1-1993

Planning and Certifying Software System Reliability

J. H. Poore

Harlan D. Mills

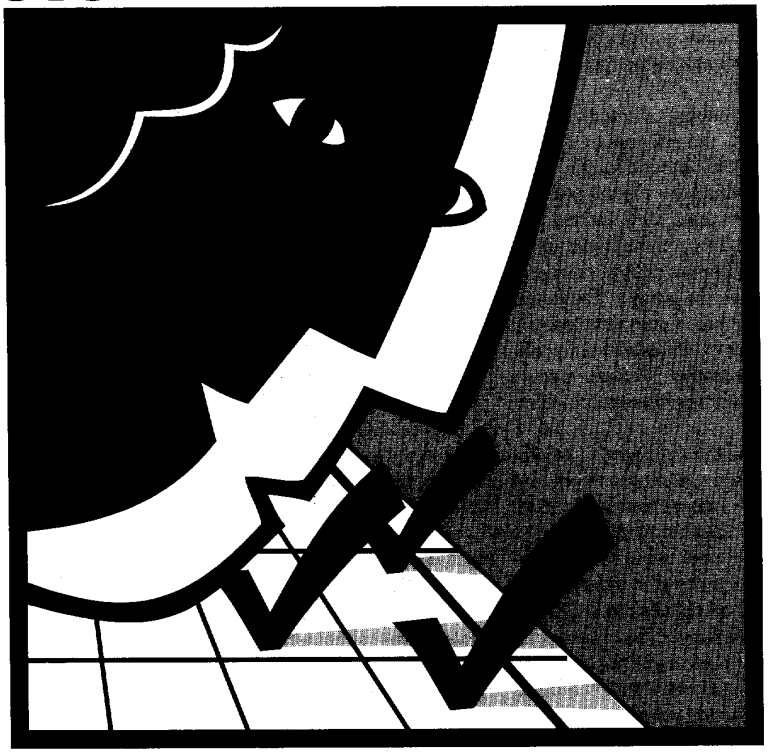
D. Mutchler

FEATURE

PLANNING AND CERTIFYING SOFTWARE SYSTEM RELIABILITY

Developers often view software reliability as unapproachable. But an increased understanding of planning and certification activities can help in selecting models and manipulating them in reliability analysis.

J.H. POORE
University of Tennessee
HARLAN D. MILLS
Software Engineering Technology
DAVID MUTCHLER
University of Tennessee



Hardware-reliability engineers have long been able to design a hardware system to a target reliability by determining the reliability of system components or allocating reliability budgets to component developers. Software engineers can also design for reliability, but they seldom do because they view the process as too complex or not applicable to software. With the growing emphasis on reuse, however, and the need to demonstrate that the software to be reused is indeed reliable, they can no longer afford to shy away from reliability planning.

To make reliability planning and certification more accessible, we developed an approach based on the use of three mathematical models — the sampling,[1,2] component,[1,3,4] and certification[5] models — although other models may be equally suitable. This approach, which helps reduce reliability analysis to a problem that can be evaluated and manipulated through a series of spreadsheets, addresses the three reasons we believe most developers avoid these activities:

- They do not differentiate between planning and certifying and the tasks associated with each.
- They find it difficult to chose from among the many available reliability models.
- They find the mathematical models difficult to manipulate for what-if analyses.

0740-7459/93/0100/0088/$03.00 © IEEE

The first reason is the result of trying to apply concepts that are relatively new to software engineering, the second stems from a lack of consensus about reliability itself, and the third may be caused by a lack of tools for manipulating model results. As part of our experiment, we developed a system to handle all the calculations in a spreadsheet format.

Our approach has three aspects, which address these reasons for avoidance. First, we believe that developers need to thoroughly understand the tasks involved in planning and certification. Second, armed with that insight, they can choose a reliability model similar to the ones we describe here. Finally, they can develop a spreadsheet system similar to our own to manipulate model results in enlightening what-if analyses.

This approach was motivated by our interest in applying the Cleanroom software-engineering method in environments that require extensive code reuse. Two models for certification, including the one we used in our experiment, are part of accepted Cleanroom practice.[5-8]

WHAT IS RELIABILITY?

Like hardware reliability, software reliability is based on modes of failure. Hardware modes of failure — wear, design flaws, and unintentional environmental phenomena — are more tangible because hardware is a physical entity. In fact, it is this very physical quality that prompts hardware designers to assume that hardware cannot be perfect. Ironically, the same designers often assign perfect reliability to a software component because it can't "wear out," for example.

But software does have a mode of failure, which is based on the assumption that design and development are not perfect processes. The mistakes made during these processes manifest as faults in the code, which are revealed as inputs are processed. That is, a failure occurs when the software does not perform according to specification for an input history.

Thus, like hardware, software is deemed reliable in relation to its use and intended performance. Use, the basic unit of reliability measurement, can be a keystroke, work session, transaction, completed telephone call, or any other unit of performance appropriate to the service the software is expected to perform.

To quantify reliability in a meaningful way, software use must be modeled as a random process in which a use is selected according to some probability distribution, or *use distribution*. Reliability then becomes the probability that the software will perform according to specification for a randomly selected use. When the software fails to meet specification during use, a failure occurs.

Reliability can be a useful metric. You can use it to help guide software development. You can also use it to assess a program's fitness for use by conducting experiments to establish empirical evidence of quality.

These dual uses of reliability have slightly different definitions. Reliability as a function of time, perhaps the more traditional definition, addresses the design of software that will operate according to specification for a period of time. But you can also use a simpler definition — reliability is the probability that a randomly chosen use (test case) will be processed correctly.

YOU CAN DEFINE RELIABILITY AS THE PROBABILITY THAT A USE WILL BE PROCESSED CORRECTLY.

In most instances, we use this simpler definition because it is well suited to the idea of conducting experiments to establish empirical evidence of quality. It also proves to be a very conservative notion of reliability, well suited to dealing with the reuse of software for which little may be known about the process of its development but much may be known about its operational history.

