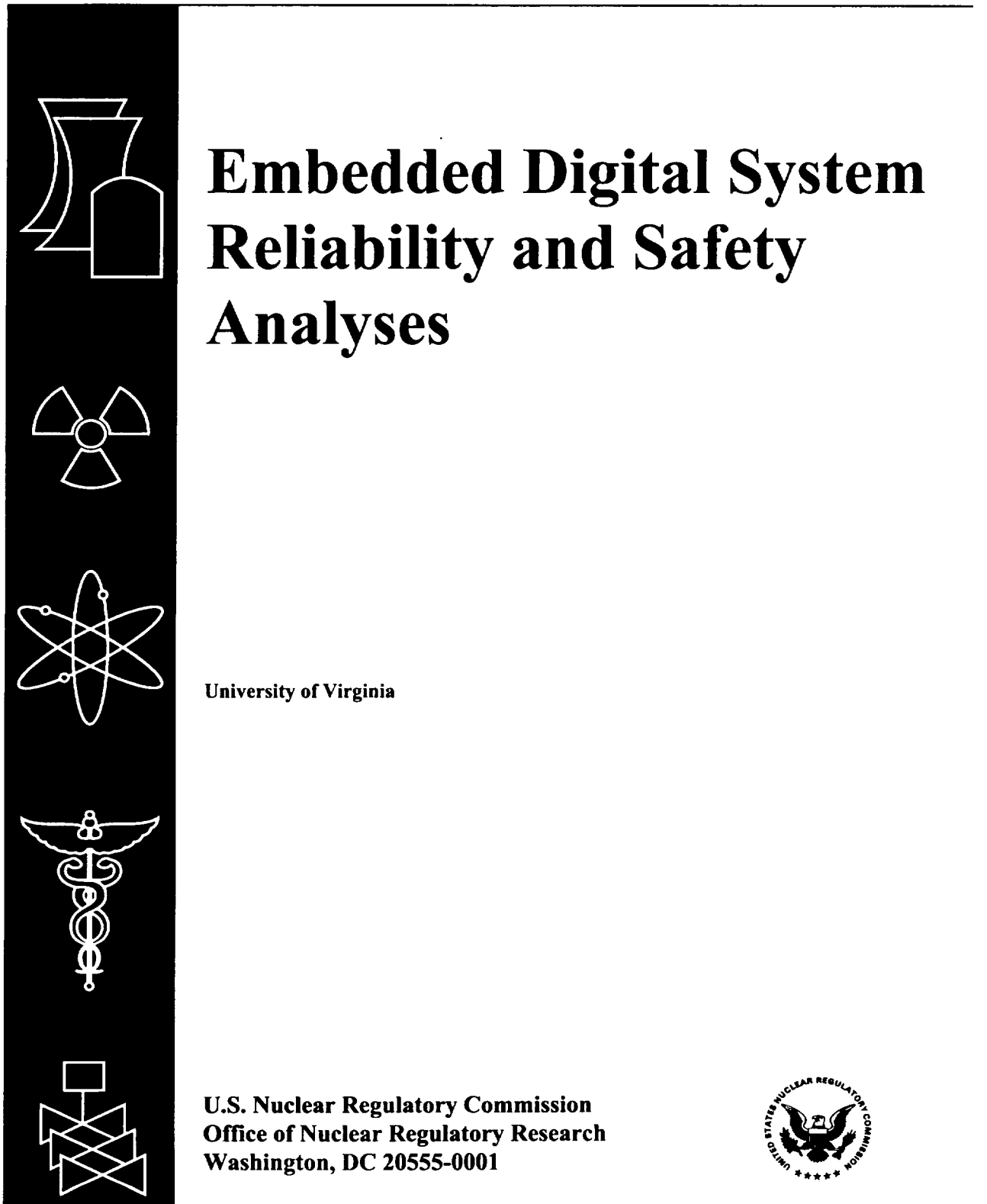
NUREG/GR-0020



Embedded Digital System Reliability and Safety Analyses

University of Virginia

U.S. Nuclear Regulatory Commission
Office of Nuclear Regulatory Research
Washington, DC 20555-0001

AVAILABILITY OF REFERENCE MATERIALS IN NRC PUBLICATIONS

NRC Reference Material

As of November 1999, you may electronically access NUREG-series publications and other NRC records at NRC's Public Electronic Reading Room at www.nrc.gov/NRC/ADAMS/index.html. Publicly released records include, to name a few, NUREG-series publications; *Federal Register* notices; applicant, licensee, and vendor documents and correspondence; NRC correspondence and internal memoranda; bulletins and information notices; inspection and investigative reports; licensee event reports; and Commission papers and their attachments.

NRC publications in the NUREG series, NRC regulations, and *Title 10, Energy*, in the Code of *Federal Regulations* may also be purchased from one of these two sources.

1. The Superintendent of Documents
   U.S. Government Printing Office
   Mail Stop SSOP
   Washington, DC 20402–0001
   Internet: bookstore.gpo.gov
   Telephone: 202-512-1800
   Fax: 202-512-2250
2. The National Technical Information Service
   Springfield, VA 22161–0002
   www.ntis.gov
   1–800–553–6847 or, locally, 703–605–6000

A single copy of each NRC draft report for comment is available free, to the extent of supply, upon written request as follows:

Address: Office of the Chief Information Officer,
Reproduction and Distribution
Services Section
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001
E-mail: DISTRIBUTION@nrc.gov
Facsimile: 301–415–2289

Some publications in the NUREG series that are posted at NRC's Web site address www.nrc.gov/NRC/NUREGS/indexnum.html are updated periodically and may differ from the last printed version. Although references to material found on a Web site bear the date the material was accessed, the material available on the date cited may subsequently be removed from the site.

Non-NRC Reference Material

Documents available from public and special technical libraries include all open literature items, such as books, journal articles, and transactions, *Federal Register* notices, Federal and State legislation, and congressional reports. Such documents as theses, dissertations, foreign reports and translations, and non-NRC conference proceedings may be purchased from their sponsoring organization.

Copies of industry codes and standards used in a substantive manner in the NRC regulatory process are maintained at—

The NRC Technical Library
Two White Flint North
11545 Rockville Pike
Rockville, MD 20852–2738

These standards are available in the library for reference use by the public. Codes and standards are usually copyrighted and may be purchased from the originating organization or, if they are American National Standards, from—

American National Standards Institute
11 West 42nd Street
New York, NY 10036–8002
www.ansi.org
212–642–4900

Legally binding regulatory requirements are stated only in laws; NRC regulations; licenses, including technical specifications; or orders, not in NUREG-series publications. The views expressed in contractor-prepared publications in this series are not necessarily those of the NRC.

The NUREG series comprises (1) technical and administrative reports and books prepared by the staff (NUREG–XXXX) or agency contractors (NUREG/CR–XXXX), (2) proceedings of conferences (NUREG/CP–XXXX), (3) reports resulting from international agreements (NUREG/IA–XXXX), (4) brochures (NUREG/BR–XXXX), and (5) compilations of legal decisions and orders of the Commission and Atomic and Safety Licensing Boards and of Directors' decisions under Section 2.206 of NRC's regulations (NUREG–0750).

NUREG/GR-0020

Embedded Digital System Reliability and Safety Analyses

Manuscript Completed: November 2000
Date Published: February 2001

Prepared by
L.M. Kaufman, B.W. Johnson

University of Virginia
Department of Electrical Engineering
Center for Safety-Critical Systems
Thornton Hall
Charlottesville, VA 22904

J.A. Calvert, NRC Project Manager

Prepared for
Division of Engineering Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001
NRC Job Code K6907

ABSTRACT

The migration from analog to digital systems in the instrumentation and control (I & C) within a nuclear power plant has increased the complexity of the instrumentation. The I & C systems being developed are computer-based consisting of embedded digital hardware and software components. These embedded systems perform many varying and highly complex functions that are integral to the safety-critical requirements of a nuclear power plant. The need to understand the effects of various failure modes, including common cause failures and common mode failures, in these systems is becoming increasingly important because the failure of an I & C system could lead to risk significant events.

In order to understand the effects of common cause and common mode failures on a system, a survey of existing definitions and applications of these definitions as they apply to digital embedded systems was performed. From this survey, it was found that the definitions and analysis treated the hardware and the software as independent entities. However when embedded digital systems are in actual operation, there is tight integration of the hardware and software components; that is, the function realized by a such system cannot be partitioned between hardware and software but must be analyzed in a unified manner. In addition to addressing the limitations of the existing common cause and common mode failure definitions, a detailed assessment of existing modeling techniques to assess their effects is also presented.

DISCLAIMER

This publication was prepared with the support of the U.S. Nuclear Regulatory Commission (NRC) Grant Program. This program supports basic, advanced, and developmental scientific research for a public purpose in areas related to nuclear safety. The grantee bears prime responsibility for the conduct of the research and exercises judgement and original thought toward attaining the scientific goals. The opinions, findings, conclusions, and recommendations expressed herein are therefore those of the authors and do not necessarily reflect the views of the NRC.

CONTENTS

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

EXECUTIVE SUMMARY

The migration from analog to digital systems in the instrumentation and control (I & C) within a nuclear power plant has increased the complexity of the instrumentation. The I & C systems being developed are computer-based consisting of embedded digital hardware and software components. These embedded systems perform many varying and highly complex functions that are integral to the safety-critical requirements of a nuclear power plant. The need to understand the effects of various failure modes, including common cause failures (CCF) and common mode failures (CMF), in these systems is becoming increasingly important because the failure of an I & C system could lead to risk significant events.

In order to understand how common cause and common mode failures can affect a system, a survey of existing definitions and applications of these definitions as they apply to digital embedded systems was performed and can be found in Section 4: Common Mode and Common Cause Failures. From this survey, it was found that the definitions and analysis treated the hardware and the software as independent entities. However when embedded digital systems are in actual operation, there is tight integration of the hardware and software components; that is, the function realized by a such system cannot be partitioned between hardware and software but must be analyzed in a unified manner and this problem is discussed in detail in Section 4.4: Limitations of Existing CMF and CCF Definitions with Regards to Embedded Digital Systems. In addition to addressing the limitations of the existing common cause and common mode failure definitions, a detailed assessment of existing modeling techniques to assess their effects is also presented.

Typical modeling techniques used to assess safety and reliability include Markov models, Petri nets and traditional and dynamic fault trees. Each approach is discussed in Section 3.1: Dependability Modeling and Analysis. In these techniques, the models require various parameters that reflect system behavior. In order to estimate these parameters for embedded systems, the integrated action and interaction between the hardware and the software components must be captured. By modeling the embedded system using hardware description languages (HDL), the behavior of both the software and the hardware can be jointly examined; that is, a behavioral model can be developed that encompasses the interaction between the software and the hardware components encountered in actual operation. Using this HDL model, a series of fault injection experiments (see *Section 1.2.2: Dependability Analysis Modeling Methodology (Kaufman, 1997)*) can be performed to estimate various parameters.

The fault injection experiments perturb the correct operational environment for the embedded system by corrupting the information that is being processed. As a result of the fault injection experiment, the effects that faults have on the system can be analyzed. The quantitative parameter that is associated with how faults affect a system is *coverage*. *Coverage* is a conditional probability that estimates the likelihood that a system will continue to operate correctly or fail in a safe manner under the presence of a fault given that it was operating correctly prior to the fault

occurrence. A detailed discussion of the various aspects of coverage modeling and estimation can be found in *Appendix A: Coverage Modeling.*

At the present time, this research effort is focusing on the methodology required to develop safety and reliability models using fault injection and the application of such modeling in parallel with system design. It is believed that by performing safety and reliability analysis jointly during the design cycle, potential problems can be identified early in the process and corrected in a more cost efficient manner. Currently, software development is not a focus issue of this research. It is assumed the software being tested is appropriate for the given application and its development adheres to current software practices for verification and validation. Hence, this research focuses on the methodology required for modeling safety and reliability for embedded digital systems.

The results from this stage of the research include the following observations:

- the existing definitions for common cause and common mode failures are inconsistent in their terminology
- the existing definitions for common cause and common mode failures treat hardware and software independently
- the accuracy of the safety and reliability models depends upon the realism of the parameters
- recognition of the need to estimate model parameters for the embedded system

By continuing research in the following areas, it is believed that a flexible methodology for determining the safety and reliability of embedded digital systems will be realized.

Future Research:

- develop a methodology for accurate safety and reliability modeling of the dependencies between hardware and software in embedded digital systems
- incorporate the effects of common mode and cause failures in modeling and the estimation of the parameters for the models
- integration of commercial-off-the-shelf components in design and safety and reliability modeling

Acronyms

CCF: common cause failure

CMF: common mode failure

COTS: commercial off the shelf

GSPN: generalized stochastic Petri net

HDL: hardware description language

I & C: instrumentation and control

SPN: stochastic Petri net

1. Introduction

The migration from analog to digital systems for instrumentation and control (I & C) within a nuclear power plant has increased the complexity of the instrumentation. The I & C systems being developed are computer-based consisting of embedded digital hardware and software components. These systems are performing many varying and highly complex functions that are integral to the safety-critical requirements of a nuclear power plant, and the failure of an I & C system could lead to risk significant events. In order to prevent such situations from arising, there is a great need for these systems to be dependable; that is, these systems must provide a specified quality of service. It is the system designer's responsibility for demonstrating that a given system is dependable (Carter, 1987). Hence, there is a definite need for dependability analysis to be performed during the design phase. At the University of Virginia, the dependability analysis is performed concurrently with system development as depicted in Figure 1.The

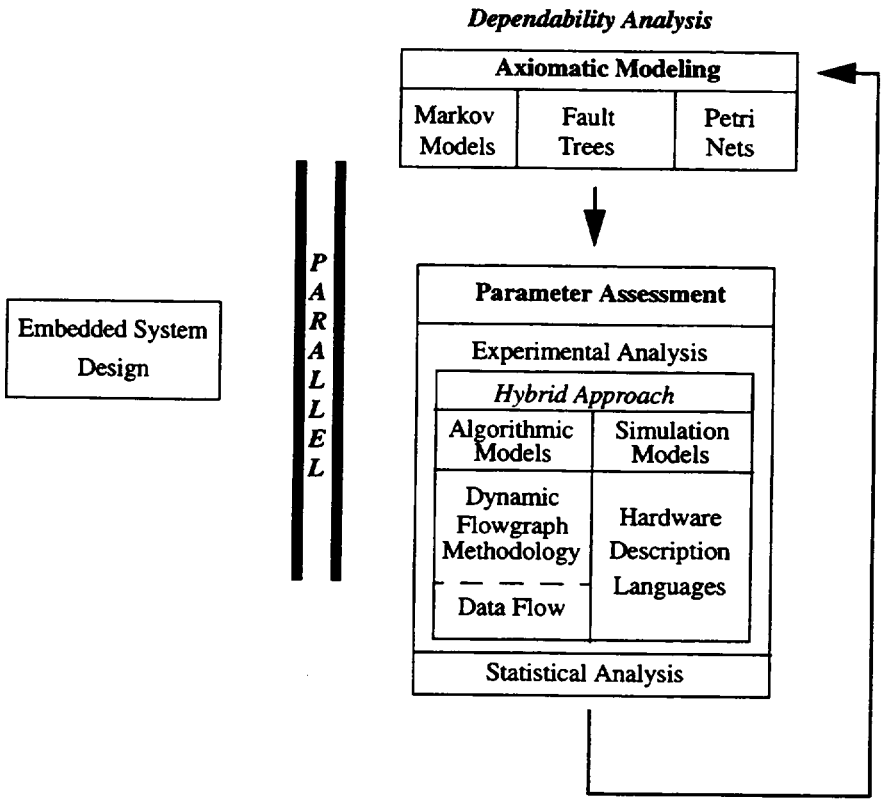


Figure 1: The University of Virginia Methodology

advantages of such concurrency are that if it can be determined during design that the system does not meet its dependability requirements, then design modifications can be made early in the design cycle to minimize cost. A detailed discussion of the benefits associated with performing dependability analysis concurrently with system design

can be found in Section 1.2.1: Design Time Line.

Dependability analysis itself provides a measure of the quality of service that a particular system delivers (Laprie, 1985) and the system model provides the ability to assess the various dependability metrics. Among the metrics that are typically used to characterize dependability are reliability, availability and safety (Laprie, 1985; Laprie, 1992). Reliability is defined as the conditional probability that a system performs correctly for a time interval, given that it was performing correctly at the beginning of the interval. Reliability is typically reported as the probability that a system functions correctly for a given time period. Availability is the probability that a system is operating correctly at a given instance of time. Availability is typically reported in terms of the maximum allowable downtime that is permitted for a component or a system. Safety is the probability that a system is either operating correctly or in the presence of a fault, it discontinues its operations in a well-defined, safe manner (Jallote, 1994; Johnson, 1989). Safety can be described as the number of allowable unsafe events permitted in a given period of time. It is the calculation of these metrics that can substantiate whether or not a system is dependable; that is, a system is deemed to be dependable if these metrics meet or exceed requirements. The calculation of these metrics requires system modeling and analysis, which is both quantitative and qualitative in nature.

The system model typically is a qualitative system representation; that is, it represents system interactions and behavior as interpreted by the analyst. The three primary types of axiomatic models, which are analytical representations of a system, used for dependability modeling as shown in Figure 1 are summarized in Table 1. A

Table 1: Comparative analysis of axiomatic models

Features	*Markov Models (Kaarlin, 1975; Cassandros, 1993)*	*Petri Nets (Murata, 1989; Peterson, 1981; Cassandros, 1993)*	*Fault Trees*	
			Traditional	*Dynamic (Dugan et al, 1992)*
System representation	represents the system in terms of the system states and transitions between the states	• represents the system in terms of conditions and events • easy to represent concurrency and contention	logical representation of the interrelationships between a critical event and its underlying cause(s)	• logical representation of the interrelationships between a critical event and its underlying cause(s) • represents certain sequence dependent failures
Typical parameters	• failure rate • repair rate • coverage	• failure rate • repair rate • coverage • failure probability	• failure probability • failure rate • coverage	• coverage • failure rate
Distributions of timed events	• exponential • Weibull	exponential	any distribution	• exponential • Weibull

Table 1: Comparative analysis of axiomatic models

Features	*Markov Models (Kaarlin, 1975; Cassandros, 1993)*	*Petri Nets (Murata, 1989; Peterson, 1981; Cassandros, 1993)*	*Fault Trees*	
			Traditional	*Dynamic (Dugan et al, 1992)*
Closed form solution	differential equations	conversion to Markov model	combinational logic (Boolean algebra)	combinational logic (Boolean algebra) and conversion to Markov model
Qualitative analysis	not applicable	reachability: determines if a system can ever reach a certain state	cutset based analysis	cutset based analysis

detailed discussion of these modeling techniques can be found in Section 3: Probability Risk Assessment (PRA) and Digital Embedded Systems.

The analysis of such models provides a quantitative basis for the various dependability metrics. This analysis also demonstrates the differences among the various metrics and identifies the parameters required for solution of a given model. The gathering of data for certain parameters is experimentally obtained using algorithmic and simulation based models (The University of Virginia's *hybrid* approach). The data flow and dynamic flowgraph methodology (Garrett, Yau, Guarro and Apostolakis, 1995; Garrett, Guarro and Apostolakis, 1995) provide an algorithmic based approach that can be used in conjunction with hardware description languages (HDLs) to simulate the system behavior and functionality in the presence of faults. A summary of these algorithmic models is given in Table 2 and a detailed discussion of these techniques is presented in Section 3.2: Hybrid Modeling.

Table 2: Algorithmic models used during experimental analysis

Features	*Data Flow*	*Dynamic Flowgraph*
Dependability analysis axiomatic model	Can be mapped with any model	Requires multi-valued fault tree
Simulation capabilities	Allows for fault injection	Allows for fault injection
Primary application	parameter estimation	design verification, fault and failure analysis and test vector generation analysis

1.1 Quantitative Differences Among the Dependability Metrics

The system model provides the basis for any quantitative dependability analysis. This analysis provides for estimation of the various dependability metrics and it also demonstrates their differences. For example, the differences between the reliability and the safety metrics for a given system can be analyzed using a continuous time

Markov model (Karlin and Taylor, 1975).

A simplex system, which is a system containing only a single component, that cannot be repaired is depicted in Figure 2. In this model, there are three possible states that the system can be in at a given time:

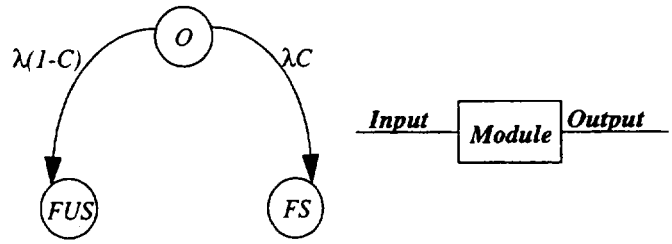


Figure 2: Continuous time Markov model of a simplex system without repair

(1) *Operational*: state *O* representing a system that is functioning correctly;

(2) *Failed-safe*: state *FS* representing a system whose failure is detected allowing it to fail in a *safe* manner;

(3) *Failed-unsafe*: state *FUS* representing a system whose failure goes undetected allowing it to fail in an *unsafe* manner.

The concepts of safe and unsafe are highly dependent upon the application and they directly impact a given system's model. The parameters required by this model are: (1) the failure rate, λ and (2) the fault coverage, C. The failure rate is the rate at which failures occur and this information can be readily obtained from data collection or the Mil-Spec Handbook. Coverage is a conditional probability that if a fault occurs, it is detected and the system can recover. Conversely, an uncovered fault leads to immediate system failure regardless of the system's state. The estimates that are used for the coverage parameter are not as easily obtained as the failure rate. Rather, the coverage estimate is typically made from experimentally testing a system under various fault conditions to determine if the system recognizes the fault and recovers and, if available, from existing data. Using these parameters, the safety and the reliability of the simplex system can be calculated using techniques presented in either (Johnson, 1989) or (McCormick, 1981).

For the model in Figure 2, the metrics are

$$\textit{Safety: } S(t) = p_O(t) + p_{FS}(t) = C + (1 - C)e^{-\lambda t} \tag{1}$$

that is, safety is statistically defined as the probability of being in any state other than the failed unsafe state.

$$\textit{Reliability: } R(t) = p_O(t) = e^{-\lambda t} \tag{2}$$

that is, reliability is statistically defined as the probability of being in any operational state for a specified time interval. For any continuous time Markov model, the sum of the probabilities of being in any of the model states is one; that is, for the simplex system without repair

$$P(t) = p_O(t) + p_{FS}(t) + p_{FU}(t) = 1 . \tag{3}$$

Additionally, the analytical differences between safety and reliability for any system are demonstrated in Equation (1) and Equation (2).

The system safety is the sum of the probabilities of being either in the operational or the failed safe states, however, reliability is simply the probability of being in the operational state for a specified time interval. It must be noted that for more complex systems the number of operational states is greater than one, and the probability of being in such states must be reflected in the expressions for both reliability and safety. Hence, the probability associated with a system failing safe is greater than or equal to the probability associated with system reliability regardless of its configuration; that is,

$$S(t) \geq R(t) \tag{4}$$

In assessing both the reliability and safety for the simplex system, the sensitivity of the metrics to the parameters must be examined. Since the component failure rate is more readily available than the coverage, the sensitivity of both the reliability and the safety for the simplex system without repair to the coverage parameter needs further investigation.

Assuming that the component failure rate is $\lambda = 10^{-4}$ failures per day and that the mission time is $t = 100$ days, the effects of various coverage values on the simplex system is shown in Table 3. This table demonstrates that

Table 3: Sensitivity of Dependability Metrics to Coverage for a Simplex System

C	*R(t)*	*S(t)*
0.9	0.9048374180	0.9904837418
0.99	0.9048374180	0.9990483742
0.999	0.9048374180	0.9999048374
0.9999	0.9048374180	0.9999904837
0.99999	0.9048374180	0.9999990484
0.999999	0.9048374180	0.9999999048
. . .	. . .	. . .

the reliability of the simplex system is not affected by coverage, which is indicated in Equation (2). However, the

system safety increases with coverage, which follows from Equation (1). In order to better demonstrate the effects of the coverage parameter on these metrics, examination of additional systems is required.

