An Improved Model for Software Maintenance Cost Estimation

[1]Chamkaur Singh, [2]Dr. Neeraj Sharma, [3] Dr. Narender Kumar
[1]Research scholar, [2]Professor, [3]Assistant Professor,
[1] I.K. Gujral Punjab Technical University, Jalandhar, Punjab, India

Abstract: Software maintenance is a vast subject that includes improvements in capabilities, error correction, optimization and removal of obsolete capabilities. There is need to develop a mechanism for evaluating, controlling and making changes due to a predictable nature of change. So during usage of software if work is done to emend it then it is considered as maintenance work. Both COCOMO and PUTNAM models have been used by various researchers which have their own merits and demerits so in the third phase we have combined both to take their advantages. That's why in this paper we did hybridization of both COCOMO and PUTNAM models. The proposed model is tested for two datasets name as tomcat and Bug prediction dataset. The proposed performance in terms of the functional point which is coming high i.e. 85.65 % and must be high for high productivity of the software product which increases the demand in the market and functioning of the software developed. Also, it is shown that the efforts per person are also low, which is of 4.88 % which must be low for efficient cost estimation in terms of maintenance. The source lines of code can be high and can be low according to the development of the software requirement which needs to be developed according to the user prospective.

***Keywords*: Tomcat, Bug prediction, dataset, Software maintenance cost estimation, COCOMO, PUTNAM**

I. INTRODUCTION

Software maintenance is a vast subject that includes improvements in capabilities, error correction, optimization and removal of obsolete capabilities. There is need to develop a mechanism for evaluating, controlling and making changes due to a predictable nature of change. So during usage of software if work is done to emend it then it is considered as maintenance work. Over a period of time new technologies are introduced, expanding the software maintenance activities in order to fulfil customer requirements and making the system more efficient.

While developing software there is need to prepare an appropriate plan for maintenance activity that is considered as an important aspect of software maintenance. In which modifications are to be done those are specified by the plan. Due to change in any requirement cost should be included to develop a budget of software. This means maintenance cost is not only die to poor design but change in customer environmental and expectations or needs in which system has been developed effects the maintenance cost. Further software maintenance is a plan in which scope of maintenance, maintenance team or person and cost estimation of software maintenance is included in it.

Software maintenance cost is based on the changes made in software after its delivery to the end user. According to the up-gradation in the technology there should be up gradation in the software. Further internal issues in software may require maintenance. Maintenance cost is generally 75% of the total cost of software development. The factors given below are included in software maintenance:

- The first one is corrective maintenance in which almost 20% of cost is due to error correction or issues in the software after its delivery and usages by the customer.
- Adaptive maintenance is the second factor in which almost 25% of cost is due to changes in the software to stay effective in changing environment.
- The third factor is perfective maintenance in which 5% of the cost is due to improvement in the software to improve the performance.
- Enhancement is the last factor in which almost 50% of cost is due to the innovations to make the software up-to-date
- Numbers of models have been developed to calculate the software maintenance cost and various companies use some of them to calculate the maintenance cost.

This paper has been divided into different sections. The first section gives brief introduction about software maintenance cost, the second section talks about literature review and then different models used in this paper is discussed in third section. The fourth section includes Dataset Acquisition and Pre-processing. the fifth section covers the various approaches used. results are discussed in sixth section.

II. PREVIOUS WORK

There is increase in the service life of old systems by development and increase in interest of software reengineering. Software engineering has become an important domain of research by increase in service life of old systems. In information technology

(IT) infrastructure, reengineering major objective concentrate on bring down the investment cost. In order to achieve it capitalizing and maintenance cost on modern IT infrastructure is reduced by making it more flexible to the varying needs. In legacy system or new system there is need to take steps for reengineering system from available option of investing that makes it quite challenging. Return on investment (ROI) is considered as one of the better approach where its reengineering system is tough to compute. As in this there is need to assume the project cost that be determined by the need of reengineering. For reengineering the legacy system a general approach towards decision making is given by **Shashank Sharma, et.al, (2014).** Cost estimation requirement specific approach has been presented by them and for assessment of reengineering a ROI computation is proposed. There is rapid change in environment in this mobile communication age and challenges are faced by change in requirement of software project. The software development risk can be reduced by overcoming the impact of change in requirements. To decrease risk related to software requirements **Sen-Tarng Lai, (2015)** surveyed major software development models and development of adaptable requirements change software is recommended by them. Iterative and incremental development approach has been applied in agile development and mainly focuses on communication of client and workable software. In handling change in requirements in software development use of agile development is considered as suitable approach. At the same time many defects are present in existing agile development maintenance such as:

- Development documents control
- User story inspection
- CM system

There is need to improve the maintenance defects of agile development so a Agile Development Maintainability Measurement (ADMM) model is proposed by **Sen-Tarng Lai** that helps in collecting and analysing the critical quality factors of agile development maintainability. Then Agile Development Maintainability Enhancement (ADME) model is proposed on the basis of ADMM model that can be deploy for reducing the risk of change in requirements.

