International Journal of Technical Innovation in Modern Engineering & Science (IJTIMES)
Impact Factor: 5.22 (SJIF-2017), e-ISSN: 2455-2585
Volume 5, Issue 04, April-2019

Assessing the Complexity of Software Bug Severity using Evolutionary SOM Clustering Approach

M.Chalapathi Rao[1] ,Dr.P.Suryanarayana Babu[2]

[1]*Research Scholar, Rayalaseema University, Kurnool, Andhra Pradesh, India*
[2]*Research Supervisor, Department of Computer Science, Rayalaseema University Kurnool, Andhra Pradesh, India.*

Abstract— As software demands increase and software delivery span decreased, ensuring software quality becomes a challenge. However, due to software complexity and inadequate testing, no software can claim to be error-free. Maintenance activities are therefore required to ensure that the software works smoothly. Software bug repositories are of great source of knowledge. When a bug is identified by a tester or software engineer, its related information is entered in Bug Tracking System. During its life time, a bug enters into various bug states for resolution. These bug tracking systems enable users to report the bugs found while the software is running. However, the severity prediction has recently gained a lot of attention in software maintenance. Bugs with higher severity should be corrected before bugs with lower severity. In this paper, an evolutionary interactive approach is proposed to analyze the bug reports and assesses the severity. This paper presents a SBCC – Software Bug Complexity Cluster using Self Organizing Maps (SOM) approach. In this SBCC a feature matrix is constructed using bug durations and software bug complexities are grouped into different clusters including Blocker, Critical, major, trivial and minor by specifying severity using two different methods, namely- k-means and SOM. The accuracy of different bug complexities are estimated using bug duration, proximity error and different distance functions. Our systematic study has ascertained that the proximity error and fitness on SOM has better accuracy and performance compared to K-Means. The collected results showed that the SBCC attained better performance with respect to fitness and cluster proximity error.

Keywords— SBCC, SOM, Severity, K-Means, Complexit, Prediction.

I. INTRODUCTION

Software influences a wide variety of human operations and their use is growing enormously. Due to increased demand for reduced delivery time, it is becoming critical to maintain quality though reducing delivery time. That's why; different testing techniques are used to ensure software quality [1]. Nevertheless, the possibility of latent bug presence throughout the software could not be discarded despite robust testing.

Predicting software defects is an important task in the software life cycle. During software testing, the unexpected behavior is identified and manifest as a defect it can be found during either testing or in use of the product. The exploitable bugs identified once the enlistment of the software effect the software's accuracy and quality. Bug tracking (BTS) systems enable these software bugs to be reported by both users and developers. It can enable more bugs to be identified and solved and thus improve the overall quality of the software produced [2][3].

Triager analyzes the reported bugs in the Bug Tracking System to calculate their validity, correctness, importance, severity and also to verify their duplicity, thus assigning them to the relevant developer to solve them. Triager is the person who analyzes and refines the bugs reported using his knowledge and experience. The bug training process starts with Review of Bug Report to evaluate the bug after review and the developer is assigned the bug based on its severity, such as bug severity, priority etc.

As reported in the literature [4][5][6], these bug tracking systems contain a large number of bugs. Therefore, developers must make a choice to solve among all reported bugs. The bug severity is characterized as the software functionality effect of the bug [7]. But, for a huge number of bug reports, assigning severity manually is a really complex job and time consuming. Also, the accuracy of the identification relies on the experience and knowledge of the triager to analyze the bug. As a result, the need to simplify the entire application bug incidence estimation complexity method has increased to create bug triage far more efficient and less time consuming. Therefore it is necessary to automate the process the task of grouping bugs based on severity. Many researchers used machine learning algorithms in earlier studies to optimize the bug triage process [8] and identify deception in reported bugs [9]. But so far, it is still far from reliable accuracy and room for improvement is still there. Therefore, this paper attempts to cluster software bugs based on their severity with increased accuracy.

Finally, the main contribution of the paper is to device an evolutionary interactive approach called Software Bug Complexity Cluster (SBCC) clustering algorithm to assess and predict complexity of software defects into different clusters or groups. The proposed process is constructed by adapting the Self Organization Maps (SOM) clustering techniques to retrieve useful information after the pertaining to different clusters and enable them to group from different data perspectives including Blocker, Critical, major, trivial and minor by specifying severity using two different methods, namely- k-means and SOM. The Software Bug Complexity Clustering algorithm efficiently assesses the severity of

software bugs. This algorithm also constructs a feature matrix using defect cost duration. A systematic study is conducted on SBCC using SOM to assess the software bug complexity used effectively with high accuracy rate. Furthermore, a comparison of cluster fitness and cluster proximity error measures are applied to compare the proposed prediction model with the existing standard K-means algorithm.

The rest of the paper is organized as follows: In Section 2, we present basic preliminaries and related literature work associated with software defect detection. In section 3 the step wise description of the proposed approach is elicited. Dataset description, evaluation methodology and Comprehensive analysis are presented in section 4 and conclusion and future work will be in section 5.

