Component Criticality Analysis to Minimizing Soft Errors Risk

Muhammad Sheikh Sadi[1], D. G. Myers[1], Cesar Ortega Sanchez[1], and Jan Jurjens[2]
[1] Curtin University of Technology, Australia
[2] TU Dortmund, Germany

Abstract— Minimizing the risk of system failure arising from soft errors in any computer structure requires identifying those components whose failure is likely to impact on system functionality. Clearly, the degree of soft errors tolerance is not the same for all components. Hence, to improve soft errors tolerance, critical components can be flagged and measures can be taken to lower their criticalities at an early design phase. This improvement is achieved in this paper by presenting a criticality ranking (among the components) formed by combining a prediction of soft errors, the severity of them, and a propagation of the soft errors at the system modeling phase; and also by pointing out ways to apply changes in the model to minimize the risk of degradation of desired functionalities. Case study results are given to validate the approach.

Index Terms— Criticality Analysis, Soft Errors, Reliability Risks, UML Metrics

1 INTRODUCTION

The temporary unintended changes of states resulting from the latching of single-event transients (transient voltage fluctuations) create transient faults in a circuit and when these faults are executed in the system, the resultant error is defined as a soft error. Soft errors cannot create physical damage to a chip, but can be catastrophic for the desired functionalities of the system [1], [2], [3]. Specifically, they are a matter of concern in those systems where high reliability is a necessity [4], [5], [6]. Space programs (where a system cannot malfunction while in flight), banking transactions (where a momentary failure may cause a huge difference in balances), mission critical embedded applications, and so forth, are a few examples where soft errors are severe. Increases in system complexity, reduction in operational voltages, exponential growth of the number of transistors per chip, increases in clock frequencies and shrinking of devices significantly increase the rate of soft errors [7], [8].

Prior research to cope with soft errors mostly focuses on post-design phases, such as circuit level solutions, logic level solutions, spatial redundancy, temporal redundancy, and/or error correction codes. Early detection and correction of such problems during the design phase is much more likely to be successful than detection once the system is operational [9]. Estimating reliability (or at least identifying failure-prone components) early in the life-cycle of a design is therefore preferable [10], [11]. From a pure dependability viewpoint, critical components attract more attention of soft errors tolerant approaches than others do, since reliability of a system is correlated with the criticality of the system [12], [13]. Hence, an approach is needed at the design stage to highlight those components where soft errors are critical.

This paper examines the use of metrics to identify those components in a system model that are soft errors critical in creating impacts in system functionality. It also investigates how to encourage the designer to explore changes that could be made in the existing model. For example, how the criticalities of top-ranked (critical) components could be minimized, or how these components could be replaced with alternatives and/or with less critical components is examined. The objective is to keep the functionality and other constraints of the system unaffected or to make a trade-off between them, with the goal to minimize the risks of soft errors. Case studies illustrate the effectiveness of this approach in determining components' criticality rankings and then lowering their criticalities. The model is expressed in Unified Modeling Language (UML) since this allows the modeler to describe different views on a system, including the physical layer [14], [15].

The paper is organized as follows. Section 2 describes related work. Section 3 outlines the methodology to measure and reduce components criticality employed in this research. This methodology is applied to a real-life case study in Section 4. Section 5 discusses the results obtained. Finally, in Section 6, conclusions and recommendations for future extensions are discussed.

2 RELATED WORK

Software based approaches to tolerate soft errors include redundant programs to detect [16], [17], [18], [19] and/or recover from the problem [20], duplicating instructions [21], [22], task duplication [23], dual use of super scalar data paths [24], and Error detection and Correction Codes (ECC) [25]. Chip level Redundant Threading (CRT) [16] used a load value queue such that redundant executions can always see an identical view of memory. Although the load value queue produced an identical view of memory for both leading and trailing threads, integrating this into the chip multiprocessor environment requires significant changes. In [17], the authors described the concept of sphere of replication in aiding the design and discussion of fault tolerant Simultaneously and Redundantly Threaded (SRT) processors. The parts of the computer system that fall outside the sphere are not replicated and must be protected by other means such as information redundancy. AR-SMT (Active-stream/Redundant-stream Simultaneous Multithreading) [18] increases the memory requirement and bandwidth pressure two times, since both threads required accessing the cache and individual memory. Doubling the memory may stress the memory hierarchy and degrade performance. Walcott et al. [26] used redundant multi threading to determine the architectural vulnerability factor, and Shye et al. [27] used process level redundancy to detect soft errors. In redundant multi threading, two identical threads are executed independently over some period and the outputs of the threads are compared to verify the correctness. EDDI [21], and SWIFT [22] duplicated instructions and program data to detect soft errors. Both redundant programs and duplicating instructions create higher memory requirements and increase register pressure. Error detection and Correction Codes (ECC) [25] adds extra bits with the original bit sequence to detect error. Using ECC to combinational logic blocks is complicated, and requires additional logic and calculations with already timing-critical paths.

Hardware solutions for soft errors mitigation mainly emphasize circuit level solutions, logic level solutions and architectural solutions. At the circuit level, gate sizing techniques [28], [29], [30] increasing capacitance [31], [32], resistive hardening [33] are commonly used to increase the critical charge (Qcrit) of the circuit node as high as possible. However, these techniques tend to increase power consumption and lower the speed of the circuit. Logic level solutions [34], [35], [36] mainly propose detection and recovery in combinational circuits by using redundant or self-checking circuits. Architectural solutions mainly introduce redundant hardware in the system to make the whole system more robust against soft errors. They include dynamic implementation verification architecture (DIVA) [37], and block-level duplication used in IBM Z-series machines [38]. DIVA [37] in its method of fault protection assumed that the checker is always correct and it proceeds using the checker's result in case of a mismatch. So, faults in the checker itself must be detected through alternative techniques.

Hardware and software combined approaches [39], [40], [20], [41], [23], [42], [43] use the parallel processing capacity of chip multiprocessors (CMPs) and redundant multi threading to detect and recover the problem. Mohamed et al. [41] shows Chip Level Redundantly Threaded Multiprocessor with Recovery (CRTR), where the basic idea is to run each program twice, as two identical threads, on a simultaneous multithreaded processor. One of the more interesting matters in the CRTR scheme is that there are certain faults from which it cannot recover. If a register value is written prior to committing an instruction, and if a fault corrupts that register after the committing of the instruction, then CRTR fails to recover from that problem. In Simultaneously and Redundantly Threaded processors

with Recovery (SRTR) scheme [20], there is a probability of fault corrupting both threads since the leading thread and trailing thread execute on the same processor. Others [40], [23], [42], [43] have followed similar approaches. However, in all cases the system is vulnerable to soft error problems in key areas. In software-based approaches, the complex use of threads presents a difficult programming model. In hardware-based approaches, duplication suffers not only from overhead due to synchronizing duplicate threads, but also from inherent performance overhead due to additional hardware. Moreover, these post-functional design phase approaches can increase time delays and power overhead without offering any performance gain.

Few approaches [44], [45] dealt with the static complexities of the system as a risk assessment methodology to minimize the risks of faults. McCabe [46] introduced Cyclomatic complexity, which is measured based on program graphs. However, these static approaches do not deal with the matter of how a module functions in its executing environment. A fault may not manifest itself into a failure if never executed. Cortellessa et al. [9], and Yacoub et al. [13] defined dynamic metrics that include dynamic complexity and dynamic coupling metrics to measure the quality of software architecture. To assess the severity of the components they have defined only three levels of system failure. However, in real life scenarios, only three severity levels are not sufficient to represent several possible failure modes. Criticality analysis at the sub-system level along with failure Mode and Effect Analysis (FMEA) is also becoming popular in fault tolerant research. A few common methods for assessing criticality in FMEA are Risk Priority Number (RPN) [47], the MIL_STD 1629A Criticality Number ranking [48], and the multi-criteria Pareto ranking [49]. However, difficulties in calculating failure rate values or probability of failure make Criticality Number ranking, and the multi-criteria Pareto ranking unpopular to researchers. Sherer [50] has shown a risk assessment methodology by measuring the consequences of errors in different modules. However, the high complexity of the method in real-life applications makes it obsolete. Moreover, the method is applied at the later stages of the system design, which can mean a huge cost increase.

3 A METHODOLOGY TO MEASURE AND REDUCE COMPONENT'S CRITICALITY

Complexity is taken as a measure of the likelihood of a component to be affected by soft errors. Severity of failure of components is taken as a measure of the impact of a system's functionality being affected by a component suffering a soft error. The methodology presented here is to measure the complexity and severity of each component, plus the propagation of soft errors from that component, and then take the product as a measure of criticality. The model is examined by refactoring to lower component criticality by maintaining constraints. The details of these steps are outlined as follows.

