International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 7, September - 2012

Measuring Software Design Class Metrics:- A Tool Approach

Tincy Rani, Manisha Sanyal, Sushil Garg
CES Dept. RIMT-IET, CSE Dept. GGI-IET, CSE Dept. RIMT-IET.

Abstract

Software design metrics analyzes the structural properties of your UML Model. Use of object oriented measures of design size, coupling and complexity to increase system quality and quality assurance effectiveness, find more faults earlier and save development cost and time. The purpose of this paper is to evaluate the class metrics to determine the object oriented design quality of a software system. The Metrics are the well known quantifiable approach to express any attribute. A class is a template from which objects can be created. This set of objects share a common structure and a common behavior manifested by the set of methods. Three class metrics described here measure the complexity of a class using class metrics (numattr, numops, numpubops, noc etc).

1. Introduction: -

Object oriented software development techniques introduce new elements to measure the software complexity, in software development and in final product. The backbone of any software system is its design. Software design metrics tool analyzes the structural properties of UML models. Where UML represents a unification of the concept and notations. The goal is for UML to become a common language for creating models of object-oriented computer software. Object oriented metrics trace errors in design phase before execution. Metrics are used to determine the development, operation and maintenance of software. The software design metric tool is an on-going initiative in the software metrics research group (SMRG) at King Fand University of petroleum and minerals (KFUPM) to develop a software measurement tool that can be used in software quality assurance.[1]

Software design metrics analyzes the structural properties of your UML Model. Use of object oriented measures of design size, coupling and complexity to increase system quality and quality assurance effectiveness, find more faults earlier and save development cost and time.

2. Related work [1]:-

Several commercial as well as open-source object oriented metric tool exit today. Promined commercial ones include Software metrics [2]. Broland together control center (TCC) [3] and Jhawks [4]. Software metric computes a number of metrics on XMI files.

3. Feature of Software design metrics [5]:-

There are many feature of SD Metrics tool which are given below :-

3.1. Comprehensive design measurement SD Metrics ships with a rich set of object-oriented (OO) design measures covering structural properties of design elements from all UML1.x and UML2.0/2.1/2.2/2.3 diagram types. Measure all the import design attributes - size, coupling, complexity and more - at all levels of detail, from the model, subsystem, package level down to classes and operations.

3.2. Design rule checking

Design rules and heuristics automatically check your UML design for completeness, consistency, correctness, design style issues such as dependency cycles, and more.

3.3. Early quality feedback

The later a fault is found in the development process, the more expensive it is to fix. SD Metrics finds problems at the design stage, before they are committed to source code.

3.4. Extensible set of design measures and design rules

You are not restricted to the built-in set of measures and rules. SD Metrics has a flexible and powerful mechanism to define and calculate new design rules.

3.5. Compare designs

Calculate size metrics deltas to quantify the growth in size between two versions of a design, identify parts of the system design that have undergone much change, or evaluate alternative solutions to a design problem.

3.6. Interoperability with UML tools

SD Metrics is designed to work with all UML modeling tools with XMI export.

- SD Metrics supports all XMI Versions 1.0, 1.1, 1.2, 2.0, and 2.1 currently in use.
- Flexible XMI import, configurable to deal with proprietary UML Meta model extensions, and tools that deviate from XMI standards.
- Use SD Metrics with reverse engineering tools that produce XMI files from C++, Java, Delphi, or Smalltalk source code, .NET assemblies, etc., to perform design measurement on such sources.

3.7. Data export

Design measurement data is most effectively used when subjected to powerful statistical analysis procedures. SD Metrics exports measurement data and descriptive statistics in various formats (tab-separated text tables, HTML, OpenOffice.org Calc, and XML for Microsoft Excel XP) for easy import in office applications, spreadsheet software and statistical packages.

3.8. Interactive GUI

With the easy-to-use SD Metrics graphical user interface, you can interactively explore

measurement data, identify outliers, and browse histograms and Kiviat charts.

3.9. Command line interface

The measurement and data export features are also accessible via a command line interface. Automated analysis runs allow integrating SD Metrics in your development environment.

