Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 54 (2015) 335 – 342

Procedia Computer Science

Eleventh International Multi-Conference on Information Processing-2015 (IMCIP-2015)

An Impact of Linear Regression Models for Improving the Software Quality with Estimated Cost

Arun Kumar Marandi* and Danish Ali Khan

Department of Computer Applications, National Institute of Technology, Jamshedpur 831 014, India

Abstract

Developing organizations are spends lots of money to finding the Errors and bugs. In this article, an application of defects removal effectiveness to improve the software quality and fault prone analysis, methods are finding the solution of parameters in linear regression models with cost estimating method. It describes the approach of quantitative quality management through defect removal effectiveness and statistical process control of cost analysis with historical project data. Software quality is going continuously monetary benefit to perform well management planning, and achieve a new height. In this methodology, Software quality model can make timely predictions of reliability indications; it's enabling to improve software development processes by target reducing the estimated cost for software products and improve the techniques for more effectively and efficiently.

Peer-review under responsibility of organizing committee of the Eleventh International Multi-Conference on Information Processing-2015 (IMCIP-2015)

Keywords: Cost estimation; Defect injection; Defect removal; Linear regression model; Quality management.

1. Introduction

Improved software tools could decrease the testing resources needed to achieve a high level of quality software products. In this approach effect, an improved infrastructure would make developing resources more effectively and it is Marginal cost of testing prior to shipping down to the right closer to the asymptote of high quality software products. Software product is to achieve an appropriate level of software quality. To develop, Developers and designers are trying to produce readable, reliable, maintainable, reusable testable code[1]. Software develop effort estimation is the process of predicting the most realistic effort required to develop software based on available information. However, the Software development lifecycle (SDLC) phases are applied to predict effort for singleton life cycle stages. Defect removal effectiveness is a direct indicator of the capability of a software development process in removing defects before the software is delivered. It is one of few, perhaps the only, process indicators that bear a direct correlation with the quality of the software's field performance[2,3]. The paper deals several aspects of defect removal effectiveness including overall effectiveness, inspection effectiveness, test effectiveness, phase specific effectiveness and the role of defect removal effectiveness under quality planning[4,5]. High quality software attributes to a defect – free product, which is competent of producing predictable results and remains deliverable within time and cost constraints[3,6]. It should be manageable with minimum interferences. It should also be maintainable, dependable, understandable, and efficient. To increased

*Corresponding author. Tel.: +91-9931317668.
E-mail address: arunmarandi.ca@nitjsr.ac.in

Peer-review under responsibility of organizing committee of the Eleventh International Multi-Conference on Information Processing-2015 (IMCIP-2015)
doi:10.1016/j.procs.2015.06.039

competitiveness in today's business world Thus, a systematic approach towards high quality software development is required due technological advances hardware complexity and frequently changing business requirements[4,5,7]. The defect framework is based on analyzing the defects that had emerged from various stages of software development like Requirements, Design, Coding, Testing and Timeline (defects due to lack of time during development). This study is not limited to just identifying the origin of defects at various phases of software development but also finds out the reasons for such defects, and defect preventive measures are proposed for each type of defects. Defect injection metric based on severity of the defect rather than just defect count, which gives the number of adjusted defects produced by a project at various phases[8]. The defect injection metric value, once calculated, serves as a yardstick to make a comparison in the improvements made in the software process development between similar set of projects[9]. Software can be considered a product whose production is fundamentally similar to other products.

2. Literature Survey

There are several studies conducted by different researchers for producing high quality software and reducing the estimated cost, A. Schiffaucrova and V. Thomson (2006) propose quality cost models, in this model practical use of cost of quality suggests that even though quality is consider an important issue. Dale and Wan (2002), it focuses on quality costing method policies for industrial level. Most of the researcher only focuses on finding the defects on software developing process for these study Fang Chengbin (2008) introduced a tool called bug tracing system (BTS), for defect tracing, has the advantage of popularity and low cost, and also improves the accurate tracking and identify the defects where it is located on software developing lifecycle (SDLC) phases. Stefan Wagner (2008) finding the defects and summarizes the work on defects classifications approaches that have been proposed by two companies IBM and HP. The IBM approach is called Orthogonal Defect Classification (ODC) and the HP approach is based on three dimensions – Defect Origin, Types and models. Pankaj Jalote and Naresh Agarwal (2007) stressed on how analysis of defects found in first iteration, letter on how to retrieve the feedback for defect prevention to next iterations, leading to quality and productivity improvement[10]. To Improving, the quality of software can be approaches using the same basic principles support by quality leads to W. Edwards Deming, Philip B, Crosby, and Harold F. Dodge[8]. Show that, it will be possible to predict the potential cost savings and defect reduction expected. The quality of software is heavily influence by proper attention to every phase of development[5]. However, high quality software should have as few defects as possible. It is accepts that defects will be injected and the objective is to deliver software with few defects within the estimated budget.

