Proposed Metrics for Process Capability Analysis in Improving Software Quality: An Empirical Study

Pooja Jha*, K S Patnaik**
* Department of CS, JRU, Ranchi, India
** Department of CSE, BIT, Mesra, Ranchi, India

Article Info

Article history:

Received Jun 12th, 2015
Revised Aug 20th, 2015
Accepted Aug 26th, 2015

Keyword:

Metrics
Organization Baselines
Process Capability Analysis
AD GoF tests
Quality

ABSTRACT

A software project faces its top expense on defect removal; thereby delaying the schedules. There has been increasing demand for high quality software. Here, high quality software means, delivering defect free software and meeting the predictable results within time and cost constraints. Software defect prediction strives to improve software quality and testing efficiency. The research work presented here is an empirical study and analyzes importance of different metrics used in the organization. The paper examines the impact of LSL and USL, known as organization baselines, on various projects and proposes four metrics for capability analysis metrics. These can prove beneficial for categorizing the process of software development. These metrics aim to improve the ongoing software development process and are helpful in determining the quality of these processes in terms of their specification limits. Also, the paper aims to test if a sample of data came from a population with a specific distribution. This has the advantage of allowing a more sensitive test and the disadvantage that critical values must be calculated for each distribution.

Corresponding Author:

Pooja Jha,
Department of CS,
JRU, Ranchi,
India.
Email: pooja.jha.ism@gmail.com

1. INTRODUCTION

There has been increasing demand for high quality software. Here, high quality software means, production and delivery of defect free software and meeting the predictable results within constraints of time and cost. A defect in a software product is observed as an imperfection and undesired behavior that prevent the product from maintain its normal behavior in terms of quality. There is a need for managing these defects that may occur in the product. When a defect is detected in the initial phases of the software development, it not only saves time, cost and resources to rework on that product but also prevents the migration of defects from one phase to another; thereby not letting additions in number of defects. Defect Detection and Defect Prevention are considered as the two most common approaches towards defect management. Defect Detection is the technique involved in detecting defects and its origin whereas Defect Prevention is defined as the technique for minimizing these defects and preventing them from reappearing in the product. A software project faces its top expense on defect removal; thereby delaying the schedules. An effective defect removal can lead to minimization of the development time and production of good quality software. As with increase in complexities of software added functionalities and hardware, changing business requirements and increasing competitive nature of business world, organizations are stressing on development of high quality software product requirements. It is advisable for the organizations to undergo defect detection in early stages of software development so as to gain confidence of the probable customers.

Process capability analysis (PCA) and Six Sigma methodology occupy crucial places in quality and process improvement initiatives. As a fundamental technique for quality and process improvement efforts, PCA is used to upgrade processes, products or services to maintain higher levels of customer satisfaction.

The research paper presented is based on a case study of software organization involved in testing and development of software projects. The study focus to analyze the use of metrics in projects (study is based on nine of such projects) developed and tested by the organization. Some commonly used metrics in development and testing of projects are Defect Density (DD), Cost of Quality (COQ) and Defect Removal Efficiency (DRE). The study proposes four metrics for improving the development process of these projects by analyzing the capability of ongoing software development processes. Further, the study uses Anderson–Darling test which is a goodness-of-fit test that allows to control the hypothesis that the distribution of a random variable observed in a sample follows a certain theoretical distribution. In particular, it allows to test whether the empirical distribution obtained corresponds to a normal distribution. Section 2 discusses the various research works in this area in form of Literature Review. Section 3 discusses the objectives of the research paper. A brief about company profile is presented in Section 4. Section 5 deals with new proposed metric. Section 6 presents the results and discussions on findings. A conclusion and future scope is presented in Section 7. The limitation carried during the research is briefed in Section 8.

2. LITERATURE REVIEW

Software defect prediction strives to improve software quality and testing efficiency by constructing predictive classification models from code attributes to enable a timely identification of fault-prone modules [1]. A framework for comparative software defect prediction experiments was proposed and applied in a large-scale empirical comparison of 22 classifiers over 10 public domain data sets from the NASA Metrics Data repository. The result showed an appealing degree of predictive accuracy supporting the view that metric-based classification is useful. It is said that finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during requirements and design phase [2]. It is also said that about 80% of the avoidable rework comes from 20% of defects. The paper [3] defines questions regarding defect detection techniques and presents a survey of empirical studies on testing and inspection techniques.

The effort [4] is an analysis based on data obtained for five selected projects from leading software companies of varying software production competence. The purpose of the paper was to provide information on various methods and practices supporting defect detection and prevention leading to thriving software generation. On an average 13 % to 15% of inspection and 25% - 30% of testing out of whole project effort time is required for 99% - 99.75% of defect elimination. Advancement in fundamental engineering aspects of software development enables IT enterprises to develop a more cost effective and better quality product through aptly organized defect detection and prevention strategies. The work [5] analyzed data obtained for five different projects from progressive software company of varying software production capabilities. The defect prevention technique involved proactive, reactive and retrospective moves to uncover 70% of defects during inspections and developer unit testing. The validation testing uncovered about 29% of defects. Inspection becomes imperative in creating much more ideal software in factories through enhanced methodologies of aided and unaided inspection schemes.

