Volume 3, Issue 7, July 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Estimate Software Quality by Using Rejection Method Approach with Clustering Techniques

Dr. P.K. Suri[1] , Rajni Rana[2]
[1]*Dean, Research and Development, Chairman, CSE/IT/MCA, HCTM Technical Campus, Kaithal, Haryana, India.*
[2]*HCTM Technical Campus, Kaithal, Haryana, India.*

Abstract— Software Quality analysis is one of the major criteria required to discuss to analyze the software life cycle as well as software processes. Software quality has been defined under different parameters. Software risk analysis is one of such criteria required to discuss to identify the software quality. When a Software is planned or being developed according to the type of software as well as the efforts required to develop the software collectively defines the software risk. Such as the availability of the required software, hardware, manpower all are the predictive risk factors. In this work, these all risk factors are defined to analyze the software quality. The proposed work is about to the use of rejection method for improving the software quality

Keywords: Software Risk, Quality, Rejection method, Clustering, Approach, Analysis.

I. INTRODUCTION

A. Software Errors, Faults And Failures

The origin of software failures lies in a software error made by a programmer. An error can be a grammatical error in one or more of the code lines, or a logical error in carrying out one or more of the client's requirements. However, not all software errors become software faults. In other words, in some cases, the software error can cause improper functioning of the software in general or in a specific application. In many other cases, erroneous code lines will not affect the functionality of the software as a whole; in *a part* of these cases, the fault will be corrected or "neutralized" by subsequent code lines. We are interested mainly in the software failures that disrupt our use of the software. This requires us to examine the relationship between software faults and software failures. Do all software faults end with software failures? Not necessarily: a software fault becomes a software failure only when it is "activated" – when the software user tries to apply the specific, faulty application. In many situations, a software fault is never activated due to the user's lack of interest in the specific application or to the fact that the combination of conditions necessary to activate the fault never occurs.

B. Software Quality

Software quality is:
1. The degree to which a system, component, or process meets specified requirements.
2. The degree to which a system, component, or process meets customer or user needs or expectations.

C. Classification of the causes of software errors

As software errors are the cause of poor software quality, it is important to investigate the causes of these errors in order to prevent them. A software error can be "code error", a "procedure error", a "documentation error", or a "software data error". It should be emphasized that the causes of all these errors are human, made by systems analysts, programmers, software testers, documentation experts, managers and sometimes clients and their representatives. Even in rare cases where software errors may be caused by the development environment (interpreters, wizards, automatic software generators, etc.), it is reasonable to claim that it is human error that caused the failure of the development environment tool. The causes of software errors can be further classified as follows according to the stages of the software development process in which they occur.

1) Faulty Definition of Requirements:

The faulty definition of requirements, usually prepared by the client, is one of the main causes of software errors. The most common errors of this type are:

- Erroneous definition of requirements.
- Absence of vital requirements.
- Incomplete definition of requirements. For instance, one of the requirements of a municipality's local tax software system refers to discounts granted to various segments of the population: senior citizens, parents of large families, and so forth. Unfortunately, a discount granted to students was not included in the requirements document.
- Inclusion of unnecessary requirements, functions that are not expected to be needed in the near future.

2) Client–Developer Communication Failures:

Misunderstandings resulting from defective client–developer communication are additional causes for the errors that prevail in the early stages of the development process:

- Misunderstanding of the client's instructions as stated in the requirement document.
- Misunderstanding of the client's requirements changes presented to the developer in written form during the development period.
- Misunderstanding of the client's requirements changes presented orally to the developer during the development period.
- Misunderstanding of the client's responses to the design problems presented by the developer.
- Lack of attention to client messages referring to requirements changes and to client responses to questions raised by the developer on the part of the developer.

3) Deliberate Deviations From Software Requirements:

In several circumstances, developers may deliberately deviate from the documented requirements, actions that often cause software errors. The errors in these cases are byproducts of the changes. The most common situations of deliberate deviation are:

- The developer reuses software modules taken from an earlier project without sufficient analysis of the changes and adaptations needed to correctly fulfill all the new requirements.
- Due to time or budget pressures, the developer decides to omit part of the required functions in an attempt to cope with these pressures.
- Developer-initiated, unapproved improvements to the software, introduced without the client's approval, frequently disregard requirements that seem minor to the developer. Such "minor" changes may, eventually, cause software errors.

