

Enhancing Software Productivity Using Metrics and Improving Software Performance Using Proposed Model: An Empirical Study of Effect of Variance

Pooja Jha and K S Patnaik

Department of CS, JRU Ranchi -835222, Jharkhand India

pooja.jha.ism@gmail.com

Department of CSE, BIT, Mesra Ranchi - 835215, Jharkhand India

kspatnaik@bitmesra.ac.in

Abstract

Determining the success of a software project is an evaluation of performance of the software. It has been always been a matter of concern as developers face hindrance due to issues like cost constraints, schedule constraints, productivity constraints, effort involved and quality maintenance. Developers judge the success or failure of a software project by analysis of defect during the development of software. Hence, the developers pursue various activities during entire software development cycle to ensure elimination of the defects prevailing in the development. Here, the role of metrics cannot be overlooked. A software quality is the magnitude or the measure of acceptance or rejection of software. It is critical issue to measure software quality attributes, and here role of identified set of software metrics becomes important. Software metrics embraces many activities during the software development phases. Also, there are certain parameters based on which the performance of a software project is evaluated. This research paper empirical based study of an organization evaluating the projects on various performance parameters. The research is supported with different statistical methods. At last, the research proposes a regression based model – Productivity with aim to improve the performance of software development process. Residual Analysis is done to validate the proposed model.

Keyword: Software metrics, Defect, Multiple regressions, Effort, COQ

1. Introduction

Software quality is the grade to which software possesses a set of desired attributes.

The contribution of these attributes cannot be overlooked for accessing the quality of the software. Hence, it is critical issue to measure software quality attributes, and here role of identified set of software metrics becomes important. Although, the need for human decision in evaluating quality of software cannot be entirely eliminated with the use of software metrics, but it can make the software quality more visible within an organization or any project.

Software quality is the field of study and practice that describes the enviable attributes of software products. Defect Management Approach and Defect Management Approach are the two ever-present approaches to Software Quality. A software defect can be regarded as any failure to satisfy the end-user requirements. The approach of Defect Management is involved in counting and managing defects. More matured software development organizations use tools like defect leakage metrics and control charts to measure and improve development process capability. The approach of Quality Attributes follows quality models like ISO/IEC 9126 describing a hierarchy of six quality characteristics, dealing with software functionality, dependability, usability, competence, maintainability, and transferable or portability. Software Quality is bifurcated into Software Functional Quality and Software Non-functional Quality.

Software Functional Quality defines suitability of the software to its functional requirements, as well as satisfying the end-users. Software Non-functional Quality involves suitability of the structural requirements in the delivery of functional requirements. Software code and internal structure are associated in this approach.

Software metrics embraces many activities, all of them involving some degree of the measurements. The standard software quality metrics is not a new concept and have been in use for several years. The standard software quality metrics provide useful information and permit the evaluation of trends and the quantifiable analysis of quality, starting with system test of the project. The measurements quantify -the number of faults in generic software, normalized by software size, the responsiveness of development and customer support organizations in resolving customers' problems and the impact of software field fixes on customers. The different parameters (attributes) used to measure software quality are as given in table 1[1]:

The research paper presented is empirical based study of software organization involved in testing and development of software projects. The study is an attempt to identify the different metrics for performance indicators for the organization use. Section 2 discusses the various research works in this area in form of Literature Review. Section 3 deals with a brief introduction of the organization. Research method adopted during the research is discussed in Section 4. Section 5 deals analysis of different projects under study. Section 6 presents statistical analysis of the projects. A regression based model for Productivity is being proposed in Section 7. Section 8 deals results and analysis is presented. Section 9 deals with conclusion of the study. The limitation and future scope are carried during the research is briefed in Section 10.

Table 1: Parameters for measuring Software Quality

Parameters	Characteristic
Testability	Measures the easiness to test the software and to extent it can be tested.
Usability	Measures the extend the software is user-friendly
Understandability	Measures the easiness the software can be made understood to a layman about its functions / purpose.
Consistency	Measures extend the software is consistent in terms of GUI, Conventions.
Efficiency	Defines the amount of work that the software can do in defined timeframe with minimum resources used.
Effectiveness	Measures extent to which the software satisfies the user and meets user requirements.
Accuracy	Measures extent the software works with and gives correct results.
Maintainability	Measures the easiness the software can be maintained in terms of enhancement / new features / bugs fixes
Reliability	Measures the reliability of the software in performing required functions under different conditions.
Portability	Measures the easiness the software can be transported and run in different environments.
Security	Measures extend to which the software is secured in terms of access authorization and personal data like passwords.
Robustness	Measures robustness of the software under unexpected events like software crash, power-off etc and saves its data.
Extendibility	Measures the easiness to test the software and to what extent it can be tested.

2. Literature Review

A number of authors [2] [3] [4] have described measures of defect containment using a variety of terms, including defect removal al efficiency, defect removal al effectiveness (DRE), total containment efficiency/effectiveness (TCE), and phase containment efficiency/effectiveness (PCE). Many times the terms “efficiency” and “effectiveness” seem to be used somewhat interchangeably although they are actually different indicators. In the context of defect containment, one might consider any approach that removes a high percentage of defects present to be a process, even if it is very costly. Hence, defect containment metrics are a valid indication of effectiveness, but do not measure efficiency. Containment metrics is used, if one forecasts the number of defect s likely to be present. Efficiency takes cost into consideration. “Cost per defect” is not a valid metric to compare projects, as it in effect penalizes projects that have high incoming quality-poor quality invariably leads to a lower cost per defect despite higher total cost. The selection of software metrics for building software quality prediction models is a search-based software engineering

problem. Defect prediction models are necessary in aiding project managers for better utilizing valuable project resources for software quality improvement. The efficacy and usefulness of a fault-proneness prediction model is only as good as the quality of the software measurement data.