II. BASIC PRELIMINARIES AND ASSOCIATED WORK

Most of the existing software defect prediction studies in the literature are limited in carrying out relative empirical analysis of all learning methods. Some of them have used few methods and provide an association among them and others just discussed or proposed a method by extending them based on accessible learning techniques [10].

Software Bug severity prediction will helps to decide the next action to do with reported bugs. During severity prediction in BTS, it is a difficult and time preserving task to assess the severity of software bugs due to large number of bug reports. Since its identification, many researchers have been trying to automate the bug severity prediction process.

The bug severity prediction using BTS first was proposed using supervised machine learning techniques [11]. The system is designed to predict the developer to whom bug ought to be assigned. Support Vector Machine (SVM), C4.5, and Naïve Bayes and other machine learning algorithms extended and developed a recommender system for bug reporting [12]. A new method for predicting the estimation of software bug is presented using bug duration [13] [14]. A new bug discovering method using bug fix memories: a program-specific bug is developed by the bug fix history analysis will be described in [15].

A new automated software bug severity prediction method called SEVERIS was proposed by [16] to assign a level of severity to bug reports. Using text mining algorithms, Eclipse, Mozilla bug reports were pre-processed and naïve Bayes classifier was used. Different approaches to machine learning algorithms namely K-nearest neighbor, J48, RIPER, SVM, and Naïve Bayes, have been applied to IV & V bug reports of NASA [17]

This present research will discusses K-means and Self Organizing Maps (SOM). The k-means technique is feasible for implementation, recognized as the best used technique for partitioned clustering. The technique's execution time is highly effective. The parameter k is regarded as the input in order to maintain maximum similarity with the intra cluster, as well as minimal similarity with the inter cluster. The k clusters therefore into a separate group of n data objects. The mean particle value in a cluster is measured by cluster similarity, also known as the cluster center or gravity center.

The Self-Organizing Map (SOM) has proven to be useful in many applications as among the most prevalent neural network designs [17]. It forms part of the category of dynamic learning networks. Use the SOM to cluster data without knowing input data class memberships. Also termed the SOM was SOFM, the Self-Organizing Feature Map, and can be used to identify characteristics intrinsic in the problem. SOM provides a topology that preserves mapping from high-dimensional space to map units. Map units or neurons typically form a two-dimensional lattice and thus modeling is a navigation of high-dimensional space on a plane.

In this paper, an evolutionary interactive approach is proposed to analyze the bug reports and assesses the severity. This paper presents a SBCC – Software Bug Complexity Cluster using Self Organizing Maps (SOM) approach. In this SBCC a feature matrix is constructed using bug durations and software bug complexities are grouped into different clusters including Blocker, Critical, major, trivial and minor by specifying severity using two different methods, namely-k-means and SOM. The accuracy of different bug complexities are estimated using bug duration, proximity error and different distance functions. Our systematic study has ascertained that the proximity error and fitness on SOM has better accuracy and performance compared to K-Means.

III. SBCC: SOFTWARE BUG COMPLEXITY CLUSTERING USING SOM

The entire process of severity prediction proposed in this paper can be summarized into 3 major initiations. In the first step, a detailed data acquisition is presented. In the second step, feature matrix construction is detailed. In the third step we will propose an integrated approach for estimating software bug complexity.

3.1. Dataset Description

The goal of this research is Eclipse and Mozilla as it has a broad field of development and Eclipse is a large and mature OSS project. The experiment considers Eclipse bug reporting cases to conduct severity-based bug classification. Eclipse bug reports are expected to be of good quality [18] as Eclipse developers are themselves and use technical terms to define bug reporting. This will help create a more accurate dictionary of terms for specifying the level of bug severity including blocker, critical, enhancement, major, minor, normal, trivial in this study. Normal bug tracker involves manual examination to evaluate the severity of such bug reports and to categorize bug reports into serious or non-serious categories. [6].

The severity level of critical, blocker and major is therefore considered to be severe while the severity level of minor and trivial is not considered to be severe. The Eclipse data set contains four components UI, SWT, Debug, and Core. This study perceives core instances of bug reporting. Table 1 shows the data sets deemed in this study.

TABLE 1: Bug report instances of components of Eclipse\

Component	Severe Bug Report	Non Severe Bug Report
Core	1514	1487
Debug	1514	1400
SWT	1618	1393
UI	1764	1432

Eclipse's UI component represents the IDE interface and the debug component is the component relating to all the program's debugging activities. Eclipse's SWT component is abbreviated as a standard widget tool for all widgets used in Eclipse software development, and Eclipse's core component is the main IDE infrastructure that includes compiler, API model, code selection and evaluation support, etc. The generalized model for classifying reports of these components can therefore be used to classify reports of any other Eclipse component.

3.2. Feature Matrix Construction by Estimating Bug Duration

In a Bug tracking system [19], the life cycle of a bug follows a set of defined states and transformations. A bug enters the system either in the Unconfirmed or in the "New" state. The bug will initially be classified as New if the submitter is trusted. The Triagers confirm bugs ' existence, validity and non-duplicity and move them from Unconfirmed to New. Developers are assigned bugs and moved to Assigned once a bug has been tried and assigned to the right product, severity and priority.

