Transitive Coupling (TC) and Fault Proneness (FP) in Object Oriented Systems: A New Methodology

Abstract:
Demand for Quality Software has undergone with rapid growth during the last few years. This is leading to an increase in the development of metrics for measuring the properties of software such as Coupling, Cohesion or Inheritance that can be used in early quality assessments. Quality Models that explore the relationship between these properties and quality attributes such as fault proneness, maintainability, effort or productivity are needed to use these metrics effectively. The empirical studies carried out in recent literature concluded that none of the existing metrics particularly coupling and coherence was fever or not effective in ranking the reusability of object oriented components. Based on this conclusion we would like to propose a new coupling metric measuring model that can achieve expected performance to rank the reusability of the components. Since all of the existing metrics relied on binary relations between classes and methods the most or all of the existing models are not able to find the actual coupling metric value between any two classes. The proposed coupling metric measurement model considers transitive relations between methods and classes. We use a set of nine design metrics in our work. The proposed model predicts faulty classes with more than 80% accuracy.

Keywords: Transitive Coupling, Software Metrics, PCA, Fault Proneness and Software Quality.

I. Introduction
Software Metrics have become a key element in several domains of Software Engineering [1]. Various parameters in connection with the software products and processes are assessed through the use of Software Metrics. By applying these metrics to software, it becomes possible to gather numerical data related to it [2]. The information gather can be subjected to analysis, and comparison with past data can be done to ensure an enhancement of the quality of the software. The true value of software metrics comes from their association with important external software attributes such as Testability, Reliability and Maintainability [3]. Metrics are then used to predict software quality attributes for any real time application [4].
A large number of object-oriented (OO) metrics have been proposed in literature [5]. OO Design and Development are popular concepts in today's software development environment. They are often heralded as the silver bullet for solving
software problems, while in reality there is no silver bullet; object oriented development has proved its value for systems that must be maintained and modified[6,7,8]. OO software development requires a different approach from more traditional functional decomposition and data flow development methods. While the functional and data flow approaches commence by considering the systems behavior and/or data separately, object oriented analysis approaches the problem by looking for system entities that combine them. OO analysis and design focuses on objects as the primary agents involved in a computation; each class of data and related operations are collected into a single system entity. They are used to assess different software attributes such as size, complexity, coupling and cohesion [6,10,12,15].

Most of the static metrics treat coupling and cohesion as an intransitive relation that is, they only consider direct coupling between classes and direct similarity between methods. Suppose Class A is coupled to Class B, which is in turn coupled to Class C, but there is no direct coupling between Class A and Class C. Clearly a change in Class C may have consequences for Class A because of the indirect coupling through Class B. The indirect similarity between two methods is similar to the indirect coupling between two classes. We therefore decided to develop alternative coupling metric in the hope of achieving superior performance, we proposed a new coupling metric named as Transitive Coupling (TC). Since all the metrics treated class interaction and method similarity as binary quantities, one obvious step was to develop measures that reflected the extent to which a pair of classes was coupled or a pair of methods resembled each other[7,13,17]. Because none of the measures treated coupling or similarity as transitive relations, we decided that such indirect dependencies should be incorporated into the new metrics. We also take account of the functional complexity of the classes and the methods in the proposed metrics. Empirically explore the relationship between object-oriented design metrics and fault proneness of object-oriented system classes[8,18,20]. We collected data from different open source Java projects which contain more classes. Result of this study shows that many metrics are based on comparable ideas and provide redundant information. It is shown that by using a subset of metrics in the prediction models can be built to identify the faulty classes. The proposed model predicts faulty classes with more than 80% accuracy.

This study have done literature survey about object oriented design metrics. Finally we proposed a new coupling metric named as Transitive Coupling (TC) and we tested the Fault Proneness of the classes under CBO and TC. The main aim of this work is to implement a system which reads all the class files in a project and displays the metric values and the fault proneness of classes under CBO and Transitive Coupling(TC). To obtain this aim, the following objectives are required to be fulfilled -
1) To identify the Software quality metrics, which are evaluate the OO concepts, design structures, and software quality attributes.
2) To identify their accepted ranges for selected software quality OO metrics.
3) To develop the Object Oriented Design Metric Analyzer (OODMA) Software Tool to asses OO metrics and Fault Proneness.