In the time-based definition of reliability, the choice of time as the random variable is based on the idea that randomly selected uses (according to a use distribution) will cause paths through the program to execute randomly; consequently, as operating time increases, the probability of encountering a fault in the code increases. Time can be execution time, calendar time, number of instructions executed, number of input cases, or number of uses, to name the most common interpretations. These conditions represent a constant failure rate.

Using the definition that reliability is the probability that the software will give the correct result for a single, randomly chosen (according to the use distribution) use, then the mean time to failure is the average number of uses between failures. MTTF and reliability can be related mathematically in the models.

Some models deal with system reliability (in all uses of "system," we are referring to a software system) as a function of the modules or units that comprise the system. Other models estimate or predict system reliability without regard to what the system comprises.

PLANNING VERSUS CERTIFICATION

An understanding of reliability planning and certification is based on the progression of four basic ideas:

- Systems are composed of components.
- Component reliability can be measured.
- System reliability can be calculated from component information.
- System certification may be based on a different model from that used in reliability planning.

Systems are composed of components. We define a system as a collection of programs and system files such that the system files are accessed and altered only by the programs in the collection. This definition is not intended to rule out systems, but is given to establish the boundary of responsibility. Clearly, if files are altered by agents outside the system, we cannot vouch for the consequences. These programs and system files are what we mean by components. Components may be systems, modules, packages, programs, or files.

An additional constraint on the system, to satisfy a technical assumption of one of the mathematical models, is that it must be proper. A proper system must have a single entry and a single exit, and for each executable component there must be a path from the entrance through the component to the exit.

If a system is being planned that will comprise new and reused components, in the final analysis you will either use or not use a specific component. You can make this binary decision on the basis of somewhat crude information. In particular, you must know or conjecture how the component will interplay with the rest of the system and what its reliability will be during this interplay, so that you can assess the effect on the reliability of the system as a whole. The quality of component information must be good enough to support a determination that it is the best among alternatives, including a newly developed component.

In reliability planning, you must model the interaction of all system components — an inexact activity. While the error in this process is bearable for reaching the binary decision to use or not use a given component, it need not be accepted in calculating final system reliability. For this reason, we recommend that you independently certify the completed system on the basis of statistical use testing.

In this type of testing, the testing process constitutes a statistical experiment. It consists of processing a random sample of test cases selected according to intended system use to present empirical evidence that the system performs correctly. The statistical qualities of the testing process let you make scientific statements about the predicted reliability of the system, in essence certifying it.

Component reliability. The quality of information about the component must be good enough to support your decision to use or not use it. There are many sources of component information, and even a crude form of any of these may be enough

CALCULATING THE METRICS

The mathematics of the sampling, component, and certification models are based on the relationship of reliability and the mean time to failure. MTTF is the average number of uses between failures. Time is measured as the number of uses (or test cases). Reliability is related to MTTF by

$$\text{MTTF} = \frac{1}{1 - \text{reliability}}$$

If time is interpreted in any other way, the relationship is

$$\text{MTTF} = \frac{L}{1 - \text{reliability}}$$

where L is the average number of time units per use. By defining L, you can choose an arbitrary time unit or convert various time units to a common one and move easily between MTTF and reliability.

Sampling model. If $100c$ is the percentage of confidence you want in the experiment, the number of test cases that must run without a failure to report a reliability of r is

$$\text{Number of test cases} = \left\lceil \frac{\log(1-c)}{\log(r)} \right\rceil$$

The $100c$-percent confidence means that if you adopt this testing method and test software frequently, it is almost certain that the claims would be true at least $100c$ percent of the time. This is not the same as saying that the claim is right with probability c; there is no probability involved in the claim itself because a claim is either right or wrong. As Table A shows, you can obtain additional confidence without greatly increasing the number of tests; additional reliability, however, does require large increases in the number of tests.

This formula uses a zero-failures certification method. The software is tested on m random test cases (chosen according to the use distribution) and is certified if no failures occur. The number of tests is m, the minimal number to ensure that unreliable software is not certified too frequently.

TABLE A
ZERO FAILURES

Reliability	Confidence level (percent)			
	90	95	99	99.9
0.9	22	29	44	66
0.95	45	59	90	135
0.99	230	299	459	688
0.999	2302	2995	4603	6905

Another approach is to test m_k random test cases, where k is any nonnegative integer, and certify if at most k failures are found. m_k is chosen just as m was; it is the minimal number of tests that ensures that unreliable software is not certified too frequently. For example, to allow up to two failures, and certify with 90-percent confidence that r is at least 0.99, you must run 531 test cases.

You can compute m_k numerically for any k and for the other parameters as follows: m_k must satisfy

$$\Pr(k \text{ or fewer failures in } m_k \text{ trials} \mid \text{actual reliability} < r) \le 1 - c$$

where Pr is probability. The smallest m that satisfies this requirement is the smallest m such that

$$\sum_{j=m-k}^{m} \binom{m}{j} r^j (1-r)^{m-j} \le (1-c)$$

You can then solve for m numerically.

The major disadvantage of this k-failures method over the

to make that binary decision.

♦ *Developer's records.* If the component developer has by reputation or contract asserted the component's reliability, you may be able to use this assertion.

♦ *Development method.* If a certain development method was used and that method includes a reliability standard, you may be able to know the reliability by knowing the method.

♦ *Proof of correctness.* If the component has been verified by a mathematical proof of correctness, you may be able to attribute a high degree of reliability to it.[9]

♦ *Field performance.* If records of field use and failures are available, you can estimate reliability from the field data.

♦ *Statistical experiment.* You can always conduct a specific statistical testing experiment using the sampling model as described in the box on pp. 90-92.

System reliability. To calculate system reliability using the component model as shown in the box on pp. 90-92, you will need both estimates of component reliabilities and the structure of component interactions. The structure and relative frequency of these interactions is determined by transition probabilities — the probability of transition from one component to another. You must estimate transition probabilities on the basis of design and intended use.