A bug's total fix duration can be calculated by reviewing the bug-status, bug-creation-date, and bug-modified date attributes. If the status of the bug is "fixed" or "resolved," then the duration of the fix is the differences between the bug-modified-date and the created-date bug. This difference can be transformed into a granular time such as hours, days, weeks, months, etc. To enable the application of data mining tools and algorithms requiring the operation of a nominal target class, we discretized the time to resolve values using an algorithm for binding the same frequency. The size of each bin generated by the discretization algorithm approximately corresponds to the segmentation that could naturally be used for scheduling purposes [20].

3.3. SBCC: Software Bug Complexity cluster

The extracted feature data set displays the current status of each bug. Because we want to predict bugs ' lifetime at confirmation time, we need to roll back to the "New" status. Roll-back is achieved incrementally by applying the changing history of the bug in reverse order until a particular state is reached.

3.3.1. Estimating software bug complexity using K-means

The k-means technique, acknowledged as the best employed technique for partitioned clustering is feasible for implementation. The execution time of the technique is very effective. The parameter k is considered as the input so as to maintain maximum similarity with intra cluster, likewise minimum similarity with inter cluster [21]. Consequently, the k clusters to a separate group of n data objects. The mean value of the particles in a cluster is measured by similarity in cluster, which is also known as the center of cluster or center of gravity. The selection of data particle in k random is made at the outset. Each of the particles represents a mean value of cluster or center. Subsequently, considering the like similarity distance between the mean of the cluster and particle, the remaining particles accordingly, is allocated to the clusters. The new value of mean is then calculated by each cluster. The procedure persists until the accomplishment of convergence criterion.

Pseudo code of K-Means algorithm

- Centroid vectors of cluster are inherited from the datasets.
- Each data particle is assigned to the nearby cluster centroids.
- Using above equation cluster centroid vector cj is recalculated.
- Until a convergence criterion is attained, second and third steps are repeated.

3.3.2. Estimating software bug complexity using Self-Organizing Map (SOM)

Professor Kohonen has developed the Self-Organizing Map. In many applications, the SOM has proven to be useful one of the most popular neural network models. It is part of the competitive learning networks category. Based on unsupervised learning, which means there is no need for human intervention during the learning process and little need to be known about the input data characteristics.

Use the SOM to cluster data without knowing the memberships of the input data class. The SOM was also called SOFM, the Self-Organizing Feature Map, which can be used to detect features inherent in the problem. Provides a topology that preserves mapping to map units from high-dimensional space. Map units or neurons generally form a two-dimensional lattice and therefore mapping is a mapping of high-dimensional space on a plane.

Preserving topology property means that the mapping preserves the relative distance between the points. Points near each other in the input space are mapped to nearby map units in the SOM. The SOM can therefore be used as a high-dimensional data analysis tool in the cluster. Furthermore, the SOM can generalize the ability. The generalization means that the network can recognize and characterize incoming inputs. A new input is assimilated to the map unit to which it is mapped.

An SOM's main goal is to transform an incoming signal pattern of arbitrary dimension into a discrete one-or two-dimensional map, and to accomplish this transformation in a topologically ordered manner. During competitive learning, the neurons will be selectively tuned to different input patterns or classes of input patterns. The locations of the so tuned neurons are ordered and on the lattice a meaningful coordinate system is created for the input features. The SOM therefore forms the required input pattern topographic map.

Pseudo code of SOM Algorithm

1. Weights of each node are initialized.
2. From the training data set, a vector is selected randomly.
3. To calculate which weights are most like the input vector, each node is examined. The winning node is generally referred to as the Best Matching Unit (BMU).
4. Then it calculates the BMU's neighborhood. Over time, the number of neighbors decreases.
5. By becoming more like the sample vector, the winning weight is rewarded. Also the neighbors become more like the vector of the sample. The closer a node is to the BMU, the less it learns, the more its weights are altered and the further the neighbor is from the BMU.
6. For N iterations, repeat step 2.

IV. EXPERIMENTAL ANALYSIS

In this section we will revise the performance of our proposed approach SBCC using SOM with K-Means algorithm. The performance of the proposed approach is evaluated based on different evaluation measure as listed below. The datasets are used for evaluation in this is study are five datasets, namely DS1, DS2, DS3, ECLIPSE and MOZILLA. The proposed RFCM clustering technique marks the data with class labels into seven different classes; Blocker, Critical, Enhancement, Major, Minor, Normal, and Trivial. We chose Eclipse JDT Core18 for the testing of our model and this dataset. This data set is designed to perform bug prediction at the class level. This approach can be extended to file, package, project and other levels in a straight forward manner.

A. Evaluation Measures

The SBCC technique is selected by using the predefined distance measures, fitness and Proximity error to sense the selected metric.