Activities include in software development methods are:

- Planning
- Testing
- Analysis
- Development
- Retirement
- Maintenance

Traditional and agile are two categories into which all activities are divided. **Dawid Zima, (2015)** have given review on some of the popular agile, traditional and open source development methods. Mainly they have focused on maintenance and testing stage of all the reviewed methods. To assess the project manpower software size approximation is considered as good reference along with the initial project planning of budget and schedule. Cost of software maintenance has been considered by various organizations due to constantly growth in cost of software maintenance. By maintenance costs cost of software development are over weighted and almost 80% budget of software industries are on maintenance. Under development estimating the size of software is widely assumed in function point techniques and on other hand calculation of function point maintenance plan is different from a development plan. Every message in a sequence chart will be collected as function points in COSMIC-FFP where developed system is analyzed from scratch. In this the redundant messages like previous version passed messages to objects are not counted as function points. To distinguish between future maintenance costs and past development costs there is need to make a distinction between enhanced or new and unchanged functions. For maintenance projects to distinguish between existing and new or changed function points a method is proposed by **Chi-Jui Lin, et.al, (2016)** that act as a supporting tool for COSMIC-FFP. From regression analysis number of lines of code per Cfsu is obtained by comparing the differences between two versions of a system. This will results in more accurately assess of messages involved in a maintenance project.

Most realistic effort required to develop or maintain software is predicted by the process of software development effort estimation. In order to avoid losses caused by poor estimation the development of estimation models and appropriate techniques is necessary. But till that time there was no appropriate method for agile development where customer causing time consuming estimation process involves frequent iterations. **Kayhan Moharreri, et.al, (2016)** have proposed a Auto Estimate automated estimation methodology to address those issues and the proposed method is complementing Agile manual planning poker. Along with the actual effort time of agile story its cards are also extracted from auto estimate leverages features. The proposed approach is evaluated it for effort prediction by alternate machine learning algorithms. When compared to planning poker estimates in the late phases of a project a better performance is achieved using selected machine learning methods. The outcomes are depicted within a real world setting and are evaluated in terms of value, applicability and accuracy. Successful systems must be established to evolve or die. The object oriented software systems are built for long time but due to legacy software system it will start degrading. Various reengineering patterns are identified that helps in capturing best practice in reengineering and reverse object OOPs legacy systems. Re-implementation of older systems is the main focus of software reengineering that makes it better maintainable. Refactoring is similar to re-engineering with-in an OOPs context. **Ankit B. Desai, et.al, (2016)** have resembled the cost of refactoring using RCE by using given object oriented refactoring opportunities. These are:

- Class misuse
- Lack of use of inheritance concept

- Violation of the principle of encapsulation
- Misplaced polymorphism
- Misuse of inheritance

Use of software has been spread to most of the areas from last few decades that results in increase of demands on quality of system and its complexity. Various standards and processes are completed and met within stipulated deadlines and product defect ratio is dictated by developing a boundary condition and software complexity. During deployment lifecycle requirements will be added as well as adjusted that result to change in the software. In order to ensure the quality of product all changes need to be tested and refined. In software engineering predictions of changes and defects of software plays a major part. The software quality is assessed by count of software defects and effort needed to fix software is also estimated but it is not very accurate. Many standard and norms are involved in industrial development and **Shahab Nadir, et.al, (2016)** research models correction efforts from numerous viewpoints. The overall product effort and effort need to fixes the software are combined to ensure quality of software and software update is considered as most common approach. Some software updates are implemented with each release and there are specific characteristics of each defect and its is project and process parameters dependent. In their work used software process model contained those factors and their influence on software defect rate is illustrated in it. During development lifecycle different phases are developed by software and effort need to fix those defects are depends on the phases where they are originated or detected. Describing the distribution of defects during these segments helps control quality assurance and limit the effort needed to increase product quality.

For developing a family of software systems of the common reuse practice that is widely used is clone and own. But if it is not supported with valuable indicator then its effectiveness loses that guide the source of new products. On the basis of clone and own an approach to support derivation of new product variants is proposed by **Eddy Ghabach, et.al, (2018)** in which possible scenarios are provided as operations to perform or achieve the derivation. Before product derivation a constraints system is generated that helps the software engineer selection of the suitable scenarios and operations on the basis of preferences. Along with it cost estimation for each operation and scenarios is also proposed by them that makes software engineer rely on it as an additional parameter that helps in achieving the derivation. A case study is used by them to validate their approach where results show that amount of time and efforts required to achieve a product derivation get reduced by using it.

