ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

An Evolutionary Study of Software Matrices for Object Oriented Products

Anushree Asodiya*, Ashi Jain, Anurag Punde
* Computer Science & Engineering, Acropolis Institute of Technology & Research, Indore, India

Abstract

Increase in the use of technology leads in the development of much new software. As soft ware development is a complex process, it is difficult to measure the software qualities and quantities. To measure the quality of developed and developing soft ware's in terms of efficiency, cost, maintenance, time, scalability etc various matrices are proposed related to various constructs like class, couplings, cohesion, inheritance, information hiding, polymorphism etc .It is often difficult to determine which metric is used in which area. There are 27 matrices proposed for software measurement by various researchers which are used to measure the quality of products. The goal is to achieve the objective, reproducible, quantifiable measurement. The basic purpose of our research is to study various matrices in object oriented software development environment .in this paper we investigate the matrices on class and couplings. They are firstly defined and examined on different code of languages.

Keywords: Software matrices.

Introduction

Software development is not a process to develop software only, but it is also about quality. As we know software is a collection of number of programs, so it is very difficult for a single developer to solve such a large or complex problem occurs during software development Sometimes implementation of software becomes so large and complex too that is not remains as writing codes only but it is also about to follow some guidelines, writing documentation, and writing unit test.[1] But unit test are not enough we have to spot problematic area using matrices. They tells us that our software follow our certain standard. Some matrices tell us about software quality and give us software quality assurance.

Software quality is field of study that describes the desirable attribute of software product. There are two approaches to measure software quality, first is defect management approach and second is quality attribute approach. The defect management approach is used in counting and management of defects. Commonly occurred

Defects are missed or misunderstood requirement, Functional logic, timing, coding, desirable output, maintenance etc. the quality attribute approach is easily explained by fixed quality models like ISO/IEC.

Software assurance is a part of license agreement of most large organizations. **Software assurance** is explained as "to measure of level of quality that software is free from vulnerabilities, which is intentionally or accidentally inserted at anytime during the development of software. The main objective of software assurance is that the processes, procedures, and products used to produce software contains all requirements and standards .SQA ensures the Software Development process, which includes processes such as software design, coding, reviews of codes, software configuration management, testing, release management ,etc.. SQA is all about goals, commitments, abilities, activities, measurements, and verifications.

Software quality can be measure using software matrices. Now the question arise what is matrices? It is defined as "STANDARDS OF MEASUREMENT". for example to measure the physical quantity in physics we use centimeter ,kilo gram ,etc as like as in computer science ,matrices are used to measure the quantitative measure degree to which a system ,system component and process possess a given attribute. There are 22 matrices known based on polymorphism, class coupling, inheritance, etc. In this study we are mainly focusing upon class and coupling matrices. Our study gives you examined study of coding languages based on class and coupling matrices.

Matrices set & pragmatic data collection

The list of matrices that we have chosen for our study is scheduled in table below. The matrices which we have studied are based on class and couplings. A class is to describe one or more objects in OOPS. [1, 2, 3, 4] All classes may consist of variable definitions and methods that can be run by equivalent objects. Coupling is a dominating technique for assessing relationships among different components of software. Coupling is possible, if any type of connection or relationship exists between two components. The matrices which are listed below are briefly explained in our study.

S. No	Name of Metrics	Object Oriented aspects	Sources
1	Response for a class (RFC)	Class	[Chaidamber94]
2	Number of Attributes per Class (NOA)	Class	[Henderson 96]
3	Number of methods per Class (NOM)	Class	[Henderson 96]
4	Weighted Methods per Class (WAC)	Coupling	[Chaidamber94]
5	Couplings between Objects (CBO)	Coupling	[Chaidamber94
6	Data Abstraction Couplings (DAC)	Coupling	[Henderson 96]
7	Message passing Couplings (MPC)	Coupling	[Henderson 96]
8	Coupling Factor (CF)	Coupling	[Harrison98]

Matrices definition & applications

As we stated earlier that matrices are ‘Standards of Measurement’. [5] In development of software, a metric is the measurement of a attribute of a software Performance or efficiency. A metric may be used directly and sometimes as a component in an algorithm. They are not only used in testing of errors but they can also provide variety of information on various aspects like software quality, schedule of software project, functionality of software and result they produce. The matrices chosen for our study are class and couplings matrices.