3.1 Measuring the Complexities of the Components

There is a correlation between the likelihood of faults/errors/failures and the complexity of a system [12], [51]. The more complex the component, the greater the probability is of fault/error/failure proneness. Complexity analysis does not measure the impact of components in system functionality, but it shows the rank of likelihood of having faults/errors/failures among the components. Complexity is measured, in this paper, by an assessment of Execution Time (ET) during simulation and Message-In-and-Out frequency (MIO).

3.1.1 Execution Time during Simulation

The Failure-In-Time (FIT) of a system due to soft error is proportional to the fraction of time in which the system is susceptible to soft error if the circuit type, transistor sizes, node capacitances, temperature etc. are kept at constant [52]. Hence, the fractional time that a component uses in the execution of a system can flag the error proneness of that component. Using ET during simulation to measure a component's complexity is a novel approach. Components are

executed for a specific operation. Users can specify any operation that seems to be involved with all components. The longer duration to perform the selected operation implies that the component is being used more frequently and/or that it is experiencing many state changes. A soft error occurs at any access point of these components can spread towards all communicating components through the large number of behavioural linkages until the soft error affected component remains in execution. Hence, the likelihood of soft error may be increased if the component takes a longer ET. The calculated components' ET from the system model will flag the relative ET variations among the components. Hence, it is not a matter of concern how small the figure is but the relative ET variation will be used as one of the parameters of their complexity ranking. The method of measuring ET during simulation (to perform an operation by a component) can be shown as follows. Component state S is a function of time: S (t) where t denotes time. An external function F () is required to be executed to perform the operation F (S (t_i)) → S (t_j)): where S (t_i) is the state of a component at t_i and S (t_j) is the state of that component at t_j. Hence, ET, to execute the function F () that changes the state of the pth component from S (t_i) to S (t_j), is:

$$\mathrm{ET_p}\ (\mathrm{F}\,(\mathrm{S}\,(\mathrm{t_i})) \rightarrow \mathrm{S}\,(\mathrm{t_j})) = \sum_{j=1}^{n} d_{pj} \qquad (1)$$

Where n is the total number of state changes in the pth component's behaviour execution and d_{pj} is the duration in the jth slot of changing states of pth component. Since UML does not specify how to simulate the architecture models, Rational Rhapsody [53] is used to gain execution data via simulation. The model is executed in tracing mode. Several tracing commands are used to execute the model. The state transition times for the components are saved to a log file. At the end of the simulation, that log file is analyzed to calculate the total ET of the components to perform a selected operation. In the execution time calculation, when a component communicates with other components, every slot of time taken by itself and in all in-between events on the way of communication till the message (In UML models, components communicate with each other by message passing) is received by the other end is taken into account. The time trace commands used in the executed UML models return each slot of time along the way of message passing; hence, communication time is also taken into account in this calculation.

3.1.2 Message-In-and-Out Frequency

In a model-based system, components are often interdependent. Hence, a failure or error can easily propagate to other components. The malfunctioning behaviour of a component in a highly interdependent design cannot be easily isolated. Therefore, this dependence is considered a valuable measurement for both "a posteriori" and "a priori" analysis [54]. A posteriori analysis is conducted to trace those design aspects that were more likely to produce errors and hence correlate errors with design quality metrics. A priori analysis makes use of this dependence measurement to assess the reliability of designs in an early development phase. This paper accepts a priori analysis, since it saves both cost and time. In a system model, components communicate with each other by messages passing among them. Number of messages from and to a component shows the measure of dependence with other components. Components with more dependence could easily manifest themselves into a failure of the system because services of these components are frequently accessed by other components [13].

To figure out this fault proneness, a component's Message-In-and-Out frequency (MIO) which is the ratio of number of messages from and to a component in a scenario and total number of messages in that scenario is calculated. More specifically, a component with higher Message-In-and-Out frequency (MIO) is more likely to cause changes in the whole system if there arise any architectural or behavioral change in that component. Define MIO_{i_k} as the MIO for ith component in kth scenario. $\mathrm{M}_{(i,j)}$ is the message between component i and component j (where

j=1,….,m , $i \neq j$, and m is the number of messages from ith component to other components) in kth scenario, and n_k is the total number of messages, communicating among all the components, in that scenario. Then, MIO_i can be derived as shown in (2).

$$MIO_{i_k} = \frac{|\sum_{j=1}^{m} M_{(i,j)} | i \neq j|}{n_k} \quad . \tag{2}$$

For each component, Total MIO (TMIO) in all possible different scenarios can be calculated using (3). TMIO for ith component is:

$$TMIO_i = \sum_{k=1}^{n'} P(Sc_k) MIO_{i_k} \quad . \tag{3}$$

Where n' is the total number of scenarios in the system, $P(Sc_k)$ is the probability of kth Scenario in that system, and MIO_{i_k} is the MIO for ith component in kth scenario.

3.1.3 Overall Complexity

The Overall Complexity of the ith Component (OCC_i) is the summation of different complexity factors for that component. The equation to derive OCC_i is shown in (4).

$$OCC_i = ET_i + TMIO_i \tag{4}$$

Where ET_i and $TMIO_i$ are Execution Time, and Message-In-and-Out frequency for the ith component. Since, ET_i and $TMIO_i$ are independent on each other, OCC_i is calculated using the summation of these two factors.

3.2 Measuring the Severity of Failure of Components

A single soft error in a particular component could have a greater effect than multiple soft errors in another, or a set of, components. For this reason, the effects of soft errors in the whole system need to be analyzed by injecting transient faults (which will create soft errors if activated) into each component. These results are merged with the component's complexities to obtain a better measure of their impact on system if they are affected by soft errors. The severity of failures of components and messages is determined by the Failure Mode and Effects Analysis (FMEA) method [48]. FMEA is a procedure for the analysis of potential failure modes within a system by classifying severity or determination of the failure's effect upon the system. FMEA is an analysis technique that facilitates the identification of potential problems in the design or process by examining the effects of lower level failures [55]. Failure causes are any errors or defects in the process, design, or item. Effects analysis refers to studying the consequences of those failures. The FMEA determines, by failure mode analysis, the effect of each failure and ranks each failure according to the severity of a failure effect. Hosseini et al. [56] suggested evaluation criteria and a ranking system for the severity of effects for a design FMEA as shown in Table 1. Transient faults were injected at each component, into one bit at a time. The reason is that transient faults change the value of one bit at a time and the probability of changing two bits and/or two faults are almost zero. The test is done in simulation mode of the UML model. The 'Rhapsody Developer' tool is used for simulating the model. The fault injection is made by changing one bit of the parameter value, or anywhere in code or in the parameter name. For example, if the correct value of parameter 'x' is '7' then '7' can be changed to '8' or '9' or any other combinations. The parameter name 'x' can be changed to 'y', 'v', or any other combinations. The combinations can be generated as follows. The ASCII value of character 'x' is 58 in hex and 0111010 in binary. Therefore, changing one bit is possible in any of the six bits that will in turn generate a different character in each case. Similarly, the faulty combinations for the parameter value (the

binary value of ‘7’ is 000111) can be generated. The effects in the overall system are analyzed by the FMEA method. To validate the analysis results, several tests (three to five) are done for each component and the average integer (floor) value is taken as the resultant rank of severity. If in first three cases, the effects are almost the same then the test is terminated. The effect in the system functionality is evaluated manually by FMEA. Failure modes are classified into one of the ten categories as shown in Table 1. To analyze the cause and effect of the failure in the system better, domain expertise is required. Hence, the system is studied in detail before performing severity analysis.

TABLE 1
EVALUATION CRITERIA AND RANKING SYSTEM OF FMEA

Linguistic Terms for Severity of a Failure Mode	Rank
Hazardous	10
Serious	9
Extreme	8
Major	7
Significant	6
Moderate	5
Low	4
Minor	3
Very minor	2
No effect	1

3.3 Measuring Propagation of Soft Errors from the Components

Before measuring the component’s propagation of soft errors, the component’s complexity and severity are multiplied together to measure their combined impact (if there is any soft error) on the whole system. Measuring the propagation of soft errors refines this impact to obtain a clearer picture of the impact or criticality of each component. The method of measuring the propagation of soft errors is shown in Fig. 1, which is a scenario of a system model showing three components: C_1, C_2, and C_3.

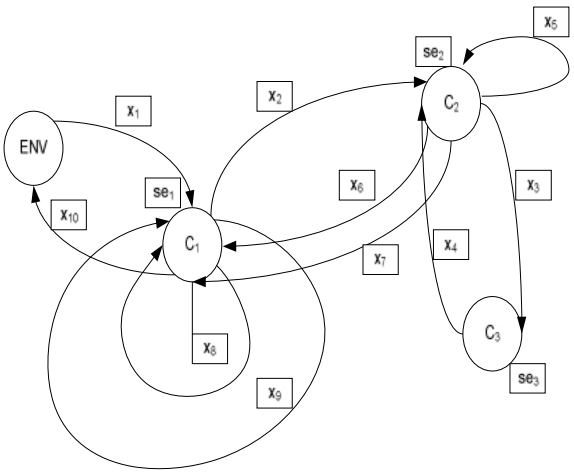

Fig. 1. An Example Scenario of a System Model to Measure the Propagation of Soft Errors

ENV denotes the environment communicating with the system. The product of complexity and severity of these three components are se_1, se_2, and se_3 respectively. In Fig. 1, x_1,….,x_{10} indicate the severity in corresponding messages where indexing is made according to their time of occurrences in the whole scenario. Soft errors may be

propagated via message communication. The propagation of soft errors from or in the environment is not considered. To measure soft errors propagation through message passing involves finding the increase in the level of consequences of each message. A soft error in C_1 (before it passes a second message) sees an increase in level of consequences in C_2 to se_1x_2, since soft errors may propagate from C_1 to C_2 through the passed message.

