

Software Reliability Qualification Model

S.VELMOURUGAN^{1*}, P.DHAVACHELVAN², and R.BASKARAN³

¹ Centre for Reliability, Dr.VSI Estate, Thiruvannamiyur, Chennai -600041

² Department of Computer Science, Pondicherry University, Puducherry, India

³ School of Computer Science, Anna University, Chennai, India

(Received on November 18, 2010, revised on July 22, 2011 and February 27, 2012)

Abstract: Present software development process provides the developer, flexibility through functional programming technique to embed the third party objects into their software product in order to cut down the unnecessary duplication of coding. The existing models for estimation of software reliability are based on past failure data and whenever the software developer aims to estimate the reliability of the software product, it is compounded with the lack of past failure data of the third party objects, resulting in the loss of accuracy on the estimation process results incomplete estimation. The SRQM model proposed is used to predict the reliability of the frozen software packages based on actual usage testing. Also SRQM introduced the cell based modeling of an application. Using this cell based modeling the developer can point out which part of the application needs reliability improvement to meet the user requirements and also set criteria to qualify a product for the reliability requirements.

Keywords: Software reliability, reliability estimation, Mean time to Run (MRTF), testing, software engineering

1. Introduction

Modern functional programming technique has simplified the software development process [1]. Various computer languages supporting functional programming technique have become popular in recent years, proving that functional programming is the future software development process [2]. Moreover, the modern software development process does not strictly follow through all the phases of the software development life cycle (SDLC) because of the following reasons that, each piece of the complex software is being developed by various groups and may not be belonging to the same division or unit, hence generalization of reliability factors, assumptions *etc.*, of the development process is tedious and also, moreover in real scenario developers are under tremendous pressure to release their product in time, where in there is a chance of missing or manipulation of past failure data[3]. There is various software reliability estimation models used to predict the reliability of the application either based on the failure data collected during development or extrapolating the observed data [2]. The survey on software reliability models referred in this paper left with the conclusion that the existing models are not addressing the methodology to qualify a product for the target reliability meeting qualification criteria. As a solution to the lacunas, a model is developed that could be used to estimate and determine the qualification criteria of full-scale range of software packages using actual-usage testing.

*Corresponding author's email: velmourougan@gmail.com

Further, SRQM uses the concept of Mean Runs to Failure (MRTF) that is useful to modify the actual failure rate of the software based on test results. The meaningful estimation of the reliability of the software product, from the customer point of view will be the reliability estimation of the frozen software product based on usage testing [2],[3],[4]. This paper presents a new SRQM model to qualify a software product to the customer reliability requirements. The major advantages of such estimation are: (1) The user witnesses the actual software reliability estimation process, (2) There is no question of error feeding due to fixing of some errors, (3) Software considered for estimation being a black box facilitates easy verification and validation in the presence of user, (4) The reported number of failures will be permanent and hence the user has the liberty to alter the reliability in-terms of confidence and its associated risk [5], (5) Estimated reliability is neither predicted nor extrapolated and it is purely the true field reliability.

Notation

θ	Upper MRTF
α	Consumer risk
β	Producer risk
d	Discrimination Ratio
i	Number of installations
ATR	Average test runs in hours
MRTF	Mean runs to failure
t	Time

2. Software Reliability Qualification Model

The Software Reliability Qualification Model estimates the reliability of software product based on end user testing methodology. This model divides the total software package into small standard cells and each cell is tested using the standard real time test methods over a period of execution time, which is also estimated, by established statistical methods [6]. The tested results are analyzed and compared with the scoring tables exclusively provided by this model to arrive at a risk factor and actual number of failures reported is the real time usage risk of the software. The estimated real time risk factor is multiplied by the actual number of failures over an estimated test time will be the failure rate of the specific cell under test. Finally, the failure rate of parallel-series combination [7], [8] of these cells will end up with the total failure rate of the system which in turn provides the “Mean Runs To Failure (MRTF)” of the Software Package. Moreover, SRQM outlines the procedures to estimate the reliability, usage of scoring sheet to test the software product and to estimate the time to test.