Given the system structure, component reliabilities, and transition probabilities, you can perform calculations in an enlightening what-if analysis. What if a certain component were more or less reliable? How would that affect system reliability? What if a certain component were perfect, with a reliability of 1.0?

You can also estimate the sensitivity of the total system to each component through what-if analysis. You can calculate system reliability from component information, or you can stipulate a system reliability and calculate an allocated reliability to the components. The what-if analysis gives insight to the decision to use or not

zero-failures method is that it requires more tests. A second disadvantage is that it certifies software with known errors; if the errors are corrected, the certification is no longer valid because the test was conducted on the software before the changes. The statement must apply to the software on which the experiment was conducted.

The advantage of the k-failures method over the zero-failures method is that the k-failures method will deny certification less often. If the software has an MTTF of 500 with a goal MTTF of 100, the zero-failures method will deny certification more than a third of the time; the k-failures (two-failures) method will deny certification less than 10 percent of the time.

Thus, the sampling model can produce certification errors in two ways: First, it can certify software that is, in fact, unreliable. Second, it can deny certification to software that is, in fact, reliable. The zero-failures method lets you control the likelihood of errors of the first kind by setting the confidence level as desired. You can use the k-failures method to control the likelihood of errors of the second kind as well by setting k large enough. However, you pay a price for this extra control in more tests.

Component model. Calculations in the planning spreadsheets of the CRM are based on a Markov model. Here we present only the formulas used in the calculations.

Consider a system that consists of n components, 1 to n, with component 1 being the single entry point to the system. Let r_i denote the probability that when component i is being executed, the system continues to another component without an error. Thus, $(1 - r_i)$ is the probability of a failure (fatal error) during component i's execution. For $i = 1, ..., n$ and for $j = 1, ..., n, T$, where component T is interpreted as the successful termination of the system, define p_{ij} by

$r_i p_{ij}$ = probability that component j will be executed next if component i is currently being executed

When i is fixed, p_{ij}'s sum to r_i. The model makes the Markovian assumption that transfer of control (to another component, to successful termination, or to unsuccessful termination) is conditionally independent of execution history.

Calculating system reliabilities and component sensitivities requires the following: For i and j from 1 to n, define n by n matrices $\mathbf{G}$ and $\mathbf{H}$ by $G_{ij} = p_{ij}$ and $H_{ij} = r_i p_{ij}$. Perform two matrix inversions to obtain $\mathbf{S} = (\mathbf{I} - \mathbf{G})^{-1}$ and $\mathbf{T} = (\mathbf{I} - \mathbf{H})^{-1}$. For i from 1 to n, define column vectors f and R by $f_i = r_i p_{iT}$ and $R = Tf$.

Because component 1 is the sole entry point to the system, system reliability R is R_1. The sensitivity of system reliability to the reliability of component i is (by definition)

$$\frac{\partial R_1}{\partial r_i}$$

and is given by

$$\frac{\partial R_1}{\partial r_i} = R_i \frac{T_{1i}}{r_i}$$

Suppose you set a target system reliability R_{tgt}. To meet it, the r_i needed for component i, assuming all other components are reliable, is

$$r_i = \left[\frac{(c_i(1 - R_{tgt})}{R_{tgt} - a_i} + 1 \right]^{-1}$$

where

$$c_i = \frac{1}{S_{ii}} \quad \text{and} \quad a_i = 1 - \left(\frac{S_{1i}}{S_{ii}} \right)$$

You can use this formula to allocate reliabilities to the components. However, because allocated reliabilities yield a system reliability less than the target reliability, you must increase each allo-

to use a component. Coupled with the sampling model, it can show the scale and scope of effort needed to demonstrate the required levels of component reliabilities. Finally, it can shed light on the reasonableness of building a system to desired levels of reliability on the basis of reusing a specific collection of components.

System certification. Certification of a completed system is based on a different model from that used in planning, and the model criteria are substantially different. Developers must take responsibility for the completed system as an operational entity, without regard for the parts and why or how they are used. A good certification model must focus on the performance of the system in statistical use testing and in field use. Exponential growth in MTTF is the goal in certification.

System certification should be based on generating or selecting inputs and input histories according to the system's intended use. To the extent that files are a part of the system, you must achieve representative steady states in these files through input histories. You can conduct a statistical experiment to collect data points on performance and use a reliability model to analyze them and predict system reliability.

RELIABILITY MODELS

To illustrate our approach, we used the sampling, component, and certification models, which have been useful in practice.[10] Each model has certain mathematical properties, described in the box on pp. 90-92. You can apply the models to your development process to the extent that your process is characterized by these properties. Because many aspects of planning and certifying system reliability do not require an exact analysis, the models can be meaningful to your process even if they only partially characterize it.

Sampling model. This model is useful for

cated reliability somewhat to meet the target.

Certification model. In the certification model, $MTTF_k$ denotes the MTTF of version k of the system. Suppose that for all k, $MTTF_k = (B)(MTTF_{k-1})$ where B is some constant. Then $MTTF_k = AB^k$ where $A = MTTF_0$.

The certification model has three independent aspects:

1. The parametric form of AB^k, which is used to estimate the MTTF of version k.
2. The corrected-log least-squares technique, which is used to compute A and B from the data points.
3. The technique for obtaining the data points.

You can estimate A and B directly from statistical data using either maximum-likelihood or least-squares techniques. However, the corrected-log least-squares technique is not only a better estimator but also a simpler computation.

This technique starts with the original equation $MTTF_k = AB^k$. You then take the logarithms of both sides to get

$$\log(MTTF_k) = \log A + k(\log B)$$

By letting a equal $\log A$ and b equal $\log B$, you can then rewrite the equation as

$$\log(MTTF_k) = a + kb$$

You can compute the estimates for a and b using standard linear regression. This minimizes the sum of the squares of the differences between the logarithms of $MTTF_0,\ldots,MTTF_{n-1}$ and the estimates for $a + 0b,\ldots, a + (n-1)b$. To do this linear regression, take partial derivatives with respect to a and b, set them to zero, and solve the two linear equations that result.

