Utilizing CK Metrics Suite to UML Models: a Case Study of Microarray MIDAS Software

Taysir Hassan A. Soliman¹, Adel El-Swesy², Saddam Hussein Ahmed³
Faculty of Computer and Information Systems,
Assiut University, Egypt.
¹taysirsh@yahoo.com
²sweisy@aun.edu.eg
³sadamhussin2003@hotmail.com

Abstract — Software metrics provide essential means for software practitioners to assess its quality. However, to assess software quality, it is important to assess its UML models because of UML wide and recent usage as an object-oriented modeling language. But the issue is which type of software metrics can be utilized on UML models. One of the most important software metrics suite is Chidamber and Kemerer metrics suite, known by CK suite [1]. In the current work, an automated tool is developed to compute the six CK metrics by gathering the required information from class diagrams, activity diagrams, and sequence diagrams. In addition, extra information is collected from system designer, such as the relation between methods and their corresponding activity diagrams and which attributes they use. The proposed automated tool operates on XML standard file format [10] to provide independence from a specific UML tool. To evaluate the applicability and quality of this tool, it has been applied to two examples: an online registration system and one of the bioinformatics Microarray tools (MIDAS).

Keywords — Object-oriented systems, UML models, Software metrics

I. INTRODUCTION

Measurement enables the organization to improve the software process, assist in planning, track and control the software project, and assess the quality of the software thus produced. The demand for high quality software continues to increase due to our full dependence on using software in different domains. As a result, quality becomes more of a differentiator between products than it ever has been before. It is widely accepted in software engineering that the quality of a software system should be assured from the initial phases of its life cycle.

A number of metrics have been proposed to help assessing the design of a software system like: Lines of Code [24], Comment Percentage [2], number of modules [25], unique subjects count [26], Data Abstraction Coupling (DAC) [27], Knot metric [28], and Complexity metrics (e.g. Cyclomatic Complexity [5] and Halstead Complexity [29]). However, most of existing approaches to measure design metrics like previous ones involve analysis of thousands of source code lines. This provides information to help in improving internal product characteristics after completion of the product. As a result, it is not clear how to apply existing metrics at the early stages of the software development process.

Many metrics for object-oriented software have been proposed in the literature. A study of object-oriented design quality metrics has been proposed by Magnus Andersson and Patrik Vestergren [6]; they performed an extensive study and a comparison of the current OOP quality metrics. Also, they outlined the most important quality models, including CK suite. In conjunction with the available quality metrics, K. Mustafa and R. A. Khan [7] proposed a framework for developing software quality metrics. Their framework acts as a guideline to develop real quality metrics and proves their mathematical validity, utility, and reliability. In addition, Hoda Hosny and Aida Zakaria [8] developed another framework related to aspect-oriented software engineering, where quality metrics had been used in aspect-oriented software engineering. However, one of the difficulties with comparing and evaluating these metrics is in interpreting and understanding their exact definition. For example, when counting methods in a class, should constructors, finalisers/destructors and accessor methods count as ordinary methods? Should methods that are inherited but not defined in a class be included?

With the increasing use of Unified Modeling Language (UML) to model object-oriented systems at early stages of software development process, research is required to investigate how the proposed metrics can be practically measured purely from UML models and prior to the implementation of the system. Being able to measure software metrics accurately from UML models is important because of the following reasons:

- UML is inherently related to the object-oriented paradigm.
- UML describes a system’s structural view.
- UML design models are neutral since it describes the system regardless of the used language to develop the software.
- Quality of the software system can be measured in the early design stages, where changes can be made easily. We can apply the metrics to the final implementation as well as design model to determine which parts do not conform to the design of the system.
Furthermore, Jacqueline A. McQuillan and James F. Power [9] extracted these metrics from OOP designs, especially UML models. However, they did not mention how to implement quality metrics in UML. Recently, A. McQuillan and J. F. Power [16] related classical CK metrics suite to UML models. They used a middle level model to define metrics over Java programs and specified outcome metrics on UML diagrams. In addition, other research has exploited the use of metamodels in UML. El-Wakil et al. propose the use of XQuery as a metric definition language to extract metric data from XMI design documents, specifically UML designs [37]. Harmer and Wilkie, working from a relational schema, express metric definitions as SQL queries over this schema [38]. Baroni et al. have built a library called FLAME (Formal Library for Aiding Metrics Extraction) that uses the Object Constraint Language (OCL) and the UML metamodel as a mechanism for defining UML-based metrics [39]. Goul’ao et al. have utilised this approach for defining component based metrics and used the UML 2.0 metamodel as a basis for their definitions [40].