The rest of the paper is as follows: section 3 describes about the proposed framework then after discuss about the different models after that the result and discussion in section follows by the conclusion and future scope.

3. Proposed Framework

Our paper aims to understand the impact of economic effort and review on the defect removal effectiveness of the software products and statistical analysis to enhancing the software quality and reducing cost. The paper first presents a simplified Economical model of software quality, and develops the details of the model at the level of individual defects removal at each level of software developing phase. The main objective of the model to identify approaches to improve the software quality and reduced the marginal cost of software products.

3.1 Conceptual economic model

In this section paper, describe about the proposed model. The cost of an inadequate infrastructure for software testing can also be expressed as the benefit of an improved infrastructure for software testing. These values (cost and benefit) are symmetrical[11]. They are properly measured as either the minimum amount of money all members of society would collectively require to forego the improved infrastructure or as the maximum amount of money all members of society would collectively pay for the improved infrastructure[12]. An appropriate measure of the economic impact of an inadequate infrastructure for software testing is the profit differences of developers and users between conditions with the current testing infrastructure and conditions with the counterfactual infrastructure. This can be expressed by

summing over all developers and users as follows

$$\Delta \text{ Economic} = \Sigma\,\Delta \text{ developer's profits} + \Sigma\,\Delta \text{ User's profits.}$$

The appropriate measure of the value developers would place on an improved infrastructure for software testing is their profit difference between conditions with the current testing infrastructure and conditions with the counterfactual infrastructure. Profits are firm revenues minus costs. Suppose the firm produces a single software product (q) at a price (p) total revenues are $TR = pq$.

The profit, π, the developer receives over the entire product life cycle is

$$\pi = pq - \left[\sum_{i=1}^{n} w1i.x1i\left(\sum_{i=1}^{r} w2ix2i + \sum_{i=1}^{s} w3ix3i\right)q\right] \tag{1}$$

where the first term is the revenues, the second is cost.

With improvement in testing infrastructure, resource use in the developing phase ($x_{1i} \ldots x_{1n}$) will change. Fewer bugs will be embodied in shipped products, thus resource use for after – sales service ($x_{31} \ldots x_{3s}$) will also change. With improvements in product quality demand may increase, increasing sales of the software products (q) and thereby changing the resource use in software production and distribution ($x_1 \ldots x_r$). In this scenario developer are producing a better product and production – estimated cost will change, product prices (p) will also change. Software quality not only affects price (p), but also the resources per unit sold needed, for after – sales service. At the appropriate measure of the value end users would place on an improved infrastructure for software testing is their profit difference between conditions with the current testing infrastructure and conditions with the counterfactual infrastructure[12]. End user's profits are modelled as a function of the difference in revenues and production costs. End user's total revenues are expressed as the price time's quality for the product the firm produces: $TR = py$

Let the end user expend n inputs or resources ($x_{1i} \ldots x_{1n}$) prior to purchasing software and that the prices for the resources are ($w_{1i} \ldots w_{1n}$). These costs may include, search costs or delay costs from uncertainty over the quality of available software[12]. The end – user's profit (π) can be expressed as its product life – cycle revenue minus its costs.

$$\pi = py - \left[\sum_{i=1}^{n} w1ix1i\left(\sum_{i=1}^{r} w2ix2i + \sum_{i=1}^{s} w3ix3i + \sum_{i=1}^{v} w4ix4i\right)y\right] \tag{2}$$

Thus, the benefit of an improved software testing infrastructure to an end user is the change in profit. Software testing leads to reductions in economic cost as reflected in the combined profits of developers and end users. The magnitude and distribution of impacts between developers and end users depends on the underlying relationships in the developing quality function of software quality.

