5.0 HARDWARE/SOFTWARE SYSTEM RELIABILITY MODELING

Reliability modeling of combined hardware and software systems is in many ways analogous to reliability modeling of purely hardware systems. Reliability block diagrams of system elements are developed and employed. Individual hardware platforms and the software assigned to those platforms are independent of other hardware/software platforms.

Reliability block diagrams that accurately portray the interrelationship between the hardware platforms and the software executing on the platforms are developed and used in estimating reliability metrics. For complex structures, state diagrams are developed to accurately portray the unique interrelationships of the structure being modeled.

This section provides the techniques applicable to the reliability modeling of combined hardware and software systems. For series systems, the process is straightforward. For complex, redundant, repairable systems, an abbreviated overview of the system modeling process is provided. Because these complex models are unique to the system being modeled, detailed procedures for developing these complex models cannot be provided.

The overview provided is used to help the analyst identify those system properties which are unique to redundant combined hardware and software systems. The majority of this section is dedicated to describing the development of software failure rates that are a composite of the multiple processes that may be executing during any time period.

5.1 Basic Reliability Model.

A basic reliability model for a hardware/software system can be prepared. However, the result is two reliability models; one for the software and one for the hardware elements of the system. The basic hardware reliability model consists of all hardware elements of the system in series so that the overall logistics support requirements for spares, maintenance personnel, training, etc., can be readily assessed based on the failure rates of the equipment.

Individual software components (CSUs) as a rule do not fail independently and are not subject to wear out. Software components fail in association with the operational profile and must be modeled that way. Also, software components are not replaced independently like hardware units because they do not wear out and the unit itself is not independent.

For hardware/software systems, the results of the basic software reliability model can be used to estimate the number and types of equipment that must be supplied when a software maintenance facility is required as a part of the contractual effort.

5.2 Mission Reliability Model.

The mission reliability model is used to support allocations of system requirements to individual hardware and software elements and to support assessment of the compliance to requirements.

Reliability modeling methods used to model combined HW/SW systems for the purposes of reliability estimation and allocation need to accurately assess the interdependence between individual software elements, the hardware platforms on which these software elements execute, and the services provided by the system being analyzed.

Additionally, the methods used need to be based on and compatible with modern system engineering methods. This results in a modeling procedure consisting of six steps:

1) Development of a system Failure Modes and Effects Analysis (FMEA)
2) Development of the top level reliability model based on the system FMEA results
3) Development of detailed reliability block diagrams and models based on the top level reliability model
4) Modeling the combined hardware/software elements of the system
5) Development of hardware models and appropriate hardware failure rates
6) Use of the appropriate software failure rate modeling and calculation procedures of this notebook to predict software failure rate.

The reliability model(s) developed using these steps can then be used to estimate the reliability of the system being analyzed.

System reliability modeling is based on system FMEA, a bottom-up reliability analysis technique that provides a mapping between failures and their impact on system services. These FMEAs need not always be a formal analysis since the results are expressed in the reliability model block diagrams. A system-level adaptation of Matrix FMEA techniques which results in a compact, readily usable display of the needed FMEA information is used to support the development of system reliability models.

Estimation of system reliability characteristics is based on the reliability block diagrams and Markov state diagrams developed from the system FMEA and the individual hardware and software component failure rates. Estimation techniques for hardware reliability and maintainability characteristics are well known and can be applied to the hardware portions of combined hardware/software systems.

Software reliability characteristics can be estimated using the procedures provided in this notebook. For redundant, fault tolerant systems, software recovery characteristics are system design and implementation dependent. These recovery characteristics will need to be estimated on a case by case basis in conjunction with performance modeling and estimation.

Complex, redundant systems and system elements are modeled using state diagrams to accurately portray the possible operational and non-operational states. In general, these state diagrams will be complex enough to require access to automated tools for solution. They are not, strictly speaking, intractable. However, the labor required to manually determine a specific closed form solution for a state diagram that has been developed to model a specific design being analyzed is usually prohibitive. Automated solutions of these state diagrams are possible both analytically and through simulation. The

use of automated tools to support complex modeling is highly recommended. However, the user of automated reliability modeling tools will need to determine whether or not the numerical accuracy needed for their specific situation is supported.

Analytic solutions to complex diagrams often require the solution of a transition matrix with potentially significant losses in numerical accuracy due to the multiple arithmetic operations compounded by the accuracy limitations of the processing platform used. Monte Carlo solutions to state diagrams can result in both numerical inaccuracy due to the methods employed and may involve substantial cost for multiple program runs. The costs associated with Monte Carlo solutions rise dramatically as the accuracy requirements increase.

Maintainability, the time required to isolate and correct a fault in the design, is not used in the reliability modeling and allocation discussed in this section. Software maintenance is expected to proceed in parallel with ongoing system operation following a software failure. Thus, the time required to re-establish system operation following a software failure is used as the repair or recovery rate in the modeling of software elements of combined HW/SW elements. Note that Mean Time to Software Recovery (MTSWR) is not to be confused with MTTR. The MTSWR includes only the time to recover operations and does not include time to permanently repair the software.

Software maintenance will result in a software failure rate that is not constant over time due to the software corrections being implemented. However, for the purposes of modeling and allocation of combined hardware and software systems, an assumption of constant software failure rate during any operational period (i.e., between fixes), when the software is under configuration control is justified.

5.2.1 System FMEA Development. Reliability modeling of combined HW/SW systems, whether for reliability allocation or estimating purposes, is approached on a functional service basis using a matrix FMEA approach. The resultant FMEA can then be used to develop a HW/SW system reliability block diagram of independent elements. The individual series/parallel elements of the reliability block diagram can then be modeled. Non-redundant systems can be modeled as series strings of hardware and HW/SW system elements.

Development of a system FMEA to support creation of reliability models for use in reliability estimation and allocation begins with the use of the functional decomposition that has been developed as a part of the system engineering process. For small or relatively simple system structures, system functional analysis may have been omitted as a formal procedure. If the system level functional decomposition is not available, the reliability engineer may find it necessary to recreate this analysis using Data Flow Diagrams.