The rest of the paper is organized as follows. In section II, Overview of Object Oriented Design Metrics discussed, section III explores the Proposed Coupling Metrics, section IV Fault Proneness, section V shows the experimental validations, section VI with conclusion and that followed by references.

II. Overview of Object Oriented Design Metrics
Object Oriented Design and Development is an interesting area of current research and many authors have done great deal of work in recent years. In fact Object Oriented Development requires not only a different approach to design and implementation, but also a different approach to software metrics. To produce high quality Object Oriented applications a strong emphasis on design aspects is
highly necessary. Software metrics make it possible for software engineers to measure and predict software processes, necessary resources for a project and products relevant for a software development effort. A software measure provides software engineers with a means of quantifying the assessment of a software product[19]. Measurement can be used throughout a software project to assist in estimation, productivity assessment, quality control and project control. The design of complex software based systems often proceeds with virtually no measurement. The introduction of Object Oriented Methods to software development has changed the process of building and managing software in a profound way. Changes in design and implementation techniques also require new ways of measuring software systems. Design Metrics (Albert Dieter Ritzhaupt, 2004) play an important role in helping developers to understand design aspects of software techniques and, hence, improve software quality and developers productivity[20,24]. In addition, the focus on process has increased the demand for software measures or metrics with which one can manage the process. The need for such metrics is particularly rare, when an organization is adopting a new technology for which a suitable metric design plays a crucial role. We present existing and new software metrics useful in the different phases of the Object Oriented Software Development cycle[21,27,30,35]. The importance of properly defined metrics is of immense importance in the Object Oriented design. If the metrics are properly defined, we can avoid problems that will be more expensive to correct during the latter phases of Object Oriented Software Development[22]. This helps researchers and practitioners better in understanding and selects software metrics suitable for their purposes. In today’s software industry, the aim is to deliver high quality software product to the customers. No doubt that the software quality can make or break a company. So software Quality plays a vital in the software industry. Metrics help validate and calibrate generic models of software productivity and reliability an object oriented program paradigm uses encapsulation, inheritance and polymorphism, and has different program structure than in procedural languages[23]. The design components that are exclusive and define the architecture of an OOD are objects, classes and the relationship between them. Object oriented design is intended to capture the fundamental structure of an object oriented program. Thus a set of components which can help to evaluate, represent and implement an object oriented design should include attributes, methods, objects (classes), relationships and class hierarchies[24]. Software quality must be addressed during the whole process of software development. Measuring software quality in the early stages of software development is the key to develop high quality software[25]. Product quality has some attributes such as Functionality, Usability, Effectiveness, Understandability, Reusability and Maintainability. Large number of software metrics have been proposed in software engineering to measure the quality attributes of the software in early stages[26,29,38]. We can estimate the overall design quality of the system from design information. There are so many models, which can be effectively used in monitoring the quality of software product.

A. Traditional Metrics
In an object-oriented system, traditional metrics are generally applied to the methods that comprise the operations of a class. A method is a component of an object that operates on data in response to messages and is defined as part of the declaration of a class[27,38,39]. The importance of properly defined metrics is of immense importance in the Object Oriented design. If the metrics are properly defined, we can avoid problems that will be more expensive to correct during the latter phases of Object Oriented Software Development[22]. This helps researchers and practitioners better in understanding and selects software metrics suitable for their purposes. In today’s software industry, the aim is to deliver high quality software product to the customers. No doubt that the software quality can make or break a company. So software Quality plays a vital in the software industry. Metrics help validate and calibrate generic models of software productivity and reliability an object oriented program paradigm uses encapsulation, inheritance and polymorphism, and has different program structure than in procedural languages[23]. The design components that are exclusive and define the architecture of an OOD are objects, classes and the relationship between them. Object oriented design is intended to capture the fundamental structure of an object oriented program. Thus a set of components which can help to evaluate, represent and implement an object oriented design should include attributes, methods, objects (classes), relationships and class hierarchies[24]. Software quality must be addressed during the whole process of software development. Measuring software quality in the early stages of software development is the key to develop high quality software[25]. Product quality has some attributes such as Functionality, Usability, Effectiveness, Understandability, Reusability and Maintainability. Large number of software metrics have been proposed in software engineering to measure the quality attributes of the software in early stages[26,29,38]. We can estimate the overall design quality of the system from design information. There are so many models, which can be effectively used in monitoring the quality of software product.
cyclomatic complexity of individual methods can be combined with other measures to evaluate the complexity of the class. Although this metric is specifically applicable to the evaluation of complexity, it also is related to all of the other attributes. Complexity metrics can be used to calculate essential information about constancy and maintainability of software system from source code. It also provides advice during the software project to help control the design. In the testing and maintain phase, complexity metrics provide detail information about software module to identify the areas of possible instability. Cyclomatic complexity (McCabe) can be used to evaluate the complexity of a method[36]. This metric measures the complexity of a method or procedure. The idea is to draw the sequence a program may take as a graph with all possible paths. The complexity is calculated as “connections - nodes + 2” and will give a number denoting how complex the method is. See the following figure. Since complexity will increase the possibility of errors, a too high McCabe number should be avoided