In Figure 3, the continuous time Markov models for a duplex, a triplex and a quadraplex system without

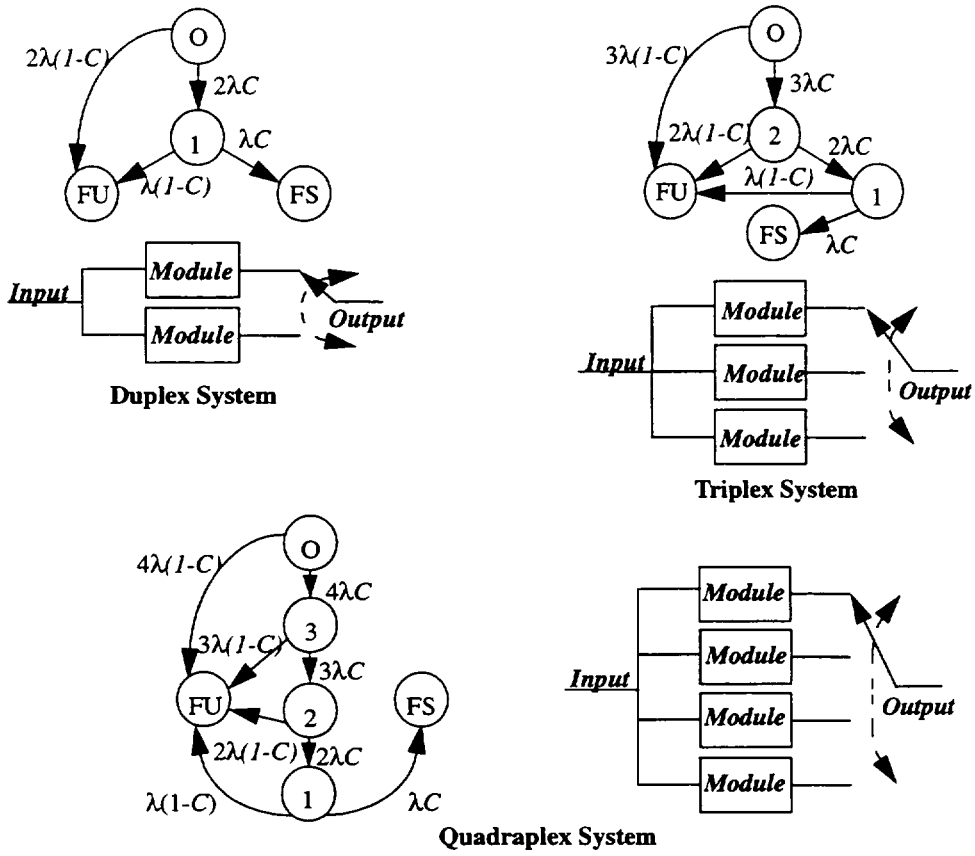


Figure 3: Continuous time Markov models of systems without repair

repair are shown. The numbered states in each model correspond to operational states that are reached after a covered fault has been encountered. Assuming that the component failure rate is still $\lambda = 10^{-4}$ failures per day and the mission time is $t = 100$ days, the effects of various coverage values on these three architectures are presented in Table 4. For each of these systems, both the reliability and the safety increase as the coverage increases. As expected, the system safety exceeds the system reliability for each configuration. Similarly, the overall system reliability and safety increases with component redundancy until the number of components increases from three to four. Rather, both the safety and the reliability for the system decrease when the redundancy increases from a triplex to a quadraplex system. This phenomena is graphically depicted in Figures 4 and 5.

In Figure 4, the sensitivity to the dependability metrics to the coverage parameter and redundancy is demonstrated for all four system configurations. From this graphic, it is apparent that the system reliability increases

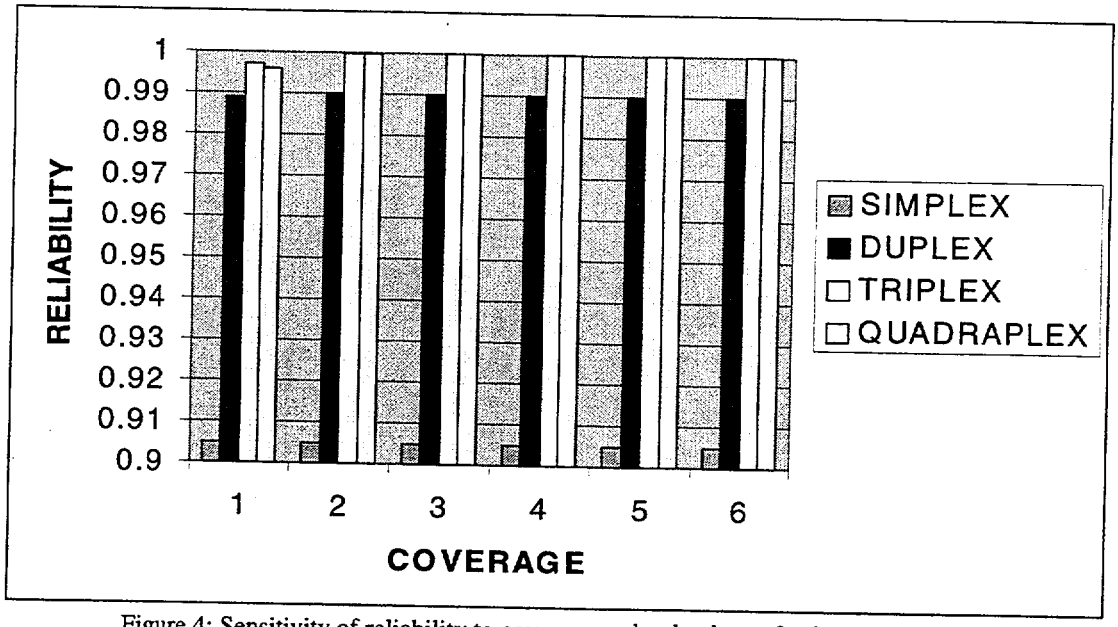


Figure 4: Sensitivity of reliability to coverage and redundancy for four architectures

Table 4: Sensitivity of Dependability Metrics to Coverage

C	*Duplex*		*Triplex*		*Quadraplex*	
	R(t)	*S(t)*	*R(t)*	*S(t)*	*R(t)*	*S(t)*
0.9	0.9890154300	0.9970064320	0.997017200	0.998017919	0.99602588	0.99602588
0.99	0.9899014981	0.9997005084	0.999700568	0.999801525	0.99960204	0.99960205
0.999	0.9899901991	0.9999700994	0.999969167	0.9999801501	0.99996020	0.99996022
0.9999	0.9899990197	0.9999800994	0.999996029	0.999997013	0.99999601	0.99999603
0.99999	0.9899999018	0.9999997010	0.999999716	0.999999801	0.99999960	0.99999962
0.999999	0.9899999900	0.9999998010	0.999999985	0.999999990	0.99999996	0.99999998
. . .	. . .	. . .	. . .	. . .	. . .	. . .

as the number of components increases to three. However, there is additional resolution required to determine the effect of adding a fourth component. This phenomena is shown in Figure 5.

The graphics in Figure 5 demonstrate that the system reliability (and the same is true for safety) actually decreases as the number of redundant components increases from three to four. Hence, it is shown that by simply adding redundant components to a design does not improve system dependability. Rather, the metrics that comprise dependability can actually decrease in value as the number of redundant components exceeds three. Additionally, the

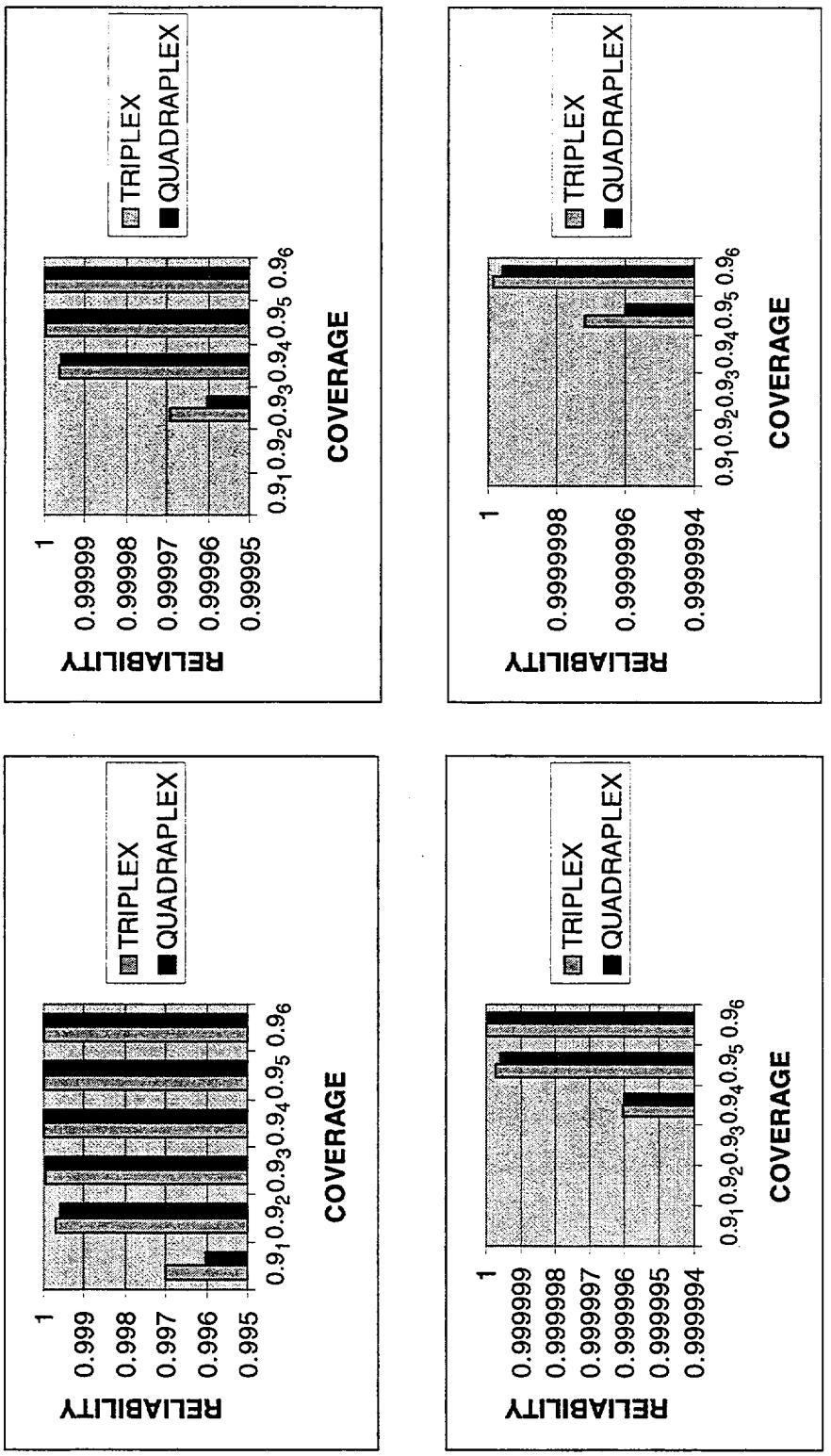


Figure 5: Reliability comparison between a triplex and a quadraplex system

sensitivity of the metrics to the coverage parameter is highlighted; that is, the analysis of dependability metrics is sensitive to both redundancy and coverage. Therefore, it is imperative that consideration must be given both to the amount of redundancy that is contained in the design of an embedded digital system and the required level of coverage that is needed to meet the required level of dependability.

Hence, the application of dependability analysis to embedded digital systems during the design phase is extremely important. This concurrency allows for dependability information to be available to the designer in a timely manner so that the appropriate design decisions can be made to ensure that a sufficient level of coverage can be obtained to achieve the required level of dependability. The subtleties of implementing such an analysis are quite complex and require careful planning. Such planning requires integration of both the design and dependability modeling so that the system developer can be aware of any potential difficulties as they arise and make appropriate design modifications. Hence, a defined dependability analysis methodology is required.

1.2 The University of Virginia Dependability Analysis Methodology

In order to maximize the benefits obtainable from dependability analysis, careful consideration must given to its application. If the analysis can be performed in parallel with system design as shown in Figure 1, then a myriad of useful information can be obtained. For example, such an analysis might demonstrate the need for certain design changes at an early stage in the design cycle, or it could demonstrate that commercial off the shelf (COTS) components are an acceptable design choice, both of which can mitigate cost. Obviously, the analysis is sensitive to the model that is used and great care must be taken in selecting the appropriate model. For embedded digital systems, such models must consider both hardware and software and their interaction. In performing an effective and comprehensive dependability analysis, the design time line, the modeling methodology and its application must be carefully defined.

1.2.1 Design Time Line

In demonstrating whether or not a system meets its dependability requirements, such analysis should ideally be performed in parallel with the design cycle. Often, the development of a system is partitioned so that one group is responsible for developing the system and another group is responsible for determining whether or not the system meets dependability specifications and requirements. In Figure 6, the *Traditional Design Timeline* illustrates this dilemma.

Initially, the system developers design and construct the first system, *Model 1*, and then the dependability analysis is performed. However before the dependability analysis is completed for the first system, the second design iteration for the system, *Model 2*, will typically have commenced. This phenomena often forces designers to neglect

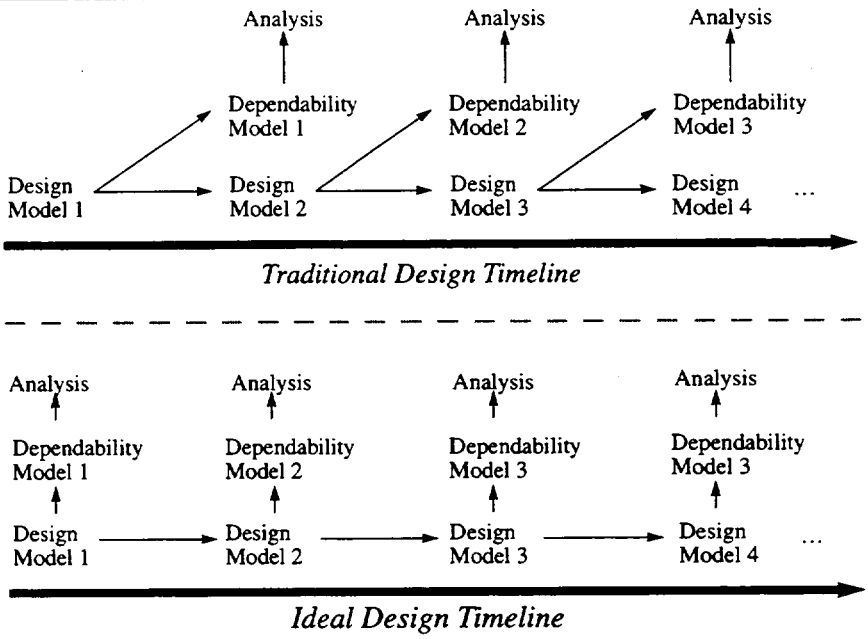


Figure 6: Analysis skew illustration

the dependability concerns from the first design iteration, and this information lag is inherent throughout the entire design process. This approach can prove to be disastrous when applied to safety-critical system design. Dependability analysis performed late in the design cycle can reveal shortcomings of early design decisions whose modification could be prohibitively expensive. Likewise, failure to address these shortcomings could be potentially fatal. If the dependability analysis is performed in parallel with the system design, as indicated by the *Ideal Design Timeline* in Figure 6, then dependability concerns can be accommodated throughout the entire design cycle; that is as the dependability analysis identifies concerns, then the designers can correct for such problems.

By performing dependability analysis in parallel with the design, the analysis of certain features of embedded digital systems can be better defined. The interaction of hardware and software in these systems is very complex, and at times the separation between the two is lost. If this interaction is examined in parallel with the design, then potential risks from this interaction can be determined early in the design cycle and the appropriate modifications to either the hardware or the software can be made in a more timely and cost efficient manner. Hence, there is a definite need for dependability analysis and modeling to be performed in parallel with the design of embedded digital systems, especially for systems used in safety-critical applications.

1.2.2 Dependability Analysis Modeling Methodology (Kaufman, 1997)

In performing dependability analysis, great care must be given in selecting the appropriate model. There are three primary types of models used in dependability analysis:

1. *axiomatic models*: analytical models that are used to model structure and the dependability and/or performance behavior of a system (Arlat et al, 1993).
2. *empirical models*: statistical models that are used to model complex and detailed descriptions of a system's parameter(s) using data collected from physical models.
3. *physical models*: prototypes that actually implement the hardware and/or the software of an actual system.

There is a hierarchical relationship among these models and this relationship is shown in Figure 7. The highest level

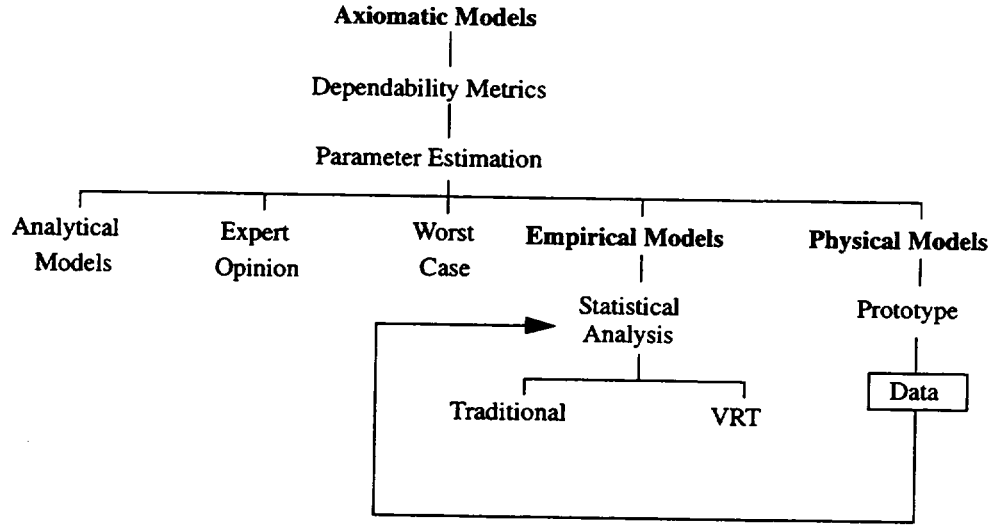


Figure 7: Modeling hierarchy (Kaufman, 1997)

model available is the axiomatic model. The parameters required by axiomatic models can include: (1) failure rates; (2) repair rates; and (3) coverage. The parameters used within the various axiomatic models are obtained in several ways. They can be estimated by assuming a worst case scenario, using expert opinion, or from other analytical models. If a more accurate parameter estimate is required and a detailed behavioral and structural description of a system is available, then empirical and physical models of the system can be constructed.

Empirical models allow for specific analysis of parameters using a simulation approach (Arlat et al, 1993). This approach can use either a statistical analysis or a physical model. Statistical analysis involves simulation-based testing of the empirical model to estimate fault coverage. In simulation-based modeling, a model is exercised repeatedly so that data can be collected to calculate the statistics of interest. The parameter estimation is based upon the data collected for a given set of *fault injection* (Arlat et al, 1990, Arlat et al, 1991; Daehn, 1991; Segall et al, 1988) experiments. This process is shown in Figure 8.

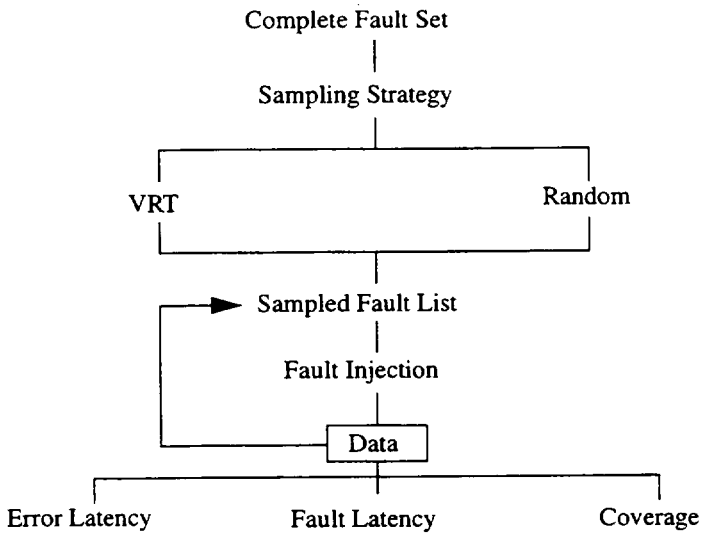


Figure 8: Parameter estimation hierarchy (Kaufman, 1997)

Fault injection is simply a technique for testing a system with respect to a class of inputs, that is faults, to determine how they affect the system. One of the difficulties in fault injection is determining the set of faults that should be injected. Since exhaustive testing of all possible faults that a system may encounter during its lifetime is impractical, a statistically valid sample must be selected to create a fault list. This statistically valid sample is a representative sample of the entire fault space that can be generated by random sampling or by using a variance reduction technique (VRT). Once the sampled fault list is generated, these faults are then deliberately injected into the system. The effects of the faults on the system's behavior can be observed to determine if they are properly handled by the system or to determine if an unsafe system failure occurs. For Figure 1 on page 1, fault injection is achieved using the hybrid approach; that is, faults are deliberately injected into simulation models and their subsequent effects are determined using algorithmic models. The data generated from these experiments provides fault and error latency and coverage estimates.

1.2.3 Application of Dependability Analysis: A Hierarchical Approach

Throughout system design, there are various levels of abstraction that define the system at a particular instance of time. These different levels of abstraction are identified in Figure 9. In this figure, a hierarchical based approach is presented to define the level of detail required by a given model. There are three levels of model abstraction defined that provide various levels of modeling information. Throughout system design, the various modeling levels can be combined to provide for overall dependability analysis and to determine which components

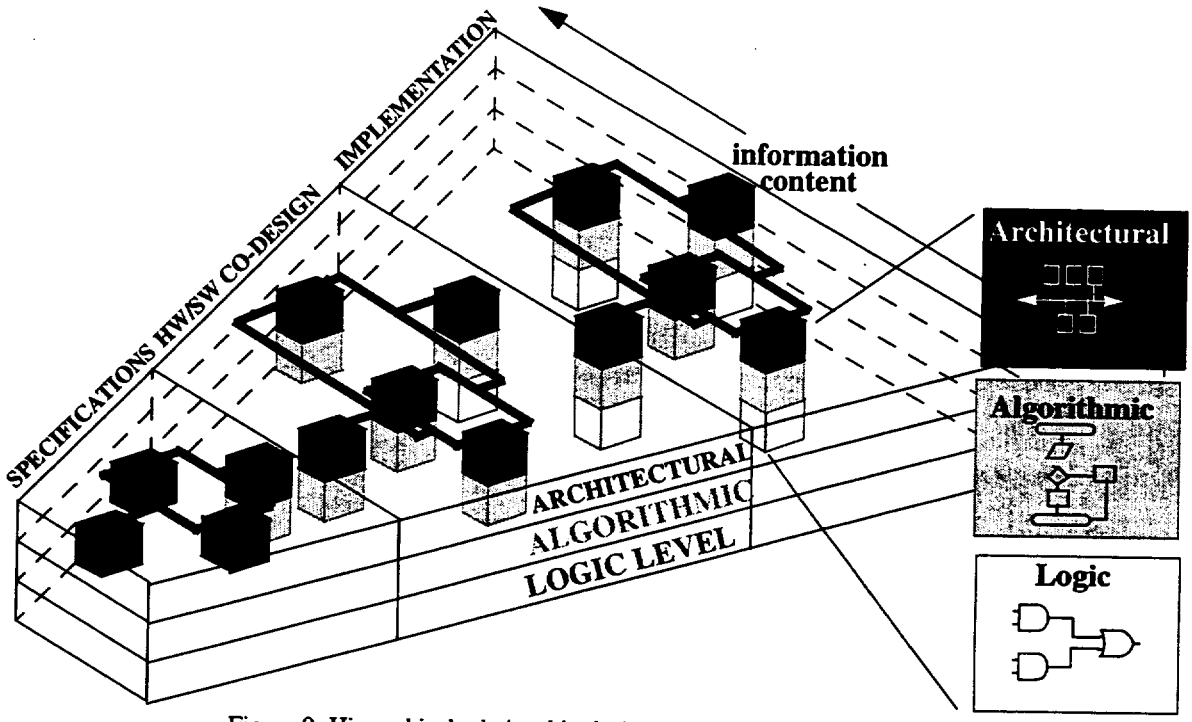


Figure 9: Hierarchical relationship during system design (Richards, 1993)

pose the greatest problems to overall system dependability.

The architectural level provides the highest level of system information. This model level explains the desired system behavior and demonstrates the interaction of various high-level components. When little information is available about a given component, then this level of modeling may be all that is available. Typically, the level of detail available for COTS components is limited, and as a result, the modeling of COTS components may only be realizable at the architectural level; that is, the manufacturers do not make low-level model information, such as the actual hardware schematic or the software, available. If it can be demonstrated during architectural system modeling and analysis that a component does not pose a risk to the overall system safety at a very early design stage, then there would be no need for further detailed modeling of such a component. Hence, COTS components could be used. If however the overall system safety is sensitive to the potential risk associated with a particular component, then a more detailed description of this component is required. Depending upon the level of detail that is ultimately required by such an analysis, a COTS component may be infeasible because of the limited knowledge available regarding its low-level design. In this case, an application specific component will need to be designed.

