International Journal of Electronics Communication and Computer Engineering
Volume 7, Issue 6, ISSN (Online): 2249–071X

Role of Risk as an External Quality Character in the Software Quality Metrics

T John Vijay[1], Dr.T.V.Rao[2], Dr. M. Gopi Chand[3], Dr. Done Harika[4].
Research Scholar, JNTU, Hyderabad
Professor& Head, CSE, PVSIT, Vijayawada
Professor & Head IT, VCE, Hyderabad
Asst. Professor, SVEC, Tadepalligudem

***Abstract* – Software quality assurance is a process for guesstimating and documenting the quality of the software products during each phase of the software development lifecycle. Measurement provides a way to assess the status of any program to determine if it is in trouble or in need of corrective action and process of improvement. This assessment helps to diagnose the status in terms of time, completion, risks and also the quality and thus ensures to take meaningful, effective remedial action (e.g., controlling requirements changes, improving response time to the developer, relaxing performance requirements, extending your schedule, adding money, or any number of options).The practice of applying software metrics to operational factors and to maintain factors is a complex task. Successful software quality assurance is highly dependent on software metrics. It needs linkage the software quality model and software metrics through quality factors in order to offer measure method for software quality assurance. The contributions of this paper to build an appropriate method of Software quality metrics based on external factors one of it is Risk, which will influence the quality of the final product. In this study, we will see how the Risk is playing a vital role in the quality of the software.**

***Keywords* – Risk, Software External Characters, Software Metrics, Software Quality.**

I. INTRODUCTION

A Metric is a quantitative measure of the degree to which a system, system component, or process possesses a given attribute. Metrics can be defined as “STANDARDS OF MEASUREMENT”, simply, a unit used for describing an attribute or a scale for measurement. Software Metrics are used to measure the quality of the project. Software metric deals with the measurement of software product and software product development process and it guides and evaluates software development [39].

These metrics might help to identify issues involved in the process like:

- How many defects are existed within the module?
- How many test cases are executed per person?
- What is the Test coverage %?

Software Test Measurement is the quantitative indication of extent, amount, dimension, capacity, or size of some attribute of a product or process. Test Metrics are used to take the decision for next phase of activities such as, estimate the cost & schedule of future projects, understand the kind of improvement required to success the project and also to take decision on process or technology to be modified etc. thus, Test Metrics are the most important to measure the quality of the software.

Software Test Measurement is the quantitative indication of extent, amount, dimension, capacity, or size of some attribute of a product or process. Test Metrics are used to take the decision for next phase of activities such as, estimate the cost & schedule of future projects, understand the kind of improvement required to success the project and also to take decision on process or technology to be modified etc. thus, Test Metrics are the most important to measure the quality of the software.

Application of test metrics

If metrics are not followed, then the work completed by the test analyst will be subjective i.e. the test report will not have the proper information to know the status of the work/project. If Metrics are involved in the project, then the exact status of the work with proper numbers/data can be published, i.e. in the Test report, one can publish the details of the project or programme like:

1. How many test cases have been designed per requirement?
2. How many test cases are yet to design?
3. How many test cases are executed?
4. How many test cases are passed/failed/blocked?
5. How many test cases are not yet executed?
6. How many defects are identified & what is the severity of those defects?
7. How many test cases are failed due to one particular defect? etc.

Based on the above metrics, test lead/manager will get the understanding of the below mentioned key points.

a) %ge of work completed
b) %ge of work yet to be completed
c) Time to complete the remaining work
d) Whether the project is going as per the schedule or lagging? etc.

Further, based on the metrics, if the project is not going to complete as per the schedule, then the manager will raise the alarm to the client and other stake holders by providing the reasons for lagging to avoid the last minute surprises.

Risk

A risk is any factor that may potentially interfere with successful completion of the project. A risk is not a problem -- a problem has already occurred. A risk is the recognition that a problem might occur, by recognizing potential problems, the project manager can attempt to avoid a problem through proper actions. The objectives of Project Risk Management are to increase the probability and impact of positive events, and decrease the probability and impact of events adverse to the project. Risk assessment is a systematic process of evaluating the potential risks that may be involved in a projected activity

or undertaking.