A. **CLASS MATRICES:** In object oriented programming class matrices are generally used to give the definition of objects. In this section we discussed four matrices. They measure the size in terms of attributes and methods which are integrated in a class.

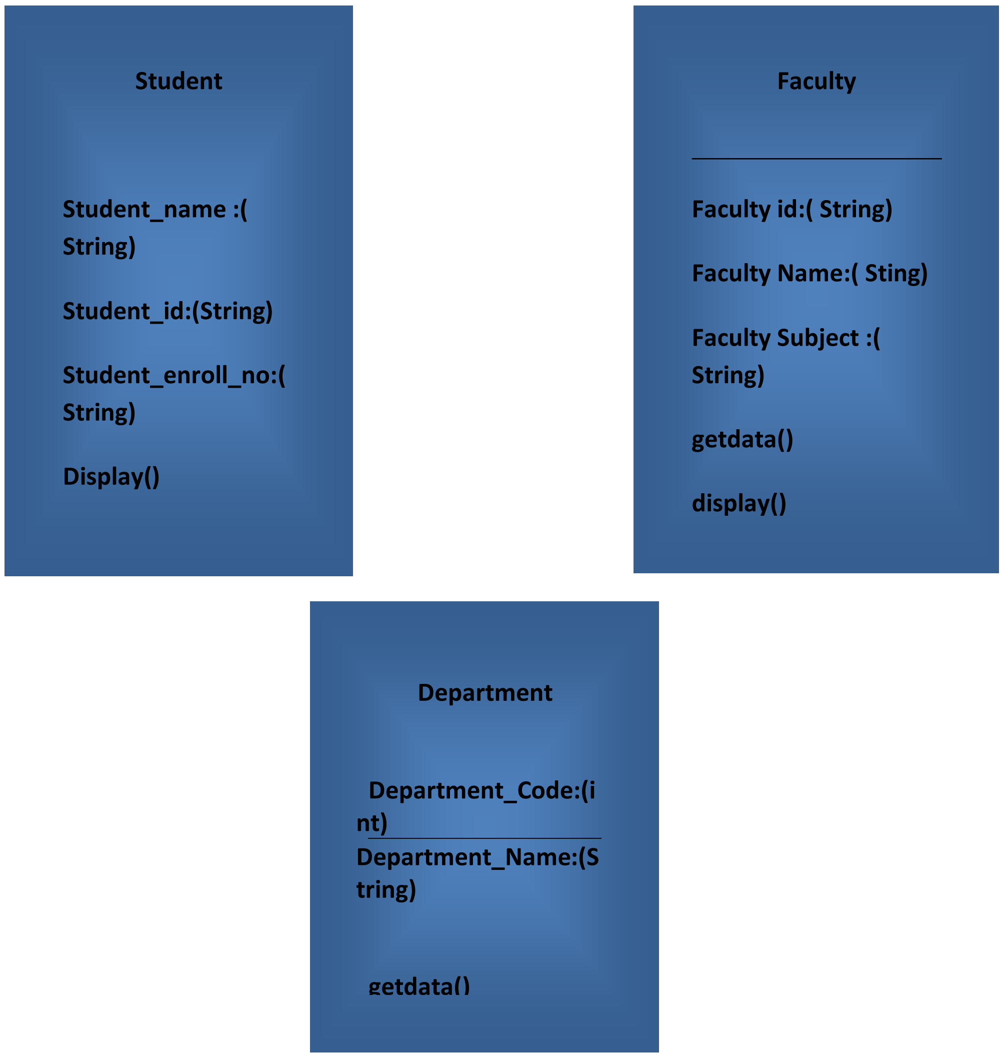
I. **Number of attributes per class (NOA**): It counts the total number of attributes which are defined in a class. Figure 1 shows departmental relationship system. The number of attributes for department class is 2.therefore NOA =2 (department class)

II. **Number of methods per class (NOM):** it counts number of methods that are in defining a class. In figure 1 Number of Methods for Student class is 3.therefore NOM=3.

