Global Journal of Computer Science and Technology
Software & Data Engineering
Volume 13 Issue 7 Version 1.0 Year 2013
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

A New Approach for Reducing the Testing Effort

By Muhammad Shahid Khan, Naveed Khan, Muhammad Abid Khan & Muhammad Ahmed Javed

Gandhara University, Pakistan

Abstract - In-Process testing metrics has been used from some years and its usage is frequently increasing. There are different metrics for software testing i.e. to measure testing progress, Mean time between arrival of error, density of errors, fixation of errors, failure rate, test execution Productivity, cost of defects, Test efficiency and efficiency checking and so on. But all these metrics are independent and have no relation with each other. There are some attributes of testing metrics which are very much homogenous and interrelated with interdisciplinary measurement. It is quit natural to inter-relate all these metrics into a single metric which should provide overall functionality of some of existing selected metrics and also depicts some new approach of testing measurement. So the derived frame work modeled a new metric. This new metric covers the measurement of major quality attributes such as Correctness, Reliability, Efficiency, Flexibility, Inter-operability, Usability and Maintainability. The derived new metric possess higher level of reliability, early predication of testing progress, less cost of correctness in maintenance phase, effectiveness in error exploration, efficient approach of measuring testing process, Compatibility of different existing metrics, reduce the corrective maintenance effort, less cost of corrective maintenance, a new stander for measuring corrective maintenance effort, high degree of flexibility and interoperability of different Tools.

GJCST-C Classification : D.2.5

Strictly as per the compliance and regulations of:

A New Approach for Reducing the Testing Effort

Muhammad Shahid Khan [α], Naveed Khan [σ], Muhammad Abid Khan [ρ] & Muhammad Ahmed Javed [ω]

Abstract - In-Process testing metrics has been used from some years and its usage is frequently increasing. There are different metrics for software testing i.e. to measure testing progress, Mean time between arrival of error, density of errors, fixation of errors, failure rate, test execution Productivity, cost of defects, Test efficiency and efficiency checking and so on. But all these metrics are independent and have no relation with each other. There are some attributes of testing metrics which are very much homogenous and interrelated with interdisciplinary measurement. It is quit natural to inter-relate all these metrics into a single metric which should provide overall functionality of some of existing selected metrics and also depicts some new approach of testing measurement. So the derived frame work modeled a new metric. This new metric covers the measurement of major quality attributes such as Correctness, Reliability, Efficiency, Flexibility, Inter-operability, Usability and Maintainability. The derived new metric possess higher level of reliability, early predication of testing progress, less cost of correctness in maintenance phase, effectiveness in error exploration, efficient approach of measuring testing process, Compatibility of different existing metrics, reduce the corrective maintenance effort, less cost of corrective maintenance, a new stander for measuring corrective maintenance effort, high degree of flexibility and interoperability of different Tools.

I. Introduction

The metrics are used to measure the software i.e. software metric is a measure of some property of a piece of software or its specifications (Class et al.1994). The different metrics are used to measure the different phases of software in order to determine the progress of software development process.

The different testing metrics are used independently to covers different aspects of software testing process. Some of these testing metrics are as below Cost of finding a defect in testing (CFDT): Test Case Adequacy: Test Case Effectiveness: Effort Variance : Schedule Variance: Schedule Slippage: Rework Effort Ratio: Review Effort Ratio: Requirements Stability Index: Requirements Creep: Weighted Defect Density (Jaana Lindroos, 2005).

A software will remain in the market for a long time if it is developed with disciplined approach such type of software are easier to use and easily modifiable. Such software is the result of good effort. A type of software, may not modify and difficult to use. Such software should be the result of unsuccessful efforts and undisciplined approach. For the development of the software required effort is divided into two parts.

In system software errors and failures are increase the negative impact due to this the failure cost of the software is also increase (Beheshti et al., 1995).

Recent research on the quality of the software has resulted in the wide range of software metrics and analysis techniques (Prather, 1995).