The specifications, which are derived from the system's requirements, provide the information that is used to develop the architectural level. At the algorithmic level, the actual design of the embedded digital system is performed. At this design level, component selection is made and the functional interaction among the various

components is defined. A complete understanding of both the behavior and the function of the various components is required. The final level of model abstraction is the logic level, which is where the actual implementation of the system occurs. At this level, detailed knowledge of the logic components required to implement the various functions is required.

By performing system development in this hierarchical modeling manner, it allows for the inclusion of information from these various design abstractions in a single, unified model. The benefit of such an approach is that the required level of modeling detail of the various system components is directly defined by the dependability analysis being performed. For each of these levels of abstraction, a fault list exists and fault injection can be performed. Hence, dependability analysis can be performed either at each level or across varying levels of the design abstraction in parallel with the design, as discussed in Section 1.2.1: Design Time Line. This type of analysis provides the designer with information regarding the potential risks associated with the various components at the various levels of abstraction. Such information is very beneficial in making design choices; that is, this information could have significant impact on deciding whether or not to use COTS hardware and software or application specific components.

1.3 Document Organization

The organization of the document is as follows. In Section 2: Background, the various design methods used for achieving dependability are introduced and their application to embedded digital systems is defined. In Section 3: Probability Risk Assessment (PRA) and Digital Embedded Systems, the available dependability modeling and analysis techniques for embedded digital systems are presented. Section 4: Common Mode and Common Cause Failures presents an overview of existing definitions for common mode and common cause failures and the limitations applying of the existing common mode and common cause failure definitions to embedded digital systems. In Appendix A: Coverage Modeling, a detailed overview of the various coverage modeling techniques are presented.

2. Background

In order to understand the concept of reliability and safety modeling, there are some fundamental definitions that must be presented. Among these are *fault*, *error* and *failure* (Johnson, 1989; Laprie, 1985). There is a cause and effect relationship among these terms and this relationship is depicted in the three-universe model, as shown in Figure 10. In this model, a fault occurs in the physical universe. The fault itself is a physical defect, imperfection or flaw that occurs within some hardware or software component (Johnson, 1989). An example of a fault could be a broken ground connection in a printed circuit board or an infinite loop in a computer program. The manifestation of a fault is

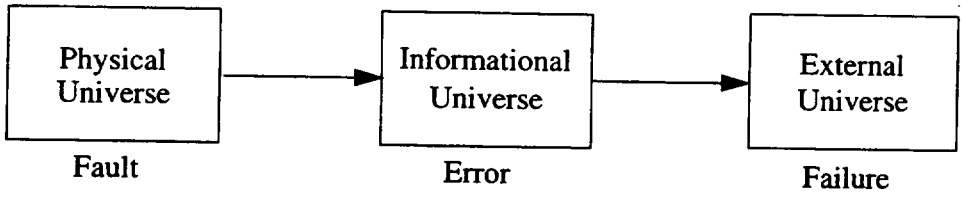


Figure 10: Three-universe model (Johnson, 1989; Laprie, 1985)

an error, which occurs in the informational universe. An error is any deviation from correctness or accuracy. It can be represented by parity errors or by typographical mistakes. Finally, the third universe is the external universe that contains failures. A failure is simply the nonperformance of some expected action. Hence, a failure is the result of both a fault and an error. An example of a failure would be a quarter toll machine accepting dimes as payment in full.

It is the cause and effect relationship among the components of the three-universe model that implies two important parameters: fault latency and error latency. Fault latency is simply the amount of time between the occurrence of a fault and its resulting error. Similarly, error latency is the amount of time between the occurrence of an error and its resulting failure.

In explaining the three-universe model, the causes of faults are described as are their differences with errors and failures. In order to completely understand the characteristics of faults, which is required for accurate dependability modeling, additional fault attributes must be examined. There are five major characteristics that are critical in describing faults: cause, nature, duration, extent, and value (Nelson, 1982). These characteristics are shown in Figure 11.

The fault cause is broken down into its four primary types. The fault nature simply specifies the type of fault present. The fault duration defines the length of time in which a fault is active. A permanent fault is active indefinitely; a transient fault appears and disappears within a short period of time; an intermittent fault appears and disappears repeatedly. The fault extent defines whether or not a fault is local to a given piece of hardware and/or software or if it affects the hardware and/or software globally. Finally, the fault value is either determinant, which means the fault value remains unchanged throughout a given time interval, or it is indeterminate, which means its value can change over time.

In order to achieve high levels of reliability and safety in a system, it is paramount that the number of faults that manifest themselves as errors and failures must be minimized. In order to achieve this minimization, certain design considerations must be weighed and implemented. Among the various design methods that are used to achieve dependability are defense in depth, redundancy, diversity and robustness.

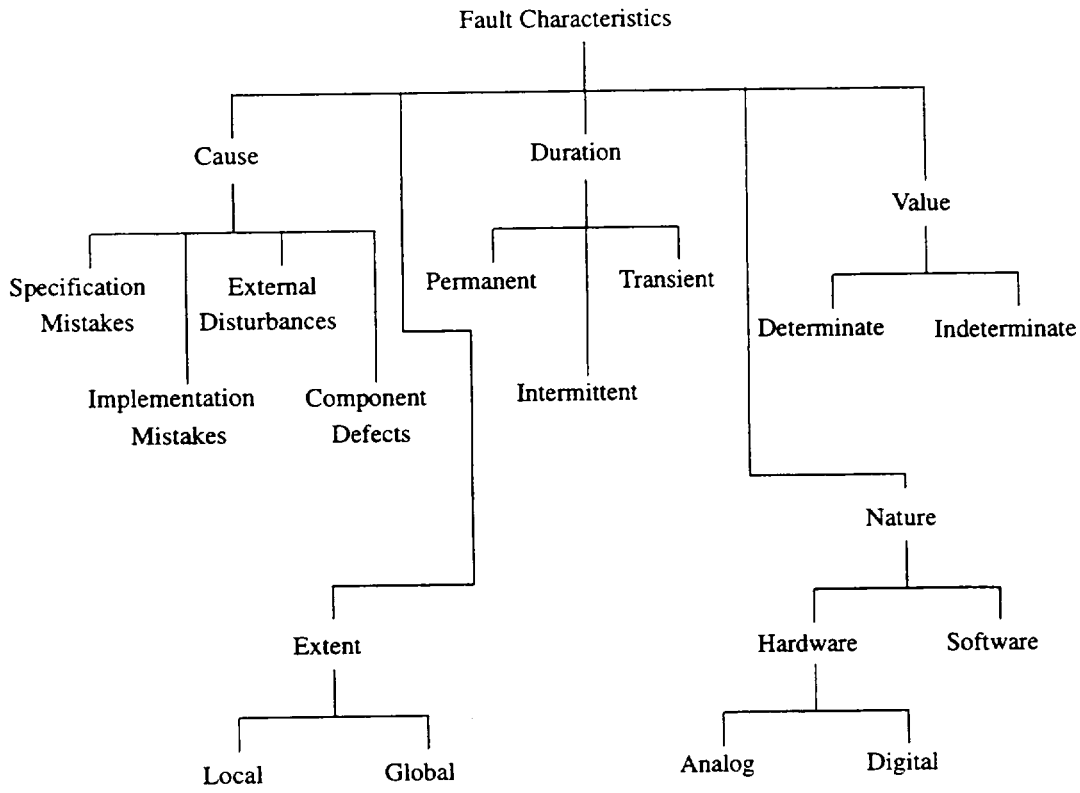


Figure 11: Fault characteristics (Nelson, 1982)

2.1 Defense in Depth

Defense in depth approaches safety by implementing multiple protective echelons in a concentric nature within the design; that is, system safety is achieved by separating safety features such that the failure of one echelon does not disable the safety systems contained within other echelons. Hence, defense in depth assumes independence of the echelons (NUREG/CR-6303). The purpose of defense in depth is to compensate for potential human and system failures by providing successive barriers to prevent system failure (DOE-STD-3009-94) using redundancy and/or diversity. If a failure occurs that affects multiple echelons, then the independence assumption is no longer valid.

2.2 Redundancy

Redundancy is the addition of information, resources or time beyond what is needed for normal system operation.The four types of redundancy techniques are hardware redundancy, software redundancy, time redundancy and information redundancy as shown in Figure 12.

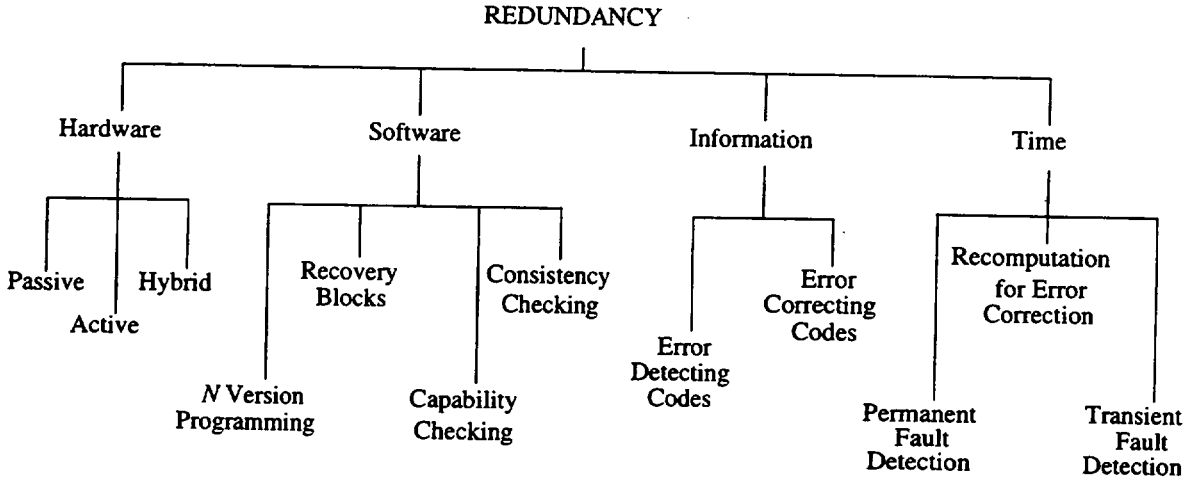


Figure 12: Redundancy schemes (Johnson, 1989)

2.2.1 Hardware Redundancy

Hardware redundancy is the addition of extra hardware for the purpose of detecting or tolerating faults (Johnson, 1989). There are three basic forms of hardware redundancy: passive, active and hybrid. Passive techniques use the principle of *fault masking*, which simply prevents faults from resulting in errors. Such approaches are designed to achieve dependability without requiring any action by either the operator or the system. Most passive hardware redundancy approaches implement the concept of majority voting as shown in Figure 13. In this approach,

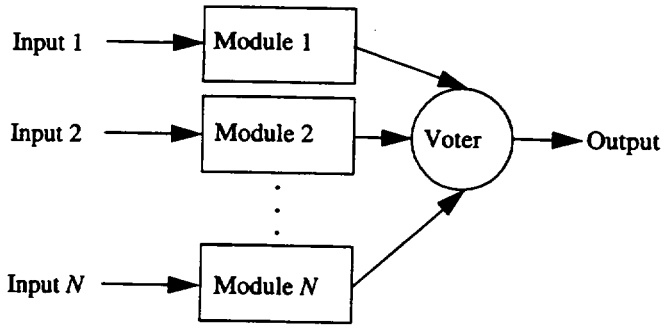


Figure 13: Passive hardware redundancy example

N is selected as an odd number so that a majority voting arrangement can be used. Hence, passive techniques do not require fault detection or system reconfiguration.

Active hardware redundancy requires fault detection, fault location and fault recovery. Fault masking cannot

be achieved with this approach; that is, this approach does not attempt to prevent faults from manifesting themselves as errors within the system. Hence, active approaches are used in applications that can tolerate momentary erroneous results. An example of this approach can be found in Figure 14. In this example, duplication with comparison is

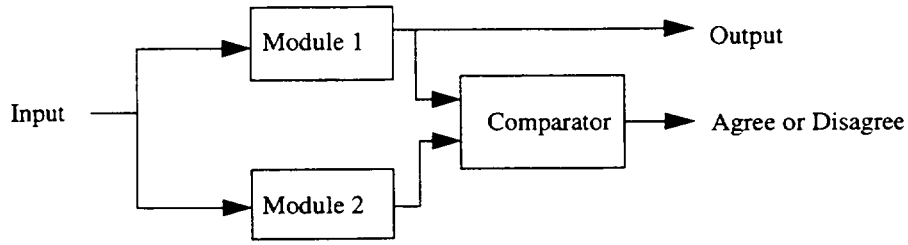


Figure 14: Active hardware redundancy example

implemented. The basic concept of this approach is the development of two identical pieces of hardware, performing the same computations in parallel, and comparing the results of the computations. If a disagreement arises between the outputs, then an error message is generated.

The fundamental concept of hybrid hardware redundancy is to combine the attractive features of both the active and passive approaches. Fault masking is used to prevent the system from producing erroneous results, and fault detection, fault location and fault recovery are used for system reconfiguration in the event of a fault occurrence.

2.2.2 Software Redundancy

Software redundancy can be achieved in many ways, including consistency checks, capability checks and *N*-version programming. A consistency check uses *a priori* knowledge about information to verify its correctness; that is, it is a check for consistency between actual results and expected results. A consistency check can be implemented in either hardware or software, but it is typically a software feature (Johnson, 1989). An example of consistency checking is that the generation of a memory address within a computer should never lie outside the address range of available memory.

Capability checks demonstrate whether or not a system possesses its expected capabilities; that is, it is the verification of an expected capability or feature. An example of capability checking is a simple computer memory test. A processor can write specific information to specific memory locations and then read the information stored in those locations to verify that the data is properly stored and retrieved.

The concept of *N*-version programming (Chen and Avizienis, 1971) is to design and to code the various software components *N* times and to compare the *N* results. Each of the *N* components is designed and coded by separate groups of programmers from the same set of specifications. It is hoped that by developing *N* independent

software designs, the same mistakes will not be made by the N different groups of programmers. However, the Knight and Leveson experiment (Knight and Levenson, 1986) suggests that the assumptions that the same errors will not be made may prove to be untrue; that is, different programmers are apt to make the same mistakes. Hence, there may not be complete independence among the various programs.

2.2.3 Time Redundancy

Time redundancy uses additional time to perform functions to reduce the need for additional hardware. Time redundancy is used in systems where time is less critical than the cost of additional hardware. The basic concept of time redundancy is the repetition of computations in such a manner to allow for fault detection. It can be used to distinguish between permanent and transient faults.

2.2.4 Information Redundancy

Information redundancy is the addition of extra information to allow for fault detection or fault masking. Typical examples of information redundancy are error detecting and error correcting codes that are formed by the addition of redundant information to data words. It consists of codes and codewords, where in general, a code is a means of representing information via a set of well-defined rules.

2.2.5 Redundancy Limitations

The implementation of these four redundancy techniques is very dependent upon the application. Additionally, all types of redundancy assume component independence; that is, each component, either hardware or software, acts independently in the system while performing its task. However, the simultaneous failures of the redundant components violate this independence assumption. It has been shown in (Dugan, 1992) that such correlation among the components greatly affects their availability and decreases the benefit of a redundant system. As a result of such failures, system safety features that are used for system protection can be defeated and the benefits provided by redundancy are mitigated. Additionally, it is demonstrated in Section 1.1: Quantitative Differences Among the Dependability Metrics that the system dependability metrics are sensitive to both redundancy and to the coverage parameter. Therefore, it is imperative that consideration must be given both to the amount of redundancy that is contained in the design of an embedded digital system and the required level of coverage that is needed to meet the required level of dependability.

2.3 Diversity

The principle of diversity is to provide several ways of detecting and responding to a significant event by

using different logic, algorithms, or means of actuation. In achieving these responses and detections, the independence of different subsystems is assumed. Typically, diversity can be implemented in one of the six following ways: human diversity; design diversity; software diversity; functional diversity; signal diversity; and equipment diversity (NUREG/CR-6303).

2.3.1 Human Diversity

Considering the effect of humans during the different stages of building a safety system, it has been determined that human diversity is very helpful. Typically, human diversity is implemented by having separate designers and redundant divisions of maintenance personnel to reduce the possibility of replicating same error. It is hoped that this division will lessen the possibilities of a human error creating a catastrophic situation. However, it has been demonstrated that humans are apt to make the same mistakes, and as a result, the same errors have been replicated in separate designs.

2.3.2 Design Diversity

In design diversity, different design approaches in both hardware and software are implemented. By using different designs approaches, it is hoped that the same errors that can be produced from common influences will not be introduced in the various designs. From using different designs, it is hoped that each design will be independent in nature having different failure modes. However, a factor that weakens this argument is that different designs can still use similar approaches or elements, and as a result, there is not complete independence among the designs.

2.3.3 Software Diversity

Software diversity implies that the use of different programs designed and implemented by different designers will not repeat the same errors, which is analogous to the concept of *N*-version programming. Again, it is shown experimentally in (Knight and Levenson, 1986) that the assumptions that the same errors will not be made may prove to be untrue. As a result, different programs designed and implemented by different designers may not be totally independent.

2.3.4 Functional Diversity

In functional diversity, systems use multiple subsystems to perform different physical functions. These various subsystems are designed to isolate their functionality so that the failure of one subsystem does not affect another subsystem. Even though these subsystems perform different functions, the combination of these subsystems may have overlapping safety effects to prevent system failure. The implementation of functional diversity can be useful if a combination of the functionally diverse systems can mitigate the effects of failure; that is, they prevent the

occurrence of a catastrophic event in the case of a fault occurrence. It is imperative in implementing functional diversity that each subsystem is completely independent, else the failure of one subsystem could affect another subsystem and possibly compromise system safety.

2.3.5 Signal Diversity

Signal diversity uses different sense parameters to initiate a protective action. Sense parameter are generated within a system and are monitored to determine if the system is operating correctly. These parameters are assumed to be independent such that any one of the parameters may indicate an abnormal condition. The isolation of these parameters is paramount, else the assumption of independence is not valid.

2.3.6 Equipment Diversity

Equipment diversity is the use of different equipment to perform similar safety functions. Such equipment must be sufficiently unalike so that its vulnerability to common failures is decreased. The use of different manufactures' equipment does not guarantee diversity as they may use the same COTS components. For example, most computers use Intel microprocessors. If two personal computers (PCs) using the industry standard architecture (ISA) manufactured by different companies both use the same Intel microprocessor, then equipment diversity cannot be implemented because both PCs contain the same COTS microprocessors.

The use of diverse computer equipment may have an effect on software diversity. For example, the use of different computer architectures among machines to implement equipment diversity forces the use of diverse compilers, linkers and support software. Hence if properly applied, then hardware diversity can necessitate the use of software diversity.

2.3.7 Diversity Limitations

Each of the six major types of diversity assume independence in their implementation. However, certain innate features of each of the diversity methods demonstrates that this independence assumption may be violated. Hence, system safety features that are used for system protection can be defeated if the independence assumption is violated, which is also the same problem that plagues redundancy.

2.4 Robustness

A robust system combines elements of redundancy, diversity and defense in depth to achieve safety and reliability. The underlying premise upon which each of these methods is based is independence. Each method assumes component independence; that is the failure of one component does not affect other components in the

system. However, certain failures classes exist which nullify the assumption of independence (National Research Council, 1997). For example, a failure in a computer memory device could be a stuck bit; that is, one of the binary digits would always have a value of either 0 or 1 and would never change. If this failed device were storing information, then the executing software could be affected by this condition when it retrieved the information from this memory device. Hence, the assumption of independence among components is no longer valid because a fault in digital hardware manifests itself in the software that is executing on the hardware. This type of fault condition is becoming more commonplace today with the proliferation of embedded digital hardware and software systems.

2.5 Embedded Digital Systems

In developing the various schema for designing dependable systems, hardware and software are typically viewed independently. However, today's design trends are to implement systems that perform very complex functions, and by using software in conjunction with hardware, such systems are easier to implement. These systems are called embedded digital systems; that is, an embedded digital system combines digital hardware and software for functionality. By integrating digital hardware and software, it has become easier to implement complex functions that are used in a wide variety of applications. In addition to being used for nuclear I & C, they are also used in life-critical applications, such as aircraft flight control and hospital monitoring devices, where a failure could be catastrophic and could result in loss of life. Embedded digital systems are also used in areas such as banking and the stock market where a failure could result in loss of opportunity, and possibly, in loss of money. Hence, the use of embedded digital systems is pervasive throughout modern society.

However, the inherent attributes of an embedded digital system make it increasingly difficult to implement them and to validate and to verify that the system meets functional requirements and specifications; that is, the increased complexity of embedded digital systems leads to complex designs that are difficult to analyze. There are existing techniques to perform dependability analyses for software and hardware independently, however, minimal research has been performed in modeling software and hardware together. With the proliferation of digital embedded digital systems throughout all aspects of life, there is a definite need for a dependability modeling and analysis methodology for such systems.

3. Probability Risk Assessment (PRA) and Digital Embedded Systems

In broad terms, a PRA entails (Kaplan and Garrick, 1981)

1. qualitatively identifying accident scenarios: What can go wrong?
2. quantifying the likelihood of these scenarios: What is the likelihood?
3. assessing the qualitative consequences of the scenarios: What are the consequences?

This general definition is not limited to any particular form of modeling or application. Rather, PRA is a methodology that can potentially be used in many different applications encompassing a myriad of modeling techniques and tools. One emerging area in the application of a PRA is in the modeling of digital embedded systems, and in particular, digital I & C systems.

For a digital embedded system, there are multiple issues that must be addressed in the development of a PRA, including various questions that need to be answered. Among these questions are the following:

1. What can fail in either the hardware or the software or both?
2. What failure modes can occur?
3. What are the likelihoods or rates of occurrence of the failures?
4. What are the consequences to the digital system's performance with the occurrence of a fault?

The solutions to such questions provides information that can be used in the digital embedded system PRA or in higher level PRAs, such as that for a nuclear power plant. In order to obtain solutions to these and other questions, system level dependability modeling and analysis needs to be performed.

3.1 Dependability Modeling and Analysis

Dependability modeling and analysis of digital embedded digital systems requires the development of axiomatic, empirical and physical models (see Figure 7). Axiomatic models provide quantitative analysis for the dependability metrics and empirical and physical models provide the parameter estimates required by the axiomatic models. The axiomatic models can be used in a hierarchical manner in parallel with system design. By analyzing the systems in parallel with design, the system safety and reliability can be determined at the various levels of design abstraction. As a result of such analysis, the sensitivity of the dependability metrics to various components can be identified early in the design cycle. The benefit of such information can have significant impact on the final system design. For example, design decisions regarding the use of COTS versus application specific components can be made as a result of this analysis.

Typically, little if any detailed modeling information is available for COTS components. Hence, the level of modeling abstraction for such a component is limited. If it can be demonstrated that system dependability is not sensitive to the COTS component, that is the COTS failure does not impact the overall system dependability at the level of abstraction for which modeling information is available, then such a component can be incorporated in the design. However if it is demonstrated the system dependability is sensitive to the failure of a COTS component and a lower level of modeling abstraction is required, then an application specific component would be required in the design to alleviate this problem. Obviously, the selection of the dependability analysis and modeling methodology is

critical in this assessment. The various analysis and modeling techniques that are typically used in dependability analysis include Markov models, fault trees, Petri nets and dynamic flowgraphs.

3.1.1 Markov Models (Karlin, 1975; Cassandros, 1993)

Markov modeling is a technique used for calculating various system dependability metrics, including safety and reliability. It defines a system in terms of the various states in which the components can be at a particular instance of time and it reflects the possible transitions between these various states. The statistical basis for this model is that of a Markov process whose fundamental premises, which are referred to as the memoryless property, are (Cassandros, 1993)

1. all past state information is irrelevant; that is, state memory is not needed
2. the length of time that the current process has been in a given state is irrelevant; that is, state age memory is not needed

As a result of the second aspect of the memoryless property, the inter-event times, that is times at which the system transitions between various states, are exponentially distributed. If the second aspect of the memoryless property is relaxed, then the inter-event times are no longer constrained to be exponentially distributed; that is, the state transitions can occur at any time and the inter-event times can possess arbitrary distributions. Such a process is called semi-Markov.