1. Similarity Measures

The similarity measures which are used with relevance feedback are as follows:

- **Euclidean Distance :** The two data objects are p = (p1,p2,p3,...,pn) and q = (q1,q2,q3,...,qn) are plotted in n-dimensional Euclidean space, thus the distance between p->q or from q->p of a complete data set D can be represented as:

$$d(p,q) = d(q,p) = \sqrt{(q1-p1)2} + (q2-p2)2 + \ldots\ldots\ldots + (qn-pn)2$$
$$= \sqrt{\sum_{i=1}^{n}(qi-pi)2}$$

- **Cosine Distance:** In order to measure the similarity as an alternative to dissimilarity, this specific distance is used. In fact, this distance is enclosed by 0 and 1. The cosine distance which is often employed to measure clustered data is thus depicted by:

$$\text{Cos(xi,yi)} = \left[\frac{xi \bullet yi}{|xi||yi|}\right]$$

The two data vectors here used are xi and yi, wherein xi.yi represents the dot product and |xi| and |yi| corresponds to the data vector of length X.

- **Chi-Square Distance:** The chi-square distance *d* between objects x and y in an *m*-dimensional object space is represented as

$$\text{D (x, y)} = \sum_{i=1}^{m}\left(\frac{1}{sum_i}\right) * \left[\left(\frac{x_i}{size_x}\right) - \left(\frac{y_i}{size_y}\right)\right]^2$$

Here sum of tuple values as sum_i for attribute *i* taking place in the training dataset, and $size_x$ is the sum of all values in the object *x*.

- **Camberra Distance:** The Camberra distance *d* between objects x and y in an *m*-dimensional object space is given by:

$$D(x,y)=\sum_{i=1}^{m}\left|\frac{(x_i-y_i)}{(x_i+y_i)}\right|$$

Where y = (y1,y2,y3,.., ym) and x = (x1,x2,x3,..., xm) are two points in Camberra dimensional space of m.

- **City Block Distance:** The well-known distance city block also called a Manhattan similarity measure may be perceived as

$$D(x,y)=\sum_{i=1}^{m}\left|(xi-yi)\right|$$

1. Accuracy

Accuracy should be measured by taking into consideration the true positives and true negatives.

2. Fitness

The average distance between a cluster centroid and the objects in the dataset. The clustering technique uses this value to elucidate the data sets is represented as:

$$J(P)=\sum_{k=1}^{n}\sum_{i=1}^{c}\left[\mu_{ik}\right]^{m}\left\|x_k-v_i\right\|^2$$

3. Proximity error based on number of clusters

The probability of occurrence of error in the specified dataset or algorithm is calculated by this, which would attempt to remove that specified dataset having maximum errors as well as evade error prone algorithms. The data items were randomly opted from the dataset for each particle C and are used as prototypes to calculate a partition matrix for each particle, as per:

$$U_{ik}=1/(\Sigma_{j=1}^{C}(d_{ik}^2/d_{jk}^2))$$

Subsequently, the centroid associated with the each partition matrix is computed and in turn are utilized as the initial population of particle. This calculates the probability of occurrence of proximity error in that particular dataset or technique

$$E=\Sigma_N^{k1}\Sigma_N^{k2}|p_{k1k2}^{o}-p_{k1k2}^{r}|$$

Where $p=[p_{k1k2}]=\Sigma_{i=1}^{C}\min(u_{ik1},u_{ik2})$

B. Severity Prediction using cluster fitness on of Different Datasets Based on K-Means Technique and SOM

The table 2 demonstrates that for all datasets, SOM is offering low fitness value, as it is un-depended on outdated information. From Table 2, it is emphasized that for entire datasets, SOM divulges the minimum fitness value within a short duration. A significant performance improvement is observed by varying inertia in SOM.

TABLE 2: Severity Prediction using cluster fitness on of Different Datasets Based on K-Means Technique and SOM

Datasets	K-means	SOM
DS1	26.54	34.52
DS2	59.5	89.5
DS3	66.3	86.3
Mozilla	29.35	45.21
Eclipse	19.55	34.86
Average	43.42	54.906

C. Severity Prediction Using Cluster Proximity Error on Different Datasets Using K-Means and SOM

The Table 3 demonstrates that for Mozilla and Eclipse datasets, SOM is offering better Proximity Error compared to K-Means algorithm when k=3, k=5 and k=7. From Table 3, it is emphasized that for DS1, DS2, and DS3 datasets, SOM divulges the nearer fitness value within a short duration.

TABLE 3: Severity Prediction Using Cluster Proximity Error on Different Datasets Using K-Means and SOM

Datasets	Number of Clusters	Traditional K-Means	SOM
Mozilla	k=3	74.52	84.7
	k=5	75.52	84.7
	k=7	58.8	86.4
Eclipse	k=3	78.3	90.1
	k=5	71.27	76.2
	k=7	65.07	76.2
DS1	k=3	30.25	24.3
	k=5	23.87	23.9
	k=7	21	19
DS2	k=3	24.99	31.5
	k=5	29.03	24.3
	k=7	25.92	16.1
DS3	k=3	22.92	25.5
	k=5	16.18	56.1
	k=7	7.12	39.1