After passing the 2nd message, there is an increase in level of consequences in C_2: s_1x_2 and after passing the 3rd message, there is an increase in level of consequences in C_3: $se_1x_2se_2x_3$.

Similarly, after passing the 9th message, there is an increase in the level of consequences in C_1: $se_1x_2se_2x_3se_3x_4se_2x_5se_2(x_6+x_7)se_1x_8se_1x_9$

The total Consequences (CON) in the system can be defined as $CONC_{1_1}$ (= $se_1x_2se_2x_3se_3x_4se_2x_5se_2(x_6+x_7)se_1x_8se_1x_9$)

If the soft error occurs in C_1 within the 2nd and 8th messages then the consequences ($CONC_{1_2}$ = $se_1x_8se_1x_9$) can be propagated in the system after passing the 8th message.

If the soft error occurs in C_1 after passing the 8th message then the consequences ($CONC_{1_3}$ = se_1x_9) can be propagated in the system with the 9th message. If there is any indefinite conditional loops and number of iterations (at the exit time from the loop) is q then total consequence will be multiplied by q.

In the same way, if soft errors occur in C_2, and/or in C_3 then the increase in level of consequences can be checked at different stages of message passing.

The consequences in the system can be measured as follows.

$CONC_{2_1}$ = $se_2x_3se_3x_4se_2x_5se_2(x_6+x_7)se_1x_8se_1x_9$

$CONC_{2_2}$ = $se_2x_5se_2(x_6+x_7)se_1x_8se_1x_9$

$CONC_{2_3}$ = $se_2(x_6+x_7)se_1x_8se_1x_9$

$CONC_{2_4}$ = $se_2x_7se_1x_8se_1x_9$

$CONC_{3_1}$ = $se_3x_4se_2x_5se_2(x_6+x_7)se_1x_8se_1x_9$

The total propagation of soft errors in the three components due to injected fault(s) at each component can be derived as follows.

$$CONC_{1_T} = \sum_{i=1}^{3} CONC_{1i}$$

$$CONC_{2_T} = \sum_{i=1}^{4} CONC_{2i}$$

$$CONC_{3_T} = CONC_{3_1}$$

If the values of se_1, se_2, and se_3; and $x_1,\ldots, x_{10}$ are known then the above propagations can be derived. The total soft errors propagation in the whole system (due to a soft error in any component) can be shown as follows:

$$CON(C_i) = \sum_{k=1}^{n'} P(Sc_k) \times CON(C_{i_k}) \quad (5)$$

Where n' is the total number of scenarios in the system, $P(Sc_k)$ is the probability of the kth scenario, and $CON(C_{i_k})$ is the propagation of soft errors from the ith component in the kth scenario.

3.4 Measuring Criticalities of the Components

For each component, criticality is the product of complexity, severity, and the propagation of soft errors. The combined impact of complexity and severity is used to calculate the propagation of soft errors. Criticality is

calculated by taking the product of complexity, severity, and the propagation of soft errors. If the criticality of the ith component is Cr_i then the equation to derive it can be shown as (6).

$$Cr_i = \prod (OCC_i, CON(C_i), Sev(C_i)) \quad (6)$$

Where, OCC_i is the overall complexity of the component, $CON(C_i)$ is the propagation of soft errors from the component, and $Sev(C_i)$ is the severity of the component. $OCC_i, CON(C_i), Sev(C_i)$ are dependent on each other; i.e. for any increase in complexity there is a high probability that the severity will increase, and if the product of complexity and severity increases then the probability of propagation of soft errors will increase too. Hence, criticality is taken as the product of overall complexity, severity, and propagation of soft errors. The methodology of criticality analysis is shown in Fig. 2.

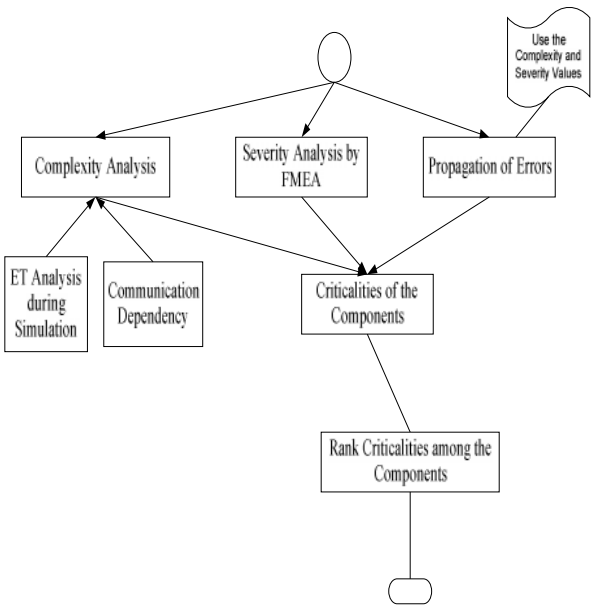

Fig.2. Methodology of Criticality Analysis

Once the criticalities of the components are calculated, they are ranked to obtain the criticality ranking among the components. The result is then analyzed to flag the components that, if affected by soft errors, can influence the system the most. This research then targets these components to lower their criticalities in order to minimize the risks of soft errors.

3.5 Lowering the Criticalities of Components

Component criticality suggests to the designers where in the system design changes are necessary or helpful to minimize soft errors risk. These changes can be done by applying a suitable approach where he/she may change the architecture or behavioural model of the component to lower its complexity, and/or severity, and/or propagation of soft errors. Refactoring is a good candidate for this type of approach.

In software engineering, "Refactoring" source code means improving it without changing its overall results, and is sometimes informally referred to as "cleaning it up". Refactoring neither fixes bugs nor adds new functionality, though the cleaning process might precede either of these activities. Rather, refactoring improves the understandability of the code, changes its internal structure and design, and removes dead code to make the code easier to comprehend, more easily maintained, and more amenable to change [57]. UML model refactoring is the

equivalent of source code refactoring at the model level, with the objective of preserving model behaviour [58]. Refactoring re-structures the model, at the conceptual level, to improve quality factors such as maintainability, efficiency and fault tolerance without introducing any new behaviour [59]. The purpose of model refactoring is to alter the model based on user's requirements by keeping the functionality and other constraints of the system unaffected. Model refactoring can be done by replacing components or sub-systems with ones that are more elegant, merging/splitting the states but keeping the behaviour unchanged, altering code readability or understandability, formal concept analysis, and graph transformation, etc. It can be detailed by using an example, which consists of Fig. 3 and Fig. 4. Fig. 3 shows an example statechart of user's access to server, and Fig. 4 shows this statechart after refactoring. Two states in the Fig. 3, named as "Pass to Sever" and "Retrieved" are merged into one state, "Verifying", in the refactored statechart (Fig. 4). The actions used in "Pass to Sever" are copied into the "Verifying" state.

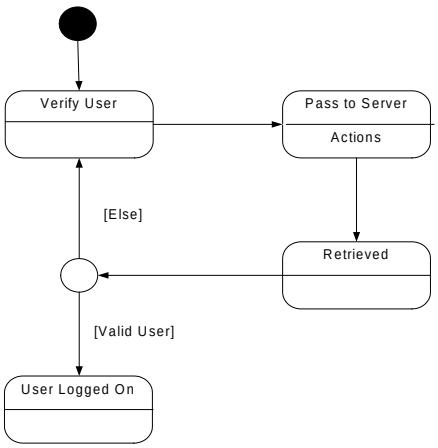

Fig. 3. An Example Statechart of 'User's Access to Server' before Refactoring

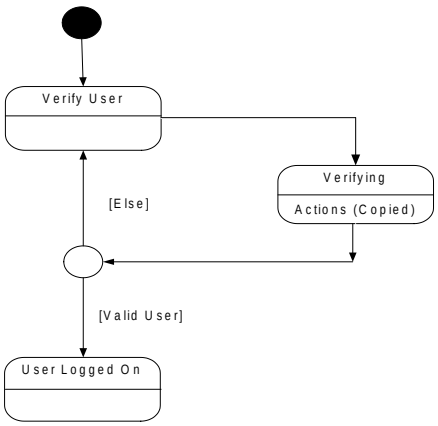

Fig. 4. An Example Statechart of 'User's Access to Server' after Refactoring

Once the criticality ranking is returned, a model can be refactored with the goal of reducing the criticalities of the components. Lowering the criticalities can be achieved by reducing any of the multiplying factors: complexity, severity or propagation of soft errors. For example, if in Fig. 2, C_1 is the most critical component; the reason behind its criticality is the product of its complexity, severity, and propagation of soft errors (from C_1). Then the design is analyzed to find which multiplying factor is causing the large criticality value. If its complexity is very high, then the reason is probably that its ET is high and/or its communication dependency with C_2, and C_3 is high. Higher values of ET imply that this component is being called more frequently than others are, and/or it is changing a greater number

of states than others. If the severity of C_1 shows a higher value, then it means the fault in it has more effect on the overall system functionality than either C_2 or C_3. If the large criticality is due to the propagation of soft errors from C_1, then that component is seen, in most of the communication scenarios, as a starting node. Refactoring can be applied on the architecture or behavioral model of the component to lower the complexity, and/or severity, and/or propagation of soft errors of the components. Fig. 5 details the methodology of lowering component criticality by refactoring.