From the previous step, you have estimates α for $\log A$ and β for $\log B$. Tentative estimates for A and B are e^{α} and e^{β}. The estimate of the MTTF for version d is $e^{\alpha}(e^{\beta})^d$. However, this estimate is biased; its mean is not equal to the true value of AB^d. Using the model's assumption that the MTTF (measured by sampling) for version k is an independent random variable exponentially distributed with mean AB^k, you can compute an unbiased estimate.

To get confidence intervals for various aspects of the curve from a least-squares linear regression, assume that the residuals are independent and normally distributed with a mean of zero and common variance of σ^2. You can estimate this variance and then compute a confidence interval for the log-transformed data.

The power of a model is the ratio of the released product's predicted MTTF to the number of tests. To estimate the power of the certification model, assume that the estimates of A and B are exact and that all the data points lie exactly on the curve. Suppose that version n is released after versions 0 through $n-1$ have been tested. The estimated MTTF of version n is then AB^n; the number of tests conducted is

$$\sum_{k=0}^{n-1} AB^k = \frac{A(B^n - 1)}{(B - 1)}$$

so the power of the certification model is the ratio of the released product's predicted MTTF to the number of tests:

$$AB^n : \frac{A(B^n - 1)}{(B - 1)}$$

which is equal to

$$\left[\frac{(B-1)(B^n)}{(B^n - 1)}\right] \approx (B-1)$$

for B not near 1. As we described earlier, the power of the sampling model is about 1:2 at 90-percent confidence levels and less at higher confidence levels. Thus, if you do N tests under the certification model, you expect to do roughly $2N(B-1)$ tests to achieve the same level of MTTF certification under the sampling model.

estimating the reliability of an existing component as an entity without regard to its composition.

In a simple sampling experiment, you must draw a number of test cases from the use distribution — which mirrors actual software use — run these test cases, and record the number processed correctly. You can then report that the software has the estimated reliability at a set confidence level. The appeal of the sampling model is that you can make a quantitative claim about software quality subject to only two sources of error: sampling error, which you can control through a set confidence level, and error in the use distribution.

The sampling model has drawbacks as well. It may be difficult to model use distribution. Testing may be expensive. If you need a 90-percent or better confidence level, the number of tests required to demonstrate a certain MTTF is more than twice that MTTF. Certification will often be denied because of bad luck if the true MTTF is close to the certified MTTF. For example, if the certified MTTF is 100 and the true MTTF is 200, then certification at 90-percent confidence will be denied merely because of chance more than two-thirds of the time! Even if the true MTTF is 1,000, certification will still be denied more than 20 percent of the time.

Because the sampling model is conservative in its estimates, certification of unreliable software is rare. But this very conservatism coupled with the economic pressure to limit the number of tests might deny certification to much software that is in fact quite reliable. Fortunately, there are mathematical ways to control the likelihood that reliable software is denied certification, but for the most part you are better off certifying the system through statistical testing.

Component model. The component model is useful for estimating how the reliability of components — both new and used — can affect system reliability. You can calculate system reliability from information about the components, or you can stipulate a system reliability and calculate an allocated reliability to the components.

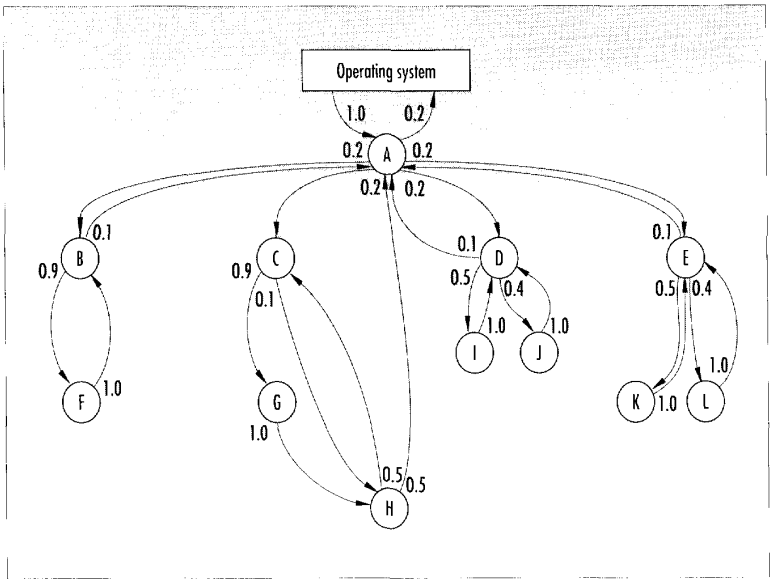


Figure 1. A software system as a network of components. Each directed arc, labeled with its transition probability, indicates that control passes from one component to another in the direction of the arrow.

The component model can use existing component data to estimate system reliability with no additional testing. Also, the planner can use the component model for what-if analyses. The model's simplicity is both a strength and a weakness. It makes the model easy to use, but reliability estimates may be inaccurate. For the qualitative decisions involved in planning system reliability, this inaccuracy is acceptable. For certifying the complete system, however, a more accurate model is required.

Certification model. Certification is essentially a statistical experiment to collect data points, which are the times between failures on successive system versions or engineering-change levels. A curve is fitted to this data, which is useful in monitoring progress during development and predicting the MTTF of the released product. The model assumes that MTTF grows exponentially over successive versions of a system.