Long term negative effect of choosing sub-optimal solutions is described by technical debt that helps in achieving a short term benefits like cost saving, etc. An automated production system is maintained and developed by software, electrical and mechanical like various disciplines. Due to the interlinked dependencies between the disciplines there can be negative or unexpected impact on other discipline by change in one discipline while maintenance and development period. Some necessary tasks are omitted by manually derived task list and there is unintentional introduction of TD in it. Also, the attention of TD due to lacking knowledge sometimes outdoes the short-term cost saving. For a PS domain solution selection process to assist it a workflow is proposed by **Suhyun Cha, et.al, (2018).** The workflow may include:

- Efficient change effort estimation tool allowing for multidiscipline
- TD interest estimate

Well defined job list for a modification is derived to estimate the change effort KAMP4aPS is selected as a tool. TD can be foresee and appropriate solution can be choosen on the basis of these exhaustive estimation and supplementary cost factors that fits the present situation of business.

III. MODELS

3.1 COCOMO: Constructive cost model is the abbreviation of COCOMO which is very famous model for schedule and cost estimation that is originally published in the text software engineering economics. COCOMO 81 was the original model and on the basis of 63 completed projects from various domains this model is defined during 1970s and early 1980s. Center for software engineering and systems developed and published the COCOMO II that address the emerging issues from advancements and changes. New cost drivers, new set of parameters values and scale factors use the introduction of new functional forms among main upgrades.

Three sub-models make COCOMO II:

- Applications composition
- Early design
- Post-Architecture

Effort are computed by application composition model and schedule to develop the system. Reusable and other reusable assets are integrated to develop the system or application composition model to compute effort and schedule. As compared to other models application composition model has different estimation form. In terms of object points or application points is used to measure size input and effort is calculated by productivity rate. In early stages of the project early design model is used when the project information is not detailed enough for a fine-grained estimate. Alternatively post-architecture model is used when the detailed information is available in another word development environment or high level design is determined. Source line of code are used as early post-architecture and early design models as basic size unit and same architecture form is followed. The general form of the effort formulas of the COCOMO 81, Early Design, and

Post-Architecture models can be written as

$$PM = A * Size^{B} * \prod_{i}^{p} EM_i$$

Where, effort estimate in person months is denoted by PM, multiplicative constant as A in which historical data is used to calibrate. Estimated size of the software is estimated by Size measured in KSLOC and COCOMO I exponential constant and COCOMO II scale factors is denoted by B. Effort multiplier estimated size of the software is estimated by Sizes is specified by EM that represents the multiplicative component of the equation.

3.2 PUTNAM

Putnam model is another common software maintenance cost estimation model. The equation for this model is:

$$C = size * B^{1/3} * T^{4/3}$$

Total efforts per person:

$$B = 1/T^{4} * (size/C)^{3}$$

T= Time required in years

LOC= Size of software

Where: Value of C depends on the development environment and It is calculated on the basis of historical projects data.

3.3 HYBRID MODEL

In this paper we have proposed the new approach which is the hybridization of both the COCOMO and PUTNAM models. Both COCOMO and PUTNAM used individually have some limitations but when we used it collectively then it will give us improved results. Disadvantage of using COCOMO model is that most of the times stages of software development leads to failures in estimation. On other hand in estimating cost this is simple model. The disadvantage of using PUTNAM individually is that other aspects of software development life cycle are not taken into consideration and at the same time use of it is beneficial as it is based on two variables time and size.

For hybrid model we have evaluated the source lines of code used for the execution which is the input to the KLOC evaluation which gives total source lines of code for the execution of the software system. Then we have used the product size which deals with the effort applied for the person per month for both COCOMO which is further applied as the input in the PUTNAM formula of Effort applied per month which will be our hybrid scenario for Effort per person evaluation performance. The same process is done for the functional point process. Firstly we have calculated the function point for COCOMO using its formulations and then added in the PUTNAM evaluation of the functional point which gives us the hybrid performance evaluation of the functional point.

In our proposed approach the performance evaluation is done using the formulas which are given below:

Source Lines of Code = Total Number of instruction executed / Total number of rows and columns executed

Program Size = Total number of rows * total number of columns executed

Effort Person per Month = (Program size*Total number of class labels*Time Executed by the system)/ total rows and columns executed

Functional Point = 2*(Effort Person per Month * Execution Time ^4/3) / (Scaling Factor^1/3 * S);

Predicted Object Point = (Effort Person per Month *Total length of the data to be processed)

IV. DATASET ACQUISITION AND PRE-PROCESSING

4.1 Dataset Acquisition

The experiment considers Bug Id reports instances of Matlab to perform classification of Sentimental based on severity. Matlab is an open source following development tools has been used in the development of this work. There are many other tools which can be used in this development as it be counting enters onto person and his interest. Therefore the used tools are

- 3 GB or more of RAM

- Processor-I3 or extended versions
- MATLABR2016a

Different developers used it worldwide as users of eclipse are developer and bug reports are defined using technical terms. For specifying bug severity level more accurate dictionary of terms is created to study it.