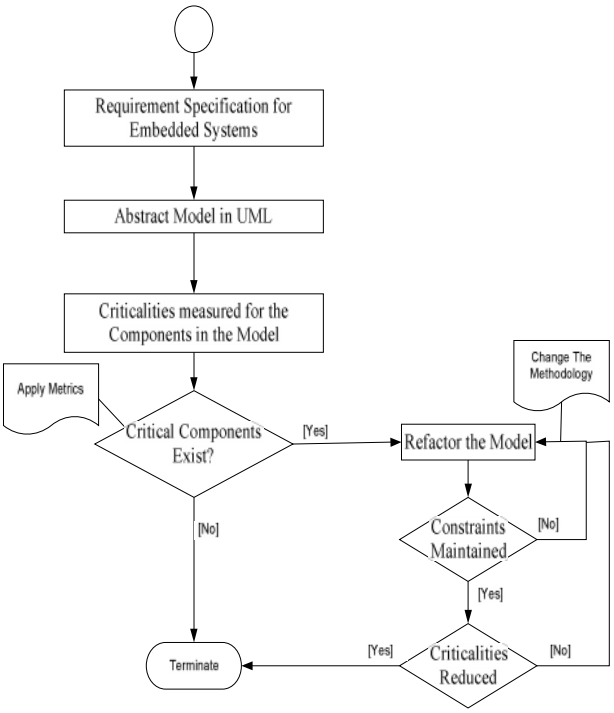

Fig. 5. Methodology to lower the criticalities of the Components by Refactoring

As shown in Fig. 5, initially, the abstract model (in UML) is created from the given specifications. The model is then analyzed to measure the criticalities of its components. Component criticalities need to be compared with a threshold value that users need to determine (for simplicity, the threshold value is ignored in the example). The large variations among components' criticalities are taken as the guideline of flagging the components as critical. If critical components exist in the model, then the model is analyzed to be refactored to lower the components' criticalities. Special attention needs to be given to the top-ranked components to lower their criticalities. Other components can be examined in turn later, according to their criticality ranking. Several trial and error iterations are needed to achieve the goal of lowering a component's criticality. In each trial, checks must be made to ensure the refactoring does not interfere with the functionality of the system; otherwise, the model will have to go through another refactoring method. If these constraints are maintained, then the lowering process will check whether components' criticalities are sufficiently reduced or not. If the check is successful, then the process will terminate. If not, another iteration of the above steps will occur.

4 CASE STUDY

Examination of a system model - an Automated Rail Car system - illustrates the application of the metrics. This model is a safety critical application, which meets real-time criteria and illustrates a broad class of systems that must have high reliability.

4.1 An Automated Rail Car System

The Automated Rail Car system assumes each pair of adjacent stations is connected by two rail tracks. Several railcars are available to transport passengers between terminals. A control centre receives, processes, and sends system data to various components of the system. Here, an example of such a system is selected where there are four terminals and eight cars. A high-level object-model diagram of the Automated Rail Car system and more-detailed diagrams of its composites are shown in Fig. 6.

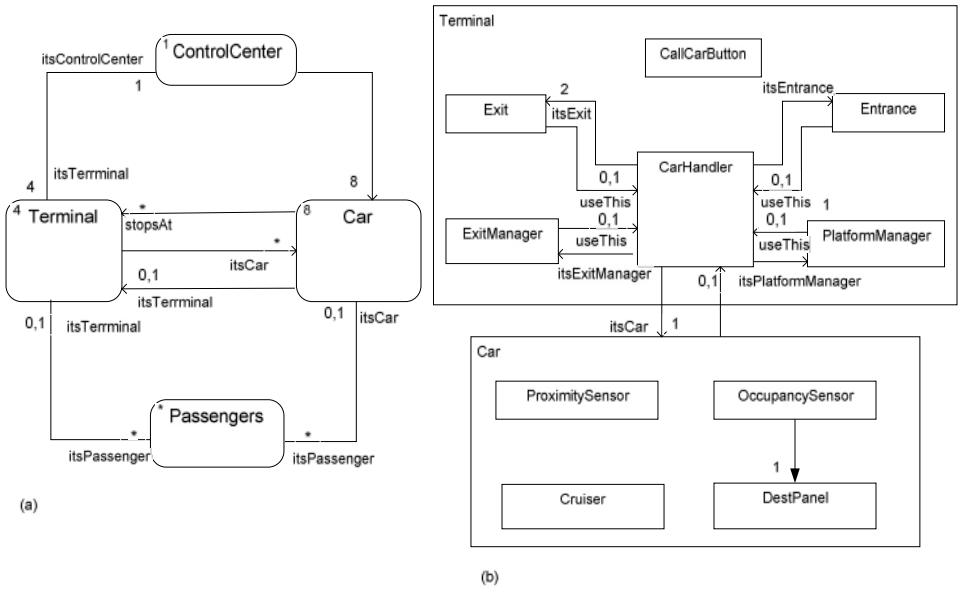

Fig. 6. (a) High-level object-model diagram for the Automated Rail Car system; and (b) more-detailed diagrams of the components - Terminal and Car.

A Car has four main components: ProximitySensor, Cruiser, DestPanel, and OccupancySensor; and a Terminal has six main components: CarHandler, PlatformManager, CallCarButton, Entrance, Exit, and ExitManager. There is another component in the system named as ControlCentre, which communicates with various system components for receiving, processing, and providing system data. Throughout this paper, components' names are abbreviated as follows: Car (Cr), ProximitySensor (Ps), Cruiser (Cs), DestPanel (Dp), OccupancySensor (Os), Terminal (Tm), CarHandler (Ch), CallCarButton (Cb), Entrance (En), Exit (Ex), ExitManager (Em), ControlCenter (Cc), PlatformManager (Pm). En and Ex are software drivers for the relevant rail segments. Pm and Em allocate platforms and exits to Ch. In contrast to other components, Ch is a concept that does not come from the problem domain, but is introduced by a domain expert during modeling. It handles the transactions between a Cr and a Tm. A special Ch is created whenever a Cr approaches a Tm; it is destroyed when the Cr departs. As such, it serves as a proxy object for the Cr within the Tm. The primitive component Pm sends events to the Cr's behavior (to its statechart) based on the distance to the approached Tm. The Cs can be off, engaged, or disengaged. The car is to maintain maximum speed as long as it never comes within 80 meters of any other car. Each Cr has its own Dp.

4.1.1 ET Analysis in the Automated Rail Car System

The state changes of the Cr used to measure the ET of the components in the ARCS, are shown in Fig. 7. Cr stays at 'Idle' state at any Tm. If the event is generated to move Cr from its source to destination then it reaches to its 'Departure' state where it continues its travel only if the smallest distance to any other car (in front) is at least 100 meters. Cr [0] is initially idled at Tm [0]. It is selected to move from its source (it is assumed as Tm [0]) to its destination (it is assumed as Tm [3]). Here, this test case is chosen randomly. The user can choose any possible test case. The system is executed to move Cr to the 'Idle' state. Then, the event to move Cr to Tm [3] (from Tm [0]) is

triggered. Since, before moving to 'Idle' state, some components change their behaviour, to calculate the ET, the time to move to the 'Idle' state is also considered, with the time to move to destination from the 'Idle' state. ET for each sub-system is calculated by (7) and this equation is derived from (1).

$$TR_p(\{\text{Automated Rail Car System}\}(\{\text{Car}[0]\text{ is at Rest at Terminal}[0]\} \rightarrow \{\text{Car}[0]\text{ is at Terminal}[3]\}) = \sum_{j=1}^{n} d_{pj} \quad (7)$$

Where, the symbols are defined in (1). Calculated ET for the different components to take Cr from Tm [0] to Tm [3] is shown in Table 2.