The functional decomposition of the system is used to identify the Hardware Configuration Items (HWCIs), the Computer Software Configuration Items (CSCIs) along with the processing provided by these CSCIs, and the allocation of CSCIs to various HWCIs within the system. The analyst can then begin to create the system level FMEA that will support reliability modeling of the combined HW and SW system.

The system level FMEA, illustrated in Figure 5-1, is a mapping of the hardware and software components of the system onto the system services provided. To create the FMEA, the analyst first constructs a matrix with each of the hardware CIs and their associated software CSCIs or CSCs as appropriate along the vertical axis. The horizontal axis is formed by the system services or outputs of the system grouped in convenient ways that support the desired analysis.

A grouping of system services by system operating mode often supports development of the various models required by the system specification. The HWCIs and CSCIs are then mapped onto the system services or outputs based on the impact on system services caused by the failure of each hardware and software element. In performing this mapping, the analyst will need to assess the impact of failure of both hardware platforms and software configuration items.

The failure impact of software elements will need to be examined in depth based on the data flow that has been established for the system design. Similarly, the failure of hardware platforms will need to be examined for its impact on software provided services using the data flow diagrams for the system software resident on the hardware platform.

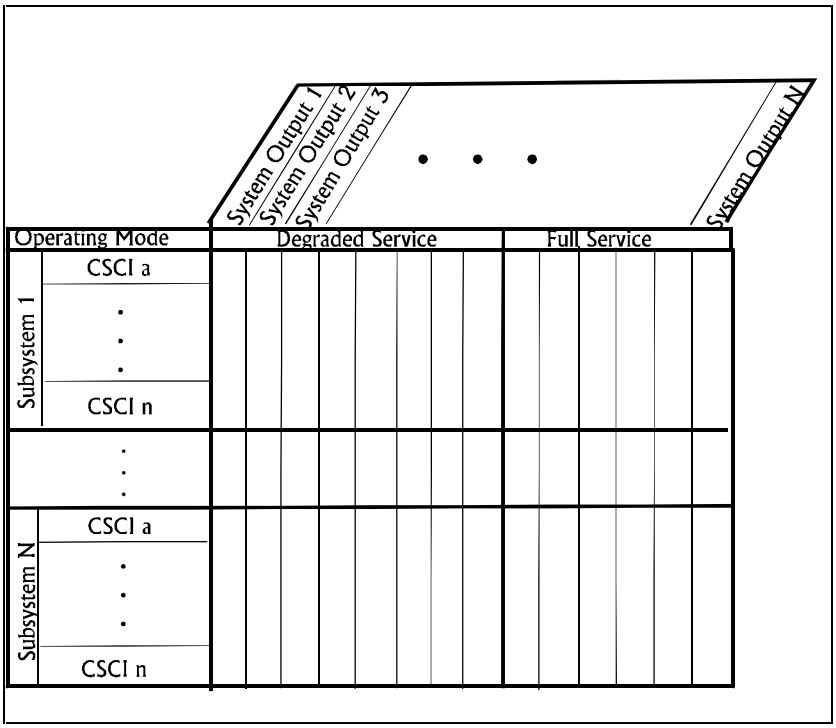

FIGURE 5-1. Example of System-Level Functional FMEA

Table 5-1 illustrates some of the generic failure modes for software systems. This chart is useful when analyzing the software at any level of architecture.

TABLE 5-1. Software Failure Modes

Undesired outputs	**Undesired processes**	**False alarms**	**Error handling**	**Timing**	**Sequences**
Output is invalid	Output is valid but process for getting output is not acceptable	Exception handling executes when there is no exception	Exception handling does not execute when there is an exception	Timing window is missed	Functions executed correctly but in incorrect order
Output is valid but not expected	Making a calculation based on invalid or incorrect inputs.		Exception handling executes incorrectly when there is an exception	Software functions properly but not within expected time	

Once the system level FMEA has been completed, the analyst can examine the hardware and software that are required for any particular mode or set of system services. A reliability model for these services can then be developed.

5.2.2 System Level Reliability Model Development.
The system level reliability model, expressed as a reliability block diagram for combined HW/SW systems, is developed based on system FMEA, using a procedure that is analogous to that used for purely hardware systems.

The FMEA results are used to determine which hardware and software elements are required to provide a set of system services required by the system mission or mode being modeled. The analyst then proceeds to develop a reliability block diagram that consists of a set of series blocks for each of the independent HW/SW subsystems that must be operational to provide the services being modeled.

At the system level of modeling, separation of the hardware and software elements in the system is not needed. For small systems and equipment, "system" level modeling as a separate activity may not be required. For these small systems, the methods discussed below under development of detailed reliability models may be directly applied.

5.2.3 Developing Detailed Reliability Models.
The system level reliability models which have been developed are then further decomposed to produce reliability models, expressed as reliability block diagrams, of increasing detail. These block diagrams are developed. The decomposition process is stopped when the reliability model block diagrams are sufficiently detailed to show all hardware and combined hardware/software elements of the system as single blocks.

The reliability analyst must use care in assessing the level of detail required by the system being analyzed. The detailed block diagrams must allow separation of hardware and software elements as a next step. However, the block diagrams must not be so detailed prior to the separation of the hardware and software elements that hardware and software interdependencies for redundant structures are lost from the model(s).

5.2.4 Reliability Modeling of Hardware/Software Elements.
Once the detailed reliability models are complete, the reliability analyst must further decompose those system elements containing hardware that hosts executing software. For the purposes of reliability modeling, software includes firmware which is configurable or under configuration control and hardware includes firmware which is not. Also, the media that the software is stored on should be treated as hardware. Decomposition of series elements which contain hardware and software is straightforward. Modeling of complex, redundant systems with recovery, is more difficult and requires a highly skilled analyst.

