6.0 RELIABILITY ALLOCATION

Reliability Allocation deals with the setting of reliability goals for individual subsystems such that a specified reliability goal is met and the hardware and software subsystem goals are well-balanced among themselves. Well-balanced usually refers to approximate relative equality of development time, difficulty, risk, or to the minimization of overall development cost.

6.1 System Reliability Allocation.
Reliability allocations for hardware/software systems can be started as soon as the system reliability models have been created. The initial values allocated to the system itself should either be the specified values for the various reliability metrics for the system, or a set of reliability values which are marginally more difficult to achieve than the specified values. Reliability values that are slightly more aggressive than the required values are sometimes allocated to the system to allow for later system functionality growth and to allow those parts of the system which cannot achieve their allocated values to be given some additional reliability margin later in the design process.

The apportionment of reliability values between the various subsystems and elements can be made on the basis of complexity, criticality, estimated achievable reliability, or any other factors considered appropriate by the analyst making the allocation. The procedures provided for allocation of software failure rates can be applied to both hardware and system elements provided the user recognizes that these elements typically operate concurrently.

System-level allocations are successively decomposed using the reliability model(s) until an appropriate set of reliability measures has been apportioned to each hardware and hardware/software component of the system.

6.2 Hardware Reliability Allocation.
The allocation of reliability values to lower-tiered hardware elements is a continuation of the allocation process begun at the system level. The system level hardware reliability models are used to successively apportion the required reliability measures among the various individual pieces of hardware and from the hardware equipment level to the various internal elements. For existing hardware items, the reliability allocations used should be based on the reliability performance of previously produced equipment. Reliability allocations to internal elements of existing hardware are not typically performed. Hardware equipment level allocations are further allocated to various internal elements within the equipment.

6.3 Software Reliability Allocation.
The first step in the allocation process is to describe the system configuration (system reliability model). Next, trial component reliability allocations are selected, using best engineering judgment. Compute system reliability for this set of component reliability values. Compare the result against the goal. Adjust component reliability values to move system reliability toward the

goal, and component reliability values toward better balance. Repeat the process until the desired goal and good balance are achieved[1].

The allocation of a system requirement to software elements makes sense only at the software system or CSCI level. Once software CSCIs have been allocated reliability requirements, a different approach is needed to allocate the software CSCI requirements to lower levels. The allocation of system requirements to hardware and software elements can be illustrated through a simple example.

Example 6.1:

A space vehicle will use three identical onboard computers, each with an independently designed and developed software program that performs the identical functions. The overall reliability goal for the vehicle is 0.999 for a 100-hr mission. Estimated development costs are:

Software		**Hardware**	
Reliability (100hr)	**Cost**	**Reliability (100hr)**	**Cost**
0 91	$ 3M	0 91	$ 300K
0.92	$ 3.25M	0.93	$ 400K
0.93	$ 3.5M	0.95	$ 500K
0.94	$ 3.75M	0.96	$ 600K
0.95	$ 4M	0.97	$ 800K
0.97	$ 4.5M	0.98	$ 1.5M
0.99	$ 5M	0.99	$ 5M

The computers operate simultaneously. Outputs are subject to a Built-in-Test (BIT). If any output passes, the system is operational. BIT is considered to have a reliability of 1. Allocate reliability goals to minimize cost. This is a concurrently functioning system with an event diagram of Figure 6-1.

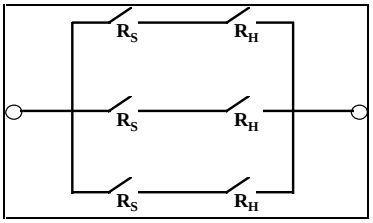

FIGURE 6-1. Event Diagram for Reliability Allocation

[1] Musa, John D;Iannino, A.; Okumoto, K; Software Reliability Measurement, Prediction, Application, McGraw-Hill, 1987.

R_S denotes the reliability of the software and R_H the computer hardware. Let R_{sys} be the overall system reliability. Applying the rules for event logic, the system reliability is given by

$$R_{sys} = 1 - (1 - R_H R_S)^3 .$$

For $R_{sys} = 0.999$, $R_H R_S = 0.9$

Try $R_H = 0.95$ and $R_S = 0.95$. This will give the required system reliability. Note that the estimated cost of achieving this hardware reliability is relatively low and the total cost is \$4.5M.

Try a more difficult goal for hardware: $R_H = 0.96$. An $R_S = 0.94$ will give the required combination to meet system reliability. Now the total cost is \$4.35M.

A trial goal for hardware of $R_H = 0.97$ and $R_S = 0.93$ will give the required combination to meet system reliability. Now the total cost is \$4.3M.

A trial goal for of $R_H = 0.98$ and $R_S = 0.92$ shows that we have gone too far. Now the total cost is \$4.75M. Thus, it is best to choose $R_H = 0.97$ and $R_S = 0.93$ to meet R_{sys} at low cost.

The reliability model for software differs significantly from hardware due to its inherent operating characteristics. For each mode in a software system's (CSCI) operation, different software modules (CSCs) will be executing. Each mode will have a unique time of operation associated with it. A model should be developed for the software portion of a system to illustrate the modules which will be operating during each system mode, and indicate the duration of each system mode. An example of this type of model is shown in Table 6.1 for a missile system.