A given system can be represented by a number of different states which defines the system in terms of the functional capabilities of its components at different instances of time; that is, the state that a given system is in at a particular instance of time reflects which components are operational and which have failed. These states can reflect the condition of either hardware or software or a combination of such components. The state transition parameters that are typically required by such a dependability model are

1. λ: the failure rate of a particular component; the component of interest is usually denoted by a subscript.
2. μ: the repair rate of a particular component; the component of interest is usually denoted by a subscript.
3. C: the coverage factor (please see Appendix A for a detailed explanation).

The dependability metrics, as discussed in Section 1: Introduction, are determined by determining the probability that the system is in a particular state. In order to derive these solutions in closed forms, a series of differential equations must be solved. The solution of these equations can be quite rigorous, and as a result, several software tools have

been developed. A summary of some of the available software tools can be found in Table 5.

Table 5: Markov Model Software Tools

	Manufacturer	*Application Considerations*
CARE III	COSMIC	Cannot model repair.
CARMS	DAINA	Solves Markov, semi-Markov and nonlinear Markov models.
MARKOV1	Decision Systems Associates	
SURE	COSMIC	Calculates upper and lower bounds for semi-Markov models.
REST	NASA	Provides estimate in terms of bounds
PC Availability		
HARP		
SHARPE	Duke University	

3.1.1.1 Application of Markov Models to Digital Embedded Systems

Markov models have been used to analyze computer networks (Bateman et al, 1989) and programmable electronic systems (PES) used in industry to protect and control processes (Stavrianidis, 1993). The availability modeling of a Fiber Distributed Data Interface (FDDI) computer network is presented in (Bateman et al, 1989). Models for both two and three-ring wiring concentrator networks are analyzed. In (Stavrianidis, 1993), multiple, common PES architectures are analyzed to determine what hardware failures affect their overall reliability.

3.1.1.2 Advantages and Disadvantages of Modeling Digital Embedded Systems with Markov Models

Markov models can be used to represent hardware, software and their combined interactions in a single model to provide various information. For example, a Markov model can determine the probability of a system being in a particular state at a particular time and it can provide estimates for both safety and reliability. The behavior of system components is represented as states and the various transitions between the states represent the failure and the repair rates for the various components.

In developing Markov model representations for system analysis, the state space required expands exponentially with the complexity of a system. Hence for certain models, closed form solutions cannot be obtained because of this state space explosion. If a Markov process is used, then all failure and repair rates must be assumed to be exponentially distributed. If different distributions are required for any of the transition rates, then the models are significantly more difficult to solve; that is, the complexity of the resulting differential equations is considerably

greater.

3.1.2 Fault Trees

Fault trees are modeling techniques which have been extensively used in risk and reliability analysis. From a system design perspective, fault tree analysis provides a logical framework for understanding the ways in which a system can fail. A fault tree is not a tree in the graphical sense, rather it is a logical representation of the interrelationships between a potential critical event and its underlying cause(s). Fault tree analysis provides the probability that system failure will occur, that is the critical event, and the listing of all the combination of all events, that is the underlying causes, that will ultimately lead to system failure. Hence, it illustrates the behavioral relationship among the system's various components and how the combination of failures among these various components can ultimately result in system failure. The parameters that are needed to solve for a closed form solution are typically

1. probability of component failure
2. *C*: the coverage factor (please see Appendix A for a detailed explanation).[1]

In the continued refinement of fault tree analysis techniques, two type of fault trees have developed: static and dynamic.

3.1.2.1 Static Fault Trees

A static fault tree is the dual of a reliability block diagram (RBD) (Johnson, 1989; Modarres, 1993). An RBD is a success-oriented network that describes the function of a system in terms of its reliability. In such a network, both series and parallel systems can be represented. Systems that contain a combination of series and parallel components are represented using an appropriate combination of these relationships. A fault tree provides the logical representation of the dual of reliability, unreliability, for such systems, and it too assumes independence among the components. The series functionality for failure in a fault tree is represented by the logical AND and the parallel functionality for failure is represented by the logical OR. Additionally, static fault trees can model other logical representations. For a more detailed list of static fault tree gates, the reader is advised to consult (Modarres, 1993). Using these symbols, the cause and effect relationship between the basic fault event and the top event, system failure, can be realized.

The only sequence dependencies that can be modeled using static fault trees are those that can be represented by a Priority AND. For this gate, an output occurs only if the input faults occur in a specific sequence. Since this gate represents the only sequencing of faults that a static fault tree can model, the applicability of static

1. Not all fault tree analysis tools support coverage.

fault trees is limited; that is, static fault trees cannot fully describe the behavior of systems that can fail due to complex sequencing of faults. There are various software tools available that can provide this analysis and a list of some of these tools can be found in Table 6.

Table 6: Static Fault Tree Software Tools

	Manufacturer	*Application Considerations*
BRAVO	JBF Associates	Commercial version of SAPHIRE
CAFTA+	SAIC	Supports sensitivity analysis
CARE III	COSMIC	
ETA-II	SAIC	Performs analysis via CAFTA+
Formal-FTA	Formal Software	Supports Monte Carlo simulation
FTRAN	Rex Thompson & Partners	Supports Monte Carlo simulation
IMPORTANCE	National Energy Software Center	
RESULTS III	Management Sciences	
RKP606	Innovative Timely Solutions	
SAPHIRE	Idaho National Laboratory	
TRACE	COSMIC	Supports Monte Carlo simulation

3.1.2.2 Dynamic Fault Trees (Dugan et al, 1992)

Certain sequence dependent failures cannot be modeled using static fault trees; that is, failures which result from the ordering of specific events cannot be modeled using static fault trees. Some of these dependencies, which can be modeled using Markov models, have been represented with special gates that complement the existing static fault tree gates. Hence, dynamic fault trees extend the applicability of static fault trees.

Among the sequence dependencies that can be modeled using dynamic fault trees are functional dependencies and sparing. Functional dependencies are considered to be the occurrence of some event, called the trigger event, that causes a set of dependent components to become unusable. Sparing involves the sequencing of events associated with the replacement of a failed component with either a hot, warm or cold spare. The software tools available that can provide this analysis are listed inTable 7.

3.1.2.3 Application of Fault Trees to Digital Embedded Systems

Historically, fault tree analysis was introduced in 1962 at Bell Telephone Laboratories in connection with a safety evaluation of the launching system for the *Minuteman* missile. Subsequently, this technique has been refined by many people and it has been applied to various industries, including aerospace, medical, and nuclear. In particular,

Table 7: Dynamic Fault Tree Software Tools

	Manufacturer	*Application Considerations*
DIFTree	University of Virginia, USA	Supports dynamic gates and static gates; provides a sensitivity analysis
GALILEO	University of Virginia, USA	Supports dynamic gates and static gates

fault trees have been used to successfully analyze various aspects of nuclear power plant safety, such as the 1975 Reactor Safety Study (United States Atomic Energy Commission, 1975). In addition, both static and dynamic fault trees can be used to analyze a variety of embedded system applications, including robot systems (Harpel et al, 1997; Khodabandehloo, 1996), rail interlock systems (Min et al, 1994), cardiac pacemakers (Mojdehbakhsh et al, 1994), avionics subsystems (Shimeall et al, 1991), flight control systems (Shimeall et all, 1991) general computer systems (Dugan et al, 1992; Dugan et al, 1994; Kaufman et al, 1999) and I&C systems in nuclear power plants (Khobare et al, 1998).

(Khodabandehloo, 1996) presents static fault tree analyses of a hydraulic pick-and-place robot and operator-robot interaction in a welding cell. These analyses consider hardware, software and robot-operator interactions using both fault trees and event trees. The fault tree analysis helps in identifying the problem areas to help in reducing failures. In (Harpel et al, 1997), a robot system is also analyzed using static fault trees. In particular, an analysis of the robot's digital control system is performed that includes the effects of coverage. The resulting analysis for various robot configurations is obtained with and without the coverage parameter. From this analysis, the sensitivity of the system's reliability to coverage is effectively demonstrated. In (Shimeall et al, 1991), the micro-processor based flight control system software of an A-6E aircraft is analyzed using fault trees to identify which conditions can lead to an unsafe situation, the inadvertent firing of a live missile during practice flights.

In (Mojdehbakhsh et al, 1994), fault trees are used to model the software safety requirements of a cardiac pacemaker. Fault trees are used in this application to validate and to verify the software requirements. A fault tree analysis technique is applied to determine the reliability of the mission avionics subsystem of the Advanced System integration Demonstration (ASID) project in (Shimeall et al, 1991). In this analysis, multiple independent sub-trees are identified and solved separately to reduce the complexity of the fault trees. The solutions for each of these sub-trees are integrated in the final fault tree to provide a single system reliability estimate.

Fault trees can be used to provide a comparative analysis of different reliable architectures which use both hardware and software, such as distributed recovery blocks, recovery block and *N*-version programming (Dugan et al, 1994). In (Dugan et al, 1992), the functions and hardware associated with an avionics system are analyzed using dynamic fault trees to derive system reliability. Similarly, static fault tree analysis is used to prove that microprocessor based fail safe system is a feasible option in (Min et al, 1994). This analysis establishes the safety

concerns in a rail interlock system, thus allowing the identification of the situations under which the safety assurance relay is to be released. In (Kaufman et al, 1999), the reliability of an embedded control system for the Kuiper Airborne Observatory's oscillating secondary mirror is obtained using dynamic fault trees. Even though software failure information is not available for this system, the sensitivity analysis provided by the fault tree allowed for the identification of the software components that are most problematic to the overall system reliability.

In (Khobare et al, 1998), a programmable digital comparator system found in nuclear power plant I&C systems, which consists of a triplicated microcomputer system, is analyzed using static fault tree. This system has replaced a large number of alarm meters by generating the contacts that are used in various safety related functions. On the generation of a command signal, which is derived using 2 out of 3 voting, the control rods fall under gravity. The operation of this system is derived from the embedded system; that is, its operation results from the interaction of the software and the hardware. Although there is a provision to take into account software reliability during the analysis, software failure rates are still being determined and reliability values are as yet to be determined with the desired level of confidence.

3.1.2.4 Advantages and Disadvantages of Modeling Digital Embedded System with Fault Trees

Static and dynamic fault trees can represent hardware, software and their interaction in embedded systems. Static fault trees are limited in their application because certain functional and sequence dependencies cannot be modeled. Dynamic fault trees overcome this limitation with the addition of logic gates that model such dependencies. The analysis from either type of fault tree provides only the likelihood of system failure and the combination of component failures that can trigger system failure. Fault trees do not provide any state information or timing. Additionally, fault trees cannot sufficiently model repair because the introduction of feedback to the model can create an inconsistency that prevents solving for the top event probability.

3.1.3 Petri Nets (Murata, 1989; Peterson, 1981; Cassandros, 1993)

Petri nets are graphical representations of the interaction between system components that represent their dynamic behavior. They are closely related to finite state automata (Cassandros, 1993; Kohavi, 1978; Rosen, 1995) in that they both manipulate events according to a set of specific rules. Petri nets visually demonstrate the relationship between system components, including both hardware and software. Over the years, the concept of a Petri net has been extended to incorporate timing, stochastic processes and information flow.

The Petri net concept has been extended to incorporate timing information and these models are called Timed Petri nets. Timed Petri nets allow the inclusion of timing information of real time systems to be included in the scope of the modeling (Leveson et al, 1987). Hence, the sequencing and scheduling of system behavior can be modeled. Stochastic Petri nets (SPNs) and generalized stochastic Petri nets (GSPNs) allow for additional timing

information and probabilistic information to be modeled to better represent the dynamic behavior of a system. Colored Petri nets (CPNs) produce information flow models to provide a convenient way of representing a system's functional behavior (Rao et al, 1996; Rozenberg, 1990).

The graphical techniques that are used in all types of Petri nets allow systems to be analyzed for various properties, including

1. reachability: can a system ever reach a certain state, and if so, what event occurrences place a system in this state.
2. recoverability: if a system reaches a certain unsafe state, can the system successfully recover?

Hence, all Petri nets qualitatively demonstrate system safety. The information that can be processed from Petri nets can be obtained using existing software tools and a sampling of the available software tools can be found in Table 8.

Table 8: Petri Net Software Tools

	Manufacturer	*Application Considerations*
Artifex	Artis Srl, Italy	Supports Petri nets with time
CPN-AMI	MASI Laboratory, University of Paris, France	Supports place/transition nets and colored Petri nets
DSPNexpress	GMD-FIRST Technical University of Berlin, Germany	Supports stochastic Petri nets
EDS Petri Net Tool	Defense Research Agency (Farnborough) UK	Supports Petri nets with time, place/ transition nets and stochastic Petri nets
MISS-RdP	ISI, Toulouse, France	Supports Petri nets with time, place/ transition nets and stochastic Petri nets
WebSPN	University of Catania, Italy	Supports stochastic Petri nets and non-Markov model Petri nets

3.1.3.1 Application of Petri Nets to Digital Embedded Systems

Timed Petri nets allow for the modeling of systems which depend heavily on timing information, such as communication systems (Berthismieu et al, 1991), real time systems, such as flexible manufacturing systems (FMS) (Peng et al, 1993) and aircraft flight control systems (Shimeall et al, 1991). In (Berthismieu et al, 1991) a communications protocol operation is modeled to reduce errors due to loss of messages and time-out values. This model illustrates the use of timed Petri nets to evaluate critically time-dependent properties. The FMS (Peng et al, 1993) consists of manufacturing cells and a material handling mechanism, controlled by computer hardware and real time software, which requires a control program to satisfy the timing requirements for different functions. The controlling hardware-software system is modeled by a modified timed Petri net and the controlled mechanical portion

is modeled by an activity diagram. In (Shimeall et al, 1991), the micro-processor based flight control system software of an A-6E aircraft is analyzed using timed Petri nets to identify which conditions are required for missile launches in both combat and practice runs.

GSPNs and SPNs can be used in performance evaluation of multiprocessor systems (Marsan et al, 1984; Jin et al, 1995). The multiprocessor systems in (Marsan et al, 1984) have performance constraints due to overhead introduced by the supervisory program, contention for use of resources and concurrent process synchronization. Hence, early analysis is essential in order to avoid conflicts at later stages. GSPNs and SPNs can model timing related features of computer systems, such as concurrency and synchronization. Other features such as communication and interaction between different processors can also be accommodated (Marsan et al, 1984). The SPN model in (Jin et al, 1995) is for a system that consists of plural workstations and a database server. The model takes into account the CPUs and the processes running on them. SPNs have also been used to diagnose computer systems (Shedler et al, 1993). These computer systems are composed of hardware components, operating system software and application programs. The system is designed as a fault tolerant computer system with spare system units for repair and the availability of the system can be evaluated.

Petri nets can be combined with other types of analysis techniques, such as a failure modes and effects analysis (FMEA), to provide requirements analysis. This method has been applied to real time embedded control systems in the Hughes aircraft company (Goddard, 1996). Safety analysis begins at the conceptual design stages and goes on to perform preliminary hazard avoidance and analysis and requirements analysis.

3.1.3.2 Advantages and Disadvantages of Modeling Digital Embedded Systems with Petri Nets

Petri nets provide a graphical representation of system behavior. This behavioral representation can include both hardware, software and the interaction of these components. From this model, the system safety can be demonstrated using simulation without the need for an analytical solution. If stochastic Petri nets are used, then the timing requirements and the probabilities associated with the transitions between states can be incorporated in the model to better represent the actual system behavior. Since Petri nets are simulation based, they do not need analytical representations for the system behavior. Hence, they cannot provide the probabilities associated with being in a particular state nor can they provide closed form solutions for reliability or safety. In order to provide such quantitative information, very complex differential equations must be derived and solved. Such equations can be generated by translating the Petri net to a Markov model, which may or may not provide a closed form solution. Additionally, Petri nets can be difficult to analyze if the system under consideration creates large reachability graphs (Leveson et al, 1987; Goddard, 1996); that is, if the system complexity creates a large reachability graph, the visual analysis is limited and the ability to obtain quantitative solutions poses the same difficulties associated with a Markov model.

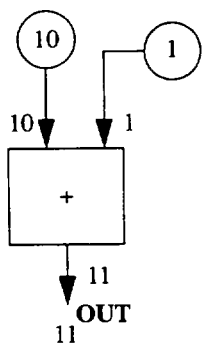
3.2 Hybrid Modeling

Hybrid modeling consists of using simulation and algorithmic models to determine various parameters required by dependability models. In lieu of building physical prototypes, the functional and behavioral aspects of a digital embedded system can be simulated in software using HDLs. In order to determine the effects of faults on the system, a series of fault injection experiments (see Section 1.2.2) can be performed on the simulated system. The effects of the injected faults can be analyzed using algorithmic models such as data flow (Choi, 1997) and dynamic flowgraphs (Garrett, Yau, Guarro and Apostolakis, 1995; Garrett, Guarro and Apostolakis, 1995) to derive parameters for the dependability models.

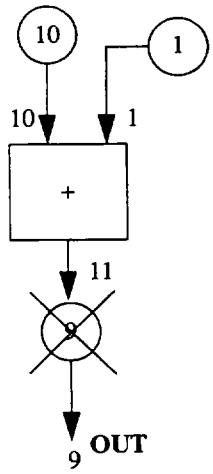
3.2.1 Data Flow (Choi, 1997)

The data flow technique provides ability to model a system using a data flow representation which is a mapping of the flow of data through the system. The system can be modeled based on its behavioral description using a HDL. The data flow approach models the informational flow obviates the need to distinguish between the hardware and software components to allow for unified modeling of the hardware/software system.

Additionally, the data flow representation also allows for fault injection by deliberately corrupting the value using a construct or component written in a HDL which performs the fault injection. For example, consider a system consisting of a sensor providing a processor with a value to be incremented and then dispatched as depicted in Figure



(a) data flow description



(b) data flow description with fault injection

Figure 15: Data flow description and data flow with fault injection

15. The behavioral description of the system is in (a) and a depiction of a fault injected in the processor is shown in

(b) where the value going to the output is *9* instead of *11*. Using the data flow construct, the system behavior can easily be represented in diagrammatic form allowing for the incorporation of fault injection. Through the use of HDLs, tags can be created on the data values, which mark them as *faulty* or *non-faulty*, *detected* or *undetected*, and this information can be collected and post processed to determine the effects of the faults on the system, and subsequently, various model parameters.

3.2.2 Applications of the Data Flow Methodology to Embedded Systems

In (Choi, 1997), the concepts of data flow methodology were applied to various triple-modular redundant (TMR) systems using majority voting (Johnson, 1989) as shown in Figure 16. In these examples, the functional

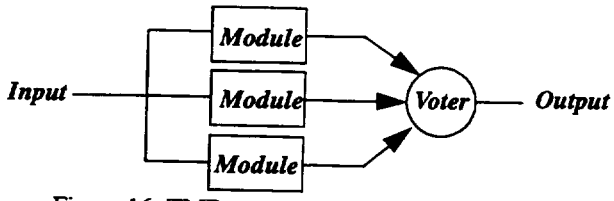


Figure 16: TMR system using majority voting

separation between the voter hardware and software components was varied, to determine the effects on system dependability, and in particular, the coverage parameter associated with the voter. The data flow methodology provided the ability to model this functional separation using a behavioral description of the system, and with fault injection, it also provided the data required for estimating voter coverage for the various configurations.

3.2.3 Advantages and Disadvantages of Modeling Digital Embedded Systems with Data Flow

Data flow modeling uses a behavioral description of an embedded system that is generated with a HDL. Using this behavioral representation, fault injection experiments can be performed to determine parameters required by any dependability model. Since there is no dependability modeling capabilities within the data flow methodology itself, it can be used with any of the three axiomatic models described in Section 3.1: Dependability Modeling and Analysis.

The use of HDLs in developing the behavioral model mitigates the need for testing a physical prototype, which can provide substantial costs savings, and such a model can easily modified to reflect design changes. Additionally, the behavioral description alleviates the need to distinguish between hardware and software components in the dependability analysis; that is, the embedded system can be modeled to include the implicit dependencies between the hardware and the software components.

3.2.4 Dynamic Flowgraphs (Garrett, Yau, Guarro and Apostolakis, 1995; Garrett, Guarro and Apostolakis, 1995)

Dynamic flowgraph methodology (DFM), which is derived from logic flowgraph methodology (Guarro and Okrent, 1984), is an integrated approach to modeling and analyzing the behavior of software driven embedded systems for the purpose of reliability and safety assessment. DFM follows the fault tree approach but it incorporates the dynamic behavior of embedded systems; that is, it incorporates timing information. It utilizes a modeling framework in which system models are developed in terms of the causal relationships between physical variables and temporal characteristics of the execution of software. DFM produces a self-contained system model from which many fault trees can be derived using an algorithmic technique.

DFM is based on multi-valued parameter discretization and logic. Hence, the analytical insight of fault trees derived from this methodology is not limited by the constraints of binary logic used in conventional fault tree analysis. The DFM can produce timed fault trees in which timing relations between key system and parameter states are systematically and formally taken into account. Hence, the fundamental goals of this methodology are the identification of how certain postulated events occur and the identification of an appropriate testing strategy based upon an analysis of the system's functional behavior.

3.2.5 Application of Dynamic Flowgraphs to Digital Embedded Systems

DFM has been used to analyze combustion module system experiments (Yau et al, Final Report, 1995), digital flight control systems (Yau et al, 1995) and various systems in nuclear reactors (NUREG/CR-6465). Specifically, the safety-critical portion of a laminar soot process experiment system in a combustion module system (Yau et al, Final Report, 1995) is modeled using the DFM. In (Yau et al, 1995), the digital flight control system of the Titan II space launch vehicle is modeled using the DFM. In (NUREG/CR-6465), the applicability of DFMs to analyzing digital control systems found in advance reactors is presented. In these models, there is no distinction made between the software and the hardware elements being modeled. As a result, certain hardware features are represented as software characteristics.

3.2.6 Advantages and Disadvantages of Modeling Digital Embedded Systems with Dynamic Flowgraphs

DFM takes into account the dynamic nature of embedded systems and allows for an analysis of the combined hardware and software components. It provides the ability to perform both inductive and deductive analyses, which can provide detailed insight on the overall system safety. However, it makes no distinguishing separation between digital hardware and software; that is, it classifies digital hardware as software during analysis.

The generated fault trees incorporate timing information, which demonstrates the dynamic behavior of a system, but they use multi-valued logic for analysis. Multi-value logic can be difficult to understand and it can create a complex solution space that can be difficult to solve.

3.2.7 Comparison of Data Flow and Dynamic Flowgrapgh Modeling

The similarities between data flow modeling and the DFM lie in their system modeling techniques; that is, they both depict the flow of information through a system. However, the two methodologies differ in the applications because DFM focuses on design verification, fault and failure analysis and test vector generation analysis and data flow is a methodology that allow for the derivation of specific parameter information. DFM is an analytical modeling tool which uses timed multi-valued fault trees to obtain system failure and behavior information through prime implicants of the fault tree. Data flow modeling, however, focuses on deriving information pertaining to how faults effect system behavior. Once this information is derived and parameter estimates made, than any of the modeling techniques presented in Section 3.1: Dependability Modeling and Analysis can be used to derive the system dependability metrics.

3.3 Parameter Estimation (Kaufman, 1997)

Regardless of the method used to determine the various dependability metrics, the quality of these metrics depends upon the accuracy of the parameters used. Typically, the parameters of most interest for dependability modeling are failure rate, repair rate and coverage. The estimates for these parameters can be found in a myriad of ways, including simulation based approaches as discussed in Section 3.2: Hybrid Modeling. However, great care must be taken to ensure accuracy of these estimates.

3.3.1 Failure Rate

The failure rate, λ, is the expected number of failures per a given time period (Shooman, 1968). For example, if a computer fails on average once every 10,000 hours, its failure rate is 1/10,000 failures per hour. The failure rate of a system can be obtained through either analytical or statistical models. Analytical modeling examines the failures of actual devices by applying appropriate models. The most common analytical technique used for electrical components is based upon the United States Department of Defense MIL-HDBK-217 (United States Department of Defense). From this handbook, the failure rate of many items, such as integrated circuits and printed circuit boards, can be estimated or predicted by using the supplied models and specifying the required parameters.

Component failure rates can also be determined using statistical models. This type of modeling is achieved by collecting data from components that have been in service for a very long period of time. This approach can only

be used in limited situations. Alternatively, system prototypes can be built and tested to obtain failure rate information, however, there are many limitations to prototype testing. Prototypes can be costly to construct, and the time for construction and testing may not fit in the design cycle. Also, depending on the failure rate, experimental testing may be impractical because of the time required to obtain the number of required data points.

