CAP-2 | 825

ASAT

MILITARY TECHNICAL COLLEGE
CAIRO - EGYPT

Computer Sofware Reliability

Dr. Ramadan Moawad
EGYPTIAN AIR ACADEMY

Tarek Niazy
Defence Language Institute

ABSTRACT

The study of software engineering is advancing very much nowadays. Yet, a lot of mysteries are included in its different areas. This paper presents a brief study of the different phases of softwar development, emphasizing the software reliability considerations.

To asses any advances in this field, one has to consider their impact on reliability. Therefore, some models have been developed to provide quantitative evalvation of software reliability.

This paper presents three of the most important software reliability models, focusing on their statistical evaluation. Interesting practical results are obtained and comparison of these results is discussed.

I. INTRODUCTION.

In the last decade research and computer applications have shown a significant seriousness of the fact that "Computer Software is going to be the corner stone in developing any computer dependent systems". Users of software products keep griping about the delay of delivery of projects due to the delay in "completion" of their software products. The "completion" of a product has a specific life cycle (development phases) that the product must undergo before being released. The dissatisfaction of the customer with the "completed system" is commonly noticed, and that is due to many problems, most of which are not independent. So, one can not attack any of these problems independently, disregarding the others. An example of these problems is the uncertainty of the end product quality; i.e., we can never assure absolutely the correctness, perfection, and hence the reliability of the software product. Therefore, software developers apply different kinds of tests to the product. The difficulties of maintaining the already existing software, in addition to its other problems, finally lead to the high cost of the product.

That is the dark face of the problem. For a brighter future of the software development, to get a better software production, there must be more practically applied and easy to implement tools. Hence, researchers in this field have introduced software reliability models. These are a means of evaluation of reliability of the software products so that a satisfactory level of confidence will be developed. And so, technical problems of the software can be overcome with the help of a new generation of tools and techniques.

This paper first reviews the different techniques of software development phases and their impact on reliability; then introduces three of the most important models: Goel-Okumoto (G/O), Musa (M), and Littewood-Verral (L/V) model presenting their statistical evaluation.

II. SOFTWARE DEVELOPMENT PHASES:

The saftware development process translates a set of requirements into an operational systems element that we call"software" [15] . When we want to establish a software for a system, we do the systems analysis that leads us to transform the ideas into requirements of the system and determine the objectives. Then the design process starts. We try to get a conceptual solution of the system, and come out with the specifications which are to be coded during the coding phase.

In the process of coding, the design solution is translated into a computer processable "Code". This way, the computer software takes its actual shape and becomes executable. So the output of the coding phase is the program to be tested in the test phase. The test phase is a check to see whether the software meets its specified standards. After declaring the test is over, the operational phase starts. During this

6

phase maintenance actions take place. Riddle [11] showed the importance and purposes of the software development environments, which provide facilities supporting the production of a software. We will clarify these phases focusing on their influence on reliability.

II.1. SOFTWARE REQUIREMENTS, OBJECTIVES ANS SPECIFICATIONS:

The purpose of software requirements is to establish the needs of the user regarding a certain product. The process of establishing requirements includes the analysis of existing systems, interviewing users, performing feasibility studies, and estimating benefits. Techniques for these activities are described in texts of system analysis [12] . Mayers [9] discussed some effects of communication misunderstanding between software developers and user organization, and as a result, errors start to come out. Yet, little is known about methods of verifying correctness of requirements.

The purpose of objectives is to set the goals and the necessary alternatives for a software system. Weinburg and Schulman [13] categorized software objectives into groups that enable us to carry out meaningful tradeoffs. Objectives should be reasonable, detailed, clear, visible, and measurable. Otherwise, they become a serious source of errors. Any vague objective could mislead the programmer or just be useless.

By matching objectives with requirements, we ensure the avoiding of translation mistakes. However, we are still faced with the problem that requirements keep changing continuously, and the consequences depend on the flexibility of the software. We stress here that the cost of recovering mistakes in this phase is far less than in a later phase.

II.2. SOFTWARE DESIGN.