4. Linear Regression Models

The paper deals about the estimation of the parameters in linear regression models; the method of least squares is typically used to estimate the regression coefficients in multiple linear regression models[13]. Let $n > k$ observations on the response variable are available, $y_1, y_2, \ldots, y_n$. Along with each observed response y_i, it will be an observation on each regresses variable and let x_{ij} denote the i^{th} observation or level of variable x_j. The data will appear as in Table 1. It assume that the error term ε in the model has $E(\varepsilon) = 0$ and $V(\varepsilon) = \sigma^2$ and that the $\{\varepsilon_i\}$ are uncorrelated random variables. The model equation

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_k x_k + \varepsilon_i \tag{3}$$

The method of least squares chooses the βs in equation (3) so that the sum of the squares of the errors ε_i is minimized. The least squares function is

$$DRE_L = \sum_{i=1}^{n} \varepsilon i^2 = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij}\right) \tag{4}$$

Table 1. Data for multiple linear regressions.

y	x_1	x_2	...	x_k
y_1	x_{11}	x_{21}	...	x_{k1}
y_2	x_{12}	x_{22}	...	x_{k2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
y_n	x_{1n}	x_{2n}	...	x_{kn}

The function DRE_L is to be minimized with respect to $\beta_0, \beta_1 \ldots \beta_k$. The least squares estimators, say $\beta_0, \beta_1, \ldots \beta_k$, must satisfy

$$\frac{\partial DRE_L}{\partial \beta_0}_{\beta_0 \ldots \beta_K} = -2 \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij} \right) x_{ij} \tag{5}$$

where $j = 1, 2, \ldots, k$.

These equations are called the *DRE* least squares normal equations. The solution to be normal equations will be the *DRE* least squares estimators of the regression coefficients. It is simpler to solve the normal.

Equations if they are expresses in matrix notation.

5. Iterative Phase Defect Removal Model

The model are divided into three basic stages are defect Injection, defect Detection and defect Removal during software development life cycle. The model gets the Defect Removal Efficiency[4] and competitively analysis[13], and reducing cost and time effectively in Iterative Phase Defect Removal Model. An empirical study conducted across several project from various service – based and product – based organizations reveals that requirement phase contain 50% to 60% of total defects. 15% to 30% of defects are at design phase. Implementation phase contains 10% to 20% of defects[9,12,13]. Remaining are miscellaneous defects that occurs because of bad fixes. Bad fixes are injection of secondary defects due to bad repair of defects. The common cause for defect occurrences at requirements phase are requirement incompleteness, inconsistency, ambiguity, requirement change and requirement presentation. The common reasons for defect occurrences at design phase are non-conformance to external specification, internal specifications, logical specifications, interface specifications, component specification, security, organizational policies and standards in addition to non-conformance to design with requirement specification[9,14]. The common sources for defect occurrences at implementation phase are improper error handling, improper algorithm, programming language shortcomings and wrong data access and novice developers[3,15]. The phase wise defect injection analysis with cost, the first stage of SDLC cost should be low but the defect injection little bit higher at the cost level. Show that initially defects are injected into little bit higher, another Fig. 1(b) shows that initial percentage defects are higher in the both Fig. 1(a) and (b) analyze defects injected during SDLC process[16]. Defects are mostly introducing at requirements, design, code and testing phases. In each phases the defect location are critically unidentified. Second stage is Defect or error injection during the developing process[17]. In this stage, identify the root cause of defect injection at different levels of software development cycle. There are some Metrics to be used and implement to enhancing the quality of the products.

Third stage is Defect detection, in this stage detected the defects and counts the error percentage. Fourth stage is Defect Removal Process, in this stage the model verify and analysis the defect or unidentified defect (Bad fixing) go to re-refine the third stage. If the entire defects are reduced it redirect to the next SDLC phases and iteratively[15]. The paper deals a simple model of the defect removal process to develop formulas for enhancing the quality and reducing the work effort. Phase – wise defect injection analysis with cost it clear that low level phase has less amount spent during the software development life cycle[17]. The paper deals detection method Defects are introduced throughout the software development life cycle and the art of testing is to find as many of them as possible when they are inserted. It is widely recognized that there is a parabolic curve of defect insertion. The starting point is the requirement specification, which begins by inserting 60% of the defects[16]. The curve terminates with live or production, where the intended result is to find zero defects. The tester should report on completion of the project, the defect detection efficiency. This looks