3.3.2 Repair Rate

The repair rate, μ, is the expected number of repairs per unit time (Johnson, 1989). For example, repair technicians might be able to fix a failed computer system at a rate of two every hour. Hence, the system repair rate is two repairs per hour. System repair rates are typically determined using experimental data collected over a period of time. As an example, consider a constructed system that has faults injected into it and that these faults manifest themselves as failures. Repair technicians with varying skill levels and qualifications are sent to repair the system. The data from their work is collected and the repair rates can then be calculated. It should be noted that in order to accurately estimate the repair rate, a sufficient number of faults must be injected into the system (Johnson, 1989).

3.3.3 Fault Coverage[1]

Fault coverage, C, is the conditional probability that a system recovers given that a fault has occurred (Bouricius et al, 1969). The impact of the fault coverage parameter on the dependability metrics is well documented in (Arnold, 1973; Bouricius et al, 1969; Dugan and Trivedi, 1989; Johnson, 1989) and in Section 1: Introduction of this document. Hence, there is a definite need for an accurate estimate of this parameter. Of all of the dependability parameters mentioned, coverage is the most difficult to estimate. There are very few analytical methods available to estimate coverage. Those methods that do exist require very specific parameters that are difficult, if not impossible, to know *a priori*. Since exhaustive testing is often impractical, fault coverage estimation is typically performed by using a statistical model which requires data points that are generated via system simulation (Law and Kelton, 1991).

This statistical approach begins with selecting a random fault from the entire fault set of the system. This fault is either injected into a system prototype or model and then the prototype or model is monitored to determine the effects of the fault. The number of fault injection experiments (Arlat et al, 1990; Arlat et al, 1991; Daehn, 1991; Segall et al, 1988) that must be performed is directly related to the required fault coverage of a given system.

In order to achieve a high degree of fault coverage, a large number of injection experiments must be performed. If the desired coverage estimate cannot be met due to resource limitations, then the system must be redesigned so that the designer can show that the system has the required level of dependability. Hence, there is a need for a means to estimate a given system's coverage factor so that the designer can demonstrate that the system

1. See Appendix A for a detailed coverage modeling discussion.

meets the required level of dependability via metrics such as reliability, safety and availability.

4. Common Mode and Common Cause Failures

Certain uncovered failures can defeat system redundancy, diversity and defense in depth and nullify the assumption of independence (National Research Council, 1997). These failures, which can be called multiple independent failures, can be caused by severe environmental conditions, design errors, calibration errors, and/or functional deficiencies (Dhillon, 1983). Such failures are known as common-mode and common-cause failures and these failure must be identified during the dependability analysis. A timeline depiction of the evolution of these failure definitions can be found in Figure 17.

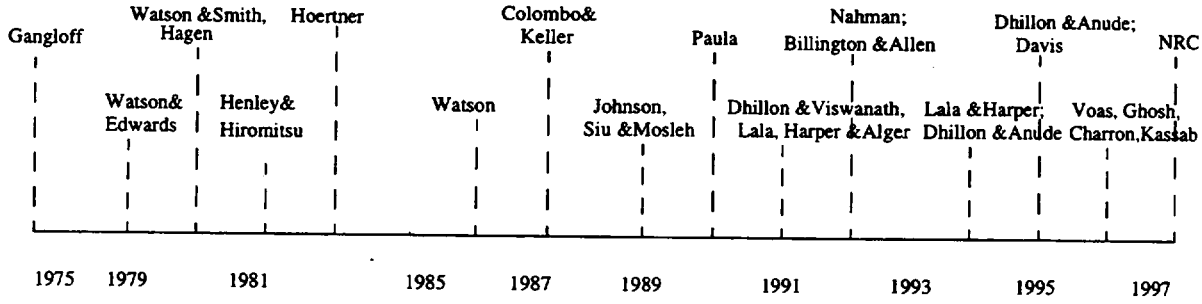


Figure 17: Timeline of the CCF and CMF definitions

4.1 Common-Mode Failure

The evolution of common-mode failure (CMF) definitions demonstrates the distinctions that have been historically made between hardware and software assessment. The vast majority of CMF definitions were developed to explain failure phenomena in hardware systems. For example in (Gangloff, 1975), CMFs are defined as any instance where multiple hardware units or components fail due to a single cause. This definition, however, omits important information, such as the time of failure and the mode of failure. (Watson and Edwards, 1979) define CMFs as the result of an event which, because of dependencies, causes a coincidence of failure states of components in two or more separate channels of a redundant system preventing the system from performing its intended function. This definition takes into account the time of failure, but it still does not include any failure modes. (Colombo and Keller, 1987; Lala and Harper, 1994) incorporate all aspects of failure, such as time of occurrence, type of cause, and mode of failure, by defining CMFs as multiple, concurrent, and dependent failures of identical equipment that fails in the

same mode. In (National Research Council, 1997), a CMF definition is presented that defines this identical equipment failing in the same mode as the stuck open mode or the fail as-is mode. In this definition, a stuck open failure mode refers to a failure caused by a break in an input, output, or intermediate logic line in a digital circuit and a fail as-is condition indicates the failure of modules in a manner such that they maintain their value when the fault occurred. In (Johnson, 1989), a CMF occurs when two or more identical modules are affected by a fault in exactly the same way at exactly the same time, when there exists at least one input combination for which the outputs of two modules are erroneous. The lone CMF definition from a software perspective is a failure which occurs when two or more software versions fail in exactly the same way for the same input (Voas, J., A. Ghosh, F. Charron, and L. Kassab).

Recently, there have been attempts to identify CMFs in terms of the embedded hardware/software system. In (Davis, 1995), the causal relationship between hardware and software failures is examined. A time delay in the propagation of a fault and error from a software unit to a hardware unit or the converse defines this causal relationship. Hence, CMFs are related failures of redundant or separate equipment embracing all such causal relationships. The equipment may fail due to severe environments, design errors, calibration and maintenance errors, hardware and software, operating systems, and consequential failures.

As these definitions for CMFs evolved, efforts were made to describe common-mode faults, which cause CMFs, to aid in defining design methodologies in safety-critical systems. A common-mode fault is defined as one that affects multiple fault containment regions (FCRs) simultaneously or nearly simultaneously (NASA, 1992). A FCR is a collection of components, either software and/or hardware, that operates correctly regardless of any arbitrary logical or electrical fault outside the region; that is, faults inside an FCR cannot propagate outside the FCR and faults outside the FCR cannot affect it (Lala et al, 1991).

4.2 Common Cause Failure

Another type of failure that needs to be identified during dependability analysis is common-cause failure (CCF). As is true for the CMF definitions, these definitions have evolved with a primary interest in explaining hardware failure conditions. In (Watson and Edwards, 1979), CCFs are defined as multiple failures, potential or real, attributable to a common-cause. This CCF definition is very similar to the CMF definition in (Gangloff, 1975). Similarly, (Watson and Smith, 1980) defined CCFs as the multiple failures of identical, redundant components that fail in the same mode within a critical time period, and result in complete system failure; that is, all failures are attributable to a common cause, which fails each of the multiple components directly. In (Henley and Hiromitsu, 1981), they studied the underlying events that created CCFs are analyzed. From this analysis, the basic common-cause events as the common-mode events of the cause. In general, CCFs are defined as the simultaneous, independent

failure of two or more components due to a shared cause or a single event that renders all the components unavailable and causes system or mission failure (Billington and Allen, 1992; Dhillon and Anude, 1994; Dhillon and Viswanath, 1991; Hagen, 1980; Nahman, 1992; National Research Council, 1997; Paula, 1990; Siu and Mosleh, 1989; Watson, 1986).

4.3 CMF and CCF Comparison

The distinction between CMF and CCF is often misunderstood due to the overlapping of their definitions, as is seen from (Gangloff, 1975) and (Watson and Edwards, 1979). There are some well known cases which have been analyzed to differentiate between these failure types and an overview of some of the more prominent failures can help to delineate between these failure types. Three examples cases of CCF are the Browns Ferry fire, the Rancho Seco plant reactor shutdown and the Three Mile Island incident (Hagen, 1980) and they are presented to highlight these failure modes. For CMFs, a nuclear power plant failure is presented (Hagen, 1980).

The Browns Ferry fire was initiated by a small lighted candle that shut down the reactor. A fire in the cable room disabled many electrical power and control circuits. In the Rancho Seco plant, a burned out push button lamp precipitated a sequence of events that shut down the reactor. During the lamp replacement, the operator dropped it into a monitoring assembly that had been removed for inspection. This carelessness created an electrical short, which cut power to two-thirds of the instrumentation and control process, causing a pressure transient to trip the reactor. The common failure of auxiliary feedwater valves was instrumental in initiating coolant loss accident at Three Mile Island.

An example of a CMF is a system in which, after almost 4.5 years of partial commercial operation, a design error was revealed when a package of diodes in a rod drive system failed during a required trip of the nuclear reactor. This failure coupled with the protection and control functions caused all other control rod systems to fail. Hence, the designed redundancy was negated. Some functional diversity which was present in the design did shut down the system (Hagen, 1980).

In light of these examples and the aforementioned definitions, a question arises as to whether CMFs should be classified under CCFs or vice versa. One approach is to call failures of redundant components due to the same cause and present at the same time as CMFs, and subsequently distinguish the CMFs as: (1) CCFs, which are failures of functionally independent redundant components caused by an external basic event; (2) causal or secondary failures, which are failures of redundant components that originate as an independent failure which then propagates via secondary effects resulting in additional failures; and (3) failures of redundant components caused by a single component failure and propagated via functional dependencies (Hoertner, 1982). In contrast, CMFs can be characterized as a subset of CCFs, characterized by multiple items failing in the same mode (Dhillon and Anude,

1994).

Another approach to classifying CCFs and CMFs is to consider both as being on the same hierarchical level. Using this assumption, both can be defined to be the possibility of system or mission failure involving multiple item failure, which may occur due to a common cause. The loss of system functions, or multiple, redundant paths, or components due to an underlying common mechanism, fault or phenomena can be attributed to either CCFs or CMFs (Watson, 1986).

4.4 Limitations of Existing CMF and CCF Definitions with Regards to Embedded Digital Systems

The majority of existing CMF definitions make a distinction between hardware and software components of a system. The existing definition for CMFs in terms of embedded digital system is based upon causality, which assumes that there is a time lag in propagation of faults and errors from a software unit to a hardware unit or vice-versa. In today's complex systems, the interaction between hardware and software is virtually indistinguishable. The time delay for fault and error propagation between hardware and software in modern embedded digital systems is typically immeasurable. Hence, it is hard to attribute system failure to the occurrence of either a hardware or a software fault. Additionally, these existing definitions can be contradictory, and as a result, they are difficult to apply. Hence, there is a definite need for coherent CMF and CCF definitions in terms of embedded system analysis.

4.5 Recommendations for Modeling Embedded Digital Systems for CMFs and CCFs

For embedded digital systems, the need to model CMFs and CCFs is an intrinsic part of performing accurate dependability analysis. In performing dependability analysis, careful attention must be given to the modeling technique used. The various axiomatic models have certain characteristics that may or may not lend themselves to modeling CMFs and CCFs.

Since timing is an important issue in characterizing CMFs and CCFs, the axiomatic modeling technique used must support this parameter. Markov models do not include timing information, that is the time at which failures occur cannot be modeled, but they do allow for the sequencing of events to be modeled, which can be used to represent both CCF and CMF. However, these resulting models can be quite complex, and as a result, it may be very difficult, if not impossible, to obtain a closed form solution. Additionally, these models require certain parameters, such as the failure rates for CMFs among components, that are difficult, if not impossible to calculate. Hence obtaining an accurate closed form solution for this model may not be realizable. Since Petri nets are statistically equivalent to Markov models, they have the same limitations in applicability to modeling CMFs and CCFs.

It has been shown in (Yau et al, Final Report, 1995; Yau, Guarro and Apostolakis, 1995) that dynamic flowgraphs can be used to model CMFs and CCFs in software, and by extension, such failures in embedded hardware and software systems. The difficulties with this modeling technique is the use of multi-valued logic. Multi-valued logic can be quite complex, and as a result, difficult to understand. Similarly, it has been shown in (Kaufman et al, 1999) that fault trees can also be used to model embedded digital systems. By using dynamic fault tree structures, sequencing of events and their functional dependencies are modeled. These type of modeling constructs can also be applied to CMFs and CCFs. However, static fault trees do not incorporate timing, which limits their applicability. Hence, a mixture of existing modeling techniques might provide the best modeling environment for incorporating the affects of CCFs and CMFs in dependability modeling. Depending on which dependability model(s) is (are) selected, the accuracy of the parameters required for solution directly impacts the quality of the solution. Therefore, it is imperative that accurate parameter estimations be made. The proposed methodology for accurate estimation of the parameters required for safety and reliability modeling of the dependencies between hardware and software in embedded digital systems is the *hybrid approach* mentioned in Figure 1 on page 1 and defined in *Section 3.2: Hybrid Modeling*.

In the *hybrid modeling approach*, behavioral prototypes of the embedded digital system will be constructed using HDLs. The effects of various failures, including CCFs and CMFs, will be determined using error injection experiments. The error injection experiments will perturb the correct operational environment for the embedded system by corrupting the information that is being processed and their effects on the system will be quantified. The quantitative parameter that is associated with how these errors effect a system is *coverage*. Using the coverage parameter and other parameters obtained form these experiments, it is believed that accurate estimates for safety and reliability can be made for embedded digital systems from their axiomatic models, regardless of the modeling technique used, and these estimates can be applied to the system PRA.

APPENDIX A Coverage Modeling (Kaufman, 1997; Kaufman and Johnson, 1998)

1. Introduction

Using the fault characteristics and the cause and effect relationship between faults, errors and failures as presented in the three-universe model, fault coverage models can be developed. Since a small change in coverage can result in great variations in dependability metrics, it is imperative that an accurate estimate of fault coverage be made. There are both mathematical and qualitative expressions for fault coverage. The mathematical definition is that fault coverage, C, is the conditional probability that a system recovers given that a fault has occurred (Bouricius et al, 1969). It is written as

$$C = P(\textit{fault processed correctly} \mid \textit{fault existence}). \tag{A.1}$$

Qualitatively, coverage is a measure of the system's ability to detect, locate, contain and recover from the presence of a fault. There are four primary types of fault coverage available: (1) fault detection coverage; (2) fault location coverage; (3) fault containment coverage; and (4) fault recovery coverage. Thus, the term *fault processed correctly* implies at least one of the four coverage types. A more detailed description of the fault coverage types follows.

Fault detection coverage is the system's ability to detect a fault. For example, a typical system requirement is that a certain percentage of all faults must be detected. The fault detection coverage is then a measure of the system's ability to meet the desired fault detection requirement. Fault location coverage measures a system's ability to locate the cause of faults. A typical system requirement is that faults within replaceable components must be located. Hence, fault location coverage is a measure of the success with which such faults are located. Fault containment coverage measures a system's ability to contain faults within a predefined boundary. For example, if a fault in a sub-system is detected and located, then preventing the effects of the fault from propagating in the system is a measure of fault containment coverage. Finally, fault recovery coverage measures the system's ability to automatically recover from faults and to maintain the correct operation. If a system is required to possess a high fault recovery coverage, then it must also possess high fault detection, fault location and fault containment coverages (Johnson, 1989).

The type of coverage required is highly application specific. For example, fail-safe systems require specific knowledge of the fault detection coverage. Conversely, highly-reliable systems that use sparing techniques (Johnson, 1989) require knowledge of the fault recovery coverage. Regardless of the type of coverage information that is required by a system, the methodology used to estimate the coverage parameter is the same. Throughout the remainder of this paper, fault coverage is defined to mean any of the four fault coverage categories that are required for a given application.

Fault coverage is examined in two distinct ways: (1) coverage modeling and (2) parameter estimation. As its name implies, fault coverage modeling is a development of a model for the response of a component to the occurrence of a fault. Parameter estimation is needed for values that are required by coverage models. The parameters can be estimated by inserting faults into a given system prototype or model and collecting the required data.

1.1 Axiomatic Models of Fault Coverage

Axiomatic modeling of fault coverage is a behavioral representation of a system's response to faults. These models are embedded in the overall system model, and the actual number of coverage models required is a function of the system under test. There have been numerous refinements to the axiomatic fault coverage models and the various models that have been developed are presented in the following sections. These models are categorized into two sections: error handling without time limitation and error handling with time limitations.

1.1.1 Error Handling Without Time Limitations

The initial iteration of fault coverage models ignores any type of interference that could occur during error handling, and typically consist of various forms of Markov and semi-Markov models. In these models, it is assumed that the time spent in states handling errors is negligible with respect to the time spent in states where errors are not present.

1.1.1.1 Permanent Effective Error Model (Dugan and Trivedi, 1989)

The model, shown in Figure A.1, depicts the effect of a fault and its resulting error. The fault coverage for

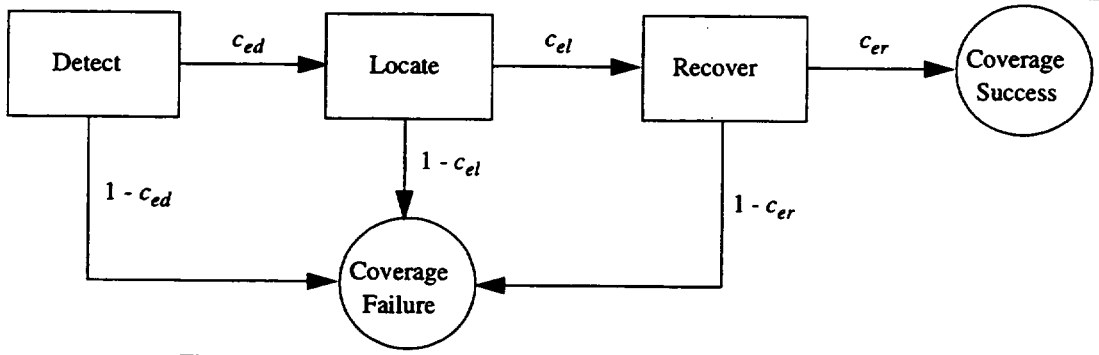


Figure A.1: Permanent effective error model (Dugan and Trivedi, 1989)

the system is given by

$$C = c_{ed} \times c_{el} \times c_{er}, \tag{A.2}$$

where c_{ed} is the error detection probability, c_{el} is the error location probability, and c_{er} is the error recovery probability. Since this model only handles permanent faults and ignores transient faults, it has very limited

applicability to real systems.

1.1.1.2 CAST Fault Coverage Model (Conn et al, 1977)

CAST, shown in Figure A.2, combines transient fault restoration and permanent fault recovery. Faults occur at a rate $\lambda + \tau$, which is the sum of the permanent and the transient fault rates respectively. Once a fault occurs, the detection state is entered with an error detection probability of c_{ed}. If the errors are not detected, then system failure occurs. However if the errors are detected, then transient recovery is attempted. The transient recovery probability is $1 - l$, where l is the transient leakage. If the transient recovery fails, then permanent recovery is attempted. In permanent recovery, the fault cause is located with probability c_{fl} and the system recovers with probability c_{sr}. If permanent recovery is successful, then $N - 1$ modules remain. If it is unsuccessful, then system failure occurs. The n subscript associated with all of the system parameters simply denotes the number of active components.

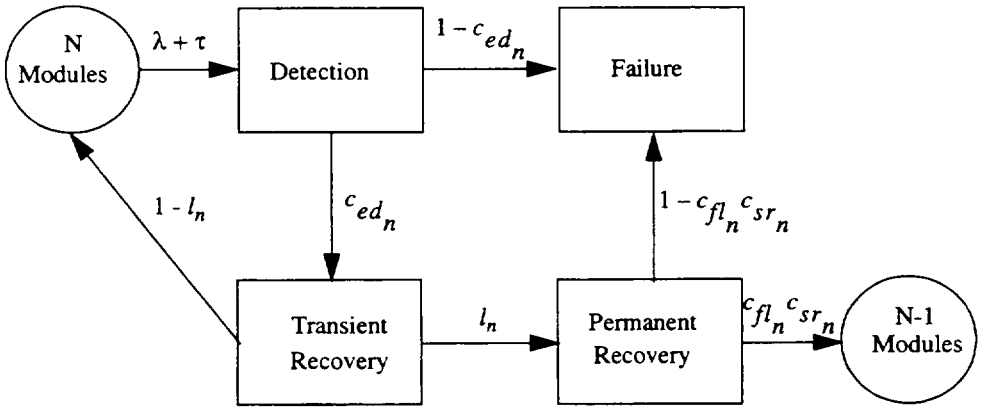


Figure A.2: CAST fault coverage model (Dugan and Trivedi, 1989)

1.1.1.3 CARE III Fault Coverage Model (Stiffler and Bryant, 1982)

The CARE III single-error model, shown in Figure A.3, is a generalized fault model realizing intermittent or permanent faults. In this model, state *A* represents the activation of an error. State *B* represents the error latency, where α_β and β_α are the transition rates between states *A* and *B*. State *P* represents the effects of the error polluting the system and occurs from state *A* at rate ρ. State *D* represents error detection, which can only occur if the error is active (state *A*) or it is polluting the system (state *P*). The rate at which an active error is detected before it can become latent or pollute the system is Δ, and the rate at which an error that is polluting the system becomes detected is $c_{ed}e_d$. If the error that is polluting the system is not detected, the error results in a failure, which is state *F*, at a rate of $(1 - c_{ed})e_d$ from state *P*. The probability of exit from state *A* to State *D* is given by

$$C = \frac{\Delta}{\Delta + \rho} + \frac{c_{ed}\rho}{\Delta + \rho} = \frac{\Delta + c_{ed}\rho}{\Delta + \rho}. \qquad \text{(A.3)}$$

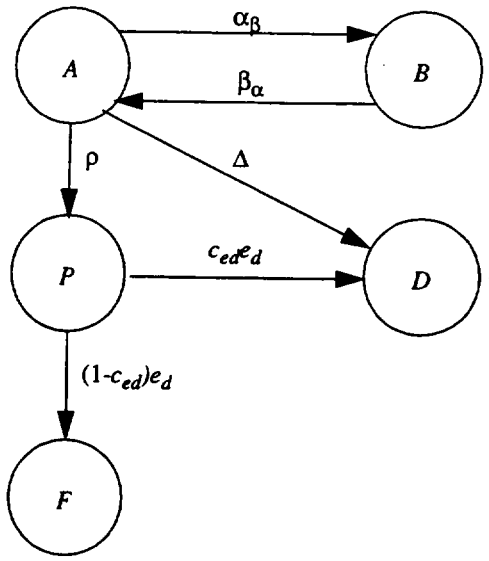


Figure A.3: CARE III single-error fault model (Conn et al, 1977)

In order to model permanent errors, α_β and β_α must be set to zero, which is the rate at which an effective error goes latent and vice versa; else, this model represents intermittent errors.

1.1.2 Error Handling With Time Limitations

In order for coverage models to be robust, consideration must be given to the lifetime of the fault and/or error. If the transient lifetime is considered, which in reality is a major concern, the models described in Appendix A Section 1.1.1: Error Handling Without Time Limitations have very little applicability in developing accurate fault coverage estimation. The following models consider transient lifetime.

1.1.2.1 ARIES Fault Coverage Model (Makam and Avizienis, 1982)

The ARIES model, shown in Figure A.4, includes permanent, transient and intermittent faults. In this model, there are three possible exits: (1) system crash; (2) normal processing; and (3) permanent fault recovery. Obviously, the system crash exit occurs when the error introduced by a fault causes system failure. The probability of a fault resulting in immediate system failure is $1-c_{sr_1}$. The fault recognition and attempted recovery probability is $f_{fr_i}i$, where i denotes the recovery phase. The number of allowable recovery phases is fixed. If during a given recovery phase the system fails, then the system crash exit is taken with probability PF_i. If during a recovery phase the system recovers from a transient error, then the normal processing exit is taken with probability PR_i. Finally, if all recovery phases are entered and successful, then the permanent fault recovery exit is taken.