4) Logical Design Errors:

Software errors can enter the system when the professionals who design the system – systems architects, software engineers, analysts, etc. – formulate the software requirements. Typical errors include:

- Definitions that represent software requirements by means of erroneous algorithms.
- Process definitions that contain sequencing errors. For example, the software requirements for a firm's debt-collection system define the debt-collection process as follows. Once a client does not pay his debts, even after receiving three successive notification letters, the details are to be reported to the sales department manager who will decide whether to proceed to the next stage, referral of the client to the legal department. The systems analyst defined the process incorrectly by stating that after sending three successive letters followed by no receipt of payment, the firm would include the name of the client on a list of clients to be handled by the legal department. The logical error was caused by the analyst's erroneous omission of the sales department phase within the debt-collection process.
- Erroneous definition of boundary conditions. For example, the client's requirements stated that a special discount will be granted to customers who make purchases more than three times in the same month. The analyst erroneously defined the software process to state that the discount would be granted to those who make purchases three times or more in the same month.
- Omission of definitions concerning reactions to illegal operation of the software system. For example, in a computerized theater ticketing system operated by the customer with no human operator interface, the software system is required to limit the sales to 10 tickets per customer. Accordingly, any request for the purchase of more than 10 tickets is "illegal". In his design, the analyst included a message stating that sales are limited to 10 tickets per customer, but did not define the system's reaction to the case where a customer (who might not have listened carefully to the message) keys in a number higher than 10. When performing the illegal request, a system "crash" is to be expected as no computerized reaction was defined for this illegal operation.

5) Coding Errors:

A broad range of reasons cause programmers to make coding errors. These include misunderstanding the design documentation, linguistic errors in the programming languages, errors in the application of CASE and other development tools, errors in data selection, and so forth.

6) Non-Compliance with Documentation and Coding Instructions:

Almost every development unit has its own documentation and coding standards that define the content, order and format of the documents, and the code created by team members. To support this requirement, the unit develops and publicizes its templates and coding instructions. Members of the development team or unit are required to comply with these requirements.

One may ask why non-compliance with these instructions should cause software errors. The quality risks of non-compliance result from the special characteristics of the software development environment. Even if the quality of the "non-complying" software is acceptable, future handling of this software (by the development and/or maintenance teams) is expected to increase the rate of errors:

- Team members who need to coordinate their own codes with code modules developed by "non-complying" team members can be expected to encounter more than the usual number of difficulties when trying to understand the software developed by the other team members.

- Individuals replacing the "non-complying" team member (who has retired or been promoted) will find it difficult to fully understand his or her work.
- The design review team will find it more difficult to review a design prepared by a non-complying team.
- The test team will find it more difficult to test the module; consequently, their efficiency is expected to be lower, leaving more errors undetected. Moreover, team members required to correct the detected errors can be expected to encounter greater difficulties when doing so. They may leave some errors only partially corrected, and even introduce new errors as a result of their incomplete grasp of the other team members' work.
- Maintenance teams required to contend with the "bugs" detected by users and to change or add to the existing software will face difficulties when trying to understand the software and its documentation. This is expected to result in an excessive number of errors and the expenditure of an excessive amount of maintenance effort.

7) Shortcomings of the Testing Process:

Shortcomings of the testing process affect the error rate by leaving a greater number of errors undetected or uncorrected. These shortcomings result from the following causes:

- Incomplete test plans leave untreated portions of the software or the application functions and states of the system.
- Failures to document and report detected errors and faults.
- Failure to promptly correct detected software faults as a result of inappropriate indications of the reasons for the fault.
- Incomplete correction of detected errors due to negligence or time pressures.

8) Procedure Errors:

Procedures direct the user with respect to the activities required at each step of the process. They are of special importance in complex software systems where the processing is conducted in several steps, each of which may feed a variety of types of data and allow for examination of the intermediate results. For example, "Eiffel", a chain of construction materials stores, has decided to grant a 5% discount to customers, who are billed once a month. The discount is offered to customers whose total purchases in the last 12 months exceed $1 million. Nevertheless, Eiffel's management has decided to withdraw this discount from customers who returned goods valued in excess of 10% of their purchases during the last three months. The chain's billing system is decentralized, so that every store processes the monthly invoices independently.