The certification model is more powerful than the sampling model, in that far fewer test cases are required to certify the system at a set reliability. It is also usually more accurate than the component model, but will yield poor estimates if the curve being fitted does not, in fact, exhibit exponential growth. A promising alternative to the certification model is the Markov Testing Model.[7,8]

TABLE 1
TRANSITION PROBABILITIES FOR THE SYSTEM IN FIGURE 1

From module	To module	Transition probability
OS	A	1.0000
A	B	0.2000
A	C	0.2000
A	D	0.2000
A	E	0.2000
B	A	0.1000
B	F	0.9000
C	G	0.9000
C	H	0.1000
D	A	0.1000
D	I	0.5000
D	J	0.4000
E	A	0.1000
E	K	0.5000
E	L	0.4000
F	B	1.0000
G	H	1.0000
H	A	0.5000
H	C	0.5000
I	D	1.0000
J	D	1.0000
K	E	1.0000
L	E	1.0000
A	OS	0.2000

APPLYING THE MODELS

Figure 1 shows a system represented by a network of components. Each directed arc indicates that control passes from one component to another in the direction of the arrow. There is a single point of entry into the system from the operating system and single point of return to the operating system and, for each node, there is a path from entry to exit that passes through that node — the definition of a proper system.

As we described earlier, you must estimate component-transition probabilities. In a well-designed system, communication among components is limited. Table 1 shows the transition probabilities for the network in Figure 1, listing the probability associated with each arc. This information may also be represented as a matrix with a zero probability assigned to impossible transitions.

Using just the transition probabilities, you can learn some interesting information about how a component can affect system reliability. Managers can use this information to rethink the system's architecture and to plan resource allocation, for example.

YOU CAN TELL A LOT ABOUT SYSTEM RELIABILITY BY LOOKING AT TRANSITION PROBABILITIES.

To assist in manipulating the necessary calculations for planning and certification, we designed the Cleanroom Reliability Manager.[10] Table 2 shows the information the CRM will display given only transition probabilities as input. The CRM uses certain defaults. For each component, the default reliability is 1.0, and the default confidence level is 90 percent. The defaults are displayed until the planner enters new information.

At the top of the table are system reliability, system MTTF, target reliability, and the test-scale factor. The CRM calculates system reliability from transition probabilities and component reliabilities. Since each component has a default reliability of 1.0, the entire system has a 1.0 reliability.

If the reliability is perfect, the MTTF is undefined (because there are no failures). Target reliability is undefined until the reliability planner enters it into the CRM. Test-scale factor lets you equate one study to another when that is possible simply by scaling all data by a constant factor; it will remain 1.0 throughout this example.

The columns reliability, confidence, test cases, and MTTF are related mathematically, and you should interpret them collectively. For example, you would need an infinite number of test cases to demonstrate a reliability of 1.0, and you would have no failures on which to base an MTTF. Test cases tells you how many test cases you must run without a failure to demonstrate the given reliability at the given level of confidence. You can enter either reliability or MTTF, whichever is more directly available, and the CRM will calculate the other.

Likewise, you can enter the number of possible or affordable test cases, and the CRM will show the reliability that such an experiment would demonstrate at the chosen confidence level. If you change the confidence level, the CRM will recalculate the number of test cases. If you enter a

TABLE 2
CRM DISPLAY GIVEN ONLY TRANSITION PROBABILITIES

System reliability: 1.0000 — Target reliability: Undefined
System MTTF: Undefined — Test-scale factor: 1.000

Module name	Reliability	Confidence	Test cases	MTTF	Sensitivity	Allocated reliability
Enter_system	1.0000	0.900	Infinite	Undefined	1.00	Undefined
A	1.0000	0.900	Infinite	Undefined	5.00	Undefined
B	1.0000	0.900	Infinite	Undefined	10.00	Undefined
C	1.0000	0.900	Infinite	Undefined	2.00	Undefined
D	1.0000	0.900	Infinite	Undefined	10.00	Undefined
E	1.0000	0.900	Infinite	Undefined	10.00	Undefined
F	1.0000	0.900	Infinite	Undefined	9.00	Undefined
G	1.0000	0.900	Infinite	Undefined	1.80	Undefined
H	1.0000	0.900	Infinite	Undefined	2.00	Undefined
I	1.0000	0.900	Infinite	Undefined	5.00	Undefined
J	1.0000	0.900	Infinite	Undefined	4.00	Undefined
K	1.0000	0.900	Infinite	Undefined	5.00	Undefined
L	1.0000	0.900	Infinite	Undefined	4.00	Undefined

TABLE 3
CRM DISPLAY GIVEN A TARGET RELIABILITY OF 0.999

System reliability: 1.0000 — Target reliability: (3)00
System MTTF: Undefined — Test-scale factor: 1.000

Module name	Reliability	Confidence	Test cases	MTTF	Sensitivity	Allocated reliability
Enter_system	1.0000	0.900	Infinite	Undefined	1.00	(3)00
A	1.0000	0.900	Infinite	Undefined	5.00	(3)80
B	1.0000	0.900	Infinite	Undefined	10.00	(4)00
C	1.0000	0.900	Infinite	Undefined	2.00	(3)50
D	1.0000	0.900	Infinite	Undefined	10.00	(4)00
E	1.0000	0.900	Infinite	Undefined	10.00	(4)00
F	1.0000	0.900	Infinite	Undefined	9.00	(3)88
G	1.0000	0.900	Infinite	Undefined	1.80	(3)44
H	1.0000	0.900	Infinite	Undefined	2.00	(3)50
I	1.0000	0.900	Infinite	Undefined	5.00	(3)80
J	1.0000	0.900	Infinite	Undefined	4.00	(3)75
K	1.0000	0.900	Infinite	Undefined	5.00	(3)80
L	1.0000	0.900	Infinite	Undefined	4.00	(3)75

TABLE 4
CRM DISPLAY GIVEN COMPONENT RELIABILITIES

System reliability: (1)64 — Target reliability: (3)00
System MTTF: 28.09 — Test-scale factor: 1.000

Module name	Reliability	Confidence	Test cases	MTTF	Sensitivity	Allocated reliability
Enter_system	1.0000	0.900	Infinite	Undefined	0.96	(3)00
A	(3)00	0.900	2301	1000.00	4.66	(3)80
B	(3)00	0.900	2301	1000.00	9.13	(4)00
C	(3)00	0.900	2301	1000.00	1.86	(3)50
D	(3)00	0.900	2301	1000.00	9.13	(4)00
E	(3)00	0.900	2301	1000.00	9.14	(4)00
F	(5)00	0.900	230257	100000.00	8.20	(3)88
G	(5)00	0.900	230257	100000.00	1.67	(3)44
H	(5)00	0.900	230257	100000.00	1.85	(3)50
I	(5)00	0.900	230257	100000.00	4.56	(3)80
J	1.0000	0.900	Infinite	Undefined	3.65	(3)75
K	1.0000	0.900	Infinite	Undefined	4.56	(3)80
L	1.0000	0.900	Infinite	Undefined	3.65	(3)75

change for any one of the four items, it calculates and displays an appropriate change to one of the other three.