Software metrics, such as the MTBF, are designed to provide objective criteria for management decisions. Precision instruments are not necessary to calculate most metrics. Simple counting or subjective estimation has been used. Most software metrics define a standard way of using attributes such as size, cost, defects, complexity, and environment to measure quality parameters such as completeness, conciseness, portability, consistency, usability, and structure.

A variety of metrics can assist in identifying risks early in the test process. Nonparametric statistical principles have been used to evaluate the effectiveness of metrics in identifying these risks, as well as other validity criteria.

These validity criteria include association, consistency, discriminative power, predictability, and tracking. The use of metrics in debugging software can more effectively reduce the scope of error when structured and modular programs are employed in software development. Metrics do not always provide useful information. Some metrics are designed for a particular purpose and may not reveal the existence of errors of which the user is unaware. The lack of an omnibus metric to detect all types of errors has spurred interest in the development of additional metrics.

(Rubin et al., 1995) have developed metrics with a mechanism termed "software process flight simulation" to allow IS professionals to explore their mental models of the software process. They emphasize choosing the right metrics for the modeling process. They evaluated the metrics: software size; software reliability; test session efficiency; test focus; and software maturity. Inputs for evaluating these metrics included: discrepancy report count, impact and subsystem charged; scheduled test time; effective test time; and test session rating. By tracking test results of these metrics, significant insight was gained on software quality, cost allocation, and test scheduling.

Effectively managing process improvement for software-quality can challenge any project manager. There are programs available to perform a quality check on programs sold by manufacturers. However, many of

Author α : Gandhara University, Peshawar, Pakistan.
E-mail : shahidkhan123@gmail.com
Author σ : E-mail : naveediit@gmail.com
Author ρ : E-mail : engrabid08@gmail.com
Author ω : E-mail : ahmed.javed725@gmail.com

these types of programs are at least three times as long as the programs that they are designed to check. Despite efforts to detect software errors, some information specialists report at least one mistake per 5,000 lines of code (Beheshti et al., 1995).

II. Proposed Methodology

This chapter will cover proposed technique, working of the model, the derived frame work as a new metric and formulae for inter-related metrics.

a) Proposed Technique

In proposed methodology a new frame work is introduced for measuring software testing process. This framework has combined different metrics of testing projects and derived a new metric which covers different testing aspects.

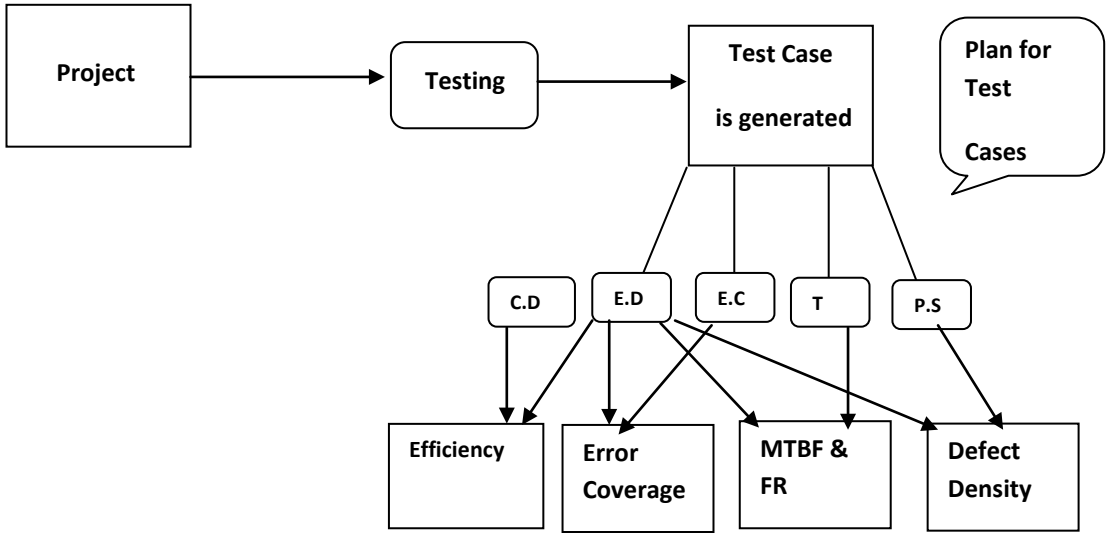


Figure 1 : Proposed Model for derived framework

b) Working of the Model

A project or system is taken to perform testing on it. At first the test cases are generated and then proper planning is done for each test case. Then the further steps will be taken to have different measures for same project such as C.D stands for Error on Customer side, E.D stands for Error Detection, E.C for Error Correction, T stands for Time taken by each test case, P.S stands for Program size tested.