9) Documentation Errors:

The documentation errors that trouble the development and maintenance teams are errors in the design documents and in the documentation integrated into the body of the software. These errors can cause additional errors in further stages of development and during maintenance. Another type of documentation error, one that affects mainly the users, is an error in the user manuals and in the "help" displays incorporated in the software. Typical errors of this type are:

- Omission of software functions.
- Errors in the explanations and instructions given to users, resulting in "dead ends" or incorrect applications.
- Listing of non-existing software functions, that is, functions planned in the early stages of development but later dropped, and functions that were active in previous versions of the software but cancelled in the current version.

D. What is Clustering and Its Various Issues

Clustering is a division of data into groups of similar objects. Representing the data by fewer clusters necessarily loses certain fine details, but achieves simplification. It models data by its clusters. Data modeling puts clustering in a historical perspective rooted in mathematics, statistics, and numerical analysis. From a machine learning perspective clusters correspond to hidden patterns, the search for clusters is unsupervised learning, and the resulting system represents a data concept. From a practical perspective clustering plays an outstanding role in data mining applications such as scientific data exploration, information retrieval and text mining, spatial database applications, Web analysis, CRM, marketing, medical diagnostics, computational biology, control and many other applications. It is important to understand the difference between clustering (unsupervised classification) and discriminant analysis (supervised classification). In supervised classification, we are provided with a collection of labeled (preclassified) patterns; the problem is to label a newly encountered, yet unlabeled, pattern. Typically, the given labeled (training) patterns are used to learn the descriptions of classes which in turn are used to label a new pattern. In the case of clustering, the problem is to group a given collection of unlabeled patterns into meaningful clusters. In a sense, labels are associated with clusters also, but these category labels are data driven; that is, they are obtained solely from the data. Data mining adds to clustering the complications of very large datasets with very any attributes of different types. This imposes unique computational requirements on relevant clustering algorithms. A variety of algorithms have recently emerged that meet these requirements and were successfully applied to real-life data mining problems. In recent years, the dramatic rise in the use of the web and the improvement in process industries in general have transformed our society into one that strongly depends on information. The huge amount of data that is generated by this process contains important information that accumulates daily in databases and is not easy to extract. This data is scattered in between the upper and

the lower limits, applying required control or strategy to this data requires lots of computational work, for having a better control strategy. The clustered data gives us a better control efficiency and performance of our system rather than working with an unorganized scattered dataset. The field of data mining developed as a means of extracting information and knowledge from databases to discover patterns or concepts that are not evident The process usually consists of the following: transforming the data to a suitable format, cleaning it, and inferring or making conclusions regarding the data. Data analysis underlies many computing applications, either in a design phase or as part of their on-line operations.

Data analysis procedures can be dichotomized as either exploratory or confirmatory, based on the availability of appropriate models for the data source, but a key element in both types of procedures (whether for hypothesis formation or decision-making) is the grouping, or classification of measurements based on either of the following:

(i) Goodness-of-fit to a postulated model,
(ii) Natural groupings (clustering) revealed through analysis.

Cluster analysis is the organization of a collection of patterns (usually represented as a vector of measurements, or a point in a multidimensional space) into clusters based on similarity. Cluster analysis is thus a tool of discovery. It may reveal associations and structure in data which, though not previously evident, nevertheless are sensible and useful once found. The results of cluster analysis may contribute to the definition of a formal classification scheme, such as a taxonomy for related process variables, parameters or objects; or suggest statistical models with which to describe control; or indicate rules for assigning new cases to classes for control and analysis purposes; or provide measures of definition, variations and change in what previously were only broad concepts. Clustering is the classification of similar objects into different groups, or more precisely, the partitioning of a data set into subsets (clusters), so that the data in each subset (ideally) share some common trait - often proximity according to some defined distance measure. Clustering is useful in several exploratory pattern-analysis, grouping, decision making, and machine-learning situations, including data mining, document retrieval, image segmentation, and pattern classification. However, in many such problems, there is little prior information (e.g., statistical models) available about the data, and the decision maker must make as few assumptions about the data as possible. It is under these restrictions that clustering methodology is particularly appropriate for the exploration of interrelationships among the data points to make an assessment (perhaps preliminary) of their structure.

