A Model-Based Soft Errors Risks Minimization Approach

Muhammad Sheikh Sadi, D. G. Myers, Cesar Ortega Sanchez
Department of Electrical and Computer Engineering
Curtin University of Technology, Australia

Jan Jurjens
Department of Computer Science
TU Dortmund
Dortmund, Germany

Abstract—**Minimizing the risk of system failure in any computer structure requires identifying those components whose failure is likely to impact on system functionality. Clearly, the degree of protection or prevention required against faults is not the same for all components. Tolerating soft errors can be much improved if critical components can be identified at an early design phase and measures are taken to lower their criticalities at that stage. This improvement is achieved by presenting a criticality ranking (among the components) formed by combining a prediction of faults, consequences of them, and a propagation of errors at the system modeling phase; and 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.**

Keywords-Criticality Analysis; Soft Errors; Reliability Risks, UML Model; Metrics

I. 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 soft errors. Soft errors cannot create a physical damage to a chip but can be catastrophic for the desired functionalities of the system [1], [2]. Specifically, they are a matter of concern in those systems where high reliability is a necessity [3]. 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 a soft error is 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 [4], [5].

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 [6]. Estimating reliability (or at least identifying failure-prone components) early in the life-cycle of a design is therefore preferable [7], [8]. Ideally, this should be at the system design level so that the designer can create required prevention or detection mechanisms in the detailed design that follows. From a pure dependability viewpoint, critical components attract more attention of protection/prevention mechanisms than others do since reliability of a system is correlated with the criticality of the system [9], [10]. 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 the 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.

II. Related Work

Researchers have evolved several measures to prevent soft errors. Much less attention has been dedicated, until now, to the integration of design processes with reliability verification techniques. Rather, a "fix-it-later" approach is still dominant. At a system level, duplicating hardware [11], [12] and then comparing the results, and/or executing several copies of software by using the same hardware [13] to detect soft errors are the most common approaches. Then, different recovery approaches are employed to recover from the soft errors. At the circuit level, the solution is mainly to increase the critical charge of a circuit node [14]. Logic level solutions [5] mainly propose detection and recovery in combinational circuits by using redundant or self-checking circuits. Gold et al. [15] proposed distributed shared memory multi-processor features that incorporate computation and memory storage redundancy to detect and recover from a single point of transient or permanent failure. Mohamed et al. [16] shows chip level redundant threading with recovery, where the basic idea is to run each program twice, as two identical threads, on a simultaneous multithreaded processor. These post-functional design phase approaches are costly as well as complex to implement. Moreover, they can increase time delays and power overhead without offering any performance gain. Timing and synchronizing issues are also matters of great concern in these approaches.

Few approaches [17], [18] dealt with the static complexities of the system as a risk assessment methodology to minimize the risks of faults. 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. [6] and Yacoub et al. [10] 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.

III. 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 failure 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.

A. Measuring the Complexities of the Components

There is a correlation between the likelihood of soft errors proneness and the complexity of a system [9], [19]. Complexity analysis does not measure the impact of components in system functionality, but it shows the rank of likelihood of soft error proneness 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).

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 errors if the circuit type, transistor sizes, node capacitances, temperature etc. are kept at constant [20]. The longer duration to perform the selected operation implies that the component is being used more frequently and/or it is experiencing many state changes. A soft error occurs at any access point and/or in any behavioral change of these components can spread towards all communicating components through the large number of behavioral linkages until the soft error affected component is in execution. 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), can be shown as (1).

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

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, Telelogic Rhapsody [21] 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.

2) Message-In-and-Out Frequency

In a model-based system, components are often interdependent. They communicate with each other by message 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 . To figure out this fault proneness, a component's 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. Define MIO_{i_k} as the MIO for ith component in kth scenario. 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, $\mathrm{MIO_i}$ can be derived as shown in (2).

$$\mathrm{MIO}_{\mathrm{i_k}} = \frac{\left| \sum_{j=1}^{m} \mathrm{M}_{(\mathrm{i,\,j})} \mid i \neq j \right|}{n_k} \quad . \qquad (2)$$

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

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

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) Overall Complexity

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

$$OCC_i = ET_i + TMIO_i \qquad (4)$$

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. For simplicity, the weights of ET and TMIO in measuring total value of complexities are assumed as equal.