In many low mature organizations, defects are detected rather late in the development process. High rework and testing effort, typically under time pressure, lead to unpredictable delivery dates and uncertain product quality. The paper in [6] discussed the importance of life cycle modeling for defect detection and prevention and presented a set of concrete, proven methods that can be used to optimize defect detection and prevention. In particular, software inspections, static code analysis, defect measurement and defect causal analysis were discussed. Another paper [7] discussed the results to date of a series of e-Workshops on software defect reduction. The reformulated heuristics can be useful both to researchers and to practitioners.

The quest for solutions to stay competitive in the marketplace brings software industries to take steps to control their products and processes by introducing metrics as part of their quality system and setting triggers for action when their capability limits are exceeded. Statistical method is proposed in [8] to control the software defect detection process together with further defect prevention analysis. The paper in [9] aims to improve effectiveness of inspection and testing approaches for achieving 99% defect-free product. It analyses defect patterns and gives concrete ranges for inspection and testing. It introduces 'depth of inspection (DI)' metric to indicate productivity and quality levels for software process. The approach presented emphasized that an optimal level of 10%-15% of inspection time and 15%-25% of testing time out of whole effort with DI value in the range of 0.4 to 0.7 at each phase of software development eliminates 99% defects in small and medium scale industries.

Another paper [10] focused on finding the total number of defects that has occurred in the software development process for five similar projects and aimed at classifying various defects using first level of Orthogonal Defect Classification (ODC), finding root causes of the defects and uses the learning of the projects as preventive ideas. The paper also showcase on how the preventive ideas are implemented in a new set of projects resulting in the reduction of the number of similar defects. One study by Fang Chenbin [11] was introduction of a tool called Bug Tracing System (BTS) for defect tracing, has the advantage of popularity and low cost, and also improves the accuracy of tracking the identified defects.

Stress [12] is given on how analysis of defects found in first iteration can provide feedback for defect prevention in later iterations, leading to quality and productivity improvement. One of the research papers [13] proposed and evaluated a general framework for software defect prediction that supports unbiased and comprehensive comparison between competing prediction systems.

The objective of another study [14] was the identification of factors that influence defect injection and defect detection. They followed an extensive literature search to find influencing factors and processed the factors to achieve consistently formulated sets of factors without duplications, then used a cluster analysis to reduce the number influencing factors to manageable-sized sets for practical application and the last step involved final groupings of factors obtained by expert interpretation of the cluster analysis results.

A paper[15] provided empirical evidence supporting the role of OO design complexity metrics (a subset of the Chidamber and Kemerer suite), in determining software defects and found, even after controlling for the size of the software, these metrics are significantly associated with defects. It was discussed in [16] about the warnings found by FindBugs, a static analysis tool and the kinds of warnings generated ,the classification of warnings into false positives, trivial bugs and serious bugs and provided insight into why static analysis tools often detect true but trivial bugs, and some information about defect warnings across the development lifetime of software release.

An explanation for this ceiling effect was presented in [17]. They used three sub-sampling techniques (under-, over-, and micro-sampling), and looked for the lower useful bound on the number of training instances. They concluded that on having much evidence for the limited information hypothesis. Further progress in learning defect predictors may not come from better algorithms.

In one of the papers [18], the initial concept of the Defect Based Process (DCA) Improvement approach is described. Its main contributions recorded were tailoring support for DCA based process improvement and addressing an identified opportunity for further investigation by integrating organizational learning mechanisms regarding cause-effect relations into the conduct of DCA.

A controlled experiment comparing the defect detection efficiency of exploratory testing (ET) and test case based testing (TCT) is presented in [19]. It was observed that TCT produced significantly more false defect reports than ET and the results showed no benefit of using predesigned test cases in terms of defect detection efficiency, emphasizing the need for further studies of manual testing. Another work [20] presented a defect prediction model based on ensemble of classifiers. They conducted a cost-benefit analysis for our ensemble, where it turns out that it is enough to inspect 32% of the code on the average, for detecting 76% of the defects.

PCA (Process Capability Analysis) involves statistical techniques throughout the product cycle [21]. PCA has become widely adopted asthe measure of performance to evaluate the ability of a process to satisfy customer requirements in terms of specification limits [22].

3. OBJECTIVES OF STUDY

The empirical research starts with comparative analysis of different projects handled by organization in terms of various defect detection and its removal using metrics. The work presented highlights defect detection and removal methods adopted during software development process by comparing 9 different projects using metrics. Further, the research proposes four new metrics intended to judge the capability of the ongoing processes within this organization. Lastly, as the sample size is not large,so for the validity for carrying out statistical procedure is done theough Anderson-Darling GoF test.

4. PROFILE OF ORGANIZATION ‘XYZ’

The case study presented here deals with development some of the software projects by a popular software organization ‘XYZ’ in Noida (India). The organization is involved in the development and testing of software projects. The organization has been lucratively handling the development of projects within cost and schedule frame, producing more contented clients.