The ARIES fault coverage probability is

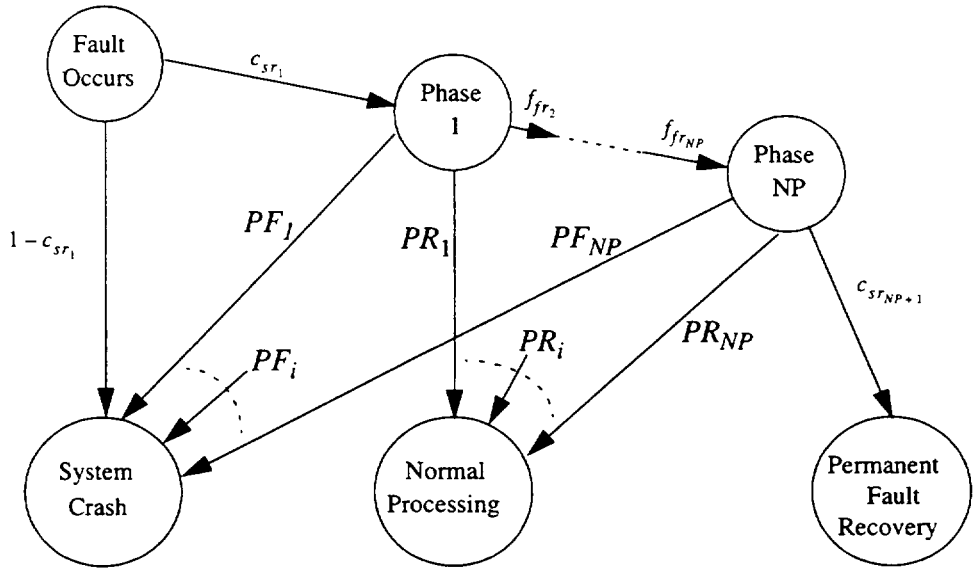


Figure A.4: ARIES fault model (Dugan and Trivedi, 1989)

$$\begin{aligned} C &= \text{transient restoration probability} + \text{permanent error recovery} \\ &= \sum_{i=1}^{NP} PR_i + c_{sr_{NP+1}} \times (\text{Coverage of Permanent Recovery Procedure}) \end{aligned} \tag{A.4}$$

If the transient lifetime is long, then successful recovery may not have a fixed probability and the expression for PR_i must be modified.

If the transient lifetimes are exponentially distributed random variables (rv) and the duration of each recovery phase is a constant, the expression for transient restoration becomes

$$\begin{aligned} PR_i &= Pr[\text{Phase } i \text{ entered}] \times Pr[\text{Phase } i \text{ successful}] \\ &\quad \times Pr[\text{transient gone before phase } i \text{ begins}] \\ &= c_{sr_{i-1}} \times ER_i \times (1 - \exp(-(T_1 + T_2 + \ldots + T_{i-1})/D)) \end{aligned} \tag{A.5}$$

where the transient error lifetimes have a mean D with various durations, T_i, and ER_i is the probability of an effective recovery procedure for phase i. This expression can be generalized for non-exponentially distributed error lifetimes as

$$PR_i = c_{sr_{i-1}} \times ER_i \times F_D(T_1 + T_2 + \ldots + T_{i-1}), \tag{A.6}$$

where F_D represents the generalized distribution. If required, this type of generalization can be applied to the recovery phase.

1.1.2.2 Modified CARE III Model (Stiffler and Bryant, 1982)

In (Stiffler and Bryant, 1982), a transient error model is discussed in which transient lifetimes are assumed to be exponentially distributed. The duration of each recovery phase is assumed to be independent and identically distributed (iid), which is more restrictive than the ARIES model. However, it allows a random number of recovery phases, and like ARIES, it accommodates general distributions for the recovery phase. The CARE III model has been further refined to include transient, intermittent or permanent faults (errors), the effect of transient lifetimes and it can be solved for both the Markov and the semi-Markov case.

1.1.2.3 Extended Stochastic Petri Net (ESPN) Fault Coverage Model (Dugan and Trivedi, 1989; Trivedi et al, 1984)

The ESPN model combines both local and global timing. This model includes the limited recovery time associated with real systems, and the ability to determine the effects of near-coincident errors (McGough, 1983) that can occur during attempted recovery. The only near-coincident errors of interest are those whose occurrence can interfere with the current recovery. It is conservatively assumed that the occurrence of a near-coincident fault will result in system failure.

Since a stochastic Petri net is used, the various fault distributions can be generalized. If all faults are exponentially distributed *rv*, then the Petri net can be converted to a Markov model and solved accordingly. If the failure rates are not exponentially distributed, then in some cases the resulting model is semi-Markov and in other cases simulation is required.

1.1.3 Limitations of Axiomatic Coverage Models

As the development of axiomatic coverage models evolved, their ability to accurately model complex failure recovery mechanisms, such as the duration of an error with consideration given to its lifetime, expanded. In all of these models, however, there is one common thread: the model transition probabilities are unknowns. But these models are useful in determining which parameters are important, so that those gathering data know what data to collect.

It is impossible to know without actually testing a system the values of the various transition probabilities. In some circumstances, it may be impossible to ascertain the exact values. If feasible, a series of fault injection experiments can be performed on a physical model to try to obtain estimates for some of these transition probabilities. Since testing the system for all possible faults is intractable, some type of sampling of the complete fault set for a system is required. Unless some empirical analysis is performed, such an estimation cannot be made. *Expert opinion* can be used to generate an abridged fault set, but it impossible to demonstrate that such a fault set is complete to guarantee an accurate coverage estimate. Similarly, *expert opinion* can be used to simply predict the various model recovery and failure rates and the accuracy of such predictions is highly subjective. Additionally, each fault detection

and recovery mechanism that resides in the system can require its own fault coverage model. Hence, the size and the complexity of a given axiomatic model increases relative to the size and the complexity of the system under test.

1.2 Empirical Models for Fault Coverage Parameter Estimation

The use of empirical models for fault coverage estimation requires detailed statistical analysis that must address four important questions (Powell et al, 1995):

(1) How can the fault coverage value be accurately estimated?

(2) How can any error in the estimate be quantified?

(3) How are fault samples selected?

(4) How can accurate estimates for fault coverage be obtained in a reasonable time?

As previously discussed, empirical models are used to estimate parameters used by axiomatic model. Empirical modeling relaxes many of the assumptions and restrictions, such as parameter estimation, present in axiomatic models. Parameter estimation requires that the system fault space be sampled in some random fashion to provide a representative sample of the entire fault set. Using the data collected from this sampled set, statistical analysis is performed to analyze the accuracy of the resulting estimated parameters. It is shown in (Arlat et al, 1993) that this technique can be used for predicting the system's expected fault coverage. There are numerous sampling strategies available, including techniques that attempt to reduce the variance of the estimate. This type of sampling is referred to as a variance reduction technique (VRT).

The purpose of VRTs is to increase the accuracy of the parameter estimate so that the required number of sample points can be further reduced. VRTs exploit some attribute of the system to increase the accuracy of the parameter estimate(s). Importance sampling, multi-stage sampling, stratified sampling and regression analysis are all examples of VRTs (Cochran, 1977; Hammersley and Handscomb, 1964).

1.2.1 Fault Coverage (Constantinescu, December 1995; Powell et al, 1993; Powell et al, 1995; Smith et al, 1995; Smith et al, 1997; Wang et al, 1994)

The mathematical model used to describe a fault processing event is

$$\begin{aligned} C &= P(\textit{fault processed correctly} | \textit{fault existence}) \\ &= p_{Y|X}(y|x) \\ &= \frac{p_{X,Y}(x,y)}{p_X(x)} \end{aligned} \tag{A.7}$$

where $X \equiv \textit{faults existence}$, and $Y \equiv \textit{fault processed correctly}$. Since coverage is based upon a series of fault

occurrences, that is a fault existence and its subsequent correct processing, the conditional probability in Equation Equation (A.7) can be considered to represent a series of discrete events. Hence, the expected value of the conditional probability for coverage can be modeled as

$$\begin{aligned} E[Y|X=x] &= \sum_{x,y\in\Omega} y \cdot P(Y=y|X=x) \\ &= \sum_{x,y\in\Omega} y \cdot p_{Y|X}(y|x) \end{aligned} \tag{A.8}$$

where Ω is the system's complete fault space. Typically, a fault processing event is considered as a Bernoulli *rv* defined as

$$y = \begin{cases} 1 \ \forall \text{ covered faults with probability } p_{Y|X}(y|x) \\ 0 \ \forall \text{ uncovered faults with probability } 1\text{- } p_{Y|X}(y|x) \end{cases} \tag{A.9}$$

During testing, there is a possibility of *no-reply*, which is the inability to obtain measures from some elements in a sample (Cochran, 1977). Such problems arise when certain faults remain *hidden* when introduced to a system or it may be impossible to introduce a specific fault. In the preceding model, no-reply faults are not included. To remove this source of possible statistical error, the indicator function given in Equation (A.9) is redefined as:

$$y_i = c(x_i) = \begin{cases} 1 \ \forall \quad \text{covered faults} \\ 0 \ \forall \ \text{uncovered faults} \\ \partial \ \forall \ \text{no-reply} \end{cases} \tag{A.10}$$

and the analysis either counts the faults as covered, uncovered or discards the experiment. Substituting the expression for y found in Equation (A.9) into Equation (A.8) yields

$$\begin{aligned} E[Y|X=x] &= \sum_{x,y\in\Omega} y \cdot p_{Y|X}(y|x) \\ &= p_{Y|X}(y|x) \\ &= C \end{aligned} \tag{A.11}$$

Similarly, the variance of the conditional probability for coverage is

$$\begin{aligned} Var[Y|X=x] &= E[Y^2|X=x] - (E[Y|X=x])^2 \\ &= C(1-C) \end{aligned} \tag{A.12}$$

Obviously, neither the *pmf* associated with a fault's existence nor the joint *pmf* associated with a fault's existence and

recovery is known *a priori*. As a result, a fault coverage experiment is necessary to determine a coverage value.

Theoretically, coverage can be determined by injecting the entire sample of N faults, which are assumed to be independent, from the fault space into a given system and calculating the ratio of properly handled faults, d, against the number of injected faults; that is (Daehn, 1991),

$$C = \frac{d}{N}, \tag{A.13}$$

The expression d, is analogous to that found in Equation (A.9); that is, the number of properly handled faults in a fault injection experiment can be modeled as a summation of a series of Bernoulli trials. Hence,

$$d = \sum_{i=1}^{N} y_i, \tag{A.14}$$

where y_i is the Bernoulli *rv* as defined in Equation (A.9) for the i^{th} fault injection experiment and Equation (A.13) can be rewritten as

$$C = \frac{1}{N}\sum_{i=1}^{N} y_i, \tag{A.15}$$

Since it is impractical to inject every fault into a system, an experiment must be developed to provide an unbiased estimate of coverage by limiting the number of fault injection experiments.

By limiting the number of fault injection experiments to n, the coverage point estimate is

$$\hat{C}_{y_n} = \frac{1}{n}\sum_{i=1}^{n} y_i, \tag{A.16}$$

which is an unbiased estimator if the equally likely constraint is valid (Hammersley and Handscomb, 1964). Assuming that a large sample size exists, the central limit theorem (CLT) can be applied to approximate the estimator. Assuming that the estimator is Gaussian, it can be shown that its variance is

$$Var\{\hat{C}_{y_n}\} = \sigma^2_{\hat{C}_{y_n}} = \frac{\sigma_y^2}{n}. \tag{A.17}$$

Since the fault coverage and the variance of y are unknown, the point estimate $\hat{C}_{y_n}$ and the variance estimate $\hat{\sigma}^2_{y_n}$ are used in Equation (A.17) (Stiffler and Bryant, 1982) to yield

$$\hat{\sigma}^2_{\hat{C}_{y_n}} = \frac{\hat{\sigma}^2_y}{n} = \frac{\hat{C}_{y_n}(1-\hat{C}_{y_n})}{n}. \tag{A.18}$$

Under these conditions, a two-sided $100\gamma\%$ confidence interval can be defined. The lower bound of the confidence interval, which is the most conservative estimate of fault coverage, is of most interest and is defined in (Papoulis, 1991) as

$$\hat{C}_{y_{n_{low}}} = \hat{C}_{y_n} - \zeta_u \sqrt{\frac{\hat{\sigma}^2_{y_n}}{n}} = \hat{\sigma}^2_y - \zeta_u \sqrt{\frac{\hat{C}_{y_n}(1-\hat{C}_{y_n})}{n}}, \tag{A.19}$$

where ζ_u is the confidence coefficient for a Gaussian distribution.

This statistical approach is the basis for many empirical models, which are reviewed in subsequent sections. In these models, VRTs are applied to provide variance reduction via various sampling techniques.

1.2.1.1 Powell et al Empirical Models (Powell et al, 1993; Powell et al, 1995)

Since exhaustive testing to determine coverage is seldom possible, fault coverage estimation is performed on a representative sample of the entire fault space. There are two approaches for performing this random sampling: sampling from the complete fault space, and sampling from subspace partitions or classes of the complete fault list, which is commonly referred to as stratified sampling.

Representative sampling (Law and Kelton, 1991) consists of sampling with replacement from a group of n faults and is applicable to non-partitioned sampling. Its unbiased coverage estimator and variance are

$$\hat{C}_y = \frac{1}{n}\sum_{i=1}^{n} y_i \frac{p_{X,Y}(x_i, y_i)}{p_X(x_i)} \tag{A.20}$$

$$Var\{\hat{C}_y\} = \frac{1}{n}\left(\sum_{x_i \in \Omega}\left[y_i \frac{p^2_{X,Y}(x_i, y_i)}{p_X(x_i)}\right] - C_y^{\,2}\right). \tag{A.21}$$

If the sample selection is chosen such that $p_X(x_i) = p^2_{X,Y}(x_i, y_i)$, then the estimate for the mean found in Equation (A.20) is equivalent to the point estimate found in Equation (A.16). Similarly, the variance for this estimate is given by Equation (A.17) and the lower side of the confidence interval is given by Equation (A.19).

1.2.1.2 Partitioned Space Sampling (Stratified Sampling)

Rather than sampling from the entire fault list, the sampling can occur from partitioned classes (Cochran, 1977; Hammersley and Handscomb, 1964; Law and Kelton,1991). By definition, the classes form M disjoint subsets such that

$$E = \bigcup_{j=1}^{M} E_j \text{ such that for every } i, j, \quad i \neq j, \quad E_i \cap E_j = 0. \tag{A.22}$$

where E is the entire fault list. The coverage factor as expressed in Equation (A.8) can be rewritten as

$$\begin{aligned} C &= \sum_{j=1}^{M} \sum_{x,y \in E_j} y \cdot p_{Y|X}(y|x) \\ &= \sum_{j=1}^{M} \sum_{x,y \in E_j} y \cdot p_{Y|X_j}(y|x_j) \cdot p_{X_j|X}(x_j|x) \\ &= \sum_{j=1}^{M} p_{X_j|X}(x_j|x) \sum_{x,y \in E_j} y \cdot p_{Y|X_j}(y|x_j) \\ &= \sum_{j=1}^{M} p_{X_j|X}(x_j|x) c(X_j) \text{ where } c(X_j) = \sum_{x,y \in E_j} y \cdot p_{Y|X_j}(y|x_j) \end{aligned} \tag{A.23}$$

Using this partitioned sampling space, two different sampling techniques can be implemented: the naive estimator and stratified sampling.

The *naive estimator* samples an equal number of faults from each class. For each sample, the coverage estimate for each class i is

$$\hat{C}_{na} = \frac{1}{n} \sum_{i=1}^{M} d_i = \frac{d}{n}, \tag{A.24}$$

The estimator's variance is

$$Var\{\hat{C}_{na}\} = \frac{1}{nM} \sum_{j=1}^{M} \left(c(X_j) - c^2(X_j)\right). \tag{A.25}$$

If all fault occurrences are not equally probable, then this estimator is biased. It can be shown that

$$E\{\hat{C}_{na}\} = \hat{c}(X_j) = \frac{1}{M} \sum_{j=1}^{M} c(X_j), \tag{A.26}$$

hence this technique provides a *naive estimator.*

The covariance between the coverage, C_{E_j}, and the fault occurrence probability, $p_X(X_j)$, for each class is

$$S_{CP} = \frac{1}{M} \sum_{j=1}^{M} (c(X_j) - \hat{c}(X_j)) \cdot \left(p_{X_j|X}(x_j|x) - \frac{1}{M}\right), \tag{A.27}$$

from which it can be proven that

$$\hat{c}(X_j) = C - MS_{CP}. \tag{A.28}$$

Depending on the sign of the covariance, the fault coverage estimator can be either pessimistic or optimistic. This result is proven with actual examples in (Powell et al, 1993; Powell et al, 1995).

In *stratified sampling* (Cochran, 1977; Hammersley and Handscomb, 1964; Law and Kelton, 1991), a number of samples, n_j, for each class, E_j, is pre-selected and a representative sample of size $n_i = n_j$ is taken for each class. The coverage factor now applies to the class rather that the complete sample space and is expressed as

$$\hat{C}_{\Omega_i} = \frac{d_i}{n_i}, \tag{A.29}$$

The coverage and variance estimates are

$$\hat{C}_{st_E} = \sum_{i=1}^{M} p_{X_i|X}(x_i|x) \cdot \hat{c}(X_i), \tag{A.30}$$

$$Var\{\hat{C}_{st_E}\} = \sum_{i=1}^{M} p_{X_i|X}(x_i|x) Var\{\hat{c}(X_i)\}. \tag{A.31}$$

Similarly, the variance of the class coverage estimator is given by

$$Var\{\hat{c}(X_i)\} = \frac{1}{n_i - 1}(\hat{c}(X_i) - \hat{c}^2(X_i)). \tag{A.32}$$

From these variance expressions, it can be seen that the variance depends upon the class sample size. To minimize the system coverage factor's variance, $Var\{\hat{C}_{st_\Omega}\}$, each class' sample size must be defined as

$$n_i = p_{X_i|X}(x_i|x)n. \tag{A.33}$$

This type of sample size allocation is referred to as a *stratified sample with representative allocation.* Using the expressions found in Equation (A.29), (A.30) and (A.33), the system coverage estimator can be expressed as

$$\hat{C}_{stR} = \frac{d}{n}, \tag{A.34}$$

which is equivalent to that for the *naive estimator.* The variance, however, differs. If the expression n_j is substituted into (A.31) and (A.32), the resulting variance is

$$Var\{\hat{C}_{stR}\} = \frac{1}{n}C - \frac{1}{n}\sum_{i=1}^{M} p_{X_i|X}(x_i|x)c^2(X_i). \tag{A.35}$$

Hence, the precision of representative stratification is not sensitive to the covariance between the coverage and the

fault activity occurrence probability for each class. As a result, there is an appreciable gain in precision for coverage estimation and this is substantiated with examples from (powell et al, 1993; Powell et al, 1995) This gain in precision is demonstrated via the improvement in the confidence interval obtained by using the variance provided by Equation (A.35) in Equation (A.19). However, the fault activity occurrence probability is an unknown, and as a result, it is difficult for representative stratification to be used accurately.

1.2.1.3 No-Reply Problem

To accommodate the no-reply problem, *a posteriori* stratification is introduced. This method uses available system information in considering different stratification techniques. Since structural information is circuit dependent, the selected stratification technique is viable only for the circuit under test. Hence, this methodology cannot be extended to a general application.

1.2.2 Cukier et al Model (Cukier et al, 1997)

The Cukier et al model is an extension of the work performed by Powell et al. In this work, fault coverage is modeled in terms of the uncovered faults. This non-coverage estimate, $\hat{\bar{C}}$, using representative sampling is

$$\hat{\bar{C}} = \frac{\bar{d}}{N} \tag{A.36}$$

where $\bar{d}$ are the number of uncovered faults and N is the number of fault injection experiments performed. The expression for non-coverage shown in Equation (A.36) is analogous to the expression for coverage shown in Equation (A.13). Similarly, the coverage expression for partitioned space sampling, which is shown in Equation (A.23), can be expressed in terms of non-coverage as

$$\bar{C} = \sum_{j=1}^{M} p_{X_j|X}(x_j|x)\bar{c}(X_j) \tag{A.37}$$

Two distinct approaches for non-coverage estimation can be made using either classical statistical methods or Bayesian techniques.

1.2.2.1 Classical Statistical Approach

The upper $100\gamma\%$ confidence limit estimator for $\bar{C}$ is defined as

$$P[\bar{C} \le \bar{C}_\gamma(X)|\bar{C}] = \gamma \tag{A.38}$$

In modeling non-coverage for an ultra-dependable system, it is shown in (Powell et al, 1996) that approximated estimations using the classical statistical approach are not valid. Hence, approximations cannot be used when

developing non-coverage estimators based upon Equation (A.38).

To allow for the multiple classes during the fault injection experiments and to minimize Equation (A.37), the upper $100\gamma\%$ confidence limit estimator for $\bar{C}$ for M classes is given by the solution of

$$\prod_{i=1}^{M}\left(1-\sum_{x_i'=0}^{x_i}\binom{n_i}{x_i'}\bar{c}_i^{x_i'}(1-\bar{c}_i)^{n_i-x_i'}\right)\forall i\in\{1,\ldots,M\},\bar{c}_i\in[0,1] \tag{A.39}$$

1.2.2.2 Bayesian Approach

In Bayesian theory, non-coverage, $\bar{C}$, and the class non-coverages, $\bar{C}_i$, are considered as *rv*. The upper $100\gamma\%$ confidence limit is defined by the distribution of the *rv*; that is

$$P[\bar{C}\le\bar{C}_\gamma(X)|X]=\gamma=x \tag{A.40}$$

In order to obtain the confidence limit defined in Equation (A.40), the posterior distribution of $\bar{C}$ is required. For representative sampling, this posterior distribution is

$$f_{\bar{C}}(\bar{c}|X=x) \tag{A.41}$$

and for stratified sampling, the posterior distribution of the non-coverage classes is simply

$$f_{\bar{C}_i}(\bar{c}_i|X_i=x_i) \tag{A.42}$$

In order to solve for the posterior distribution, an appropriate choice of the prior distribution for the non-coverage classes is required.

A beta prior distribution is used for two primary reasons: (1) the number of uncovered faults in each partition is binomially distributed and a beta prior distribution ensures that the prior and the posterior distributions are both from the same family; and (2) when the parameters of the beta distribution equal one, the obtained distribution is uniform over the interval $[0, 1]$, which means that all values of $\bar{c}_i$ have the same weight. The beta prior distributions for $\bar{C}_i$ are

$$f_{\bar{C}_i}(\bar{c}_i)=\frac{\bar{c}_i^{k_i-1}(1-\bar{c}_i)^{l_i-1}}{\beta(k_i,l_i)}\ \text{for}\ 0\le\bar{c}_i\le1,k_i>0,l_i>0 \tag{A.43}$$

where $\beta(k_i, l_i)$ is a beta function with parameters k_i and l_i (Abramowitz and Stegun, 1972).

Since the number of uncovered faults in each partition, X_i, is binomially distributed, then

$$f_{X_i}(x_i|\bar{C}_i=\bar{c}_i)=\binom{n_i}{x_i}\bar{c}_i^{x_i}(1-\bar{c}_i)^{n_i-x_i} \tag{A.44}$$

and the posterior distribution for $\bar{C}_i$ is (Aitchison and Dunsmore, 1975)

$$f_{\bar{C}_i}(\bar{c}_i|X_i = x_i) = \frac{\bar{c}_i^{k_i'-1}(1-\bar{c}_i)^{l_i'-1}}{\beta(k_i', l_i')} \tag{A.45}$$

where $k_i' = x_i + k_i'$ and $l_i' = n_i - x_i + l_i$.

The posterior distribution for $\bar{C}$ is found by combining the posterior distributions of the various $\bar{C}_i$. In (Constantinescu, December 1995), it is shown that an analytical expression for the posterior of $\bar{C}$ is too complicated for more than three classes. Thus, the posterior distribution for $\bar{C}$ can only be obtained using approximation. When a distribution cannot be exactly calculated, it is possible to theoretically exhibit all of the properties of a distribution in terms of its moments (Stuart and Ord, 1987). Similarly, distributions that have a finite number of lower order moments in common approximate each other (Stuart and Ord, 1987). The calculation of moments can be achieved using either the moment generating function (Papoulis, 1991; Stuart and Ord, 1987) or assuming independence among the classes. Once the moments have been calculated, the posterior distribution can be determined from the Pearson distribution system (Johnson and Kotz, 1970).