TABLE 6-1. Software Functions By System Mode - Example

System Mode	**Modules**	**SLOC**
Standby - 2 Hours	BIT	4000
	1760 Interface	750
	Flight Sequencing	2000
	Prelaunch Initialization	900_
	TOTAL	**7650**
Prelaunch - 20 Minutes	BIT	4000
	Navigation	1000
	Flight Sequencing	2000
	Prelaunch Initialization	900_
	Navigation Kalman Filter	2000
	TOTAL	**9900**
Post-Launch - 10 Minutes	BIT	4000
	TRN	7000
	Navigation	1000
	IIR Seeker Control	500
	Flight Sequencing	2000
	Terminal Maneuver	1000
	Other Post-Launch	24500
	Navigation Kalman Filter	2000
	TOTAL	**42000**

The software reliability model will include the number of source lines of code (SLOC) expected for each module. This data, along with other information pertaining to software development resources (personnel, computing facilities, test facilities, etc.) are used to establish initial failure intensity predictions for the software modules. These predictions can be derived using the Basic Execution Model shown in Figure 6-2. This is just one example, other models could be used also.

Reliability

$R = e^{-\lambda\tau}$

Failure Intensity

$\lambda(t) = \lambda_o e^{-(\lambda_o/\nu_o)\tau}$ $\qquad$ $\lambda(\mu) = \lambda_o(1 - \mu/\nu_o)$

λ: failure intensity (failures/CPU hour)
τ_o: CPU execution time
λ_o: initial failure intensity
ν_o: total failures after infinite τ
μ: average or expected failures experienced

Total Faults

$\nu_o = \omega_o/B$

ω_o: inherent faults @ initial system test
B: fault reduction efficiency factor

Initial Failure Intensity

$\lambda_o = fK\omega_o$

f: linear execution frequency
K: fault exposure ratio

Instruction Execution Rate

$f = r / I$

r: instruction execution rate
I: number of object instructions

Object Instructions

$I = Q_x * I_s$

Q_x: average code expansion ratio
I_s: number of source instructions

Inherent Faults

$\omega_o = \rho * I_s$

ρ: fault density in faults/lines of code

Failure Identification Personnel

$X_I = (\Theta_I)\tau + (\mu_I)\mu$

Θ_I: average failure identification effort per CPU hour
μ_I: average failure identification effort per failure

Failure Correction Personnel

$X_F = \mu_F * \mu$

μ_F: average failure correction effort per failure

FIGURE 6-2. Basic Execution Time Software Reliability Model

The Musa model is provided as a method here because it has been applied on software development programs in the past. It may be a useful technique as a first course of action for programs with no other known methods.

A few of the parameters in the Musa model are considered in this notebook. These parameters are important to software reliability allocation and estimation regardless of the model or methodology used to calculate initial failure rates. A count of the *inherent faults* at the beginning of system test is one parameter. These are faults in the code as opposed to failures, which are the resulting incorrect outputs of executed faults. The inherent fault count can be used as a basis to estimate the expected failure intensity. This is accomplished directly in Musa's model. Fundamental logic suggests that the higher the inherent fault count, the higher the expected failure intensity of the software. In Musa's model, for example, a fault density of 6 faults per thousand Source Lines of Code (KSLOC) will translate into a higher failure intensity than 3 faults per KSLOC, and require more testing time and resources to achieve a required reliability level. The estimate for inherent fault density can be based on KSLOC, Function Points[2] (see appendix for a description of function points) or any other applicable primitive.

Another important parameter impacting reliability is the *defect removal efficiency*. (See Section 7. for a prediction technique for defect removal efficiency). Musa calls this the fault reduction factor. This is a measure of the proportion of faults removed from code to faults removed plus new faults introduced. The more efficient the defect removal process, the higher the rate of reliability growth during software testing. A related set of parameters are *rate of failure identification* and *rate of failure correction*. These parameters affect the resources needed for reliability growth.

The values used for the parameters in a chosen model for software reliability allocation and estimation are highly dependent on the sophistication of both the software process and software development personnel utilized on a program. The values should reflect this. For instance, a project with a Software Engineering Institute (SEI) development maturity capability of Level 1 or Level 2 should expect the fault content of their code to be greater than or equal to six faults per KSLOC at the beginning of system test. They can also expect their defect removal efficiency to be less than 90%. Many statistics have been developed correlating process capability to defect levels. Some useful references are provided in Section 7 of this notebook.

Once system reliability requirements have been allocated to software, the predictions derived using methods as described above become the basis for initial allocations of a software reliability requirement to the *system operational modes* or *functions*. This is a very important point. The software requirement is not allocated to modules (CSCs). It does not make sense to assign piece-part reliability values to software components in the way it is done for hardware. The reliability of software is impacted significantly by the environment in which it operates. This environment is characterized by a set of inputs comprising the *input space* for the software system. At the system, or CSCI level, it is possible to determine and enumerate the set and distribution of inputs for the functions that the software is called on to operate. In this way, the software operates as a *black box*. The reliability models for software have been developed for the software system, or this black box.

[2] Jones, Capers; Applied Software Measurement, McGraw-Hill, NY, 1991.

The modules that make up a software system are contained within the black box and, during system testing, are not accessible directly from the external environment. The controllable inputs represent stimuli to the system, not the lower level software elements independently. For this reason it is impractical to attempt to derive reliability values for software modules. Another problem is that many of the software failures that occur result from interface problems between software modules and timing problems somewhere in the system that may pertain to a specific module. Piece part reliability values for software modules, even if they could be calculated, would not roll up to a software system reliability value that is even close to the true reliability of the system. Fortunately, there is another method for allocating system software reliability values, and this method can use criticality and complexity for a basis of allocation, if desired.

A top software system reliability requirement can be allocated among system operational modes, functions, or operations. This is accomplished using a *system-mode profile*, *functional profile*, or *operational profile*, depending on level of detail needed and available for a particular system. System-mode profile is the most appropriate for high-level allocations. The functional profile breaks up the system modes into a set of functions carried out during a system mode. Finally, the operational profile splits the functional profile into a distinct set of possible *runs* within a system function. The system mode profile is most applicable for allocations. Functional and operational profiles are normally used to develop and generate test cases during system testing, and are described in further detail in Section 9.