In the current work, we utilized a number of the CK metrics suite in order to assess quality of UML models, specifically class diagrams, taking metamodels into consideration through XMI files. We applied these metrics suite to two main applications: a registry system and MicroArray software (MIDAS). The paper is organized as follows: section two illustrates an explanation of current working methodology, section three explains experimental results and their interpretation; finally, for future research that we regard as important.

II. METHODOLOGY

Object-oriented methodologies require significant effort early in project life cycle to identify objects and classes, attributes and operations, relationships between objects and classes, encapsulation, inheritance, and polymorphism require designers to carefully structure the design and consider the interaction between objects. The result of this early analysis and design process is a blueprint for implementation. So, we can extract the metrics at this early stage from UML [18] design model, which captures all these aspects. As a result, this information can be a quality indicator of the proposed software. Accordingly, much effort will be saved rather than rewriting the code and helps producing high quality software. In the current work, CK suite is utilized for several reasons:

1. CK suite covers all aspects of object oriented (encapsulation and polymorphism)
2. It was chosen by SATC (Software Assurance Technology Center) at NASA Goddard Space Flight Center [3],[4] and still used widely till now.
3. Much effort was devoted for empirically validating [1],[22],[30] the original CK metrics and linking them to OOD quality attributes.
4. CK metrics was formalized on the Object Constraint Language OCL [23].
5. Most of the other metrics are built upon the original CK metrics suite.
6. CK suite is easy to collect from UML models, and could be computed in the early phases of design although it was not imposed for this purpose.
7. It is easy to lift CK metrics from the code level to the model level [9].
8. CK suite could be kinked to economic variables (productivity, rework effort, and design) to assess practicing managers [31].
9. CK suite proves to be useful in predicting class fault-proneness [32].
10. CK metrics are the most referenced among all other metrics [34].

Although more than 60 UML tools exist , they their own file format [21]. To solve this problem, the metrics extraction algorithm is run on XMI [10] standard format, proposed by the OMG (Object Management Group) and supported by almost every UML design tools. Now, a brief description of the six CK metrics and the UML diagrams used to compute these metrics [17] is as follows:

1. Weighted Methods per Class (WMC):
is equal to the sum of the complexities of each method defined in a class. If we consider the complexity of each method to be unity then the WMC is equal to the number of methods defined within that class, referred to as WMC1. WMC1 metric for a class can be obtained from the class diagrams of a UML model by identifying the class and counting the number of methods in that class. Alternatively, we can consider the complexity of each method to be McCabe’s Cyclomatic complexity [5], which is referred as WMCCc. The activity diagrams clearly contain information relevant to WMCCc. metric for a class.

2. Depth of Inheritance Tree (DIT):
is a measure of the depth of a class in the inheritance tree. It is equal to the maximum length from the class to the root of the inheritance tree. This metric can be computed for a class by taking the union of all the class diagrams in a UML model and traversing the inheritance hierarchy of the class.

3. Number Of Children (NOC):
is the number of immediate descendants of a given class that is the number of classes, which directly inherits from the class. Again, this metric can be measured for a class by taking the union of all class diagrams in a UML model and examining the inheritance relationships of the class.

4. Coupling Between Object classes (CBO):
Two classes are coupled to each other if a method of one class uses an instance variable or method of the other class. An estimate for this metric can be obtained from the class diagrams by counting all the classes to which the class has a relationship with.

5. Response For a Class (RFC):
is a measure of the number of methods that can potentially be invoked by an object of a given class. The number of methods for a class can be obtained from a class diagram but the
number of methods of other classes that are invoked by each of the methods in the class requires information about the behavior of the class. This information can be derived by inspecting the various behavioral diagrams, such as sequence and collaboration in order to identify method invocations.

6. Lack of Cohesion in Methods (LCOM): calculates the LCOM for a given class involves working out, for each possible pair of methods, whether the sets of instance variables accessed by each method have a non-empty intersection. In order to compute a value for this metric, information on the usage of instance variables by the methods of a class is required. This information cannot be obtained from a class diagram.

One can notice that 50% of the metrics extracted only from class diagram, which also contributes with 33% in extracting other metrics. As a result, class diagram is major contributor in metrics extraction [20]. In the next section, two examples are used to illustrate the application of CK metrics on UML models.

III. RESULTS AND DISCUSSIONS

In this section, two examples are explained to illustrate the utilization of CK metrics on UML diagrams: Laboratory certification system and MIDAS software.

A. EXAMPLE (1) LABORATORY CERTIFICATION SYSTEM:

Laboratory Certification System [11] is an online course registration and exam reservation system proposed by Microsoft for UML demonstration. The main objective of the system is to perform online course registration, including displaying course information, checking sessions available, and exam timing table for each session. Its UML diagrams is published by for free and other diagrams are available at Microsoft.com.

