Input Profiling for Reliability Estimation

R. A. Khan[1], K. Mustafa[2]

ABSTRACT

This paper identifies operation profile and test profile of execution as an important factor to analyze the failure behavior of software as an extension of the study by us [25]. The relationship between software reliability and (i) operational profile, and (ii) test profile has been recognized as a major issue in software engineering. Such inference is based on the premises that the identification of operational profile and test profile may lead to accurate reliability estimation. An equation to estimate reliability based on and operational profile and test profile of software is proposed. Such an estimate of software reliability by observing or evaluating failure of software would be valid for specific operational profile. It is also observed that Reliability R(t) may be defined for some specific operational profile.

1. INTRODUCTION

A series of accidents caused by unreliable software proves that the reliability of software is a matter of life and death. Software reliability is defined as the probability of failure free operations for a specified period of time in a specie environment. Failure rate of software depends critically on the environment in which it is executing. Software may frequently fail for the given input space, or in other words if software has faults, only input space will exercise that fault to affect failures. The input space causing failure will decide the software failure rate [1].

As the definition of reliability is based on the failure, which in turn depends on the nature of inputs being used by the software during execution, reliability is clearly dependent on the operational profile of software. If the operational profile of software changes dramatically, reliability of systems will need to be either recomputed or recalibrated [2]. An operational profile of software is a key factor to analyze failure behavior of software and the relationship between software reliability and operational profile has been recognized as a major issue in estimating software reliability. Accuracy of operational profile yields the greatest problem in the field of software reliability estimation.

In the development stage of a software product, its reliability can be estimated by analyzing data obtained from software tests. One of the major tasks in testing is to select the minimum set of test case sufficiently effective for revealing potential faults in a program. But the fundamental question arise- how? The test strategies are faced with the problematic question like this, since exhaustive testing is usually not tractable. It is evident that the evaluation criterion for test strategies may be used to measure the effectiveness of generated test cases [21]. Mutation analysis [22] is one of the evaluation technique that assess the quality of test cases by examining whether they can reveal certain types of faults. It is sometimes helpful to view software development as a 'root-finding' procedure where a designer submits an initial 'guess', and is iteratively test for validity until a correct solution

[1]Lecturer, Department of Computer Science, Jamia Millia Islamia, New Delhi-India
E-mail : khanraees@lycos.com

[2]Associate Professor, Department of Information Technology, Al-Hussein Bin Talal University, Jordan
E-mail :kmfarooki@yahoo.com

is converged upon. Initially [23], this principle was called 'meta-induction' and was used at a conceptual model for a new approach to software testing, called 'mutation analysis' [24].

Mutation testing is considered one of the promising testing techniques [20]. It consists of creating a sample of faulty programs from the original program under test, by injecting faults. A set of similar programs called 'mutants' is obtained after applying a single mutation operator to a single location in program. These mutants are run with an input data from a given test set. If a test set can distinguish a mutant form the original program, i.e. it produces different execution result, the mutant is said to be '*killed*'. Otherwise, the mutant is called a 'live' mutant. A mutant remain live because either it is equivalent to the original program, i.e. it is functionally identical to the original program although syntactically different, or the test set is inadequate to kill the mutant. If the mutant is 'equivalent mutant', it would always produce the same output, hence it can't be killed. Typically the equivalent mutants are distinguished approximately after testing or identified by hand. If a test set is inadequate, it can be improved by adding test cases to kill the live mutant. A test set that can kill all non-equivalent mutants is said to be 'adequate' and the 'mutation score', the adequacy of a test set, is measured as a ratio of the number of mutants killed over the total number of non-equivalent mutants.

In this paper an equation to estimate software reliability based on an operational profile and testing profile is proposed. The rest of the paper is as follows: In section 2, we discuss reliability as a key factor for the quality of software. It is further discussed that identification of failure free operation under two parameters, including time span and accurate environment under which software is executing required to estimate software reliability. Section 3 describes operational profile perspective. Section 4 contains the proposed uncertainty and reliability measuring process. In section 5, a pictorial representation is proposed to depict the interrelation of reliability, failure and nature of next input selected. Section 6 contains the established equation to correlate the reliability and operational profile. Section 7 describes testing profile perspective. Reliability has been correlated with testing profile in Section 8.

