Recovery Oriented Software Reliability Model with Fault Fixing and Treatment Processes

S.CHANDRASEKARAN[1], A.VINODHINI[2], A. SOORYA NIVEDHA[3]
Kumaraguru College of Technology[1], Velammal Engineering College[2],
K.P.R Institute of Engineering Technology[3],
Anna University, Chennai, INDIA
chandrasekaran_s@msn.com, vinodhiniashok@gmail.com, sooryagupta12@gmail.com

Abstract: - The objective of the work is to propose a recovery oriented reliability (RoR) model for the software reliability enhancement through fix and treatment strategy. The existing reliability enhancement models are emphasizing various redundancy techniques both in hardware and software without focusing a formal way of recovery time minimization from the affected or degraded states in the software systems. The recovery oriented computing is based not only on the severity of the damage caused by the uncertain stimuli or random events on system components, third party software and middleware but also on the duration of such events caused that damage. A Replication Enabled Collaborative Optimal Recovery (RECOVER) model is proposed in which the correct and timely activation of multiple recovery mechanisms through time and data replication techniques are used for the enhancement of software reliability. The functionality might have been affected by various direct or hidden faults due to the frequent changes in operating environment of the application software under consideration.

Key-Words: - Recovery oriented, Redundancy techniques, uncertain stimuli, Module Replication, Hidden fault, Collaboration of components

1 Introduction

The software reliability becomes a challenge not only in the case of designing critical applications like signaling systems but also in the verification and validation of complex scientific systems. In case of considering the effect of debugging process on an entire system in the development of a method of reliability assessment for open source project, it is necessary to grasp the deeply-intertwined factors, such as programming path, size of each component, skill of fault reporter, and so on. [1]. This work addressed two central goals where one is to describe the proposed layered architecture that encapsulates the self-correcting mechanism; and the other is to present each phase of the self-correcting mechanism in detail [2]. In this work, the authors presented an approach for the timed hazard analysis of systems that reconfigure in order to react to errors in the system, so called self-healing systems. In another work, the temporal characteristics of failure propagation in dynamic systems are captured by timed failure propagation graphs[3]. In other models, a piece of code is allowed to get executed until a specific class of instructions is encountered to determine whether bad behavior is about to be exhibited [4]. According to self-correcting or self-healing, a transformation of is essential to implement into a platform specific model with failure detection features [5]. The techniques to automatically derive assertions to effectively detect functional failures, locate the faults underlying the failures, and identify sequences of actions alternative to the failing sequence to bring the system back to an acceptable behavior are derived [6]. In reliability modeling, fix and treatment approaches are able to deal with the reality of limits to knowledge, incomplete specifications, and incomplete designs. To enhance the reliability of a software system, the design must have a set of abstractions about underlying component functionality and behavior. And at the same time, the software systems must have some level of knowledge about themselves and their environment in order to sense the operating mode changes, accumulated component and resource faults, adaptations to external environments, component evolution, and changes in system usage [7]. The reliability of such software system is further defined as its ability to maintain operation over a period of time. The software must be made as "operational", if it can successfully deliver messages from sources to alive destination(s) even when some nodes in the routing path die [8]. The approach uses a recovery and treatment functions in order to achieve a state that is able to detect permanent faults during runtime and also to recover from them by an autonomous reconfiguration which is called asynchronous Four-State-Logic (FSL) design [9]. The next evolution in the recovery approach confirms that it is suitable for deployment by verifying that it satisfies three criteria: survivability, correctness and performance. A selected rescue point provides survivability if error virtualization at that point enables the application to survive a recurrence of the fault [10]. The main focus of the paper is to propose a formal recovery model for the assessment of software reliability through the fault fixing and treatment actions satisfying the computing environment conditions.

The organization of the paper is as follows: Section 2 proposes a formal model of recovery oriented software reliability computing based on the replication feature. Section 3 explains the introduction of fault fixing and treatment processes in the calculation of the mean time between failures as a metric for software reliability for various constraints in software recovery. Section 4 explores the possibility of collaborative recovery process through optimal number of replications of the affected software modules during run time. Section 5 concludes the reliability assessment using mathematical model with fixing and treatment strategies and possible limitations of the work.

