In Retrospect of UML based Software Size Metrics

Amita Sharma[#], Preety Verma Dhaka, Dr S.S.Sarangdevot[*]**

[#]Assistant Professor,
Dept of CS &IT, I.I.S University, Jaipur
***Research*
[*]Director,
Dept of CS &IT, J.R.N University, Udaipur

***Abstract*—Size is important in the management of software development because it is a reliable predictor of project effort, duration, and cost. UML based Software Size metrics helps in determining not only the size of software at design level but also helps in understanding the behaviour and complexity of design. Many researchers have proposed numerous software size metrics for UML diagram like class diagram, sequence diagram, activity diagram etc. They illustrated metrics with examples and discussed the utility of each. This paper discusses the concept and significance of software size metrics in software development and a brief review of UML based software metrics. The goal of this paper is to describe the reasons behind the origin of the software size metrics.**

***Keywords*— Software Size Metrics, UML, Class Diagram, Sequence Diagram, Activity Diagram.**

I. INTRODUCTION

A key element of any engineering process is measurement. Measures are used to better understand the attributes of the model that we create. Realizing the importance of software metrics, numbers of metrics have been defined for software. Software developers need to explicitly state the relation between the different metrics measuring the same aspect of software. The traditional view of software development takes an algorithmic perspective. In this approach, the main building block of all software is the procedure or function. As requirements change, systems built with an algorithmic focus turn out to be very hard to maintain. Measuring parameters were function and code dependent.

The contemporary view of software development takes an object-oriented perspective. In this approach, the main building block of all software systems is the object or class. Every object has identity, state, and behavior. Thus, the Object-oriented systems have proven to be of value in building systems in all sorts of problem domains and encompassing all degrees of size and complexity [1]. Object Oriented Design needed new quality meter for software parameter estimation.

In today’s scenario quality with size is the main differentiator between various software products. Due to this reason the software designers and developers need valid measures for the evaluation, improvement and validation of product quality from initial stages [2].

Measuring complexity of software products was and still is a widely scattered research project. Estimating quality and complexity at early stages of SDLC helps in better understanding of software and proves to be more reliable product delivery. As it is still cost effective to make changes to the system. As an emerging industrial standard for object-oriented software analysis and design, UML has been widely used in presenting and visualizing software architecture. UML based metrics are trust worthy measurement and prevention techquice form future failure that may occur due to poor quality. These metrics estimates quality, complexity, reliability, cost etc.

Software size is important in the management of software development because it is a reliable predictor of project effort, duration, and cost. Researchers were looking for faster, cheaper, and more effective methods to estimate software size. Many researchers proposed numerous software size metrics for UML diagram like class diagram, sequence diagram, activity diagram etc. They illustrated metrics with examples and discussed the utility of each.

This paper in retrospect the concept and significance of software size metrics in software development. The paper also illustrates the different proposed software size metrics based on UML. The novelty of this paper is to point out the reasons behind the origin, importance and weakness of each Software size metric.

The paper is organized as follows.Section 2 provides a background of software sizing concept and research trends. Section 3 discusses the origin, advantages and challenges in different proposed UML sizing metrics. Section 4 illustrates the empirical analysis observed and section 5 summarizes the conclusions.