c) The Derived Framework as a New Metric

Test cases Id	Time	Error	Customer side Error	Corrected	Error Cover In %	MTBF	FR	Size	Efficiency In %	DD
1	12	1	1	1	100	12	.083	85	50	1.18
2	10	3	4	2	66.67	3.34	.3	78	60	3.84
-	-	-	-	-	-	-	-	-	-	
-	-	-	-	-	-	-	-	-	-	

d) Formulae for Inter-Related Metrics

i) $$MTBF = \frac{Time}{No.\ of\ Error}$$

ii) $$Failure\ Rate = \frac{No.\ of\ Error}{Time}$$

iii) $$Error\ Coverage = \frac{No.\ of\ Corrected\ Error}{No.\ of\ Errors} * 100$$

iv) Defect Density = $\frac{\text{No. of Errors}}{\text{Program Size Tested}} * 100$

v) Efficiency = $\frac{\text{No. of Error}}{\text{(No. of Error + Customer Side Error)}} * 100$

III. Implemented Proof and Results

This chapter will give you the practical approach of the system I proposed.

The system was implemented using VB.NET and SQL. The results were generated using P-IV with 2.8 GHz processor, 1GB of memory running windows XP 2003. The experiments were performed on a selected project which was developed in VB.NET and SQL server 2003. The accuracy of the system was checked by applying different values in different attributes.

a) Experiments and Results

We have selected a project for implementation of testing metrics. And twenty four (24) test cases are generated for the selected project. Where three operations such as insertion, deletion and updation test cases are generated for each of this operation of individual form of project. Black box testing method is used to check that weather the particular operation is giving the required output from specific input. If the particular operation is not performing its functionality then white box testing is performed. In white box testing method the code is tested and errors of code are detected.

Test Case Name	Insertion	Deletion	Update
	Test Case ID	Test Case ID	Test Case ID
Item Detail	1	9	17
Item Price	2	10	18
Ware House	3	11	19
Sale Detail	4	12	20
Receipt Detail	5	13	21
Good Return	6	14	22
Good Returns from customer	7	15	23
Demand Detail	8	16	24

Table 1 : Name and ID of Test Case

In the figure-1, 8-Test Cases have its own identification number for a specific (insertion, deletion, Updation) operation.

b) S-Curve Testing Plan

Test cases are made according to S-curve each test case is planned and attempted. In this way progress of the testing process is traceable and error exploration is performed more efficiently.

Each test case is passed Through S-curve plan because every test case will be planned in detail.

i. *S-Curve Plan for Insertion Operation*

For more error exploration from the selected project, different test cases are applied on the insertion operation. In insertion operation, all of eight (8) test cases of are attempted but four (4) are successful as shown in the figure-2. And graph of the insertion operation is illustrated as shown in the figure-2.

Test Cases Name	Test Case Planed	Attemted	Successful
Systems	8	8	4
Item	2	2	1
Ware House	1	1	1
Sale	1	1	0
Receipt	1	1	1
Return Goods	2	2	1
Demand	1	1	0