2 Formal RECOVER Model

The reliability of the software is depending on the techniques with which the affected components are recovered or replaced or replicated. The replacement policy and the renewal processes are included in terms of the fault fix time and treatment time which may not be linear. In the proposed recovery oriented reliability model, two different types of actions are introduced. That is, there are two sets of states one is sick and the other is recovered. Similarly two sets of actions are introduced one is treatment set and the other is relapsing set as shown in Figure 1. For example, during treatment_action1 and treatment_action2, there is a transition from one type of sick states Q_{Si} to one of the recover state set Q_{Rj}. This action depends on a set of environment variables $\{e_i\}$, function f_i and the production recovery rule p_k. Similarly during relapsing_action1 and relapsing_action2, there is a transition from one of the recover states set Q_{Rj} to sick state set Q_{Si}. This action depends on a set of environment variables $\{e_i\}$and function f_i but not on the production recovery rule p_k.

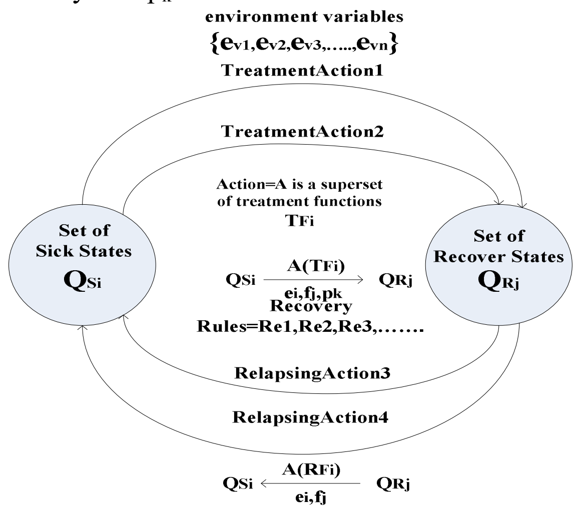

Figure1. Core states in the recoverable model

The core state transition can be represented as shown below: The left hand side is any sick state from the possible set of sickness and the right hand side represents any permitted recovered state in that set when the conditions like the environment ei, treatment function fj, and the recovery rule pk are met.

$$S_i \left| \xrightarrow[p_k]{e_i, f_j} \right. r_j$$

Based on the paradigm of recovery, the formal model is proposed. Hence it can be written as, RECOVER = <Q_{sick}, $Q_{recover}$, $T_{treatment}$, $E_{environment}$, $A_{collaborative}$, $R_{recovery}$ > or represented as a tuple of sets or multi sets like,

$$= < Q_S, Q_R, T, E, A, R >$$

where, Q_P is the set of possible past *sick states* of the system with initial state as Normal, Q_F is the set of possible future *recover states* except the normal T is the set of *treatment functions* for recovery, E is the set of *environment variables* for the system to recover, A is the set of *collaborative actions* taken by recovery measures and R is the set of *recovery rules* to improve reliability. The proposed formal recover model incorporates two levels of recovery. One is called as *Possible Recovery*, that is the inner set of states with certainty and the other type is *Probable Recovery*, which is the outer set with uncertainty through the actions performed on the affected and sick states. The model portrays a state of recovery (SoR) if an action of recovery (AoR) is performed when the condition of recovery (CoR) is satisfied.

When a number of software modules interact, then the total number of possible states of the software system contains past states and future states of each and every module. The set of past states include normal or run state, degraded state, failed state, obsolete, repaired, replaced, condemned state and the future states include ready for treatment, getting cured, recovered state. At the same time, the set of treatment functions towards recovery may include energy boosting, selectivity improvement, reliability enhancement and safety improvement functions. The set of environment variables consist of external agents, time for improvement permissible work spaces. The collaborative actions may be timed transmission, collaborative redundancy, recoverability policy checking. Considering multi-set interactions, each and every step in recovery depends on the degree of sickness or affected states(S), recovery measures (T) and the recovery rules(P) applied within the permitted environment(E) including the time of recovery.