In this model it is basically assumed that the software coding is broadly grouped and classified into standard cells [9], [10]. Any software package developed forms a combination of these standard cells. For example a Student database system software could be classified into standard cell such as fixed database, Error Handler, File based, Security, *etc*, similarly, the other type of applications would have different combination standard cells. The SRQM proposes the standard cells as fixed database, File based, Error handler, Standalone/Network based, S/W in accordance with standards, Portability, Coupling, Deadline based, Communication based, Security, Safety, Self-test and recovery, Control based, Mathematical based, and Reports/Graphs. In addition, the user has freedom to define user-defined standard cell in case the user finds a typical portion of code not being classified in the above standard list of proposed cells:

The algorithm placed in Appendix A (A.1) based on [6], the Average run time of the cells are calculated and tabulated during the estimation for MRTF in the case study.

Step 2: Testing of Standard cells

Testing of software is a science-based art, which requires lot of skills to identify the valid and invalid test cases to cover full features of the software. This model has the full flexibility that the user can establish their methodology of testing. Though, the model suggests the list of test conditions, needed to be tested in order to establish full coverage of test [9]. The test features presented here are exclusively designed as a part of present work but beyond the scope of this paper.

By using the cell/Module level Reliability Block diagram of the software as shown in Figure 1 and Figure 2, calculate the total MRTF of the software by using the well-established redundancy[8],[11],[12],[13]. Using the proposed use-based software reliability estimation model a case study is presented in this paper. MRTF of the software is estimated using the SRQM proposed to validate the accuracy of this model.

3. Case Study

The present case study on the SRQM to prove that model is capable of estimating the reliability of software developed using user point of view testing on the compiled product.

Nomenclature of the software: Spare parts- cost optimization system

Language Used: Visual Basic 6.0 as front end and Microsoft access as back end.

Description of software: Software tool to estimate the optimum number of spare parts required to be stocked in order to improve the availability of the electronic systems, cost effectively, based on scientific approach.

3.1 Inputs required

- i) Logical diagram of the software at cell level:

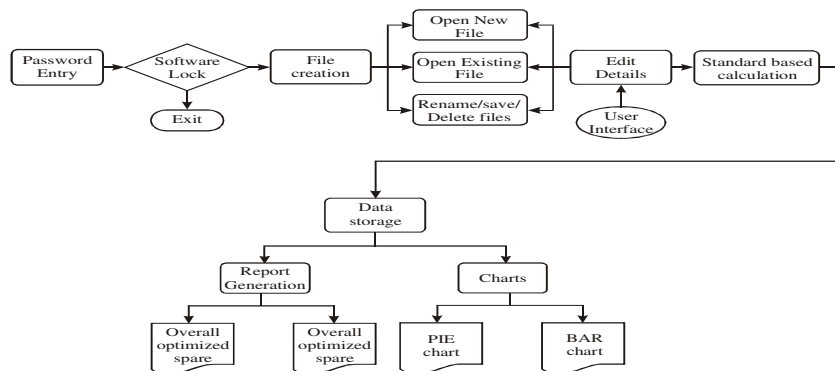


Figure 1: Functional diagram of Spare Software

The logical/functional diagram shown in Figure 1 is extracted from the software requirement specification developed based on functional requirements and customer expectations.

- ii) Reliability block diagram of the software:

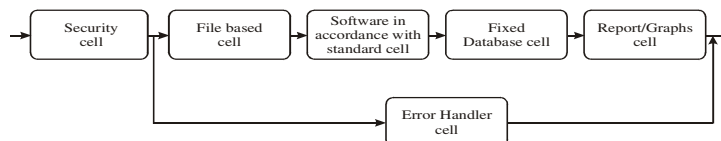


Figure 2: Reliability block diagram of Spare Software

- iii) Expected Overall MRTF of the total software: 250 Hours
- iv) Developer's Risk = 10%; v) User's Risk = 10%; vi) Discrimination Ratio = 1.5
- vii) Average Test Run Time in Hours (Estimated time to completely run the test cases for a cell as per the Test Score Sheets provided) = 48 Hrs
- viii) Expected MRTF for each cell
 - Security Cell : 750 MRTF
 - File Based Cell : 750 MRTF
 - S/W in accordance with standard : 750 MRTF
 - Fixed Database : 1000 MRTF
 - Report/Analysis : 1500 MRTF
 - Error Handler : 1200 MRTF
- ix) Average Test Run Time in Hours (Estimated time to completely run the test cases for a cell as per the Test Score Sheets provided) = 48 Hours (applicable for all cells)
- x) Functional requirement of the software (either cell level or module level)
As per the Software Requirement Specification of the Spare parts cost Optimization Software
- xi) Number of samples (Software installed in number of locations to independently run the test cases): 15 Nos