\[
N = 2 - 3 + 2 = 1 \quad N = 6 - 6 + 2 = 2 \quad N = 11 - 8 + 2 = 5
\]

Figure : 1 The McCabe complexity metrics

**Line Count ==> Size/Documentation:**
Various line counts are also applied to methods. These include counting all physical lines of code the number of statements and the number comment lines. Thresholds for evaluating the meaning of size measures may have to vary greatly depending on the coding language. However, since size limitations are based on ease of understanding by the developers and maintainers, routines of large size will always pose a higher risk in attributes such as Understandability, Reusability, and Maintainability. This metric can be used to evaluate all the attributes, but most often is a measure of Understandability, Reusability, and Maintainability.

Example:
for (i = 0; i < 100; i += 1)
    printf("hello"); /* How many lines of code is this? */
In this example we have:
- 1 Physical Line of Code (LOC)
- 2 Logical Lines of Code (LLOC)
(for statement and printf statement)
- 1 comment line

**Comment Percentage:**
The Line Count metric can be expanded to include a count of the number of comments, both on line and stand-alone. The comment percentage is calculated by the total number of comments divided by the total lines of code less the number of blank lines. The SATC has found a comment percentage of about 30% is most effective. Since comments assist developers and maintainers, this metric is used to evaluate the attributes of Understandability and Reusability.

**III. Proposed Coupling Metric**
A well designed component, in which the functionality has been appropriately distributed to its various subcomponents, is more likely to be fault free and will be easier to adapt. Appropriate distribution of function underlies two key concepts of object-oriented design: coupling and cohesion. Coupling is the extent to which the various subcomponents interact. If they are highly interdependent then changes to one are likely to have significant effects on the behavior of others[34]. Hence loose coupling between its subcomponents is a desirable characteristic of a component. If they are highly interdependent then changes to one are likely to have significant effects on the behavior of others. Lots of research effort has been devoted to developing metrics that can be used to provide a numeric assessment of coupling and cohesion from the source code or detailed design. All the measurements discussed so far are static measures; that is they can be derived by static analysis of the source code. However, object-oriented language processing features such as inheritance and polymorphism that may preclude determining precisely which two classes are involved in an interaction until runtime. Consequently, in recent years,
there has been a growing interest in dynamic coupling and cohesion metrics. Such measures are required to collect information about the interaction that occurs as the program is executing[35]. We focus on the static measurements because it did not appear feasible to collect run time information in the context of assessing components.

Limitations
Three limitations of established metrics were identified in earlier approaches. They are

First, most of the static metrics discussed above treat coupling and cohesion as an intransitive relation that is, they only consider direct coupling between classes and direct similarity between methods. Suppose Class A is coupled to Class B, which is in turn coupled to Class C, but there is no direct coupling between Class A and Class C. Clearly a change in Class C may have consequences for Class A because of the indirect coupling through Class B. The indirect similarity between two methods is similar to the indirect coupling between two classes. Although one of the cohesion metrics LCOM3 has suggested extensions to incorporate indirect relationships between methods. It treats indirect and direct cohesion in the same way and cannot numerically specify the indirect and direct cohesion.

Second, both CBO and CF treat coupling between one class and another as an all-or-nothing quantity; classes are either coupled or they are not. No account is taken of the extent to which one class is dependent on another. If a class has many interactions with another, it is much more likely that a change in one will have consequences for the other than would be the case if their coupling rested on a single interaction. All of discussed cohesion measurements also treat cohesive relation between methods as binary feature. All of the metrics lack of quantitative assessments.

Third, none of the coupling metrics takes any account of the functional complexity of the class whose coupling is being assessed. Other things being equal, a class that provides a wide range of functionality is likely to have more interactions with other classes than one that is very simple.