III. **Weighted methods per class (WMC):** WMC measures the complexity of an individual class. It is the summation of complexities of all the methods in a class. It is marker of the effort that is needed to develop and maintain a class. Consider a class A_1,and methods M_1,M_2…….M_n. suppose C_1,C_2……C_n are the complexities of given class[]

$$\mathbf{WMC=\sum_{i=1}^{n} Ci}$$

When C_1,C_2……C_n=1, then WMC=n, i.e. number of methods in the class. In figure 1



IV. Response for a class (RFC): RFC is the interaction of the class methods with another method. It is the sum of all the methods that can be executed in response to the message received by an object of a class and all discrete methods are involved directly within the class. In normal words RFC is the "Number of Different Methods and Constructors involved by a Class". In addition, all the methods which are inherited are counted, but overridden methods are not

$$RS=Ai \cup_{\text{all } j} \{R_{ij}\}$$

Where Ai= set of all methods in a class and Ri=Rij=set of methods called by class.

In figure 1, class Department has two functions get data and display.

B. COUPLING MATRICES: In simple words coupling is joining of two things together. In development of software, it is a degree by which software components are dependent upon each other. In heavily coupled, components are totally dependent on each other. In loosely coupled, components are independent to each other. Coupling matrices decrease complexity, reduce encapsulation. Potential reuse, maintainability etc.

I. **Coupling between Objects (CBO):** CBO of class is defined as count of number of other classes to which it is coupled. Two classes are called to be coupled if the

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

methods and variables declared in one class are used by the other class. In Figure2 Course class contains declaration of instances of classes' student and college. The value of CBO for class Course is 2and for student and college is zero.

II. **Data Abstraction Coupling (DAC):** Data Abstraction is a procedure for creating new data type suitable for the applications to be programmed. It also provides the ability to create user defined data types called as Abstract Data types (ADTs).

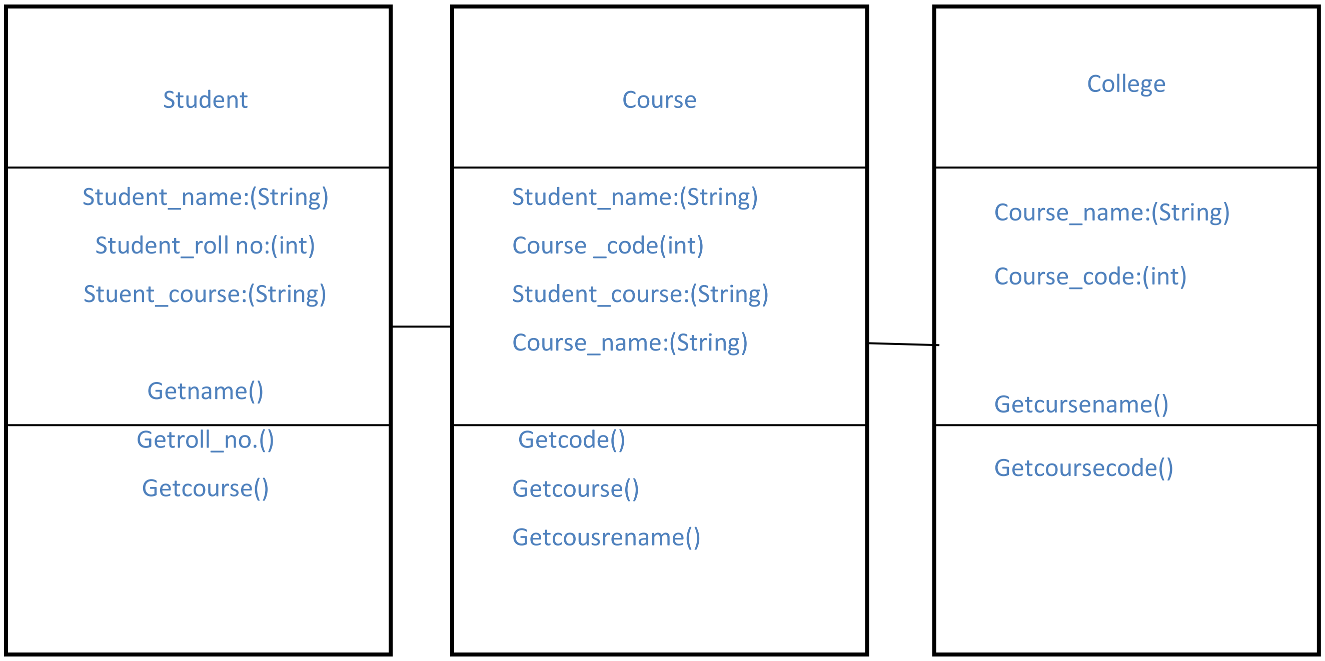
   a. Therefore,
   b. DAC=number of ADTs defined in a class.
   c. In Figure 2 there are four ADTs in class course i.e. student name, student course, course name and course code.