The study [5] focuses on the problem of attribute selection in the context of software quality estimation. A comparative investigation is presented for evaluating our proposed hybrid attribute selection approach. The results demonstrate that while some feature ranking techniques performed similarly, the automatic hybrid search algorithm performed the best among the feature subset selection methods. In a study [6], an approach to the software characteristics of the software to yield important defect-related predictors of quality was made. Systems were tested until the software passes defect presence criteria and was released. Testing criteria, based on defect count, defect density, and testing efficiency predictions exceeded specified thresholds. In addition, another type of testing efficiency—a directed graph representing the complexity of the software and defects embedded in the code—is used to evaluate the efficiency of defect detection in NASA satellite system software. Complexity metrics were found to be good predictors of defects and testing efficiency in this evolutionary process.

In [7], stress on the fact that as systems change through successive builds, the complexity characteristics of the individual modules that make up the system also change. Changes to software systems were measured on these attribute domains to provide prime indicators of potential problems introduced by the changes. Establishing a measurement baseline permits the comparison of a sequence of successive software builds. Defect density DD, that is number of defects divided by program size, present at time, is considered the de facto measure of software quality and reliability [8]. Paper [9] examines the practices of metrics in the software industry with emphasis on small organizations, explores the challenges and benefits of using software metrics in small organizations, and outlines a practical framework based on the Goal/Question/Metric paradigm for instituting metrics programs in small software organizations. Cost metrics include comparisons of estimated and actual costs [10]. A study [11], suggests, that ten minutes a day should be devoted to schedule inspection activity.

The defect find rate (the number of defects found per hour by testers) helps to determine the cost of testing and to evaluate how stable the system is. Defect densities should be recorded throughout the testing process [12]. An approach was proposed [13], to investigate that whether metrics available in the early lifecycle (i.e. Requirement metrics), metrics available in the late lifecycle (i.e. code metrics) can be used to identify fault prone modules using decision tree based.

Maximum number of defects arrives from only 20 percent of the module. More than half of the module is non-defective. Sometimes even 90 percent of the downtime arrives from at most 10 percent of defects [14].

In an article [15], a measure that can quantify relationship for high rework costs is introduced. A measure called faults-slip-through, determines the faults that would have been more cost-effective to find in an earlier phase. Previous studies [16] [17] [18] [19] [20] [21] were focused on software quality in agile and highly-iterative

development. Mnkandla et al. introduced an innovative technique for evaluating agile methodologies and determined which factors of software quality were improved [16]. Kunz et al. described a quality model, distinct metrics, and their implementation into a measurement tool for quality management [17]. Olaague et al. discussed the fault-proneness of object-oriented classes of highly iterative processes [18]. Roden et al. conducted empirical studies and examined several Bansiya and Davis quality factor models [19]. Jeon et al. [20] focused on the metrics and maturity of iterative development. These studies provided a new evaluation viewpoint of software quality and showed the advantages of agile methodologies through experiments. However, their results cannot be compared with past products measured using traditional metrics. At NTT, the inability to compare has been an obstacle for transformation from traditional development to highly iterative one.

Software defect prediction has recently attracted attention of many software quality researchers. In a paper by [22] proposed various classifications and clustering methods with an objective to predict software defect. They analyzed using classification and clustering techniques. The performance of three data mining classifier algorithms named J48, Random Forest, and Naive Bayesian Classifier (NBC) were evaluated based on various criteria.

Production of high-quality software is the exquisite need for accomplishing absolute customer satisfaction in software industry. Effective defect management is one of the crucial factors enabling successful development of high-quality products. Inspection and testing are two established methods of defect management domain. The paper [23] indicates the need for awareness and use of quality measurement of process and people in realizing effective defect management.

Another paper [24] investigated defect prediction with data from a family of widely used OSS projects based both on product and project metrics as well as on combinations of these metrics. Thus, the combined application of project and product metrics can improve the accuracy of defect prediction; enable a better guidance of the release process from project management point of view, and help identifying areas for product and process improvement.

An article [25], presented an industry experience report on the application of SPC (Statistical Process Control) in a Software Verification and Validation Unit at an Information Technology Division from a financial institution. It was concluded SPC generates an improvement in the coverage maturity level 4 practices. Another paper [26] introduced a novel hybrid method of random forest (RF) and Fuzzy C Means (FCM) clustering for building defect prediction model. Random forest algorithm is used to perform a preliminary screening of variables and to gain an importance ranks. Subsequently, the new dataset is input into the FCM technique, which is responsible for building interpretable models for predicting defects.

Defect prediction is important in order to reduce test times by allocating valuable test resources effectively. The paper [27] proposed a model using multivariate approaches in conjunction with Bayesian methods for defect predictions. The proposed model with the best results reported so far on public datasets and concluded that using multivariate approaches can perform better. A review and analysis of object oriented metrics is presented [28] for identification and validation

of object oriented metrics and out of various metrics, a limited set of metrics is identified that have a significant role in software complexity and quality measurement. An accurate prediction of the number of defects in a software product during system testing contributes not only to the management of the system testing process but also to the estimation of the product's required maintenance. A new approach, called Estimation of Defects based on Defect Decay Model (ED3M) is presented [29] that compute an estimate of the total number of defects in an ongoing testing process. ED3M is based on estimation theory.