3.2 Estimation of NTR, MRTF and Reliability

The NTR and MRTF are generated based on the time for testing of each cell is estimated and the sample calculation of NTR for a security cell of SPARE package using following data; $\theta = 750$; $\alpha = 0.1$; $\beta = 0.1$; $d = 1.5$; $i = 15$; $ATR = 48$; $\theta_0 = 750 \times 48 = 36000$; $\theta_1 = (750/1.5) \times 48 = 24000$; $A = 7.5$ using (1); $B = 0.111$ using (2) $a = -5.427$ using (3); $c = 0.111$ using (4); $b = 0.00003$ using (5) and estimated NTR = 54 Runs. NTR is actual number of times the test case is repeated for a specific cell.

Table 1: The Estimated Number of Runs and Minimum No of Samples

Name of the Cell	Expected MRTF	No of Test Benches	Number of Test Runs Estimated
Security Cell	750	15	54 Runs
File Based Cell	750	15	54 Runs
S/W in accordance with standard	750	15	54 Runs
Fixed Database	1000	15	70 Runs
Report/Analysis	1500	15	109 Runs
Error Handler	1200	15	87 Runs

Similar exercise is carried out for all the selected cells based on the algorithm as per Appendix-A has estimated and the Table 1 lists all the cells in the spare parts cost optimizations system software and their corresponding expected MRTF, number of test benches and the minimum number of test runs estimated.

3.3 Testing of standard cells:

The sample of assessment sheets for security based cell as shown in Table 2.

Table 2: Assessment Sheet for Security based Cell

Sl. no.	Test Description	Assessment			Score
		% Test	Not applicable	No. of Failures (N)	
1	Availability test	1		4	
2	Confidentiality test	1		1	
3	Data Integrity Test	1		2	
4	Integration Test	0.8		1	
5	Usability Test	0.7			
Total Score		4.5	0	8	4.5

The sample assessment sheet for security cell is presented in table 2, similar tables are developed and accordingly the total score for each cell is calculated. For the case study presented the following assessment sheets similar to Table 2 are developed namely; security cell, file based cell, software in accordance with standard cell, fixed database cell, report/graphics cell, error handler cell *etc.*,

The percentage of score for each cell is calculated after testing the cell for NTR runs based on the following.

Actual Number of Failures reported (N) = 8 Nos

Test factor (W) = 1.25 chosen from Table A1 listed in Appendix A.

$$\text{Percentage of Score (SP)} = 1 + \left(\frac{\text{Maximum score} - \text{actual score}}{\text{Maximum score}} \right) \times \text{Test factor}$$

$$SP = 1 + \left(\frac{5 - 4.5}{5} \right) \times 1.25 = 1.375$$

$$\text{Total No. of failures (r)} = \text{No. of failures reported} \times \text{Percentage of Score (N} \times \text{SP)}$$

$$\text{Total No. of failures (r)} = 8 \times 1.375 = 11$$

Table 3: Cells and respective actual number of failures (N)

Name of the Cell	No. of Test Runs Estimated	Expected MRTF	Actual No. of Failures (N)
1.Security Cell	54 Runs	750 Runs	8
2.File Based Cell	54 Runs	750 Runs	9
3.S/W in accordance with standard	54 Runs	750 Runs	12
4.Fixed Database	70 Runs	1000 Runs	4
5.Report/Analysis	109 Runs	1500 Runs	5
Error Handler	87 Runs *(53)	1200 Runs	15+

Each cell is tested for number of test runs (NTR) estimated as per the test assessment sheets and actual number of failures (N) for all the cells are calculated and listed as shown in Table 3.