A system mode is a set of functions or operations grouped for convenience in analyzing execution behavior. A system can switch among modes so that only one is in effect at a time, or it can allow several to exist simultaneously. A system mode profile is the set of system modes and their associated occurrence probabilities. There are no technical limits on how many system modes can be established. Some bases for characterizing system modes, with examples, include:

- *User Group*. Administration mode; maintenance mode.
- *Significant environmental conditions*. Overload versus normal traffic; Initialization vs. operation.
- *Criticality*. Shutdown mode for nuclear power plant in trouble.
- *User Experience*. Novice vs. expert mode.

TABLE 6-2. Sample System Mode Profile

System Mode	**Occurrence Probability**
Business Use	0.756
Personal Use	0.144
Attendants	0.062
System Administration	0.020
Maintenance	0.018

Example 6.2:

A trade study conducted for a missile system resulted in the allocation of a mission reliability requirement of 0.95 leading to 0.97 for hardware and R = 0.98 for the software system. The reliability requirement allocated to the software portion of system will be further broken down and allocated to mission segments (system modes) according to the software reliability model description shown in Table 6.1. The three mission segments (system modes) execute sequentially, and all three must execute failure-free for a reliable mission. A preliminary allocation has been performed. The allocation, by segment, is shown below:

Mission Segment	SR Allocation
Standby	0.9955
Prelaunch	0.9969
Post-Launch	0.9875

The allocation was optimized to minimize the amount of testing time needed to meet a system level requirement software reliability requirement of 0.98. The calculations were derived from the Musa model (Figure 6-2). The projected total testing effort in man-hours for system level reliability testing is 1200 hours. This effort includes the time to write and run test cases, diagnose faults, and repair software faults. This allocation also determines the initial predictions for each operational mode. Note that it is not the modules that are being tested, per se, in determining the software reliability values, rather it is the operational profile. In other words, it would be more appropriate to say that the Prelaunch stage has a failure rate of 0.02 failures per CPU hour rather than saying the Flight Sequencing software module has a failure rate of 0.002 failures/CPU hour.

Figure 6-3 shows the appropriate procedures used in different scenarios to allocate the failure rate goals to software CSCIs. Once predicted failure rates for the software CSCIs are known, Procedure 6.3-7 is used to re-allocate an appropriate failure rate goal to each software CSCI.