Fig. 1, where class diagram for the registration system is illustrated, shows all available classes (including attributes and methods) in the system and the available coupling and inheritance relationships between them. Class diagram used to compute DIT, NOC, CBO and WMC in collaboration with activity digram. Fig. 2 and Fig. 3 show the sequence and activity diagrams for the system, respectively.

In Fig. 2, all the interactions (message calling) between available object classes are shown. Objects are represented in boxes, vertical lines represent objects life time, message calling are represented in solid arrows and return actions are represented in dashed arrows. Sequence diagram used to compute RFC by keeping track of all external methods are being called by a particular class.

Fig. 3 shows UML activity diagram of Laboratory system; it illustrates control flow of RegistrationForm method belonging to RegistrationForm class. activity diagram used to compute WMCcc. WMCcc is calculated as follows:

\[
WMCcc = \text{no. of edges} - \text{no. of nodes} + 2
\]

If a method does not have an activity diagram associated with it, then its cyclomatic complexity is taken to be 1.

![Fig. 1 Laboratory system class diagram](image-url)
Fig. 2 Laboratory system sequence diagram

Fig. 3 Laboratory system activity diagram
Table I illustrates summarized metrics result for Laboratory Certification System; it shows the highest and the lowest encountered values for each of the six CK metrics. In addition, it gives average value for each metric, where average values is computed by summing all the values of each metric for each class in the system and dividing it by the total number of all available classes.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Average</th>
<th>Highest</th>
<th>Lowest</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIT</td>
<td>0.15</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>NOC</td>
<td>0.15</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>WMC</td>
<td>2.15</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>CBO</td>
<td>1.54</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>RFC</td>
<td>2.46</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>LCOM</td>
<td>0.08</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Num. Classes</td>
<td>13</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

By studying the previous results, one can reach the following analysis:

- A first finding is that the NOC value is relatively small. The project almost makes no use of inheritance.
- CBO medium values suggest that things can be factored a little higher CBO values make internal coding problems and complicated future modifications.
- WMC, RFC, LCOM, DIT values indicates qualified designed with low complexity.

B. EXAMPLE (2) MIDAS BIOINFORMATICS PACKAGE:

MIDAS [12] is a Java open source bioinformatics application that allows a user to perform normalization and other statistical data analysis on Microarray data [13], trim the raw experimental data developed at the Institute for Genomic Research[14] (currently part of J. Craig Venter Institute). Unfortunately, developers of the package did not provide any UML models so we had to reverse engineer the available source code and reconstruct the UML diagrams, using ArgoUML v0.28 [15] tool. However, only class diagrams are reversed.

Table 2 illustrates summarized metrics result for MIDAS, it shows the highest , lowest and average values for each of the six CK metrics. From Table 2 one can conclude the following analysis:

- CBO value is constantly zero; it seems that classes tend to be isolated from each other, almost no interaction between them.
- DIT values is almost balanced, three is accepted value for max inheritance tree depth.
- NOC values is little high means that amount of testing will increase.
- RFC high values draw the attention to potential encapsulation problems and increasing class complexity [19].
- WMC high values point out that some algorithmic complexity in function implementation.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Average</th>
<th>Highest</th>
<th>Lowest</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIT</td>
<td>0.21</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>NOC</td>
<td>0.2</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>WMC</td>
<td>5.9</td>
<td>74</td>
<td>0</td>
</tr>
<tr>
<td>CBO</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RFC</td>
<td>5.9</td>
<td>74</td>
<td>0</td>
</tr>
<tr>
<td>LCOM</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Num. Classes</td>
<td>236</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IV. CONCLUSIONS and FUTURE WORK

Extracting metrics from UML design models is becoming a realistic issue. Through this paper, the operation on XMI generated files rather than building extensions to current tools saves much time and effort and provides cross platform results because of the popularity of the XML. This gives us better results than extracting quality metrics from the source code, so we can use this information as initial gauge for the developers to control their designs.

One can find that each of the metrics we are using to asses software quality refers to the individual classes in the software system and not refer to the whole system; this leads to some missing information, so more general quality metrics that evaluates the system as whole unit rather than individual components can be developed. Also, criteria for the metrics should be defined. In other words, acceptable ranges (not thresholds) for each metric will have to be developed based on the effect of the metric on desirable software rather than depending on personal speculation. Although some researches was done to identify such values but most of them gives thresholds [32],[33],[36] rather than ranges while others give results based on specific software analysis [35], which cannot be generalized to all software. Another point we cannot miss is the need to develop extension of the UML models to address the lack of provided information to the metrics extraction algorithm.

REFERENCES

Metrics for Object Oriented Software: an Exploratory Analysis," Transaction on Software Engineering

IEEE Transactions Software