The contributions of this paper to build an appropriate method of Software quality metrics based on external factors one of it is Risk, which will influence the quality of the final product. In this study, we will see how the Risk is playing a vital role in the quality of the software.

II. REVIEW OF LITERATURE

Software Quality Metrics

This section concentrates on different metrics found in the software engineering literature. A classical classification of the software quality metrics: Halstead's software metrics, McCabe's cyclomatic complexity metric, RADC's methodology, Albrecht's function points metric, Ejiogu's software metrics, and Henry and Kafura's information metric.

Halstead's Software Metrics

Halstead's measure for calculation of module conciseness is essentially based on the assumption that a well-structured program is a function of only its unique operators and operands. The best predictor of time required to develop and run the program successfully was Halstead's metric for program volume.

Halstead [11] defined the following formulas of software characterization for instance

The measure of vocabulary: $n = n_1 + n_2$

Program length: $N = N_1 + N_2$

Program volume: $V = N \log_2 n$

Program level: $L = {}^{V^*}\!/_{V}$

Where

n_1 = the number of unique operators

n_2 = the number of unique operand

N_1 = the total number of operators

N_2 = the total number of operands

Christensen et al. [12] have taken the idea further and produced a metrics called difficulty. V^* is the minimal program volume assuming the minimal set of operands and operators for the implementation of given algorithm:

$$Program\ effort\ E = (V*)/L$$

$$Difficulty\ of\ implementation\ D = \frac{n_1 N_2}{2n_2}$$

$$Programming\ in\ SecondsT = \frac{E}{S}$$

based on difficulty and volume Halstead proposed an estimator for actual programming effect, namely.

Effort = difficulty * volume

McCabe's Cyclomatic complexity metrics

McCable [13] has proposed a complexity metric on mathematical graph theory. The complexity of a program is defined in terms of its control structure and is represented by the maximum number of "linearly independent" path through the program. Software developer can use this measure to determine which modules of a program are over-complex and need to be re-coded.

The formulas for the cyclomatic complexity proposed by [42] are:

$V(G) = e-n+2p$

Where

e = the number of edges in the graph

n = the number of nodes in the graph

P = the number of connected components in the graph.

The Cyclomatic complexity metric is based on the number of decision elements (IF-THEN-ELSE, DO WHILE, DO UNTIL, CASE) in the language and the number of AND, OR, and NOT phrases in each decision. The formula of the metric is: Cyclomatic complexity = number of decisions +number of conditions + 1[14].

The Essential complexity metricb is based on the amount of unstructured code in a program. Modules containing unstructured code may be more difficult to understand and maintain. The essential complexity proposed by McCable [13]:

$$EV\ (G) = V\ (G) - m$$

Where

V (G) = the cyclomatic complexity

m = the number of proper sub graphs

McCabe's Cyclomatic complexity measure has been correlated with several quality factors.

Project Metrics

PMI PMBOK (Project management institute's project management body of knowledge) describes Project Management Processes, tools and techniques and provides one set of high level businesses for all industries. The PMBOK includes all nine knowledge areas and all associated with them tools and techniques: Integration management, Scope management, Time management, Cost management, Quality management, Human Resource management, Communication Management, Risk management, and Procurement management (PMBOK [20]).

Some of those processes often are not applicable or even irrelevant to Software Development industry. CMM (Capability Maturity model) speaks about software project planning processes without mentioning specific methodologies for project estimating described in PMBOK (PMBOK). Basic key process areas (KPA) of the SEI CMM is requirement management, project planning, project tracking and oversight, subcontract management, quality assurance, and configuration management.