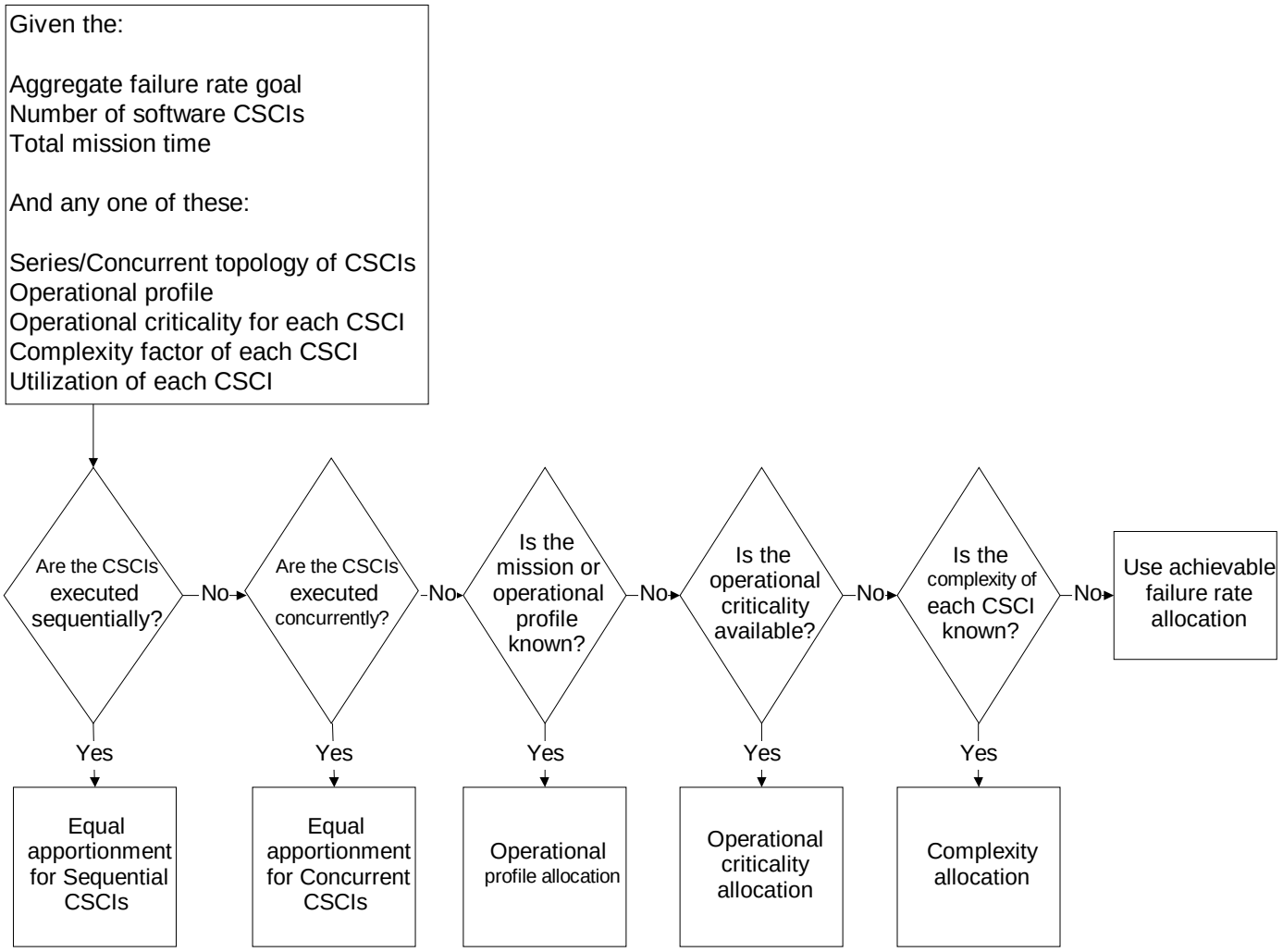

FIGURE 6-3. Reliability Allocation Procedures

Procedure 6.3-1 - Equal apportionment applied to sequential software CSCIs.
This procedure is used to allocate a failure rate goal to each individual software CSCI when the CSCIs are executed sequentially. This procedure should be used only when Λ_G, the failure rate goal of the software aggregate, and N, the number of software CSCIs in the aggregate, are known. The aggregate's failure rate goal is either specified in the requirements or is the result of an allocation performed at a higher level in the system hierarchy.

Steps
A. Determine Λ_G, the failure rate goal for the software aggregate

B. Determine N, the number of software CSCIs in the aggregate

C. For each software CSCI, assign the failure rate goal

$$\lambda_{iG} = \Lambda_G \text{ failures per hour} \qquad i=1,2,...,N \qquad \textbf{(6.1)}$$

Example:
A software aggregate is required to have a maximum of $\Lambda_G = 0.05$ failures per hour. The aggregate consists of N = 5 software CSCIs that are executed one after another, that is, the five CSCIs run sequentially. All CSCIs must succeed for the system to succeed (this is a series system).

Then, using the equal apportionment technique, the failure rate goal for the i-th software CSCI is assigned to be

$$\lambda_{iG} = \Lambda_G = 0.05 \text{ failures per hour} \quad i=1,2,...,5$$

Procedure 6.3-2 - Equal apportionment applied to concurrent software CSCIs.
This procedure is used to allocate the appropriate failure rate goal to each individual software CSCI, when the CSCIs are executed concurrently. Λ_G, the failure rate of the software aggregate, and N, the number of software CSCIs in the aggregate, are needed for this procedure.

Steps
A. Determine Λ_G, the failure rate goal for the software aggregate

B. Determine N, the number of software CSCIs in the aggregate

C. For each software CSCI, assign the failure rate goal to be

$$\lambda_{iG} = \Lambda_G / N \qquad i=1,2,...,N \qquad \textbf{(6.2)}$$

Example:
A software aggregate has a failure rate goal $\Lambda_G = 0.05$ failures per hour. The aggregate consists of N = 5 software CSCIs, which are in series and executed concurrently. Then, the allocated failure rate goal of each of the 5 software CSCIs is

$$\lambda_{iG} = \Lambda_G / N$$
$$= 0.05 / 5$$
$$= 0.01 \text{ failures per hour} \qquad i = 1,2,...,5$$

Procedure 6.3-3 - Optimized allocation based on system-mode profile.
This allocation procedure is the same as that introduced in Example 6.2. The optimization parameter in that example was estimated system testing time. Musa's Basic Execution Time Model was used in conjunction with linear programming to determine an allocation of a top software reliability requirement to operational modes which would minimize the amount of system testing required to meet the reliability objective. This procedure assumes that all system modes are independent. That is, no two system modes operate at the same time. The general procedure follows.

Steps.
A. Determine Λ_G, the failure rate goal for the software aggregate

B. Determine N, the number of System Modes in the aggregate

C. For each system mode, determine the proportion of time that the system will be operating in that mode. In other words, assign occurrence probabilities to the system modes. The failure rate relationship of the software system can then be established by

$$\Lambda_G = \sum_{i=1}^{i=N} p_i \lambda_{iG} \qquad \textbf{(6.3)}$$

where p_i is the occurrence probability of the i-th system mode.

The only unknowns in Equation 6.3 then are the λ_{iG} values. There are potentially an infinite number of combinations of these values that will meet the Λ_G requirement.

D. For each system mode, identify the software modules (CSCs or, in some cases, CSCIs) that are utilized during the system mode operation. Add the estimated source lines of code (SLOC) for each module (This was done in Table 6-1). Total the SLOC count for each system mode.

E. Use the SLOC count for each system mode as an input to the Musa model (Figure 6-2). This is the source instruction parameter (I_S) in Figure 6-2.

F. Supply values for each of the following parameters to the Musa model:

B: fault reduction efficiency factor (between 0 and 1)
r: instruction execution rate (e.g., 25 MIPS)
$\mathbf{Q_x}$: code expansion ratio (# of object instructions per SLOC)
μ_i: failure identification effort per failure (manhours)
μ_f: failure correction effort per failure (manhours)

G. Select a target failure rate for each system mode such that the combination results in the aggregate system requirement being met. Note that the value for expected failures experienced (μ) will be calculated by the model. Plugging this into the equations for failure identification personnel (X_I) and failure correction personnel (X_F) provides the total test time (in manhours) required to meet the target failure rate. (Failure identification includes time to set up and run tests and to identify if a failure has occurred. Failure correction includes time to debug the software, find the underlying fault causing the failure, and make the software change that fixes the associated fault).