3. Brief Profile of Organization 'XYZ'

Organization 'XYZ' represents the connected world that offers innovative and customer-centric information technology services and solutions. Organization 'XYZ' has more than 95,300 professionals across 51 countries, providing services to 649 global customers including Fortune 500 companies. The Consulting, Enterprise and Telecom solutions, platforms and reusable assets connect across a number of technologies to deliver tangible business value to all our stakeholders.

4. Research Method

The research work begins with identification of the various key performance indicators. These indicators determine the major contributors affecting a software project under development. A Life Cycle Development Profile of various projects is presented next in the paper. Various analyses are done for productivity and rework effort. Different statistical methods are used to test the data collected from study. Finally, two regression based models are developed and validated using different statistical analysis.

4.1 Key Performance Indicators for Project under study

The research work presented in this paper is focused on the six performance indicators. These attributes are Productivity, Quality, Defect Detection, Schedule variance, Effort Variance and CoQ (Cost of Quality). Management of a project requires the accurate monitoring of several key data points and types throughout the lifecycle of a particular project or program. Key Performance Indicators (KPIs) are the measurements defined to show the state of critical strategic goals or targets the organization has set for itself. KPIs, calculated by using sets or combinations of metrics is a measure of performance, commonly used by an organization to define and evaluate how successful it is in terms of making progress towards its long-term organizational goals. For managing any project, these parameters or attributes can play a major role in success of any project.

Software Productivity is said as deceptively simple concept, but a matter of debate [1]. There are a number of ways to measure productivity like FPA, Cyclomatic Complexity etc. The concept of software productivity is not a theoretical abstract but is a critical part of software engineering process. Software metrics that can be used as quantifiable measure of various characteristics of a software system need to be

established [30]. The metrics capture both the effort required to produce the software and the functionality provided to the software consumer. Software development companies constantly seek ways to increase both developer productivity and code quality. Software Productivity can be enhanced by reusing code to leverage existing programs, using reliability initiatives to minimize rework and adopting sound development practices and standards.

Another performance indicator is Quality. The cost of making a product can rise dramatically and profits can suffer if quality suffers at the expense of increased production rates. Monitoring quality indicators can lead to increased production rates and improved product quality. Software quality measurement quantifies the extent a system or software possesses desirable characteristics.

Defect detection can also be a major key performance indicator. Entire software development life cycle intrudes defects in some way or another; and testing helps to find as many of them as possible when they are inserted. About 60% of defects are inserted in the requirement specification. The job of tester is to report on completion of the project, the defect detection efficiency (DDE) [31].

Schedule Variance, yet another performance indicator, is the ratio of difference between the Actual End Date and Planned End Date to the difference between Planned End Date and Planned Start Date for the project. The objective of this indicator is to reduce the schedule variation by tracking it from beginning stage of the project through the end of the project, thereby plummeting time overruns. Metric for Schedule Variation is mainly used as an indicator for capability to meet milestones. It should be measured at overall project level. Schedule variation need to be calculated only when the stage is completed.

Performance indicator, Effort Variation is the difference between Estimated and Actual effort as compared against the Estimated Effort. The objective of this indicator is to study the distribution of workload by Stage and to reduce the deviation of the actual effort expended as against the estimated effort. It should be measured at overall project level. COQ basically consist of 3 types of cost – Cost for Prevention, Cost for Appraisal and Cost due to Failure. Failure Cost can be further divided into Internal Failure and External Failure. Effort is used to measure cost. This indicator evaluates the effort in terms of time spent on reviews, testing and rework against the production effort. It is useful to calculate the total effort spent for QA in a project.

4.2 Life Cycle Development Profile of the Projects under study

The most pertinent analysis in a software project is to view the life cycle of the project and to recognize process outcomes in life cycle phases. A series of life cycle phase analyses is expressed in the form of profiles. Each Life Cycle Profile (LCP) provides connectivity among phases so that project events are arranged in a natural order in rhythm with the workflow and giving the complete picture of the project at a glance. If the metric is defect, the profile gives clues about process maturity. If the metric is rework, the profile provides causal readings into cost control and could become an eminent problem definition for cost reduction initiatives. Risk can also be perceived from some profiles.

4.3 Effort Profile

Effort profiles for five projects are shown in the table 2 below. The effort profile can identify the following features in the development of project:

- The phase where effort peaks
- The share of effort devoted to requirements and design
- The share of effort given to testing
- The ratio of design effort to code effort
- The percentage of effort on project management.

Table 2: Effort Profile Vs Size for Various Projects

Project Id	Size (FP)	Requirement Analysis	Design	Coding	Testing	Project Management	Training
PID1	1565	32.54	2.18	100.00	38.47	18.55	9.99
PID2	3021	5.02	70.28	11.65	5.02	2.01	0.00
PID3	214	11.27	50.00	19.37	19.37	0.00	0.00
PID4	327	10.03	11.10	43.72	39.62	6.98	0.00
PID5	557	12.84	10.63	57.45	16.54	1.23	1.31

It is observed from the above table that the effort involved in coding for software project PID 1 is leading in comparison to other phases of software development. Next, PID 5 shows the highest contribution in coding in terms of effort, followed by PID 4, PID 3 and PID 2. A very small proportion of effort (5.02) is involved in Requirement analysis for PID 2 having size of 3021 FP whereas, PID 1 has an effort of 32.54 involved in Requirement analysis. Effort involved in testing is highest for PID 4 of size 327 FP and minimum for PID 2.