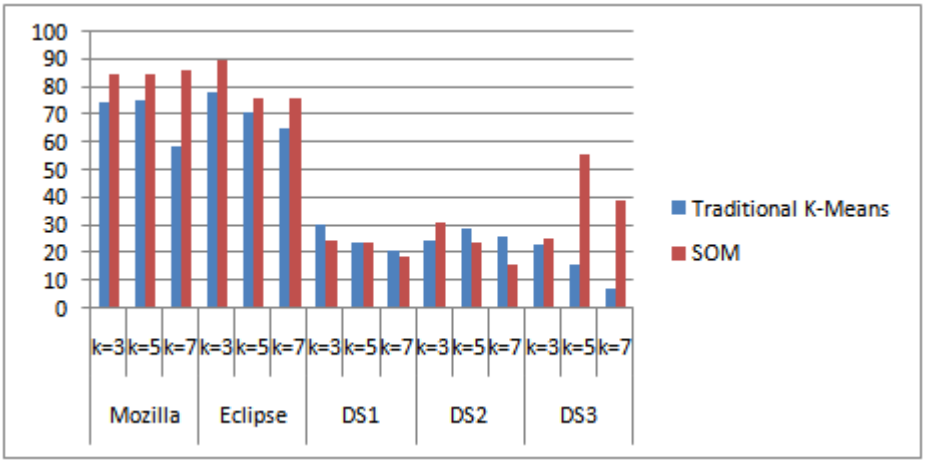

Fig 1: Severity Prediction Using Cluster Proximity Error on Different Datasets Using K-Means and SOM

From the Figure 1, it is observed that for Mozilla and Eclipse datasets, SOM is offering enhanced Proximity Error compared to K-Means algorithm with different k values. From Figure 1, it can be concluded that SOM particles are compactly grouped and it does improve clustering, in comparison to K-Means algorithm.

D. Performance Evaluation of different Datasets using K-means Technique

An assortment of similarity measures like Euclidean distance, Cosine distance, Chi-square Distance, Camberra Distance, and city block distance are analyzed for the number of clusters from different dataset, with respect to a particular seed point.

TABLE 4: Accuracy of the Eclipse's Core dataset for different distance measures with respect to K-Means technique with varying k

k	Euclidean Distance	Cosine Distance	Chi-square Distance	Camberra Distance	City Block Distance
3	33.33	33.33	66.67	33.33	66.67
5	66.67	33.33	66.67	33.33	66.67
7	66.67	66.67	66.67	33.33	66.67

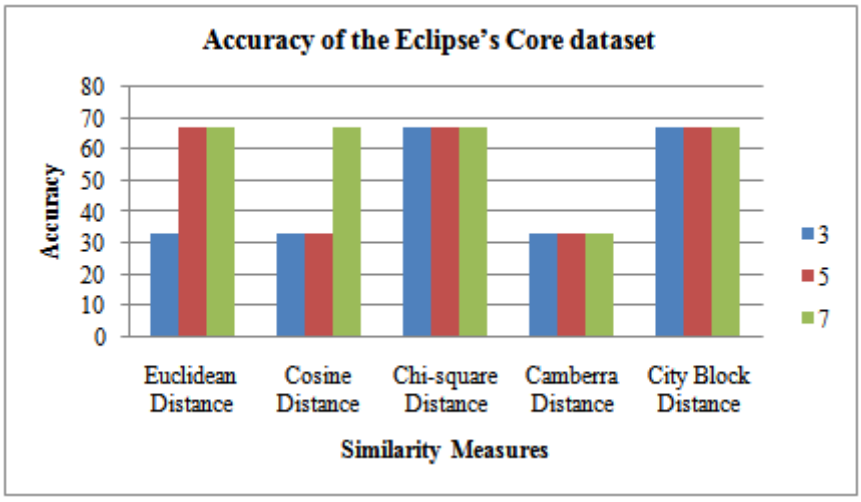

Fig 2: Accuracy of the Eclipse's Core dataset for different distance measures with respect to K-Means technique with varying k

Table 4 and Figure 2, it is observed that for Eclipse's Core dataset, the accuracy is better for Euclidean distance, Chi-square Distance and City Block Distance because of less number of classes in the dataset.

Table 5: Accuracy of the Eclipse's Core dataset on SOM Technique for different distance measures with respect to k value.

k	Euclidean Distance	Cosine Distance	Chi-square Distance	Camberra Distance	City Block Distance
3	33.33	66.67	33.33	66.67	33.33
5	33.33	66.67	33.33	66.67	33.33
7	33.33	66.67	33.33	66.67	33.33

From Table 5 and Figure 3, it is noticed that the Eclipse's Core dataset the accuracy is better for Cosine Distance and Camberra distance. With Eclipse's Core dataset k-Means is not suitable because the number of clusters increases but their no change in accuracy.