Design, generally speaking, means to form and shape according to a plan [9] . But, it is difficult to present a simple definition for design because of its natural creativity and ambiguous attitudes. That is why design decisions may be misleading. There are no definite procedures for software design, but we have some principles and practices which can be categorized as: Fault avoidence, fault detection, fault correction and fault tolerance [9] . We believe that the early avoidence of the in consistencies in both the system specifications and the early design stages saves a lot of later trouble and reliability problems in the system installation. It is also recommended not to start design until we completely establish the objectives. We still have no "perfect" means of illustration of design decumentation in order to highlight all design areas of concern. There are some software design methods that can be grouped as follows: (1) Data flow-oriented methods, e.g.,systematic activity modeling method. (2) Data structured-oriented methods; e.g., Jackson's method (3) Prespective methods; e.g., design

by objectives. Techniques and approaches of these methods are in [18] . Software reliability models are used to make trade-offs between those methods.

11.3. SOFTWARE CODING.

Coding is the implementation of design, taking into account the environment, language, and external interfaces (human and hardware) [18] . The code should be as simple and clear as possible. Writing a program sometimes would be easier with some languages than with others. Programming languages are: unstructured (such as FORTRAN) and structured (such as PASCAL). The structured language helps to improve the readability and understanding of the code. The mutual understanding between the language and the software presents new dimensions of usage for the computer as well as decreasing the error chances due to misunderstanding of requirements or objectives of the system (between the user and the designer or coder). The modular approach and the team approach are examples of the coding and implementation techniques. Software reliability models can be used here to contrast these approaches. The natural resistance of people, having old attitudes, to new methodologies is an important factor in coding problems.

II.4. SOFTWARE TESTING.

The job is not finished just by writing the program. There must be a means of assuring the reliability of the software. Therefore, the testing process is carried out. Mayers [17] stated some guide lines that we consider to be good testing objectives:

(1) Testing is the process of executing a program with the intention of finding errors.

(2) A good test case is when it has a high probability of finding a yet undiscovered error.

(3) A successful test is when it uncovers a yet undiscovered error.

Objective No.1. can serve as a definition for "Software Testing." Halin and Hansen [14] revealed the difference between testing, as the process of determining whether or not bugs exist in a program, and debugging, being the attempt to isolate the source of the problem and to find a solution.

Testing has a life cycle similar to that of the software development. It begins with the objectives of the test, designing test cases, writing them, excuting them, and finally examining the results. According to [9] , about 1/4 of the total cost of the software is spent over testing because it is considered the most decisive phase of the software development. Because of the high expense of testing, software reliability models can give an answer to the question" When do we stop testing, and guarantee the release of the program?".

From what we briefly viewed, we can say that the glory of the

test is not attained by checking how well the system features conform to anticipated needs, but how well the system performs when its user wants to do something the designer did not foresee.

II.5. SOFTWARE MAINTENANCE AND OPERATION PHASE:

This is the most important phase of the software life cycle. It practically consumes more than half of the total cost of a software. The top priority goals of software maintenance are the software reliability and fixing the discovered errors, depending on their nature. There are four types of maintenance activities: corrective, adaptive, perfective, and preventive maintenance [15] and [16] The software maintainer is actually a system analyst, a designer, a coder, and a tester. Consequently, he must be skilled, flexible, patient, creative, eager to work, and above all, he should have a broad background. The maintainer also should be able to tolerate criticism and have a good understanding of the user's culture and needs, so that he can overcome the user dissatisfaction with the "completed system". We should keep in mind that more maintenance actions will increase the total cost and not decrease it. Therefore, we have to work for high reliability with minimum maintenance actions.

III. SOFTWARE RELIABILITY MODELS.

The software reliability model is a mathematical probabilistic formulation developed to allow the reliability prediction of the software [9] . To construct the model we need to have assumptions. The more realistic those assumptions are, the more complicated the model becomes. Models are important to determine the end of testing and declare the release of the software product within specified intervals. We will present, very briefly, three of the most important software reliability models of G/O, M, and L/V. Details of their assumptions and mathematical formulation are in [1-6] .