A metric that assesses the extent of coupling between classes should take account of this. El Emam, Benlarbi and Goel [19] have presented evidence that the correlations between fault-proneness and several metrics, including RFC and CBO, disappear if the analysis controls for class size. For cohesion metrics, except RLCOM and TCC, none of other metrics takes any account of the functional complexity of the method. A metrics that is trying to assess the similarity between methods should take account of this. Some researches have suggested that validation studies should therefore control for size of class and method. We have chosen to avoid the need for this by developing coupling and cohesion metrics that are independent of class or method size/complexity.

A. Transitive Coupling (TC)
We begin by regarding any Object Oriented Software System as a directed graph, in which the vertices are the classes comprising the system. Suppose such a system comprises a set of classes C ≡ {C1, C2,...,Cm}. Let Mj ≡ {Mj,1, Mj,2,...,Mj,n} be the methods of the class Cj, and Rj,i the set of methods and instance variables in class Ci invoked by class Cj for j ≠ i (Rj,j is defined to be null). Then the edge from Cj to Ci exists if and only if Rj,j is not null. Thus an edge of the graph reflects the direct coupling of one class to another. The graph is directed since Rj,i is not necessarily equal to Rij. Rj, the set of all methods and instance variables in class Ci invoked by class Cj for j ≠ i (Rj,j is defined to be null). Then the edge from Cj to Ci exists if and only if Rj,j is not null. Thus an edge of the graph reflects the direct coupling of one class to another. The graph is directed since Rj,i is not necessarily equal to Rij. Rj, the set of all methods and instance variables in other classes that are invoked by class Cj, can be defined:

\[ R_j = \bigcup_{1 \leq i \leq m} R_{j,i} \quad (1) \]

The next step is to associate a number with each edge that reflects the extent of direct coupling from one class to another. Clearly it should be larger if the class invokes more of the other’s methods. However, this number should also reflect the fact that a class that invokes many methods has a greater likelihood of invoking methods from any particular class. We therefore define CouplD(i,j), our measure of direct coupling of class Ci to Cj, as
Note that the denominator is the total number of methods invoked by class \( C_i \) and that \( 1 \geq \text{CoupD}(i,j) \geq 0 \). The next step is to include the indirect coupling between classes. Suppose that \( \text{CoupD}(i,j) \) and \( \text{CoupD}(j,k) \) have finite values but that \( \text{CoupD}(i,k) \) is zero. Thus although there is no direct coupling between classes \( C_i \) and \( C_k \), there is a dependency because \( C_i \) invokes methods in \( C_j \) which in turn invokes methods in \( C_k \). Since the strength of this dependency depends on the two direct couplings of which it is composed, a reasonable measure is provided by their product, \( \text{CoupD}(i,j) \times \text{CoupD}(j,k) \). This notion is readily generalised. A coupling between two classes exists if there is a path from one to the other made up edges whose \( \text{CoupD} \) values are all non-zero. The strength of the coupling is the product of all those \( \text{CoupD} \) values. Thus we define \( \text{CoupT}(i,j,\pi) \), the transitive coupling between classes \( C_i \) and \( C_j \) due to a specific path \( \pi \), as

\[
\text{CoupT}(i,j,\pi) = \prod_{e \in \pi} \text{CoupD}(s,t) = \prod_{e \in \pi} \frac{|R_i|}{|R_i| + |M_j|}
\]

where \( e \) denotes the edge between vertices \( s \) and \( t \). Note first that \( \text{CoupT} \) includes the direct coupling, which corresponds to path of length one, and second that, because the \( \text{CoupD} \) values are necessarily less than one, transitive couplings due to longer paths will typically have lower values.

In general there may be more than one path having a non-zero \( \text{CoupT} \) value between any two classes[38]. This raises the question of how their values may be combined to produce an overall measure of the coupling between the classes. A pragmatic approach is to simply select the path with largest \( \text{CoupT} \) value and hence define \( \text{Coup}(i,j) \), the strength of coupling between the two classes, \( C_i \) and \( C_j \) to be:

\[
\text{Coup}(i,j) = \text{CoupT}(i,j,\pi_{\text{max}})
\]

where

\[
\pi_{\text{max}}(i,j) = \arg\max_{\pi \in \Pi} \text{CoupT}(i,j,\pi)
\]

and \( \Pi \) is the set of all paths from \( C_i \) to \( C_j \). Note that, because of the attenuation of coupling over longer paths, where a direct coupling exists, that will typically be value selected for the strength of coupling; \( \text{Coup} \). While there are obvious objections to ignoring the other weaker couplings, the ultimate test of this method is whether it is effective[37]. The results reported later in this paper suggest that this simple approach is adequate.