II. SOFTWARE SIZE METRICS: CONCEPT AND SIGNIFICANCE

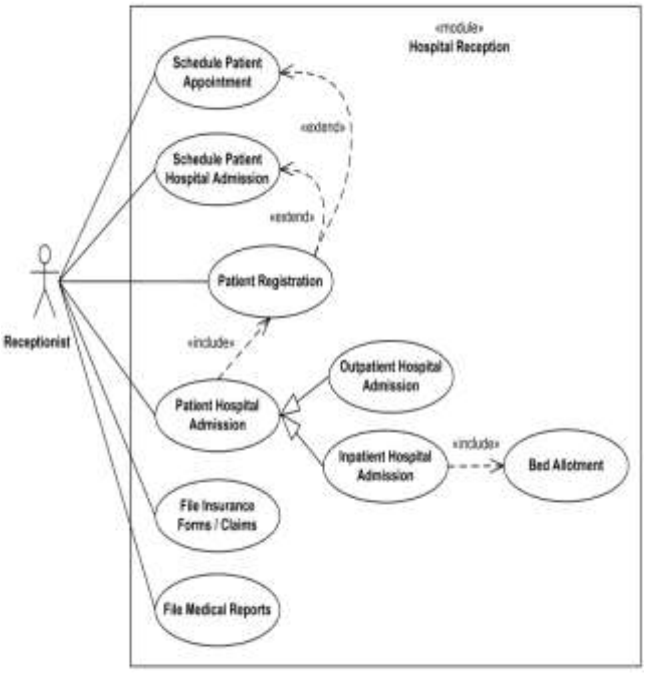
Software size is a key input to all software cost estimation models. An accurate estimate of software size is an essential element in the calculation of estimated project costs and schedules. The fact that these estimates are required very early on in the project makes size estimation a formidable task. Software metrics measure different aspects of software complexity and therefore play an important role in analyzing and improving software quality [12011-14].Traditionally size metrics has been classified as Loc and Cyclomatic complexity. With the invasion of oo systems came into existence the need for oo size metrics. Many researchers have contributed oo size metrics in field of sw engg such as NMIMP (number of methods implemented in a class), NMINH (number of inherited methods in a class), NA (number of attributes in a class), NUMPAR (number of parameters). C & k proposed six oo metrics as WMC, LOC, CBO, DIT, NOC, and LCOM. Out of these six, it can be noted that WMC can be a good indicator for faulty classes and RFC is a good indicator for OO faults.

Average Method Complexity: (AMC): The goal of uml is to provide a standard that can be used by all abject-oriented methods and to select and integrate the best element of precursor notations.. This section gives a brief study of various metrics available for measuring the size of class diagrams. Marches has emphasized on the relationships among classes specifically the inheritance and dependency relationships. The drawback of Marchesi's metric has been that the relationships as association, aggregation, and composition are not considered while being the other essential relationships of a class diagram. So, M.Genero has proposed a group of indicators to measure the complexity of class diagrams.

Further, in order to analyse the architecture complexity, In has defined a metric tree which uses as input the UML diagrams to output eight key indicators. Rufai's Metric gives the different similarity indicators for assessing the similarity between a pair of UML models based on information from their class diagrams. Zhou's Metric discusses the number of relationships, the interaction pattern, and the kinds of relationships among the classes [8]. To distinguish the complexities among the same kind of relationships and to improve the Zhou's Metric, the Kang's Metric have been proposed. Lorenz and Kidd have proposed a group of metrics as design metrics that considers only the static features of the software. Also, for measuring encapsulation, polymorphism and inheritance in the object-oriented scenarios, Brito e Abreu and Melo, has proposed the MOOD Metrics. Moreover, the three C & K metrics that can be applied to UML class diagrams are WMC, DIT and NOC.

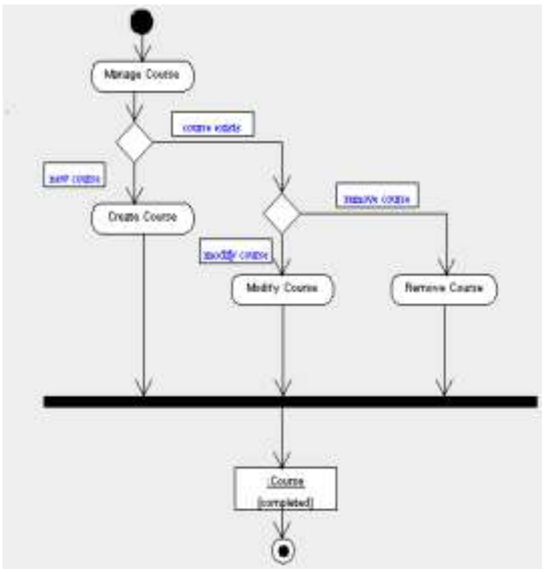
Since Uml has a vast importance in the development of Software designs , so measuring the uml designs has become an important task for the software practitioners for which they use the metrics in terms of size, complexity ,cost estimation ,quality ,etc .A lot of studies have been made so far in concern of class diagrams. Of uml .Considering a case study for "Hospital Management System", that includes several modules providing variety of functions, the Hospital Reception module is being explored via its use case diagram, which uses the duties of hospital receptionist as different use cases as follows:

1. Scheduling the patient's appointments
2. Admitting the patient to the hospital
3. Collecting the information from patient upon patient's arrival and/or by phone
4. Allotment of bed in ward
5. Bed allotment to the patient in the ward
6. Receiving, providing the payment receipt, filing insurance claims, medical reports and maintain a database for the same. .