Table 2 : Plan for insertion operation

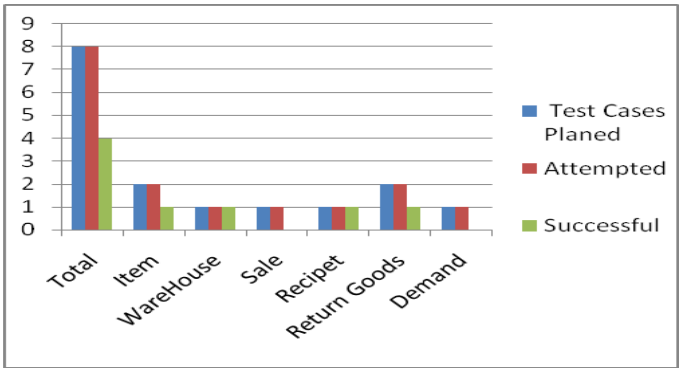


Figure 2 : Graph for insertion Operation

ii. *S-Curve Plan for Deletion Operation*

In deletion operation, all of eight (8) test cases are attempted but five (5) are successful as shown in the figure-4. And graph of deletion operation is illustrated in figure-5.

Test Cases Name	Test Case Planed	Attempted	Successful
Systems	8	8	5
Item	2	2	2
Ware House	1	1	1
Sale	1	1	0
Receipt	1	1	0
Return Goods	2	2	1
Demand	1	1	1

Table 3 : Plan for deletion operation

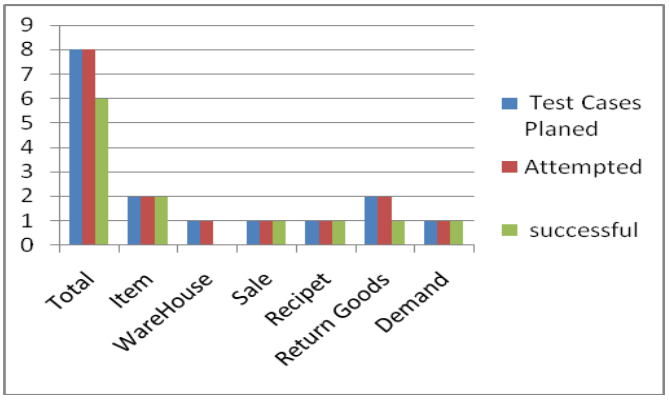


Figure 3 : Graph for Deletion Operation

iii. *S-Curve Plan for Updation Operation*

In Updation operation, all test cases are attempted eight (8) but five (5) were successful as shown in the figure-6. And graph of Updation operation is illustrated in figure-7.

Table 4 : Plan for Updation Operation

Test Cases Name	Test Case Planed	Attempted	Successful
Systems	8	8	5
Item	2	2	2
Ware House	1	1	1
Sale	1	1	0
Receipt	1	1	0
Return Goods	2	2	1
Demand	1	1	1

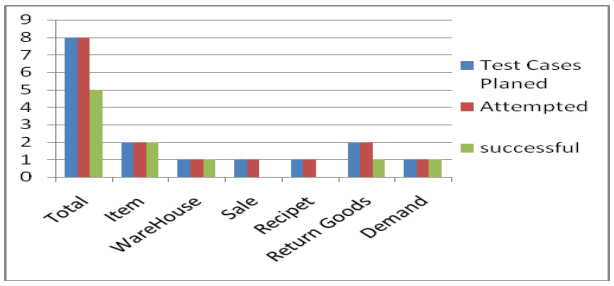
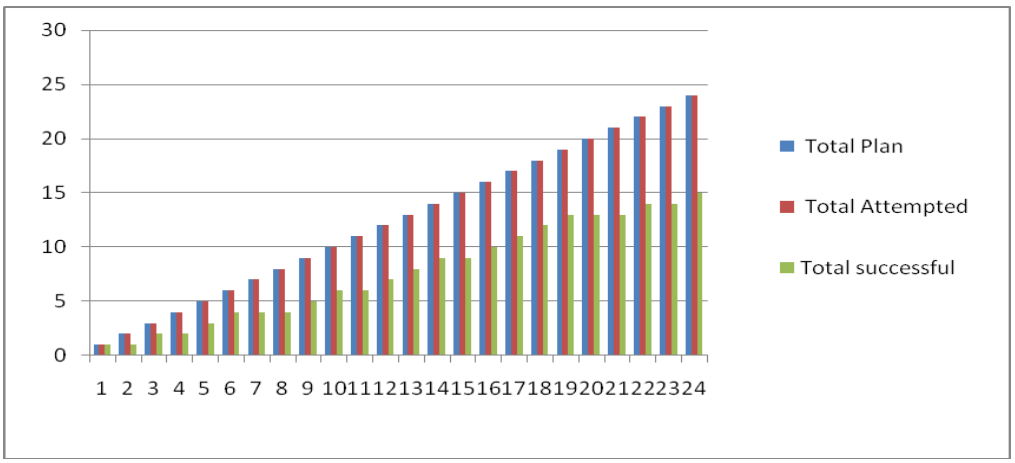