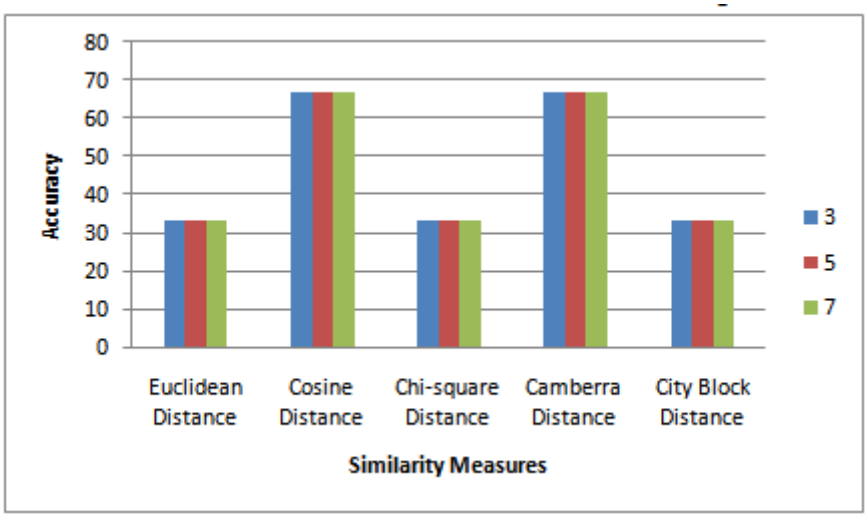

Fig 3: Comparison of accuracy of Eclipse's Core data set with different similarity measures using SOM technique

Figure 4 and Figure 5 represents SBCC Visulization on Eclipse Core Dataset using K-Means and SOM techniques.

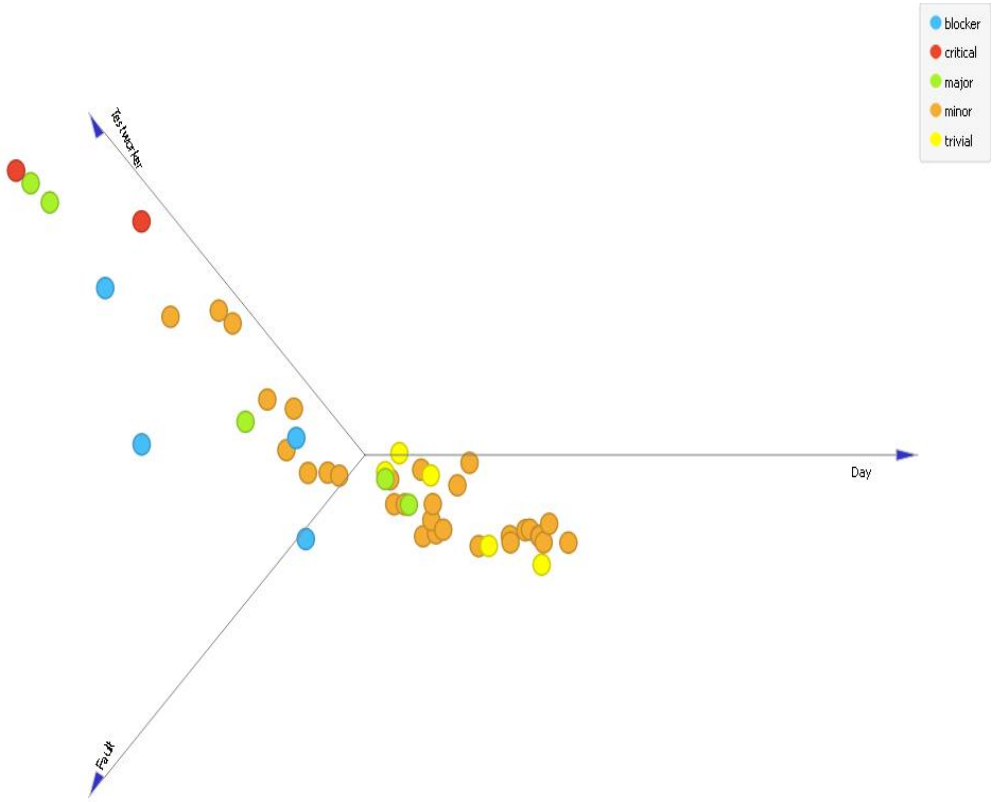

Fig 4: SBCC using K-means visualization on Eclipse Core dataset

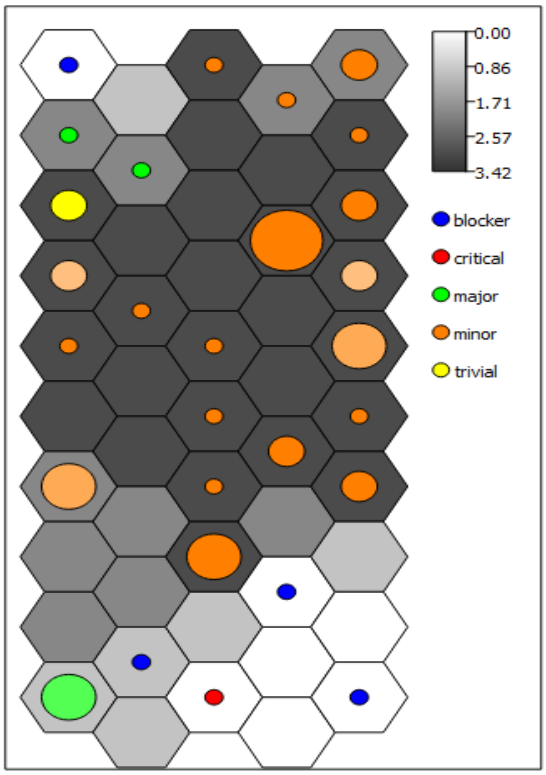

Fig 5: SBCC using SOM Visualization on Eclipse Core dataset

Figure 6 and Figure 7 represents cluster assignments of SBCC using K-means and SOM techniques on Eclipse Core Dataset.