4.1. LSL, USL, COQ

The USL or Upper Specification Limit and LSL or Lower Specification Limit are limits set by the customers' requirements. This is the variation that customers' will accept from the process. The UCL or Upper Control Limit and LCL or Lower Control Limit are limits set by process based on the actual amount of variation of the process. The Figure 1 [23] represents the control chart representing this concept.

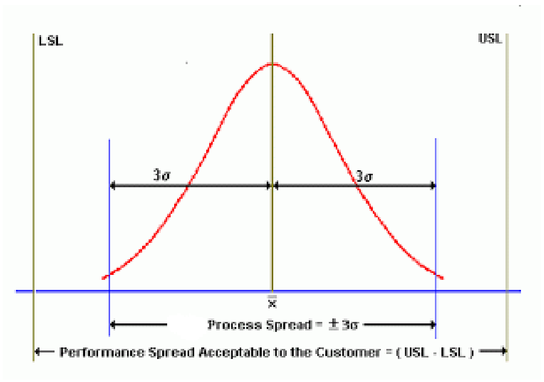
LSL
USL
3σ
3σ
x̄
Process Spread = ± 3σ
Performance Spread Acceptable to the Customer = (USL - LSL)

Figure 1: USL/LSL and UCL/LCL

The table 1 below defines the USL, LSL and Target, collectively refered as organizational baselines, for the 9 projects in terms of DD (Effort), DD (Size), COQ and DRE.

Table1. Organization 'XYZ' baseline

BASELINES	DD (Effort)	DD (Size)	CoQ	DRE
LSL	3.89	5.53	16.30	59.90
USL	6.88	29.31	34.50	80.04
TARGET	5.25	18.15	30.60	70.23

Cost of Quality is a measurement used for assessing the waste or loss from a defined process. These costs are noteworthy and need to be kept reduced or avoided. Cost of quality measurement can track modification or upgradation over time for a particular process, or it can be viewed as a standard of excellence for comparison of two or more different processes. COQ can prioritize quality improvement actions. COQ is shown as in Figure 2 [24].

COQ has two variations; *Cost of poor quality* and *Cost of good quality*. Cost of poor quality is an indication of cost of doing things wrong. Reality is that it is the cost of providing poor quality product or services which may include costs due to internal failure and costs due to external failure. Cost of good quality which includes prevention costs and appraisal costs is actually the cost of achieving good quality. The equation 1 represents calculation for COQ.

$$COQ = \sum \frac{(E+I+A+P)}{S} X\ 100\% \qquad (1)$$

Where: E = External Failure Costs
I = Internal Failure Costs
A= Appraisal Costs
P = Prevention Costs
S = Sales

Figure 2[24] depicts COQ.

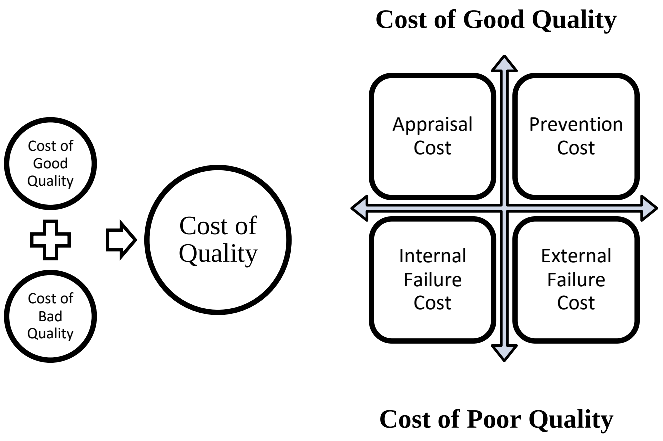

Figure 2. Cost of Quality

4.2. Metrics Used in Organization 'XYZ'

The table 2 presents some metrics followed by 'XYZ' during the development of the different projects.

Table 2. Metrics for 'XYZ'

WP Completed	Project Name	DD (Effort)	DD (Size)	DRE	OTD	CoQ
Yes	Project 1	5.77	9.08	15.15	Yes	17.42
Yes	Project 2	5.03	12.48	16.19	Yes	15.11
Yes	Project 3	5.24	12.96	17.78	Yes	16.06
Yes	Project 4	5.51	15.85	11.91	Yes	12.53
Yes	Project 5	5.27	14.00	0.00	No	18.62
Yes	Project 6	5.33	11.56	17.78	Yes	16.94
Yes	Project 7	5.45	8.33	0.00	No	10.10
Yes	Project 8	5.13	9.52	100.00	Yes	12.60
Yes	Project 9	4.57	4.76	25.00	Yes	18.43
	MEAN (Average)	5.25	10.95	22.64	78%	15.31
	Standard Deviation (σ)	0.34	3.37	30.16		2.97

As from Table 2, total of 9 projects are been provided by the organization. These projects under study are referred as Work Package (WP). The WP have been tested and delivered to the clients. Out of 9 projects, two of these projects (Project 5 and Project 7) do not have On Time Delivery (OTD) to the clients. This is interpreted as these projects failed to be delivered to the clients on the mentioned schedule or date. A summarized description of these metrics is discussed as below.