Figure 4 : Graph for Updation Operation



Over all result of test plan of all operations

IV. Comparative Study of Single Project

a) Without single framework

In selected project there are total 24-Test cases but due to lack of proper planning the system named "Goods Returns from customer" are left from testing. There are two (2) test cases of system "Goods Returns from customer" and there are three (3) operation of "Good Returns from customer" system so

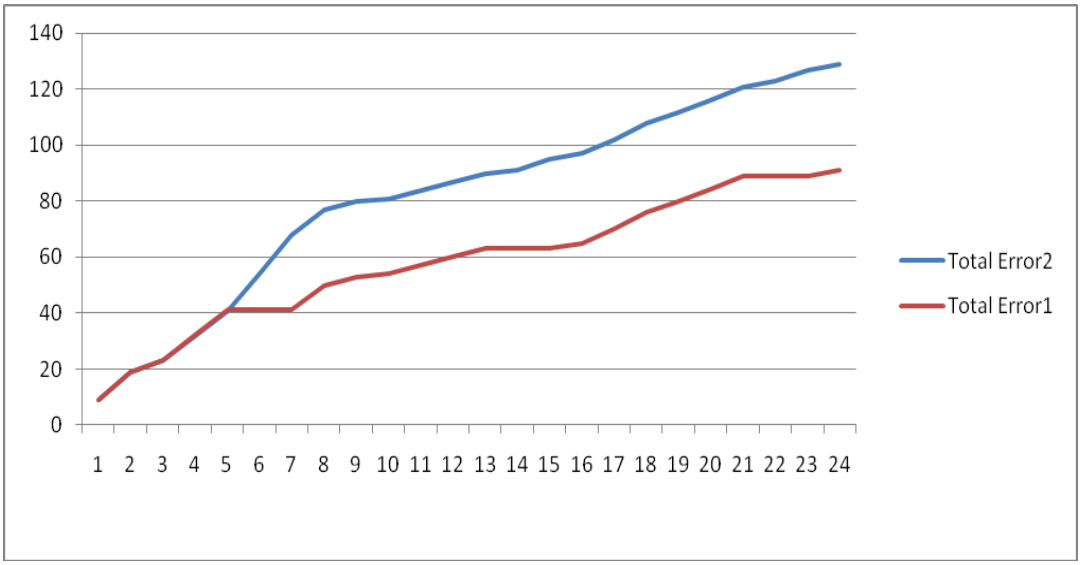
$$2*3=6$$

i.e six (6) test cases are left to be tested. So the errors of these six (6) test cases are thirty eight (38) which are left from fixing at developer side, i.e 27, 5 and 6 errors of insertion, deletion and updation respectively. The errors detected on developer side without single framework is 50,15 and 26 for insertion, deletion and updation respectively.

While using single framework there is a plan for testing via s-curve and the error are explored at big ration.

The following graph shows a clear picture of detecting errors at high ratio via using single framework. Where Error1 line shows the error detection of particular project without using single frame work and Error2 line shows the error detection of particular project with using single frame work.