«module»
Hospital Reception
Schedule Patient Appointment
«extend»
Schedule Patient Hospital Admission
«extend»
Patient Registration
Receptionist
«include»
Outpatient Hospital Admission
Patient Hospital Admission
Inpatient Hospital Admission
«include»
Bed Allotment
File Insurance Forms / Claims
File Medical Reports

Here, we consider a case study of course management system and exploring its activity diagram that basically are used for depicting the various workflows involved in a system. Here the course information is managed by the course administrator and carries out the following activities:

- Checks whether the course exists
- If new course, then moves towards to the "Create Course.
- Else if already existing course, then a check is to be made on what operation are needed such as modifying the course or removing the course
- The course administrator uses the modify operation by, "Modify Course" activity.
- The course administrator uses the remove operation with the help of, the "Remove Course" activity .This completes the activities involved used in the course management system.



Report Based On Analysis

The two case studies aid us to measure the activity and use case diagrams in terms of size. We selected the activity diagram, because it shows the control flow of the actions and it can be modeled on the lowest level of precision (architecture, class, method, and attribute). Also, by using a limited set of the UML diagrams, we solve the fo there is no one to one mapping between behavior descriptions and the source code and behavior does not always have to appear in source code as explicit statements.

The size metrics such as actions gives the number of actions involved and control flow gives the complexity of the activity diagram. Also, for use case diagram metric numass metric gives the number of associations the use case participates in and extpts gives the number of extension points of the use case. A benefit of using the class diagram as our structure diagram is that there is a one to one mapping with the source code. So, we needed another diagram that can be used to model behavior. After some investigation, We need to model the diagrams at the method and attribute level for generation of source code, because the distance to the source code is very little. The same goes for the class diagram, although this diagram has the class level.

Result and Discussion

Estimation of Size plays an important role in the field of software development. Uml as a modelling language has also increasingly emerged as a demanding trend in the software industry. The measurement of the software programs based on Uml design has therefore become the need of time. This paper aimed at summarising the importance of size metrics both traditional and also in concern of Uml. The future work is aimed towards getting the ways to measure the attributes of various Uml diagrams other than the class diagram.

UML Diagrams

Unified Modelling Language (UML) is popular today for capturing requirements and for describing the overall architecture of a software-intensive system. One of the UML constructs is a use case, which graphically depicts the way in which a user will interact with the system to perform one function or one class of functions. Three aspects of use cases can be helpful as inputs to a size estimate: the number of use cases, the number of actors involved in each use case, and the number of scenarios. An actor is a person or system that interacts with the system under consideration; typically, there is one actor per use case, but sometimes there are more. A scenario is a potential outcome from using the software; the number of scenarios can range from one to thousands or millions, depending on the system and its complexity.

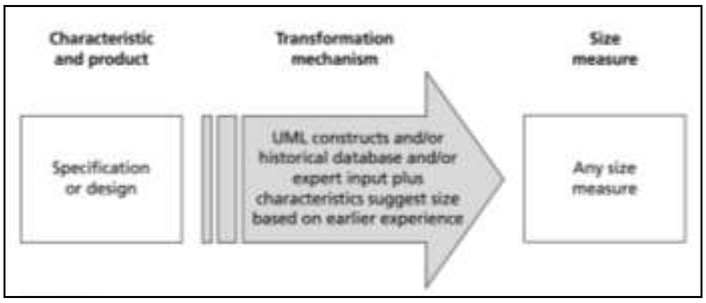

Characteristic and product
Transformation mechanism
Size measure
Specification or design
UML constructs and/or historical database and/or expert input plus characteristics suggest size based on earlier experience
Any size measure

Figure: Characteristic Flow and Transformation Process Applied in UML Designing Tool

This technique can be useful when the size estimate is required after a UML specification is done. It can also be used as a cross-check of another method; if the answers from both methods are similar, the analysts may have more confidence in the result.