2 Reliability – A Key Factor for Quality

It is well known of major factors driving any production discipline is quality of software product has three dimensions in which product operation deals with the key factor, '-reliability'[3]. Reliability is the property referring to 'how well software meets its requirements' and also 'the probability of failure free operation for the specified period of time in a specified environment'. It is a probabilistic measure that assumes that the occurrence of failure of software is a random phenomenon. This randomness means the failure may not be predicted accurately.

ISO 9126-1991 defines six quality characteristics and one of them is reliability. IEEE standard 982.2-1988 states "A software reliability management program requires the establishment of a balanced set of user quality objectives, and identification of intermediate quality objectives that will assist in achieving the user quality objectives [4]."

$$R(t) = P(X > t) \qquad (1)$$

Where R(t) is reliability of software component for time period 't' and P(X>t) is the probability that the software component survives for the time period 't'.

The failure probability, F(t), of a system is defined as the probability that the system will fail by the time 't', that is, the life of system, X, is larger than 't':

$$F(t) = P(X<=t) \qquad (2)$$

This F(t) specifies the failure probability up to a given time 't', that changes with time. A function specified for F(t) is said to be failure distribution function.

From equation (1), We have

$$R(t) = P(X>t) \qquad (3)$$
$$= 1-P(X <= t)$$
$$= 1-F(t)$$

R(t) is a monotone nonincreasing function of time t. Therefore for t=0, R(0)=1 i.e. initially component may be assumed to be working properly. And when t is too large we have $Lim_{t->\infty}$ R(t)=0 i.e. no component may be work without failure at all. Therefore, continuing with the IEEE definition of reliability as "the ability of a system or component to perform its required functions under stated conditions for the specified period of time" following observations may be drawn.

To estimate reliability, identification of failure free operation under two parameters is required. Further direct impact of these two factors on reliability estimation may be examined.

3 Operational Profile Perspective

The environment in which software is executing affects the failure of the software. Perfectly working software may also breakdown or fail if the running environment changes. After the success of Ariane 4 rocket, the maiden flight of Ariane 5 ended up in flames while design defects in the control software were unveiled by faster horizontal drifting speed of the new rocket. It seems that the greatest problem in the field of software reliability estimation is the accuracy of operational profile.

The operational profile is defined as the probability density function (over the entire input space) that best represents how the inputs would be selected during he life time of the software [5]. The operational profile of the software reflects how will it be used in practice. It consists of a specification of classes of input and the probability of their occurrence. When an existing manual or automated system is being replaced by new software system, it would be easy to assess the probable pattern of usage of new software. It may roughly correspond to the existing usage with some allowance made for the new functionality, which is included in the new software.

The operational profile is a quantitative characterization of how a software system is used, assuming occurrence probability to software operations. It is used in the test environment to ensure that the most used operation receive the most testing [6]. Given a fixed operational profile, software failure intensity decreases and its reliability increases as faults are removed. A principal assumption for using software reliability growth models to predict field behavior is that the software will be used in any environment that is similar to that in which it is tested, thereby field operational profile does not differ from its test operational profile in practice [7].

It is very difficult to anticipate about the new and innovative system to be used as users may have different expectations, background and experience, and there is no as such historical database. The problem is further compounded by the fact that operational profile may change as system is used. When the abilities and confidence of user change as they gain experience with the system, this may use the same system in more sophisticated ways. So, it may be difficult to know the operational profile in advance.

4 Uncertainty and Reliability Measurement Process

Measuring software reliability remains a difficult problem because there is no good understanding of nature of software. There is no clear definition to 'what aspects are related to software reliability'. To find suitable way software reliability and most of the aspects related to

software reliability is very difficult. A range of metrics have been developed that may be used to specify system reliability requirements. Reliability measurement is necessary to validate whether the system meets the requirements or not, if yes then to what extent? The process of measuring the reliability of a system depicted in figure 1 includes.

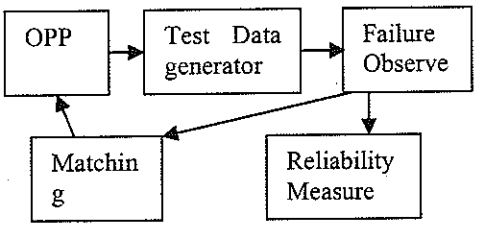

Fig. 1. Reliability Measurement Process

- Identification of operational profile (OPP).
- Collection of test data (by test data generator).
- Identification of failures observed with these tested data sets.
- Computations of software reliability after faults have been observed.