Error Detection Report		
Operations	With Single Frame Work	Without Single Frame Work
Insertion	77	50
Deletion	20	15
Updation	32	26



b) Impact on Efficiency

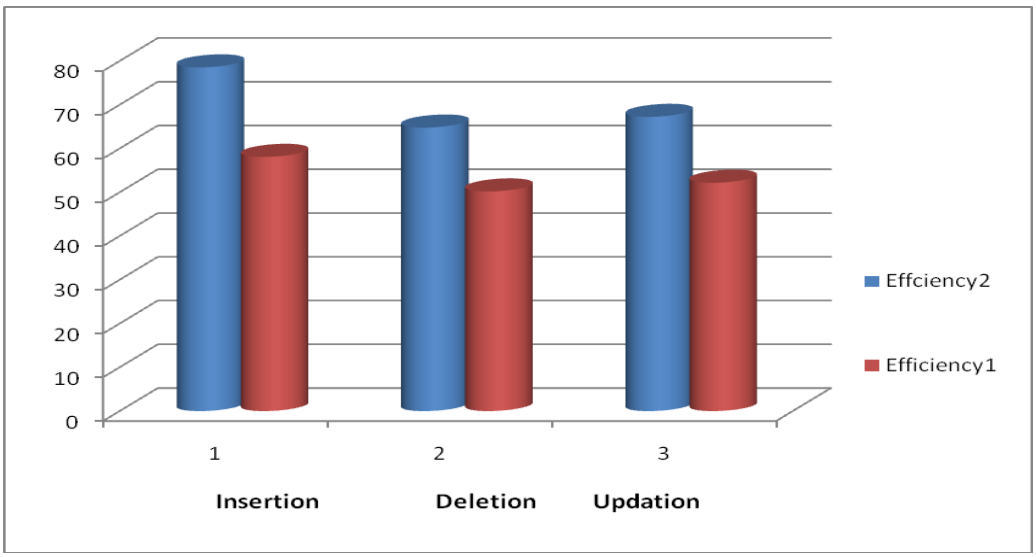
The less exploration of errors and has a direct impact on software testing efficiency. Because those all errors which are left on developer side would be appeared on customer side.

The customer side error with single frame work is 21,10,15 for insertion, deletion and updation respectively.

The customer side error without single frame work is 48,15 and 21 for insertion deletion and updation respectively as shown below:

21+**27**= 48, 10+**5**= 15, 15+**6**= 21.

Average Efficiency Report		
Operations	With Single Frame Work	Without Single Frame Work
Insertion	78.549	58.099
Deletion	64.791	50.208
Updation	67.247	52.188



Impact on corrective maintinance effort

V. Conclusion

This new frame work provides a proper way of measuring different aspects of testing process. There is a number of software testing metrics such as Failure Rate, MTTF, MTBF, Defect Density, Test Plan, Testing Efficiency, Error fixation etc. These metrics are independent and have no link with each other, but some these have homogeneous attributes with inter-disciplinary measurement. Some of these homogeneous attributes metrics has been integrated into single frame work. So from this single frame work some major measurement of testing process such that plan for testing, error detection and correction efficiency, program size tested, testing efficiency, execution time take by each test case, the reliability measurement using MTTF and FR can be performed. This derived frame provides the measurement of software testing process at different stages, from which the effect of each step taken in software testing is determined. The derived metric also provides the higher degree of flexibility where different metrics are combined in a single frame work to be used more effectively and also provides the compatibility of different metrics. The term plan testing is also included in new derived metric which is helpful in exploring more errors, and this reduces the corrective maintenance cost. From this derived metric more reliable and efficient product can be achieved with low cost of maintenance. It has a direct impact on product quality because this new metric provides the measurement of some basic attributes of quality such that Correctness, Reliability, Efficiency, Usability and Maintainability. This new metric is applied on different tools such that VB.Net and SQL and provides the interoperability of different tools.