B. *Measuring the Severity of the Failure of the 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 is determined by the Failure Mode and Effects Analysis (FMEA) method [22]. 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. Hosseini et al. [22] suggested evaluation criteria and a ranking system for the severity of effects for a design FMEA as shown in TABLE I. Transient faults are 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 transient faults are almost zero.

TABLE I. 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

C. *Measuring Propagation of Failure from the Components*

Before measuring the component's propagation of failure, its complexity and severity are multiplied together to measure there combined impact (if there is any soft error) on the whole system. Measuring the propagation of failure refines this impact to obtain a clearer picture of the impact or criticality of each component. The method of measuring the propagation of failure is shown in Fig. 1, which is a scenario of a system model showing three components: C_1, C_2, and C_3. ENV denotes the environment communicating with the system. The product of complexity and severity of these three components are s_1, s_2, and s_3 respectively.

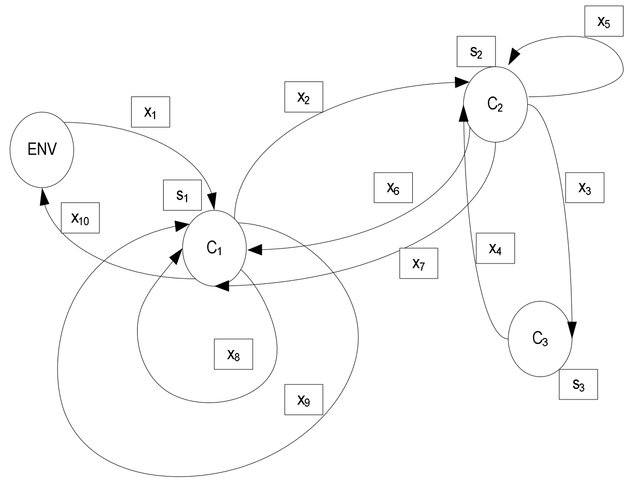

Fig. 1. An Example Scenario of a System Model to Measure the Propagation of failure

In Fig. 1, $x_1,\ldots,x_{10}$ indicate the severity in corresponding messages where indexing is made according to their time of occurrences in the whole scenario. Failures due to soft errors may be propagated via message communication. The propagation of failure from or in the environment is not considered. To measure propagation of failure 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 s_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: $s_1x_2s_2x_3$. Similarly, after passing the 9th message, there is an increase in the level of consequences in C_1: $s_1x_2s_2x_3s_3x_4s_2x_5s_2(x_6+x_7)s_1x_8s_1x_9$

The total consequences in the system can be defined as $CONC_{1_1}$ (= $s_1x_2s_2x_3s_3x_4s_2x_5s_2(x_6+x_7)s_1x_8s_1x_9$)

If the soft error occurs in C_1 within the 2nd and 8th messages then the consequences ($CONC_{1_2}$ = $s_1x_8s_1x_9$) 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 ($CONC_{1_3}$ = s_1x_9) can be propagated in the system with the 9th message. 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.

$$CONC_{2_1} = s_2x_3s_3x_4s_2x_5s_2(x_6+x_7)s_1x_8s_1x_9$$

$$CONC_{2_2} = s_2x_5s_2(x_6+x_7)s_1x_8s_1x_9$$

$$CONC_{2_3} = s_2(x_6+x_7)s_1x_8s_1x_9$$

$$CONC_{2_4} = s_2x_7s_1x_8s_1x_9$$

$$CONC_{3_1} = s_3x_4s_2x_5s_2(x_6+x_7)s_1x_8s_1x_9$$

The total propagation of failure from each component (due to a soft error in that 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 s_1, s_2, and s_3; and $x_1,\ldots, x_{10}$ are known then the above propagations can be derived. The total propagation of failure in the whole system (due to a soft error in any component) can be shown as follows:

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

where n is the total number of scenarios in the system, $P(Sc_k)$ is the probability of the kth scenario, and $CONC_{i_k}$ is the propagation of failure from the ith component in the kth scenario.

D. Measuring Criticalities of the Components

For each component, criticality is the product of complexity, severity, and the propagation of failure. The combined impact of complexity and severity is used to calculate the propagation of failure. Criticality is calculated by taking the product of complexity, severity, and the propagation of failure. If the criticality of the ith component is Cr_i then the equation to derive it can be shown as:

$$Cr_i = \prod (OCC_i, CON(C_i), Se(C_i)) \qquad (6)$$

where, OCC_i is the overall complexity of the component, $CON(C_i)$ is the propagation of failure from the component, and $Se(C_i)$ is the severity of the component. $OCC_i, CON(C_i), Se(C_i)$ 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 failure will increase too. Hence, criticality is taken as the product of overall complexity, severity, and propagation of failure.