5.2.4.1 Modeling Series Hardware/Software Elements.
Series hardware/software elements are modeled as a series string consisting of the hardware platform and the software which executes on that platform as shown previously in Figure 4-3 and discussed in Section 4.1.1. As shown in that figure, the software further decomposes into two elements; non-developmental or re-used software, and newly developed software.

Failure rates for operating systems or executives, if available, can be obtained from the supplier of the operating system or executive. Failure rates obtained from the operating system supplier are usually quoted in the number of outages caused over some period of time (e.g., a year). Failure rates for operating systems are generally quoted with respect to system operating time because the operating system is active at all times when the computer is powered and ready for processing. The reliability analyst will need to convert the failure rate given to failures per hour for compatibility with hardware failure rates.

Defect data for re-used code can be obtained from applications where the code was previously used. The availability of this data depends on the completeness of organizational record keeping and the amount of code modification that has been necessary to allow the code re-use. The data for re-used code can only be used if the operational profile of the previously developed application resembles that of the current application. If the operational profiles are consistent, then the historical defect rates can be used and then converted to failure rates using the techniques in Section 7.

Estimates of the failure rate for newly developed software are obtained using the prediction procedures provided in Section 7 of this notebook. The failure rate estimates produced by these procedures is provided in failures per CPU operating second for each software element being developed. These failure rates must then be combined as discussed in Section 5.2.6 to account for the specific software topology and timing. Additionally, the resultant software failure rate can be converted to a system operating hour form as discussed in Section 5.2.6.5.

5.2.4.2 Modeling Redundant Hardware/Software Elements.
Reliability models of redundant and configurable or recoverable HW/SW elements are significantly more complex than reliability models of series hardware/software elements. The addition of redundancy introduces complexity associated with the ability of the hardware and software to correctly respond to failure events.

Reliability modeling of redundant HW/SW elements with hot standby and automatic switch-over capability significantly increases the complexity required to properly account for system behavior. Of necessity, this discussion of modeling redundant hardware/software elements will focus on the state diagrams which are used to accurately assess the behavior of these complex systems. This discussion is designed to assist an experienced reliability analyst to determine the information which will need to be included in the state diagrams developed for the system being analyzed.

5.2.4.2.1 Redundant Hardware Models.
A general model for hardware redundancy using identical equipment is shown in Figure 5-2. Redundant system elements transition to the next higher state upon occurrence of any hardware failure.

Hardware repairs transition the system element model to the next lower state. The system is a closed-form semi-Markov process that can be solved for the appropriate reliability measures using conventional methods. Closed-form solutions for the reliability measures of interest for this type of model under most common repair restrictions, types of standby, etc., are available in the technical literature referenced by this notebook.

The model shown in Figure 5-2 provides an upper bound on the reliability of redundant hardware-only systems. Estimation of the expected reliability of hardware systems requires that the fault tolerance employed in the redundancy be included in the model. For cold standby systems, where backup elements are not powered and thus immune to failure occurrence, the model of Figure 5-2 provides a reasonable estimate if the transition rates shown from each success state to the next higher number state are adjusted to account for the constant number of elements in operation (m units).

However, for hot standby systems with automatic switch-over, the model of Figure 5-2 significantly overstates the reliability achieved by the redundant hardware elements. Failures in the fault detection, fault isolation, and fault recovery mechanisms that may lead to latent faults in backup equipment, or an inability to activate redundant system elements and resume system services in response to primary element failures, are not included in the model shown in Figure 5-2.

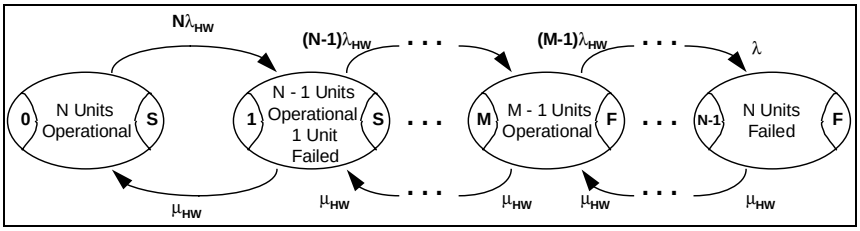

FIGURE 5-2. General Hardware Redundancy Model

Figure 5-3 is a simplified reliability model for a hardware system employing hot standby, and automatic switch-over with one of two identical elements required. The model accounts for failures in the fault detection, isolation, and recovery mechanisms. The concept of three types of "coverage" is introduced as a part of the model.

- Fault detection coverage (Cd) is the probability of detecting a fault given that a fault has occurred.

- Fault Isolation coverage (Ci) is the probability that a fault will be correctly isolated to the recoverable interface (level at which redundancy is available) given that a fault has occurred and been detected.

- Fault recovery coverage (Cr) is the probability that the redundant structure will recover system services given that a fault has occurred, been detected, and correctly isolated.

The model shown in Figure 5-3 is a simplified model since it does not separately consider the possible impact of transient failures. To account for transient failures would represent an uplift of failure rate by some percentage. The model also assumes that Cd is the same for both the primary element and the backup element. In practice, there may be different levels of fault detection coverage between primary and backup equipment due to a difference in test exposure intensity.