It signifies that the test data constructed reflects the operational profile. Reliability estimation may start with the identification of the number of faults observed. So, one of the principle difficulties due to uncertain operational profile, may not be an accurate reflection of real use of the system [8].

5 Characterizing Failure Behavior

There may be no general definition of failure, as it may be felt variously to testers, project managers, and end users. Different associates may have their own domain of experiences and expectations on failure of software leading to different reliability perceptions. In general, the failure behavior of software may be analyzed by considering the following factors:

(i) The number of faults in the software being evaluated.

(ii) The operational profile of the execution.

Let N_0 be the identical components under test. After time 't', $N_f(t)$ components have failed and Ns(t) components have survived.

$$N_0 = N_f(t) + Ns(t). \qquad (4)$$

Therefore the probability that the component under test have survived will be given by the following:

As $N_0 \to \infty$, $P \to R(t)$ that is, as the number of components under test increases to infinity, this probability of survival of that component reaches to reliability of that component i.e. $R(t) \approx Ns(t)/No$

Substituting the value of Ns(t), from equation 3:

$$R(t) \approx (N_0 - N_f(t))/N_0 \qquad (6)$$
$$\approx 1 - N_f(t)/N_0$$

Where N_0 is constant, $N_f(t)$ is increasing with time, means R(t) is decreasing.

The environment in which software is executing affects the failure of software. This is operational profile that captures the relative probability of different types of input being given to software during its execution. This input space is responsible to cause the fault to affect the failure. In industries such as telecommunication, an operational profile can be fairly easily gathered because of the records being maintained. Operational profiles are virtually impossible to guess [7]. It is difficult to guess how different users using different product with different features will behave.

Figure 2 depicts a pictorial representation of how reliability, failure and nature of next input selected are interrelated.

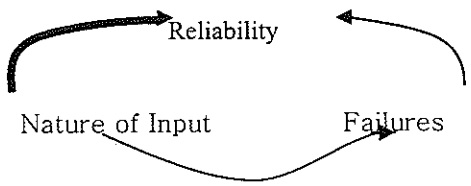

Fig. 2. Reliability Failure Cycle

Moreover, if operational profile is being changed, reliability of software needs to be recomputed according to the changed input space. Also failure of software needs to be evaluated or observed under specific operational profile to measure reliability.

6 Correlating Operational Profile to Reliability

Let F(t) be differentiable, than its first derivative, failure density function, may be defined as:

$$f(t)=d/dt(F(t)). \quad (7)$$

where f(t) is the failure density function and F(t) is the distribution function [8].

Similarly, if 'i' is the input space and 't' is the time space with i=i(t), then input space density function (IDF) may be defined as 'g(i)'.

These two pdfs may be co-related as[8]:

$$g(i)=f(t)/(di/dt)$$

di/dt is the rate of change of input space with time.

$$f(t)=g(i)di/dt. \quad (8)$$

Now probabilistic definition of reliability may be given as:

$$R(t)=1-F(t).$$

$$R(t)=1-{}_0\int^{\infty} f(t)\,dt$$

Using equation (8)

$$R(t)=1-{}_0\int^{\infty} g(i)\,(di/dt)dt \quad (9)$$
$$R(t)=1-{}_0\int^{\infty} g(i)di$$
$$\boldsymbol{R(t)=1-G_t(I)}$$

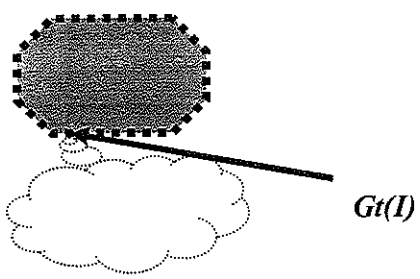

Fig. 3. Input Space Density Component

Where G(I), input space density, may be defined as the probability that the component does not survives for input space I and time 't'. Therefore reliability may be represented as –"the probability to select a set of input space, I, so that the component survives for time 't'." The input space density function may be defined as:

$$g(i)=d/dt(G_t(I))$$

i.e. the first derivative of input space density, $G_t(I)$ is said to be input space density function.

Using equation (8) and (11), it may be concluded that:

$$F(t) \propto G_t(I). \quad -(10)$$

i.e. Failure density is proportional to the input space density. Therefore to find the failure density to estimate the reliability of software, input space density needs to be specified, as failure density varied with input space.

