Research Paper: An Approach to Software Metrics i.e. Matrix-X

Er. Chanan Singh *1, Dr. Rajiv Mahajan *2

# Assistant Professor CSE/IT department, BKSJ group, Amritsar, # HOD and Principal GIMET, Amritsar
Punjab Technical University, India, respectively

Abstract

The quantized approach up to managed process of software existence is mainly carried and finished to loyalty under software metrics. This paper will remove all the conceits and conceals about these metrics with all derived demonstrations according to the type of software norms. The categorized search path will correspond a direct view of this field of software engineering to actual get predefined ideal existences. Formal modeling, constituents of software, statistical parts and other quantitative and qualitative means under the aspects like documentation, design, iterative motions, project controllability, software complexity, cost, states up to test and maintenance. All will be in a ring of categorized threshold to go far and far to meet new technologies for future and to enhance actual value of softwares with human friendly environment.

Keywords: quantization, fault, metrics, matrix-x.

1. Introduction

Among all software persistence’s the metrics now exist as the center point. And without their existence it is impossible to think about some reliable software projects to meet all its requirements. But due to such existences of metrics and some relevant techniques and models used to enhance software quality are still not fully sufficient to come up to nearly ideal projects. This is due to lack of their management and disguises about metrics.

And this quantized approach can only be maintained only through proper structured way, we give it name as Matrix-X environment.

You will able to get a unique structured way to get assistance of all the important metrics to come upto ideal softwares, after study of this paper. This Matrix-X process will able to produce quantizing and concentrated software modules. This indicator of the attributes, their randomization, analytical process, reduction of incomplete comparisons, easily measured and abstract view in deep centered way to give up more about complexity metrics with real object oriented approaches. And to give up to a structured approach towards the implementation of such metrics in a real extent.

2. Reason to drive such approach

As we touch that the existing metric norms are not fully able to satisfy the nearly ideal existence of object oriented softwares. All these norms insist to study far and research the actual implemented environment of such and some new metric links in a key structured way to get emphasize by capturing the full domain of the software before and during development. And we give it name as Matrix-X approach.

We can trace the efficiencies, performance and estimate the project through the measurements of software elements affecting to software. And metrics perform the action to provide certain values to its attributes of entities.

3. Base of Matrix – X approach

Before to go for the implementation of such approach lets 'see some base metric norms. i.e.

MSM:- Mean scale per method.

WMC : Weighted Methods per Class (WMC) Assume class C has defined M1, M2,...,Mn methods and M1, M2,...Mn have complexity as C1, C2,...Cn respectively, then WMC = ΣCi(i=1,...,n). The complexity of these methods
can be obtained via traditional methods. If the methods' complexity are 1, then WMC = n. Large WMC will limit the reusability. The larger the WMC is, the more complicated the program is, which means more time and effort should be put on this class.

**MNMA:** - Mean number per method argument.

**MDIT:** - Depth of Inheritance Tree (DIT) This is for class level measurement. DIT of the root of Inheritance tree is zero. Subclasses have DIT one more than its direct super class. The smaller the DIT, the more abstract and simpler the class is. The more a class inherits, the more complicated the class is in design. DIT also shows the number of ancestors of the class. It means the class indirectly or indirectly affected by other classes.

**CBO:** - Coupling Between Objects (CBO) CBO refers to the number of coupling between 2 classes. When a class1 calls the member functions of another class2, then class1 is coupling with class2. The smaller the CBO, the less the class affects other classes, which means the more independent the class is, which also means the less update does the class do to other classes; and therefore the less maintenance effort is needed; the bigger the CBO is, the weaker the reusability the class is; and thereafter the correcting and testing will be more complex. Therefore the better design is trying to avoid big CBO.

**MNOC:** - Number of direct Child (NOC) In inherit tree, count the number of direct subclass of a class. The bigger NOC is, the more affecting the class is. And therefore they should be paid substantial attention when being tested. The more children, the better the class is in reusability. Although the more reusability the better, its super class’s abstraction may become weaker. The number of subclasses creates some potential influence on the super class. The super class should put more rigorous checks for those subclasses.

**LCOM:** - Lack of Cohesion in Object Methods (LCOM) Assume class C has methods: M1, M2, ... , Mn, |Ii| is the set that method Mi is using as instance variables, $P = \{ (Ii, Ij) | Ii \cap Ij = \Phi \}$, $Q = \{ (Ii, Ij) | Ii \cap Ij \neq \Phi \}$. When $|P| > |Q|$ then LCOM = $|P| - |Q|$; otherwise LCOM=0. $Ii \cap Ij = \Phi$ represents Mi does not have relationship with Mj, otherwise they have relationship. LCOM represents the independence between member functions. The smaller the LCOM is, the bigger the cohesion within class becomes and also the more encapsulation it becomes; on the contrary the bigger the LCOM is, the more complex the class design is and therefore the more defects may it contain. If this situation occurs, we can dissolve a big class into smaller classes.

**RFC:** - Response For a Class (RFC) RFC measures the communication (or send messages) within the classes. The bigger the RFC, the more complex the class is and thereafter the test and maintenance will become more difficult.

**NEW RESEARCHED NORMS**

- **C_MULTIPLE**
- **S_MULTIPLE**
- **COR_MULTIPLE**
- **R_MULTIPLE**
- **P_MILTIPLE**
- **A_MUPLTIPLE**
- **CH_MULTIPLE**

4. **Implementation of Matrix-X approach**

Before to suggest the implementation by this new approach its wise to firstly see the software chart with all its elementary approach where the quality or other featurable approaches of the software depends.
Cor_mulrile conatins completeness, accuracy, consistency.

R_multiple contains accuracy, tolerance

P_multiple contains modularity, independence.

Now it is clear from above chart that the new arranged metrics will have the amazing approach.

Actually the main reason to drive such approach is to get the accurate and reliable nature of any software and it is confirmed that this approach will help the professional upto extreme level and much of the complexities will be eradicated through this behavior.

As this technique suggest firstly applying these metrics as such to the software dataset. And after it, count the resultant and predict faults, and the faults detected through this way will also able to direct the links of functions call due to else links supplied in these metrics.

After that the resultant base is clear and through attribute evaluation the overall predictions increases to 4-6% more than previous clustering notions. By this way this approach, Matrix-X will act as the smooth, reliable, time saving and mainly easy to get values about the resultants.

5. CONCLUSION

Due to ever demanding consorts about the new and new metrics and approaches to sharp the concept of fault prediction for object oriented softwares, the Matrix-X approach acts as the superset and ease categorized way to detect faults an also try to eradicate false calls between modules. And the categorized way suspends amazing results and provides us a defined and valuable approach to meet the brilliant way about software engineering.

6. REFERENCES