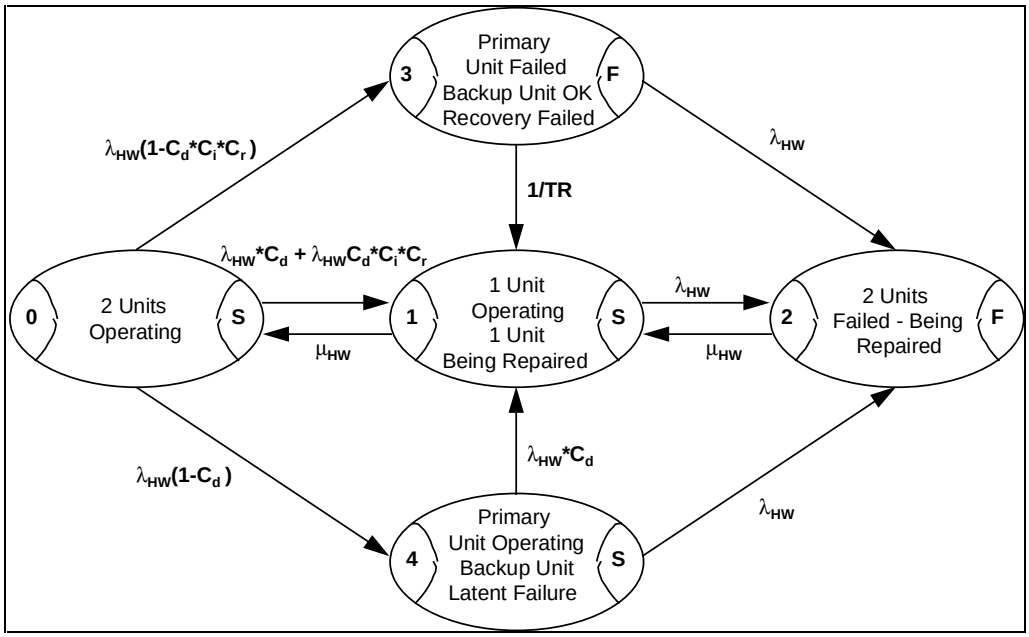

FIGURE 5-3. Hardware Reliability Model

As shown in the state diagram of Figure 5-3, the structure can transition from the full up state (1) to one of three states. The structure transitions to state 1 whenever a hardware failure occurs in the primary element that is correctly detected, isolated, and recovered from.

Similarly, a detected failure in the backup element results in a transition from state 0 to state 1. The structure transitions from state 0 to state 4 when a failure occurs in the backup equipment which is not detectable.

Failures in the primary hardware element that cannot be correctly detected, isolated, or recovered from result in a transition from state 0 to state 3. State 3 is a system state to account for the failure time accrued during manual intervention by system operations or maintenance personnel to restore lost system services.

Transitions from states 1, 3, and 4 to state 2 are caused by a hardware failure occurring prior to repair of the first failure which occurred in the system structure.

For an analogy of this model, consider a hospital operating room where it is critical that power is available for the equipment at all times. They might use a dual power supply system where both are operating continuously. If one fails the backup will automatically take over, ensuring that an operation can be carried out.

In actual practice, the model to be used will need to be based on the fault tolerant characteristics of the design being analyzed. Models that incorporate system fault behavior, such as shown in Figure 5-3, do not specifically include software as a part of the model. However, the system or structure control processing, a software based functionality, determines the model structure to account for system behavior under fault conditions.

The reliability estimates which result from the use of system reliability models that account for fault detection, isolation, and recovery are less optimistic than estimates from reliability models based only on the quantity of hardware supplied and required. The reliability of the system structure being modeled is usually very sensitive to the total fault coverage provided by the system design.

System designs that feature well-designed fault detection and isolation coupled with rapid and effective recovery of system services avoid most sudden losses of system services due to undetected latent failures in backup equipment or due to the inability of backup equipment to successfully restore system services when failures to the primary equipment occur. Similarly, models of HW/SW systems that include software as well as the fault tolerance characteristics of the system design are sensitive to the overall effectiveness of the fault detection, isolation, and recovery provided by the hardware and software designs.

5.2.4.2.2 Redundant HW/SW Models.

Inclusion of software into hardware reliability redundancy models further increases the complexity of the models. As in the hardware-only reliability models, accurate modeling of system behavior requires that fault coverage (Cd, Ci, Cr) be included into the model. Similarly, software fault coverage and the impact of long persistence faults must be included in the system models where appropriate. This results in each model of redundant HW/SW elements being uniquely tailored to the design.

5.2.4.2.2.1 Cold Standby Systems.

Redundant hardware/software systems that use cold standby techniques to provide fault tolerance can be modeled without undue difficulty as long as automatic switch-over and startup schemes are not used in the design. In general, only the hardware and software failure rates for the HW/SW elements need consideration in developing the reliability model.

For designs that use manual restoration of system services through the activation of an unpowered backup unit, an adaptation of the reliability model shown in Figure 5-4 can be used to estimate the reliability of the redundant structure. Structure state transitions are caused by either hardware or software failures. Hardware failures cause a transition to a state with one less hardware element and initiation of repair actions on the failed element if repair is allowed.

The reliability model of Figure 5-4 does not allow latent failures in the backup element to be modeled. The model assumes that failures of unpowered elements are impossible. Similarly, problems in recovering system services are not modeled since the recovery of system services must be directly managed by the system operator.