H. Add the values for X_F and X_I for each of the system modes to arrive at total system test time. Record this value.

I. Try *N* new values for failure rates for the N system modes. Record the total test time.

J. Tweak the failure rate values up and down until an optimal value which minimizes system test time is achieved. These final failure rate values are the set to be allocated to the N system modes.

Alternatively, the parameters required as described above could be entered into a linear programming model. Setting Test Time as the objective function to minimize, the optimization will determine the best individual failure rate values for the allocation. There are several linear programming packages, including those supplied with spreadsheet programs, available to perform this analysis.

Example 6.2 (continued):
The failure rate for the software CSCI is calculated by converting the reliability value of 0.98 to a failure rate. The mission time in this case is 2.5 hours. Using the standard equation for reliability (assuming exponential failure rates), $\mathbf{R = e^{-\lambda t}}$, the aggregate failure rate goal is $\mathbf{8.078\ X\ 10^{-3}}$ failures/hour. The number of system modes is 3.

Probabilities for each system mode are simply the proportion of time that each operates. Therefore,

$p_1 = 0.8$
$p_2 = 0.133$
$p_3 = 0.067$

The SLOC (I_S) count for the three system modes, as documented in Table 6-1, are:

Standby: 7,650
Prelaunch: 9,900
Post-launch: 42,000

The following values were assumed for each of the other required parameters:

B: 0.95
r: 25 MIPS
$\mathbf{Q_x}$: 4.5
μ_i: 5
μ_f: 5

The parameters of the Musa Model were set up in a spreadsheet program. The values above were entered. Using the built-in linear optimization utility, the linear program execution resulted in the following allocation of failure rate values:

Standby: 8.01×10^{-4} failures/CPU hr
Prelaunch: 6.607×10^{-3} failures/CPU hr
Post-launch: 9.8306×10^{-2} failures/CPU hr

The values determined by the model should be checked to ensure that the top failure rate will be met with the combination of allocated failure rates. In this case, using Equation 6.3, they do. These values then become the failure rates allocated to each of the respective system modes.

This procedure does not require that the System Mode factor be used as the basis of allocation. Any of the other operational profile factors listed in Table 6-3 could be substituted in the allocation (p_i values). The System Mode factor, however, makes sense for most systems.

TABLE 6-3. Operational Profile Allocation Factors

Operational Profile Factors	**Description**
Customer Profile	The occurrence probabilities associated with distinct customer types
User Profile	The occurrence probabilities associated with distinct end user types within the customer types
System Mode Profile	The occurrence probabilities associated with distinct system modes within each customer and end user types
Functional Profile	The occurrence probabilities associated with each CSCI within each system mode, user and customer profile.

Procedure 6.3-4 - Allocation based on operational criticality factors.
The operational criticality factors method allocates failure rates based on the system impact of a software failure. Criticality is a measure of

- The system's ability to continue to operate
- The system's ability to be fail-safe

For certain modes of operation, the criticality of that mode may call for a higher failure rate to be allocated. In order to meet very high failure rates, fault-tolerance or other methods may be needed.

The following procedure is used to allocate the appropriate value to the failure rate of each software CSCI in an aggregate, provided that the criticality factor of each CSCI is known.

A CSCI's criticality refers to the degree to which the reliability and/or safety of the system as a whole is dependent on the proper functioning of the CSCI. Furthermore, gradations of safety hazards translate into gradations of criticality. The greater the criticality, the lower the failure rate should be allocated.

A criticality factor c_i should be assigned to each CSCI. The value that should be assigned to c_i is related to the criticality of the CSCI i.

Steps
A. Determine Λ_G, the failure rate goal of the software aggregate

B. Determine N, the number of software CSCIs in the aggregate

C. For each i-th CSCI, i = 1, 2, ..., N, determine its criticality factor c_i. *The lower the c_i the more critical the CSCI.*

D. Determine τ'_i, the total active time of the i-th CSCI, i = 1, 2, ..., N. Determine T, the mission time of the aggregate.

E. Compute the failure rate adjustment ξ:

$$\xi = \frac{\sum_{i=1}^{N} c_i \ \tau_{i'}}{T} \qquad \textbf{(6.4)}$$

F. Compute the allocated failure rate goal of each CSCI

$$\lambda_{Gi} = \Lambda_G \cdot c_i / \xi \qquad \textbf{(6.5)}$$

(Dividing by ξ makes the allocated CSCI failure rates build up to the aggregate failure rate goal.)

Example:
Suppose a software aggregate consisting of N = 3 software CSCIs is to be developed. Assume the failure rate goal of the aggregate is $\Lambda_G = 0.002$ failures per hour. Suppose that the mission time is T = 4 hours. Furthermore, the criticality factors and the total active time of the software CSCIs are:

$c_1 = 4$ $\quad\quad$ $\tau'_1 = 2$ hours
$c_2 = 2$ $\quad\quad$ $\tau'_2 = 1$ hour
$c_3 = 1$ $\quad\quad$ $\tau'_3 = 2$ hours

(Note: In this example, since c_3 has the smallest value, it indicates that the third CSCI of this software aggregate is the most critical.)

Compute ξ using equation **(6.5):**

$$= \frac{(4)(2) + (2)(1) + (1)(2)}{4}$$

$$= 3$$

Then, the allocated failure rate goals of the software CSCIs are

$\lambda_{1G} = \Lambda_G \cdot c_1 / \xi$
$= (0.002)(4) / 3$
$= 0.0027$ failures/hour

$\lambda_{2G} = \Lambda_G \cdot c_2 / \xi$
$= (0.002)(2) / 3$
$= 0.0013$ failures/hour

$\lambda_{3G} = \Lambda_G \cdot c_3 / \xi$
$= (0.002)(1) / 3$
$= 0.00067$ failures/hour

Procedure 6.3-5 - Allocation based on complexity factors.
The following procedure is used to allocate a failure rate goal to each software CSCI in an aggregate, based on the complexity of the CSCIs. There are several types of complexity as applied to software that are summarized below. Note that several algorithms exist for computing function points and feature points. The appendix contains a description of one method for computing each of the complexity measures listed in Table 6-4.