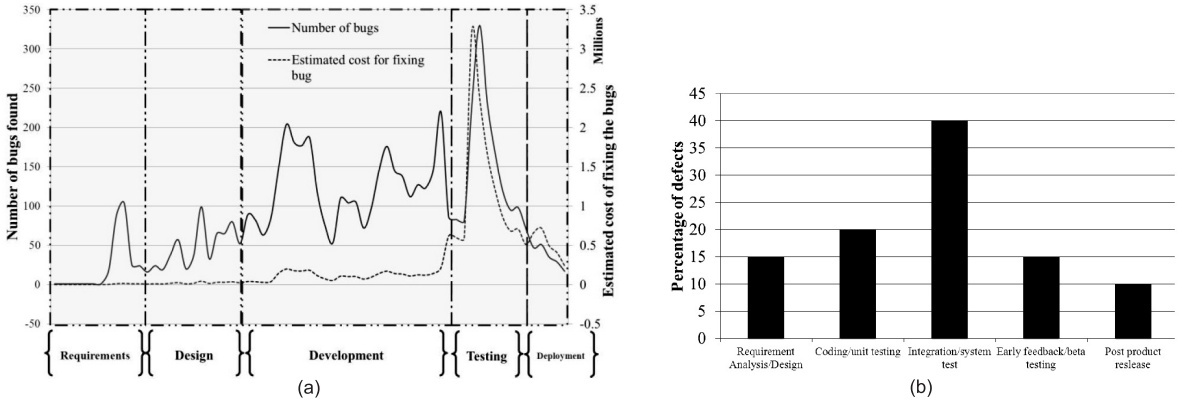

Fig. 1. (a) Phase – wise dDefect injection analysis with cost; (b) Phase – wise defect injection percentage.

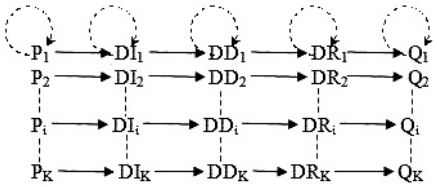

Fig. 2. Iterative phase by defect detection and removal model.

at understanding for each defect, where it was inserted and where it was detected. A perfect test process would look to identify each defect, as it is inserting. This is highly unlikely and the reality is that some defects are finding in later phases of testing. It is therefore important that testing is involved in the project from the outset, not as something that is included if there is the time, the budget and the inclination.

Notation

P_k = SDLC Phase to step k.
DI_K = the number of defects injected in the phases on entry to step k.
DD_K = the number of defects detection during the phase development to step k.
DR_K = the count of defect removed from each step k.
Q_K = Number of defect in the released software.

The defect removal efficiency for each software development life cycle, relative to the defects present is the fraction found 'P_k' times the fraction of good fixes $(1-\varepsilon_k)$. Hence the defect removal efficiency (DRE_{ck}) relative to the defects currently into the each step k is.

$$DRE_{ck} = (1 - \varepsilon_k) \cdot p_k$$

Let us assume that ε and p are constants:

$$DRE_c = (1 - \varepsilon) \cdot p \tag{6}$$

The fraction of defects removed relative to the total lifetime defects (DRE) is:

$$DRE_c = (1 - \varepsilon) \cdot p \tag{6'}$$

The fraction of defects removed in the review/inspection for each step k [1].

$$DRE_1 = MP/TD \tag{7}$$

where Phase 1 effectiveness (E_1) and Phase 2 effectiveness (E_2) [1],

$$E_1 = E_2 \tag{8}$$

(from Remus and Zilles mathematical relationships of defect removal effectiveness).

From the equation (7) and equation (8), the result can be obtained directly.

$$DRE1 = \frac{p1.OD}{\left(\frac{1}{1-\varepsilon 1}\right) \cdot OD} \tag{9}$$

$(1 - \varepsilon_k) \cdot P_k$: (Let '$OD$' is the original defects introduced into the SDLC Phases.)

$$TD = MP + (E_2 x (TD - MP))$$
$$Q_k = TD(1 - DRE_{ck})^n$$

or

$$Q_2 = TD(1 - DRE_{ck})^2 \tag{10}$$

where,

$$\begin{aligned} Q &= TD(1-E)(1-E2) \\ &= TD(1-E)^2 \\ &= (TD - MP)E2 \times (TD - MP) \times 1/MP \end{aligned} \tag{11}$$

from equation (10) and equation (11),

$$Q_2 = \frac{TD}{\mu^2} \tag{12}$$

Note that Q_2 is inversely proportional to μ^2. The quality improves as the value of Q is decreases and the value of μ is increases.