1) ***Issues of Clustering:***

- The main requirements that a clustering algorithm should satisfy are the following Dealing with different types of attributes:
- There are various attributes of data that any clustering algorithm need to satisfy, the most general taxonomy being in common use distinguishes among numeric (continuous), ordinal, and nominal variables. A numeric variable can assume any value in R. An ordinal variable assumes a small number of discrete states, and these states can be compared.
- Scalability to large datasets: The data sets could be in any possible range, varying between large extremes and they need to be normalized by the clustering algorithm.
- Ability to work with high dimensional data: The data could be multidimensional varying from 1, 2……n.; depending on the application data on which clustering is being applied
- Development of a medical inferencing system using data clustering. The data could be multidimensional varying from 1, 2……n.; depending on the application data on which clustering is being applied.
- Ability to find clusters of irregular or arbitrary shape: The shape of clusters could be any arbitrary shapes. We prefer using Euclidean distance to get a circular shape of the clusters, but still shape of clusters cannot be accurately defined
- Handling outliers: The data points on the boundary of clusters need to be handled; this is done in hierarchical methods by associating the boundary points to one of the clusters. While in fuzzy clustering, we associate membership functions to the points lying on the boundary of clusters.
- Time complexity: Complexity of the data points in terms of time has to be taken care of while clustering.
- Data order dependency: Dependency of data points on other variable can affect the clustering of data and there by the cluster centers too, so it has to be taken care beforehand.

II. PROPOSED WORK

The proposed work is about to study and analysis of software quality in a software system under the consideration of software and the hardware based analysis. The Software Reliability is one of the common measure of software quality. In our proposed work we are presenting to perform the hardware based reliability analysis at the first level. If the hardware oriented criticality is feasible then the software based analysis will be performed. For this analysis the fault analysis will be performed in each module. The analysis will be done using clustering and the rejection method. As the software based analysis is performed, a combined decision will be taken about the reliability of the whole system. The complete work is divided in 3 main steps.

1) Analysis of Software Defects and Data Collection related to software as well as hardware based defects.
2) Performing the Clustering Algorithm on Collected Data and take the decision based on rejection method
3) Conclude the overall results from the system

A) Analysis and Data Collection

There are number of different approaches to collect and analyze the data. Some of such approaches include.

- According to first approach we can use the secondary data from the company directly.
- The second approach is the online form filling. We can develop a website or design a blog to get the user response. For such kind of feedback we have to prepare a general questionnaire respective to the defect effect, cause and criticality. Once the online form is prepared we need to invite the user to give their feedback.
- The third approach is about the interviewing or the questionnaire. According to this approach we will conduct a personal interview system or the questionnaire to different users. We will record the feedback of all users. There are some chances of any kind of influence of a person while questioning or answering the system.
- Automated reliability monitoring is a continuous reporting process which periodically communicates all the failures or other critical data which have occurred since the last communication with the software organization.

B) Fuzzy based Rejection Method

Rejection sampling is a basic pseudo-random number sampling technique used to generate observations from a distribution. It is also commonly called the acceptance-rejection method or "accept-reject algorithm". It generates sampling values from an arbitrary probability distribution function $f(x)$ by using an instrumental distribution $g(x)$, under the only restriction that $f(x) < Mg(x)$ where $M > 1$ is an appropriate bound on $f(x) / g(x)$. Rejection sampling is usually used in cases where the form of $f(x)$ makes sampling difficult. Instead of sampling directly from the distribution $f(x)$, we use an envelope distribution $Mg(x)$ where sampling is easier. These samples from $Mg(x)$ are probabilistically accepted or rejected.

III. RESULTS

The presented work is implemented in matlab 7.8 environment. The results obtained from the system are given as under:

A. C-Means Clustering

We have implied the C-Means clustering on the available dataset that consist on error based data respective to different modules. It includes the modules, module size, number of errors and the error density. We have applied the C Means clustering to categorized these errors to get the better results out of it.