First derivative of reliability R'(t) may be defined as:

$$R'(t)=-d/dt(F(t))$$
$$R'(t)=-f(t)$$
$$R'(t)=-g(i)\,di/dt \quad -(11)$$

Here R'(t) is defined in terms of rate of failure of component as:

$$R'(t)=-(1/N_0)N_f'(t) \quad -(12)$$

Using equation (11) and (12),

$$-(1/N_0)N_f'(t)=-g(i)di/dt$$
$$N_f'(t)/(di/dt)=N_0g(i) \quad -(13)$$

The above derivation leads to the following conclusions:

(i) Reliability is probabilistic measure where life – time may be considered.

(ii) Rate of change of input space over entire lifetime affects the failure density function.

(iii) Rate of change of input space also affects the failure rate.

(iv) Accurate identification of input space over lifetime is essential to estimate correct reliability.

7 Testing Profile Perspective

Testing is the process of executing a program with the intention of finding design errors in a given environment. Testing can only prove the incorrectness of software but not it's correctness. It has been defined that a program is correct, if it meets its specification for all valid inputs.

The reliability of a program is 1 if it is correct and 0 of it is incorrect, taken into account that a correct program will not be worn out. Unfortunately, progress in this direction is far from the stage where techniques for developing error-free software or proving correctness can be used in practice to handle today's large software systems. Obviously, a more strict development process can reduce the possibility of design errors, but it can never guarantee 100% error-free software, as long as humans engineer software [18].

Since none of today's large software systems are error-free, it does not make sense to define software reliability according to the correctness. A more practical approach is to define software reliability as the probability that no failure (an incorrect output) will occur in a specified environment, during a specified exposure period. This definition implies that software reliability is evaluated during the execution of a program in a specific environment either during testing or during operation.

The most generic way of generating inputs for testing is to use a probability density function on the input domain to guide the input generation. In other words, each input in the input domain is associated to a probability with which the input is selected for testing. The sum of probability values associated with all possible inputs is equal to 1, therefore the probability distribution forms a probability density function. This function is called the testing profile. The design of a testing profile should be guided by the operational profile of the program under testing to ensure that the heavily used operations receive the most testing. Musa [2] described a step-by-step procedure to develop a practical operational profile for testing purposes.

8 Correlating Testing Profile to Reliability

$e^{-\lambda i\, t}$ be the probability of mutant selection for the mutants 'i', with test rate λ_i.

Average test intensity of program due to mutant 'i' may be defined as:

$$\overline{\lambda_i}\,|\,t = \lambda_i\, e^{-\lambda i\, t}$$

For maximum value of $\lambda_i\,|\,t$: $d/d\lambda_i(\overline{\lambda_i}\,|\,t)=0 \Longrightarrow$

$-t\lambda_i\, e^{-\lambda i\, t} + e^{-\lambda i\, t}=0$

$(\overline{\lambda_i}\,|\,t)_{max} \le 1/et$ i.e. bounded by 1/et.

Test rate bound for all the components

$\Sigma\,(\overline{\lambda_i}\,|\,t) = \Sigma 1/et = N/et$

Let $\sigma\,(i, \lambda)$ be the Test Density Function (TDF), with test rate λ and under the given mutant 'i'.

Expected number of test after time t under mutant 'i' with test rate λ may be defined as:

$$\overline{N_t} = {}_0\int^{\infty} \sigma\,(i, \lambda)\, e^{-\lambda t}\, d\lambda \qquad (14)$$

Expected value of test rate of overall program after usage time t may be defined as:

$$\overline{\lambda_t} = {}_0\int^{\infty} \sigma\,(i, \lambda)\, \lambda e^{-\lambda t}\, d\lambda \qquad (15)$$

To bind the test and to know when to stop testing, total number of test within software can be defined as:

$$N'_t = {}_0\int^{\infty} \sigma\,(i, \lambda)\, d\lambda \qquad (16)$$

Let us define the Erlang Distribution f(t):

$$f(t) = \frac{\lambda^r t^{r-1} e^{-\lambda t}}{(r-1)!}$$

$$= \Sigma^{r-1}_{k=0} \frac{(\lambda t)^k}{k!} e^{-\lambda t} \quad t \ge 0,\ \lambda > 0,\ r = 1, 2, \ldots$$

$$\sigma\,(i, \lambda) = \frac{N.\,\lambda^r t^{r-1} e^{-\lambda t}}{(r-1)!} \qquad (17)$$

$$\Sigma^{r-1}_{k=0} \frac{(\lambda t)^k}{k!} e^{-\lambda t} = \frac{\sigma\,(i, \lambda)}{N}$$

