Software Reliability Estimation Through Black Box and White Box Testing at Prototype Level

K.Krishna Mohan, A.K.Verma and A. Srividya
Reliability Engineering,
Indian Institute of Technology, Bombay,
kkm@ee.iitb.ac.in

Abstract
Software reliability refers to the probability of failure-free operation of a system. It is related to many aspects of software, including the testing process. Directly estimating software reliability by quantifying its related factors can be difficult. Testing is an effective sampling method to measure software reliability. Guided by the operational profile, software testing (usually black-box testing) can be used to obtain failure data, and an estimation model can be further used to analyze the data to estimate the present reliability and predict future reliability. White box testing is based on inter-component interactions which deal with probabilistic software behavior. It uses an internal perspective of the system to design test cases based on internal structure at requirements and design phases. This paper has been applied for evolution of effective reliability quantification analysis at prototype level of a financial application case study with both failure data test data of software Development Life cycle (SDLC) phases captured from defect consolidation table in the form orthogonal defect classification as well functional requirements at requirement and design phases captured through software architectural modeling paradigms.

Keywords- Black-Box, White-Box, Petri nets, orthogonal defect classification (ODC), Software architecture, Reliability estimation.

I. INTRODUCTION

Black Box testing refers to the technique of testing a system with no knowledge of the internals of the system. Black Box testers do not have access to the source code and are oblivious of the system architecture. A Black Box tester typically interacts with a system through a user interface by providing inputs and examining outputs without knowing where and how the inputs were operated upon. In Black Box testing, target software is exercised over a range of inputs and the outputs are observed for correctness.

In black-box testing, which verifies software through input/output, result of output defect is considered as software defect. The following terms used in the analysis of Black Box testing through defects.

White Box Testing refers to the technique of testing a system with knowledge of the internals of the system. White Box testers have access to the source code and are aware of the system architecture. A White Box tester typically analyzes source code, derives test cases from knowledge about the source code, and finally targets specific code paths to achieve a certain level of code coverage. A White Box tester with access to details about both operations can readily craft efficient test cases that exercise boundary conditions.

To be able to estimate operational reliability, testing must be done in accordance with the operational profile. A profile is the set of disjoint actions, operations that a program may perform, and their probabilities of occurrence. The probabilities that occur in actual operation specify the operational profile. Sometimes when a program can be used in very different environments, the operational profile for each environment may be different. Obtaining an operational profile requires dividing the input space into sufficiently small leaf partitions, and then estimating the probabilities associated with each leaf partition. A subspace with high probability may need to be further divided into smaller subspaces.

II. RELIABILITY ANALYSIS THROUGH BLACK BOX AND WHITE BOX

Software reliability refers to the probability of failure-free operation of a system. It is related to many aspects of software, including the testing process. Directly estimating software reliability by quantifying its related factors can be difficult. Testing is an effective sampling method to measure software reliability. Guided by the operational profile, software testing (usually black-box testing) can be used to obtain failure data, and an estimation model can be further used to analyze the data to

978-1-4244-8343-3/10/$26.00 ©2010 IEEE

estimate the present reliability and predict future reliability. Therefore, based on the estimation, the developers can decide whether to release the software, and the users can decide whether to adopt and use the software. Risk of using software can also be assessed based on reliability information. [1] advocates that the primary goal of testing should be to measure the dependability of tested software.

Globalization of software development has expanded rapidly in recent years and has brought a wake of changes that impact application development projects . The adoption of a new process for delivery excellence within an organization is critical to meet the time-to-market conditions and is a significant undertaking. It requires careful customization to match the organization culture, accommodate any existing procedures, and obtain buy-in among the key stakeholders and users of the process. While non functional requirements play a pivotal role in the early software reliability prediction, prototype level studies involving architectural design specifications (white box approach) must take into account the overall reliability estimation done with the intent of appropriate integration of non functional and functional features.

III. Petri Nets Analysis Of Reliability Estimation

Black box testing is based on non-functional requirements for early quantitative analysis for the reliability estimation of the application development based on the output results of the prototype development has been done using Petri net[2].

Petri Nets are found to be powerful in modeling performance and dependability of computer and communications systems. Formally, a Petri net (PN) is a 5 tuple [3].

PN = (P, T, A, M, μ0) where

P is a finite of places (drawn as a circle)

T is a finite set of transitions (drawn as bars)

A is a set of arcs connecting.

M is a multiplicity associated with the arcs in A

The Generalized Stochastic Petri Nets model for quantitative reliability prediction for process oriented development is depicted in Fig. 1.