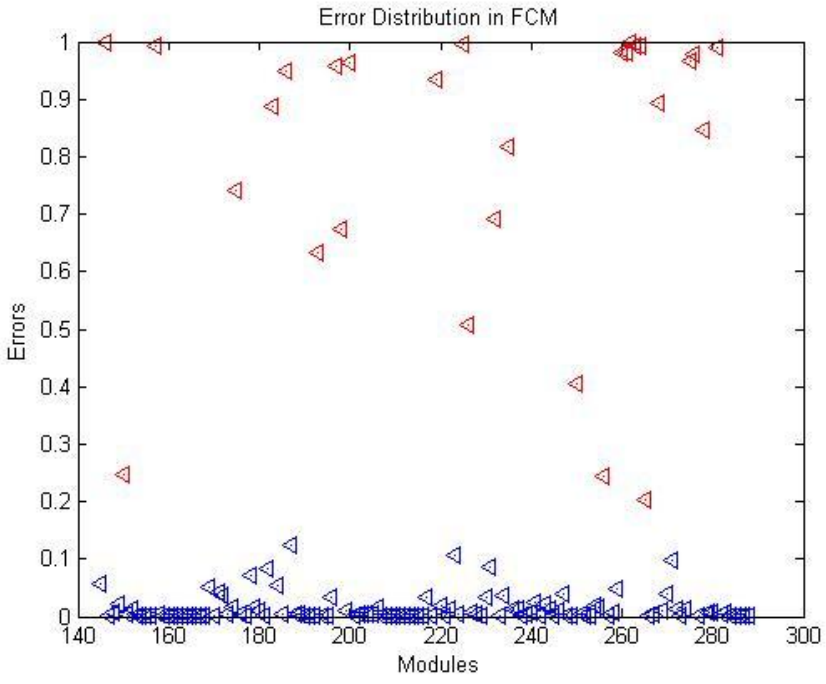

Fig1 : Fault Distribution In FCM

Here figure 1 is defines the results of error distribution as the rejection method is applied on it. The rejection method is based on Mean Square Error. When the Fault ratio higher then MSE are marked in red color, that show the rejected modules because of high fault ratio and blue marked symbols defines the low error ratio.

B. K- Means Clustering

We apply the k-means clustering for analyzing the faults on software projets. Here figure 2 is showing the results of low fault criticality in a software project. It includes the modules, module size, number of errors and the error density. Here figure 2 defines the results of error distribution as the rejection method is applied on it. The rejection method is based on Mean Square Error. When the Fault ratio higher then MSE are marked in red color, that show the rejected modules because of high fault ratio and blue marked symbols defines the low error ratio.The K-means technique detect less defects in comparison of C-Means technique.