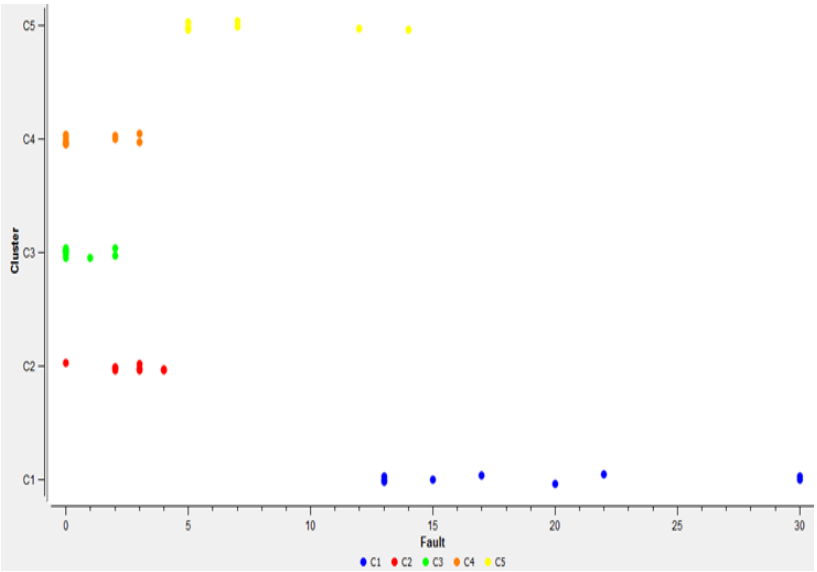

Fig 6: Cluster assignments using k-means on Eclipse core dataset

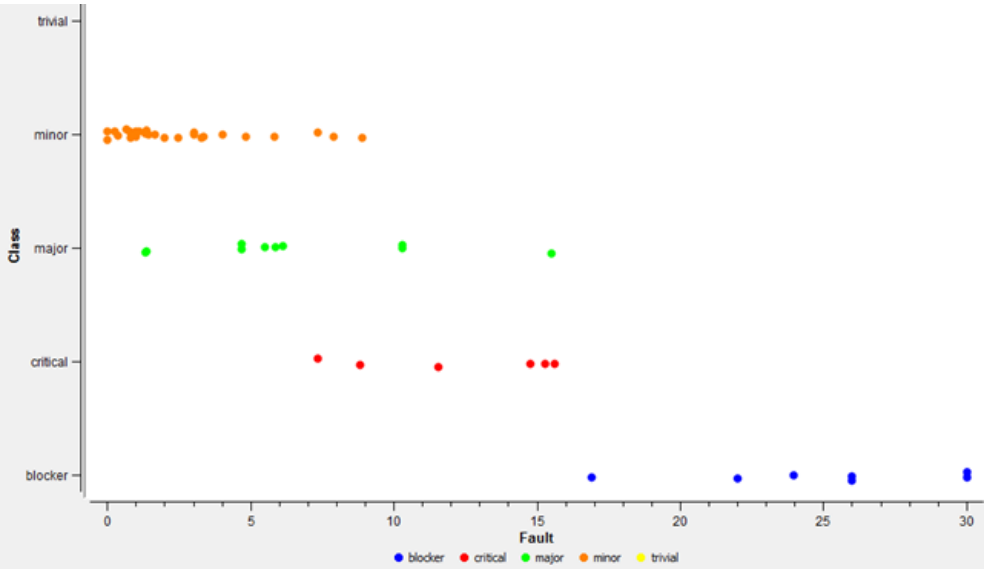

Fig 7: Cluster assignments using SOM on Eclipse Core dataset

From Figure 6 and Figure 7, it is observed that Software Bug complexity is strongly assessed using SOM compared to K-means. In Figure 7, in minor, major and critical we can notice more cluster assignments compared to K-means technique. It is due to that SOM is 85% gain in its cluster fitness, enhancement in Proximity error and different similarity measures. Hence it can be concluded from the results that SBCC using SOM performs better for estimating software bug complexity under the given experimental setup.