3.4 Calculation of Failure Rate of the Individual Cell

Repeat the assessment test one by one to test all the cells of the software under assessment and fill the content as shown in Table 4.

Table 4: Estimated Failure of the Cells

Name of the Cell	Runs Estimated	MRTF (Runs)	Failure Status				
			N	SP	W	r	Failures/ Test runs
Security Cell	54	750	8	1.1	1.25	11	0.0013
File Based Cell	54	750	7	1.4	1.25	12.25	0.0013
S/W with standard	54	750	12	1.0	1.25	15.0	0.0013
Fixed Database	70	1000	4	1.25	1.25	6.25	0.001
Report/Analysis	109	1500	5	1.5	1.25	9.37	0.0006
Error Handler	87*(53)	1200	15+	1.0	1.25	18.7	0.0188

From the Table 4 it is evident that only the error handler cell does not meet the desired MRTF. Hence, it is suggested that the MRTF can be achieved by modifying the source code of the Error handler cell (fixing of Bugs), or otherwise terminate the test runs at the 15th failure occurrence and determine the point estimate of failure Rate as follows [11],[14],[17];

$$\text{Failure rate (Error Handler)} = \frac{\text{Actual No. of failures}}{(\text{Actual test run at 15th failure occurrence}) \times \text{No. of installations}}$$

3.5 MRTF of the Software Package

The reliability equation given below is derived based on Figure 2;

$$R(\text{total Software}) = R_{\text{Security Cell}} \cdot ((R_{\text{File based cell}} \cdot R_{\text{Software in accordance with Std.}} \cdot R_{\text{Fixed Database}} \cdot R_{\text{Report/Graphs Cell}}) \parallel (R_{\text{Error Handler (EH)}}))$$

ARM1 = λ file based cell + λ software in accordance with std. cell + λ fixed database cell + λ reprint/graphs cell = λ_1 and ARM2 = λ error handler cell = λ_2

$$\text{MRTF}_{\text{ARM1}} = 4205.1282, \text{MRTF}_{\text{ARM2}} = 0.0188 \text{ and } \lambda_{\text{Security Cell}} = \mathbf{0.0013}$$

$$\text{MRTF}_{\text{ARM1} + \text{ARM2}} = (1/\lambda_1 + 1/\lambda_2) - 1/(\lambda_1 * \lambda_2) = 53.179089 \text{ hours}$$

$$\text{Total Failures per run: } 0.004089$$

$$\text{Estimated MRTF (49.7403954 Hours)} < \text{Expected MRTF (250 hours)}$$

Qualification Result: Redo-Application to meet criteria

4. Conclusion

The SRQM developed in the present work successfully overcomes most the drawbacks that are faced by the other models [17],[18]. The following are the enhancement feature over the other models. It does not require complete history of software under estimation (data throughout the SDLC) since the failure rate is assessed based on real time test. There are no derived constants that are sensitive in deriving the results. This model can be iteratively applied in order to detect and correct the faults in the software system till it satisfies the user requirements. Testing during the assessment does not assume that the software is tested in a manner similar to the anticipated operational usage. It is mostly tested for both valid and invalid operational profiles and hence it can eliminate most of critical errors. It is independent of the Source Line of codes (SLOC), Language used, and operating system. Easy to adapt and alter the reliability assessment procedure even if the software needs to be reconfigured for its requirement. End user can witness the estimation process. Accuracy of the result is purely proportional to effort of testing but not based on

any assumptions and flexibility to estimate the assessment duration (Estimated time for testing). The case study present here has truly demonstrated against the end user reliability requirement. From the above case study it was concluded that all the cells in the software could withstand the user reliability requirement except the error handler cell. The maximum attainable MRTF is 47 hours continuous running of the product. In other words the product will exhibit a failure with an average interval of 47 hours. Since the SRQM model proposed is with the goal of qualifying the software product to the desired quality. If the product is developed with the target MRTF of lesser than 47 hours then the product is fit for qualification else the product is sent back to the developer to improve the MRTF to the target set. In the case study presented in this paper the MRTF was aimed with the range from 750 to 1500 with an overall average MRTF of 250 hours. Except the error handler cell, other cells have qualified for the target MRTF; hence the feedback on the modification of the error handler cell is presented to the developer to modify the product to meet the MRTF requirement based on the user expectations. Similarly the iterative estimations of MRTF are conducted after every cycle of software modification till the target MRTF is achieved. However it is concluded that the SRQM model helps to access the qualification criteria for a product to demonstrate its reliability capabilities. However this model is developed with the assumption that the frozen software product will exhibit the exponential distribution similar to hardware and if further required it can be generalized with the suitable distribution model for further analysis.