DD (Size) is one of the metrics that is intended to calculate the ratio between the numbers of defects occurred during development of the software product to the size of the software developed. This metric is useful in telling how the software can be improved by reducing the coding defects. It is more difficult to correct defect in code at design time.It is given by equation 2 as:

$$DD\ (Size) = \frac{Total\ number\ of\ defects\ detected}{Code\ size} \tag{2}$$

Another metric DD (Effort) widely used in the organization measures the ratio between the numbers of defect occurred during software development to the effort involved in the production of desired software. It is given by the equation 3 as:

$$DD\ (Effort) = \frac{Total\ number\ of\ defects\ detected}{effort\ involved\ in\ development} \quad (3)$$

If the value of this metric is more, it can be interpreted that more effort is involved in resolving defects found in software development process.

The various advantages of calculating DD is to compare relative number of defects in different components. Identifying defect prone components allows the concentration of limited resources into areas with the highest potential return on the investment. Another use for Defect Density is to compare subsequent releases of a product to track the impact of defect reduction and quality improvement activities.

Defect Removal Effectiveness is a very important aspect of product quality. A good defect removal process promotes the release of products with lower latent defects, generating high customer confidence. The scope of the defect removal efficiency measurement is by project and verification/validation life-cycle phase.

The Defect Removal Efficiency (DRE) gives a measure of the development team ability to remove defects prior to release. It is calculated as a ratio of defects resolved/removed to total number of defects found. It is typically measured prior and at the moment of release of the product. This is represented by equation 4 as:

$$DRE = \frac{Total\ number\ of\ defects\ corrected\ /\ removed}{Total\ number\ of\ defects\ found} \quad (4)$$

Defect Removal Efficiency (DRE) is one of the commonly used metrics for measuring the efficiency of defect removal at various stages of the software development life cycle. By extrapolating the DRE metric to released products, an organization can measure the effectiveness of their quality assurance process based on the number of defects found in the product before and after its release.

4.3. Comparative Analysis of data

The data collected from organization is analyzed using different comparative methods to find the relation among various metrics. Table 2 reports project data with the Defect Density (DD) recorded for all the nine projects in terms of effort and size of the projects.

The Table 2 shows that Defect Density (Effort) is much lower for all of the nine projects in comparison to DD (Size). The gap between these two DD is observed to be recorded highest for Project 4; while Project 9 shows a relative less variation. The range of DD (Effort) varies in the limit of 4.57 to 5.77; while for DD (Size) is 4.76 to 15.85.

5. PROPOSED METRICS IN PROCESS CAPABILITY ANALYSIS

Process capability examines the variability in process characteristics whether the process is capable of producing products which conforms to specifications capability compares the output of an in-control process to the specification limits by using capability indices. The comparison is done by forming the ratio of the area spread between the process specifications to the spread of the process values, as measured by six process standard deviation units.

The research proposes a metric Process Capability (Cp) index is used to compare the given sample to its specification limits by using capability indices. A process capability index works with both the process variability and the process specifications to determine whether the process is "capable". It is the numerical summary that compares the behavior of product or process characteristics to specifications. It is given in equation 5.

$$\mathrm{Cp} = \frac{(\text{allowable range})}{6\sigma} = \frac{\text{Tolerance(T)}}{6\sigma} = \frac{(\text{USL}-\text{LSL})}{6\text{X standard deviation}} \quad (5)$$

It can be said that Cp can determine how well a process is able to meet specifications. The higher the value of the index, the more capable is the process. The process is centered at the midpoint of the specification limits if Cp=Cpk. The process if off-centered and can be accepted as lower capability than the case that the process is centered if Cpk<Cp. The reason is that it is not operating at the midpoint of the interval between the specification limits. If Cpk=0, the process mean is exactly equal to one of the specification limits. If Cpk<0, the process mean lies outside the specification limits, that is for μ>USL or μ<LSL, Cpk<0. If Cpk<-1, the entire process lies outside the specification limits. It should be noted that some authors define Cpk to be non-negative so that values less than zero are defined as zero [21]. 1<Cpk<1.33 means that the process is barely capable. If Cp>1.50, it shows the variables are critical. Automotive industry uses Cpk=1.33 as a benchmark in accessing the capability of a process [25]. The data collected from the past projects of organization XYZ is presented below in Table 3.

Table 3. Cp metrics

Organization Baselines		DD (Effort)	Cp Metric for DD (Effort)	DD (Size)	Cp Metric for DD (Size)	COQ	Cp Metric for COQ	DRE	Cp Metric for DRE
LSL		3.89		5.53		16.30		59.90	
USL		6.88	1.48	29.31	1.17	34.50	1.02	80.04	0.11
TARGET		5.25		18.15		30.60		70.23	
PROCESS PERFORMANCE INDEX METRICS	Cpk		0.154		1.81		0		0
	Cpl		0.154		6.08		0		0
	CPU		1.59		1.81		2.15		0.63