The CRM calculates sensitivity (sixth column) from the transition matrix. In our example, components B, D, and E have the greatest effect on the total system. Thus, system reliability is twice as sensitive to these components as it is to A, I, and K. The component's sensitivity shows its relative importance to system reliability, rather than any absolute information.

Allocated reliability (last column) is the reliability allocated or budgeted to each component and is calculated whenever the target reliability is changed. The allocated reliability for each component is based on the target reliability for the entire system and the sensitivity of the system to the component. If some components have higher reliabilities than are budgeted to them, the demand on other components is lower.

Thus, the CRM provides a good deal of flexibility in what-if analysis. If you change the target reliability, you will cause a reliability budget to be allocated to each component. You can also change any one of reliability, confidence, test cases, or MTTF for an individual component. By changing individual component information, you will cause a change in the calculated system reliability and system MTTF. Finally, if you change the transition probabilities assigned to network arcs, you will change the sensitivities — the relative

TABLE 5
CRM DISPLAY GIVEN INCREASED COMPONENT RELIABILITIES

System reliability: (1)89 Target reliability: (3)00
System MTTF: 99.25 Test-scale factor: 1.000

Module name	Reliability	Confidence	Test cases	MTTF	Sensitivity	Allocated reliability
Enter_system	1.0000	0.900	Infinite	Undefined	0.99	(3)00
A	(3)00	0.900	2301	1000.00	4.91	(3)80
B	(4)00	0.900	23025	10000.00	9.78	(4)00
C	(3)00	0.900	2301	1000.00	1.96	(3)50
D	(4)00	0.900	23025	10000.00	9.78	(4)00
E	(4)00	0.900	23025	10000.00	9.78	(4)00
F	(5)00	0.900	230257	100000.00	8.80	(3)88
G	(5)00	0.900	230257	100000.00	1.76	(3)44
H	(5)00	0.900	230257	100000.00	1.95	(3)50
I	(5)00	0.900	230257	100000.00	4.89	(3)80
J	1.0000	0.900	Infinite	Undefined	3.91	(3)75
K	1.0000	0.900	Infinite	Undefined	4.89	(3)80
L	1.0000	0.900	Infinite	Undefined	3.91	(3)75

TABLE 6
REVISED TRANSITION PROBABILITIES FOR THE SYSTEM IN FIGURE 1

From module	To module	Transition probability
OS	A	1.0000
A	B	0.0500
A	C	0.4000
A	D	0.4000
A	E	0.1000
B	A	0.1000
B	F	0.9000
C	G	0.9000
C	H	0.1000
D	A	0.1000
D	I	0.5000
D	J	0.4000
E	A	0.1000
E	K	0.5000
E	L	0.4000
F	B	1.0000
G	H	1.0000
H	A	0.5000
H	C	0.5000
I	D	1.0000
J	D	1.0000
K	E	1.0000
L	E	1.0000
A	OS	0.0500

contributions of each component to the system's reliability.

As an example, suppose you are certifying a system at 0.999 reliability, which in the long run means we can expect one failure in 1,000 uses. Using the CRM, you set the target reliability to the goal of 0.999 to produce the display in Table 3. To abbreviate reliability figures and to draw attention to the number of nines, the CRM displays 0.999 as (3)00, 0.99980 as (3)80, 0.99999 as (5)00, and so on.

Using the target reliability of (3)00 (0.999) and the known sensitivities, the CRM allocates a reliability budget to each component. If component reliabilities are set at the allocated levels, the system reliability will be slightly less than the target reliability because of the nature of the model.

Adjusting component reliabilities. To illustrate the relationships just described, assume that you have the following information on components A through L:

- *Components A, B, C, D, E.* New, to be programmed and certified at 0.999 reliability.
- *Components F, G and I.* Existing in a library with 0.99999 field-use reliability. Performance records are sufficiently well-established to justify this reliability claim.
- *Component H.* New, to be programmed and certified at 0.99999 reliability.
- *Components J and L.* Numerical-function library packages with such an extensive field-use record that we are justified in asserting a reliability of 1.0. (Asserting a reliability of 1.0 does not mean you can demonstrate or even believe that, however. The assertion is merely a way of taking a component with exceptionally high reliability out of play.)
- *Component K.* Existing in a library with such an extensive field-use record that we are justified in asserting a reliability of 1.0.

If you enter this component information into the model, you get the display in Table 4. The table shows that the reliability entries for components A through I have changed, which caused the entries for the associated test cases and MTTF entries to change. Now, to demonstrate a reliability of 0.999 ((3)00) and to have 90-percent confidence in the demonstration, you must run 2,301 test cases without a failure. Moreover, you might also require a similar demonstration for each new component (A through E). (Cleanroom takes a different and more efficient approach, as we will show later.) This table shows that, under our definitions, a reliability of 0.999 corresponds to an MTTF of 1,000 uses.

Entries for components *F*, *G*, and *I* — which have well-established field-use records justifying a reliability of 0.99999 ((5)00) — show the value of carefully documented field performance. To demonstrate a 90-percent confidence level in this reliability, you would have to run 230,257 randomly selected test cases without a failure!