Acknowledgements: Thanks are due to the anonymous referees who helped improve the presentation.

References

- [1] Angus J.E., Bowen J.B., and Vandenberg S.J. *Reliability Model Demonstration Study*. Rome Air Development Centre, Technical Report, Rome, New York, RADC-TR-83-207. 1983.
- [2] Musa J.D., Iannino A. and Okumoto. *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, New York, 1987.
- [3] Farr W.H. *A survey of software reliability modeling and estimation*. Naval surface weapons center, NSWC-TR-82-171, 1983.
- [4] Goel A.L. *Software reliability models: Assumptions, limitations and applicability*. IEEE transactions on software Engineering, 1985; 11(12): 1411-1423.
- [5] Bev Littlewood and John L. Verall. *A Bayesian reliability growth model for computer software*. Applied statistics, 1973; 22(3): 332-346.
- [6] *Handbook for Reliability Test Methods, Plans and Environments for Engineering, Development Qualification and Production*. Department of Defence, USA. MIL-HDBK-781A.1996.
- [7] Jones and Capers. *Programming Productivity*. McGraw Hill, New York, 1986.
- [8] Leblanc, S.P and Roman P.A. *Reliability estimation of hierarchical software systems*. Proceedings on Reliability and Maintainability symposium, Jan 28-31,2002; 249-253.
- [9] Beizer, Boris. *Software testing techniques*. Vannostrand Revihold Company, New York, 1983.
- [10] Mary A Hartz, EllenWalker and David Mahar. *Introduction to software reliability: A state of the art review*, Reliability Analysis Centre, NY, 1996.
- [11] Zygmunt Jelinski and Paul B. Morando. *Software reliability research, Statistical computer performance evaluation*. Academic press, New York, 1972.

- [12] Kung, D. C., Hsia, P., and Gao, J. *Testing Object-Oriented Software*. IEEE Computer Society Press, Los Alamitos, CA, 1998.
- [13] Lyu, Michael R. Editor, *Handbook of software reliability engineering*, IEEE computer society press, McGraw hill, 1996.
- [14] Musa J.D., Iannino A. and Okumoto. *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, New York, 1987.
- [15] Shooman, Martin L. *Software Engineering*. McGraw Hill, USA, 1983.
- [16] Lu P. and Tia J. *Applying software reliability engineering in large-scale software development*. In proc, 3rd International conference on software quality, CW3293, Lake Tahoe California/Nevada USA, Oct 4-6,1993:323-330.
- [17] Venkatesan.S, Dhavachelvan.P and Chellapan.C, *Performance analysis of mobile agent failure recovery in e-service applications*. International Journal of Computer Standards and Interfaces, 2005; 32(2): 38-43.
- [18] Dhavachelvan.P, G.V.Uma and V.S.K.Venkatachalapathy, *A New Approach in Development of Distributed Framework for Automated Software Testing Using Agents*. International Journal on Knowledge –Based Systems, 2006; 19(4): 235-247.

S. Velmourougan is presently working as Scientist at STQCIT, Chennai, STQC Directorate, Ministry of Information and Communication Technology, Govt. of India, and India. He has obtained his B.E. in Electronics and Communication Engineering and M.S in Software Reliability from Anna University, Chennai, India. He is Certified ISMS Lead Auditor and audited various organizations for the ISMS requirements. Certified ethical hacker (CEH), Certifies information security professional (CISP), Certified Reliability Professional (CRP) and Certified Software Test Manager (CSTM), He has tested various e-governance projects for the security requirements. He is involved in deriving and defining security requirement specification for the e-governance projects undertaken by government of India. He has developed various reliability engineering tools to assess the reliability of software and hardware.