TABLE 6-4. Complexity Procedures

Complexity type	**Description**	**When it can be used**
McCabe's Complexity	A measure of the branches in logic in a unit of code.	From the start of detailed design on.
Functional Complexity	A measure of the number of cohesive functions performed by the unit.	From the start of detailed design on.
SPR Function points	A measure of problem, code, and data complexity, inputs, outputs, inquiries, data files and interfaces.	From detailed design on.
SPR Feature points	A measure of algorithms, inputs, outputs, inquiries, data files and interfaces.	From detailed design on.

During the design phase an estimate complexity using any one of these techniques is available. The greater the complexity, the more effort required to achieve a particular failure rate goal. Thus, CSCIs with higher complexity should be assigned higher failure rate goals.

The complexity measure chosen must be transformed into a measure that is linearly proportional to failure rate. If the complexity factor doubles, for example, the failure rate goal should be twice as high.

Steps.

A. Determine Λ_G, the failure rate goal of the software aggregate

B. Determine N, the number of software CSCIs in the aggregate

C. For each CSCI i, i = 1,2,...,N, determine a complexity factor w_i.

D. Determine τ'_i, the total active time of the i-th CSCI, i = 1,2,..., N. Determine T, the mission time of the aggregate

E. Compute ξ:

$$\xi = \frac{\sum_{i=1}^{N} w_i \ \tau_{i'}}{T} \qquad \textbf{(6.6)}$$

G. Then, the allocated failure rate of the i-th CSCI is

$$\lambda_{iG} = \Lambda_G \cdot w_i / \xi \qquad \textbf{(6.7)}$$

Example:
A software aggregate consisting of N = 4 software CSCI is to be developed. The failure rate goal of the aggregate is Λ_G = 0.006 failures per hour. The mission time is T = 3 hours. Furthermore, the complexity factors and the total active time of the software CSCIs are:

$w_1 = 4$, $t_1 = 2$ hours
$w_2 = 2$, $t_2 = 1$ hour
$w_3 = 3$, $t_3 = 3$ hours
$w_4 = 1$, $t_4 = 2$ hours

Compute ξ using equation **(6.6):**

$$= \frac{(4)(2) + (2)(1) + (3)(3) + (1)(2)}{3}$$

$$= 7$$

Then, the failure rate goals of the software CSCIs are

$\lambda_{1G} = \Lambda_G \cdot w_1 / \xi$
$= (0.006)(4)/7 = 0.0034$ failures/hour

$\lambda_{2G} = \Lambda_G \cdot w_2 / \xi$
$= (0.006)(2) / 7 = 0.0017$ failures/hour

$\lambda_{3G} = \Lambda_G \cdot w_3 / \xi$
$= (0.006)(3) / 7 = 0.0026$ failures/hour

$\lambda_{4G} = \Lambda_G \cdot w_4 / \xi$
$= (0.006)(1) / 7 = 0.0009$ failures/hour

Procedure 6.3-6 - Allocation based on achievable failure rates.
Allocation based on achievable failure rates uses each CSCIs utilization. The utilization governs the growth rate a CSCI can be expected to experience during system test. All things being equal, the greater a CSCI's utilization, the faster its reliability will grow during system test.

The first step is forecasting each CSCI's initial failure rate as a function of predicted size, processor speed, and industry average figures for fault density and fault exposure ratio. Then the reliability growth model parameter, the decrement in failure rate (with respect to execution time) per failure occurrence, is predicted. Next, it is determined how much growth each CSCI can be expected to achieve during system test. The growth is described by the software reliability growth model. From the growth model, each CSCI's failure rate at release is predicted, taking into account that CSCI's utilization. These predicted at-release failure rates provide relative weights, which are used to apportion the overall software failure rate goal Λ_G among the CSCIs.

Using the Proposal/Pre-Contractual Stage software reliability prediction equation, the i-th CSCI's initial failure rate λ_{0i} (with respect to execution time) is obtained:

$$\lambda_0 = r_i \bullet K \bullet \omega_{0i} / I_i \qquad \textbf{(6.8)}$$

where:

r_i is the host processor speed (average number of instructions executed per second)

K is the fault exposure ratio (Musa's default is 4.20×10^{-7}, however, it is strongly suggested that the organization determine an estimate of fault exposure based on historical data.)[3]

ω_{0i} is the CSCI's fault content (use the number of developed lines of source code in the CSCI times the predicted fault density, or use the function points times the predicted faults per function points as discussed in Section 7)

I_i is the number of object instructions in the CSCI (number of source lines of code times the code expansion ratio or the number of function points times the code expansion ratio (see Table 7-9).

It is thus assumed that each CSCI is developed by a mature, reproducible software development process that produces code with the approximate industry average indicated in Section 7.2.3. Note that, in this model, the initial failure rate is primarily a function of fault density (faults per lines of code or faults per function points).

The software reliability growth model is employed to forecast the failure rate (with respect to system operating time) $\lambda_i(t)$ each CSCI will exhibit after t system operating time units of system test. Each

[3] Musa, John D;Iannino, A.; Okumoto, K; Software Reliability Measurement, Prediction, Application, McGraw-Hill, 1987.