The G/O model uses the Non Homogeneous Poisson Process as a stochastic model to describe the number of failures as a random variable (r.v.). The failure rate is time dependent. M model uses an exponential model to describe the time between failures as a r.v. The failure rate is stepwise constant, which varies at the instants of error detections. L/V model uses a Baysian approach to model the failure process considering the time to failure and the failure rate as two r.v.s and comes out with a final distribution called "Pareto distribution". To evaluate these models, we used the criteria developed in [19] . These criteria are: (1) Applicability: The relation between the model and the real system for a given environment. (2) Utility: The relation between the model and the user expressing the possibility of using model results in the decision making process. (3) Validity: The internal model capacity to reproduce the reality. We have operational, structural, and conceptual validity. The operational validity is divided into input and output validity. The output validity

is divided into: (a) Replicative Validity: which is the ability of the model to reproduce a late behaviour of the real system. This is appreciated by comparing the real system with the model outcomes (b) Predictive Validity: which is the ability of a model to predict the future behavior of the real system. The structural validity discusses the mathematical formulation and the estimators validity. The conceptual validity discusses the model assumptions, either being plausible or how close they conform with the actual observations.

To apply these criteria using statistical techniques, we used the data in [8]. The applicability and utility comparison of the three models are shown in tables 1 and 2 [7]. We can see that M model is more user - oriented than the other two models as it is completed by a calendar time component.

Table 1 The List Of Applicability Comparison

	L/V	M	G/O
Software life cycle phase:			
1. Design and coding	No	Yes	No.
2. Module test	Yes	Yes	Yes
3. Integration	Yes	Yes	Yes
4. Functional test	Yes	Yes	Yes
5. Operation	Yes	Yes	Yes
6. Maintenance	No	Yes	No
Reliability behaviour:			
7. Growth	Yes	Yes	Yes
8. Decay	Yes	No	Yes

Table 2 The List Of Utility Comparison

Model Outcomes	L/V	M	G/O
1. Mean number of residual errors	No	Yes	Yes
2. Distribution of the number of residual errors	No	No	Yes
3. Failure rate	Yes	Yes	Yes
4. Mean time to failure	Yes*	Yes	No
5. Reliability	Yes	Yes	Yes
6. Delay to reach a reliability goal	Yes	Yes	Yes
7. Cost to reach a reliability goal	No	Yes	No
8. Resources to reach a reliability goal	No	Yes	No

* This is true for the level of significance $\alpha > 1$

We are going to focus on the operational validity comparison.

The input data validity is governed by [8] and accepted as satisfactory.

THE OUPUT REPLICATIVE VALIDITY:

We used both the cumulative number of software failures and the distribution function of time to failure to determine the quality of reporudction (replicative validity). The goodness of fit, between the model behaviour and the real system, is appreciated by the mean absolute difference D_T for the cumulative number of software failures. We applied the Crammer-Von Misses" test (nw^2 test) to appreciate the quality of fit for the distribution function of time to failure. The results are shown in table 3.

Table 3 Results of The Replicative validity (No of Failures)

Data Set	L/V	M	G/O	Sample Size
1	0.0158 A	0.035 A	0.03525 A	136
2	0.019 A	0.040 A	0.03867 A	54
3	0.0218 A	0.077 A	0.05937 A	38
4	0.037 A	0.045 A	0.01854 A	53
6	0.1542 R	0.047 A	0.0498 A	73
17	0.030 A	0.041 A	0.04191 A	38

D_T critical =0.1 A.....accepted, R.....rejected

Table 4 Results of The Replicative validity (Distribution Function)

Data Set	L/V	M	G/O	Sample Size
1	0.1462 A	0.195847 A	0.19006 A	136
2	0.1568 A	0.132659 A	0.12041 A	54
3	0.0536 A	0.316719 A	0.27665 A	38
4	0.0524 A	0.275782 A	0.26233 A	53
6	0.133 A	1.57382 R	2.05548 R	73
17	0.0428 A	0.323661 A	0.24295 A	38

nw^2 critical = 0.46136 for α = 0.05 A.. accepted, R..rejected

For table 3 we chose the critical value to be 0.1, and the values of the content of the table that exceed the critical D_T are regjected. The values that go below the critical are accepted. Figs. 1, 4 and 7 represent the graphical illustrations of data set No.1 from [8] , as an example. They show how close the three models' performance and the real system behavior are. We used the statistical test (nw^2) and calculated the difference between the model distribution function performance and the real

distribution function of the system in the replicative validity. The results are shown in table 4. The contents of table 4 that go below the mentioned critical value are accepted and those that exceed the critical are rejected. Figs. 2, 5, and 8 are the graphical illustration of the conformity between the model and the reality (also for data set 1, and applied on the three models).