P. Dhavachelvan is working as the Professor and Head of Department of Computer Science, Pondicherry University, India. He has obtained his M.E. and Ph.D. in the field of Computer Science and Engineering in Anna University, Chennai, India. He is having around a decade of experience as an academician and his research areas include Software Engineering and Standards, Software Agents and Distributed Systems. He has published around 50 research papers in National and International Journals and Conferences. He is heading two research groups working on to develop the standards for Attributes Specific SDLC Models and Evaluation and Business Intelligent Logic Based Management and Evaluation of Web Services.

R. Baskaran is working as the Assistant professor in Department of computer science, Anna University, Chennai. He has obtained his M.E. and Ph.D. in the field of Computer Science and Engineering in Anna University, Chennai, India. He is having around a decade of experience as an academician and his research areas include Multimedia and principles, Software quality engineering, Software Agents and Distributed networking. He has published around 50 research papers in National and International Journals and Conferences. He is the member of various forums. He is the editor and the reviewer in various journals. He is guiding research scholars working in area of software standards for Attributes Specific SDLC Models and Evaluation and Metric Based Efficient Traffic Management and A Multi-object Image Retrieval systems.

Appendix A

A.1 Reliability Estimation Procedure

The following are the steps to estimate the reliability of the software, the concept on the calculation of ATR is derived using [6]

Set the Expected MRTF

- i. Divide the software into standard heterogeneous cells
- ii. Set the individual Expected MRTF for all the cells
- iii. Defined the functional requirement of the software (cell or module level)
- iv. Estimate the time for testing each cell or the combination of cells using the steps indicated below.
- v. Test the individual cell or the combination of cells for the requirements. Estimate the true failure rate and MRTF of each cell or combination of cells
- vi. Draw a reliability block diagram at cell level
- vii. Estimate the total MRTF and Reliability

Inputs required

- i. Logical diagram of the software at cell level
- ii. Developer's Risk in percentage or Confidence
- iii. User's Risk in percentage or confidence
- iv. Average Test Run Time in Hours (Estimated time to completely run the test cases for a cell as per the Test Score Sheets provided)
- v. Expected MRTF of the total software
- vi. Expected MRTF for each cell
- vii. Functional requirement of the software (either cell level or module level)
- viii. Reliability block diagram of the software
- ix. Number of samples (Software installed in number of locations to independently run the test cases)
- x. Estimation of MRTF and reliability of the software

Step1: Estimation of number of test runs (NTR)

The number of test runs (NTR) for testing the individual cell is calculated based on the following algorithm derived from the MIL-HDBK 781

Input: {

θ : Upper MRTF, α : Consumer risk, β : Producer risk, d : Discrimination Ratio,
i: Number of installations, ATR: Average test runs in hours }

Calculate {

$$\begin{aligned}\theta_0 &= \theta_0 \times \text{ATR} \\ \theta_1 &= (\theta_0/d) \times \text{ATR}\end{aligned}$$

$$A = \frac{(1-\beta)(d+1)}{2\alpha d} \quad (1)$$

$$B = \frac{\beta}{(1-\alpha)} \quad (2)$$

$$a = \frac{\ln B}{\ln \left(\frac{\theta_0}{\theta_1} \right)} \quad (3)$$

$$c = \frac{\ln A}{\ln \left(\frac{\theta_0}{\theta_1} \right)} \quad (4)$$

$$b = \frac{\left(\frac{1}{\theta_1} - \frac{1}{\theta_0} \right)}{\ln \left(\frac{\theta_0}{\theta_1} \right)} \quad (5)$$

}

Estimate NTR {

```
NTR = 0
n = 0
DO UNTIL (a+bt < NTR)
{ n= n++
  t = n x ATR x i }
LOOP
NTR = n
}
End :
```

Table A1: Tester confidence factor

Slno.	Description	Factor
1	Tester from a CMM level 5 Company	1
2	Certified test engineer	1.25
3	Experienced Test Engineer	1.5
4	Test Engineer	2