CSCI's utilization u_i is the CSCI's ratio of execution time to system operating time. The growth model provides the formula

$$\lambda_i(t) = \lambda_0 \bullet \exp[-\beta_i \bullet t \bullet u_i] \bullet u_i \qquad \textbf{(6.9)}$$

For each CSCI the parameter β_i can be computed from

$$\beta_i = B \bullet \frac{\lambda_{0i}}{\omega_{0i}} \qquad \textbf{(6.10)}$$

integrated with one another all at once, but through a series of incremental builds, each CSCI can have its own value of t.

A relative failure rate weight w_i can now be associated with each CSCI. The weight is computed from the formula

$$w_i = \frac{\lambda_i(t)}{\sum \lambda_i(t)} \qquad \textbf{(6.11)}$$

Finally, the failure rate goal (with respect to system operating time) λ_{Ai} to be allocated to the i-th CSCI is as follows

$$\lambda_{Ai} = \Lambda_G \bullet w_i \qquad \textbf{(6.12)}$$

If desired, the failure rate allocation can be expressed in terms of execution time by dividing by u_i.

The allocation uses failure rates expressed with respect to system operating time. Note that changing to a faster or slower processor does not affect a system-operating-time failure rate; the reduction or increase in the utilization is exactly offset by a proportionate increase (decrease) in the execution time failure rate. Therefore, the allocation obtained from this method does not need to be modified if the hardware platform changes to a faster or slower processor.

Steps.

A. Predict each CSCI's initial failure rate (with respect to the CSCI's execution time). Equation number **(6.8)** is one method for doing this. There are other formulas used in industry as well.

B. Let t equal the number of system operating time units to be expended in system test. Compute each CSCI's using failure rate at release (with respect to system operating time) by the formulas **(6.9)** and **(6.10).** Use the predicted fault removal efficiency described in Section 7.2.4.

C. Calculate each CSCI's relative failure rate weight by using formula **(6.11).**

D. Calculate the failure rate goal of each CSCI using formula **(6.12).**

Example:
A software subsystem consists of three CSCIs. The first CSCI, hosted on a 3-MIPS processor, is predicted to contain ΔI_{S1} = 10,200 lines of newly developed Ada source code and 2,000 lines of reused Ada source code, for a total of 12,200 lines of source code. The second CSCI, hosted on 4- MIPS processor, is predicted to consist of ΔI_{S2} = 20,000 lines of Ada source code, all newly developed. The third CSCI, hosted on a 4-MIPS processor, is predicted to consist of ΔI_{S3} = 45,000 lines of newly developed Ada source code. Since the code expansion ratio for Ada is 4.5 object instructions per source line of code, the number of object instructions are I_1 = 54,900; I_2 = 90,000; and I_3 = 202,500.

The first CSCI has a utilization factor of 20%; the second CSCI, 20%, and the third, 40%. The failure rate goal for the software subsystem is Λ_G = 0.00005 failures per system operating second. Table 6-5 summarizes these figures. The software has an expectation of 250 hours of operation.

TABLE 6-5. CSCI Characteristics

CSCI i	ΔI_{Si}	I_i	r_i	u_i
1	10,200	45,900	3,000,000	0.2
2	20,000	90,000	4,000,000	0.2
3	45,000	202,500	4,000,000	0.4

The predicted fault contents ω_{01}, ω_{02}, and ω_{03} are

$$\omega_{01} = \Delta I_{S1} x0.006 = 10{,}200x0.006 \approx 61$$

$$\omega_{02} = \Delta I_{S2} x0.006 = 20{,}000x0.006 = 120$$

$$\omega_{03} = \Delta I_{S3} x0.006 = 45{,}000x0.006 = 270$$

The initial failure rates λ_{01}, λ_{02}, and λ_{03} are

$$\lambda_{01} = r_1 xKx \omega_{01} / I_1 = (3x10^6)(4.20x10^{-7})(61) / (54{,}900) = 0.0014$$

$$\lambda_{02} = r_2 xKx \omega_{02} / I_2 = (4x10^6)(4.20x10^{-7})(120) / (90{,}000) = 0.00224$$

$$\lambda_{03} = r_3 xKx \omega_{03} / I_3 = (4x10^6)(4.20x10^{-7})(270) / (202{,}500) = 0.00224$$

The software reliability growth model parameters β_1, β_2, and β_3 are

$$\beta_1 = Bx\lambda_{01} / \omega_{01} = (0.955)(0.0014) / (61) = 0.0000219$$

$$\beta_2 = Bx\lambda_{02} / \omega_{02} = (0.955)(0.00224) / (120) = 0.0000178$$

$$\beta_3 = Bx\lambda_{03} / \omega_{03} = (0.955)(0.00224) / (270) = 0.0000079$$

Table 6-6 summarizes.

TABLE 6-6. Growth Model Quantities

CSCI i	ω_{0i}	λ_{0i} per system operating second	β_i per system operating second
1	61	0.0014	0.0000219
2	120	0.00224	0.0000178
3	270	0.00224	0.0000079

The number of hours system test is expected to last is 250. Thus,

$$t = 250 \text{ hr } x \frac{3600 \text{ sec}}{1 \text{ hr}} = 900{,}000 \text{ sec}$$

The failure rates (with respect to system operating time) at the end of the 900,000-second system test period are predicted to be

$$\lambda_1(900{,}000) = \lambda_{01}\exp[-t \bullet \beta_1 \bullet u_1] \bullet u_1 = 0.0014\exp[-(0.0000219)(900{,}000)](0.2)](0.2) = 5.434632x10^{-6}$$

$$\lambda_2(900{,}000) = \lambda_{02}\exp[-t \bullet \beta_2 \bullet u_2] \bullet u_2 = 0.00224\exp[-(0.0000178)(900{,}000)(0.2)](0.2) = 1.805865x10^{-5}$$

$$\lambda_3(900{,}000) = \lambda_{03}\exp[-t \bullet \beta_3 \bullet u_3] \bullet u_3 = 0.00224\exp[-(0.0000079(900{,}000)(0.4)](0.2) = 5.214039x10^{-5}$$