4.2 Pre-processing Process

This step performed the textual summary of bug reports and reason behind including it as prediction of severity level is that better results are given by summary report of company and trademark. It is better than the detail description of sentimental report. Stemming, stop word removal and tokenization is included in it. Its explanation is given below:

1. Tokenization: All punctuation marks are removed using it. To explore the words in document in this whole text is divided into separate tokens.
2. Stop word removal: Articles, prepositions, conjunction and other verbs, objectives and adverbs like frequent words are eliminating by this process. These words are removed because it contain very common and rarely contain any important information. This results in reduction of textual data and improves the performance of system.
3. Stemming: Words are reduced to their root words using it. For example compute is the root word of computerize, computed and computing.

V. APPROACHES USED FOR VARIOUS WORK

5.1 Feature Extraction using PCA

PCA i.e. Principal component analysis is required to calculate and analyse the Eigenvectors to find out the characteristics values and then to process data with its principal components. PCA is a standard technique used for compressing higher dimensional data sets into lower dimensional data for analysis of data, feature extraction, or compression of data. Principal component analysis involves calculating the Eigen value of a data covariance medium or singular value decay of a data matrix, generally after mean centring the data for every attribute.

Step 1: Get normalizes data from the dataset. 2-D data is represent as 1-D Vector by concatenating each row (or Column) into a long vector

Step 2: Pick the mean data from each data vector. It should be row wise.

Step 3: Calculate the covariance matrix to get the Eigen vectors and values.

Step 4: Analysis of covariance matrix for the eigenvectors and Eigen values .

Step 5: Sort the eigenvectors on basis of Eigen values from high to low. Select the components and generate a feature vector.

Step 6: Extract the new data set from chosen the components, we simply take the transpose of the vector and increase it on the left of the original data set, transposed.

5.2 Instance selection using Particle Swarm Optimization

For matrix size reduction PSO deals with instances selection and in order to deal with the further proceedings input it deal ease the processing. PSO is a generalized algorithm that use iterative method to find our global best solution out of given possible solutions. Particle Swarm Optimization use free space search over the position and velocity of the particle and c search for infinite spaces to acquire best optimize solution. So, Particle Swarm Optimization is usually used for optimization which is commonly known as routing optimization.

For each particle n = 1, ..., swarm do

Location of the paricle is set with a random vector Zi

Set the best location of the particle with its initial location Qi

If f (Qi) <f (gb) then

1. *Do updation in known postion of swarm: gb*
2. *Speed of particle is s as: Si*
3. *While a end is not reached do*

For every particle mn= 1, ..., Swarm do

For every measurement t = 1, ..., n do

4. Calculate fitness function

5. *Update speed of the particle: Si, Update the location of the particle: Zi, Update the best known positon: gb*

So , We get gbest by using iterative method, which is optimize solution.

5.3 Classification Using Linear Discriminant Analysis

Linear Discriminant Analysis is an effective statistical technique used for the classification perspective. It is used to calculate the linear combination of the extracted features, which are further, used for characterizing the distinct classes and give effective results in the data analysis and it is very much similar to the PCA.

Linear Discriminant Analysis is very much related to ANOVA and regression analysis, which are also used to determined one dependent variable from linear combination of other features. However, LDA use continuous independent variables and a categorical dependent variable where analysis of variance has categorical independent variables and a continuous dependent variable. Logistic regression is like to LDA than ANOVA , as it also explain a categorical variable by using continuous independent variables. These approaches are desirable in applications where it is not equitable to assume that the independent variables are generally distributed, that is a major assumption of the Linear Discriminant Analysis method.

The approach deals with the uploading procedure and then the preprocessing process is implemented on the uploaded data like Tokenization, Stop words removal and stemming and then we have divided into two parts

1. Feature extraction is done by the Principle Component Analysis
2. Instance selection is done by using the Particle Swarm Optimization

Then the reverse process is achieved and then we have used classifiers which is Linear Discriminant analysis which is used for the classification purpose and the system performance is evaluated by considering Accuracy, Precision, Recall and Processing time.

VI. RESULTS AND DISCUSSION

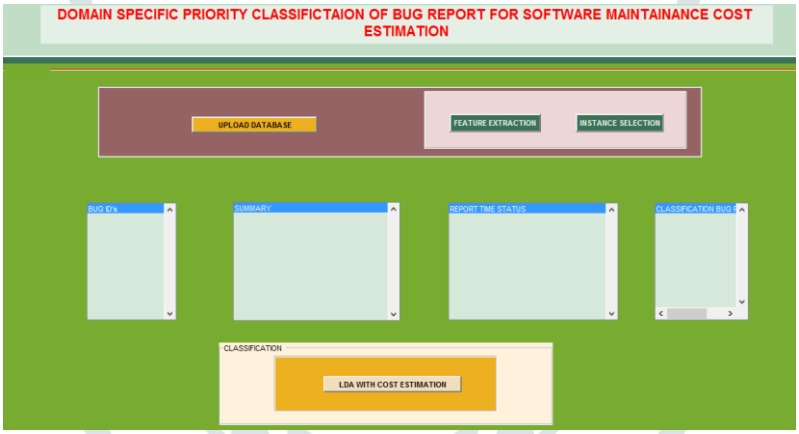


Fig 1: Graphical User Interface

The above figure shows the graphical user interface which deals with the user interface controls such as pushbuttons, panels, list boxes. In this first we upload the dataset and the data will be shown in the list boxes.