Reliability may be defined as:

$R(t)=P(X>t)$

$=(N_t<r)$

$= \Sigma^{r-1}_{k=0} P(N_t=k)$

$$R(t)= e^{-\lambda t}\Sigma^{r-1}_{k=0}\frac{(\lambda t)^k}{k!}$$

$$\mathbf{R(t) = \sigma(i, \lambda)/N} \quad \mathbf{(18)}$$

i.e. reliability of software depends on testing profile, so in order to estimate correct reliability of software, testing profile needs to be evaluated accurately.

Equation (14) can be rewritten using equation (18):

$\overline{N_t} = {}_0\int^{\infty} R(t)\, N\, e^{-\lambda t}\, d\lambda$

$= N.\, R(t)\, {}_0\int^{\infty} e^{-\lambda t}\, d\lambda = \quad N.\, R(t)\, .\Gamma 1\, (1/t)$

$$\mathbf{\overline{N_t} = N.\, R(t).(1/t)} \quad \mathbf{(19)}$$

Similarly equation (15) can be rewritten as using equation (18) as:

$\overline{\lambda_t} = {}_0\int^{\infty} N.\, R(t)\, \lambda e^{-\lambda t}\, d\lambda$

$= N.\, R(t)\, {}_0\int^{\infty} \lambda e^{-\lambda t}\, d\lambda = N.\, R(t)\, .\Gamma 2\, (1/t^2)$

$$\mathbf{\overline{\lambda_t} = N.\, R(t).(1/t^2)} \quad \mathbf{(20)}$$

Following interpretations can be drawn from the equations 18, 19 and 20:

- The most generic way of generating inputs for testing is to use a probability density function on the input domain to guide the input generation
- In order to estimate correct reliability of software, testing profile needs to be evaluated accurately.
- Average test rate affects the reliability of software components.
- Number of tests also affects the reliability of software components.
- Accurate identification of test density over lifetime is essential to estimate correct reliability.

9 Conclusion

The environment in which software is executing affects the software quality as well as reliability. The major problem with the reliability testing and prediction is that the result is specific to a particular operational profile. An operational profile is a quantitative characterization that indicates how a system will be used. To determine an operational profile, it is required to look at use from a progressive narrowing perspective — from customer down to operation — and, at each step, it is necessary to identify how often each of the elements in that step will be used. There is no doubt the test operational profile should be similar to the operational profile encountered by the software during its use.

The operational profile plays a central role in commonly used reliability prediction models [10]. However the exact operational profile of a system is usually not known during the testing and certification process before actual system usages. In best circumstances where extensive empirical and/or theoretical information on operational profile is available, only a good approximation of operational profile can be determined. In practice, there will always be uncertainty; hence the applicability of reliability prediction method is in doubt when systems operational profile cannot be accurately determined.

In some cases it is not easy to obtain or to follow the actual operational profile. Operational usage of software can be unpredictable, unknown, or different for different users. Even if a testing profile is given, testing may not be able to follow the given profile due to the non-determinism of input generation to, say, cover a certain program path.