THE OUTPUT PREDICTIVE VALIDITY:

We reestimated the three models' parameters from a minimum to a maximum sample size, calculated their predictions, and compared the results with the actual observations. In this comparison we used the real cumulative distribution functions of the execution times between failures and that of the models. Two statistical tests are applied to appreciate the quality of fit between reality and the models. These tests are the "Kolmogrov-Smirnov test" (K-S)and the "Crammer-Von Misses test" (nw^2). The results of applying these tests are shown in table 5.

Table 5 Results of The Output Predictive Validity

Data Set	K - S Statistic					nw^2 Statistic		
	L/V	M	G/O	S		(S=.4613 ∝=.05)	(S=.74346 ∝=.01)	
				∝=.05	∝=.01	L/V	M	G/O
1	$.069^{A^+}$	$.2037^{R}$	$.2082^{R}$	.140	.163	$.1229^{A^+}$	1.2634^{R}	1.4388^{R}
2	$.1999^{A^+}$	$.2317^{A}$	$.2198^{A}$	.210	.250	$.1909^{A^+}$	0.3964^{A}	0.5813^{A}
3	$.1919^{A^+}$	$.2833^{A}$	$.26748^{A}$	.264	.320	.4043	$.7759^{R}$	0.5138^{A}
4	$.0889^{A^+}$	$.1602^{A^+}$	$.1052A^{+}$	.210	.250	$.0689^{A^+}$	$.3728^{A^+}$	0.1019^{A^+}
6	$.1430^{A^+}$	$.2837^{R}$	$.2033^{A^+}$	.190	.230	$.2310^{A^+}$	1.4223^{R}	$.5251^{A}$
17	$.1050^{A^+}$	$.1108^{A^+}$	$.2453^{A^+}$	.409	.486	$.1093^{A^+}$	$.0634^{A^+}$	$.2019^{A^+}$

where: S is the critical value , R means not acceptable.
A^+ means acceptable, level of significance $\alpha = 0.05$
A " " , " " " $\alpha = 0.01$

In table (5), if any of the values below L/V, M , and G/O go below the critical value for the level of significance $\alpha = 0.05$, then it is acceptable and referred to, in our table, as A^+. When those values (below L/V, M, and G/O) exceed the critical value for $\alpha = 0.05$, but do not excced the critical value for $\alpha = 0.01$, then they are also acceptable and referred to as A. But if the

value exceeds both of the mentioned levels, then it is rejected. The critical value for the K-S_2 statistic varies with the change of the system, while in the nw^2 statistic the critical value changes only with the change of the level of significance. Figs. 3, 6, and 9 are the graphical representation of the output predictive validity of the three models.

IV. CONCLUSIONS

A brief survey over the software development phases is presented to highlight their effects on the reliability of a software product. The illustrated comparison of the outcomes of the three mentioned models emphasizes the importance of the software reliability models as a good means of judgement over the product, and hence reducing the cost of producing a reliable system.

From the applicability comparison, we can see that M model is better than the other two in the design and coding phases and in the maintenance phase. Also, M model is not applicable in the reliability decay, whereas the other two are applicable. In the utility comparison, M and G/O models calculated a mean number of residual errors. G/O model got a distribution for them, while L/V model did not. M model determined the cost and the kind of resources needed to reach a reliability goal.

From the output predictive curves, and from table 5, we can say that L/V model is more accurate than the other two. In the output replicative validity, L/V model got the lowest absolute difference with the real system behavior. Hence, L/V model predictive and replicative curves are more accurate than the other two models described.

REFERENCES:

[1] J.D. Musa, "A Theory of Software Reliability and its Applications", IEEE trans. Software Engineering, Sept. (1975).

[2] Musa, "Validity of The Execution time Theory of Software Reliability", IEEE trans. reliability, Aug. (1979).

[3] J. Musa " Software Reliability Measurement- The state of the art", Reliability in electrical and electronic components and systems, North Holland Publishing Company, (1982).