1.2.2.2.1 Calculation of Moments

The moment generating function of $\bar{C}_i$, assuming s Beta distribution $\beta(k_i', l_i')$, is

$$\phi_{\bar{C}_i} = {}_1F_1(k_i'; k_i' + l_i'; t) \tag{A.46}$$

where ${}_1F_1(k_i'; k_i' + l_i'; t)$ is the confluent hypergeometric function (Abramowitz and Stegun, 1972) and

$$\phi_{p_i\bar{C}_i} = {}_1F_1(k_i'; k_i' + l_i'; p_i t) \tag{A.47}$$

Since the moment generating function of a sum of *rv* is equivalent to the product of the moment generating function of the various *rv* (Johnson and Kotz, 1969), then

$$\phi_{\bar{C}}(t) = \prod_{i=1}^{M} {}_1F_1(k_i'; k_i' + l_i'; p_i t) \tag{A.48}$$

which is derived based upon Equation (A.23). The n^{th} derivative of the moment generating function of $\bar{C}$ for $t = 0$ defines the n^{th} moment of $\bar{C}$. Assuming that the powers of $\bar{C}_i$ are independent, then simpler expressions for the moments of $\bar{C}$ can be obtained. The rth central moment of the beta distribution, $\beta(k_i', l_i')$, using this independence assumption is

$$E[(X-\mu_X)^r] = \frac{\beta(k'_i+r, l'_i)}{\beta(k'_i, l'_i)} = \frac{k'_i(k'_i+1)\ldots(k'_i+r-1)}{(k'_i+l'_i)(k'_i+l'_i+1)\ldots(k'_i+l'_i+r-1)} \quad \text{(A.49)}$$

1.2.2.2.2 Pearson Distribution System (Johnson and Kotz, 1970) for Use as a Posterior Distribution

The Pearson distribution system is a family of seven distributions and are summarized in Table H.1. The

Table H.1: Pearson Distribution System (Johnson and Kotz, 1970)

Type I	beta distribution	Type V	inverse Gaussian distribution
Type II	symmetrical form of the function defined in Type I	Type VI	cumulative Pareto distribution representing *income*
Type III	gamma distribution	Type VII	t distribution
Type IV	no common statistical distributions are of this type; the values required for the CDF are intractable		

seven distributions are represented in a planar plot of their skewness and kurtosis coefficients. From this planar plot, the family to which a given data set belongs is determined. The Pearson distribution pdf, $f(x)$, satisfies a differential equation of the form

$$\frac{1}{f}\frac{df}{dx} = -\frac{pa+x}{pb_0+pb_1x+pb_2x^2} \quad \text{(A.50)}$$

The shape of the distribution is dependent on the values of the four parameters, which can be determined by the first four moments of the distribution $f(x)$. For a detailed summary of this relationship, the reader is advised to see (Abramowitz and Stegun, 1972; Aitchison and Dunsmore, 1975; Cukier, 1996; Cukier et al, 1997; Johnson and Kotz, 1970; Powell et al, 1996; Stuart and Ord, 1987).

1.2.2.3 Comparison of Approaches

The classical statistical and the Bayesian approach are compared for two hypothetical systems, system 1 and system 2 (Cukier et al, 1997), using stratification and simple sampling. It is assumed that the prior distribution for the Bayesian estimation is uniform; that is, the parameters of the beta distribution are equal to one. Initially, the moments used for the Bayesian analysis are calculated using both moment generating functions and the independence assumption.

The initial testing uses homogenous allocation, which requires sampling a predetermined number from each class, and representative allocation, which requires sampling the same number of faults from each class. During this testing the number of fault injection experiments that are performed is varied to determine the validity of the Bayesian approach and to compare it to the classical statistical approach.

During testing, it is shown that only the moment generating function when used with representative

allocation produces valid results for system 1; that is, the posterior distribution is of Type I. Both estimation methods are valid for system 2 when used with representative sampling. Hence, the comparison is performed using the Bayesian method is derived via moment generating functions and using representative sampling. When simple sampling is considered, the Bayesian estimations are more conservative. Using stratification, it is shown that the Bayesian estimation is less conservative than the classical statistical methods. However, this conservatism decreases as the number of fault injection experiments increases.

1.2.3 Fault Expansion Model (Smith et al, 1995; Smith et al, 1997)

Another method for sampling the complete fault space is fault expansion. In fault expansion, the fault space is subdivided into mutually exclusive subsets defined as

$$E_i = \left\{ x_{i1}, x_{i2}, \ldots, x_{i|E_i|} \right\}, \tag{A.51}$$

where E_i is the i^{th} equivalent fault set, x_{ij} is the j^{th} element of the i^{th} fault set and $|E_i|$ is the set cardinality. All equivalent fault sets are disjoint and their union is the complete fault set.

The fault expansion process consists of randomly selecting a fault and determining the set of equivalent faults. All members of E_i are removed from the fault space and fault injection is performed using only one fault. The evaluation of the x_{ik} fault is described mathematically as

$$z_i = c(x_{ij}) = \begin{cases} 1 \forall \text{covered faults} \\ 0 \forall \text{uncovered faults} \end{cases} \tag{A.52}$$

where z_i is the i^{th} sample of the z rv which describes the result of the fault injection experiment for the equivalent fault set E_i. The expected value for coverage is

$$E\{Z|X\} = \sum_{x,z \in E} z \cdot p_{Z|X}(z|x) = \sum_{i=1}^{|E|} \sum_{j=1}^{|E_i|} z_{ij} p_{Z|X}(z_{ij}|x_i) = C, \tag{A.53}$$

which is similar to Equation (A.8). There have been two VRTs developed using fault expansion and they are the Wang et al empirical model (Wang et al, 1994) and the Smith et al empirical model (Smith et al, 1997).

1.2.3.1 Wang et al Empirical Model (Wang et al, 1994)

If fault sampling occurs for the entire fault space, the total number of covered faults after m injections is simply

$$C_m = \sum_{i=1}^{m} X_i, \tag{A.54}$$

Using the binomial distribution for C_m, the $100\gamma\%$ one-sided confidence interval (Cochran, 1977; Trivedi, 1982) for the coverage estimate is

$$P(C_m \geq c_m | dc_c) = \sum_{j=s_m}^{m} \binom{m}{j} dc_c^j (1-dc_c)^{m-j} = 1-\gamma, \tag{A.55}$$

where γ is the confidence coefficient and dc_c is the desired coverage value. It is very difficult to solve Equation (A.55) for dc_c given an arbitrary value of m.

For a system with coverage near one, a Poisson distribution is a good approximation to the binomial distribution. In this case, it can be shown that dc_c is given by (Trivedi, 1982)

$$dc_c = 1 - \frac{1}{2m}\chi^2_{deg;1-\gamma}, \tag{A.56}$$

where $\chi^2_{deg;1-\gamma}$ satisfies $P(Y > \chi^2_{deg;1-\gamma}) = 1-\gamma$ and Y is chi-square distributed with $deg = 2(k-s_m+1)$ degrees of freedom. In testing, it is determined that for coverages approaching one, the value of c_m is extremely close if not equal to m. To ensure that the lower limit for the confidence interval is met or exceeded, the value of m must be extremely large. To reduce the required sample size and to meet the lower confidence interval requirement, fault expansion (Smith et al, 1995; Smith et al, 1997) is used

In sampling using fault expansion, there are two cases of interest: the infinite and the finite fault population. For the infinite population, it is shown in (Smith et al, 1995) that the best estimate for fault coverage occurs when all fault classes are of equal size. The resulting lower one-sided confidence interval for this coverage estimate is identical to that found in Equation (A.55). Since there is no appreciable variance reduction, fault expansion is not recommended. However, fault expansion is very helpful for the finite population case.

Assuming that the fault population is finite the exact coverage factor is given by Equation (A.13). Since it is impractical to inject all N faults, the value for D must be estimated. It is proven that the one-sided confidence interval for the lower limit on the estimate of D, that is D_l, using the binomial distribution is

$$P(C_{fi} \geq c_{fi} | DF_l) = \frac{\Gamma\left(dc_l \frac{N}{b} + 1\right)\Gamma\left(\frac{N}{b} - fi + 1\right)}{\Gamma\left(dc_l \frac{N}{b} - fi + 1\right)\left(\Gamma\frac{N}{b} + 1\right)} = 1-\gamma, \tag{A.57}$$

where b is the equivalence class size, N is the finite fault population size, fi is the total number of injections, dc_l is the lower limit on coverage and γ is the confidence coefficient.

From Equation (A.57), it can be determined that dc_l is a function of b/N and fi. In analyzing this equation, it is proven that the size of the population classes greatly affects the analysis. If the equivalence classes are significantly large, the impact of fault expansion is maximized. This result implies that unlike in the infinite population case, the size of the fault classes need not be equivalent; instead, it is desirable that the equivalent fault classes for covered faults should be considerably larger than those for uncovered faults.

1.2.3.2 Smith et al Empirical Model (Smith et al, 1995; Smith et al, 1997)

If all sample measurements are covered, then the variance found in Equation (A.18) is zero. Hence, no confidence interval can be calculated. To overcome this limitation, a more conservative variance estimate is needed. By converting one covered fault injection experiment into an uncovered experiment, the variance estimate is always non-zero and more conservative in nature; that is, the calculated variance will exceed the actual value. This modified variance estimate of y is

$$\hat{\sigma}'^2_{y_n} = \frac{n-1}{n^2} \tag{A.58}$$

and the resulting confidence interval for the lower bound is

$$\hat{C}_{y_{n_{low}}} = 1 - \zeta_n \sqrt{\frac{\hat{\sigma}'^2_{y_n}}{n}} = 1 - \zeta_n \sqrt{\frac{n-1}{n}}. \tag{A.59}$$

where ζ_u is selected depending upon the desired confidence level. This lower bound is consistent with other analyses of the all covered case (Miller et al, 1992).

The point estimate mean for $\hat{C}_{z_n}$ is

$$\hat{C}_{z_n} = \frac{\sum_{i=1}^{n} z_i |E_i|}{\sum_{i=1}^{n} |E_i|} = \frac{1}{m} \sum_{i=1}^{n} z_i |E_i|, \tag{A.60}$$

where

$$m = \sum_{i=1}^{n} |E_i| \tag{A.61}$$

Assuming that the sample size is sufficiently large, then the CLT can be applied, and the resulting point estimate for

$\hat{C}_{z_n}$ is

$$\hat{C}_{z_n} = \sum_{i=1}^{n} z_i p_i , \tag{A.62}$$

where $p_i = |E_i|/m$, $\sum_{i=1}^{n} p_i = 1$ and p_i is the probability that a fault lies in class i. The resulting variance is

$$\hat{\sigma}^2_{z_n} = \sum_{i=1}^{n} (z_i - \hat{C}_{z_n})^2 p_i . \tag{A.63}$$

Similarly, the variance reduction is derived assuming there exists one uncovered fault set.

Assuming that the first class E_1 is uncovered, the uncovered probability p_1 is

$$p_1 = \frac{|E_1|}{m} . \tag{A.64}$$

and the point estimate as described in Equation (A.62) becomes

$$\hat{C}_{z_n} = z_1 p_1 + \sum_{i=2}^{n} z_i p_i = \frac{m - |E_1|}{m} , \tag{A.65}$$

and the variance becomes

$$\hat{\sigma}^2_{z_n} = \frac{|E_1|(m - |E_1|)}{m^2} . \tag{A.66}$$

By assuming that the covered and the uncovered probabilities are of the following form

$$\begin{aligned} p_c &= \frac{1}{m} \sum_{i \in \text{ covered faults}} |E_i| \\ p_{\bar{c}} &= \frac{1}{m} \sum_{i \in \text{ uncovered faults}} |E_i| = \frac{|E_{\bar{c}}|}{m} \end{aligned} , \tag{A.67}$$

Equation (A.66) can be rewritten as

$$\hat{\sigma}^2_{z_n} = \frac{|E_{\bar{d}}|(m - |E_{\bar{d}}|)}{m^2} . \tag{A.68}$$

As long as $|E_{\bar{c}}| < (um)/n$, where $\bar{d}$ is the number of uncovered fault injection experiments, the variance is reduced. If the average uncovered fault class size is smaller than the average set size, then fault expansion provides variance reduction.

As is true for the random sampling case, a conservative estimation must be made for the all covered case. It is again conservatively estimated that one of the covered fault injection experiments is assumed to be uncovered to prevent a zero variance. To minimize the increase in variance, $|E_1|$ in Equation (A.68) is set equal to one. The resulting variance estimate is

$$\hat{\sigma}'^2_{z_n} = \frac{m-1}{m^2} \forall \left\{ \left(\sum_{i=1}^{n} z_i \right) = n \right\} \tag{A.69}$$

The variance reduction ratio for $\hat{C}_{z_n}$ as it relates to the original $\hat{C}_{y_n}$ point estimate is

$$v_r = \frac{\hat{\sigma}^2_{z_n}}{\hat{\sigma}^2_{y_n}} = \frac{\dfrac{|E_c|(m-|E_c|)}{m^2}}{\dfrac{\left(\sum_{l=1}^{n} y_l\right)\left(n-\sum_{l=1}^{n} y_l\right)}{n^2}} = \frac{n^2(|E_c|(m-|E_c|))}{m^2\left(\sum_{l=1}^{n} y_l\right)\left(n-\sum_{l=1}^{n} y_l\right)}. \tag{A.70}$$

which is a measure of statistical improvement that results from utilizing fault expansion. For the all covered case, the resulting variance reduction ratio is

$$v'_r = \frac{\hat{\sigma}'^2_{z_n}}{\hat{\sigma}'^2_{y_n}} = \frac{(m-1)n^2}{(n-1)m^2} \cong \frac{n}{m} \text{ if}(m \gg 1)\text{and}(n \gg 1). \tag{A.71}$$

1.2.4 Constantinescu Empirical Model (Constantinescu, December 1995)

In this model, multi-stage, stratified and combined multi-stage and stratified random sampling (Cochran, 1977) are used. Multi-stage sampling allows the use of a multidimensional event space. The number of dimensions is equivalent to the number of factors that affect coverage. This event space consists of the cross products of all possible fault locations, types, times of occurrence, durations and system workloads. Due to practical considerations, a 3-D space is used consisting of the cross products of three 1-D subspaces. These subspaces include fault location, fault occurrence times and system input values. The population is divided into consecutive *subunits*, and random sampling is performed on these newly created subunits. The coverage model found in Equation (A.16) is re-written to accommodate a multi-dimensional sample space as

$$C = \frac{\sum_{i_1=1}^{|E_1|} \cdots \sum_{i_n=1}^{|E_n|} y_{(i_1, i_2, \ldots, i_n)}}{\prod_{i=1}^{n} |E_i|}, \tag{A.72}$$

Obviously, the number of fault locations is finite by nature. Faults can occur at any time, but time can be subdivided into a finite number of small intervals. The input space, however, is not as easily defined. As system complexity increases, the number of input values becomes extremely large. To overcome this problem, stratification is used to manage the vast input space by subdividing it into smaller, more manageable subspaces called *strata*. The sum of the strata equals the original population. Each stratum is sampled *s*-independently. By combining both multi-stage and stratified sampling techniques, the effectiveness of sampling increases (Cochran, 1977; Wadsworth, 1990).

1.2.4.1 Multi-Stage Sampling for a 3-D Space

Multi-stage sampling requires random selection of members from the original population followed by consecutive random sampling from the subunits. The three sampling stages for simulated fault injection and for physical fault injection performed in 3-D space are found in Table H.2. Regardless of the sampling order that is used,

Table H.2: Fault Injection Sampling Stages

	Simulated Fault Injection	***Physical Fault Injection***
Stage 1	Select several input values at random from the 1-D input space	Randomly select the locations for fault injection for all previously selected inputs and injection times
Stage 2	Select several fault injection times at random from the 1-D fault occurrence time space for each input	Select several input values at random from the 1-D input space
Stage 3	Randomly select the locations for fault injection for all previously selected inputs and injection times	Select several fault injection times at random from the 1-D fault occurrence time space for each input

the formulae for coverage and general multi-stage sampling are applicable. Unbiased estimates for the mean and variance (Constantinescu, January 1995) are

$$\hat{C} = \frac{\sum_{i_1=1}^{|w_1|} \cdots \sum_{i_n=1}^{|w_n|} y_{(i_1, i_2, \ldots, i_n)}}{\prod_{j=1}^{n} |w_j|} \tag{A.73}$$

and

$$Var(\hat{C}) = \sum_{k=1}^{n} \left\{ \left(\frac{\left[\prod_{k'=1}^{k-1} \left(1 - \frac{|w_{k'}|}{|E_{k'}|}\right) \right] \frac{|w_k|}{|E_k|}}{\prod_{k'=1}^{k} |w_{k'}|} \right) \left(\frac{\sum_{i_1=1}^{|w_1|} \cdots \sum_{i_n=1}^{|w_n|} \left(y_{(i_1, i_2, \ldots, i_n)} - \sum_{i_n=1}^{|w_n|} y_{(i_1, i_2, \ldots, i_n)} \right)^2}{\left[\prod_{i=1}^{n-1} |w_i| \right] (|w_n| - 1)} \right) \right\}, \tag{A.74}$$

where $|w_k|$ is the sample size and $|E_k|$ is the population size at stage k for n sampling stages. The confidence level is as defined in Equation (A.19) and the indicator function is modeled in terms of a multi-dimensional space. Theorem proofs for these estimates can be found in (Constantinescu, 1994).

1.2.4.2 Stratified Sampling

In many situations, the amount of system input data needed is too large to explicitly input all data combinations. By using stratification to divide the input space into strata, the maximum number of possible inputs is greatly reduced. Typically, the maximum number of inputs grouped in a stratum is determined by the largest random number that can be practically generated, and by the type of inputs, binary or analog. For the 3-D event space for coverage, the original event space, E, is subdivided into several smaller subspaces.

Assuming the point estimate is Gaussian, the unit population, E, be subdivided into m subpopulations. If independent random sampling is performed in every stratum, then the unbiased estimate of the mean is

$$\hat{C}_{ST} = \sum_{i=1}^{m} f_{E_i}(st)\hat{C}_i \quad , \qquad p_{E_i}(st) \equiv \frac{E_{ST_i}}{E} \tag{A.75}$$

where ST stands for stratified. An unbiased variance estimate is

$$Var(\hat{C}_{ST}) = \sum_{i=1}^{m} p_{E_i}^{\,2}(st) \cdot Var(\hat{C}_i) \quad , \tag{A.76}$$

Theorem proofs for these estimates can be found in (Constantinescu, 1994).

1.2.4.3 Combined Multi-Stage and Stratified Sampling

Coverage estimator equations for combined multi-stage and stratified sampling are obtained from the simpler expressions found in theorems 1-6 and corollary 1 in (Constantinescu, December 1995). For example, the appropriate sampling order for simulated fault injection is stratification followed by multi-stage sampling. The weight of stratum j is

$$f_{E_j}(st) = \frac{\prod_{i=1}^{n} E_{ST_{ji}}}{\sum_{j=1}^{m}\prod_{i=1}^{n} E_{ST_{ji}}} \quad \text{for } i = 1, 2, \ldots, n. \tag{A.77}$$

The resulting confidence limits on the coverage can be found by substituting Equation (A.76) using the stratum weights derived in Equation (A.77) into Equation (A.19).

1.2.5 Limitations of Empirical Models for Coverage Estimation

Empirical models approach coverage estimation by implementing various VRTs. In the Powell et al method, the variance reduction is achieved using stratified sampling and two-stage sampling. However, the conditional probabilities that are required for this estimate need to be measured. These measurements can be extremely difficult, if not impossible, and the problems associated with estimating these probabilities are demonstrated by the wide range of coverage estimations shown in (Powell et al, 1993; Powell et al, 1995). In two-stage sampling, the results demonstrated that better coverage estimation can be achieved than for stratified sampling. However, for both of these methods, it is concluded that the coverage estimate is system dependent.

In the Wang et al model, fault expansion is used to provide variance reduction. In this model it is assumed that the binomial representation for coverage approximates a Poisson distribution, which is then used to calculate a single sided confidence level. In the model presented by Smith et al, they applied the CLT to determine the confidence interval associated with the coverage estimation, because it is assumed that the sample size is sufficiently large (Papoulis, 1991). Similarly in the Constantinescu method, his point estimates for the various sampling techniques discussed assumes that the point estimates are Gaussian. As a result, these assumptions limit the broad applicability of these methods. If the underlying distribution for the point estimate is not Gaussian or Poisson, then none of these approximations are appropriate.

The coverage estimation model by Cukier et al is a refinement of the Powell et al model using uncovered fault information to develop a Bayesian estimate. This model requires the conditional probabilities from the Powell et al model, which are difficult if not impossible to measure, for calculating both the moments and the Pearson parameters in the Bayesian approach. Additionally, it is assumed that the prior distribution for the coverage estimate is beta with parameters one. If however the prior distribution is not beta or the beta parameters differ, then the calculation of the posterior distribution is in error. Finally, this model also shows some dependence on the system being modeled; that is, the ability to derive a coverage estimate can be system dependent.

1.3 Physical Models

Physical models represent the actual system and involve the development of prototypes realized in either software or hardware. Additionally, these models can be realized at multiple levels of abstraction such as the transistor, gate, circuit or system level, allowing for hierarchical modeling. Ideally, all parameters can be measured at the these various levels, but there is no accepted way to keep the faults modeled consistently throughout the various levels of hierarchy. Nevertheless, the parameter estimates' measure should be consistent and should improve as the level of modeling descends to the lowest level, which is the transistor level.

The feasibility of constructing and testing hardware prototypes depends upon both the time available and the

cost of production. If only time is a concern, then a sample system can be constructed and tested in a harsh (that is, failure rate accelerated) environment. If cost is a concern, then software based prototypes can be built. The obvious problem here is the time required to simulate the models.

These software models can be tested just like an actual sample system using fault injection. For both of these techniques, an unbiased subset of the overall fault space must be used for fault injection. As discussed in Section 1.1.3: Limitations of Axiomatic Coverage Models, the selection of a fault list subset that is of sufficient size to provide an accurate fault coverage estimate requires an empirical model. If this analysis is not performed, then the selection of the list of faults to be tested is highly subjective and it may not provide a statistically meaningful result.

1.4 Conclusions

Fault coverage can be examined in two distinct ways: (1) fault coverage modeling and (2) parameter estimation. Fault coverage modeling consists of creating an axiomatic model that represents faulty component behavior within a given system. One drawback of this approach is that the size and the complexity of a system model would dramatically increase because of the state space required to represent all possible fault scenarios. Additionally, the parameters that these axiomatic models require, such as the transition rates between the various states, are not known *a priori*. As a result, these values are approximated from other analytical models or expert opinion, and the resulting fault coverage is simply an approximation with undefined confidence intervals. To overcome these problems, empirical and physical models are used.

Empirical models are statistical models that use data collected from physical models. Using fault injection with physical models, data pertaining to the various dependability parameters, including fault coverage, is collected and parameter estimation can be made from the empirical models. Dependability testing must be incorporated during the design cycle, and via fault injection, the various dependability parameters, including fault coverage, can be estimated for a given confidence interval from the empirical models using VRTs.

Fault coverage estimation is typically achieved via point estimation and Bayesian techniques. Point estimation, assuming that a large enough data set exists, can use the CLT, which implies a Gaussian distribution, from which a two-sided confidence interval can be calculated. Special attention is given to the lower bound because it is the most pessimistic estimate. Additionally, a coverage point estimate can be obtained if coverage is assumed to be binomially distributed. This distribution can be estimated by a Poisson distribution from which a single-sided confidence interval can be extracted. In all of these methods, various sampling schemes are implemented to provide variance reduction for the coverage estimates.

The limitation of these approaches is that the empirical models base their parameter estimates upon an *a priori* selection for the distribution of the point estimate, and in the case of Bayesian techniques, the coverage

estimate is based upon an *a priori* selection of the prior distribution. As discussed in Section 1.1.9: Limitations of Empirical Models for Coverage Estimation, this *a priori* selection of the various distributions limits the applicability of these existing empirical models. Additionally, some of these models are system dependent, which further limits their applicability.

References

1. Abramowitz, M. and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*, Dover Publications, New York, 1972.

2. Aitchison, J. and I. R. Dunsmore, *Statistical Prediction Analysis*, Cambridge University Press, Cambridge, England, 1975.

3. Arlat, J., M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins and D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications," *IEEE Transaction on Software Engineering*, vol. 16, no. 2, pp. 166-181, February, 1990.

4. Arlat, J., A. Costes, Y. Crouzet, J.-C. Laprie, and D. Powell, "Fault Injection and Dependability Evaluation of Fault Tolerant Systems," *IEEE Transactions on Computers*, vol. 42, no. 8, pp. 913-923, August 1993.

5. Arlat, J., Y. Crouzet and J.-C. Laprie, "Fault Injection for the Experimental Validation of Fault Tolerance," *Proceedings of the Esprit Conference*, pp. 791-805, November, 1991.

6. Arnold, T. F., "The Concept of Coverage and Its Effect on the Reliability Model of Repairable Systems," *IEEE Transaction on Computers*, vol. 22, March, 1973.

7. Bateman, K. A. and E. R. Cortes, "Availability Modeling of FDDI Networks," *Proceedings of Annual Reliability and Maintainability Symposium*, pp. 389-395, 1989.

8. Berthismieu, B. and M. Diaz, "Modeling and Verification of Time Dependent Systems using Time Petri Nets", *IEEE Transactions on Software Engineering*, vol. 17, no. 3, pp. 259-273, March 1991.

9. Billington, R., and R. Allen, *Reliability Evaluation of Engineering Systems: Concepts and Techniques*, Plenum Press, pp. 352-359, 1992.