μ0 is the marking that denotes the number of tokens for each place in c Execution of a Petri Net is controlled by the multiplicity associated with the arc and distribution of tokens in the Petri net. By changing distribution of tokens in places, which may reflect the occurrence of events or execution of operations, Petri net executes by firing transitions. The Simulation for the quantitative analysis of software reliability using Petri Nets is done by using TIME NET 4.0. Among all the phases in every module of the application mentioned earlier, parallelism exists.

Here we are focusing on the financial module whose reliability is found out using Petri Nets for each of its cycles. Table1 is formed based on the practical data available for the financial module. The total defects found and the total defects repaired are entirely random processes.

Assuming underlying exponential distribution:

Failure rate = total no of defects found/total time spent on review/testing

Repair rate = total no of defects repair/total time spent for rework

MTTF = 1/ Failure rate; MTTR = 1/ Repair rate

Notations: The P0, P1, T0 and T1 are the Requirements up state, down state, failure rate and repair rate, respectively. TheP2, P3, T2 and T3 are the Design up state, down state, failure rate and repair rate, respectively. The P4, P5, T4 and T5 are the Coding up state, down state, failure rate and repair rate, respectively. The P6, P7, T6 and T7 are the Unit Testing up state, down state, failure rate and repair rate, respectively. The P8, P9, T8 and T9 are the IST Support up state, down state, failure rate and repair rate, respectively.T10 to T16 are immediate transitions. P10 and P11 are the EFT module up and down states respectively.

The present module is a Reparability model of Generalized Stochastic Petri Nets.

This model is applicable for each phase having independent repair facilities only.

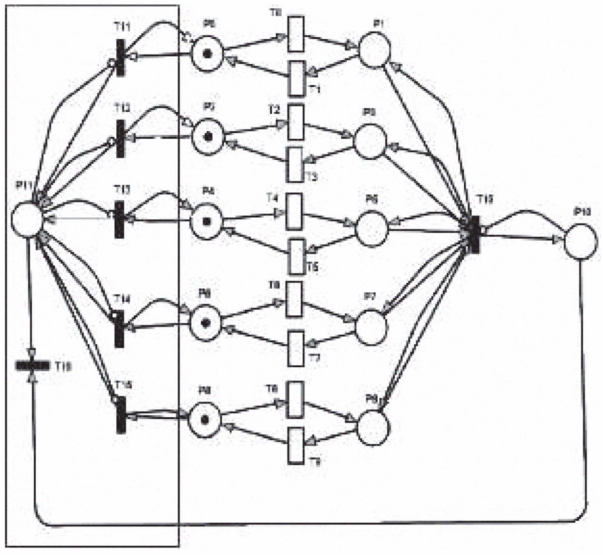

Figure 1. Generalized stochastic petri nets model

IV. White Box Modeling – Architecture Based Reliability Evaluation

Modeling Markov based software reliability estimations at the architectural level predicates the means to realize the desirability of improvement in early software life cycle. Seminal papers by Wang et al. [4 -7] in the area of architecture-based software reliability estimation have paved the way for advancements in this field, with a clear conceptual rigour. It was reported in them how while some reliability modeling approaches rely on the black-box approach without considering internal system structures, most of them also adopt the white-box approach focusing on granularity of the architecture. Each methodology has its distinct advantages and limitations. We propose a hybrid combination approach based on weightage factors to integrate the results of these reliability studies for a more meaningful quantification. Software architecture is a high level abstraction which describes the components, their topologies and interactions. The four commonly used architectural styles [4 - 7] are: batch sequential style, parallel/pipe-filter style, fault tolerant style and call-and-return style. For the illustrative study employed in this paper, we consider a batch-sequential style where components are executed in a sequential manner as in bank transactions.

The problem space, challenges and strategies in architecture-based software reliability estimation were explained in a position paper by Krka et al. [8]. Thirteen ingredients required in the reliability analysis of architectural models were identified that aid in the thorough capture of the pertinent factors for pragmatic consideration. Either through functionally similar and predecessor systems or extensive usage of operational profiles or test runs, one can collate data for parametrizing individual software component reliabilities.

Xiaguang et al. [9] presented a general model for component-based software reliability, enabling reliability estimation for software system constituted by components with different forms of reliability specification. *".. a component can have different reliabilities under different contexts or with different duration. Therefore, a mapping, rather than a single datum, should be provided for the description of component reliability."* [9]

Musa [10] explained in greater detail the significance of developing operational profiles in software reliability engineering. The usage profile, also known as the operational profile, describes the utilization of a software system of users in terms of user-initiated events and the probability of occurrence of these events. An authentic operation profile, when captured is a direct pointer to optimal allocation of development resources to functions on the basis of usage.