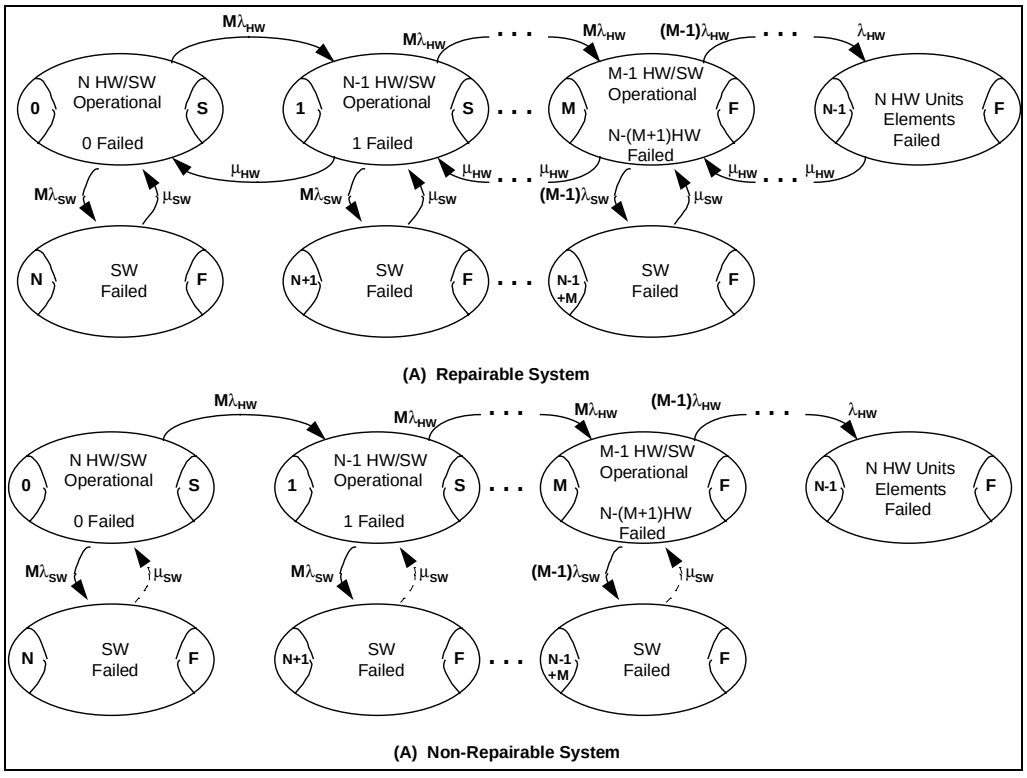

FIGURE 5-4. HW/SW Reliability Model

Software failures result in system recovery using the same processing hardware and a restart of the failed software. Both repairable and non-repairable systems are allowed to have software restarted to

enable recovery from software failures. Inclusion of the transition path allowing recovery from software failures is optional for non-repairable systems. The existence or lack of this transition path will depend on how the equipment is operationally employed.

5.2.4.2.2.2 Hot Standby Systems.

Reliability modeling of hot standby HW/SW structures requires consideration of hardware and software failure rates, fault detection, isolation, recovery coverage, and repair/recovery rates.

The effect of long persistence software failures on the reliability achieved by hot standby redundant structures is included in the software fault coverage estimates for recovery coverage. Depending on the system design being modeled, all or most of these parameters will be used to help identify states and/or transition rates between structure states. The exact state diagrams that result from an FMEA of the HW/SW redundant structure will depend on the design being evaluated. An example of a reliability model for a very simple redundant structure is discussed below.

Figure 5-5 presents a simplified state diagram for a hardware/software structure with one of two identical elements required. The model shown is for a hot standby system with automatic switch-over. In modeling this structure, five parameters of interest are recognized. The model states depend on primary hardware platform state (operational or failed), primary software state (operational or failed), backup hardware platform state, backup software state, and recovery status.

Recovery status is defined to have two states, successful or failed. A successful recovery indicates that the structure has successfully transitioned from primary equipment to the backup equipment after failure of either the primary hardware or software. Alternatively, successful recovery can indicate that a failure in a backup equipment was successfully detected, allowing repair of the backup equipment to commence. A failed recovery indicates that either recovery from primary to backup equipment has failed or that a failure has occurred in the backup equipment which has not been detected.

Since there are five parameters of interest, each of which has two possible values, a total of 32 possible states would be expected. However, some of the 32 possible states cannot exist in practice. Also, some of the states that can exist are functional duplicates that can be merged. For example, a state with a hardware failure in the primary equipment and operational software in the primary equipment can by shown to be one of the 32 possible states. However, the state is impossible because software cannot be operational on a failed hardware platform. The two states that can exist for (1) a failed backup equipment with successful recovery and (2) a failed primary equipment with successful recovery can be shown to be functionally equivalent since successful recovery implies that whichever hardware remains operational has been assigned to primary processing as a part of the recovery process.