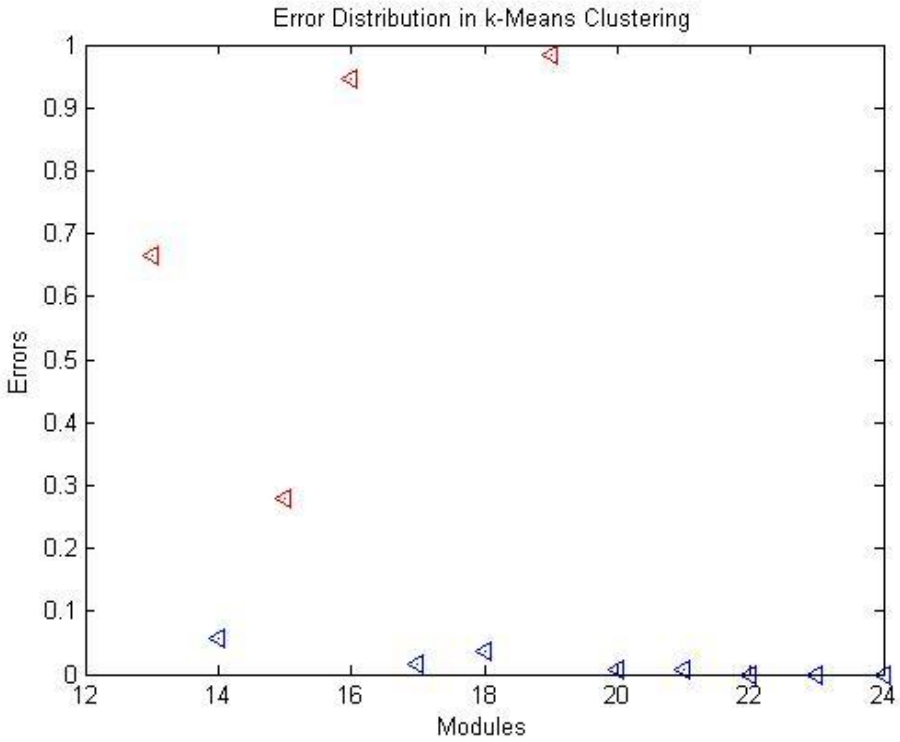

Fig 2 : Fault Analysis In K-Means

So, the figure shows that k-means technique is not the better than c-means technique.

C. Hierarchical Clustering

We apply hierarchical clustering for analyzing the faults into the software projects by taking error density,error count,lines of code as input and then the results is showing in figure.

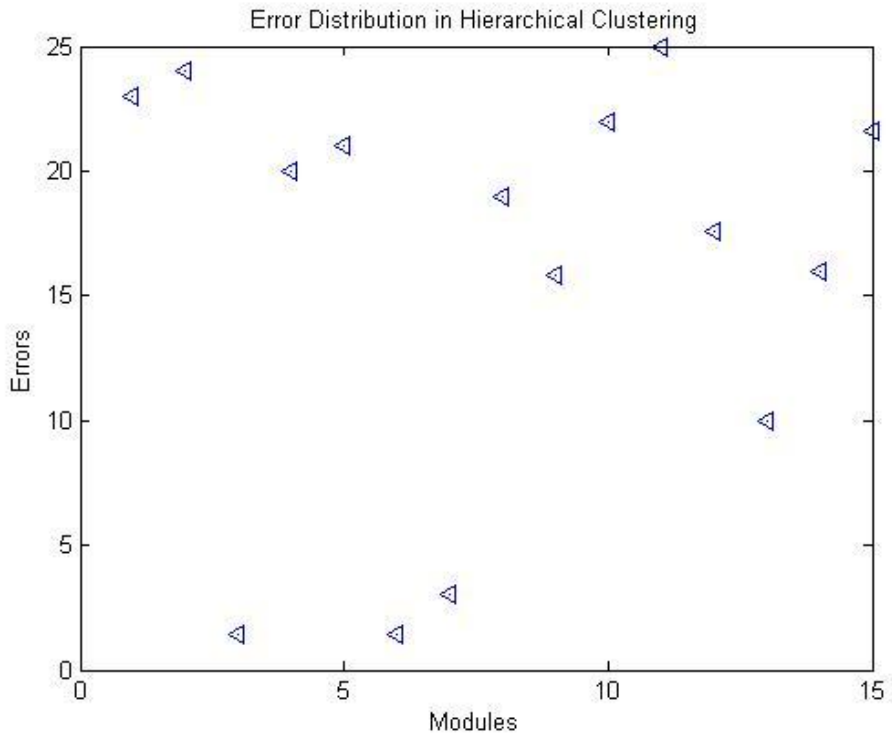

Fig 3 : Fault Criticality in Hierarchical Clustering

Here figure 3 is showing the results of low fault criticality in a software project. As we can see here the blue symbols shows the modules having no error or the less error. The decision is based on the criticality of the software fault and its relation with the software module.