Hamlet et al. [11] developed a microscopic theory that describes in principle how component developers can make measurements that are later used by system designers to calculate system reliability without implementation and test. The basis of this foundational theory involves the concepts of profile mappings and component sub-domains.

Component-based reliability model developed by Cheung [12] is considered for practical application in this work. State transition diagrams are then evolved. A state represents the execution of a single component and the Markovian transition probability from one state to another is obtained from the operation profile of the system. In the batch sequential architectural style [4], a transition takes place from one state to the next upon the execution of a component, after which the control of the system is released to the next component. Ref. [6] summarizes the building of state transition matrix for batch sequential style. Assuming that k components constitute the architecture, there will be k states in the Markov model. The entries of the transition matrix M are obtained as follows [6]:

$$\begin{cases} M(i,j) = R_i P_{ij}, \text{ if } S_i \text{ can reach } S_j \\ M(i,j) = 0, \text{ if } S_i \text{ cannot reach } S_j \end{cases}, \quad for\ 1 \le i, j \le k$$

M(i,j) is the probability of transitioning from state Si to state Sj.

From the Cheung's model [12], the overall system reliability can be computed as follows:

$$R = (-1)^{k+1} \frac{|E|}{|I - M|} * R_k$$

where I is the identitiy matrix of size k x k. |I-M| is the determinant of matrix (I-M) and |E| is the determinant of the remaining matrix excluding the n^{th} row and first column of the matrix (I-M).

The above procedure is generic for evaluating any architectural style's reliability, the only difference being the entries of the transition matrix varying from style to style. [4, 6] give a detailed description of the construction of these matrices for various architectural styles. For the sake of simplicity of illustration of our proposed methodology, we take into account only the batch sequential style. Architectures are heterogeneous if they involve a combination of these various styles in entirety.

An exhaustive mathematical description of the software component transition probability diagram as postulated by Xiaguang *et al.* [9] is reproduced below:

"A component probability transition diagram G is a 11-tuple <C, T, h, S, E, M, P, R, D, g, f>, where C is the set of components, T is the set of transitions, h:T→ C ×C is a function from T to C×C, S⊆C is the set of initial components, E⊆C is the set of terminal components, M is the set of usage models, P is the domain of probability, R is the domain of reliability, D is domain of durations, g:T→ M×P×D is a function from T to M×P×D, f:C→ (M→ (D→ R)) is a function from C to (M→ (D→ R)). Usually, P is {x| x∈ R and 0≤ x ≤1}, D is {x|x ∈ R and x ≥ 0}."

V. Proposed Methodology:

Let us say, the reliability evaluated on the basis of black box approach is R_B. Let the reliability evaluated for the architecture using the white box approach as outlined in the previous section be R_W. Let W_B and W_W be the weightage factors to be assigned to the significance attached to the black box and the white box approaches of software reliability evaluation, respectively, such that $W_B + W_W = 1$. Then, the overall reliability is given as: $W_B R_B + W_W R_W$.

VI. Illustrative Case Study

The case study application discussed about Electronic fund transfer (EFT) module of a financial application. The software reliability prediction using Petri Nets and experimental testing has been carried only on the EFT module over three cycles/builds with simulations using Petri Nets. The results obtained from early software reliability prediction using Petrinets have been considered prior to the actual implementation of the application development. Table II provides the total defects implemented from the prototype and the simulation results from Petrinets. Ref. [13] elaborates on the of simulation results, whose procedural outline is the basis for the analysis of the case study of the EFT application considered in this paper.

TABLE II. Simulation Results

EFT Module	Total Defects	%Reliability	%Unreliability
Cycle1	111	84.12711	15.87289
Cycle2	44	88.45584	11.54416
Cycle3	17	92.04223	7.957777

The usage profile has been compiled for EFT of a nationalized bank. A brief explanation of the procedure followed in a successful bank transaction can be understood with the help of the sequence diagram shown in Fig. 2.