5. Analysis of Projects under Study

The five projects under study have been analyzed for better imminent about these projects. The projects are compared using the productivity graph, rework distribution and team size distribution.

5.1 Productivity graph

Projecting productivity is all about measurements. These measurements involve the rate at which a software engineer produces software and the accompanied documentation. Productivity is defined as the ratio of size in FP to effort in PM. The Productivity Graph highlighting the relationship for the projects under consideration is shown in figure 1 below.

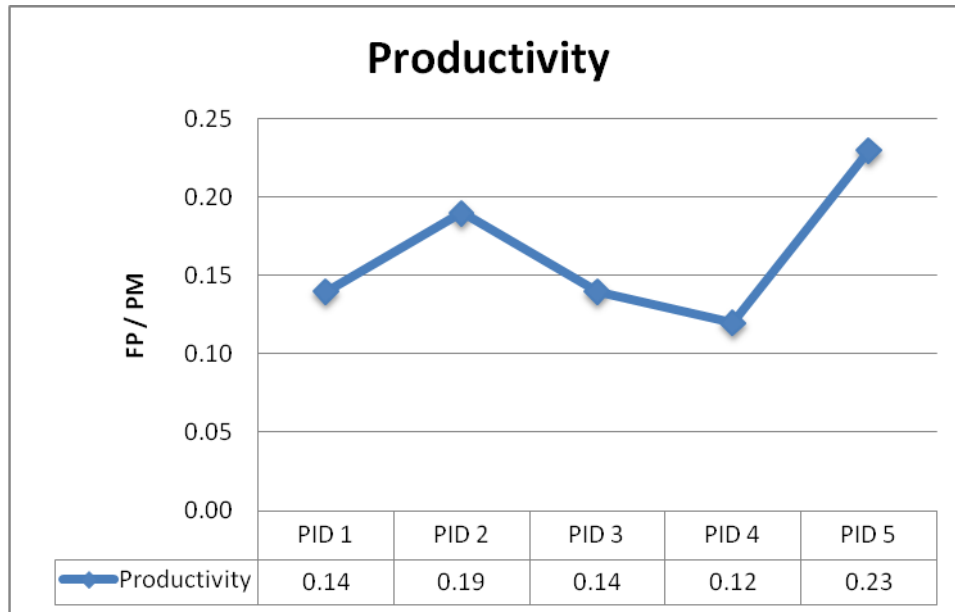


Figure 1: Productivity Graph

Figure 1 above shows the productivity graph for the five projects. The project referred by their id is shown as PID 1 to PID 5 in the above graph. The project id PID 5 has the highest productivity of 0.23 or 23 %, while project PID1 and PID3 has 0.14 or 14%. Project id PID 2 has productivity of 0.19 or 19% and PID 4 has a productivity of 0.12 or 12%.

5.2 Distribution of rework

Current software projects spend about 40% to 60% of their effort on avoidable rework. The rework consists of effort spend in fixing difficulties. According to [32], about 80% of avoidable rework comes from 20% of defects.

Reducing rework effort will have bigger benefit to the client especially when the client's time is involved in reviewing the artifacts. The impact of reducing rework is the number of defects reported during the peer review and the cost saving of lowering such defects before going to the client. The rework can be measured as in equation 1 below.

$$\text{Rework Effort} = \frac{(\text{Effort for Reviews and Rework on Test Cases})}{(\text{Effort for Test Case Preparation})} \times 100\% \tag{1}$$

The rework in terms of effort is shown below in table 3 for the five software projects under study. It is observed that the rework (effort) is highest 2010 PM for project id PID 1 and is lowest for PID 4, having a value of 48.

Table 3: Rework (effort) Distribution among projects

Project Id	Planned Effort	Actual Effort	Rework (Effort)
PID 1	7650	11312	2010
PID 2	11200	15936	1744
PID 3	1631	1580	50
PID 4	2736	2814	48
PID 5	2100	2437	56

Above table shows that for PID1 the actual effort is more than the planned effort and effort involved in rework can be a factor for the observed deviation. It is highest for PID1 and lowest for PID4.

5.3 Distribution of Team Size of projects under study

The software projects are handled by different teams in organization XYZ. A good team must be having good knowledge of resolving problems as well as experience. Table 4 shows the team size distribution of the various projects. The projects have been developed on different language platforms like Java, WebServices, VXML, Shell script, Android, C Objective, J2EE.

Table 4: Team size and languages in various projects

Project Id	Team Size	Platform/Languages
PIDI1	12	Objective C, J2EE
PIDI2	12	Java, WebServices, VXML, Shell script
PIDI3	6	Java, WebServices
PIDI4	4	Java
PIDI5	12	Android, C Objective

6. Statistical Analysis of Projects under Study Using Metrics

The data collected for study from five different projects of Organization XYZ have been tested using statistical analysis. The different analysis includes Schedule Variance (SV), Effort Variance (EV), Productivity Variance (PV), Quality Variance (QV), Defect Density Variance and COQ Variance.