V. CONCLUSION

Assessing the severity of the software defects in software development and maintenance process is a critical task and which will affect the overall success of software product. Severity prediction requires historical data for finding out critical bugs. In this paper, an evolutionary interactive SBCC – Software Bug Complexity Cluster using Self Organizing Maps (SOM) approach is presented to analyze the bug reports and assesses the severity. In this SBCC, the software bug complexity or severity is predicted using bug durations by clustering them into different clusters including Blocker, Critical, major, trivial and minor. The SBCC performance is evaluated by different similarity measures namely Euclidean distance, Cosine distance, Chi-square Distance, Camberra Distance, and city block distance along with fitness and proximity error by specifying severity using two different methods, namely- k-means and SOM. Our systematic study has carried out on different datasets, the evaluation process is implemented. From the experimental study it is ascertained

that the software bug severity is assed nearly 85% using SOM compared to K-means using proximity error and fitness. We can involve other Machine Learning techniques as a future work and provide an extensive comparison among them. Furthermore this research work can be extended for online databases for real time severity prediction of software bug complexities and other software metrics are considered to assess the complexity of a future software bug.

ACKNOWLEDGMENT

The first author is thankful to Rayalaseema University, Kurnool and CMR Technical Campus, Hyderabad for providing extensive support for carrying this research work.

REFERENCES

[1] Beizer, Boris. Software testing techniques. Dreamtech Press, 2003.
[2] Raymond, Eric.The cathedral and the bazaar. Knowledge, Technology & Policy 12; 1999. no. 23-49.
[3] "15 Most Popular Bug Tracking Software to Ease Your Defect Management Process", http://www.softwaretestinghelp.com/popular-bugtracking- software/", Feb 12, 2015.
[4] Tian, Yuan, Daniel Lo, and Chengnian Sun. Drone. Predicting priority of reported bugs by multi-factor analysis. In Software Maintenance(ICSM), 2013 29th IEEE International Conference on. IEEE ;2013 , pp. 200-209.
[5] Ahsan, Syed Nadeem, Javed Ferzund, and Franz Wotawa. Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine..In Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on, . IEEE; 2009.pp. 216-221.
[6] Lamkanfi, Ahmed, Serge Demeyer, Emanuel Giger, and Bart Goethals. Predicting the severity of a reported bug. In Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on, pp. IEEE; 2010. pp.1-10.
[7] "Defect severity classification in software testing (with an example)", http://www.zyxware.com/articles/3559/defect-severity-classification-insoftware-testing-with-an-example, May 24, 2013.
[8] Alenezi, Mamdouh, Kenneth Magel, and Shadi Banitaan. Efficient bug triaging using text mining. Journal of Software 8;2003. no. 9.
[9] Runeson, Per, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. In Software Engineering, 2007. ICSE 2007. 29th International Conference on., IEEE ; 2007.pp. 499-510.
[10] S. Adiu & N. Geethanjali (2013) "Classification of defects in software using decision tree algorithm", International Journal of Engineering Science and Technology (IJEST), Vol. 5, Issue 6, pp. 1332-1340. 12.
[11] Murphy, Gail C., and D. Cubranic. Automatic bug triage using text categorization. In Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering; 2004.
[12] Anvik, John, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug?. InProceedings of the 28th international conference on Software engineering,. ACM ;2006. pp. 361-370 .
[13] Cathrin Weiß, Rahul Premraj, Thomas Zimmermann, Andreas Zeller, 2007, How Long will it Take to Fix This Bug? Intl. conf. on software engineering, IEEE Computer Society Wanshington, DC, USA, pp. 1-8.
[14] Cathrin Weiß, Rahul Premraj, Thomas Zimmermann, Andreas Zeller, 2007, Predicting Effort to Fix Software Bugs, Proceedings of the 9th Workshop Software Reengineering.
[15] Sunghun Kim, Kai Pan, E. James Whitehead, Jr., 2006, Memories of Bug Fixes, SIGSOFT'06/FSE-14, November 5–11, Portland, Oregon, USA.
[16] Menzies, Tim, and Andrian Marcus. Automated severity assessment of software defect reports.In Software Maintenance, 2008. ICSM 2008. IEEE International Conference on, IEEE;2008.pp. 346-355.
[17] Chaturvedi, K. K., and V. B. Singh. Determining bug severity using machine learning techniques. In Software Engineering (CONSEG), 2012 CSI Sixth International Conference on, IEEE; 2012, pp. 1-6.
[18] Bettenburg, Nicolas, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann.What makes a good bug report?. InProceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, ACM ; 2008 pp. 308-318.
[19] Bugzilla Documentation – Life Cycle of a Bug. Available online at http://www.bugzilla.org/docs/tip/html/lifecycle.html.
[20] K.Prasanna, M.Seetha, A. P. Siva Kumar "CApriori: Conviction based Apriori Algorithm for Discovering Frequent Determinant Patterns from High Dimensional Datasets", published in the proceedings of IEEE International Conference on Science , Engineering and Management, IEEE ICSEMR 2014, Nov 27-29, 2014 with ISBN: 978-1-4799-7613-3
[21] K.Prasanna, M.S.P.Kumar, G.Suraya Narayana., "A Novel Benchmark K-Means Clustering on Continuous data ", published in the proceeding of International Journal of Computer Science and Engineering, Vol. 3 No. 8 with ISSN: 0975-3397, 2011.