E. 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 behavioral model of the component to lower its complexity, and/or severity, and/or propagation of failure. Refactoring is a good candidate for this type of approach. UML model 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 [23]. 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 failure.

IV. CASE STUDIES

Real-life case study: an Automated Rail Car System illustrates the application of the metrics. This is a safety critical application, which must meet real-time criteria. It is chosen, as it is illustrative of a broad class of systems that must have high reliability. 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.

A. ET Analysis in the Automated Rail Car System

The state changes of the Car, which are used to measure the ET of the components in the Automated Rail Car System, can be described as follows. Calculated ET for the different components to take a car from Terminal [0] to Terminal [3] is shown in TABLE II.

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

Components	Normalized values of ET	TMIO	OCC
Car	.933	1.14	2.073
ProximitySensor	.023	0.21	0.233
Cruiser	.0074	0.5	0.5074
DestPanel	.005	-	0.005
OccupancySensor	.004	-	0.004
Terminal	.0015	0.21	0.2115
CarHandler	.014	1.07	1.084
CallCarButton	.002	-	0.002
Entrance	.004	0.14	0.144
Exit	.0025	0.14	0.1425
ExitManager	.003	0.21	0.213
PlatformManager	.0074	0.21	0.2174
ControlCenter	.0005	-	0.0005

B. MIO and TMIO Analysis in the Automated Rail Car System

MIO and TMIO for all components are calculated using the equations as shown in (2) and (3). All values of TMIO for all sub-systems are shown in TABLE II. The blank cells for TMIO in Table II indicate no message from the corresponding component in the corresponding scenario.

C. Overall Complexities of the Components in the Automated Rail Car System

The total complexities of all components are calculated using the equation as shown in (4). The last column in TABLE II shows the measured values of each component's complexity. Their values are normalized by taking the ratio between them and the total value. To simplify the representation of columns' headings, ‘Overall Complexities of the Components’ is abbreviated as OCC in TABLE II. In the chosen model’s scenarios, all components’ participations are not there. Hence, MIO and TMIO for some components may not be measured for this example. Only ET is taken to measure the overall complexities of those components.

D. *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 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. 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 III shows the E/F rations for this example. If these rations are ranked in an ascending order then it is observed that TABLE II has a similar ranking to TABLE III until the Cruiser component. The next ratio is equal for ProximtySensor, PlatformManager, and ExitManager where, in TABLE II, 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.

TABLE III. E/F RATIOS OF THE COMPONENTS IN AUTOMATED RAIL CAR SYSTEM

Components	Number of Transient Faults	Number of Soft Errors	E/F Ratio
Car	10	8	0.8
ProximitySensor		4	0.4
Cruiser		5	0.5
DestPanel		1	0.1
OccupancySensor		1	0.1
Terminal		3	0.3
CarHandler		7	0.7
CallCarButton		1	0.1
Entrance		3	0.3
Exit		3	0.3
ExitManager		4	0.4
PlatformManager		4	0.4
ControlCenter		1	0.1

E. *Measuring the Severity of Failure of the Components in the Automated Rail Car System Model*

Severity of the failure of the components was determined by the FMEA method. The severities of the failure of Car, ProximitySensor, Cruiser, DestPanel, OccupancySensor, Terminal, CarHandler, CallCarButton, Entrance, Exit, ExitManager, ControlCenter, and PlatformManager are calculated as 9, 8, 4, 1, 8, 6, 10, 1, 8, 9, 9, 1, and 6 respectively (ranking is based on TABLE I and the method as described in Section III.B).