Test Product and Process Metrics

Test process metrics provide information about preparation for testing, test execution and test progress. Some testing metrics ([27,32,33]) as following:

1. Number of test cases designed
2. Number of test cases executed
3. DA = Number of defects rejected / Total number of defects *100%
4. Bad Fix Defect =Number of Bad Fix Defect / Total number of valid defects *100%
5. Test case defect density = (Number of failed tests / Number of executed test cases) *100
6. Total actual execution time/ total estimated execution time
7. Average execution time of a test case

Test product metrics provide information about the test state and testing status of a software product. Using these metrics we can measure the products test state and indicative level quality, useful for product release decision [33].

1. Test Efficiency = $(D_T/(D_T+D_U)*100$
2. Test Effectiveness = $(D_T/(D_F+D_U)*100$
3. Test improvement TI = number of defects detected by the test team during / source lines of code in thousands
4. Test time over development time TD = number of business days used for product testing / number of business days used for product
5. Test cost normalized to product size (TCS) = total cost of testing the product in dollars / source lines of code in thousands
6. Test cost as a ration of development cost (TCD) = total cost of testing the product in dollars / total cost of developing the product in dollars
7. Test improvement in product quality = Number of defects found in the product after release / source lines of code in thousands
8. Cost per defect unit = Total cost of a specific test phase in dollars / number of defects found in the product after release
9. Test effectiveness for driving out defects in each test phase = $(D_D/(D_D+D_N)*100$
10. Performance test efficiency (PTE) = requirement during perform test / (requirement during performance time + requirement after signoff of performance time) * 100%
11. Cost per defect unit = Total cost of a specific test phase in dollars / number of defects found in the product after release
12. Estimated time for testing
13. Actual testing time
14. % of time spent = (actual time spent / Estimating time)*100

Where

DD: Number of defects of this defect type that are detected after the test phase.

DT: Number of defects found by the test team during the product cycle

DU: Number of defects of found in the product under test (before official release) DF : Number of defects found in the product after release the test phase

DN: Number of defects of this defect type (any particular type) that remain uncovered after the test phase.

Statistical Analysis

The revisions on the software measurement methods, developed with the purpose of improving their consistency must be empirically evaluated so as to determine to what extent is the pursued goal fulfilled. The most used statistical methods are given in the following table ([33] ~ [39]). Some commonly used statistical methodology (include nonparametric tests) are discussed as follow:

1. Ordinary least square regression models: Ordinary least square regression (OLS) model is used to subsystem defects or defect densities prediction
2. Poisson models: Poisson analysis applied to library unit aggregation defect analysis
3. Binomial analysis: Calculation the probability of defect injection
4. Ordered response models: Defect proneness
5. Proportional hazards models: Failure analysis incorporating software characteristics
6. Factor analysis: Evaluation of design languages based on code measurement
7. Bayesian networks: Analysis of the relationship between defects detecting during test and residual defects delivered
8. Spearman rank correlation coefficient: Spearman's coefficient can be used when both dependent (outcome; response) variable and independent (predictor) variable are ordinal numeric, or when one variable is an ordinal numeric and the other is a continuous variable.
9. Pearson or multiple correlations: Pearson correlation is widely used in statistics to measure the degree of the relationship between linear related variables. For the Pearson correlation, both variables should be normally distributed
10. Mann – Whitney U test: Mann – Whitney U test is a non-parametric statistical hypothesis test for assessing whether one of two samples of independent observations tends to have larger values than the other
11. Wald-Wolfowitz two-sample Run test: Wald-Wolfowitz two-sample Run test is used to examine whether two samples come from populations having same distribution.
12. Median test for two samples: To test whether or not two samples come from same population, median test is used. It is more efficient than run test each sample should be size 10 at least.
13. Sign test for match pairs: When one member of the pair is associated with the treatment A and the other with treatment B, sign test has wide applicability.
14. Run test for randomness: Run test is used for examining whether or not a set of observations constitutes a random sample from an infinite population. Test of randomness is of major importance because the assumption of randomness underlies statistical inference.
15. Wilcoxon signed rank test for matcher pairs: Where there is some kind of pairing between observations in two samples, ordinary two sample tests are not appropriate.
16. Kolmogorov-Smirnov test: Where there is unequal number of observations in two samples, Kolmogorov-Smirnov test is appropriate. This test is used to test whether there is any significant difference between two treatments A and B

III. MATERIAL AND METHODS

We conducted a survey from questionnaires and project dashboards. Method can be defined as the main structure on which the research is based. Method can include different areas of research and proposed hypothesis.