For tracking and control of projects effort variance and schedule variance are chosen as metrics that are derived from the two measurements, effort and time. Schedule Variance (SV) indicates the difference in the actual and planned schedule of a project calculated at a given point of time. Schedule Variance of a project can also be found in terms of earned value at any given point of time as the difference between Earned Value and the planned budget. The Schedule Variance is calculated by the formula given in equation 2 as:

$$\text{Schedule Variance} = \{(\text{Actual Duration} - \text{Estimated Duration}) / \text{Estimated Duration}\} * 100 \quad (2)$$

Effort Variance (EV) is a simple metric which measures deviation from estimated effort for a particular piece of work. Schedule variance along with effort variance can provide information about resource utilization. Effort variance indicates estimation capability as well as project implementation skills, and could contain signals in both directions. But the value of this metrics depends on the dependability of estimation and planning process. Poor estimation also makes the metric poor. Unreliable planning makes this metric equally unreliable. It can be calculated as in equation 3 as:

$$\text{Effort Variance} = \{(\text{Actual Efforts} - \text{Estimated Efforts}) / \text{Estimated Efforts}\} * 100 \quad (3)$$

Software Productivity can be increased by increased code knowledge, reducing rework and reduced debugging. Productivity Variance (PV), another metric, can be used to understand the variations. It is a simple metric given by formula as in equation 4 as below:

$$\text{Productivity Variance} = \{(\text{Actual Productivity} - \text{Estimated Productivity}) / \text{Estimated Productivity}\} * 100 \quad (4)$$

In simple terms, software quality is the totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs. To measure the deviation between actual and planned /estimated quality, metric named Quality Variance (QV) is evaluated as given by equation 5.

$$\text{Quality Variance} = \{(\text{Actual Quality} - \text{Estimated Quality}) / \text{Estimated Quality}\} * 100 \quad (5)$$

Defect Density as defined, is the number of confirmed defects detected in software/component during a defined period of development/operation divided by the size of the software/component. Defect Density Variance (DDV) is the metric used for measuring the deviation between planned and actual defect density. It is given as in equation 6.

$$\text{Defect Density Variance} = \{(\text{Actual Defect Density} - \text{Estimated Defect Density}) / \text{Estimated Defect Density}\} * 100 \quad (6)$$

COQ can be stated as sum of Cost of Control and Cost of Failure of Control. The cost of control includes prevention cost and appraisal cost. Cost of Failure of Control includes internal failure cost and external failure cost. The COQ Variance (COQV) can be an important metric for organization to know the deviation between planned and actual COQ and plan for improvement tasks if negative variation occurs.

This is shown in equation 7 below.

$$\text{COQ Variance} = \{(\text{Actual COQ} - \text{Estimated COQ}) / \text{Estimated COQ}\} * 100 \tag{7}$$

The data obtained using the above metrics is shown below in figure 2-7. From fig 2, it is observed that three of the five projects are having schedule variance of 0, meaning there is no difference between the planned and actual schedule for these projects. Project with id PID 2 and PID 5 shows that these projects are having positive deviation means these reports are 16.44% and 3.19% respectively, ahead of planned schedule. Figure 3 interprets project id PID 2 is 42.55% ahead in terms of its productivity whereas it is 23.58% for project id PID 4. For quality, it can be interpreted from figure 4 that all the projects are having 0 values, meaning there is no difference between the planned and actual quality parameters. For defect density, it is seen that project id PID 4 is having highest value of 23.77%; and lowest of 6.99 for project id PID 5 (figure 7). Figure 6 shows project id PID 1 is having 30% in COQ variance, recorded highest and lowest is recorded for PID 4 of 8.23%. Figure 7 shows effort is 18.57 for id PID 2 and 4.97 for id PID 4.