TABLE IV. CRITICALITIES OF THE COMPONENTS IN AUTOMATED RAIL CAR SYSTEM

Component Name	CICS	Propagation of Soft Errors	Criticality
Car	18.657	4.1	76.4937
ProximitySe-nsor	1.864	10	18.64
Cruiser	2.0296	.000000058	0.000000117717
DestPanel	0.005	-	0.005
OccupancyS-ensor	0.032	-	0.032
Terminal	1.152	.0017	0.0000000072576
CarHandler	10.84	2.1	22.764
CallCarButt-on	0.002	-	0.002
Entrance	1.152	.0000000063	0.0000000072576
Exit	1.2825	.000014	0.000017955
ExitManager	1.917	.0013	0.0024921
PlatformMa-nager	1.3044	.0000064	0.00000834816
ControlCent-er	0.0005	-	0.0005

F. *Measuring Propagation of Soft Errors from the Components in the Automated Rail Car System*

The second column in TABLE IV shows the combined impact of complexity and severity. 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 IV. Since, in the chosen model’s scenarios, participation of all components is not considered, only the participant component’s soft errors propagations can be derived in this paper. The corresponding cells for DestPanel, OccupancySensor, CallCarButton, and ControlCenter are blank in the table since they did not participate in the scenarios.

G. *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 IV. The results show that Car is the most critical component followed by CarHandler and ProximitySensor.

H. *Lowering the Criticalities of the Components in the Automated Rail Car System*

As shown in TABLE IV, Car 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 Car. That part of the model dealing with Car behaviour was carefully examined to determine refactoring possibilities. Two states and their internal codes were merged to reduce the time complexity.

Comparison among the calculated normalized ET of the components of the refactored model and existing model (to take a car from Terminal [0] to Terminal [3]) is shown in TABLE V.

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

Components	Normalized ET of Refactored Model	Normalized ET of Existing Model
Car	0.899	.995
ProximitySensor	0.00051	.0015
Cruiser	.00048	.00048
DestPanel	.00016	.00032
OccupancySensor	0.00035	.00026
Terminal	.00013	.000096
CarHandler	.00089	.00089
CallCarButton	.000032	.00013
Entrance	0.00022	.00026
Exit	.00016	.00016
ExitManager	.000096	.000192
Platform Manager	0.00064	.00048
ControlCenter	.000032	.000032

TABLE V shows that refactoring the model lowered the ET of Car and ProximitySensor to a measurable extent. Others, except for OccupancySensor and PlatformManager, were also lowered.

V. CONCLUSIONS

This paper develops metrics for complexity analysis that could be analyzed in the early system design phase based on UML artifacts, develops a severity assessment methodology by analyzing UML model simulation results, and develops the methodology of measuring the propagation of failures from the components. This paper then integrates the three different methods to rank the component's criticality that highlight the variations of the impact of soft errors among the components. It then shows how possible changes can be made in the existing design to lower the criticalities of the components to minimize the risks of soft errors. In summary, the approach presented in this paper is effective in measuring the soft errors risks of the components in a system and in lowering the criticalities of components to minimize the risks of functional degradation.

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 (Available Online),* 2008.

[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] 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.

[4] 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.

[5] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim, "Robust system design with built-in soft-error resilience," *Computer,* vol. 38, pp. 43-52, 2005.

[6] 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.

[7] J. Jurjens and S. Wagner, "Component-based development of dependable systems with UML," *Lecture Notes in Computer Science,* vol. 3778, pp. 320-344, 2005.

[8] 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.

[9] J. M. a. T. Khoshgoftaar, "Software metrics for reliability assessment," in *Handbook of Software Reliability Eng., M. Lyu ed., Chapter 12,* 1996, pp. 493-529.

[10] 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.

[11] Meaney, P.J., Swaney, S.B., Sanda, P.N., Spainhower, L., "IBM z990 soft error detection and recovery.," IEEE Transactions on Device and Materials Reliability, Vol. 5, 2005.

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

[13] Xinping, Z., Wei, Q., "Prototyping a fault-tolerant multiprocessor SoC with run-time fault recovery," 43rd ACM/IEEE Design Automation Conference, pp. 53 – 56, 2006.

[14] Cazeaux, J.M., Rossi, D., Omana, M., Metra, C., Chatterjee, A., "On transistor level gate sizing for increased robustness to transient faults," 11th IEEE International On-Line Testing Symposium, pp. 23 – 28, 2005.

[15] 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.

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

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

[18] Chidamber, S.R., Kemerer, C.F., "A metrics suite for object oriented design," IEEE Transactions on Software Engineering, Vol. 20, pp. 476-493, 1994.

[19] Harrison, R., Counsell, S., Nithi, R., "Coupling metrics for object-oriented design," IEEE Comput. Soc, Bethesda, MD, USA, pp. 150-157, 1998.

[20] 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.

[21] http://www.telelogic.com/Products/rhapsody/index.cfm

[22] 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.

[23] 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.