In this model tester tests the product and raises the defects. We have taken the count of the defects from seven similar projects with a project span of not less than six months. We have collected the Risk register data for each of the project and developed a relationship between the External character Risk and the project quality indicators i.e defects.

Regarding the trademark restrictions we renamed the project names as P1, P2, P3 … P7. The collected data is

shown in Table-I.

Projects	DEFECTS	RISK
P1	245	29
P2	409	49
P3	298	36
P4	150	18
P5	352	42
P6	294	35
P7	**184**	**37**

Table-I

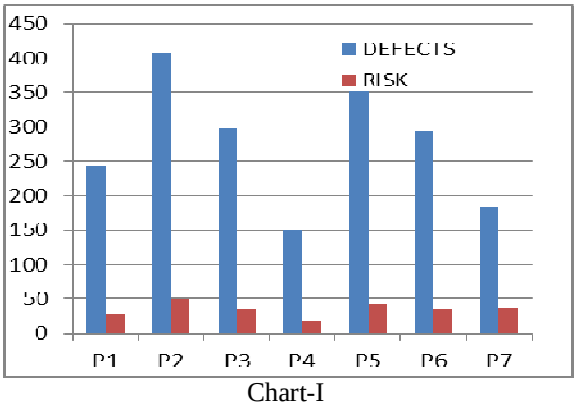

Chart-I

F-Test Two-Sample for Variances		
	Variable 1	*Variable 2*
Mean	276	35.14286
Variance	8262.333	95.80952
Observations	7	7
df	6	6
F	86.23708	
P(F<=f) one-tail	1.48E-05	
F Critical one-tail	4.283866	

Table-II

*The Data gathered through the survey and the project dash boards is analyzed by the statistical tools which are given below.

1. Ordinary least square regression models: Ordinary least square regression (OLS) model is used to subsystem defects or defect densities prediction
2. Poisson models: Poisson analysis applied to library unit aggregation defect analysis
3. Binomial analysis: Calculation the probability of defect injection
4. Ordered response models: Defect proneness
5. Proportional hazards models: Failure analysis incorporating software characteristics
6. Factor analysis: Evaluation of design languages based on code measurement
7. Bayesian networks: Analysis of the relationship between defects detecting during test and residual defects delivered
8. Spearman rank correlation coefficient: Spearman's coefficient can be used when both dependent (outcome; response) variable and independent (predictor) variable are ordinal numeric, or when one variable is an ordinal numeric and the other is a continuous variable.
9. Pearson or multiple correlations: Pearson correlation is widely used in statistics to measure the degree of the relationship between linear related variables. For the Pearson correlation, both variables should be normally distributed
10. Mann – Whitney U test: Mann – Whitney U test is a non-parametric statistical hypothesis test for assessing whether one of two samples of independent observations tends to have larger values than the other
11. Wald-Wolfowitz two-sample Run test: Wald-Wolfowitz two-sample Run test is used to examine whether two samples come from populations having same distribution.
12. Median test for two samples: To test whether or not two samples come from same population, median test is used. It is more efficient than run test each sample should be size 10 at least.
13. Sign test for match pairs: When one member of the pair is associated with the treatment A and the other with treatment B, sign test has wide applicability.
14. Run test for randomness: Run test is used for examining whether or not a set of observations constitutes a random sample from an infinite population. Test of randomness is of major importance because the assumption of randomness underlies statistical inference.
15. Wilcoxon signed rank test for matcher pairs: Where there is some kind of pairing between observations in two samples, ordinary two sample tests are not appropriate.
16. Kolmogorov-Smirnov test: Where there is unequal number of observations in two samples, Kolmogorov-Smirnov test is appropriate. This test is used to test whether there is any significant difference between two treatments A and B.