3.10. Supported Platforms

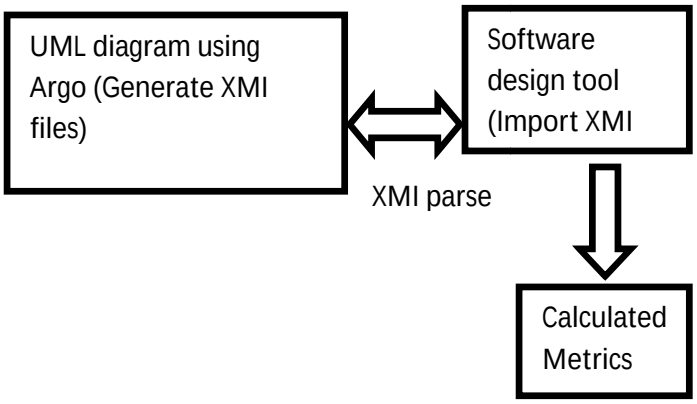
SD Metrics runs on all platforms that support the Java 6 runtime environment (Windows XP/Vista/7, UNIX and Linux)

3.11. Speed

SD Metrics is fast. A 120MB XMI file with hundreds of thousands of design elements is processed in a matter of seconds.

4. Methodology:-

Software design metrics work on java and c# source code or XMI file generated from a UML case Tool in order to extract elementary metric information such as attribute of a class, operations, functions, parameters etc. the working of Software design metrics is depicted in the diagram below.



Software design metrics Work on bases of following steps:-

Step 1:- Parsing project source code or UMl models in XMI format.

Step 2:- After you specified your project files select project and calculate metrics.

Step 3:- Viewing the metrics results in tabular and graphical formats using Histogram and Kiviat Diagram.

Step 4:- Exporting the metric results and reporting in a number of formats.

5. Case Study

In this we calculate the metrics of class (Hospital management system) diagram. This is the part of UML diagram. There are nine classes in this diagram and number of operation and number of attribute in every class. SD metrics calculate the all the operation and attribute of every class automatically, show histograms and Kiviat diagram. Following number of metrics are there that are calculated by SD metrics.

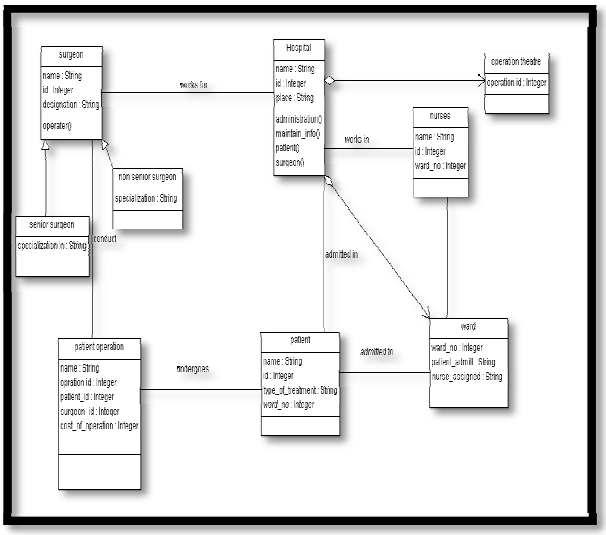


Fig 1. Class diagram of hospital management system

Metrics of class diagram

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 modeled (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.

Same as metric NumOps, but only counts operations with public visibility. 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).

Counts the number of children of the class, their children, and so on.

Metric: **NumAnc**

The number of ancestors of the class.

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 is 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. The longest path from the class to a leaf node in the inheritance hierarchy below the class.

Metric: **OpsInh**

The number of inherited operations.

This is calculated as the sum of metric NumOps taken over all ancestor classes of the class.

Class Metrics calculated by SD metric:-

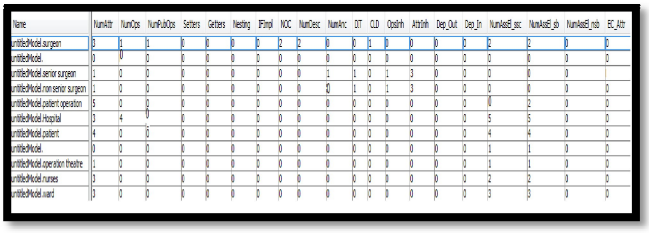


Figure 2:- Class Metrics

Histogram : - The histogram view allows you to quickly browse through histograms and cumulative distribution graphs for all metrics, to get an idea of the distributions of the metrics and visually identify possible outliers:-

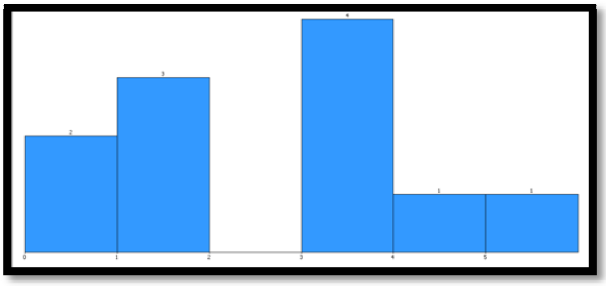


Figure 3:- Histogram of number of attribute in a class

The view also provides descriptive statistics for the metric; including maximum and minimum values, mean, standard deviation, and percentiles.