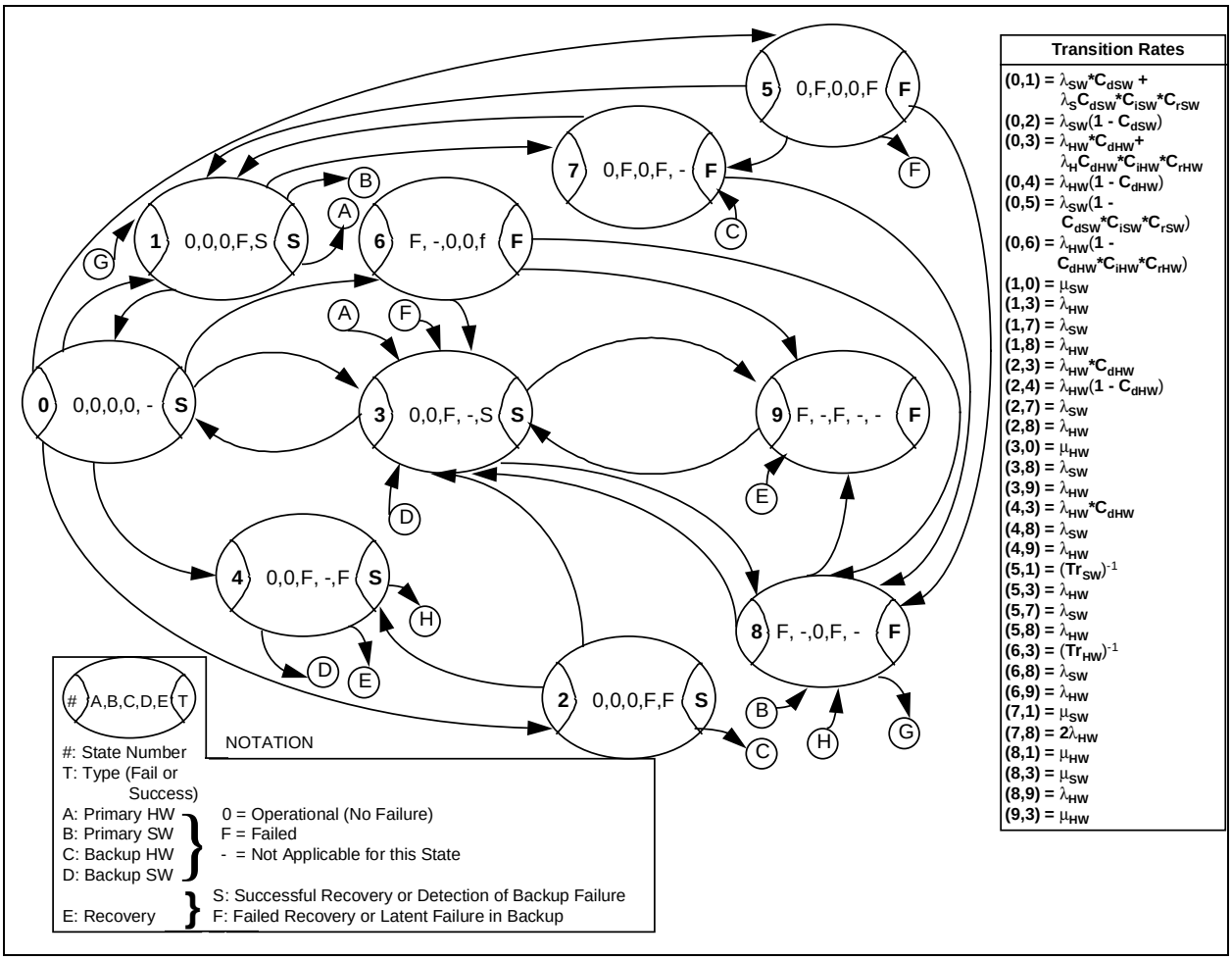

FIGURE 5-5. Simplified State Diagram

For the model of Figure 5-5, a total of ten states result, with the following definitions:

State 0: Success State - Fully Operational State

State 1: Success State - Backup has a detected software failure which is being recovered from.

State 2: Success State - System is operational with a latent software failure in the backup element.

State 3: Success State - System is operational with a detected hardware failure in the backup element.

State 4: Success State - System operational with a latent hardware failure in the backup element.

State 5: Failed State - Primary software has failed, recovery to the backup hardware and software has not been successful. System operations intervention will be required to restore system operation on either hardware platform.

State 6: Failed State - Primary hardware has failed. The recovery process has failed. Either incorrect detection, isolation, or incomplete recovery has occurred. Manual intervention by the system operator will be required to restore system services on the backup equipment.

State 7: Failed State - Software failures have occurred on both primary and backup system elements.

State 8: Failed State - The primary hardware and backup software have failed. Both elements are down, recovery is not possible without manual intervention by the system operator and/or maintenance personnel.
State 9: Failed State - Both hardware elements have failed.

As shown in Figure 5-5, transitions between states occur due to either failures in the hardware or software or due to the status of the recovery process. Using state diagrams that model the impact of hardware, software, and fault coverage for both hardware and software failures results in more accurate assessment of the potential reliability of redundant systems. Also, accurate models that reflect the system design decisions which have been made provide a basis for evaluating the reliability demands of candidate architectural approaches early in the design process.

5.2.5 Hardware Failure and Repair Rates.
Individual hardware components which are identified as blocks on the detailed reliability block diagrams should be decomposed into detailed internal models of the hardware where appropriate.

Hardware failure rates for use in combined HW/SW models should be obtained from the same sources as those traditionally used for hardware only reliability models. In service, field reliability records are the best estimators of expected hardware failure rates. When field reliability records are not available, reliability test results are the next best estimator of expected hardware reliability performance.

5.2.6 Software Failure Rates.
Determining software failure rates for use in combined hardware/software models requires that the software being analyzed be treated as a subsystem. A software subsystem, like hardware, can be viewed as a hierarchy. As far as reliability is concerned, however, the hierarchy consists of functions or operations rather than components (the term *function* is used for this discussion). A CSCI can perform one or more functions. The term *CSCI* could be used interchangeably with the term *function* in this section. It is used in Section 6 on Allocations in place of the function term. However, CSCI is not completely accurate here. A function is a capability of the system from the end user's perspective. This could be accomplished by a CSCI. It could also, however, be accomplished by a combination of CSCs or CSCIs. Therefore, the term function is used as it is more general. The concepts of functions and operations are described thoroughly in Section 9 on Operational Profiles.

The software functions that comprise a system will be related to one another in two ways: a particular timing configuration and a particular reliability topology.

Timing configuration is a concern when the various functions are active and inactive during a period of interest. Topology concerns the number of functions in the system that can fail before the system fails.

5.2.6.1 Timing Configurations.
Several different timing configurations are possible. The major timing relationships among software functions are "concurrent" and "sequential." Functions will be termed concurrent if they are active simultaneously. The functions are sequential if they are active one after the other. It is also possible for function times to partially overlap, resulting in a hybrid concurrent/sequential timing configuration.

Concurrently active software functions are found in systems that are serviced by more than one CPU, for example in a multiprocessing system or a distributed system. They are also found in single processor systems with preemptive schedulers. Process synchronization is a reliability consideration in concurrent processes.

5.2.6.2 Reliability Topology.
Reliability topology is the relationship between the failure of an individual function to the failure of the aggregate system. Generally, software functions or operations are related in a "series" topology, meaning that the failure of one function results in the failure of the software system. Software fault tolerance techniques can result in systems that can survive the failure of one or more functions. Software fault tolerance consists of a set of techniques which are not covered by this notebook, but are described in depth in the general software engineering literature.

5.2.6.3 Notation.
Capital letters will be used to refer to the aggregate system and lowercase letters to refer to a single function. The average aggregate failure rate will be denoted Λ, and the aggregate reliability representation will be denoted R(t). The functional failure rate is denoted λ, or λ_k for the k-th function.

5.2.6.4 Software Failure Rate Adjustment.
A computer program's failure rate can be expressed with respect to three different time frames of reference:

- execution time
- system operating time
- calendar time

Execution time is CPU time; it only accumulates or increments when the program is executing instructions. System operating time increments whenever the hardware/software system as a whole is operating. Calendar time, short periods of which are called wall-clock time, is always incrementing.