IV. Results and Discussion

Though there is no direct relation between form the results it is very clear that the Risk in any project is directly influencing the quality of the final product. A risk is any factor that may potentially interfere not only with successful completion of the project but also results in the quality of the project.

It is suggested that Risks can be controlled by continually monitoring and correcting risky conditions. This involves the use of reviews, inspections, risk reduction milestones, development of contingency, and similar management techniques. Controlling risk involves developing a risk reduction plan, then tracking to that plan.

In order to continue to improve its software product, processes, and customer services. Future research is need to extend and improve the methodology to extend metrics that have been validated on one project, using our criteria, valid measures of quality on future software project.

References

[1] Dromey RG. A model for software product quality. IEEE Transaction on Software Engineering. 1995;21:146-162.
[2] Jacobson I, Booch G, Rumbaugh J. The unified software development process, Addison Wesley; 1999.
[3] Krruchtem P. The rational unified process: an introduction, Addison Wesley; 2000.

[4] ISO 9001:2005, Quality management system Fundamentals and vocabulary; 2005.
[5] ISO 9001:2001, Quality management system Requirements; 2001.
[6] ISO/IEC25010: Software engineering– system and software quality requirement and evaluation (SQuaRE)- system and software quality model; 2011.
[7] Esaki K. System quality requirement and evaluation, importance of application of the ISO/IEC 25000 series, Global Perspectives of Engineering Management. 2013;2(2):52-59.
[8] Al-Qutaish RE. Quality models in software engineering literature: An analytical and comparative study. Journal of American Science. 2010;6(3):166-175.
[9] Bowen TP, Gray BW, Jay TT. RADC-TR-85-37, RADS, Griffiss Air Face Base N. Y.; 1985.
[10] Shanthi PM, Duraiswamy K. An empirical validation of software quality metric suits on open source software for fault-proneness prediction in object oriented system, European journal of Scientific Research. 2011: 5(2):168-181.
[11] Halstead, MH. Elements of software Science, New York, North-Holland; 1978.
[12] McCable TJ. A complexity measure, IEEE Transaction on Software Engineering, 1976; SE-2(4):308-320.
[13] Arthur, LJ., Measuring programmer productivity and software quality, John Wiley & Son, New York; 1985.
[14] Albrech AJ, Gaffney JE. Software function, source lines of code and development effort function: a software service validation, IEEE Transaction on Software Engineering. 1983;SE-9(6):639-648.
[15] Kemerer CF, Porter BS. Improving the reliability of function point measurement: an empirical study, IEEE Transactions on Software Engineering. 1992;18(11):1101-1024.
[16] Fjiogu LO. A unified theory of software metrics, Softmetrix, Inc. Chicago, IL. 1988;232238.
[17] Fjiogu LO. Beyond structured programming, An introduction to the principle of applied software metrics, structured programming, Springer- verleg, N. Y.; 1990.
[18] Henry S, Kafura D. The evaluation of software systems' structure using qualitative software metrics, Software- practice and Experience. 1984;14(6):561-573.
[19] PMBOK, A guide to the project management Body of Knowledge. Project Management Institute Standards Committee; 2002. 30. Cavano JP. Software reliability measurement: Prediction, estimation, and assessment, Journal of Systems and Software. 1984;4:269-275.
[20] Walston CE, Felix CPA. Method of programming measurement, and estimation, IBM Systems Journal. 1977;16: 54-73.
[21] Boehm BW. Software Engineering Economics. Englewood Cliffs, NJ, Prentice Hall; 1981.
[22] Khatibi V, Jawawi DNA. Software cost estimation methods: a review, Journal of Emerging Trends in Computing and Information Sciences. 2011;2(1):21-29.