[4] A.L. Goel and K. Okumoto, "Time - Dependent Error-Detection Rate Model for Software Reliability and other performence Measures", IEEE trans. reliability, Aug. (1979).

[5] B. Littewood and J.L. Verrall, "A Bayesian Reliability growth Model for Software Reliability" , in Conf. Rec. IEEE Symp. Comput. Software Reliability N.Y., Apr. 30-May. 2, (1973).

[6] B. Littewood, "Theories of Software Reliability: How good Are They and How Can They Be Improved", IEEE Trans. Rel., Sep. (1980).

[7] R. Moawad, R. Troy, "Assessment of Software Reliability Models", IEEE COMPSAC, Chicago, Nov. (1982).

[8] J.D. Musa, "Software Reliability Data", Report from Data Analysis Control of seftware (DACS), RADC, Jan. (1980).

[9] Mayers, G., "Software Reliability Principles and Practices", A Wiley Series, (1976).

[10] R.Moawad, "Software Reliability:Modeling and Model Evaluation", Ph.D.Thesis, Dec. (1981).

[11] W.E. Riddle, "Software Development Environments," IEEE COMPSAC , pp. 220-224 (1980).

[12] J.E. Bingham and G.W.P. Davies, "A Handbook of Systems Analysis", N.Y. Halsted, (1972).

[13] G.M.Weinburg and E.L. Schulman, "Goals and Performance in computer Programming, " Human Factors, 16 (1), 70-77 (1974).

[14] T.G.Hallin and R.C. Hansen, "Toward a Better Method of Software Testing", IEEE COMPSAC (1978).

[15] P.S. Pressman, "Software Engineering a Practitioner's Approach", McGraw- Hill Book Company, (1982).

[16] R.L.Glass and R.A. Noiseux, "Software Maintenance guide Book", N.J., (1981).

[17] Mayers, G., "The Art of Software Testing", Wieley, (1979)

[18] L.J. Peters, Forword by L.D. Belady, "Software Design: Methods and Techniques", N.Y., (1981).

[19] R.Moawad, "Criteria for Mathematical Model Evaluation", first conference on Operations Research and its mililitary applications, M.T.C. Cairo, EGYPT, 27-29 Nov. (1984).

Appendix "A"

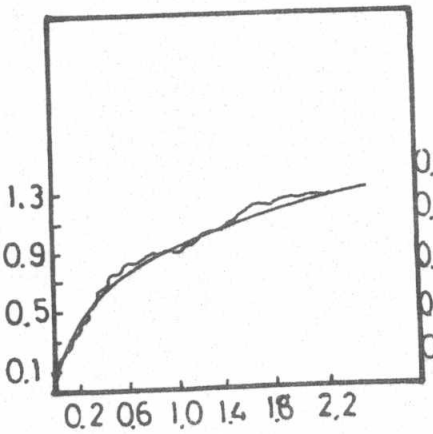

Fig. 1.

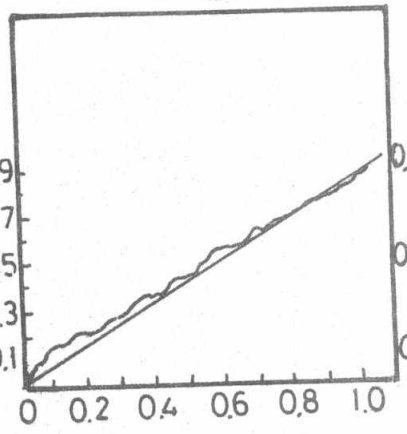

Fig. 2.

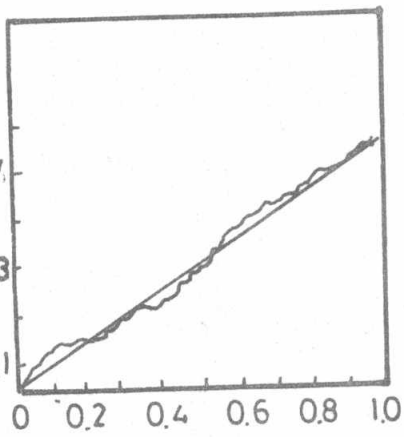

Fig. 3.