The ratio of a CSCI's execution time to system operating time is the CSCI's utilization u_i. The utilization can exceed 100% if copies of the software run on multiple CPUs reading different input streams. The CSCI's execution-time failure rate is multiplied by the CSCI's utilization to obtain the system-operating-time failure rate.

A software program can only fail when it is executing. The failures uncover faults, and the removal of the faults results in reliability growth. Thus, software reliability growth curves are based on cumulative execution time and express a single program's failure rate in terms of execution time. For scheduling purposes, execution time can be converted to calendar time.

During the operation of a system, programs may not operate continuously. For example, some of the programs may time-share a single CPU. Also, multiple CPUs may be present, allowing program executions to overlap. In order to combine the failure rates of the various programs with one another to arrive at an overall software failure rate, it is first necessary to translate all the program failure rates into a common time frame of reference. This frame of reference is system operating time, the same time frame used to express hardware failure rates.

If the programs are in a series configuration, then the (average) failure rate is simply the sum of the system-operating-time failure rates of the individual functions. This result can be derived as follows. Suppose there are N functions that run during the time period T. Let λ_i be the execution-time failure rate for the i-th function. Let $\mu(T)$ be the expected number of failures during that period. The expected number of failures contributed by the i-th function is $\lambda_i u_i T$. Thus

$$\mu(T) = \sum \lambda_i u_i T \qquad \textbf{(5.1)}$$

The overall failure rate is

$$\Lambda = \frac{\mu(T)}{T} = \sum \lambda_i u_i \qquad \textbf{(5.2)}$$

The sum $\Sigma\lambda_i u_i$ is seen to be the sum of the functions' system-operating-time failure rates.

5.2.6.5 SW Reliability Combination Models.
The solution for the reliability of an aggregate of series functions 1, ..., N is calculated by first determining the average failure rate

$$\Lambda = \frac{\sum_{k=1}^{N} \lambda_k \tau_{k'}}{T} \qquad \textbf{(5.3)}$$

where λ_k is the failure rate of the i-th function and τ_k' is the amount of time function k is active during period [0,T].

Procedures 5.2.6.5-1 through 5.2.6.5-3, below provide specific solutions for modeling the failure rate of software which is sequentially active, concurrently active, and for mission software where the activation times are indeterminate.

Procedure 5.2.6.5-1 - Sequentially active software model.
In this situation, software functions 1 through N are active one after the other. The time t_k is the point at which function k finishes and function (k+1) is activated.

The mission time T will lie between the times t_i and t_{i+1}. The average failure rate is

$$\Lambda = \frac{\sum_{j=1}^{i} \lambda_j (t_j - t_{j-1}) + \lambda_{i+1}(T - t_i)}{T} \qquad \textbf{(5.4)}$$

Sometimes the functions are not active consecutively; a period during which no program is active can be represented by a pseudo-function whose failure rate is zero. If a function is active intermittently, that is, for several piece-wise continuous periods, then a pseudo-function can be created for each such period. All pseudo-functions created for a particular function will have the same failure rate as that function.

Steps.
A. Determine the failure rate and stopping time of each function.

B. For a particular time T of interest, use the above formula to determine the average failure rate.

Example:
Suppose that there are four sequentially active functions, whose characteristics are provided in Table 5-2. Find R(100).

TABLE 5-2. Series Sequential Example

Function i	Start Time	End time t_i	Failure Rate λ_i
1	0	45	$3x10^{-5}$
2	45	200	$6x10^{-5}$
3	200	300	$2x10^{-5}$
4	300	800	$8x10^{-5}$

The average failure rate at time t = 100 is calculated by using equation **(5.4)**:

$$\Lambda = \left[\sum_{j=1}^{i} \lambda_j (t_j - t_{j-1}) + \lambda_{j+1} (T - t_i) \right] / T$$

$$= \frac{(45 - 0)(3x10^{-5}) + (6x10^{-5})(100 - 45)}{100} = 0.0000465$$

The reliability at time 100 is obtained as

$$R(100) = \exp[-\Lambda(100)] = \exp[-(0.0000465)(100)] = 0.995$$

Procedure 5.2.6.5-2 - Concurrently active software model.
If throughout a time interval software functions 1, ..., N are concurrently active, then the aggregate failure rate is

$$\Lambda = \sum_{k=1}^{N} \lambda_k \qquad \textbf{(5.5)}$$

If all functions have the same failure rate λ, the aggregate failure rate will be $\Lambda = N\lambda$.

Steps.
A. Start with the function failure rates λ_k.
B. Use the above formula to determine the average failure rate for the aggregate.

Example:
Suppose that there are three functions that run concurrently. The first function has a failure rate of $1.0x10^{-5}$, the second a failure rate of $4x10^{-4}$, and the third has a failure rate of $3x10^{-5}$. Find the aggregate failure rate.

The aggregate failure rate is the sum of the three failure rates:

$$\Lambda = \sum_{k=1}^{3} \lambda_k = (1x10^{-5}) + (4x10^{-4}) + (3x10^{-5}) = 0.00044$$

Procedure 5.2.6.5-3 - Mission oriented software combination model.
In many practical cases, the exact function starting and stopping times are unknown or non-deterministic. As long as the failure rate λ_j and total active time τ'_j for each operational mode are known, the average failure rate, and hence the reliability, can be obtained.

A mission-oriented system is described by means of a mission operational profile and consists of V consecutive time periods, called phases. During each phase the mission has to accomplish a specified task (similar to the function concept). An example of a space vehicle's mission phases are *ground operation*, *launch*, and *orbit*. Furthermore, at any point in time the system is in one of M possible operational modes. The effective operating time X_j for the j-th operational mode is given by

$$X_j = \sum_{i=1}^{V} t_i q_{ij} , \quad j = 1,2,\ldots,M \qquad \textbf{(5.6)}$$

or

$$X = TQ \qquad \textbf{(5.7)}$$

where t_i is the duration of the i-th mission phase and q_{ij} is the fraction of time the j-th mode is utilized during that phase.