Another metric proposed is *Process Performance Index (PPI).* The capability index considers only the spread of the characteristic in relation to specification limits assumes two-sided specification limits. The product can be unworthy if the mean is not set appropriately. The Process Performance Index takes account of the mean (μ) and is defined as in equation 8:

$$Cpk = \min [(USL - \mu)/3\sigma, (\mu - LSL)/3\sigma] \quad (8)$$

The process performance index can also accommodate one sided specification limits. For upper specification limit, it is given by equation 9.

$$Cpk = (USL - \mu)/3\sigma \quad (9)$$

For lower specification limit, it is given by equation 10.

$$Cpk = (\mu - LSL)/ 3\sigma \quad (10)$$

The research further proposes two more metrics for further validation of ongoing software processes; CPU and Cpl. CPU measures the capability of the upper half of the process; Cpl measures the capability of the lower half of the process. The CPU metric is given by equation 11.

$$\text{CPU} = \frac{(USL - Mean)}{3\ X\ standard\ deviation} \quad (11)$$

The Cpl metric is given by equation 12.

$$\text{Cpl} = \frac{(Mean - LSL)}{3\ X\ standard\ deviation} \quad (12)$$

Cpk, CPU, and CPL indicate the ideal performance of your process if special causes were eliminated. If these potential indices are compared to benchmarks in the field, it can be helpful to determine whether to improve the process or not.

Next, the research drives towards fitting of the data using Anderson Darling (AD) Goodness of Fit (GoF) test.

In most of the statistical methods, it is assumed that there exists an underlying distribution in the derivation of results. But, there are chances of risks if our data follow a specific distribution. If the assumptions are made wrong, then results obtained may be invalid and of no use.

According to [26, 27, 28], there are two main approaches for checking distribution assumptions. One is empirical based approach, which is easy to understand and implement. These are intuitive based graphical properties. Another approach is GoF (Goodness of Fit) test. These tests are more formal, statistical procedures for assessing underlying distribution of data. The results are more quantifiable and reliable than those of empirical based approach. Anderson-Darling (A-D) and Kolmogorov-Smirnov (K-S) tests are two most commonly used tests. The correct implementations of statistical procedures require establishing underlying distribution of data-set. Firstly, it requires whether normal applies before we can implement statistical procedures.

In GoF, two distribution elements are basically considered; Cumulative Distribution Function (cdf) and Probability Distribution Function (pdf). AD and KS uses cdf and belongs to class of "distance tests". AD test is selected because of two features. Firstly, it is one of the best distance tests that can be applied to both small samples and large samples of data. Also, various statistical packages are available for applying AD and KS tests.

To implement distance tests, firstly it is assumed that there is normal (pre-specified) distribution. Next, distribution parameters; mean and variance are estimated from the data. This process builds distribution hypothesis, known as null hypothesis (H_0). Then the assumed or hypothesized distribution is tested using data set. Finally, H_0 is rejected, if any of the elements composing H_0 is not supported by data.
The A-D test has the form as in equation –

$$\mathrm{AD} = \sum_{i=1}^{n} \frac{1-2i}{n} \{\ln(F_0[z_i]) + \ln(1-F_0[z_{(n+1-i)}])\} - n; \ldots \qquad (13)$$

Where F_0 is assumed distribution with sample estimated parameters (μ,σ);
Z_i is the standardized sorted ith sample value
n is the sample size
ln is the natural logarithm taking values from 1 to n.
At significance level of α =0.05, the null hypothesis is rejected if AD statistic is greater than the AD*. This is given by equation 14

$$\mathrm{AD}^* = \mathrm{AD} * (1+(\frac{0.75}{n}) + (\frac{2.25}{n^2})) \qquad (14)$$

The steps followed in the AD GoF test for Normality is summarized as:

$$Z = \left(\frac{x-\mu}{\sigma}\right) \qquad (15)$$

1. Sort original X sample and standardized
2. Establish the null hypothesis
3. Obtain the distribution parameters μ,σ
4. Obtain the F(Z) cumulative probability
5. Obtain logarithm of the above.
6. Sort cumulative probability in descending order
7. Find values of 1- F(Z)
8. Find logarithm of the result of above step
9. Evaluate test statistics i.e. AD and AD*
10. Reject null hypothesis if AD* < AD

The research next tests AD GoF test for sample of Defect Density (Effort) by following the above steps. Table 4 shows the descriptive statistic for DD(Effort).

Table 4. Descriptive Statistic of Probable Defect Density (Effort) Data for AD GoF test

Variable	N	Mean (μ)	Median(σ)
Data Set	9	5.25	5.27

For the smallest element, we have the normal probability of smallest variable within estimated μ and σ as in equation 16:

$$P_{\mu=5.25,\sigma=0.34}^{(4.57)} = \text{Normal}\,(\frac{4.57\text{-}5.25}{0.34}) = F_0(Z) = F_0(-2) = 0.022 \qquad (16)$$

In this case, AD value obtained is 0.253, whereas AD* calculated from equation 14 is 0.677. Table 5 shows intermediate values for AD GoF test for Normality.