The table shows a system reliability of 0.964 ((1)64), the same as an MTTF of

TABLE 7
CRM DISPLAY GIVEN REVISED TRANSITION PROBABILITIES

System reliability: (1)54 Target reliability: (3)00
System MTTF: 21.92 Test-scale factor: 1.000

Module name	Reliability	Confidence	Test cases	MTTF	Sensitivity	Allocated reliability
Enter_system	1.0000	0.900	Infinite	Undefined	0.95	(3)00
A	(3)00	0.900	2301	1000.00	18.25	(4)50
B	(4)00	0.900	23025	10000.00	9.09	(4)00
C	(3)00	0.900	2301	1000.00	14.54	(4)37
D	(4)00	0.900	23025	10000.00	72.73	(4)87
E	(4)00	0.900	23025	10000.00	18.18	(4)50
F	(5)00	0.900	230257	100000.00	8.18	(3)88
G	(5)00	0.900	230257	100000.00	13.08	(4)30
H	(5)00	0.900	230257	100000.00	14.53	(4)37
I	(5)00	0.900	230257	100000.00	36.36	(4)75
J	1.0000	0.900	Infinite	Undefined	29.09	(4)68
K	1.0000	0.900	Infinite	Undefined	9.09	(4)00
L	1.0000	0.900	Infinite	Undefined	7.27	(3)87

TABLE 8
CRM DISPLAY GIVEN VERY HIGH COMPONENT RELIABILITIES

System reliability: (2)68 Target reliability: (3)00
System MTTF: 318.18 Test-scale factor: 1.000

Module name	Reliability	Confidence	Test cases	MTTF	Sensitivity	Allocated reliability
Enter_system	1.0000	0.900	Infinite	Undefined	1.00	(3)00
A	(5)00	0.900	230257	100000.00	19.87	(4)50
B	(4)00	0.900	23025	10000.00	9.92	(4)00
C	(5)00	0.900	230257	100000.00	15.90	(4)37
D	(5)00	0.900	230257	100000.00	79.48	(4)87
E	(5)00	0.900	230257	100000.00	19.87	(4)50
F	(5)00	0.900	230257	100000.00	8.92	(3)88
G	(5)00	0.900	230257	100000.00	14.31	(4)30
H	(5)00	0.900	230257	100000.00	15.90	(4)37
I	(5)00	0.900	230257	100000.00	39.74	(4)75
J	1.0000	0.900	Infinite	Undefined	31.79	(4)68
K	1.0000	0.900	Infinite	Undefined	9.94	(4)00
L	1.0000	0.900	Infinite	Undefined	7.95	(3)87

28.09 uses, which the CRM calculated from the network relationships and component information. Because the allocated-reliability column shows that some components with high sensitivities (B, D, and E) have lower reliabilities than allocated, it should not be surprising that system reliability is less than target reliability.

New components B, D, and E (components A and C, although new, have lower sensitivities) are actually more critical to system reliability than the reused components with well-established records of highly reliable performance. In Table 5, the reliabilities for components B, D and E have been increased, and, as you would expect, system reliability has also increased.

Adjusting system structure. The most fundamental change a planner can make is to revise the network that describes component interaction. The most radical change is to remove or add a node and associated arcs, which corresponds to a major architectural change. A less radical change is to remove or add arcs — change the transition probabilities without changing the components themselves — after carefully studying and analyzing the system's intended use. Table 6 shows transition probabilities revised from those in Table 1. The reliability information for Table 6 is shown in Table 7 (which contrasts with Table 5).

TABLE 9
CERTIFICATION OF INCREMENTS

Version number	Observed MTTF	Predicted reliability	Predicted MTTF	$MTTF_0$	Improvement factor
		Increment 1			
0	1.00	—	—	—	—
1	6.00	—	—	—	—
2	1.00	(0)23	0.81	2.09	0.59
3	16.00	(0)77	4.38	1.09	1.36
4	560.00	(2)57	232.62	0.37	3.60
		Increment 2			
0	1.00	—	—	—	—
1	1.00	—	—	—	—
2	1.75	(0)37	1.60	1.02	0.99
3	11.00	(0)81	5.38	0.79	1.55
4	37.00	(1)63	27.64	0.55	2.20
5	49.00	(1)88	86.47	0.58	2.31
6	200.00	(2)68	316.52	0.51	2.53
		Increment 3			
0	1.00	—	—	—	—
1	1.00	—	—	—	—
2	15.00	(1)23	13.04	0.64	2.31
3	116.00	(2)34	153.84	0.42	4.29
		Increment 4			
0	2.00	—	—	—	—
1	1.50	—	—	—	—
2	38.50	(1)78	46.90	0.57	3.97
3	28.00	(2)18	122.40	0.81	3.55
4	12.00	(1)87	80.95	1.55	2.26
5	29.00	(2)05	106.30	1.98	1.99
6	87.00	(2)53	216.24	2.10	1.98
7	200.00	(2)78	472.58	2.13	2.00

A change like this will definitely affect system reliability and component sensitivities, as a comparison of Tables 5 and 7 shows. Particularly important is that the relative values within each table are different, which may cause the planner to reallocate development resources. In Table 5, components B, D, and E affect system reliability the most; in Table 7, components D, I, and J have the greatest effect.

Also important is that the components in Table 7 have much higher sensitivities than those in Table 5. This difference implies that changes in component reliabilities will affect system reliability more under the revised transition probabilities (as in Table 7) than under the original ones (as in Table 5). Table 8 shows what happens when component reliabilities are pushed very high.

Cleanroom certification. Continuing with the assumed component information, if you are going to enter the new components individually into a repository or library, you must certify each one individually. In this illustration of Cleanroom certification, we assume that new components do not have an existence or applicability outside the new system. The task is to certify the completed system, not its components, although you should know a great deal by now about how each component affects system reliability.

Cleanroom certification is based on the Cleanroom approach to management and development, which means that system development will be segmented into increments, and each increment will be certified to the target level. Since the increments are cumulative, component interactions are fully certified in the final increment.