III. **Massage passing coupling (MPC):** MPC is defined as the number of statements defines in a class. Therefore, if two different methods of a class A is used by same method in class B, then MPC=2.In Figure 2,MPC value for class Course is four as methods in class course calls Student_name,Student_course,Course_name ,Course_code.

IV. **Coupling Factor (CF):** CF is defined as the ratio of the maximum probable number of couplings in the class to the actual number of couplings not comes in to inheritance. That is, CF counts the number of inter-class communications. Coupling which is done because of use of inheritance is not integrated in CF because class is heavily coupled with its associates through inheritance.

   a. $CF = \sum\sum [is\_client\ (C_i, C_j)]$
      i. $TC^2 - TC$

Where TC is the total number of class. If no class are coupled then CF=0%.If all class coupled then CF is 100%.



Conclusion & future work

Software development is not a process to develop software only, but it is also about quality. As we know software is a collection of number of programs, so it is very difficult for a single developer to solve such a large or complex problem occurs during software development Sometimes implementation of software becomes so large and complex too that is not remains as writing codes only but it is also about to follow some guidelines, writing documentation, and writing unit test. Software testing is the promising that help us to not only analyzed the product that will develop but also help us to estimate the efficiency of that product before code. With the use of software matrices, the development team can judge the changes to be made in the application. In this work we have use only two of the matrices i.e. class and coupling that help us to judge the complexity of the code, efficiency, maintenance of the code. We can partially assure about the reuse of the code. But if in future we will add two more matrices i.e. inheritance and polymorphism matrices than that can help us in more other sequences like abstraction, reusability, scalability etc.

References

1. S.R.Chidamber and C.F.Kamerer, A metrics Suite for Object-Oriented Design. IEEE Trans. Software ngineering, vol. SE-20, no.6, 476-493, 1994.
2. Shyam R. Chidamber, Chris F. Kemerer, A METRICS SUITE FOR OBJECT ORIENTED DESIGN, 1993
3. B.Henderson-sellers, Object-Oriented Metrics, Measures of Complexity. Prentice Hall, 1996.
4. R.Harrison, S.J.Counsell, and R.V.Nithi, An Evaluation of MOOD set of ObjectOriented Software Metrics. IEEE Trans. Software Engineering, vol. SE-24, no.6, pp. 491-496 June1998.
5. L.Briand , W.Daly and J. Wust, Unified Framework for Cohesion Measurement in Object-Oriented Systems. Empirical Software Engineering, 3 65-117, 1998.
6. L.Briand , W.Daly and J. Wust, A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE Transactions on software Engineering, 25, 91-121,1999.
7. L.Briand , W.Daly and J. Wust, Exploring the relationships between design measures and software quality. Journal of Systems and Software, 5 245-273, 2000.
8. Y.Lee, B.Liang, S.Wu and F.Wang, Measuring the Coupling and Cohesion of an Object-Oriented program based on Information flow, 1995.
9. McCabe & Associates, McCabe Object Oriented Tool User’s Instructions, 1994.
10. McCabe, T.J. "A Complexity Measure." IEEE Transactions on Software Engineering 2, 4 (April 1976): 308-320.