Kiviat diagram:-

This view shows information for one UML design element at a time. We can browse

through Kiviat diagrams for all elements. The diagram shows the measurement values of all metrics for the selected element. Each axis of the graph represents one metric, ac labeled in the graph.

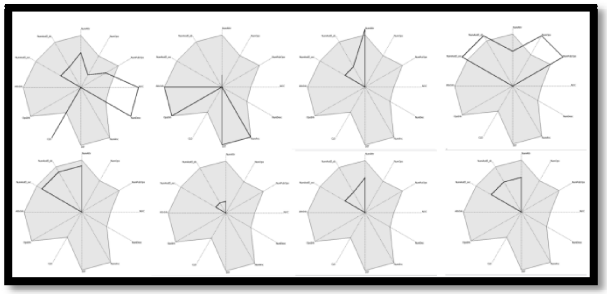


Figure 4:- Kiviat diagram

The thick line connects the metric values of the selected element for each metric on the axes. If the element has many large values, the area enclosed by the thick line will be large. So the size of the enclosed area serves as an indicator of the criticality of the element.

6. Conclusion and future scope:-

UML tool provide supports for working with UML language for the development of various types of information system. In this paper we discuss about SD Metric tool, which calculate the Metrics of all UML diagrams. By analyzing Metrics, a developer can correct those areas of software process that are the cause of software defects. SD Metric tool trace error in design phase before execution. This save time and effort spend for testing in coding and maintenance phase.

In Future work our overall goal to develop a new automated tool which generates the sequence of any UML diagram.

7. References

[1] Jarallah S. Alghamdi, Raimi A. Rufai and Sohel M. Khan," OOMeter: A Software Quality Assurance Tool".

[2] www.sdmetrics.com

[3] Richard C. Gronback, "Using Audits, Metrics and Refactoring in Borland® Together® ControlCenter™", Borland Together White Papers, December 1, 2002.

[4] Virtual Machinery, "Object-Oriented Software Metrics - Introduction and overview", Virtual Machinery Website, http://www.virtualmachinery.com/jhawkmetrics.htm

[5] Tincy.Rani, Suruchi," Measuring software quality attributes - a tool approach.

[6] F. Abreu, M. Goulao, R. Esteves, "Toward the Design Quality Evaluation of Object-Oriented Software Systems", 5th International Conference on Software Quality, Austin, Texas, October 1995.

[7] S. Ambler, "The Elements of UML Style", Cambridge University Press, 2003. A comprehensive collection of style guidelines for the UML. Also available online at www.agilemodeling.com/style

[8] S, Benlarbi, K. El Emam, N. Goel, S. Rai, "Thresholds for Object-Oriented Measures", Proceedings of ISSRE2000, 24-37, 2000.

[9] W. Brown, R. Malveau, H, McCormick, T Mowbray, "AntiPatterns: Refactoring Software, Architectures, and Projects in Crises", Wiley, 1998.

[10] L. Briand, W. Melo, J. Wuest, "Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects", IEEE Transactions on Software Engineering, 28 (7), 706-720, 2002.

[11] L. Briand, J. Wuest, "Empirical Studies of Quality Models in Object-Oriented Systems", Advances in Computers Vol. 59, 97-166, 2002.

[12] Object-Oriented Designs, Empirical Software Engineering: An International Journal, Vol 6, No 1, 11-58, 2001.

[13] S. Chidamber, C. Kemerer, "A Metrics Suite for Object-Oriented Design", IEEE Transactions on Software Engineering, 20 (6), 476-493, 1994.

[14] Object Management Group, "OMG Unified Modeling Language Specification", Version 1.5, OMG Adopted Formal Specification formal/03-03-01, 2003.

[15] Object Management Group, "UML 2.0 Superstructure Specification", OMG Adopted Formal.