Table 5. Intermediate values for AD GoF test for Normality with Defect Density (Effort) Dataset

DD (Effort)	i	Sorted	F(Xi)	1-F(Xi)	1-F(Xn-i+1)
5.77	1	4.57	0.021033	0.978967	0.06357028
5.03	2	5.03	0.2518	0.7482	0.22527429
5.24	3	5.13	0.354832	0.645168	0.2821104
5.51	4	5.24	0.481605	0.518395	0.41264388
5.27	5	5.27	0.517082	0.482918	0.48291776
5.33	6	5.33	0.587356	0.412644	0.51839536
5.45	7	5.45	0.71789	0.28211	0.64516832
5.13	8	5.51	0.774726	0.225274	0.74820025
4.57	9	5.77	0.93643	0.06357	0.97896712

Similarly, for Defect Density Table 6 and 7 shows the AD statistic and intermediate values for AD GoF test for Normality with Defect Density (Size) Dataset.

Table 6. Descriptive Statistic of Probable Defect Density (Size) Data for AD GoF test

Variable	N	Mean (μ)	Median(σ)
Data Set	9	10.95	11.56

Table 7. Intermediate values for AD GoF test for Normality with Defect Density (Size) Dataset

DD (Size)	i	Sorted	F(Xi)	1-F(Xi)	1-F(Xn-i+1)
9.08	1	4.76	0.033094	0.966906	0.072852
12.48	2	8.33	0.21846	0.78154	0.182545
12.96	3	9.08	0.289525	0.710475	0.275257
15.85	4	9.52	0.335725	0.664275	0.324733
14	5	11.56	0.571975	0.428025	0.428025
11.56	6	12.48	0.675267	0.324733	0.664275
8.33	7	12.96	0.724743	0.275257	0.710475
9.52	8	14	0.817455	0.182545	0.78154
4.76	9	15.85	0.927148	0.072852	0.966906

AD value obtained is 0.172 and AD* is 0.191. Table 8 and 10 shows statistic data for DRE and COQ and similar test is conducted for DRE and COQ and the results are given as in Table 9 and 11.

Table 8. Descriptive Statistic of Probable DRE Data for AD GoF test

Variable	N	Mean (μ)	Median(σ)
Data Set	9	15.31	16.05

Table 9. Intermediate values for AD GoF test for Normality with DRE Dataset

DD (Size)	i	Sorted	F(Xi)	1-F(Xi)	1-F(Xn-i+1)
15.152	1	0	0.226356	0.773644	0.005157
16.19	2	0	0.226356	0.773644	0.468875
17.778	3	11.905	0.360871	0.639129	0.564104
11.905	4	15.152	0.401889	0.598111	0.564104
0	5	16.19	0.415258	0.584742	0.584742
17.778	6	17.778	0.435896	0.564104	0.598111
0	7	17.778	0.435896	0.564104	0.639129
100	8	25	0.531125	0.468875	0.773644
25	9	100	0.994843	0.005157	0.773644

Table 10. Descriptive Statistic of Probable COQ Data for AD GoF test

Variable	N	Mean (μ)	Median(σ)
Data Set	9	22.64	16.19

Table 11: Intermediate values for AD GoF test for Normality with COQ Dataset

DD (Size)	i	Sorted	F(Xi)	1-F(Xi)	1-F(Xn-i+1)
17.423	1	10.1	0.039678	0.960322	0.132677
15.113	2	12.533	0.174751	0.825249	0.147042
16.055	3	12.599	0.180531	0.819469	0.238766
12.533	4	15.113	0.47321	0.52679	0.29206
18.622	5	16.055	0.598648	0.401352	0.401352
16.939	6	16.939	0.70794	0.29206	0.52679
10.1	7	17.423	0.761234	0.238766	0.819469
12.599	8	18.43	0.852958	0.147042	0.825249
18.43	9	18.622	0.867323	0.132677	0.960322

The AD and AD* for DRE are 1.418 and 1.576; and for COQ, it is 0.333 and 0.370 respectively. The normal probability curve for the above results is shown below as in Figure 3- 6.

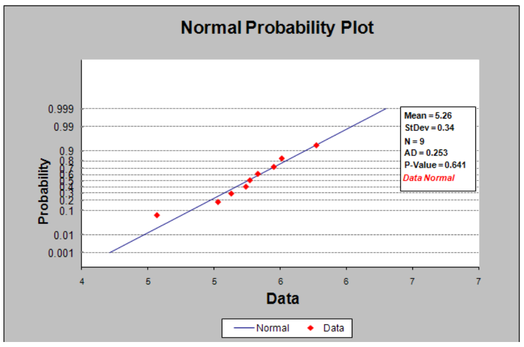

Figure 3. Normal Probability Plot for AD GoF test for DD (Effort)

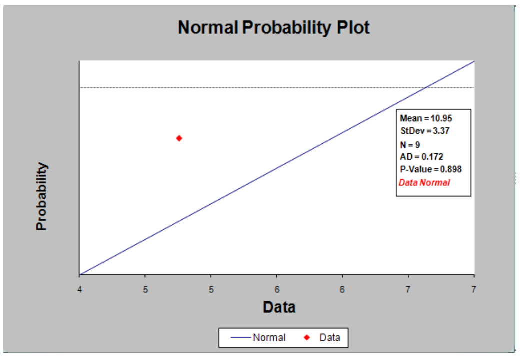

Figure 4. Normal Probability Plot for AD GoF test for DD (Size)