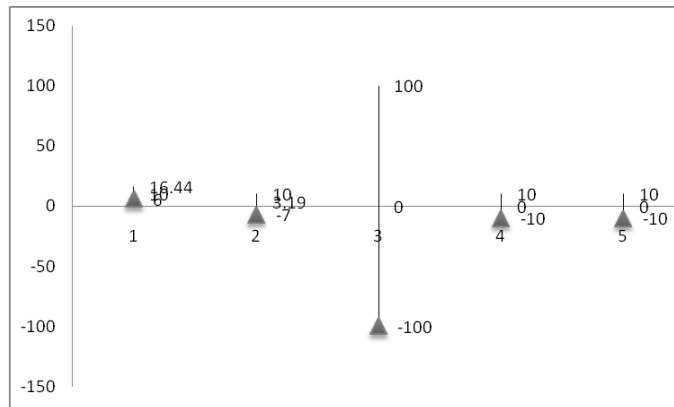


Figure 2: Schedule Variance

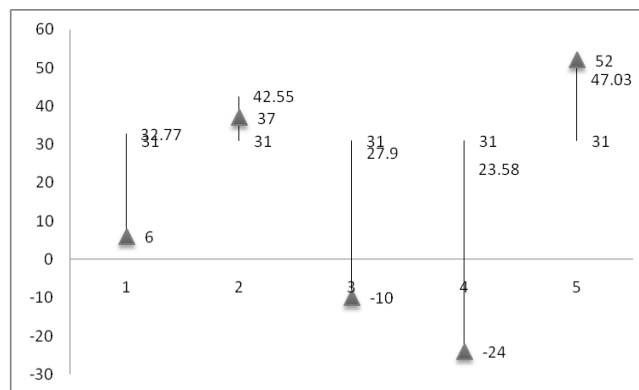


Figure 3: Productivity Variance

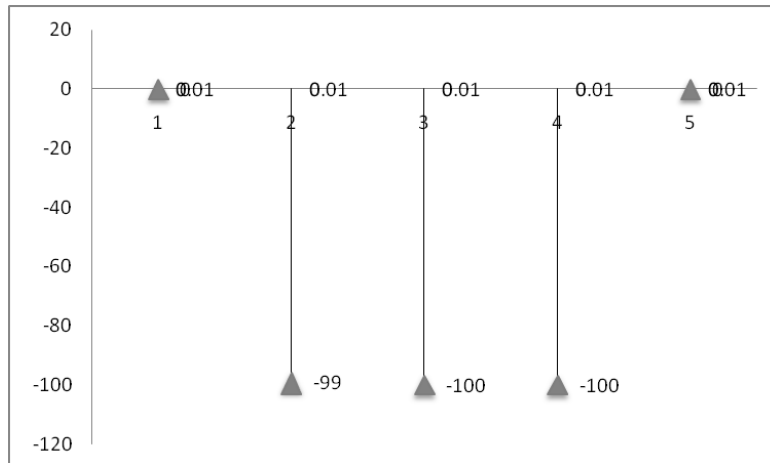


Figure 4: Quality Variance

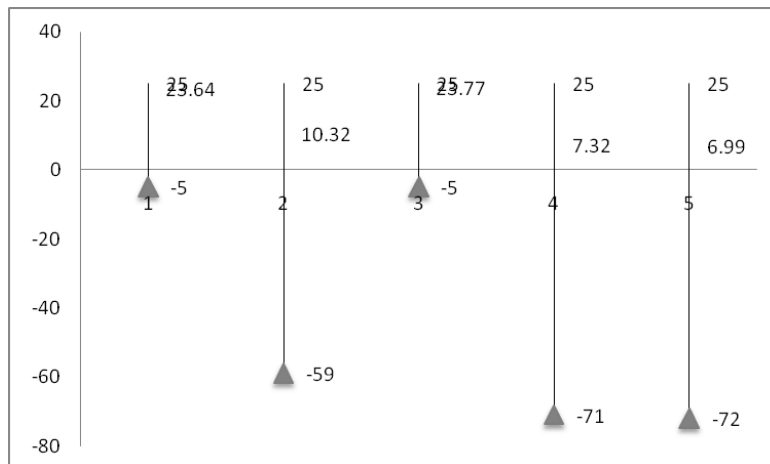


Figure 5: Defect Density Variance

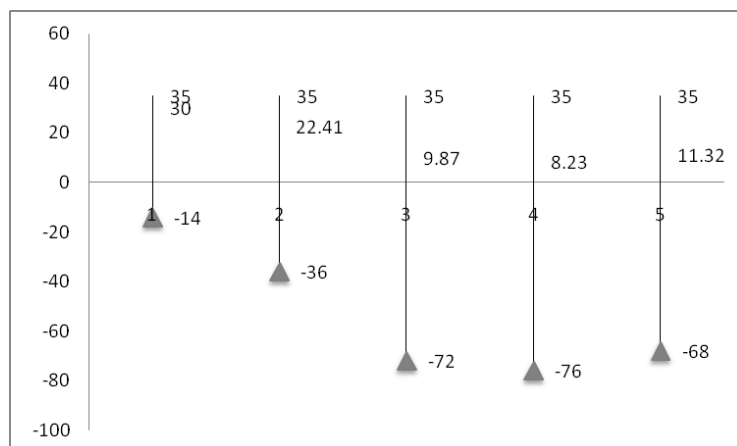


Figure 6: COQ Variance

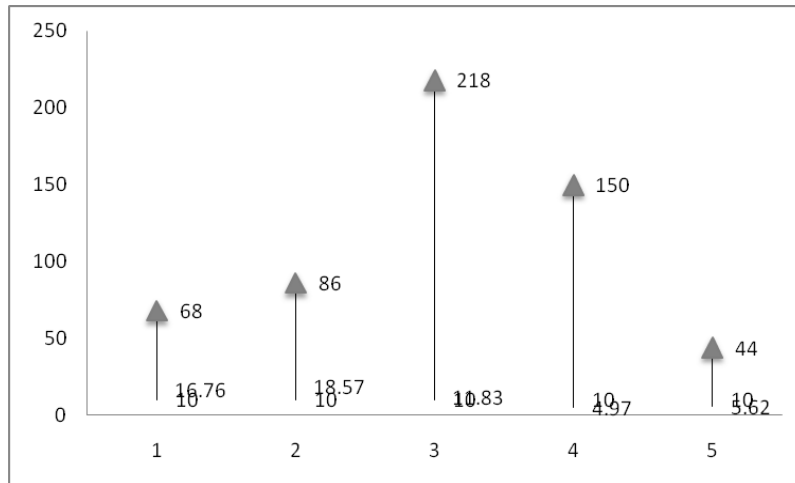


Figure 7: Effort Variance

7. Proposed Model

The research paper attempts to develop non-linear regression models and tries to validate the model using residual analysis. The model, Productivity, is regression model which takes three independent variables namely, size, schedule variance and defect density. The general equation for multiple regressions is given as in equation 8.

$$y = a + b_1x_1 + b_2x_2 + b_3x_3 \quad (8)$$

The variables used are as-

y : the value of the Dependent variable (y), what is being predicted or explained

a (Alpha) : the Constant or intercept

b_1 : the Slope (Beta coefficient) for x_1

x_1 : First independent variable that is explaining the variance in y . Here, x_1 is effort variance

b_2 ; the Slope (Beta coefficient) for x_2

x_2 :Second independent variable that is explaining the variance in y . Here, x_2 is schedule variance

b_3 : the Slope (Beta coefficient) for x_3

x_3 :Third independent variable that is explaining the variance in y . Here, x_3 is size in FP.