Having established a measure for the strength of coupling between pairs of classes, the final step is to use this as a basis for a measure of the total coupling of a software system[36]. This is readily achieved by summing all the couplings of all class pairs and dividing by the total number of such pairs. The weighted transitive coupling (WT\( \text{Coup} \)) of a system is thus defined

\[
\text{WT\text{Coup}} = \frac{\sum_{i \neq j} \text{Coup}(i,j)}{m^2 - m}
\]

where \( m \) is the number of classes in the system.

**IV. Fault Proneness (FP)**

When new releases of software are needed to generate it is times consuming task to the testers to test the new generations of software. If testers will know in advance which classes of methods are fault-prone and its fault proneness then they can concentrate on to these classes. It results saving of time. In the proposed model of finding the fault proneness, the fault prone classes are identified by using the OO metrics like weighted methods per class (WMC), Coupling between objects (CBO), Response for class (RFC). The metrics are calculated and matrix is created depending on to the calculated metrics.

**A. Fault Proneness Algorithm**

1. **Step 1:** Start
2. **Step 2:** Read source package
3. **Step 3:** Get all class names and contents
4. **Step 4:** Get all class dependencies
5. **Step 5:** Find the Weighted Methods per Class
6. **Step 6:** Find Response for class
7. **Step 7:** Find Coupling Between the Objects
8. **Step 8:** Construct adjacency matrix for metrics
9. **Step 9:** Get range of matrix to find fault-proneness
10. **Step 10:** If range is between some range
11. **Step 11:** Then Fault-Proneness is low, high, Medium or very high
12. **Step 12:** Print whether FP is high, low, very high, and medium
Step 13: By using metrics values Find Fault-Proneness

Step 14: Stop

CBO is the number of classes coupled to the given class [36]. The coupling may be afferent coupling or efferent coupling. Efferent coupling is the number of classes that the measured class is depended upon and afferent coupling is the number of classes that depends upon the measured class.

After Calculation of WMC, RFC and CBO, I am creating their matrix, where each row represents the class name and column represents their respective value of WMC, RFC and CBO. Then by observing the above matrix we need to set the parameter range for metrics, like Very low, Low, Medium, High and very high. To find fault prone-ness of class we need to set a range to find fault-proneness of different releases no need to set a range of metrics.

Then by using knowledge base we will calculate the very low and very high parameters of the object oriented matrix and then by using centroid method equation, we calculate fault proneness. The centroid method equation used in our fault prone module is:

\[(\text{max} + \text{min}) / 2\]

Consider the following where we are considering the last release of the software, in our case it is third generation.

- From the third generation matrix first we will get the sum of all WMC and CBO of the matrix that is equal to 17.00
- Then gets the average by dividing sum from number of classes that is 17/ 4= 4.25.
- And then we consider average * 2 (that is 8.5=da) as the maximum value for any class. (here value refers to the sum of WMC+CBO of any class).
- Then for the first class value=3 then we did like (3/8.5) then we get its fault ness that is 0.3529
- This will be continuing for all classes.
- And we consider a class as low faulty which gives a value b/w 0 to 25% of da that is 0 to 2.125(8.5*0.25=2.125)
- And we consider a class as Medium faulty which gives a value b/w 26 to 50% of da that is > 2.125 to <= 4.25(8.5*0.5=4.25)
- And we consider a class as High faulty which gives a value b/w 51 to 75% of da that is >4.25 to <=6.375(8.5*0.75=6.375)
- And we consider a class as Very High faulty which gives a value b/w 76 to 100% of da that is >6.375 to <= 8.5(8.5*1.0=8.5)

Figure: Class Diagram for OODMA Software Tool

V. Experimental Validations

We consider 5 Java projects mentioned below that are licensed as GNU open source from various domains are:

Project 1: BLACKDUCKKODERS (http://www.koders.com): 10 version
Project 2: STRAR UML-One of the UML tool to design UML diagrams (http://www.osalt.com/staruml): 5.0 version.
Project 3: OpenOffice Draw 3.0 (http://www.openoffice.org/product/draw.html): 3.0 version
Project 4: InfraRecorder 0.50 (http://infrarecorder.org/): 0.5 version
Project 5: Gimpshop 2.2.11
At first we will run the jar file form command prompt. Then we will get the metric values for all the classes which we provided.