Metrics of SDMetric

Metric NumAttr: The number of attributes in the class. The metric counts all properties regardless of their type (data type, class or interface), visibility, changeability (read only or not), and owner scope (class-scope, i.e. static, or instance attribute). Not counted are inherited properties, and properties that are members of an association, i.e., that represent navigable association ends.

Metric NumOps: The number of operations in a class. Includes all operations in the class that are explicitly modelled (overriding operations, constructors, destructors), regardless of their visibility, owner scope (class-scope, i.e., static), or whether they are abstract or not. Inherited operations are not counted.

Metric NumPubOps: The number of public operations in a class. This is same as metric NumOps, but only counts operations with public visibility. It measures the size of the class in terms of its public interface.

Metric Setters: The number of operations with a name starting with 'set'. Note that this metric does not always yield accurate results. For example, an operation settle Account will be counted as setter method.

Metric Getters: The number of operations with a name starting with 'get', 'is', or 'has'. Note that this metric does not always yield accurate results. For example, an operation isolate Node will be counted as getter method.

Metric Nesting: The nesting level of the class (for inner classes). Measures how deeply a class is nested within other classes. Classes not defined in the context of another class have nesting level 0, their inner classes have nesting level 1, etc. Nesting levels deeper than 1 are unusual; an excessive nesting structure is difficult to understand, and should be revised.

Metric IFImpl: The number of interfaces the class implements. This only counts direct interface realization links from the class to the interface. For example, if a class C implements an interface I, which extends some other interfaces, only interface I will be counted, but not the interfaces that I extends (even though class c implements those interfaces, too).

Metric NOC: The number of children of the class (UML Generalization). Similar to export coupling, NOC indicates the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class. A large number of child classes may indicate improper abstraction of the parent class.

Metric NumDesc: The number of descendents of the class (UML Generalization). This counts the number of children of the class, their children, and so on.

Metric NumAnc: The number of ancestors of the class. This counts the number of parents of the class, their parents, and so on. If multiple inheritances are not used, the metric yields the same values as DIT.

Metric DIT: The depth of the class in the inheritance hierarchy. This is calculated as the longest path from the class to the root of the inheritance tree. The DIT for a class that has no parents is 0.Classes with high DIT inherits from many classes and thus more difficult to understand. Also, classes with high DIT may not be proper specializations of all of their ancestor classes.

Metric CLD: Class to leaf depth. This is the longest path from the class to a leaf node in the inheritance hierarchy below the class.

Metric OpsInh: The number of inherited operations. A large number of child classes may indicate ion of the parent class. The number of descendents of the class (UML Counts the number of children of the class, their number of ancestors of the class. parents of the class, their parents, and so on. If multiple inheritances are not used, the metric yields the same values as The depth of the class in the inheritance This is calculated as the longest path from the root of the inheritance tree. The DIT for a class that has no parents is 0.Classes with from many classes and thus is more difficult to understand. Also, classes with high DIT may not be proper specializations of Class to leaf depth. The longest path from the class to a leaf node in the inheritance hierarchy number of inherited operations. This is calculated as the sum of metric NumOps taken over all ancestor classes of the class.

Lines of Codes

This method attempts to assess the likely number of lines of code in the finished software product. Clearly, an actual count can be made only when the product is complete; lines of code are often considered to be inappropriate for size estimates early in the project life cycle. However, since many of the size-estimation methods express size in terms of lines of code, we can consider lines of code as a separate method in that it expresses the size of a system in a particular way.

Function Point Analysis

Function points were developed by Albrecht (1979) at IBM as a way to measure the amount of functionality in a system.