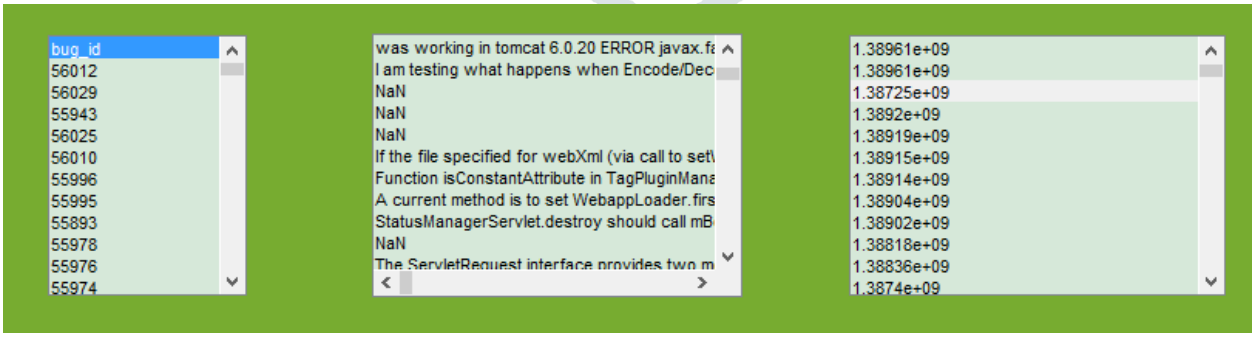


Fig 2: Data Uploaded

The above figure shows the data uploaded in the list boxes after clicking on the upload button and shows the total bug Ids, description of the bugs, timestamps of the arriving bugs which act as the processing data for the classifications of priority levels.

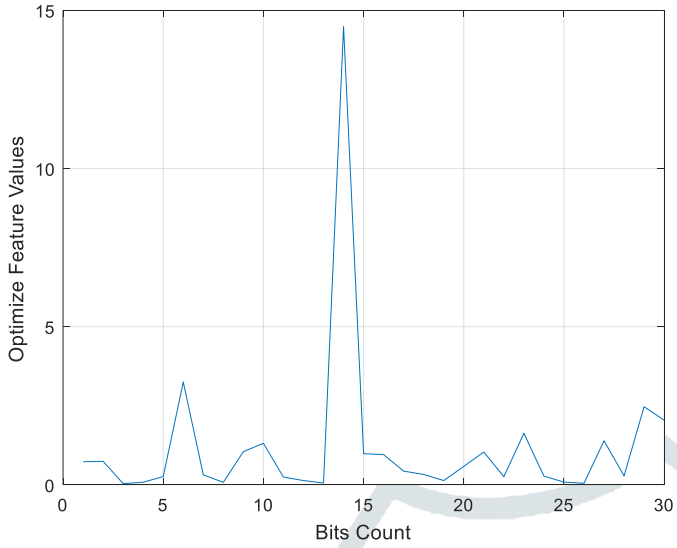


Fig 3: Instance selections

The above figure shows the instance selections of the processes data which deals with selections of relevant feature values which acts as optimize characteristic value for each bug id in terms of entropy and variances of the data.

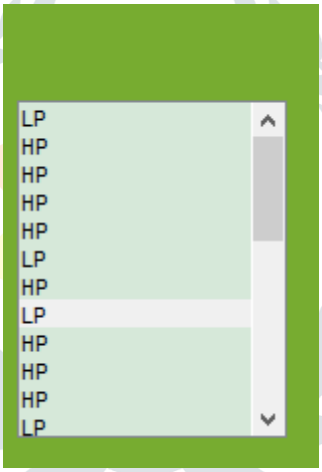
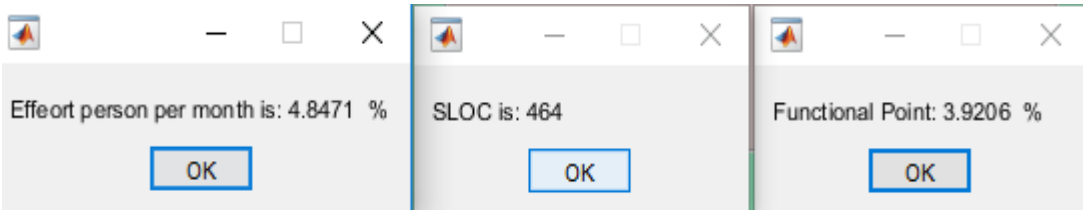


Fig 4: Priority Classifications

The above figure shows the classification of the priority levels which act as categorizing priorities levels based on the bug processing data in terms of high level and low level which shows that which bug is having high priority and low priority to be solved for the evaluation of the performance of the system.