For illustration, a set of software recovery steps at any point of time can be represented as,

$$\text{Recovery}(t) = < S_{ti}, T_{pj}, E_{nk}, P_{r\,l} > \text{ where}$$

$S_{ti}=\{ S_{ci}, S_{pi}, S_{wi}, S_{ei}, S_{ri}, S_{fi} \}$, is being a set that consists of a collection of states like current, process-treatment, wait, error, recovery, and final as mentioned above. The inter-dependent recovery measure is, T_{pj}={Transition*Permission} which a set of permission to perform the transition. Similarly the environment needed for applying the recovery rule is mentioned as below:

$$E_{nk} = \{E_{1k}, E_{2k}, E_{3k}, \ldots\ldots, E_{nk}\}$$

$$P_{r\,l} = \text{Recovery Rules.}$$

The interacting recovery model can be further represented as a sequence of recovery phases of the software module as,

$[\text{STEP}]_1,[\text{STEP}]_2,\ldots,[\text{STEP}]_n$. The number of STEP elements in this sequence indicates the recovery stages which when increases, the treatment functions are not effective or fixing is not correct.

2.1 Reliability and Renewal Processes

In software reliability theory, renewal processes describe the model of a subsystem or an item that is in continuous operation needs to be replaced for each failure within a negligible amount of time by a new statistically identical subsystem or item. The proposed recovery model discusses not only the renewal processes but also the conditions for renewal and the replacement policy in terms of recovery rules. The generation of a renewal process can be done by introducing a positive random replacement within a time distribution, $G(x)$. An alternating renewal process is a process with two states, which alternate from one state to the other after a stay (sojourn) time distributed according to $F(x)$ and $G(x)$, respectively.

For reliability applications, Γ_i denotes the i-th failure-free operating time and Γ_i' the i-th repair time. These random variables are distributed according to[11]

$F_A(x)$ for Γ_0 and F(x) for Γ_i, $i \geq 1$ and

$G_A(x)$ for Γ_0' and G(x) for Γ_i', ≥ 1 .

In this transition diagram, the recovery of the system is demonstrated in three different possibilities as shown below:

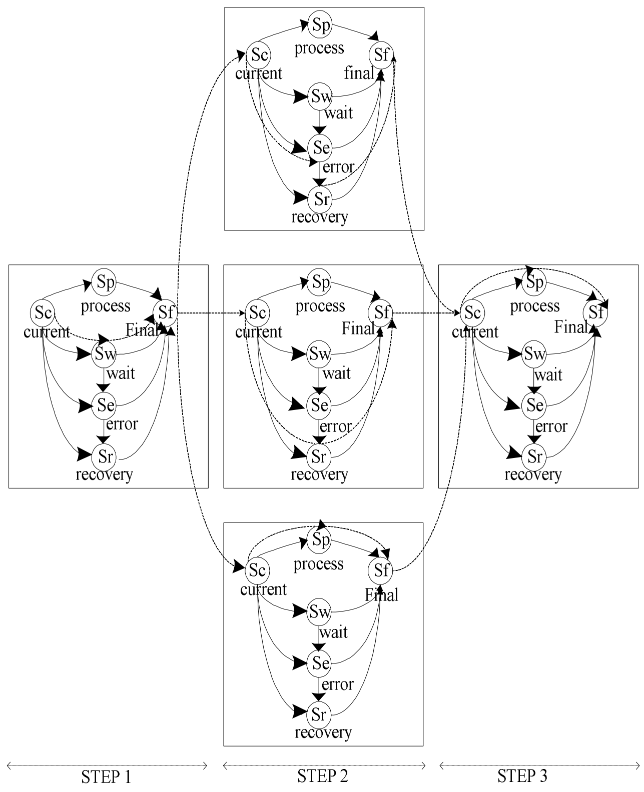

Figure 2: STEP based recovery model

3 Reliability Assessment with Fix and Recovery

J. T. Duane proved that the failure data of different systems during their development programs exhibit a linear property where the cumulative mean time between failures (MTBF) versus cumulative operating time followed a straight line when plotted on log-log paper. In the above model, Duane demonstrated that if N(t) is the number of failures by time T, the observed mean (average) time between failures, $MTBF_c$ at time T is, $\mathrm{MTBF_c} = \mathrm{T} / \mathrm{N(t)}$. The development of TAAF (Test-Analyze-And-Fix) time considered in reliability growth is the accumulated number of missions, events, launches, or discrete periods of operation, *e.g.* month of TAAF or cycles of operations to a failure. But in the above TAAF time, the recovery time is not considered which is a critical factor in distributed computing like collaborative web services, federated business services and virtual applications on grid and cloud platforms. Hence the focus of the work is to consider the recovery time for the reliability enhancement of such applications using fix and recovery through self-replication modules as and when demanded.