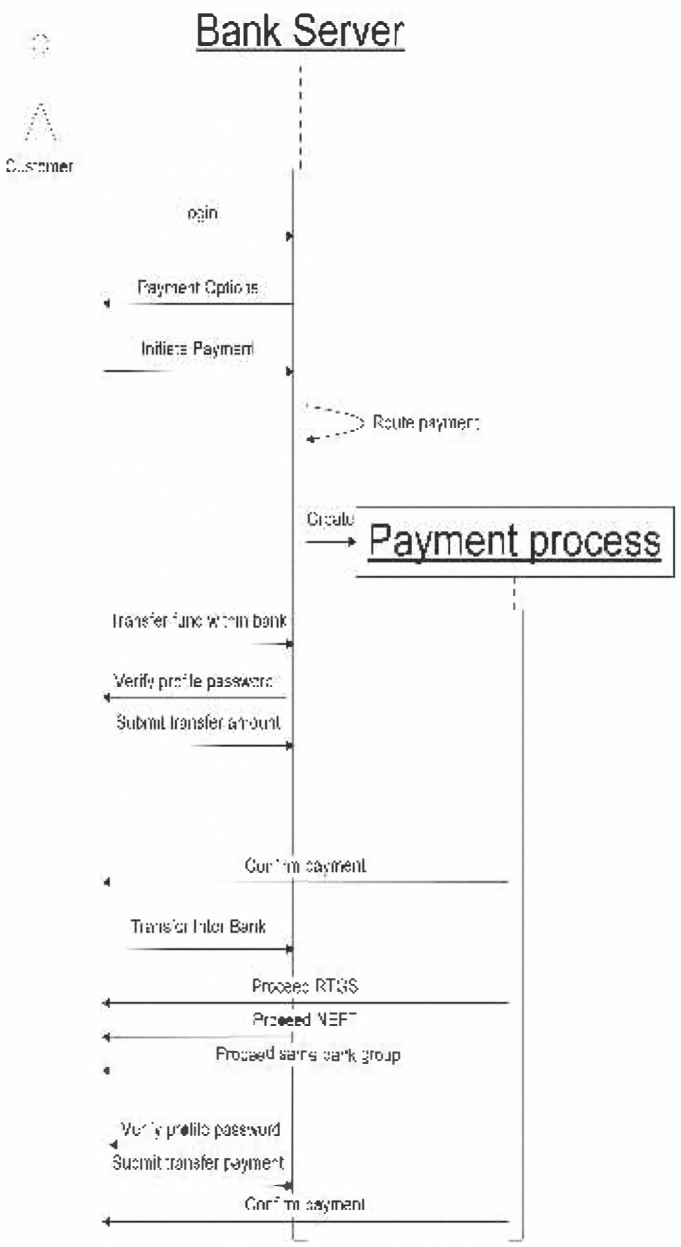

Figure 2. Sequential diagram of EFT

A state transition diagram for the sequential operation of an Intra bank transfer has been considered here to show the steps involved. All possible use cases are depicted in totality in Fig. 3 in the form of state transition diagram view of EFT module. The various states are a: log in, b: payment options, c: initiate payment, d: interbank transfer, e: transfer with in bank, f: Proceed RTGS, g: Proceed NEFT, h: Proceed bank's allied group, i: verify profile password, j: submit transfer amount, k: confirm payment

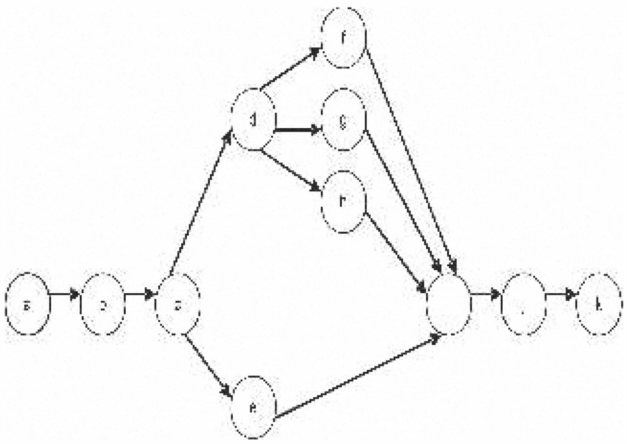

Figure 3. State transition diagram of EFT

For a sequential intra bank money transfer, the following state transition diagram in Figure 4 suffices.

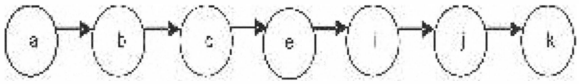

Figure 4. Sequential intra bank money transfer

The following data compiled through practical tests and compilation of relevant usage profiles for the interbank transfer is used for the calculation purposes

R_a=0.995, R_b=0.991, R_c=0.993, R_e=0.996, R_i=0.990, R_j=0.998, R_k=0.999

P_{ab}=0.95, P_{bc}=0.96, P_{cc}=0.99, P_{ei}=0.99, P_{ij}=0.92, R_{jk}=0.97

The state transition view of the system, depicted in Figure 3 which is subset of Fig. 4, we obtain transition matrix M from the procedure discussed above in section IV.