The regression equation obtained from the data collected is presented in equation 9 as:

$$y = 34.854 + 0.00034345x_1 + 0.67533x_2 + (-)0.2172x_3 \quad (9)$$

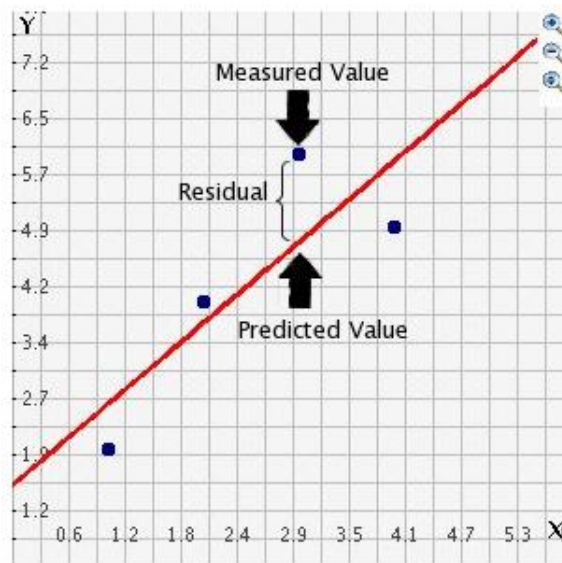
The model assumes that Productivity is predicted to increase 0.000343 when the size variable goes up by one, decrease by 0.217 when defect density variable goes up by one, increase by 0.675 when schedule variance variable goes up by one and is predicted to be 34.85 when all variables are null.

8. Results and Analysis

The model proposed for the organization is statistically verified using residual analysis. Because a linear regression model is not always appropriate for the data, the appropriateness of the model is assessed by defining residuals and examining residual plots.

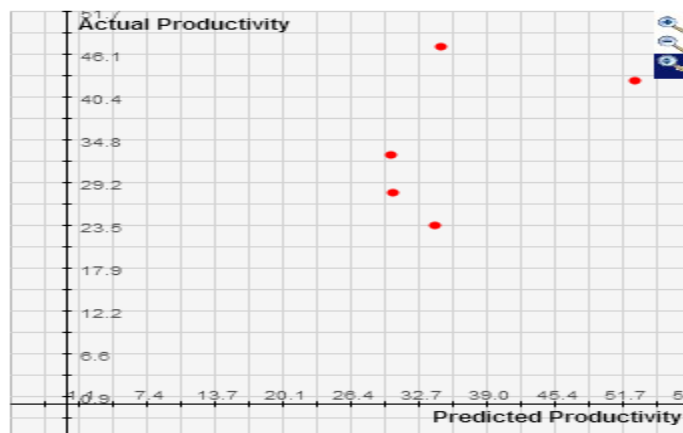
The residuals from a fitted model are the differences between the responses observed at each combination values of the explanatory variables and the corresponding prediction of the response computed using the regression function. Mathematically, the definition of the residual for the *i*th observation in the data set is written in equation 10. This is shown graphically in figure ---below.

$$\text{Residual} = y(\text{ observed}) - y(\text{ predicted}) \tag{10}$$



The result obtained is shown in figure---

Residual Analysis



The value of R is 0.5066. This is a moderate positive correlation, which means there is a tendency for high X variable scores go with high Y variable scores (and vice versa). The value of R², the coefficient of determination, is 0.2566.

9. Conclusion

From the above study, it is evident that defect detection and prevention during various phases of software development can improve various parameters of performance of an organization. Defect management should be an ongoing and rigorous process so that the software project under development can be completed successfully within the constraints of schedule and cost. Defect prevention activities not only saves time, but also prevents the extra effort involved in rework to be done for fixing of the encountered defects and helps in improving the quality of the software product. The research work proposes a regression based model Productivity for evaluating the various parameters. Using various statistical methods, the validity of these models is verified.

10. Limitations of Study and Future Scope

The research work presented in this paper is constricted to five software projects handled by the organization A due to privacy reasons. The model could be more appropriately tested perhaps if more data were available. Also the research could have been expanded more widely, if more different data from various organizations would be collected. This lays in the future scope of the study. Also, the work is intended to be extended in the study defect management in various phases of software development.

References

- [1] Robert Sidler MSIS 488 Fall 2002
- [2] Jones, C., 2008, Applied Software Measurement, 3rd edit ion. New York: McGraw Hill
- [3] Jones, C., 2010, Software engineering best practices, New York: McGraw Hill.
- [4] Kan, S., 2003. Metrics and Models in software quality engineering, 2nd edit ion. Boston: Addison- Wesley.
- [5] Gao, Kehan, et al. , 2011, "Choosing software metrics for defect prediction: an investigation on feature selection techniques." Software: Practice and Experience 41.5 (2011): 579-606.
- [6] Schneidewind, Norm, and Mike Hinchey, 2009, "A complexity reliability model." 20th IEEE International Symposium on. Software Reliability Engineering.