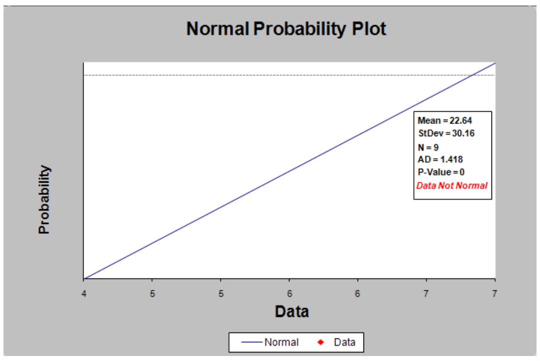

Figure 5. Normal Probability Plot for AD GoF test for DRE

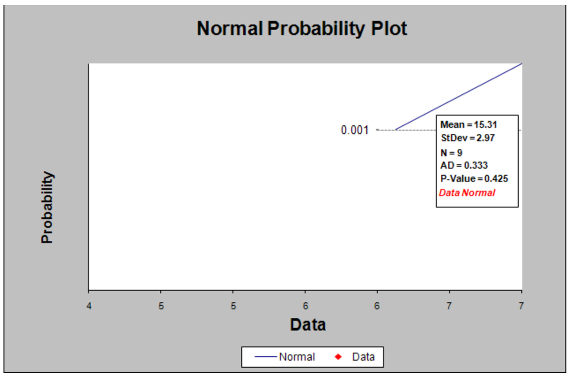

Figure 6: Normal Probability Plot for AD GoF test for COQ

6. RESULTS AND ANALYSIS

Results from Table 5 shows that for DD (Effort), Cp metric has a value of 1.48, which can be interpreted to be satisfactory for existing process. The value of Cp metric for DD (Size) is 1.17 interpreted to be adequate process. The Cp metric values for COQ and DRE are obtained as 1.02 and 0.11, which is interpreted as adequate and unsatisfactorily process respectively. The result of Cp metric for DRE process can be improved by first reducing the variation in the process and then shifting the mean of the process towards the target.

In the case of GoF test for DD (Effort), AD yields a value 0.253, which is less than AD* value of 0.677. The p-value observed is 0.641. Therefore, AD GoF test doesnot reject the null hypothesis that these samples have been drawn from a Normal (5.25, 0.34) population and we can assume normality of data. For DD (Size), the AD statistics obtained is 0.172 and AD* is 0.19. Here also, AD<AD* and observed p-value is 0.897, meaning null hypothesis is accepted. AD statistics in the case of DRE is is 1.418 and AD* is 1.576, whereas p-value is 0. This condition rejects the null hypothesis as p-value is less than 0.05. For GoF test on COQ, the value of AD is 0.333 and AD* is 0.370. The p-value is 0.425, which is more than acceptable value of 0.05.

7. CONCLUSION

The proposed metrics can be beneficial in determining the effectiveness of software development process. When the capability index of the process is known, the areas that need to improve can be focused. This will be aiding in the process of defect detection and removal; thereby advancing the removal efficiency and quality of the software to be produced. This will further add to more satisfied customers.

The AD GoF Normality test is performed to determine the underlying distribution of data. Various parameters are estimated and checked to know about correct implementation of statistical procedures.

8. LIMITATION OF STUDY AND FUTURE SCOPE

The study is limited to only 9 projects provided by the organization due to confidential issues. The study becomes confined due to limitation of data provided. Also, the study is restricted to only one software organization, so a comparative analysis is not possible to know about the trends in defect detection and removal.

The future scope of this research lies in the study of defect detection and removal during various phases of software development. Also, it will enrich by proposing new metrics that will be used for easy detection of the defects that prevails in the software during development in its lifecycle.

REFERENCES

[1] Lessmann, Stefan, et al. "Benchmarking classification models for software defect prediction: A proposed framework and novel findings.", *IEEE Transactions on Software Engineering*, Volume/issue: 34(4). pp. 485-496, 2008

[2] Barry Boehm and Victor R. Basili. 2001. Software Defect Reduction Top 10 List, *Computer*, Volume/issue: 34(1).pp.135-137, 2008

[3] Runeson, Per, et al. "What do we know about defect detection methods? [Software testing]." *IEEE Software*, Volume/issue: 23(3), pp.82-90, 2006

[4] Suma V and T R Gopalakrishnan Nair , " Effective Defect Prevention Approach in Software Process for Achieving Better Quality Levels", *Proceedings of World Academy of Science, Engineering and Technology*, Volume 32, 2008

[5] Suma, V., and TR Gopalakrishnan Nair. "Enhanced approaches in defect detection and prevention strategies in small and medium scale industries." IEEE The Third International Conference on. Software Engineering Advances,pp. 389 – 393, 2008

[6] Van Moll, J. H., et al. "The importance of life cycle modeling to defect detection and prevention." *Proceedings. Of 10th International Workshop on. IEEE Software Technology and Engineering Practice,* pp. 144-155, 2002.