M =

states	a	b	c	e	i	j	k
a	0	0.9450	0	0	0	0	0
b	0	1	0.9510	0	0	0	0
c	0	0	1	0.9830	0	0	0
e	0	0	0	1	0.9860	0	0
i	0	0	0	0	1	0.9100	0
j	0	0	0	0	0	1	0.9680
k	0	0	0	0	0	0	0

From the Cheung's model [12], the overall system reliability can be computed as follows:

$$R = (-1)^{k+1} \frac{|E|}{|I-M|} * R_k$$

where I is the identitiy matrix of size k x k. |I-M| is the determinant of matrix (I-M) and |E| is the determinant of the remaining matrix excluding the n^{th} row and first column of the matrix (I-M).

Det(E)= |D| = 0.7673

n = 7, where n is number of states in the state transition diagram.

T(1, 7) = [(-1)n+1 |D|] / [|I-M|] = 0.7673

Architectural Reliability of the system, R_w = T(1, 7) * R7 = 0.7673 * 0.99 = 0.7596 = 75.96%.

Reliability from the black box approach as estimated at prototype level prior to the implementation of actual development, R_B is 92.04% from cycle3 in the Table II.

Let the weightage factors be 50% each.

Overall Reliability = $W_B R_B + W_w R_w$ = 84%

VII. Conclusions

Modeling Markov based software reliability estimations at the architectural level predicates the means to realize the desirability of improvement in early software life cycle. While non functional requirements play a pivotal role in the early software reliability prediction, prototype level studies involving architectural design specifications (white box approach) must take into account the overall reliability estimation done with the intent of appropriate integration of non functional and functional features. We have evolved an effective reliability quantification analysis at prototype level of a financial application case study with both non functional test data of SDLC phases captured from defect consolidation table in the form orthogonal defect classification as well functional requirements at requirement and design phases captured through software architectural modeling paradigms. Only batch sequential style architecture has been considered for illustration. Other styles could be pegged in the generic formulation.

References

[1] D. Hamlet, "Foundations of software testing: dependability theory," Proc. second ACM SIGSOFT symposium on Foundations of software engineering, 128 – 139, 1994M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[2] J. L. Peterson, "Petri Nets," ACM Computing Surveys, Vol 9 No.3, pp. 223-252,1977.

[3] T. Murata, "Petri Nets: Properties, Analysis and Applications", Proc. of IEEE, Vol. 77 No.4, pp. 541-560, 1989.

[4] W. L. Wang, D. Pan, and M-H. Chen, "Architecture based software reliability modeling," The Journal of Systems and Software, Vol. 79, pp. 132-146,2006.

[5] W. L. Wang and M. H. Chen, "Heterogeneous software reliability modeling," Proc. 13th Intl. Symp. on Software Reliability Engineering, Minneapolis, USA, Nov. 2002.

[6] W. L. Wang, Y. Wu, and M. H. Chen, "An architecture-based software reliability model", Proc. Pacific Rim International Symposium on Dependable Computing, Hongkong, pp. 143–150, 1999.

[7] W. L. Wang, and D. Scannell, "An architecture-based software reliability modeling tool and its support for teaching," Proc. 35th ASEE/IEEE Frontiers in Education Conference, Indianapolis, USA, 2005.

[8] I. Krka, L. Cheung, G. Edwards, L. Golubchik, and N. Medvidovic, "Architecture-Based Software Reliability Estimation: Problem Space, Challenges, and Strategies," Proc. DSN Workshop on Architecting Dependable Systems, Arkansas, USA, 2008.

[9] M. Xiaoguang and D. Yongjin, "A general model for component-based software reliability," Proc. 29th EUROMICRO Conf. New Waves in System Architecture, Spain, 2003.

[10] J. D. Musa, "Operational profiles in software reliability engineering," IEEE Software, Vol. 10, No. 2, pp, 14-32, 1993.

[11] D. Hamlet, D. Mason, and D. Woit, "Theory of system reliability based on components", Proc. 23rd International Conference on Software Engineering, Ontario, Canada, pp. 361-370, 2001.

[12] R. C. Cheung, "A user-oriented software reliability model", IEEE Transactions on Software Engineering, Vol. 6, No. 2, pp. 118-125, 1980.

[13] K.Krishna Mohan , A.K. Verma, A.Srividya ,G.Varaprasada Rao, Ravi Kumar Gedela, "Early Quantitative Software Reliability Prediction Using Petri-nets", proc. IEEE Region 10 Colloquium and the Third ICIIS, Kharagpur, INDIA December 8-10, 2008.