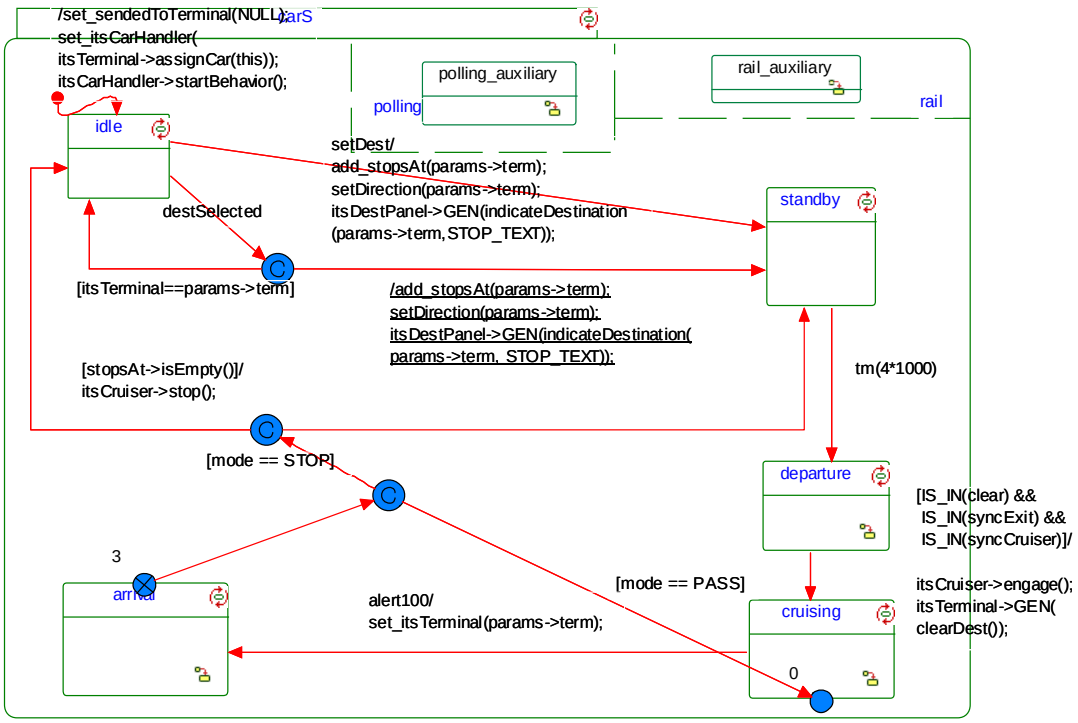

Fig. 7. Statechart Diagram of Car

TABLE 2

ET OF THE COMPONENTS TO MOVE A CAR FROM TERMINAL [0] TO TERMINAL [3]

Components	ET	Normalized Values
Cr	31s 45 ms	.933
Ps	47 ms	.023
Cs	15 ms	.0074
Dp	10 ms	.005
Os	8 ms	.004
Tm	3 ms	.0015
Ch	28 ms	.014
Cb	4 ms	.002
En	8 ms	.004
Ex	5 ms	.0025
Em	6 ms	.003
Pm	15 ms	.0074
Cc	1 ms	.0005

As mentioned in Section 3.1.1, UML does not specify an action model; Rational Rhapsody [106] is used to gain execution data via simulation. The model is executed in tracing mode. Several tracing commands are used to execute the model. The state transition times for the components are saved to a log file. At the end of the simulation, that log file is analysed to calculate the total ET of the components to perform a selected operation. In the execution time calculation, when a component communicates with other components, every slot of time taken by itself and in all in-between events on the way of communication till the message (In UML models, components communicate with each other by message passing) is received by the other end is taken into account.

4.1.2 MIO and TMIO Analysis in the Automated Rail Car System

There are two scenarios in this Automated Rail Car system. These are: i) CarApproachesATerminal, and ii) CarDepartsATerminal. Message-In-and-Out frequencies (MIO) for all components are calculated using (2) and (3) and for two different scenarios. All values of MIO and TMIO for all sub-systems are shown in Table 3.

TABLE 3
MIO, TMIO AND OVERALL COMPLEXITIES OF ALL COMPONENTS IN AUTOMATED RAIL CARS SYSTEM

Components	MIO in CAATS	MIO in CDATS	TMIO	OCC
Cr	0.64	0.5	1.14	2.073
Ps	0.21	-	0.21	0.233
Cs	0.29	0.21	0.5	0.5074
Dp	-	-	-	0.005
Os	-	-	-	0.004
Tm	0.14	0.07	0.21	0.2115
Ch	0.43	0.64	1.07	1.084
Cb	-	-	-	0.002
En	0.14	-	0.14	0.144
Ex	-	0.14	0.14	0.1425
Em	-	0.21	0.21	0.213
Pm	0.14	0.07	0.21	0.2174
Cc	-	-	-	0.0005

The probabilities of the two scenarios are assumed as equal since, if a Cr departs, there is an equal probability that it will arrive at another Tm. There may arise some other scenarios, but for simplicity only, these two scenarios are considered. The blank cells in Table 3 indicate no message from the corresponding component in the corresponding scenario. Cr has the highest value of TMIO as it communicates mostly with other components. TMIO for other components are also shown in this table.

4.1.3 Overall Complexities of the Components in the Automated Rail Car System

The total complexities of all components are calculated using (4). The last column in Table 3 shows the measured values of each component's complexity. Their values are normalized by taking the ratio between them and the total value. In Table 3, to simplify the representation of column headings, CarApproachesATerminal Scenario, CarDepartsATerminal Scenario, and Overall Complexities of the Components are abbreviated as CAATS, CDATS, and OCC respectively. As shown in Table 3, Cr is the most complex component and is followed by Ch, and Cs. The highest value of ET and largest communication dependencies (with other components) took the complexity value of Cr to the highest value. In the two scenarios, not all components participated. Hence, MIO and TMIO for some components were not measured in this example. ET alone is used to measure the overall complexities of those components.

TABLE 4
E/F RATIONS OF THE COMPONENTS IN THE AUTOMATED RAIL CAR SYSTEM

Components	Number of Trials	Number of Faults	E/F Ratio
Cr	10	8	0.8
Ps		4	0.4
Cs		5	0.5
Dp		1	0.1
Os		1	0.1
Tm		3	0.3
Ch		7	0.7
Cb		1	0.1
En		3	0.3
Ex		3	0.3
Em		4	0.4
Pm		4	0.4
Cc		1	0.1

4.1.4 Validating the Complexities of the Components in the Automated Rail Car System

To validate the component's complexity measurement, trials were conducted whereby transient faults were injected into the components of the selected system. The fault injection procedure is detailed in Section 3.2. The probabilities of occurrences of soft errors in the components are calculated by taking the ratios between total number of soft error occurrences and total number of fault injections. This ratio can be defined as the Error/Fault injections (E/F) Ratio. A fault, if activated in the execution of the system, creates an error in the system. In this paper, only those soft errors are counted that cause any degradation or failure in system functionality. If the fault does not create any degradation in the system then it was not taken as a matter of concern here. Ten trials were made for transient fault injection into every component. The more trials are performed, the better the expected result. However, for this large example model, it is expected that ten trials in each component can give a good idea about their probabilities of occurrences of soft errors. Table 4 shows the E/F ratio for this example. If these ratios are ranked in an ascending order then it is observed that Table 3 has a similar ranking to Table 4 until the Cs component. The next ratio is equal for Ps, Pm, and Em where, in Table 3, their complexity values differ a little. If that slight difference is neglected, then the complexity ranking for these components shows similar results in these tables. Other results also show a very similar complexity order as in Table 3. Hence, it can be concluded that complexity analysis is able to measure the likelihood of error occurrences among the components of the Automated Rail Car system.

4.1.5 Comparison with Static Complexity Analysis

This comparison shows the contribution of dynamic metrics to measure the complexities of the components over static metrics. The dynamic complexities of the components for the Automated Rail Car system are obtained as outlined in Section 3.1.3. Static complexities are calculated by using McCabe's Cyclomatic complexity theorem [46]. The last Column in Table 5 shows the Cyclomatic Complexities of the components of the Automated Rail Car system. Table 5 also allows comparison among the dynamic complexity, static complexity, and E/F ratios of the components. E/F ratios of the components are calculated as described in Section 4.1.4. The values shown in Table 5 are all normalized to make the comparison possible. The normalization is done by dividing each element in each column with the highest value in the corresponding column. Since in all three columns Cr has the maximum value, the normalized value for Cr is obtained as '1'. The results show that both dynamic complexity and E/F ratio return a similar ranking. Static complexity, on the other hand, returned a completely different ranking and, in most cases, it failed to distinguish among the complexities of the components. Static complexity analysis, for instance, returned the same complexity value for Dp, Os, Tm, Cb, En, Ex, Em, Pm, Cc. Hence, dynamic complexity is more significant than

static complexity in component complexity analysis.