[23] Putnam LH. A general empirical solution to the macro software and software sizing and estimating problem. IEEE Transaction on Software Engineering. 1978;SE4(4):345-361.
[24] Kemerer CF. An empirical validation of software code estimation models, Communications of the ACM. 2008;30(5):416-429.
[25] Premal BN, Kale KV. A brief overview of software testing metrics, International Journal of Computer Science and Engineering. 2011;1(3/1):204-211.
[26] Pressman RS. Making Software engineering happen: A Guide for instituting the technology, Prentice Hall, New Jersey; 1988.
[27] Kan SH. Metrics and models in software quality engineering, chapter 4, software quality metrics overview, Addison-Wesley professional; 2002.
[28] Basili VR, Weiss DM. A methodology for collecting valid software engineering data, IEEE Transactions on Software Engineering. 1984;SE-10:728-738.
[29] Daskalantonakis, MK. A practical view of software measurement and implementation Experience within Motorola. IEEE Transactions on Software Engineering. 1992;SE-18: 998-1010.
[30] Kuar A, Suri B, Sharma A. Software testing product metrics – A Survey, In Proceeding of national Conference in Challenges & Opportunities in Information Technology, RIMT-JET, Mandi Gobindgarti; 2007.
[31] Farooq SU. Quadri SMK, Ahmad N. Software measurements and metrics: role in effective software testing. Internal Journal of Engineering Science and Technology. 2011; 3(1):671-680.
[32] Lei S, smith MR. Evaluation of several non-paramedic bootstrap methods to estimate Conference Interval for Software Metrics, IEEE Transactions on Software Engineering. 2003;29(1):996-1004.
[33] Pandian CR. Software metrics – A guide to planning, Analysis, and Application, CRC press Company; 2004.
[34] Juristo, N. and Moreno, AM. Basic of software Engineering Experimentation, Kluwer Academic, publisher, Boston; 2003.
[35] Dumake R, Lother M, Wille C. Situation and treads in Software Measurement – A statistical Analysis of the SML Metrics Biolography, Dumke / Abran: Software Measurement and Estimation, Shaker Publisher. 2002;298-514.
[36] Dao M, Huchard M, Libourel T, Leblance H. A new approach to factorization – introducing metrics. In proceeding of the IEEE Symposium on Software Metrics, METRICS. 2002;27:236.
[37] Fenton N, Krause P, Neil M. Software measurement: Uncertainty and causal modeling, IEEE Software, July / August. 2002;116-122.
[38] Vennila G, Anitha P, Karthik R, Krishnamoorthy P. A study of evaluation information to determine the software quality assurance, International Journal of Research and Reviews in Software Engineering. 2011;1(1):1-8.
[39] Ma Y, He K, Du D, Liu J, Yan Y. A complexity metrics set for large-scale object oriented software system, In proceedings of the Sixth IEEE International Conference on Computer and Information Technology, Washington, DC, USA. 2006;189-189
[40] McCall JA, Richards PK, Walters GF. Factors in software quality, RADC TR-77-369: 1977. (Rome: Rome Air Development Center)
[41] Boehm BW, Brow JR, Lipow M, McLeod G, Merritt M. Characteristics of software quality. North Holland Publishing. Amsterdam, the Netherlands; 1978.
[42] Grady, RB. Practical software metrics for project management and process improvement, Prentice Hall; 1992.
[43] Dromey RG. Concerning the Chimera (software quality). IEEE Software. 1996;1:3343.
[44] Drown DJ, Khoshgoftaar TM, Seiya N. Evaluation any sampling and software quality model of high assurance systems, IEEE Transaction on systems, Mean and Cybernetics, Part A: Systems and Human. 2009;39(5):1097-1107.
[45] Tomar AB, Thakare VM. A systematic study of software quality models, International Journal of software engineering & application. 2011;12(4):61-70.
[46] Boehm BW, Brown JR, Lipow M. Quantitative evaluation of software quality, In Proceeding of the 2nd International Conference on Software engineering. 1976; 592605.
[47] Christensen, K., FIstos, P and Smith, CP. A perspective on the software science, IBM systems Journal. 1988;29(4):372-387.