The recovery action can be thought of TAFAR (Test-Analyze-Fix-And-Recover) which will enhance not only the reliability but also the return on investment (ROI) including the recovery cost of the software components. In the recovery oriented reliability approach, the mean time between failures is explored in detail considering the fault detection time to decide whether the fault can be recovered or not. Additional parameters like fix time, decide time and recover time are also included in the calculation of MTBF.

Mathematically MTBF is equal to the sum of time between the occurrence and its fix (TOF),time between fix and recovery (TFR) and the time between the last recovery and next occurrence(TRO).

Let the number of failures and their corresponding recovery techniques over a time period of T be $\mathrm{N_{FR}(T)}$.

The cumulative time between last recovery and next occurrence (RO) has to be analyzed instead of simply the MTBF value with respect to cumulative operating time. Hence the MTBF over a period T with number of failures and recovery actions $\mathrm{N_{FR}(T)}$ can be represented as,

MTBF = MTOF(resource constrained)+
MTFR (resource constrained) +
MTRO (constant).

The value for the mean time between last recovery and next occurrence (MTRO) is considered as a fixed constant since the software component after recovery will not exhibit any behavioral difference with that of the original behavior under the same conditions. Hence the term MTRO can be taken as a universal constant "c".

In reliability engineering MTOF, the mean time between fault occurrence and fix can be written as,

$$MTOF = \sum_r [\,\Delta Xfr \,/\, Pfr.\rho fr\,] \quad (1)$$

and MTFR, the mean time between fix and recover can be written as,

$$MTFR = \sum_r [\,\Delta Xrr \,/\, Prr.\rho rr\,]. \quad (2)$$

The term ΔXfr represents the change in resource requirement for fixing the errors, Prr is the quantity of resource to fix and ρrr the utilization factor of that resource. Similarly, ΔXrr represents the change in resource requirement for recovering the errors, Pfr is the quantity of resource to recover and ρfr the utilization factor of that resource. In time domain, ΔXfr and ΔXrr may be represented as the product of the number of failures during the period and the mean time to fix it and recover it respectively.

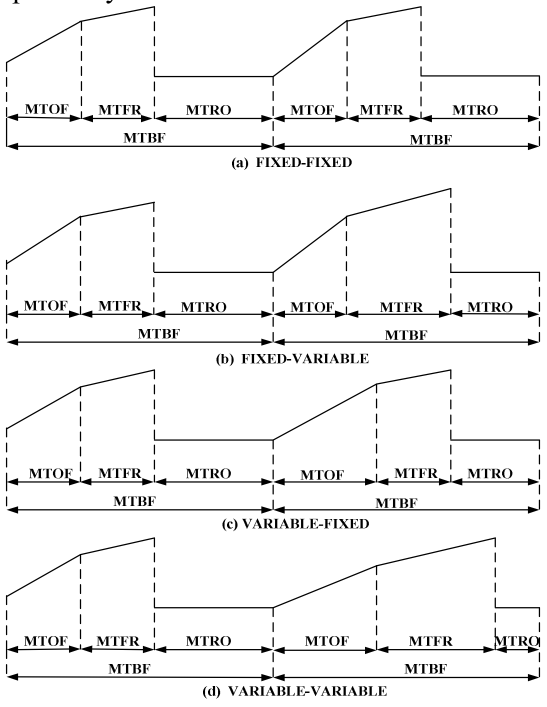

Figure 3: MTOF and MTFR

In figure 3.a, the MTOF and MTFR are fixed across the two MTBFs and in figure 3.b, MTOF is fixed, but MTFR varies across the two MTBFs.