TABLE 5
COMPARISON AMONG DYNAMIC COMPLEXITY, STATIC COMPLEXITY, AND E/F RATIONS OF THE COMPONENTS OF THE AUTOMATED RAIL CAR SYSTEM

Components	E/F Ratio (Normalized)	Dynamic Complexities of the Components (Normalized)	Static Complexities (Normalized)
Cr	1	1	1
Ps	0.5	0.112	0.625
Cs	0.625	0.245	0.75
Dp	0.125	0.002	0.25
Os	0.125	0.0019	0.25
Tm	0.375	0.102	0.25
Ch	0.875	0.523	0.375
Cb	0.125	0.001	0.25
En	0.375	0.069	0.25
Ex	0.375	0.069	0.25
Em	0.5	0.103	0.25
Pm	0.5	0.105	0.25
Cc	0.125	0.0002	0.25

4.1.6 Measuring the Severity of Components in the Automated Rail Car System

Severity of components and messages were determined by the FMEA method, where the effects of soft errors (here soft errors that degrade the system functionality are considered) in each component were analyzed by injecting transient faults (fault injections detail is shown in Section 3.2). Table 6 shows the FMEA on the components of this example.

TABLE 6
FMEA ON THE COMPONENTS OF AUTOMATED RAIL CAR SYSTEM

CN	Failure Mode	Effect of Failure	Severity of Failure	SR
Cr	Failed to poll on the Cr	The system was held in between Departure and Cruising states	Serious	9
Ps	Calculation of distance was wrong	The system could not arrive at any terminal and was continuously cruising	Extreme	8
Cs	Wrong speed was calculated (low rather than Full)	The system became very slow and took much more time to reach the destination.	Low	4
Ch	Direction parameter changed (Entrance to exit)	The system was running till in cruise, then the system terminated the whole execution	Hazardous	10
Cb	No Failure (Error injected at 'Calling car method')	No change in the selected operation of the system	No effect	1
Os	Occupancy was True rather than False	Failed to Depart	Extreme	8
Pm	Failed to provide free terminal at destination	The system could not reach at the destination and it was infinitely looping at Arrival	Significant	6
Tm	Calculation of destination terminal was wrong. (Error injected at Reactions in State)	Failed to reach at the destination though it started journey from source terminal, cruising was OK and looping infinitely at Arrival state	Significant	6
Cc	Several transient faults were injected at different places	No change in the selected operation of the system	No effect	1
Dp	Several transient faults were injected at different places	No change in the selected operation of the system	No effect	1
En	Failed to send Parameter to Ch to complete Move operation	Held at Departure state	Extreme	8
Ex	Failed to send Parameter to Ch to complete Move operation	Could not exit from the first Tm	Serious	9
Em	Exit's counter clock wise direction was made 'FALSE'	Could not exit from the first Tm	Serious	9

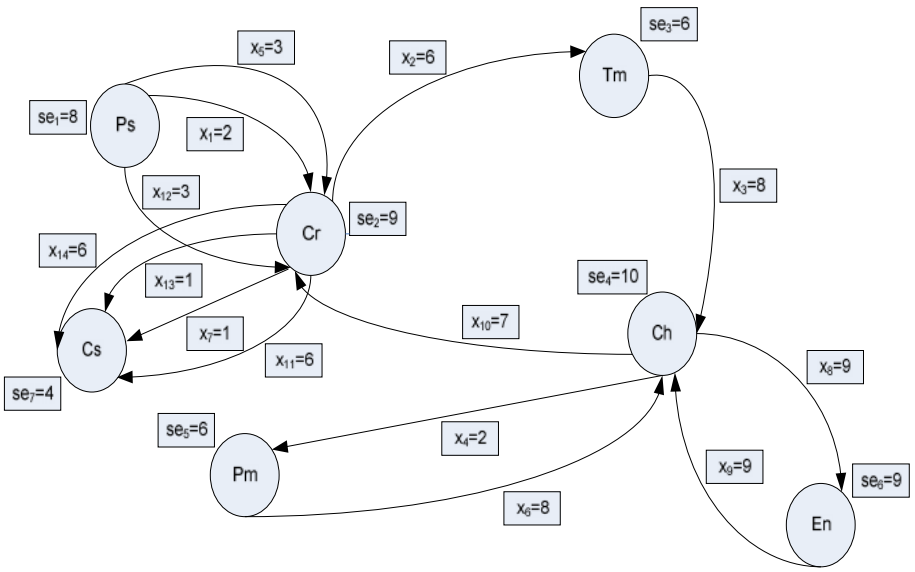

Fig. 8. Severity of components and messages in CarApproachesATerminal Scenario

To measure the severities of communicating messages, transient faults were injected at each message individually and then the consequences arising from the soft errors (as mentioned in Section 1, soft errors are the effect of transient faults) were checked and evaluated at execution. The severities of the messages in CarApproachesATerminal Scenario are shown in Fig. 8. The values are not shown in tabular form. A few titles of the columns in Table 6 are abbreviated as Component Name (CN) and Severity Rank (SR). The eighth and ninth messages in Fig. 8, which are messages from Ch to En, and En to Ch respectively, have the highest severities among the severities of the communicating messages. The second level of severity is with the third and sixth messages. The lowest severity rank is with the seventh and thirteenth messages.

4.1.7 Measuring Propagation of Soft Errors from the Components in the Automated Rail Car System

Propagations of soft errors from the components are calculated as outlined in Section 3.3. The second column in Table 7 shows the combined impact of complexity and severity.

TABLE 7
CRITICALITIES OF THE COMPONENTS IN THE AUTOMATED RAIL CAR SYSTEM

Components	CICS	Propagation of Soft Errors	Criticality
Cr	18.657	4.1	76.4937
Ps	1.864	10	18.64
Cs	2.0296	.000000058	0.000000117717
Dp	0.005	-	0.005
Os	0.032	-	0.032
Tm	1.152	.0017	0.0000000072576
Ch	10.84	2.1	22.764
Cb	0.002	-	0.002
En	1.152	.0000000063	0.0000000072576
Ex	1.2825	.000014	0.000017955
Em	1.917	.0013	0.0024921
Pm	1.3044	.0000064	0.00000834816
Cc	0.0005	-	0.0005

The column heading: ‘Combined Impact of Complexity and Severity’ is abbreviated as CICS. This combined impact is used to calculate the propagation of soft errors for both of the scenarios in this example. They are shown (as normalized values) in the third Column of Table 7. Since, in these two scenarios, participation of all components is not considered, only the participant component’s error propagations can be derived using the methods outlined in Section 3.3. The corresponding cells for Dp, Os, Cb, and Cc are blank in the table since they did not participate in the scenarios. Table 7 shows that Ps has the highest value for the propagation of soft errors, followed by Cr and Ch. Ps started the communication among the components, and the flow of the first message was towards almost all other components. On the other hand, Cs, and En had negligible propagation of soft errors since their place in the scenario is almost at the end and so, they communicated with very few components.

4.1.8 Measuring Criticalities of the Components in the Automated Rail Car System

Component criticality is calculated by taking the product of complexity, severity, and propagation of soft errors. In the case of absence of propagation of soft errors, the combined impact of complexity and severity were deemed final criticalities. Component criticality is shown in the last column of Table 7. The results show that Cr is the most critical component followed by Ch and Ps. The reason behind having the maximum value is that the product of complexity, severity, and propagation of soft errors of Cr is the highest. Cs, En, and Pm have negligible propagation of soft errors. Cs is a highly complex component and En has a high severity value, but their relative positions in the two communication scenarios are at the end, and hence the probability of the propagation of soft errors towards other components is less. The results also show the other rankings of criticalities among the components.

4.1.9 Effectiveness of Propagation of Soft Errors in Measuring Criticalities of the Components

The comparison between criticalities, with and without considering the propagation of soft errors of the components, is shown in Fig. 9 to validate the effectiveness of propagation of soft errors in measuring criticalities of the components. Component criticality without considering propagation of soft errors is calculated by the product of complexity and severity only. As shown in Fig. 9, if the propagation of soft errors is not considered then it fails to distinguish criticalities among Ps, Cs, Tm, En, Ex, Em, and Pm.