Performance parameters for COCOMO model



The above figure shows the performance evaluations of the COCOMO model which deals with the effort required per person which is coming 4.84 % and function point which is coming 3.92 % and Source lines of code is coming 464.

Performance parameters for PUTNAM model

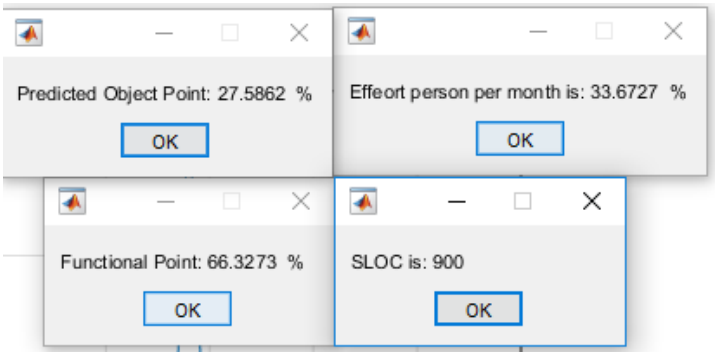


Fig 5: Performance Evaluations

The above figure shows the performance evaluations of the PUTNAM model which deals with the predicted object point which is coming approximate 27.5 % and effort required per person which is coming 33.67 % and function point which is coming 66.32 % and Source lines of code is coming 900. A function point deals with the amount of functionality of the process or an information system provides to a user. Function points are used to compute a functional size measurement of software.

PERFORMANCE USING HYBRID MODEL

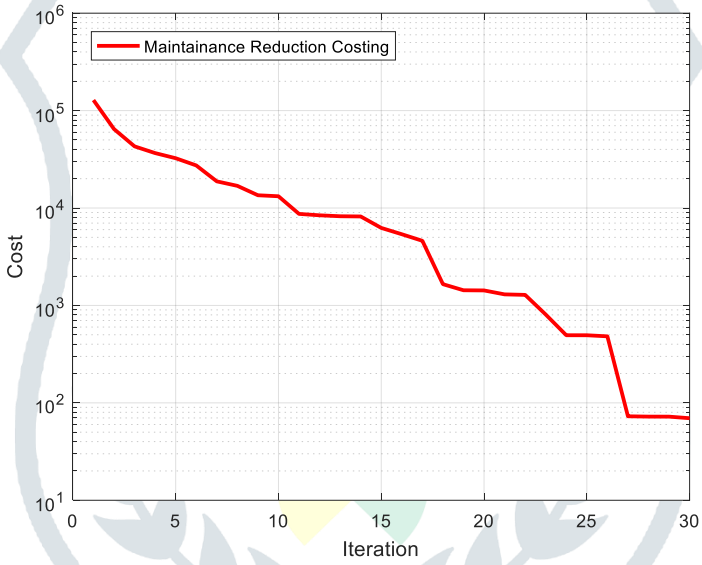


Fig 6: Maintenance Cost Reduction

The fig 6 shows the cost reduction using the proposed hybrid approach which shows that the cost is reducing in the decline fashion and shows that the proposed approach is able to achieve low maintenance cost which is our desired output. The maintenance cost must be reduced for achieving high estimations.

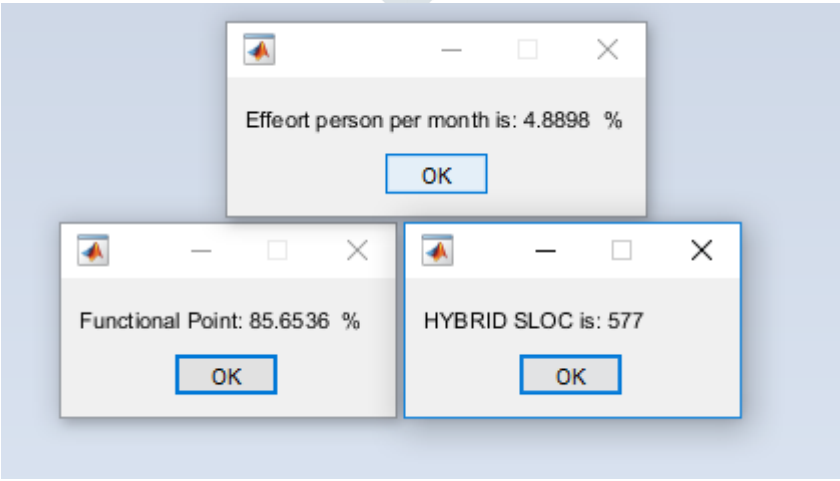


Fig 7: Performance parameters

The fig 7 shows the proposed performance in terms of the functional point which is coming high i.e. 85.65 % and is must be high for high productivity of the software product which increases the demand in the market and functioning of the software developed. Also it is shown that the efforts per person are also low which is of 4.88 % which must be low for efficient cost estimation in terms of maintenance. The source lines of code can be high and can be low according to the development of the software requirement which needs to be developed according to the user prospective.