References Références Referencias

1. Beheshti, H.M. and Worley, J.K. (1995), "Automated systems and reliability", Industrial Management & Data Systems, Vol. 95 No. 1, pp. 5-9.
2. Chan, F.T., Chen, T.Y., Mak, I.K. and Yu, Y.T. (1996), "Proportional sampling strategy: guidelines for software testing practitioners", Information and Software Technology, Vol. 38 No. 12, pp. 775-82.
3. Classe, A. (1994), "Down with downtime", Accountancy, Vol. 114 No. 1212, pp. 57-60.
4. Cole, B. (1996), "Embedded top priority: reliability", Electronic Engineering Times, No. 932, p. 57.
5. Dhillon, B.S. (1987), Reliability in Computer System Design, Norwood, NJ.
6. E. Simmons, "When Will We be Done Testing? Software Defect Arrival Modeling Using the Weibull Distribution," presented at Pacific Northwest Software Quality Conference, Portland, OR, 2000.
7. Feigenbaum, A.V. (1983), Total Quality Control, McGraw Hill, New York, NY.
8. Fenton, N. (1994), "Software measurement: a necessary scientific basis", IEEE Transactions on Software Engineering, Vol. 20 No. 3, pp. 199-206.
9. Foody, M. (1995), "When is software ready for release?", Unix Review, Vol. 13 No. 3, pp. 35-41.
10. http://www.freetutes.com/systemanalysis/sa2-error-distribution-with-phases.html
11. http://en.wikipedia.org/wiki/Software_metric.
12. Jaana Lindroos. Feb 2005 the Role of Program Structure in Software Maintenance, University of Helsinki.
13. Kvanli, A.H., Guynes, C.S. and Pavur, R.J. (1996), Introduction to Business Statistics, West Publishing Co., New York, NY.
14. Littlewood, B. and Strigini, L. (1993), "Validation of ultrahigh dependability for software-based systems", Communications of the ACM, Vol. 36 No. 11, pp. 69-73.
15. Nesi, P. and Campanai, M. (1996), "Metric framework for object-oriented real-time systems specification languages", Journal of Systems and Software, Vol. 34 No. 1, pp. 43-65.
16. Nunamaker, J.F., Chen, M. and Purdin, T.D.M. (1990), "Systems development in information systems research", Journal of Management Information Systems, Vol. 7 No. 3, pp. 89-106.
17. Omdahl, T.P. (1988), Reliability, Availability, and Maintainability, ASQC Quality Press, Milwaukee.
18. Prather, R.E. (1995), "Design and analysis of hierarchical software metrics", ACM Computing Surveys, Vol. 27 No. 4, pp. 497-518.
19. Prof. Stafford. December 2003 Software Maintenance As Part of the Software Life Cycle, Tufts University.
20. Putnam, L.H. and Myers, W. (1992), Measures for Excellence, Prentice-Hall, Upper Saddle River, NJ.
21. Ross, P.E. (1994), "The day the software crashed", Forbes, Vol. 153 No. 9, pp. 142-56.
22. Rubin, H.A., Johnson, M. and Yourdon, E. (1995), "Software process flight simulation: dynamic modeling tools and metrics", Information Systems Management, Vol. 12 No. 3, pp. 40-52.
23. Schmidt, D., Fayad, M. and Johnson, R. (1996), "Software patterns", Communications of the ACM, Vol. 39 No. 10, pp. 36-9.
24. Schneidewind, N.F. (1992), "Methodology for validating software metrics", IEEE Transactions on Software Engineering, Vol. 18 No. 5, pp. 410-22.
25. Schneidewind, N.F. (1993), "Software reliability model with optimal selection of failure data", IEEE Transactions on Software Engineering, Vol. 19 No. 11, pp. 1095-104.
26. Stark, G.E., Durst, R.C. and Pelnik, T.M. (1992), "Evaluation of software metrics for NASA's Mission Control Center", Software Quality Journal, Vol. 1 No. 2, pp. 115-32.

27. Van Genuchten, M. (1993), "Analysis and improvement of software engineering processes", Information & Management, Vol. 25, pp. 43-9.
28. Vogel, D.R. and Wetherbe, J.C. (1984), "MIS research: a profile of leading journals and universities", Data Base, Vol. 16 No. 3.
29. Zhu, H.(1996), "A formal analysis of the subsume relation between software test adequacy criteria", IEEE Transactions on Software Engineering, Vol. 22 No. 4, pp. 248-55.

Global Journals Inc. (US) Guidelines Handbook 2013

www.GlobalJournals.org