In the proposed software reliability growth model, there are eight possible combinations in which the three parameters like fault occurrence, recovery and fixing time may be varying based on the components used in the software application. The error fixing and recovery phases are resource constrained. That is the resources needed to fix different types of errors are limited and their utilization is also considered. Similarly the recovery from different types of errors needs different resources to be used with different utilization factors. But the main objective is to maximize the mean time between failures to enhance the reliability. To illustrate, an example is considered where application software has been tested for 4 hours and it is found that there are 80 faults in the application. As per Duane Model, the cumulative failure rate or average failure rate can be calculated as

$$C(t) = \frac{N(t)}{t} = \frac{80}{4} = 20\, faults\, /\, hour.$$

In the proposed TAFAR model, the test period is considered as a combination of both error fixing time and necessary recovery time. If the error fixing or detection time for all the 80 faults is say 3 hours, then it may be assumed that the reason for the above fix time is due to training the personnel, developing the test cases and activating the automated debugging of the failed components or modules. During the remaining one hour period, all the 80 faults can be eliminated or recovered with the help of the team work or with the help of any third party error correction software. Hence, it may be written as,

$$C(t) = Fx(t) + Re(t) \qquad (3)$$

where $Fx(t)$ can be called as cumulative fix rate and $Re(t)$ as cumulative recovery rate.

As in the above example, $Fx(t) = 80/3$ and $Re(t) = 80/1$

In time perspective, it can be written as,

$$C(t) = [Nf(t) / MTOF] + [Nr(t) / MTFR] \qquad (4)$$

Here $Nf(t) = Nr(t) = Nfr(t)$.

So the cumulative failure rate,

$$C(t) = Nfr(t)\, [(MTFR + MTOF) / (MTFR)(MTOF)]$$

Taking natural logarithm on both sides,

$$ln\,[C(t)] = ln[\,Nfr(t)] + ln\,[MTFR + MTOF] - ln\,[MTFR] - ln[MTOF] \qquad (5)$$

As per Duane Model α denotes the shape of the growth curve and λ is the size parameter of the curve.In other words in the ln–ln plot, it can be interpreted as $y = \delta + \alpha(x)$ where

$$x = (MTFR)(MTOF) / (MTFR + MTOF) \qquad (6)$$

$$y = \delta + \alpha\, x \qquad (7)$$

To make it simple, represent MTOF and MTFR can be represented as Tf and Tr respectively.

Taking anti-logarithm on both sides, $C(t) = \lambda[(Tf + Tr)]^{\alpha 1}\, [(Tf)(Tr)]^{-\alpha 2}$ (8)

4 Replication Enabled Collaborative Optimal Recovery (RECOVER)

The proposed recovery oriented software reliability model with fix and treatment techniques can be realized by the following implementation phases. Each and every phase has to contribute not only to fix the errors and their nature but also to determine the optimal recovery techniques. These techniques should be intelligent enough to correct or bypass the condition so as to minimize the delay in recovery. The correctness of the treatment function for any sick software module depends on the intelligent action triggered on the module or by a function considering the environment variable at that point of time. It clearly indicates there needs to be collaboration between various treatment objects or functions based on the context of the ill-state through a sensible insulation of recovery module. For example, a low response process due to concurrent request at the interface may be identified quickly and same component can be virtualized to solve the situation. Many such virtual components can be composed to satisfy the user expectation with minimum delay. The implementation phases are given below:

Timed Replication

During timed replication, a single process is replicated into another similar process. Consider a case where a computation has to occur quickly. Under such a situation, the output that is generated may not always be correct. To check the correctness, the same input has to be replicated and computed again.

$$P \equiv P'', \qquad fi \equiv fj$$

The process P and the replicated process P'' should be equivalent when the function f_i in the process P and the function f_j in process P" should be equivalent thus becoming reflexive, symmetric and transitive. The process of renewal or replication should be carried out at the administrative side as soon as the performance deceleration is detected.

