics. Also, low variances have shown to cause a difficulty in finding relationships between metric values and project data [30]. More systems should be investigated to further validate the proposed metrics.

(iii) The proposed metrics do not consider the use of classes that are predefined in the programming language in their calculation. It is, however, possible to overwrite such classes and doing so may cause unpredicted changes in the system. It may be worth investigating if the use of such classes should be considered in the metrics.

5. Conclusion

The objective of this research was to identify the class attributes that convey information regarding the maintenance of that class in order to develop a suite of metrics

that measure such attributes. A suite of metrics has been developed to provide additional information that will assist in the maintenance of object-oriented systems. The metrics have been designed to measure attributes that lead to the difficulty in maintaining classes as well as attributes that describe potential effects of class changes. The use of the proposed metrics will enable maintainers to better understand the complexity of classes as well as the potential effects of changing classes.

One possible future work is the modification of existing modeling languages in order to display values associated with the proposed metrics and convey information such as influence relationships between classes. If such work were to be completed, it may be more valuable to be able to easily visualize the influence of classes as well as their complexities. It is also possible that the metrics be applied to methods instead of classes. Further, it is possible that providing information regarding influence at a method level is as useful, if not more useful, than at a class level. An application of the proposed metrics could be the development of predictive models. Neural networks and other data mining techniques may be used to predict project outcomes by using the values obtained from the proposed metrics, such as quality and maintainability.

References

[1] I. Bluemke, "Object oriented metrics useful in the prediction of class testing complexity," in *Proceedings of the 27th Euromicro Conference*, pp. 130–136, 2001.

[2] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[3] T. J. McCabe, "Complexity Measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, 1976.

[4] S. R. Chidamber and C. F. Kemerer, "Metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[5] M. Bundschuh and C. Dekkers, "Object-oriented metrics," in *The IT Measurement Compendium*, M. Bundschuh and C. Dekkers, Eds., pp. 241–255, Springer, Berlin, Germany, 2008.

[6] E. J. Weyuker, "Evaluating software complexity measures," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1357–1365, 1988.

[7] I. Vessey and R. Weber, "Research on structured programming: an wmpiricist's evaluation," *IEEE Transactions on Software Engineering*, vol. 10, no. 4, pp. 397–407, 1984.

[8] T. Wand and R. Weber, "Toward a theory of the deep structure of information systems," in *Proceedings of International Conference Information System*, pp. 61–71, 1990.

[9] V. R. Basili and B. T. Perricone, "Software errors and complexity: an empirical investigatio," *Communications of the ACM*, vol. 27, no. 1, pp. 42–52, 1984.

[10] M. H. Tang, M. H. Kao, and M. H. Chen, "An empirical study on object-oriented metrics," in *Proceedings of the 6th International Software Metrics Symposium*, pp. 242–249, November 1999.

[11] S. Sarkar, A. C. Kak, and G. M. Rama, "Metrics for measuring the quality of modularization of large-scale object-oriented software," *IEEE Transactions on Software Engineering*, vol. 34, no. 5, pp. 700–720, 2008.

[12] Y. Zhou, B. Xu, and H. Leung, "On the ability of complexity metrics to predict fault-prone classes in object-oriented systems," *Journal of Systems and Software*, vol. 83, no. 4, pp. 660–674, 2010.

[13] H. M. Olague, L. H. Etzkorn, S. L. Messimer, and H. S. Delugach, "An empirical validation of object-oriented class complexity metrics and their ability to predict error-prone classes in highly iterative, or agile, software: a case study," *Journal of Software Maintenance and Evolution*, vol. 20, no. 3, pp. 171–197, 2008.

[14] L. C. Briand and J. W. Daly, "A unified framework for coupling measurement in object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 25, no. 1, pp. 91–121, 1999.

[15] L. C. Briand, J. Wüst, J. W. Daly, and D. Victor Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," *Journal of Systems and Software*, vol. 51, no. 1, pp. 245–273, 2000.

[16] F. T. Sheldon and H. Chung, "Measuring the complexity of class diagrams in reverse engineering," *Journal of Software Maintenance and Evolution*, vol. 18, no. 5, pp. 333–350, 2006.

[17] R. Shatnawi, W. Li, J. Swain, and T. Newman, "Finding software metrics threshold values using ROC curves," *Journal of Software Maintenance and Evolution*, vol. 22, no. 1, pp. 1–16, 2010.

[18] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *The Journal of Systems and Software*, vol. 23, no. 2, pp. 111–122, 1993.

[19] N. I. Churcher and M. J. Shepperd, "'Comments on' a metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 21, no. 3, pp. 263–265, 1995.

[20] W. Li, "Another metric suite for object-oriented programming," *Journal of Systems and Software*, vol. 44, no. 2, pp. 155–162, 1998.

[21] T. Mayer and T. Hall, "A critical analysis of current OO design metrics," *Software Quality Journal*, vol. 8, no. 2, pp. 97–110, 1999.

[22] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial use of metrics for object-oriented software: an exploratory analysis," *IEEE Transactions on Software Engineering*, vol. 24, no. 8, pp. 629–639, 1998.

[23] M. Hitz and B. Montazeri, "Chidamber and kemerer's metrics suite: a measurement theory perspective," *IEEE Transactions on Software Engineering*, vol. 22, no. 4, pp. 267–271, 1996.

[24] H. S. Chae, Y. R. Kwon, and D. H. Bae, "A cohesion measure for classes in object-oriented classes," *Software*, vol. 30, no. 12, pp. 1405–1431, 2000.

[25] J. A. Dallal, "Improving the applicability of object-oriented class cohesion metrics," *The Journal of Systems and Software*, vol. 53, no. 9, pp. 914–928, 2011.

[26] K. A. M. Ferreira, M. A. S. Bigonha, R. S. Bigonha, L. F. O. Mendes, and H. C. Almeida, "Identifying thresholds for object-oriented software metrics," *The Journal of Systems and Software*, vol. 85, no. 2, pp. 244–257, 2012.

[27] L. Etzkorn, J. Bansiya, and C. Davis, "Design and code complexity metrics for OO classes," *Journal of Object-Oriented Programming*, vol. 12, no. 1, pp. 35–40, 1999.

[28] K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai, "The confounding effect of class size on the validity of object-oriented metrics," *IEEE Transactions on Software Engineering*, vol. 27, no. 7, pp. 630–650, 2001.

[29] W. M. Evanco, "Comments on 'The confounding effect of class size on the validity of object-oriented metrics'," *IEEE Transactions on Software Engineering*, vol. 29, no. 7, pp. 670–672, 2003.

[30] G. Succi, W. Pedrycz, S. Djokic, P. Zuliani, and B. Russo, "An empirical exploration of the distributions of the Chidamber and Kemerer object-oriented metrics suite," *Empirical Software Engineering*, vol. 10, no. 1, pp. 81–103, 2005.