Performance Evaluation --- Dataset 2 –Bug Report

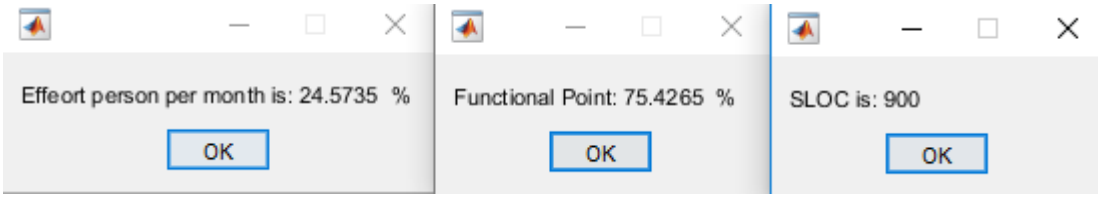


Fig 8: Performance Evaluation using PUTNAM model for Bug prediction Dataset

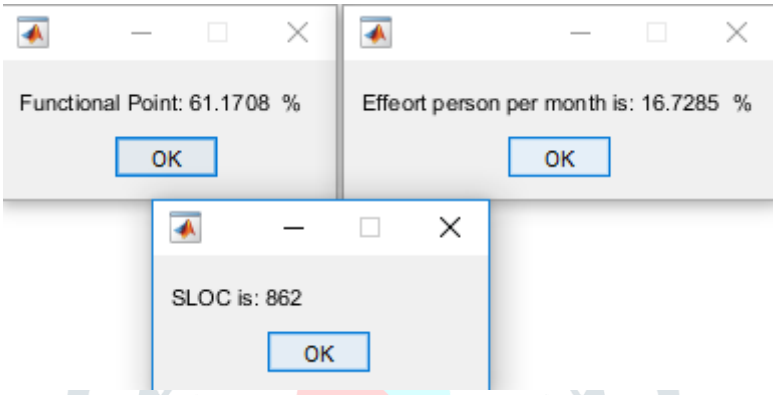


Fig 9: Performance Evaluation using COCOMO model for Bug prediction Dataset

The above figure 8 and 9 shows the performance evaluation for the bug prediction dataset which shows the model performance with the different datasets reliability and shows the functional point and efforts per person performance evaluations

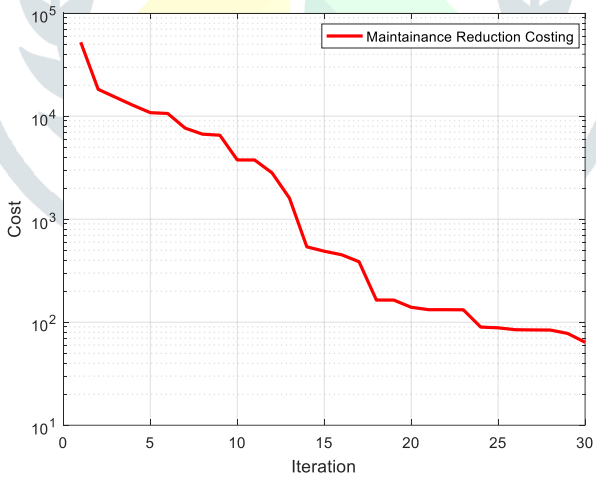


Fig 10: Maintenance Cost Reduction for Hybrid Model with Bug prediction Dataset

The fig 10 shows the cost reduction using the proposed hybrid approach on the bug prediction dataset which shows the reduction of the cost estimations which is our desirable output as its one of the main concern for the reduction of the costs and estimations of higher priority risks.

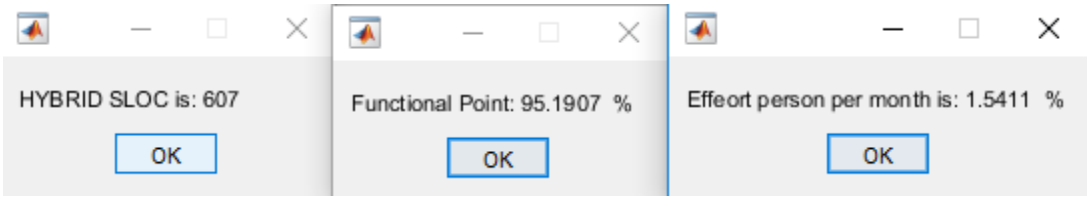


Fig 11: Performance Evaluation of the proposed model for the bug prediction dataset

The fig 11 shows the proposed performance for the bug prediction dataset in terms of the functional point which is coming high i.e. 95.19 % and is must be high for high productivity of the software product which increases the demand in the market and functioning of the software developed. Also it is shown that the efforts per person are also low which is of 1.5411 % which must be low for efficient cost estimation in terms of maintenance.