6. Result and Discussion

The models relates the discussed economic factors and other technical factors with the aim to statistical analysis the defects factors at each level. The defect – detection techniques used to plan the quality assurance in a development project. Later on, it used the models as a basis for reviewing the empirical literature and hence describes only briefly the assumptions and equations[19]. A simplified version of this models is available that can be used to plan the quality assurance of a developing project using historical data. It summaries the empirical knowledge available for the quality of defect – detection techniques introducing the approach in general and then describing the relevant studies and results for each of the models factors for different types of techniques and defects in general[20]. The field of quality assurance and defect – detection techniques in particular has been subject to a number of empirical studies over the last decades.

These studies were used to assess specific techniques or to validate certain law and theories about defect – detection. The first category of defect – detection techniques its look at is also the most important one in terms of practical usage. Dynamic testing is a technique that executes software with the aim to find failures[13]. The second categories of defect – detection techniques under consideration are review and inspections, i.e., document reading with the aim to improve them. In most cases review is used interchangeably. It can be identify differences mainly in the process of the inspections[19]. In this techniques it start with analyzing the effectiveness of inspections and reviews that is later used in the approximation of the difficulty[12,14]. It observes a stable mean value that it is close to 30%. However, the range of values is huge. This suggests that an inspection is dependent on other factors to be effective. The efficiency relates

Table 2. Historical project data set.

Projects/ modules	Projects of requirements	Requirement review defects	Design review defects	Code review defects	DIT defects	SIT defects	Post release defects
p2	21	12	10	18	13	48	1
p3	28	25	11	24	19	70	1
p4	24	8	5	22	19	43	2
p5	19	15	5	17	10	30	1
p6	29	6	7	23	22	58	3
p7	17	6	3	15	8	28	1
p8	27	9	2	26	12	41	1
p9	23	10	5	23	8	52	1
p10	19	7	1	19	14	38	2
p11	13	11	3	15	6	33	0
p12	15	14	5	14	9	30	1
p13	24	5	8	22	20	60	1
p14	18	8	3	18	14	41	1
p15	21	9	5	18	11	35	2
p16	24	12	8	19	14	48	1
p17	16	8	2	15	13	32	0
p18	22	14	5	22	11	55	1
p19	13	6	8	13	10	22	1
p20	29	12	5	25	20	52	2
p21	12	15	8	12	9	18	1
p22	23	26	10	18	15	31	1
p23	19	11	6	13	6	17	1
p24	20	8	2	16	15	32	0
p25	27	7	11	25	12	36	1
p26	27	14	5	25	14	41	1
p27	15	7	4	15	8	27	2
p28	13	9	3	11	7	21	1
p29	23	13	10	20	12	37	2
p30	26	4	2	24	13	35	3

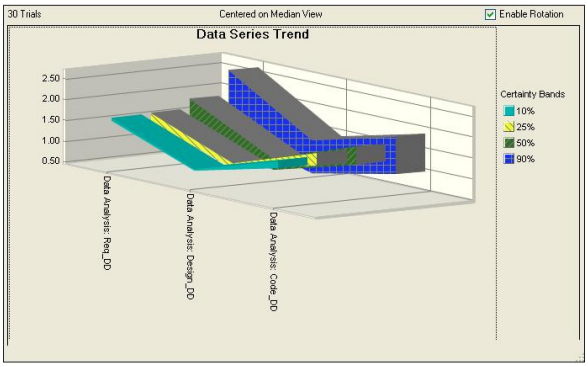

Fig. 3. Analysis by defect detection by given data series.

the effectiveness with the spent effort. Again, this is not directly usable in the analytical model but nevertheless can give further insights into the relationships of factors. Analytical analysis techniques are evaluated in[13,14]. Interface consistency rules and anomaly analysis revealed 2 and 4 faults of 28, respectively. It also analyzed the effectiveness of programming code bug finding tools in[1,4]. After eliminating the false positives, the tools were able to find 95% of the known defects over several projects[20]. However, the defects had mainly a low severity. For the severest defects the effectiveness reduced to 12%–15%, for the second severest defects even to 10%. For lower severities the effectiveness lies between 80%–95%.

7. Conclusion and Future Scope

The paper presents application of the defects origin iterative method for finding the approximated solution of the Statistical Analysis of Defect Removal Effectiveness problem. The software companies spend more amount of cost on finding the defects and bad fixing. At these averages below 95% in cumulative Statistical Analysis of Defect Removal Effectiveness is not adequate in software quality methods and needs immediate improvements. Any company or government group that does not measure Defects and does not know how efficient they are in finding software bugs or defect prior to release is in urgent need of remedial quality improvements. When companies that do not measure pre-defects are studied by the author during on-site benchmarks, they are almost always below 85% in Statistical Analysis of Defect Removal Effectiveness and usually lack adequate software quality methodologies; inadequate defect prevention and inadequate defect removal effectiveness are strongly correlated with failure to measure phase defect removal efficiency. For software quality is not only free but leads to shorter development schedules, lower development costs, and greatly reduced costs for maintenance and total costs for ownership.