D. Comparative Analysis of Different Clustering Approaches

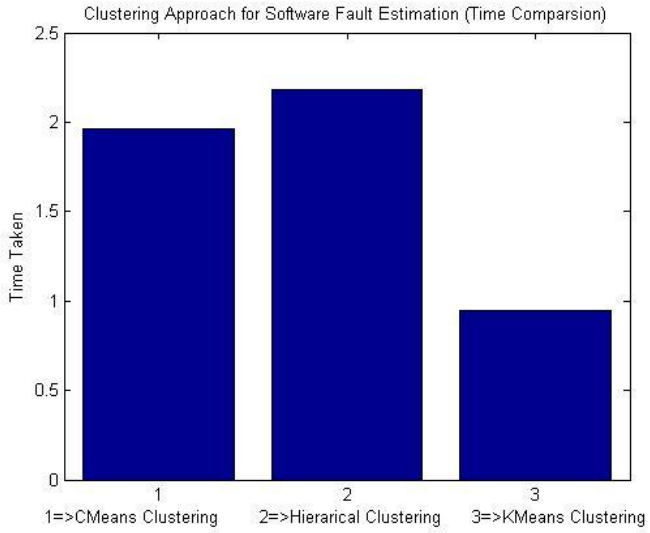

Fig 4: Time Comparison Of Different Clustering Approaches

Here figure 4 is showing the final results in terms of comparison between used three clustering approaches. As we can see the maximum time is taken by hierarchical clustering and least time is taken by K Means Clustering.

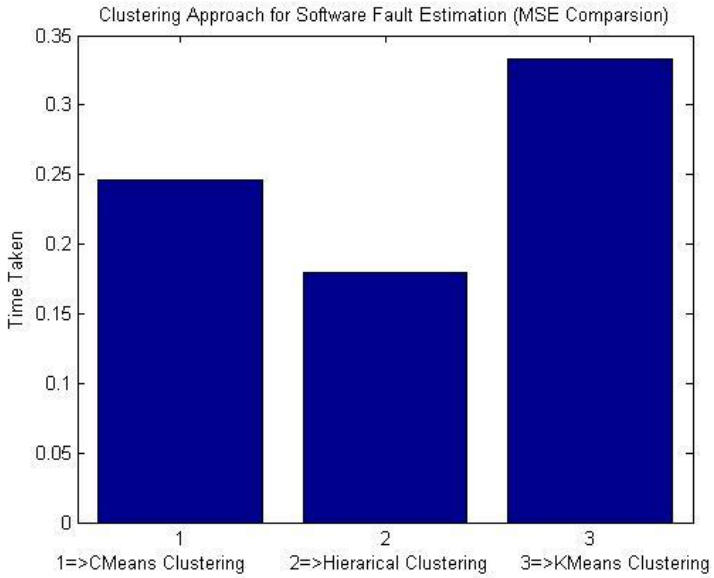

Fig 5: MSE Comparison of Different Clustering Approaches

Here figure 5 is showing the final results in terms of comparison between used three clustering approaches. As we can see the maximum MSE is given by K Means clustering and least time is taken by Hierarchical Means Clustering.

IV. CONCLUSION

In this present work we have defined a fuzzy based software quality estimation approach to analyze the software criticality. In this work a layered approach is defined to identify the software criticality on individual factor and later on association mining is implemented to perform the combined analysis. The obtained results classify the available software modules under different criticality levels.

ACKNOWLEDGEMENT

Sincere Thanks to HCTM Technical Campus Management Kaithal-136027, Haryana, India, for their constant encouragement.

REFERENCES

[1] PuneetDhiman, Manish, RakeshChawla" A Clustered Approach to Analyze the Software Quality using Software Defects"2012

[2] R.Karthik, N.Manikandan,"Defect Association and complexity prediction by mining Association and Clustering Rules" Volume 7, 2010.

[3] Deepak Gupta, VinayKr.Goyal, Harish Mittal"Analysis of clustering Techniques for software quality prediction".
[4] Florac, W., (1992). Software Quality Measurement : A Framework for Counting Problems and Defects, Technical Report CMU/SEI-92-TR-22.
[5] Adeel, K., (Aug. 2005). Defect prevention techniques and its usage in requirement gathering – industry practices , Proceedings of Engineering Sciences and Technology, ISBN 978-0-7803-9442-1, SCONEST, IEEE Computer Society Publisher, pp 1-5.
[6] Ayman E.Khedr, Mostafa Sayed(2013),"Development And Evaluation Of A Defect Tracking Model For Classifying The Inserted Defect Data", European Scientific Journal April 2013 Edition Vol.9, No.12 Issn: 1857 – 7881 (Print) E - Issn 1857- 7431.
[7] Mrs. Bharati,(2009) "A Comparative Analysis of Fuzzy C-Means Clustering and K Means Clustering Algorithms", *International Journal Of Computational Engineering Research / ISSN: 2250–3005*