In order to determine the duration of each mode, phase and utilization of software functions, determine the operational profile. Assign probabilities to each customer type, user type within each customer type and function profile within each user type and customer type. Finally determine the probabilities for each CSCI in a given mission or operational profile.

There are a variety of techniques for predicting the failure rate of the CSCI. Any of the techniques in Section 7 can be employed. One of those techniques is shown here.

Suppose there are N components in the aggregate. Let $SLOC_k$ be the executable source lines of code per function that are active during some mode j. Then the total SLOC per operational mode is defined as:

$$SLOC_j = \sum_{K=1}^{N} SLOC_{Ki} \quad k = 1,2,\ldots,N \qquad \textbf{(5.8)}$$
$$j = 1,2,\ldots M$$

Where:

$$SLOC_K = 0 \qquad \textbf{(5.9)}$$

When function k is not active in operational mode j.

Using the techniques for predicting failure rate given source lines of code in Section 7, calculate the failure rate for each operational mode.

Let λ_j be the failure rate of the j-th operational mode. The (average) failure rate over the mission can be calculated as

$$\Lambda = \frac{\sum_{j=1}^{M} \lambda_j X_j}{\text{mission duration}} \qquad \textbf{(5.10)}$$

Steps.

A. Determine the operational mission profile using Table 5-3.

TABLE 5-3. Operational Profile

Step	Description
Determine customer profile	Characteristics of multiple customers and occurrence probabilities
Determine user profile	Characteristics of multiple users within each customer profile and occurrence probabilities
Determine system mode profile	Characteristics of execution behavior and occurrence probabilities
Functional profile	Quantitatively describes relative use of different software functions (tasks or work to be done by system).

B. From the mission profile, form the row matrix of the durations $t_1, t_2, ..., t_N$ of the mission phases, where V is the number of mission phases. Next determine q_{ij} which are the fraction of time the j-th mode is utilized during the i-th phase, where M is the number of operational modes.

C. Compute the row matrix X by using equations **(5.7)** and **(5.8).**

The elements $x_1, x_2, ..., x_M$ are the effective operating times for each operational mode.

D. Use one of the prediction techniques in Section 7 to determine the failure rate of the CSCIs or functions. One technique is to determine the size of each CSCI or function that is executed in each operational mode. Use equations **(5.9)** and **(5.10).**

E. Using the function and/or operational mode information, compute the failure rate for each operational mode and denote the failure rates of each operational mode as $\lambda_1, \lambda_2, \ldots, \lambda_j$.

F. Calculate the average failure rate over the mission by applying equation 11.

G. Note that this technique can also model HWCIs and CSCIs by using the hardware prediction techniques for the HWCIs.

Example:
A mission has V = 8 phases. The names and durations are given by Table 5-4.

TABLE 5-4. Mission Phases

Phase Number i	Phase Name	Duration t_i (hours)
1	Start-Up	0.1
2	Taxi	0.1
3	Climb	0.2
4	Loiter	1.0
5	Attack	0.3
6	Return	0.2
7	Land	0.1
8	Shutdown	0.2

During the mission, there are M=4 operational modes (see Table 5-5).

TABLE 5-5. Operational Modes

Operational Mode j	Mode Name
1	Idle
2	Scan
3	Track
4	Maintenance

Suppose the mode utilization is

$$Q = \begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 1.0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 \\ 0 & 0.8 & 0.2 & 0 \\ 0 & 0.33 & 0.67 & 0 \\ 0.5 & 0.5 & 0 & 0 \\ 1.0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

From Table 5-4, the phase durations are

$$T = [\,0.1 \quad 0.1 \quad 0.2 \quad 1.0 \quad 0.3 \quad 0.2 \quad 0.1 \quad 0.2\,]$$

Then the effective operating times are computed as follows and summarized in Table 5-6.

$$X = TQ = [\,0.5 \quad 1.1 \quad 0.4 \quad 0.2\,]$$

TABLE 5-6. Effective Operating Time in Modes

Operational Mode j	Mode Name	Effective Time x_j
1	Idle	0.5
2	Scan	1.1
3	Track	0.4
4	Maintenance	0.2

Suppose that there are N = 5 software CSCIs (see Table 5-7).

TABLE 5-7. Software CSCIs

S/W CSCI k	Name	SLOC	Operational modes active in
1	Exec	3,000	All
2	Test	3,000	Maintenance
3	Scan	10,000	Scan
4	Track	5,000	Track
5	Calibrate	2,000	Idle and Maintenance

Assuming that equation (13) is used to calculate failure rate with r = 3 MIPS (million instructions per second), $K = 4.2*10^{-7}$, $\omega_0 = .006$ faults per SLOC, and the language is Ada the failure rates shown in Table 5-8 are calculated for each operational mode j. The initial failure rate is 6.048 for each operational mode. In this example, 500 hours of system test is scheduled. Therefore, each operational mode has the following test time expected, and therefore expected failure rate.

TABLE 5-8. Operational Mode Failure Rates

Operational mode j	Mode name	λ_0	SLOC per mode	ω_0	Expected system test time per mode	Failure rate per mode
1	Idle	6.048 failures per hour	5,000	30	.5/2.2 * 500 hours = 113.7 hours	.00211E-6 failures per hour
2	Scan	6.048 failures per hour	13,000	78	1.1/2.2 * 100 hours = 250 hours	.0608E-6 failures per hour
3	Track	6.048 failures per hour	8,000	48	.4/2.2 * 100 hours = 90.9 hours	114E-6 failures per hour
4	Maintenance	6.048 failures per hour	5,000	30	.2/2.2 * 100 hours = 45.45 hours	1003E-6 failures per hour

From Table 5-4, the mission time is found by summing the phase durations, yielding 2.2 hours. The average failure rate is found by multiplying the matrix λ_j by the matrix X_j.

$$\Lambda = \frac{\sum_{j=1}^{M} \lambda_j X_j}{\text{Mission duration}} =$$

$$\frac{.00211E-6+.0608E-6+114E-6+1003E-6}{2.2\text{hours}} = 1117.06E-6$$

failures per hour