References

[1] Sakthi Kumaresh and R. Baskaran, Defect Analysis and Prevention for Software Process Quality Improvement, *International Journal of Computer Application*, (0975–8887) vol. 8, no. 7, (October 2010).
[2] V. Suma and T. R. Gopalakrishnan Nair Defect Management Strategies in Software Development, Recent, *Advances in Technologies*, Maurizio A. Strangio (ED.), (2009).
[3] Stephen H. Kan, Metrics and Models in Software Quality Engineering, Second Edition; Pearson, (2003).
[4] Tom Walton, Quality Planning for Software Development, *IEEE*, (7695–1372), (2001).
[5] K. S. Jasmine and R. Vasantha, DRE – A Quality Metric for Component Based Software Products, *Proceeding of World Academy of Science, Engineering and Technology*, vol. 23, ISSN 1307–6884, (2007).
[6] T. R. Gopalakrishnan Nair and V. Suma, A Paradigm for Metric Based Inspection Process for Enhancing Defect Management, *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 3, (May 2010).
[7] L. K. Singh, A. K. Tripathi and G. Vinod, Software Reliability Early Prediction in Architectural Design Phase, *Overview and Limitations: Journal on Software Engineering and Applications* vol. 4, pp. 181–186, (2011).
[8] Guangtai Liang, Qian Wu, Qianxiang Wang and Hong Mei, An Effective Defect Detection and Warning Prioritization Approach for Resource Leaks, *IEEE* 30th *International Conference on Computer Software and Applications*, (0730–3157), (2012).
[9] Sandeep Kumar Nayak, Raees Ahmad Khan and Md. Rizwan Beg, Requirement Defect Identification an Early Stage Perspective, *IJCSI International Journal of Computer Science Issues*, vol. 9, Issue 5, no. 1, (September 2012).
[10] Pankaj Jalote and Naresh Agarwal, Using Defect Analysis Feedback for Improving Quality and Productivity in Iterative Software Development, In *Proc. ITI* 3rd *International Conference on Information and Communications Technology*, pp. 703–713, (2007).
[11] Liang Tian and A. Noore, Modeling Distributed Software Defect Removal Effectiveness in the Presence of Code Churn, *Mathematical and Computer Modelling*, vol. 41, ISSN 379–389, (2005).
[12] S. Wagner, A Model and Sensitivity Analysis of the Quality Economics of Defect – Detection Techniques, In *Proc. International Symposium on Software Testing and Analysis (ISSTA '06), ACM Press*, (2006).
[13] Abdul K. Khan, Software Excellence Augmentation Through Defect Analysis and Avoidance, Science, *Technology and Art Research Journal*, (Jan.–Mar. 2013).
[14] R. Jayanthi and M. Lilly Florence, A Study on Software Metrics and Phase Based Defect Removal Pattern Technique For Project Management, *International Journal of Soft Computing and Engineering*, (September 2013).
[15] A. Singh and R. Singh, Assuring Software Quality using Data Mining Methodology: A Literature Study, *International Conference on Information Systems and Computer Networks (ISCON)*, (2013).
[16] B. Singh and S. P. Kannojia, A Review on Software Quality Models, *International Conference on Communication Systems and Network Technologies (CSNT)*, (2013).
[17] A. Berry and Z. Milosevic, Real – Time Analytics for Legacy Data Streams in Health: Monitoring Health Data Quality, 17th *IEEE International on Enterprise Distributed Object Computing Conference (EDOC)*, (2013).
[18] S. Bhattacharjee, A. Bhadauria and I. K. G. Kumar, Managing Product Delivery Quality in Distributed Software Development, 8th *International Conference on Global Software Engineering (ICGSE)*, (2013).
[19] L. L. Minku, D. Sudholt and X. Yao, Improved Evolutionnary Algorithm Design for the Projec Scheduling Problem Based on Runtime Analysis, *IEEE Computer Society.*, (2014).
[20] G. Gopinath and D. Vijay, Towards Reduction of Cost of Software Quality by Implementing Regression Automation, *Journal of Industrial and Intelligent Information*, vol. 2, (2014).