10. Bouricius, W. G., W. C. Carter and P. R. Schneider, "Reliability Modeling Techniques for Self-repairing Computer Systems," *Proceedings of the 24th ACM Annual Conference*, pp. 295-309, March, 1969.

11. Carter, W. C. and J. Abraham, "Design and Evaluation Tools for Fault Tolerant Systems," *Proceedings of the AIAA Computers in Aerospace Conference*, 1987.

12. Cassandras, C. G., *Discrete Event Systems: Modeling and Performance Analysis*, Aksen Associates Inc. and Richard D. Irwin Inc., Boston, Massachusetts, 1993.

13. Chen, L. and A. Avizienis, "N-version programming: A Fault Tolerant Approach to Reliability of Software Operation," *IEEE Transactions on Computers*, vol. C-20, no. 11, pp. 1322-1331, November 1971.

14. Choi, C. Y., *Dependable System Hardware/Software Codesign using Data Flow Models*, Doctoral Dissertation, University of Virginia, 1997.

15. Cochran, W. G., *Sampling Techniques*, John Wiley and Sons, New York, 1977.

16. Colombo, A. G., and A. Z. Keller, editors, *Reliability Modeling and Applications: Proceedings of the ISPRA Course, held at Joint Research Centre, Ispra, Italy*, D. Reidel Publishing Company, pp. 1-32, 1987.

17. Conn, R. B., P.M. Merryman and K.L. Whitelaw, "CAST - A Complimentary Analytic-Simulative Technique for Modeling Fault Tolerant Computing Systems," *Proceedings of the AIAA Computers and Aerospace Conference*, November 1977.

18. Constantinescu, C.,"Statistical Techniques for Estimating Fault Coverage Probabilities," Technical Report, Duke University, 1994.

19. Constantinescu, C., "Estimation of the Coverage Probabilities by 3-Stage Sampling," *Proceedings of the Annual Reliability Symposium*, pp. 132-136, January, 1995.

20. Constantinescu, C., "Using Multi-Stage and Stratified Sampling for Inferring Fault Coverage Probabilities," *IEEE Transactions on Reliability*, vol. 44, no. 4, pp. 632-639, December 1995.

21. Cukier, M., "Estimation of the Coverage of Fault Tolerant Systems," Doctoral Dissertation, National

Polytechnic Institute of Toulouse, France, LAAS-Report no. 96290, July 1996.

22. Cukier, M., J. Arlat and D. Powell, "Frequentist and Bayesian Estimations with Stratified Fault Injection," *Proceedings of the DCCA-6*, pp. 38-57, March 1997.

23. Daehn, W., "Fault Injection Using Small Fault Samples," *Journal of Electronic Testing: Theory and Applications*, vol. 2, pp. 191-203, 1991.

24. Davis, C., "Common Mode Failures in Information Systems," *SciTech Journal*, July-August 1995.

25. Dhillon, B. S., *Reliability Engineering in Systems Design and Operation*, Van Nostrand Reinhold Company, pp. 70-71, 1983.

26. Dhillon, B. S., and O. C. Anude, O. C., "Common-cause Failure Analysis of a Redundant System with Repairable Units," *International Journal of Systems Science*, vol. 25, no. 3, pp. 527-540, March 1994.

27. Dhillon, B. S., and H. C. Viswanath, "Reliability Analysis of a Non-identical Parallel System with Common-cause Failures," *Microelectronics and Reliability*, vol. 31, no. 2-3, pp. 429-41, 1991.

28. DOE-STD-3009-94

29. Dugan, J. B., "Correlated Hardware Failures in Redundant Systems," *Dependable Computing for Critical Applications 2* (J. F. Meyer and R. D. Schlichting eds.), Springer Verlag, New York, pp. 157-174, 1992.

30. Dugan, J. B., S. Doyle, F. A. Patterson-Hine, "Simple Models of Hardware and Software Fault Tolerance, *Proceedings of Annual Reliability and Maintainability Symposium*, pp. 124-129, 1994.

31. Dugan, J. B. and K. S. Trivedi, "Coverage Modeling for Dependability Analysis of Fault-Tolerant Systems," *IEEE Transaction on Computers*, vol. 38, no. 6, pp. 775-787, June, 1989.

32. Dugan, J. B., S. J. Bavuso and M. A. Boyd, "Dynamic Fault Tree Models for Fault Tolerant Computer Systems," *IEEE Transaction on Reliability*, vol. 41, no. 3, pp. 363-376, June, 1992.

33. Gangloff, W. C., "Common-mode Failure Analysis," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-94, no. 1, pp. 27-30, January - February 1975.

34. Garrett, C., M. Yau, S. Guarro and G. Apostolakis, "Assessing the Dependability of Embedded Software Systems using the Dynamic Flowgraph Methodology," *Dependable Computing for Critical Applications 4* (J. F. Meyer and R. D. Schlichting eds.), Springer Verlag, New York, pp. 161-184, 1995.

35. Garrett, C., M. Yau, S. Guarro and G. Apostolakis, "The Dynamic Flowgraph Methodology for Assessing the Dependability of Embedded Software Systems," *IEEE Transaction on Systems, Man and Cybernetics*, vol 25, no. 5, pp. 824-840, May 1995.

36. Goddard, P. L., "A Combined Analysis Approach to Assessing Requirements for Safety Critical Real-time Systems", *Proceedings of Annual Reliability and Maintainability Symposium*, pp. 110-115, 1996.

37. Guarro, S. and D. Okrent, "The Logic Flowgraph: A New Approach to Process Failure Modeling and Diagnosis for Disturbance Analysis Applications," *Nuclear Technology*, vol. 67, 00. 11-22, pp. 348-359.

38. Gulati, R. and J. B. Dugan, "A Modular Approach for Analyzing Static and Dynamic Fault Trees," *Proceedings of Annual Reliability and Maintainability Symposium*, pp. 57-63, 1997.

39. Hagen, E. W., "Common-mode/Common-cause Failure: A Review," *Nuclear Engineering and Design*, vol. 59, pp. 423-431, 1980.

40. Hammersley, J. M. and D.C. Handscomb, *Monte Carlo Methods*, Methuen and Company Ltd., London, 1964.

41. Harpel, B. M., J. B. Dugan, I. D. Walker, J. R. Cavallaro. "Analysis of Robots for Hazardous Environments", *Proceedings of Annual Reliability and Maintainability Symposium*, pp. 111-116, 1997.

42. Henley, E. J., and K. Hiromitsu, *Reliability Engineering and Risk Assessment*, Prentice-Hall, pp. 120-127, 385-390, 1981.

43. Hoertner, H., "Problems of Failure with Respect to Systems Reliability," *Nuclear Engineering and Design*, vol. 71, no. 3, pp. 387-389, 1982.

44. Jallote, P., *Fault Tolerance in Distributed Systems*, PTR Prentice-Hall, Englewood Cliffs, New Jersey, 1994.

45. Jin, Q., Y. Sugasawa, "Representation and Analysis of Behavior for Multiprocessor Systems by using Stochastic Petri Nets", *Mathematical and Computer Modeling*, vol. 22, no. 10-12, pp 109-116, 1995

46. Johnson, B. W., *Design and Analysis of Fault Tolerant Digital Systems*, Addison Wesley, New York, 1989.

47. Johnson, N. L. and S. Kotz, *Distributions in Statistics - Discrete Distributions*, John Wiley and Sons, New York, 1969.

48. Johnson, N. L. and S. Kotz, *Distributions in Statistics - Continuous Univariate Distributions -1*, John Wiley and Sons, New York, 1970.

49. Kaplan, S. and B. J. Garrick, "On the Quantitative Definition of Risk," *Risk Analysis*, vol. 1, no. 1, pp. 11-27, March 1981.

50. Karlin, S. and H. M. Taylor, *A First Course in Stochastic Processes*, Second Edition, Academic Press, New York, 1975.

51. Kaufman, L. M., *Dependability Analysis for Ultra-Dependable Systems using Statistics of the Extremes*, Doctoral Dissertation, University of Virginia, 1997.

52. Kaufman, Lori M. and Barry W. Johnson, "The Importance of Fault Detection Coverage in Safety Critical Systems," *Proceedings of the Twenty-Sixth Water Reactor Safety Information Meeting, Volume 2*, U.S. Nuclear Regulatory Commission, NUREG/CP-0166, October 1998, pp. 5-28.

53. Kaufman, L. M., R. Manian, K. Vemuri, and J. B. Dugan, "System-Reliability Analysis of an Embedded Hardware/Software System using Fault Trees," *Proceedings of the Reliability and Maintainability Symposium*, January 1999, pp. 135-141.

54. Knight, J. C. and N. G. Leveson, "An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming," *IEEE Transactions on Software Engineering*, vol. SE-12, no. 1, pp. 96-109, January 1986.

55. Khobare, S. K, S. V. Shrikhande, Umesh Chandra and G. Govindraj, "Reliability Analysis of Microcomputer Circuit Modules and Computer-based Control Systems Important to Safety of Nuclear Power Plants," *Reliability Engineering and System Safety*, vol. 59,no. 2, pp253-258, 1998.

56. Khodabandehloo, K., "Analyses of Robot Systems using Fault and Event Trees: Case Studies," *Reliability Engineering and System Safety*, vol. 53, no. 3, pp 247-264, September 1996.

57. Kohavi, Z., *Switching and Finite Automata Theory*, McGraw-Hill, New York, 1978.

58. Lala, J. H., and R. E. Harper, "Architectural Principles for Safety-critical Real-time Applications," *Proceedings of the IEEE*, vol. 82, no. 1, pp.25-40, January 1994.

59. Lala, J. H., R. E. Harper, and L. S. Alger, "A Design Approach for Ultrareliable Real-time Systems," *IEEE Computer*, pp. 12-22, May 1991.

60. Laprie, J. C., "Dependable Computing and Fault Tolerance: Concepts and Terminology," *Proceedings of the 15th Annual International Symposium on Fault Tolerant Computing*, pp. 2-11 June 19-21, 1985.

61. Laprie, J. C., *Dependability: Basic Concepts and Terminology - In English, French, German and Japanese*, Springer-Verlag, Vienna, 1992.

62. Law, A. M. and W. D. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, Inc., New York, 1991.

63. Leveson, N. G. and J. L. Stolzy, "Safety Analysis using Petri Nets", *IEEE Transactions on Software Engineering*, vol. SE- 13, no. 3, pp. 386-397, March 1987.

64. Makam, S. V. and A. Avizienis, "ARIES 81: A Reliability and Lifecycle Evaluation Tool for Fault Tolerant

Systems," *Proceedings of the 12th Annual Fault Tolerant Computing Symposium*, 1982.

65. Marsan, M. A., G. Conte and G. Balbo, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems", *ACM Transactions on Computer Systems*, vol. 2, no. 2, pp. 83-122, May 1984.

66. McCormick, N. J., *Reliability and Risk Analysis: Methods and Nuclear Power Applications*, Academic Press, Inc., San Diego, California, 1981.

67. McGough, J., "Effects of Near-Coincident Faults in Multiprocessor Systems," *Proceedings of the 5th IEEE/ AIAA Digital Avionics Systems Conference*, pp. 16.6.1-16.6.7, November 1983.

68. Miller, K. W., L. J. Morell, R. E. Noonan, S. K. Park, D. M. Nicol, B. W. Murrill and J. M. Voas, "Estimating the Probability of Failure When Testing Reveals No Failures," *IEEE Trans. Software Engineering*, vol. 18, no. 1, pp. 33-43, January 1992.

69. Min, Y., Y. Zhou, Z. Li, C. Ye, "A Fail-safe Microprocessor-based System for Interlocking on Railways," *Proceedings of Annual Reliability and Maintainability Symposium*, pp. 415-420, 1994.

70. Modarres, M., What Every Engineer Should Know about Reliability and Risk Analysis, Marcel Dekker, New York, 1993.

71. Murata, T., "Petri Nets: Properties, Analysis and Applications," Proceedings of the IEEE, vol. 77, pp. 541-580, 1989.

72. Mojdehbakhsh, R., S. Subramanian, R. Vishnuvajjala, W. Tsai and L. Elliott, "A Process for Software Requirements Safety Analysis," Proceedings of the International Symposium on Software Reliability Engineering, pp. 45-54, 1994.

73. Nahman, J. M., "Minimal Paths and Cutsets of Networks Exposed to Common-cause Failures," IEEE Transactions on Reliability, vol 41, no 1, pp. 76-80, 84, March 1992.

74. NASA-GB-1740.13-96, *NASA Guidelines for Safety Critical Software, Analysis and Development*, pp. 85-86, 1996.

75. NASA Contractor Report 189632, Volume II, "Advanced Information Processing System: The Army Fault Tolerant Architecture Conceptual Study," pp. 8.1-8.36, July 1992.

76. National Research Council, *Digital Instrumentation and Control Systems in Nuclear Power Plants: Safety and Reliability Issues*, National Academy Press, pp. 43-51, 1997.

77. NUREG/CR- 6303, *Method for Performing Diversity and Defense-in-depth Analyses of Reactor Protection Systems*, prepared by G. G. Preckshot, Lawrence Livermore National Laboratory, December 1994.

78. NUREG/CR-6465, *Development of Tools for Safety Analysis of Control Software in Advance Reactors*, prepared by S. Guarro, M. Yao and M. Motamed, ASCA, Inc., April 1996.

79. Nelson, V. P. and B. D. Carroll, "Fault-Tolerant Computing (A Tutorial)," presented at the *AIAA Fault Tolerant Computing Workshop*, November 8-10, 1982.

80. Papoulis, A., *Random Variables and Stochastic Processes*, McGraw-Hill, Inc., New York, 1991.

81. Paula, H. M., "Database Features that are Needed to Support CCF Analysis: An Analyst's Perspective," *Nuclear Safety*, vol. 31, no. 2, pp. 159-173, April - June 1990.

82. Peng, Z. and A. Torne, "A Petri Net Based Modeling and Synthesis Technique for Real-Time Systems", *Proceeding of the 5th Euromicro Workshop on Real-Time Systems*, 1993.

83. Peterson, J. L., *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

84. Powell, D., M. Cukier and J. Arlat, "On Stratified Sampling for High Coverage Estimations," *Proceedings of EDDC-2*, pp. 37-54, Oct. 1996.

85. Powell, D., E. Martins, J. Arlat and Y. Crouzet, "Estimators for Fault Tolerance Coverage Evaluation," *Proceedings of the 23rd Fault Tolerant Computing Symposium*, pp. 229-237, June 1993.

86. Powell, D., E. Martins, J. Arlat and Y. Crouzet, "Estimators for Fault Tolerance Coverage Evaluation," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 261-274, February 1995.

87. Presentation by Mark Richards, ARPA Technology Kickoff Meeting, RASSP Program, Oct. 21-22, 1993.

88. Rao, R., G. Swaminathan, B. W. Johnson, J. Aylor, "Synthesis of Reliability Models from Behavioral-Performance Models", *Proceedings of Annual Reliability and Maintainability Symposium*, pp. 292-297, 1994.

89. Rao, R., A. Raman, B. W. Johnson, "Reliability Analysis using the ADEPT-REST Interface", *Proceedings of Annual Reliability and Maintainability Symposium*, pp. 73-81, 1996.

90. Richards, M., ARPA Kickoff Meeting RASSP Program Presentation, Oct. 21-23, 1993, Arlington, VA.

91. Rosen, K. H., *Discrete Mathematics and Its Applications, 3rd Edition*, McGraw-Hill, New York, 1995.

92. Rosenberg, G., "Advances in Petri Nets 1990", *Lecture Notes in Computer Science*, vol. 483, Springer, New York, 1990.

93. Segall, Z., D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownacki, J. Barton, D. Rancey, A. Robinson and T. Lin, "FIAT - Fault Injection based Automatic Testing Environment," *Proceedings of the 18th International Symposium on Fault Tolerant Computing*, pp. 102-107, 1988.

94. Shedler, G. S. and S. Moriguchi, "Diagnosis of Computer Systems by Stochastic Petri Nets Part II (Theory)", *IEICE Transactions Fundamentals*, vol. E76 A, no. 4, pp 565-579, April 1993.

95. Shimeall, T. J., R. J. McGraw, Jr., and J. A. Gill, "Software Safety Analysis in Heterogeneous Multiprocessor Control Systems," *Proceedings of Annual Reliability and Maintainability Symposium*, pp. 290-294, 1991.

96. Shooman, M. L., *Probabilistic Reliability: An Engineering Approach*, McGraw-Hill, New York, 1968.

97. Siu, N. and A. Mosleh, "Treating Data Uncertainties in CCF Analysis," *Nuclear Technology*, vol. 84, no. 3, pp. 265-81, March 1989.

98. Smith, D. T., B. W. Johnson, J.A. Profeta and D.G. Bozzolo, "A Method to Determine Equivalent Fault Classes for Permanent and Transient Faults," *Proceedings of the Annual Reliability Symposium*, pp. 418-424, January 1995.

99. Smith, D. T., B. W. Johnson, N. Andrianos, and J.A. Profeta III, "A Variance Reduction Technique via Fault Expansion for Fault Coverage Estimation," *IEEE Transactions on Reliability*, vol 46, no. 3, pp. 366-374, September, 1997.

100. Stavrianidis, P., "Reliability and Uncertainty Analysis of Hardware Failures of a Programmable Electronic System," *Reliability Engineering and System Safety*, vol. 39, pp. 309-324, 1993.

101. Stiffler, J. J. and L. A. Bryant, *CARE III Phase III Report - Mathematical Description*, Contr. Rep. 3566, NASA, Nov. 1982.

102. Stuart, A. and J. K. Ord, *Distribution Theory*, vol. 1, Edward Arnold, London, 1987.

103. Trivedi, K. S., *Probability and Statistics with Reliability, Queueing and Computer Science Applications*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

104. Trivedi, K. S., J. B Dugan, R. Geist and M. Smotherman, "Modeling Imperfect Coverage in Fault-Tolerant Systems," *Proceedings of the 14th Fault Tolerant Computing Symposium*, pp. 77-82, June 1984.

105. United States Atomic Energy Commission, *An Assessment of Accident Risks in U.S. Commercial Nuclear Power Plants*, WASH-1400, 1975.

106. United States Department of Defense, *Military Standardization Handbook: Reliability Prediction of*

Electronic Equipment, MIL-HDBK-217, Revision F Notice 2, February 1995.

107. Voas, J. A. Ghosh, F. Charron, and L. Kassab, *Reducing Uncertainty about Common Mode Failures*, http:// www.rstcorp.com.

108. Wadsworth, H. M. (editor), *Handbook of Statistical Methods for Engineers and Scientists*, McGraw-Hill, New York, 1990.

109. Wang, W., K. S. Trivedi, B. V. Shah, and J. A. Profeta III, "The Impact of Fault Expansion on the Internal Estimate for Fault Detection Coverage," *Proceedings of the 24th Annual International Symposium on Fault Tolerant Computing*, pp.330-337, June 1994.

110. Watson, I. A., "Analysis of Dependent Events and Multiple Unavailabilities with Particular Reference to Common-cause Failures," *Nuclear Engineering and Design*, vol. 93, pp. 227-44, 1986.

111. Watson, I. A., and G. T. Edwards, "Common-mode Failures in Redundant Systems," *Nuclear Technology*, vol. 46, no. 2, pp. 183-91, November - December 1979.

112. Watson, I. A., and A. M. Smith, "Common-cause Failures - A Dilemma in Perspective," *Proceedings of Annual Reliability and Maintainability Symposium*, pp. 332-339, 1980.

113. Yau, M., S. Guarro, R. Mulvihill, T. Milici, "Application of Dynamic Flowgraph Techniques for Safety Analysis and Testing of Space Systems Software," ASCA Inc., Final Report prepared for NASA, Lewis Research Center, 1995.

114. Yau, M., S. Guarro, G. Apostolakis, "Demonstration of the Dynamic Flowgraph Methodology using the Titan II Space Launch Vehicle Digital Flight Control System," Reliability Engineering and System Safety, vol. 49, no. 3, pp. 335-353, 1995.

GLOSSARY

Algorithmic level model provides the highest level of system information. This model level explains the desired system behavior and demonstrates the interaction of various high-level components. When little information is available about a given component, then this level of modeling may be all that is available.

Architectural level model provides the actual design of the embedded system. At this design level, component selection is made and the functional interaction among the various components is defined.

Axiomatic models are analytical models that are used to model structure and the dependability and/or performance behavior of a system.

Coverage is the conditional probability that given the occurrence of a fault in a system it is detected and the system recovers.

Defense in depth approaches safety by implementing multiple protective echelons in a concentric nature within the design: that is, system safety is achieved by separating safety features such that the failure of one echelon does not disable the safety systems contained within other echelons.

Dependability is a measure of the quality of service that given system provides.

Diversity provides several ways of detecting and responding to a significant event by using different logic, algorithms, or means of actuation.

Embedded system means the integration of hardware and software components to produce a functional system.

Empirical model are statistical models that model complex and detailed descriptions of a system's parameter(s) using data collected from physical models.

Error is any deviation from correctness or accuracy. It is the manifestation of a fault.

Failure means the nonperformance of some expected action. It is the manifestation of an error.

Fault is a physical defect, imperfection or flaw that occurs within some hardware or software component.

Hybrid modeling consists of using simulation and algorithmic models to determine various parameters required by dependability models.

Logic level model provides the actual implementation of the system.

Physical models are prototypes that actually implement the hardware and/or the software of an actual system.

Redundancy is the addition of information, resources or time beyond what is needed for normal system operation.

Reliability is the probability that a system is an operational state for a specified time interval.

Robustness combines elements of redundancy, diversity and defense in depth to achieve safety and reliability.

Safety means that given the occurrence of a fault, the system recognizes the fault occurrence and can continue correct operation or it can shut down in a safe manner.

NRC FORM 335 (2-89) NRCM 1102, 3201, 3202

U.S. NUCLEAR REGULATORY COMMISSION

BIBLIOGRAPHIC DATA SHEET

(See instructions on the reverse)

1. REPORT NUMBER (Assigned by NRC, Add Vol., Supp., Rev., and Addendum Numbers, If any.)

NUREG/GR-0020

2. TITLE AND SUBTITLE

Embedded Digital System Reliability and Safety Analyses

3. DATE REPORT PUBLISHED

MONTH: February YEAR: 2001

4. FIN OR GRANT NUMBER

K6907

5. AUTHOR(S)

L.M. Kaufman, B.W. Johnson

6. TYPE OF REPORT

Technical

7. PERIOD COVERED *(Inclusive Dates)*

8. PERFORMING ORGANIZATION - NAME AND ADDRESS *(If NRC, provide Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address; if contractor, provide name and mailing address.)*

University of Virginia
Department of Electrical Engineering
Center for Safety-Critical Systems - Thornton Hall
Charlottesville, VA 22904

9. SPONSORING ORGANIZATION - NAME AND ADDRESS *(If NRC, type "Same as above"; if contractor, provide NRC Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address.)*

Division of Engineering Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

10. SUPPLEMENTARY NOTES

J.A. Calvert, NRC Project Manager

11. ABSTRACT *(200 words or less)*

The migration from analog to digital systems in the instrumentation and control (I &C) within a nuclear power plant has increased the complexity of the instrumentation. The I & C systems being developed are computer-based consisting of embedded digital hardware and software components. These embedded systems perform many varying and highly complex functions that are integral to the safety-critical requirements of a nuclear power plant. The need to understand the effects of various failure modes, including common cause failures and common mode failures, in these systems is becoming increasingly important because the failure of an I & C system could leak to risk significant events. In order to understand the effects of common cause and common mode failures on a system, a survey of existing definitions and applications of these definitions as they apply to digital embedded systems was performed. From this survey, it was found that the definitions and analysis treated the hardware and the software as independent entities. However when embedded digital systems are in actual operation, there is tight integration of the hardware and software components; that is, the function realized by such a system cannot be partitioned between hardware and software but must be analyzed in a unified manner. In addition to addressing the limitations of the existing common cause and common mode failure definitions, a detailed assessment of existing modeling techniques to assess their effects is also presented.

12. KEY WORDS/DESCRIPTORS *(List words or phrases that will assist researchers in locating the report.)*

instrumentation and control (I & C)
embedded digital systems
hardware components
software components
failure modes

13. AVAILABILITY STATEMENT

unlimited

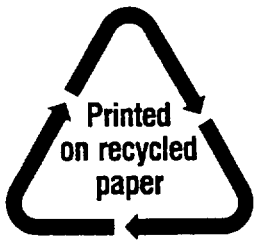
14. SECURITY CLASSIFICATION

(This Page) unclassified

(This Report) unclassified

15. NUMBER OF PAGES

16. PRICE



Federal Recycling Program