Table: EI Table

FTR's	***DATA ELEMENTS***		
	1-4	*5-15*	*>15*
0-1	*LOW*	*Low*	*Average*
2	*LOW*	*Average*	*High*
3 or More	*Average*	*High*	*High*

Table: Shared EO and EQ Table

FTR's	***DATA ELEMENTS***		
	1-5	*6-19*	*>19*
0-1	*LOW*	*Low*	*Average*
2-3	*LOW*	*Average*	*High*
> 3	*Average*	*High*	*High*

Table: Values for transactions

Rating	***VALUES***		
	EO	*EQ*	*EI*
Low	*4*	*3*	*3*
Average	*5*	*4*	*4*
High	*7*	*6*	*6*

Like all components, EQ's are rated and scored. Basically, an EQ is rated (Low, Average or High) like an EO, but assigned a value like and EI. The rating is based upon the total number of unique (combined unique input and out sides) data elements (DET's) and the file types referenced (FTR's) (combined unique input and output sides). If the same FTR is used on the input and output side, then it is counted only one time. If the same DET is used on the input and output side, then it is only counted one time.

For both ILF's and EIF's the number of record element types and the number of data elements types are used to determine a ranking of low, average or high. A Record Element Type is a user recognizable subgroup of data elements within an ILF or EIF. A Data Element Type (DET) is a unique user recognizable, non recursive field on an ILF or EIF.

Table: Table used to evaluate Rating of EI, EO, EQ

RET's	***DATA ELEMENTS***		
	1-19	*20-50*	*> 50*
1	*Low*	*Low*	*Average*
2-5	*Low*	*Average*	*High*
> 5	*Average*	*High*	*High*

Table: Values for transactions for ILF & EIF

Rating	***VALUES***	
	ILF	*EIF*
Low	*4*	*3*
Average	*5*	*4*
High	*7*	*6*

The counts for each level of complexity for each type of component can be entered into a table such as the following one. Each count is multiplied by the numerical rating shown to determine the rated value. The rated values on each row are summed across the table, giving a total value for each type of component. These totals are then summed across the table, giving a total value for each type of component. These totals are then summed down to arrive at the Total Number of Unadjusted Function Points.

The value adjustment factor (VAF) is based on 14 general system characteristics (GSC's) that rate the general functionality of the application being counted. Each characteristic has associated descriptions that help determine the degrees of influence of the characteristics. The degrees of influence range on a scale of zero to five, from no influence to strong influence. The IFPUG Counting Practices Manual provides detailed evaluation criteria for each of the GSC'S, the table below is intended to provide an overview of each GSC. Rate each factor (Fi, i=1 to14) on a scale of 0 to 5

REFERENCES

[1] S. M. Metev and V. P. Veiko, *Laser Assisted Microtechnology*, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.

[2] J. Breckling, Ed., *The Analysis of Directional Time Series: Applications to Wind Speed and Direction*, ser. Lecture Notes in Statistics. Berlin, Germany: Springer, 1989, vol. 61.

[3] S. Zhang, C. Zhu, J. K. O. Sin, and P. K. T. Mok, "A novel ultrathin elevated channel low-temperature poly-Si TFT," *IEEE Electron Device Lett.*, vol. 20, pp. 569–571, Nov. 1999.

[4] M. Wegmuller, J. P. von der Weid, P. Oberson, and N. Gisin, "High resolution fiber distributed measurements with coherent OFDR," in *Proc. ECOC'00*, 2000, paper 11.3.4, p. 109.

[5] R. E. Sorace, V. S. Reinhardt, and S. A. Vaughn, "High-speed digital-to-RF converter," U.S. Patent 5 668 842, Sept. 16, 1997.

[6] (2002) The IEEE website. [Online]. Available: http://www.ieee.org/

[7] M. Shell. (2002) IEEEtran homepage on CTAN. [Online]. Available: http://www.ctan.org/tex-archive/macros/latex/contrib/supported/IEEEtran/

[8] *FLEXChip Signal Processor (MC68175/D)*, Motorola, 1996.

[9] "PDCA12-70 data sheet," Opto Speed SA, Mezzovico, Switzerland.

[10] A. Karnik, "Performance of TCP congestion control with rate feedback: TCP/ABR and rate adaptive TCP/IP," M. Eng. thesis, Indian Institute of Science, Bangalore, India, Jan. 1999.

[11] J. Padhye, V. Firoiu, and D. Towsley, "A stochastic model of TCP Reno congestion avoidance and control," Univ. of Massachusetts, Amherst, MA, CMPSCI Tech. Rep. 99-02, 1999.

[12] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, IEEE Std. 802.11, 1997.