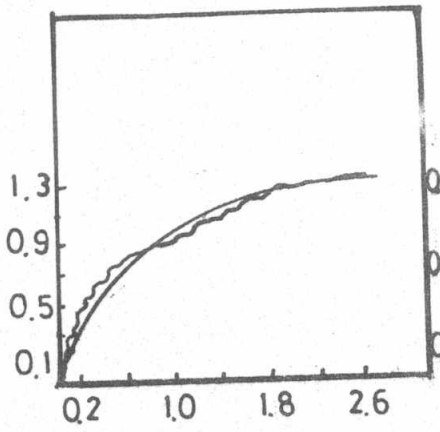

Fig. 4.

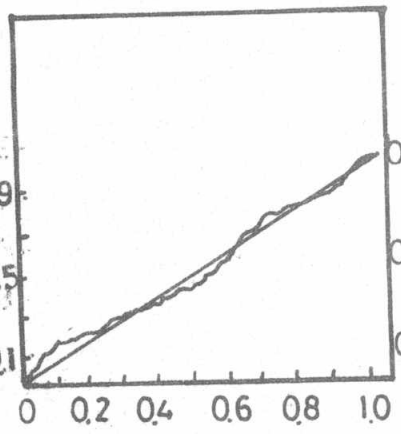

Fig. 5.

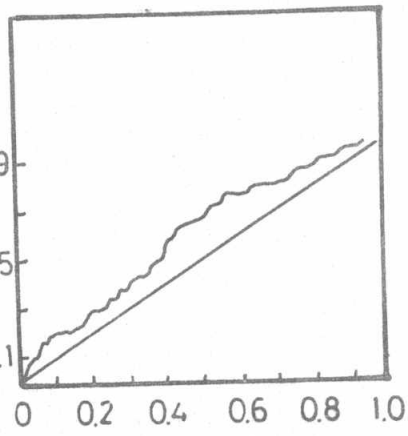

Fig. 6.

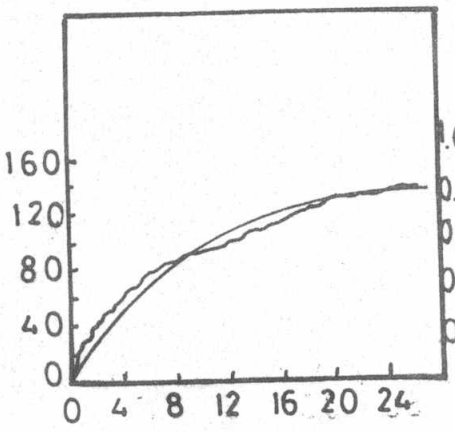

Fig. 7.

Output Replicative Validity

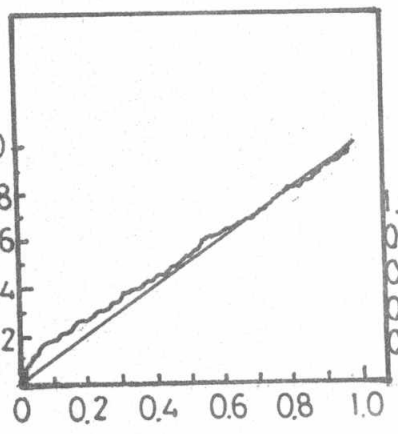

Fig. 8.

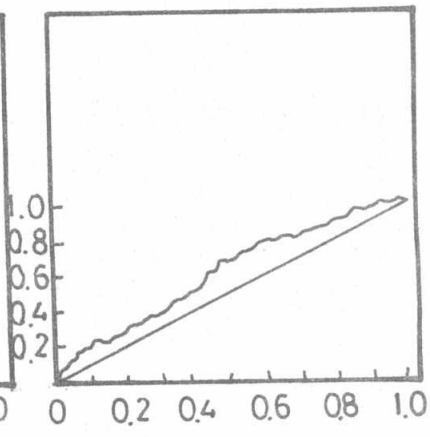

Fig. 9.

Output Predictive Validity

Figs. 1, 4, and 7 are for the no of failures function.
Figs. 2, 5 and 8 are for the distribution function.
Figs. 3, 6, and 9 are for the output predictive validity.