The sum of the failure rates is

$$\sum_{1}^{3} \lambda_i (900{,}000) = 5.9393847 X 10^{-5}$$

The computed failure rates weights are

$$w_1 = \lambda_1 (900{,}000) / 7.5633672 x 10^{-5} = .0718547$$

$$w_2 = \lambda_2 (900{,}000) / 7.5633672 x 10^{-5} = 0.2387647$$

$$w_3 = \lambda_3 (900{,}000) / 7.5633672 x 10^{-5} = 0.6893807$$

The allocated failure rates (with respect to system operating time) are computed by multiplying the system failure rate goal (Λ_G = 0.00005 failures per system operating second) by each of the weights w_1, w_2, and w_3:

$$\lambda_{G1} = \Lambda_G x w_1 = 0.00005x0.0718547 = 3.592735x 10^{-6}$$

$$\lambda_{G2} = \Lambda_G x w_2 = 0.00005x.2387647 = 11.938235x 10^{-6}$$

$$\lambda_{G3} = \Lambda_G x w_3 = 0.00005x0.6893807 = 34.469035x 10^{-6}$$

Procedure 6.3-7 - Re-allocation based on predicted failure rates.

Once failure rate predictions become available for the CSCIs, the allocations can be revised. If the predicted failure rate of the aggregate is less than or equal the goal, the CSCIs are re-allocated failure rates based on the ratio of the predicted failure rate of the CSCI to the predicted failure rate of the aggregate. The failure rate "goal" is expressed in terms of the failure rate at release. The predicted failure rates at release are obtained by using the appropriate software reliability prediction model and a software reliability growth model. If the new failure rate goal of a particular CSCI is greater than its predicted failure rate, it may be possible to use some resources from that CSCI and use the resources to help other CSCIs reach their respective goals. If the predicted failure rate of the aggregate is greater than the goal, then actions such as re-design or resource re-allocation should be considered.

Steps.

A. Using the procedures in Section 7, predict the failure rates the N CSCIs will have at release:

$$\lambda_{1P}, \lambda_{2P}, ..., \lambda_{NP} \qquad \textbf{(6.13)}$$

The P stands for "predicted."

B. Combine the predicted CSCI failure rates to obtain a predicted failure rate for the aggregate:

$$\Lambda_P = \frac{\sum_{i=1}^{N} \lambda_{iP}\, \tau_{i'}}{T} \qquad \textbf{(6.14)}$$

C. If $\Lambda_P \leq \Lambda_G$, then the failure rates are re-allocated using the formula:

$$\lambda_i = \frac{\lambda_{iP}}{\Lambda_P} \bullet \Lambda_G \quad , \quad i = 1,2,\ldots,N \qquad \textbf{(6.15)}$$

If $\Lambda_P > \Lambda_G$, then it is likely that the failure rate goal will not be met, and redesign and/or management action is required.

Example:
A software aggregate consisting of N = 3 software CSCIs is to be developed with the failure rate goal of $\Lambda_G = 0.8$ failures per hour. The total active times of the software CSCIs are

$t_1 = 2$ hours
$t_2 = 3$ hours
$t_3 = 1$ hour

After the detailed design phase, the predicted failure rates of the software CSCIs are

$\lambda_{1P} = 0.5$ failures/hour
$\lambda_{2P} = 0.9$ failures/hour
$\lambda_{3P} = 0.2$ failures/hour

Compute Λ_p using equation **(6.14)**:

$$= \frac{(0.5)(2) + (0.9)(3) + (0.2)(1)}{6} = 0.65$$

The predicted failure rate of the aggregate is less than its failure rate goal $\Lambda_G = 0.8$. Therefore, the re-allocated failure rate goals of the software CSCIs are

$\lambda_{1G} = \lambda_{1P} \cdot \Lambda_G / \Lambda_P$
$= (0.5)(0.8)/(0.65)$
$= 0.62$ failures/hour

$\lambda_{2G} = \lambda_{2P} \cdot \Lambda_G / \Lambda_P$
$= (0.9)(0.8)/(0.65)$
$= 1.1$ failures/hour

$\lambda_{3G} = \lambda_{3P} \cdot \Lambda_G / \Lambda_P$
$= (0.2)(0.8)/(0.65)$
$= 0.25$ failures/hour

6.4 Hardware/Software Allocations.

Once the hardware/software elements of the system have been allocated a set of reliability goals, these allocations must be apportioned between the hardware platform and the executing software. To apportion the allocated reliability measures between the hardware and software, the analyst should first determine if the actual reliability of any of the elements of the hardware/software combination is known. The hardware element may be an existing design with proven reliability.

Prior to apportioning the allocated reliability values to the hardware platform or software elements, those elements with known reliability should be assigned a set of allocated values that reflect their known reliability performance. These allocations should be subtracted from the hardware/software total allocations.

The remaining elements can then be apportioned a set of reliability goals based on mission operational profile, operational criticality, complexity, achievable failure rate, or such other factors as the analyst may deem appropriate to the system being developed. Hardware elements of hardware/software combinations which have allocations that need to be further developed should be treated in exactly the same way as purely hardware elements.

The combination of hardware and software configuration items can also be modeled by using the procedures used above. For example, the mission allocation can model the mission profile of HWCIs as well as CSCIs.