Table 1: Performance Comparison using TOMCAT dataset

Model	COCOMO	PUTNAM	HYBRID
SLOC	464	900	577
Efforts Per Person	4.84 %	33.67 %	4.89 %
Functional Point	3.92 %	66.32 %	85.65 %

Table 2: Performance Comparison using Bug Prediction dataset

Model	COCOMO	PUTNAM	HYBRID
SLOC	862	900	607
Efforts Per Person	16.72 %	24.5735 %	1.541 %
Functional Point	61.17 %	75.42 %	95.19 %

Conclusion: Software maintenance cost is derived from changes done in software after it has been delivered to end user. According to the up-gradation in the technology there should be up gradation in the software. Further internal issues in software may require maintenance. Due to issues of individual models we have hybrid both COCOMO and PUTNAM model. The proposed performance in terms of the functional point which is coming high i.e. 85.65 % and is must be high for high productivity of the software product which increases the demand in the market and functioning of the software developed. Also it is shown that the efforts per person are also low which is of 4.88 % which must be low for efficient cost estimation in terms of maintenance. The source lines of code can be high and can be low according to the development of the software requirement which needs to be developed according to the user prospective.

REFERENCES

[1] L. H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," IEEE Transaction on Software Engineering, SE-4, no. 4, pp. 345-361, 1978.

[2] B. W. Boehm, "Software Engineering Economics," Englewood Cliffs, NJ: Prentice-Hall, 1981.

[3] R. W. Wolverton, "The cost of developing large-scale software," IEEE Transactions on Computing, pp. 615-636, 1978.

[4] J. J. Bailey, and V. R. Basili, "A meta-model for software development resource expenditures," in Proceedings of 5th Int. Conference on Software Engineering, IEEE/ACM/NBS, pp. 107-116, 1981.

[5] E. A. Nelson, "Management handbook for the estimation of computer programming costs," Systems Development Corporation, AD-A648750, 1966.

[6] R. W. Jensen, "A comparison of the Jensen and COCOMO schedule and cost estimation models," in Proceedings of International Society of Parametric Analysis, pp. 96-106, 1984.

[7] C. E. Walston and C. P. Felix, "A Method of Programming measurement and Estimation," IBM Systems Journal, Vol. 16, no. 1, pp. 54-73, 1977.

[8] Shashank Sharma, Dr. Sumit Srivastava, "Design & Evaluation of Hybrid Framework for Software Maintenance", 2014 IEEE International Advance Computing Conference (IACC), pp. 1459-1468, 2014.

[9] Sen-Tarng Lai, "A maintainability enhancement procedure for reducing agile software development Risk", International Journal of Software Engineering & Applications (IJSEA), pp. 29-40, 2015.

[10] Dawid Zima, "Modern methods of software development", Task quarterly, pp. 481-493, 2015.

[11] Chi-Jui Lin, Dow-Ming Yeh, "A Software Maintenance Project Size Estimation Tool Based On Cosmic Full Function Point", 2016 International Computer Symposium, pp. 555-560, 2016.

[12] Kayhan Moharreri, Alhad Vinayak Sapre, Jayashree Ramanathan, Rajiv Ramnath, "Cost-Effective Supervised Learning Models for Software Effort Estimation in Agile Environments", 2016 IEEE 40th Annual Computer Software and Applications Conference, pp. 135-140, 2016.

[13] Ankit B. Desai, Jekishan K. Parmar, "Refactoring Cost Estimation (RCE) Model for Object Oriented System", 2016 IEEE 6th International Conference on Advanced Computing, pp. 214-218, 2016.

[14] Shahab Nadir, Prof. Detlef Streitferdt, Christina Burggraf, "Industrial software developments effort estimation model", 2016 International Conference on Computational Science and Computational Intelligence, pp. 1244-1252, 2016.

[15] Eddy Ghabach, Mireille Blay-Fornarino, Franjieh El Khoury, Badih Baz, "Clone-and-Own Software Product Derivation Based on Developer Preferences and Cost Estimation", pp. 1-6, 2018.

[16] Suhyun Cha, Quang Huan Dong, Birgit Vogel-Heuser, "Preventing Technical Debt for Automated Production System Maintenance using Systematic Change Effort Estimation with Considering Contingent Cost", 2018 IEEE 16th International Conference on Industrial Informatics (INDIN), pp. 595-601, 2018.