10 References

1. Jiantao Pan: *"Software reliability"*. Carnegie Mellon University, 18-849b Dependable Embedded Systems, 1999 http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/
2. Pankaj Jalote: *"An Integrated Approach to Software Engineering"*, Second Edition, Nrosa Publishing House, 2002
3. J.P. Cavano and J.A. McCall. : *"A Framework for the measurement of software quality"*. Proceeding ACM software quality assurance Workshop, page 133-139 ACM Nov 1978.
4. J.D. Musa, A. Iannino and K. Okumoto. *"Software Reliability Measurement, prediction application"*. McGraw Hill Book Company, 1987.
5. Software Confidence for the Digital Age : *"Research Resources"*, Cigital Labs. www.cigitallabs.com/resources/definitions/software_reliability.html
6. J.D. Musa, G. Fuoco, N. Irving, et al.: *"The operational profile. Handbook of Software Reliability Engineering"*. 1996, chap-5, pp, 167-216 McGraw Hill.
7. Lori M. Kaufman, Joanne B. Duges, Barry W. Johnson, : *"Using Statistics of the Extreme for software Reliability Analysis, IEEE"* Transaction on Reliability Vol. 48 No.3 Sep, 1999.
8. Probability Density Function http://ikpe1101.ikp.kfa-juelich.de/briefbook_data_analysis/node218.html
9. K.S. Trivedi. *"Probability & Statistical with Reliability Queuing and Computer Science Application"*. Prentice-Hall, Inc. Englewood Cliffs, NJ, 1982.
10. M. Dyer, *":Probability Models(Operational Profile Testing"*, J. Marciniak, ed. Encyclopaedia of Software Engineering, John Wiley & Sons pp 824-837, 1994.
11. IAN Sommerville, *"Software Engineering"*, 6th Edition, Pearson Education-1998.
12. Tom Adams, *"Total Variance Approach to software Reliability Estimation"*, IEEE Transaction on Software Engineering Vol-22, No-9, sep-1996.
13. Roger Cooke and Tim Bedford, *"Reliability Database in perspective"*, IEEE Transaction on Reliability Vol. 51 No-3 sep-2002.
14. V. Ramesh, David W. Twigg, R. Sandadi, Ilak C. Sharma, *"Reliability Analysis of Systems with Operation-Time Management"* IEEE Transaction on Reliability, Vol. 51 No.1 March-2002.
15. Linda Rosenberg, Ted Hammer Jack Shaw *"Software Metrics and Reliability"*, 9th International Symposium, "BEST PAPER" Award, November, 1998, Germany, http://satc.gsfc.nasa.gov/support/ISSRE_NOV98 software_metrics_and_reliability.html
16. Mazzuchi, T. A., Ozekici, S. Singpurwalla, N.D. and Soyer, R. *"Software Reliability and the Operational Profile"* Stochastic Foundations, to be submitted to Handbook of Statistics: Statistics in Industry,(C. R. Rao and R. Khattree, Eds.),2000.
17. Ozekici, S. Ozcelikyurek, S. and Altinel, K. *"Testing of Software with an Operational Profile"*, Naval Research Logistics, accepted for publication, 2000.
18. Yinong Chen *"Testing For Software Reliability"*, December 2000 Feature Article http://www.testfocus.co.za/Feature%20articles/Testing%20For%20Software%20Reliability.htm

19. J.D. Musa, *"Operational Profiles in Software Reliability Engineering"*, IEEE Software, Vol. 10, No. 2, March 1993.

20. Anna Derezinska, *"Object Oriented Mutation to Asses the Quality of Tests"*, Proceedings of the 19th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03) 2003, IEEE.

21. Philippe Chevally, *"Applying Mutation Analysis for Object Oriented Programs using a reflective approach"*, IEEE 2001 (267-270).

22. R. DeMillo, R. Lipton, and F. Sayward, *"Hints on Test Data Selection: Help for the Practicing Programmer"*, Computer, 11 (4): 34-41: April, 1978

23. R. DeMillo, R. Lipton, and F. Sayward, *"Program Mutation: A new Approach to Program Testing"*, Software Testing InfoTech, 1979.

24. R. DeMillo, *"Testing adequacy and program mutation"*, ACM 1989, 355-356.

25. Khan R A, Mustafa K, Ahson S I, *"Operational Profile- A Key factor for Reliability Estimation"*, Proceedings of 7th International Conference on Information Technology, CIT 2004, Hydrabad, pp. 347-354, 2004.

Authors' Biography :

Raees A Khan is working as a Lecturer in the Department of Computer Science, Faculty of Natural Science, Jamia Millia Islamia (A Central University) New Delhi. Dr. Khan did his Ph. D. in Computer Science (Software Engineering) from Jamia Millia Islamia, N. Delhi. His area of interest is Software Quality Assurance, Software Testing, and Software Security. Dr. Khan published a book on software quality titled "Software Quality: Concepts and Practices" with Narosa Publication

Khurram Mustafa is working as a Reader in the Department of Computer Science, Faculty of Natural Science, Jamia Millia Islamia (A Central University) New Delhi. Dr. Mustafa did his Ph. D. from IIT Delhi. His area of interest is E-Learning, Software Engineering, Multimedia Application.