[7] Shull, Forrest, et al. "What we have learned about fighting defects." *IEEE Proceedings of Eighth Symposium on Software Metrics*, pp. 249,2002

[8] Hong, G. Y., M. Xie, and P. Shanmugan. "A statistical method for controlling software defect detection process." *Computers & industrial engineering*, Volume/issue: 37(1), pp.137-140, 1999

[9] Suma, V., and T. R. Gopalakrishnan Nair. "Better defect detection and prevention through improved inspection and testing approach in small and medium scale software industry." *International Journal of Productivity and Quality Management*, Volume/issue: 6(1), pp. 71-90, 2010

[10] Kumaresh, Sakthi, and R. Baskaran. "Defect analysis and prevention for software process quality improvement." *International Journal of Computer Applications,* Volume/issue: 8(7), 2010

[11] Pan Tiejun, Zheng Leina, Fang Chengbin, "Defect Tracing System Based on Orthogonal Defect Classification" *Computer Engineering and Applications*, Volume 43, pp. 9-10, 2008

[12] Pankaj Jalote, Naresh Agarwal, "Using Defect Analysis Feedback for Improving Quality and Productivity in Iterative Software Development" *In proc- IEEE ITI 3rd International Conference on Information and Communications Technology*, pp. 703-713, 2005

[13] Song, Qinbao, et al. "A general software defect-proneness prediction framework." *IEEE Transactions on Software Engineering,* Volume/issue: 37(3) , pp. 356-370, 2011

[14] Jacobs, Jef, et al. "Identification of factors that influence defect injection and detection in development of software intensive products." *Information and Software Technology*, Volume/issue: 49(7),pp. 774-789, 2007

[15] Subramanyam, Ramanath, and Mayuram S. Krishnan. "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects." *IEEE Transactions on Software Engineering,* Volume/issue: 29(4), pp. 297-310, 2003

[16] Ayewah, Nathaniel, et al. "Evaluating static analysis defect warnings on production software." *7th ACM Proceedings of the SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering.*, pp. 1-8, 2007

[17] Menzies, Tim, et al. "Implications of ceiling effects in defect predictors." *ACM Proceedings of the 4th international workshop on Predictor models in software engineering*., 2008

[18] Kalinowski, Marcos, Guilherme H. Travassos, and David N. Card. "Towards a defect prevention based process improvement approach." *IEEE 34th Euromicro Conference Software Engineering and Advanced Applications*, 2008

[19] Itkonen, Juha, Mika V. Mantyla, and Casper Lassenius. "Defect detection efficiency: Test case based vs. exploratory testing." *IEEE First International Symposium on Empirical Software Engineering and Measurement*, 2007

[20] Tosun, Ayse, Burak Turhan, and Ayse Bener. "Ensemble of software defect predictors: a case study." *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement.*, 2008.

[21] Montgomery, D. C, *Statistical Quality Control- A Modern Introduction*, Wiley., ISBN: 978047233979, USA, 2009

[22] English, J. R. & Taylor G. D. Process capability analysis- a robustness study. International Journal of Production Research, Volume/issue 31(7), pp.1621-1635, 1993

[23] http://lecturehub.ie/2013/10/24/the-difference-between-usllsl-and-ucllcl/

[24] www.hcltech.c om/sites/default/files/Defect_Prevention_Whitepaper.pdf

[25] Automotive Industry Action Group (AIAG) , *Measurement system analysis*, (3rd ed.) Southfield, 2002

[26] Jorge Luis Romeu, Christian E. Grethlein, "*A Practical Guide to Statistical Analysis of Material Property Data*", AMPTIAC, 2000

[27] V. K. Rohatgi, "*An introduction to probability theory and mathematical statistics*", Wiley NY, 1976

[28] Nancy R. Mann, Ray E. Schafer, Nozer D. Singpurwalla, "*Methods for statistical analysis of reliability and life data",* Wiley, 1974

BIOGRAPHIES OF AUTHORS

Pooja Jha (F'2011) has done her B.Sc (CS) in 2002 and MSc (IT) in 2004 from BIT, Mesra, and Ranchi. She is pursuing her PhD in Computer Science & Engineering from BIT, Mesra. Her area of research is Software Metrics and Software Engineering.
She is having work experience of 5 years and research of 3 years. Currently, she is working as an Assistant Professor in department of Computer Science at Jharkhand Rai University, Ranchi, India.

K S Patnaik is working as Associate Professor in the department of Computer Science and Engineering, BIT, Mesra, Ranchi. He has done BE (EE) from NIT, Silchar, Assam (India) and ME (SE), BIT, Mesra, Ranchi, Jharkhand (India). He obtained his PhD in Engineering from BIT, Mesra, Ranchi, Jharkhand (India).
He has a work and research experience of 11 years. His area of interest includes Intelligent System Design, Software Engg, Soft Computing and Image Analysis. He has a number of publications in national and international journals. He is member of professional bodies like ISTE, IE (I), ACM, CI Lab-Univ. of Manitoba, Winnipeg, Canada.