- [7] Munson, J.C., Werries, D.S., 1996, "Measuring software evolution", Proceedings of the 3rd International Software Metrics Symposium, pp. 41 – 51
- [8] N.E. Fenton, N. Ohlsson, 2000, "Quantitative Analysis of Faults and Failures in a Complex Software System", IEEE Trans. on Software Engineering, 26(8):797-814, August
- [9] Hisham M. Haddad, Nancy C. Ross, and Donald E. Meredith, 2012, "A Framework for Instituting Software Metrics in Small Software Organizations", International Journal of Software Engineering (IJSE) Vol.5 No.1, pp 69-98, January
- [10] F.J. Koch, 2002 "Metrics and the Immature Software Process," <http://www.qpmg.com/metrics.htm>
- [11] V. Rubenstein and J. Boria, Small Organizations, 2007, "Small Interventions, the IT Metrics and Productivity Institute: Best Practices in Software Development, Management, and Maintenance," .
- [12] Stephen H. Kan, 2003, "Metrics and Models in Software Quality Engineering"
- [13] Parvinder S. Sandhu, Amanpreet S. Brar, Raman Goel, Jagdeep Kaur, Sanyam Anand, 2010, "A Model for Early Prediction of Faults in Software Systems", The 2nd International Conference on Computer and Automation Engineering (ICCAE), Vol 4, pp. 281-285
- [14] B. B. and Victor R Basili, 2001, "Software defect reduction top 10 list," IEEE, vol. 15, no. 1, pp. 795–825
- [15] Denis Duka, Lovre Hribar, 2010, "Fault Slip Through Measurement in Software Development Process", IEEE ELMAR Proceedings, pp. 177 – 182
- [16] E. Mnkandla, B. Dwolatzky, 2006, "Defining Agile Software Quality Assurance", Proceedings of the International Conference on Software Engineering Advances
- [17] M. Kunz, R. Dunke, N. Zenker, 2008, "Software Metrics for Agile Software Development", the 19th Australian Conference on Software Engineering, pp.673-678,
- [18] H. Olague, L. Etzkorn, A. Gholston, S. Quttlebaum, 2007, "Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes", Vol.33 No.6, IEEE Transaction on Software Engineering
- [19] P. Roden, S. Virani, L. Etzkorn, S. Messeimer, "An Empirical Study of the Relationship of Stability Metrics and QMOOD Auality Models Over Software Development Using Highly Iterative or Agile Software Processes", Seventh IEEE International Working Conference on Source Code Analysis and Manipulation, pp.171-179
- [20] S. Jeon, M. Han, E. Lee, K. Lee, 2011, "Quality Attribute driven Agile Development", Ninth International on Software Engineering, Management and Applications, pp.203-210.

- [21] T. Fujii, T. Dohi and T. Fujiwara, 2011 "Towards quantitative software reliability assessment in incremental development processes," Proceedings of ICSE, pp.41-50
- [22] Chug, Anuradha, and Shafali Dhall, 2013, "Software Defect Prediction Using Supervised Learning Algorithm and Unsupervised Learning Algorithm."
- [23] T.R. Gopalakrishnan Nair, V. Suma, P. Kumar Tiwari, 2012, "Significance of depth of inspection and inspection performance metrics for consistent defect management in software industry", IET Software., Vol. 6, Issue 6, pp. 524 – 535
- [24] Dindin Wahyudin, Alexander Schatten, Dietmar Winkler, A Min Tjoa, Stefan Biffl, "Defect Prediction using Combined Product and Project Metrics A Case Study from the Open Source "Apache" MyFaces Project Family", 34th IEEE Euromicro Conference Software Engineering and Advanced Applications, pp 207-215
- [25] Carla Fernández-Corrales, Marcelo Jenkins, Jorge Villegas, 2013, "Application of Statistical Process Control to Software Defect Metrics: an Industry Experience Report", ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 323-331,
- [26] Pushphavathi, T.P. Suma, V. ; Ramaswamy, V., 2014, "A novel method for software defect prediction: Hybrid of FCM and random forest ", IEEE International Conference on Electronics and Communication Systems (ICECS), pp. 1-5
- [27] Turhan, B., Bener, A., 2007, "A Multivariate Analysis of Static Code Attributes for Defect Prediction", IEEE Seventh International Conference on Quality Software, pp. 231 - 237
- [28] Mukesh Bansal, C.P.Agrawal, 2014 , "Critical Analysis of Object Oriented Metrics in Software Development", IEEE Fourth International Conference on Advanced Computing & Communication Technologies (ACCT), pp. 197 - 201
- [29] Syed Waseem Haider, João W. Cangussu, Kendra M.L. Cooper, Ram Dantu, 2008, "Estimation of Defects Based on Defect Decay Model:ED3M ", IEEE Transactions On Software Engineering, Vol. 34, No. 3, May/June
- [30] Bordoloi, Bijay and Joe Luchetski , 2000, "Software Metrics: Quantifying and Analyzing software for Total Quality Management", Systems Development Handbook (P. Tinnirello, ed) 4 ed. CRC Press LLC,
- [31] Grant Obermaier, 2010, Early Defect Detection, Transition Consulting Limited – India, Feb 11
- [32] Software Engineering:, 2007, "Barry W. Boehm's Lifetime Contributions to Software Development, Management and Research", edited by Richard W Selby, Published by John Wiley and Sons, New Jersey