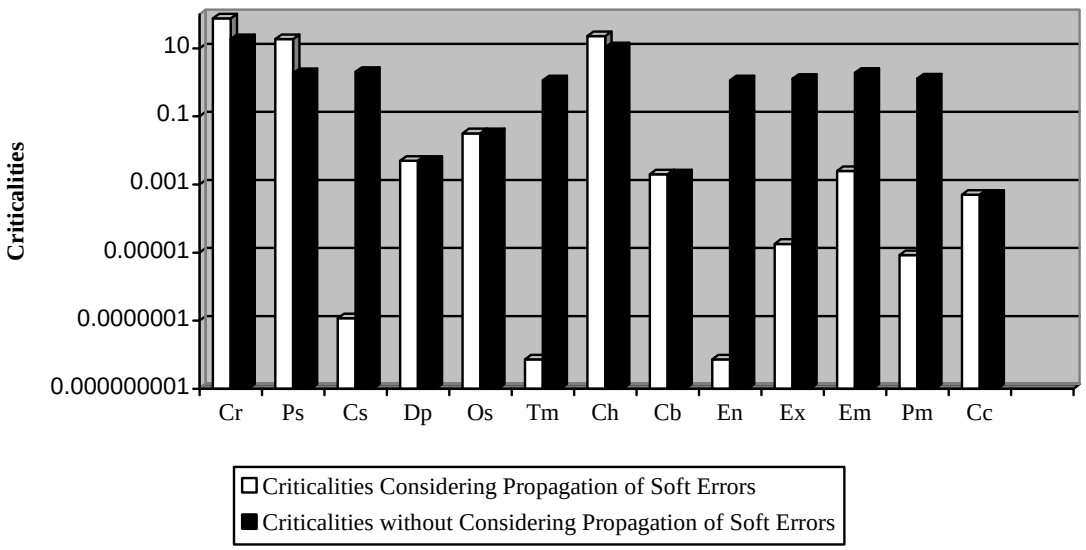

Fig. 9. Comparison between Criticalities with and without considering Propagation of soft errors

Criticality considering propagation of soft errors, on the other hand, is able to return those components’ criticality

ranking. Usually, if a soft error arises within a component then its effect may continue to all following connected components. A soft error in the initial components will therefore tend to have a more devastating effect on system functionalities. Hence, the probability of the consequences of Ps, which is a starting node in the CarApproachesATerminal Scenario, cannot be equal to the probability of the consequences of Tm or Em, which are at the end of propagation.

TABLE 8
COMPARISON AMONG THE CALCULATED NORMALIZED ET OF THE COMPONENTS OF REFACTORED MODEL AND EXISTING MODEL

Components	Normalized ET of Refactored Model	Normalized ET of Existing Model
Cr	0.899	.995
Ps	0.00051	.0015
Cs	.00048	.00048
Dp	.00016	.00032
Os	0.00035	.00026
Tm	.00013	.000096
Ch	.00089	.00089
Cb	.000032	.00013
En	0.00022	.00026
Ex	.00016	.00016
Em	.000096	.000192
Pm	0.00064	.00048
Cc	.000032	.000032

4.1.10 Lowering the criticalities of the components in the Automated Rail Car System

As shown in Table 7, Cr is the most critical component and it has large criticality differences with the second most critical and other components. Hence, the first target for reduction of its criticality is Cr. The part of the model that deals with the behaviour of Cr was carefully examined to determine refactoring possibilities. All the states and their internal and/or external codes used in triggers, as well as actions, were checked. These areas were where refactoring could achieve the goal of reducing Cr's time complexity while keeping its functionality, performance and other constraints unaffected. Two states (Activity and Departure states in Fig. 7) and their internal codes were merged to a single state to reduce the time complexity without affecting the functionality, and performance of the system. Since refactoring is done by merging or splitting states etc. in this case, it will have negligible impact on power consumption. Comparison among the calculated normalized ET of the components of the refactored model and existing model (to take a Cr from Tm [0] to Tm [3]) is shown in Table 8. A lower ET will result in lower complexity as well as lower criticality of the components. Table 8 shows that refactoring the model lowered the ET of Cr and Ps to a measurable extent. Others, except for Os and Pm, were also lowered. The criticalities of Os and Pm were not so high, and the increases in their criticalities due to refactoring the model were not large. For this reason, the increases in these two components can be viewed as negligible. In summary, applying refactoring is effective in lowering the criticalities of the components.

5 CONCLUSIONS AND FUTURE WORK

This paper presents significant steps in developing metrics of a complexity analysis for the early system design phase based on UML artifacts. The paper develops a severity assessment methodology by analyzing UML model simulation results, develops a methodology to measure the propagation of soft errors of the components, returns a criticality ranking among the components, and develops possible ways to encourage the designer to explore changes

that could be made in the existing model to lower the criticalities of the components.

Two heuristics based metrics are developed to measure the complexity of a component: assessment of execution time during simulation and Message-In-and-Out frequency. Both of the metrics are obtained from the UML specifications that can be used in the early design phase of a system. These are dynamic metrics, i.e., working on the execution phase of the model. Developed metrics are validated by calculating the E/F ratio in the component.

To measure the severity of a component, transient faults were injected at each component and the effects were analyzed using the FMEA method during UML model simulation. The resultant severity is the average of several tests applied on each component.

To measure the propagation of soft errors from the components, all possible cases of soft errors propagation were considered and probabilistic formulae were developed. The mathematical figures returned from the application of those probabilistic formulae show the propagation of soft errors from the components.

This paper integrates the above three analyses to rank the component's criticality that highlight the variations of the impact of soft errors among the components.

This paper then shows how possible changes can be made in the existing design to lower the criticalities of the components to minimize the risk of soft errors. The main contribution of this step is to flag how corrective measures can be adopted to minimize the soft errors risk of the system. The corrective measures are illustrated in the case study, in which the criticalities of the top-ranked (critical) components could be minimized or how these components could be replaced with less critical components. The objective is to keep the functionality and other constraints of the system unaffected or to make a trade off between them, with a goal to minimize the risks of soft errors.

In summary, the approach presented in this paper is effective in measuring the soft error risk of the components in a system and in lowering the criticalities of components to minimize the risk of functional degradation.

Some possible steps could be taken to extend this paper. The followings are suggested as future work.

This paper deals with the criticality of the individual components. These criticalities are relative to the system. The top most critical component in one system may not be a critical component when considered among the domain of all (or many) systems. Therefore, measures are still needed to determine the criticality of the whole system. Then, lowering the criticality could occur across the whole system. Another direction for further research is to investigate which weights might be more appropriate for the two different complexity factors used in equation (4), since in the current approach, the weights of ET and TMIO in measuring total value of complexities were assumed as equal. However, in reality, it is not the case. Hence, steps are required to find out the appropriate weights of ET and TMIO in measuring the complexity of each component. The current work claims that the top-ranked critical components can create a severe impact in the system if they are affected by soft errors. This is a relative measure. There should be a threshold value of a component's criticality that could flag whether any component is crossing the criticality boundary or not. In addition, these threshold values should be universal, and able to be applied to the early design phase of any system.

REFERENCES

[1] A. Timor, A. Mendelson, Y. Birk, and N. Suri, "Using under utilized CPU resources to enhance its reliability," Dependable and Secure Computing, IEEE Transactions on, vol. 7, no. 1, pp. 94-109, 2010.

[2] E. L. Rhod, C. A. L. Lisboa, L. Carro, M. S. Reorda, and M. Violante, "Hardware and Software Transparency in the Protection of Programs Against SEUs and SETs," Journal of Electronic Testing, vol. 24, pp. 45-56, 2008.

[3] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: an architectural perspective," in 11th International Symposium on High-Performance Computer Architecture, San Francisco, CA, USA, pp. 243 - 247, 2005, pp. 243-7.

[4] R. K. Iyer, N. M. Nakka, Z. T. Kalbarczyk, and S. Mitra, "Recent advances and new avenues in hardware-level reliability support," Micro, IEEE, vol. 25, pp. 18-29, 2005.

[5] V. Narayanan and Y. Xie, "Reliability concerns in embedded system designs," Computer, vol. 39, pp. 118-120, 2006.

[6] S. Tosun, "Reliability-centric system design for embedded systems," Ph.D. Thesis, Syracuse University, United States -- New York, 2005.

[7] G. P. Saggese, N. J. Wang, Z. T. Kalbarczyk, S. J. Patel, and R. K. Iyer, "An experimental study of soft errors in microprocessors," Micro, IEEE, vol. 25, pp. 30-39, 2005.

[8] Y. Crouzet, J. Collet, and J. Arlat, "Mitigating soft errors to prevent a hard threat to dependable computing," in 11th IEEE International On-Line Testing Symposium, IOLTS, pp. 295-298, 2005, pp. 295-298.

[9] V. Cortellessa, K. Goseva-Popstojanova, K. Appukkutty, A. R. Guedem, A. Hassan, R. Elnaggar, W. Abdelmoez, and H. H. Ammar, "Model-based performance risk analysis," IEEE Transactions on Software Engineering, vol. 31, pp. 3-20, 2005.

[10] J. Jurjens and S. Wagner, "Component-based development of dependable systems with UML," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 3778 NCS, pp. 320-344, 2005.

[11] M. D. C. A. Bondavalli, D. Latella, I. Majzik, A. Pataricza, and G. Savoia, "Dependability Analysis in the Early Phases of UML Based System Design," Journal of Computer Systems Science and Engineering, vol. 16, pp. 265--275, 2001.

[12] J. M. a. T. Khoshgoftaar, "Software Metrics for Reliability Assessment," Handbook of Software Reliability Eng., M. Lyu ed., chapter 12, pp. 493-529, 1996.