Intelligence Action Enabling

In intelligence action enabling phase, the incoming process is split into two or more sub processes. The sub processes generated are executed in parallel or in sequence. This separation or segmentation of the process is referred to as action intelligence.

$$S(Pijk) = (Pi1 + Pi2)t1 + (Pj1 + Pj2)t2 + (Pk1 + Pk2)t3$$

Here the process P_{ijk} is split up into sub processes P_i, P_j, P_k where the sub-process p_i is further split up into P_{i1} and P_{i2}, P_j into P_{j1} and P_{j2} and P_k into P_{k1} and P_{k2}. The variables t_1,t_2 and t_3 represent three different time slots over which an individual pair of processes are executed.

Joining
During this operation the two simple processes are joined together to produce a single process provided the constituent processes have similar features. Thus, with single process, the execution reduces the time and memory. It can be represented as shown below:

$$P1 + P2 \Rightarrow \sum_{i=1}^{2} Pi = Pj$$

The simple process P_1, P_2 and P_3 are joined together to form a single process P_j where P_1 and P_2 possess similar functionalities.

Collaboration
When many processes are to be combined, an action called collaboration has to be performed if the processes are asynchronous in terms of their start time, dead line and execution time. It involves the summation of more than two processes to give a single process.

$$Pi1 + Pi2 \ldots\ldots Pin \equiv> Pk$$

Optimal Delay
There may be a few events that demand the execution of the same process after a fixed span of time. For example a warning message may be sent at regular intervals. This necessitates a delay action to be inserted by which a process is delayed to some known duration.

$$P \xrightarrow{\text{Delay}} P_d$$

Context Sensitiveness
The optimal recovery through collaborative replications can be achieved by two important techniques. One is location identification in which the correct process with accurate error detection can be identified to initiate the replication processes and the second one is being the sensible installation process. The installation should check the environment variables and also the workspace that control the number of renewals to be permitted based on the interface capacity.

5. Conclusion

The work proposes a software reliability assessment model focusing the fault fixing and treatment processes. A formal approach using recoverability of the affected software module is explored using sickness conditions, treatment actions, environment conditions and recovery rules. The recovery oriented model gives a better assessment of reliability when high availability of the software system is needed where renewal processes are automated. The implementation of the model can be done using optimal replication and collaborative recovery strategies to enhance the reliability of the software. The limitation of the model is exact determination of fault occurrence and detection time along with the states of software. When multiple failures are detected, then the series nature of replication and collaboration processes poses a serious limitation in the overall recovery process. The future work is focused towards the restoration of all state invariables and control variables to decide the correct stage for replication in a more formal way using parallel Turing machine approach

REFERENCES

[1] Tamura.Y and Yamanda.S, A Method of User-Oriented Reliability Assessment for Open Source Software and its Applications, *International Conference on IEEE*, 2006, Vol.3, pp. 2185-2190.

[2] Jeongmin Park Ea.,Tl, Self-Healing Mechanism for Reliable Computing, *International Journal of Multimedia and Ubiquitous Engineering*, 2008,pp. 1-12.

[3] Sherif Abdelwahed , Gabor Karsai , Gautam Biswas, A Consistency-Based Robust Diagnosis for Temporal Causal Systems, *International Workshop on Principles of Diagnosis*,2005.

[4] Stelios Sidiroglou Ea.,Tl, Hardware Support for Self-Healing Software Services, 2009, pp. 1-7.

[5] Neeraj Mohan, Parvinder S. Sandhu, and Hardeep Singh, Impacts of Faults in Different Software Systems: A Survey *World Academy of Science*,2009.

[6] Alessandra Gorla, Achieving Cost-Effective Software Reliability through Self-Healing, *Computing And Informatics,* 2010, pp.1-22.

[7] Chandrasekaran Subramanium ea.,tl, Application Safety Enhancement Model using Self Checking with Software Enzymes, *ICMV,*2011,pp.1-7.

[8] Philip Koopman, Elements of the Self-Healing System Problem Space, 2003, pp.1-6.

[9] Thara Angskun, Achieving Cost-Effective Software Reliablity Through Self-Healing, *Computing and* Informatics, 2010, pp.1-8.

[10] Panhofer Thomas ea,.tl, A Case Study:Reliablity estimation and experimental results of a self-healing asynchronous circuit, 2010,pp. 91-98.

[11] Stelios Sidiroglou ea,.tl, ASSURE:Automatic Software Self-Healing Using REscue points, 2009,1-12.