We have columns like class, CA, WMC, DIT, NOC, CBO, NPM, RFC, LCOM and TC. Class column displays the classes name in the application with class name and remaining columns displays values of metrics of a particular metrics. Internally it will create a .txt file with these metric values in root directory from where we are calling the jar file.

After running the jar file it will display the class name and metric values in the command prompt. At the same Internally it will create a metrics.txt file with these metric values in root directory from where we are calling the jar file.

After running the jar file it will create 3 files internally in the root directory ,from where we are calling the jar file.

It will take pcainput.txt file for doing the PCA Analysis. Internally we will do the PCA (Principal Component Analysis) for the above data. i.e. taking all the metric values for all the classes. After calculating PCA it will display the results in a graph.
Figure: 6 Taking the number for extracting the Eigen values

VI. Conclusion

The main goal of this paper is to fulfilled by implementation of the automation tool with more effective coupling metric TC and Fault proneness evaluation for OO environments. The SATC proposed nine metrics for OO systems like LOC, CC, CP, NOC, DIT, WMC, RFC and CBO metrics for assessing five quality attributes - efficiency, complexity, understandability, reusability, testability and maintainability. These metrics uses key concepts of OO designs like methods, classes, coupling, cohesion, messages and inheritance. Inheritance is the main concepts in Object Oriented Programming (OOP), which explains reuse of the code. Since all the metrics treated class interaction and method similarity as binary quantities, one obvious step was to develop measures that reflected the extent to which a pair of classes was coupled or a pair of methods resembled each other. Because none of the measures treated coupling or similarity as transitive relations, we decided that such indirect dependencies should be incorporated into a new metric. We also take account of the functional complexity of the classes and the methods in the proposed metric. Empirically we explored the relationship between object-oriented design metrics and fault proneness of object-oriented system classes. The proposed model predicts faulty classes with more than 80% accuracy.

Acknowledgements

The Authors would be thankful and appreciate Bala. M and the R&D Cell, ECET, Hyderabad for gathering the information and prepare to this Research Paper.

About the Authors:

Amjan Shaik is working as a Professor Department of Computer Science and Engineering, ECET, Hyderabad, India. He has received M.Tech.(Computer Science and Technology) from Andhra University. He has been published and presented good number of Research and Technical papers in International Journals, International Conferences and National Conferences. His main research interests are Software Engineering, Software Metrics, Software Testing, Software Quality and Object Oriented Design.

M. Sudhir Kumar is working as an Associate Professor, Department of CSE at Ellenki College of Engineering and Technology (ECET), Hyderabad, India. He has received B.Tech(CSE) from JNTU Kakinada and M.Tech(SE) from JNTUH Hyderabad. He has been presented and published 2 articles in International Conferences and 1 Research paper in International Journal. His main research interests are Software Engineering, Software Metrics and Information Security.

Md. Riyazuddin is working as an Assistant Professor, Department of Information Technology at Muffakhamjah College of Engineering and Technology (MJCET), Banjarahills, Hyderabad, India. He has received B.Tech (IT) from Kakatiya University, Warangal and M.Tech (CSE) from JNTUH Hyderabad. He has been published 1 Research Paper in International Journal. His main research interests are Data Mining, Cloud Computing, Software Engineering and Networks.

S.V. Achuta Rao is working as a Professor and Head, Department of CSE and IT at VCET, Nunna, Vijayawada, India. He has received M.Tech. (Computer Science and Engineering) from JNTU, Kakinada, India. Presently, he is a Research Scholar of Rayalaseema University (RU), Kurnool, India. He has been Published 6 Research Papers in various International Journals. His main research interests are Bio-
Mohd Zainuddin Naveed is working as an Assistant Professor, Department of Computer Science and Engineering at Muffakhamjah College of Engineering and Technology (MCET), Banjarahills, Hyderabad, India. He has received M.Tech (CSE) from Shadan College of Engineering and Technology, Affiliated to JNTUH Hyderabad. He has been published 1 Research Paper in International Journal. His main research interests are Data Mining, Information Security and Software Engineering.

References:


[38] Set Laboratories, UX Metrics, 1994.