[13] S. M. Yacoub and H. H. Ammar, "A methodology for architecture-level reliability risk analysis," IEEE Transactions on Software Engineering, vol. 28, pp. 529-547, 2002.

[14] S. K. Wood, D. H. Akehurst, O. Uzenkov, W. G. J. Howells, and K. D. McDonald-Maier, "A model-driven development approach to mapping UML state diagrams to synthesizable VHDL," IEEE Transactions on Computers, vol. 57, pp. 1357-1371, 2008.

[15] L. Wang, E. Wong, and D. Xu, "A threat model driven approach for security testing," Piscataway, NJ 08855-1331, United States, 2007, p. 4273336.

[16] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt, "Detailed design and evaluation of redundant multi-threading alternatives," in 29th Annual International Symposium on Computer Architecture, pp. 99-110, 2002, pp. 99-110.

[17] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," in 27th International Symposium on Computer Architecture, 2000, pp. 25-36.

[18] E. Rotenberg, "AR-SMT: a microarchitectural approach to fault tolerance in microprocessors," in 29th Annual International Symposium on Fault-Tolerant Computing, 1999, pp. 84-91.

[19] J. C. Smolens, B. T. Gold, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk, "Fingerprinting: Bounding soft-error detection latency and bandwidth," New York, NY 10036-5701, United States, 2004, pp. 224-234.

[20] T. N. Vijaykumar, I. Pomeranz, and K. Cheng, "Transient-fault recovery using simultaneous multithreading," in 29th Annual International Symposium on Computer Architecture, 2002, pp. 87-98.

[21] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Error detection by duplicated instructions in super-scalar processors," Reliability, IEEE Transactions on, vol. 51, pp. 63-75, 2002.

[22] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August, "SWIFT: software implemented fault tolerance," Los Alamitos, CA, USA, 2005, pp. 243-54.

[23] Y. Xie, L. Li, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Reliability-aware co-synthesis for embedded systems," in 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2004, pp. 41-50.

[24] J. Ray, J. C. Hoe, and B. Falsafi, "Dual use of superscalar datapath for transient-fault detection and recovery," in 34th ACM/IEEE International Symposium on Microarchitecture, 2001, pp. 214-224.

[25] C. L. Chen and M. Y. Hsiao, "Error-Correcting Codes for Semiconductor Memory Applications: A State-Of-The-Art Review," IBM Journal of Research and Development, vol. 28, pp. 124-134, 1984.

[26] K. R. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic prediction of architectural vulnerability from microarchitectural state," New York, NY 10016-5997, United States, 2007, pp. 516-527.

[27] A. Shye, J. Blomstedt, T. Moseley, V. Janapa Reddi, and D. Connors, "PLR: A Software Approach to Transient Fault Tolerance for Multi-Core Architectures," Dependable and Secure Computing, IEEE Transactions on, To be Appeared.

[28] J. K. Park and J. T. Kim, "A soft error mitigation technique for constrained gate-level designs," IEICE Electronics Express, vol. 5, pp. 698-704, 2008.

[29] N. Miskov-Zivanov and D. Marculescu, "MARS-C: modeling and reduction of soft errors in combinational circuits," Piscataway, NJ, USA, 2006, pp. 767-72.

[30] Z. Quming and K. Mohanram, "Cost-effective radiation hardening technique for combinational logic," Piscataway, NJ, USA, 2004, pp. 100-6.

[31] Oma, M. a, D. Rossi, and C. Metra, "Novel Transient Fault Hardened Static Latch," Charlotte, NC, United states, 2003, pp. 886-892.

[32] P. R. STMicroelectronics Release, "New chip technology from STmicroelectronics eliminates soft error threat to electronic systems," Available at www.st.com/stonline/press/news/year2003/t1394h.htm, 2003.

[33] L. R. Rockett, Jr., "Simulated SEU hardened scaled CMOS SRAM cell design using gated resistors," IEEE Transactions on Nuclear Science, vol. 39, pp. 1532-41, 1992.

[34] M. Z. S. Mitra, N. Seifert, TM Mak and K. Kim. Soft and IFIP, "Soft Error Resilient System Design through Error Correction," VLSI-SoC, January, 2006.

[35] M. Zhang, "Analysis and design of soft-error tolerant circuits," Ph.D. Thesis, University of Illinois at Urbana-Champaign, United States -- Illinois, 2006.

[36] M. Zhang, S. Mitra, T. M. Mak, N. Seifert, N. J. Wang, Q. Shi, K. S. Kim, N. R. Shanbhag, and S. J. Patel, "Sequential Element Design With Built-In Soft Error Resilience," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 14, pp. 1368-1378, 2006.

[37] T. M. Austin, "DIVA: a reliable substrate for deep submicron microarchitecture design," in 32nd Annual International Symposium on Microarchitecture, 1999, pp. 196-207.

[38] P. J. Meaney, S. B. Swaney, P. N. Sanda, and L. Spainhower, "IBM z990 soft error detection and recovery," Device and Materials Reliability, IEEE Transactions on, vol. 5, pp. 419-427, 2005.

[39] B. T. Gold, J. Kim, J. C. Smolens, E. S. Chung, V. Liaskovitis, E. Nurvitadhi, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk, "TRUSS: a reliable, scalable server architecture," Micro, IEEE, vol. 25, pp. 51-59, 2005.

[40] S. Krishnamohan, "Efficient techniques for modeling and mitigation of soft errors in nanometer-scale static CMOS logic circuits," Ph.D. Thesis, Michigan State University, United States -- Michigan, 2005.

[41] A. G. Mohamed, S. Chad, T. N. Vijaykumar, and P. Irith, "Transient-fault recovery for chip multiprocessors," IEEE Micro, vol. 23, p. 76, 2003.

[42] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," in 31st Annual International Symposium on Computer Architecture, 2004, pp. 276-287.

[43] M. W. Rashid, E. J. Tan, M. C. Huang, and D. H. Albonesi, "Power-efficient error tolerance in chip multiprocessors," Micro, IEEE, vol. 25, pp. 60-70, 2005.

[44] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," IEEE Transactions on Software Engineering, vol. 20, pp. 476-93, 1994.

[45] R. Harrison, S. J. Counsell, and R. V. Nithi, "An evaluation of the MOOD set of object-oriented software metrics," IEEE Transactions on Software Engineering, vol. 24, pp. 491-6, 1998.

[46] T. J. McCabe, "A Complexity Measure," Software Engineering, IEEE Transactions on, vol. SE-2, pp. 308-320, 1976.

[47] J. B. Bowles, "An assessment of RPN prioritization in a failure modes effects and criticality analysis," Journal of the IEST, vol. 47, pp. 51-6, 2004.

[48] N. author, "Procedures for Performing failure Mode Effect and Criticality Analysis," in US MIL_STD_1629 Nov. 1974, US MIL_STD_1629A Nov. 1980, US MIL_STD_1629A/Notice 2, Nov.1984, 1984.

[49] J. B. Bowles, The new SAE FMECA standard Annual Reliability and Maintainability Symposium. 1998 Proceedings. International Symposium on Product Quality and Integrity, 1998.

[50] Sherer, "Methodology for the Assessment of Software Risk," PhD Thesis, Wharton School, University of Pennsylvania, 1988.

[51] H. H. Ammar, T. Nikzadeh, and J. B. Dugan, "A methodology for risk assessment of functional specification of software systems using colored Petri nets," in Fourth International Software Metrics Symposium, Los Alamitos, CA, USA, 1997, pp. 108-17.

[52] H. T. Nguyen, Y. Yagil, N. Seifert, and M. Reitsma, "Chip-level soft error estimation method," Device and Materials Reliability, IEEE Transactions on, vol. 5, pp. 365-381, 2005.

[53] Rational Rhapsody, http://www.ibm.com/developerworks/rational/products/rhapsody/

[54] M. Hitz and B. Montazeri, "Measuring product attributes of object-oriented systems," in Proceedings of the 5th European Software Engineering Conference, ESEC'95, Sep 25-28 1995 Berlin, Germany: Springer-Verlag GmbH & Company KG, 1995, p. 124.

[55] P. E. Evans, "Failure mode effects and criticality analysis," Los Angeles, CA, USA, 1987, pp. 168-71.

[56] S. M. Seyed-Hosseini, N. Safaei, and M. J. Asgharpour, "Reprioritization of failures in a system failure mode and effects analysis by decision making trial and evaluation laboratory technique," Reliability Engineering & System Safety, vol. 91, pp. 872-81, 2006.

[57] Wikipedia, http://en.wikipedia.org/wiki.

[58] S. Hosseini and M. A. Azgomi, "UML model refactoring with emphasis on behavior preservation," Piscataway, NJ 08855-1331, United States, 2008, pp. 125-128.

[59] S. Gerson, P. Damien, T. Yves Le, J. Jean-Marc, z, and quel, "Refactoring UML Models," in Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools: Springer-Verlag, 2001.