The data in Table 9 is based on an actual project that resulted in approximately 24,000 lines of Ada, of which half was newly developed code and half was reused from a library.[10] The formulas used in the CRM to calculate the data in Table 9 are given in the box on pp. 90-92. The increments, with components given in the order they were implemented, are

- *Increment 1.* Components A, B, and F
- *Increment 2.* Components A, B, F, C, G, H
- *Increment 3.* Components A, B, F, C, G, H, D, I, J
- *Increment 4.* Components A, B, F, C, G, H, D, I, J, E, K, L

Version number (first column in the table) indicates the engineering-change level. Generally, an immediate-repair policy was followed with respect to failures in testing and changes to the code. Whenever it was clear that successive failures were independent, testing continued without code changes and recompilation. When code changes were made, each recompilation resulted in a new version of the system and each new version fixed one or more faults from the previous version. Because there is no failure in the last version, it would be overly conservative to enter just the number of tests run. Thus, the certification model's criterion for stopping testing is based on the last entry for each increment being double the actual number of test cases that ran without failure.

$MTTF_0$ (fifth column) is the estimated MTTF of the software's initial version. Improvement factor (last column) is the estimated factor by which each successive version is an improvement over its predecessor.

The number of test cases required for certification with this model depends not only on the number of failures but also on when they are observed.

To certify increment 1, for example, we had to run 300 randomly generated test cases to reach a predicted reliability of 0.9957 ((2)57). We detected six failures and corrected the faults during certification.

In certifying increment 2, we brought the cumulative number of test cases to 556 and the cumulative error count to 19. We stopped testing when the certification model predicted a 0.9968 ((2)68) reliability.

In certifying increment 3, we required 656 (cumulative) test cases and brought the cumulative error count to 25. We stopped testing when the certification model predicted a 0.9934 ((2)34) reliability.

Finally, in certifying increment 4, the total number of test cases was 989, with 36 total failures (increment 4 is, of course, the total system). We stopped testing when the certification model predicted a 0.9978 ((2)78) reliability.

A few more metrics from this project may be of interest. Of the 36 operational failures, seven were in or related to the reused library software, three were Ada compiler errors, and six were a consequence of the Cleanroom team's lack of Ada knowledge. Twenty failures were caused by logic errors and file-related errors. Of course, all errors are significant in certification because it is from a user perspective, and the customer isn't going to care who made the errors or why.

At this point, we would be justified in putting this system into a library and noting that it has a predicted reliability of 0.99 under the certification model. However, to estimate reliability to this level under the sampling model and to have a certain level of confidence in the demonstration, we might require additional tests. Ideally, the system would first be released to a statistically selected group of users, for whom it might amass a half million failure-free uses, which would justify a claim of five 9s with high confidence. Next the system would be made generally available and followed to see if, after billions of uses, it has earned the status "six sigma."

Software-reliability models can be applied to software development in typical industrial settings, including the development of entirely new systems and those based on reuse. The models are independent of language and development method, but, for the models to be meaningful, the software must be of high quality. Therefore, these efforts are most significant when they are used in the context of a high-quality development methodology such as Cleanroom software engineering.

Techniques for estimating the reliability of individual software units and an entire system as it is being constructed and prepared for release are within the reach of most organizations. Mathematical complexities notwithstanding, reliability planning and certification lend themselves to straightforward spreadsheet manipulations. ♦

ACKNOWLEDGMENTS

We thank R. L. Linger and M. Pleszkoch of the IBM Cleanroom Software Technology Center and C. Trammell of the University of Maryland for their assistance with earlier versions of this article. We also acknowledge suggestions made by anonymous *IEEE Software* referees.

REFERENCES

1. J. Poore, D. Mutchler, and H. Mills, "STARS-Cleanroom Reliability: Cleanroom Ideas in the STARS Environment," Tech. Report IBM STARS CDRL 1710, Software Engineering Technology (available from Asset, Morgantown, W.V.), 1989.
2. D. Parnas, A. van Schouwen, and S. Kwan, "Evaluation of Safety-Critical Software," *Comm. ACM*, June 1990, pp. 636-648.
3. R. Cheung, "A User-Oriented Software Reliability Model," *IEEE Trans. Software Engineering*, Mar. 1980, pp. 118-125.
4. K. Siegrist, "Reliability of Systems with Markov Transfer of Control," *IEEE Trans. Software Engineering*, July 1988, pp. 1049-1053.
5. A. Currit, M. Dyer, and H. Mills, "Certifying the Reliability of Software," *IEEE Trans. Software Engineering*, Jan. 1986, pp. 3-11.
6. H. Mills, M. Dyer, and R. Linger, "Cleanroom Software Engineering," *IEEE Software*, Sept. 1987, pp. 19-24.
7. J. Whittaker, "Markov Chain Techniques for Software Testing and Reliability Analysis," PhD dissertation, CS Dept., Univ. of Tennessee, Knoxville, May 1992.
8. J. Whittaker and J. Poore, "Markov Analysis of Software Specifications," *ACM Trans. Software Engineering and Methodology*, to be published.
9. R. Linger, H. Mills, and B. Witt, *Structured Programming: Theory and Practice*, Addison-Wesley, Reading, Mass., 1979.
10. J. Poore et al., "Cleanroom Reliability Manager: A Case Study Using Cleanroom with Box Structures ADL," Tech. Report IBM STARS CDRL 1940, Software Engineering Technology, (available from Asset, Morgantown, W.V.), 1990.

J. H. Poore is a professor and department head of computer science at the University of Tennessee. His interests are in science policy and software engineering.

Poore received a PhD in information and computer science from Georgia Tech. He is a member of the ACM and IEEE.

Harlan D. Mills is a professor of computer science at the Florida Institute of Technology and president of Software Engineering Technology. He has written or coauthored six books and more than 50 refereed technical journal articles on topics related to software engineering.

Mills received a PhD in mathematics from Iowa State University. He is an honorary fellow of Wesleyan University and a fellow of IBM, the American Computer Programming Association, and the IEEE. He also holds the Warnier Prize for contributions to computer science.

David Mutchler is an assistant professor of computer science at the University of Tennessee. His research interests include the probabilistic analysis of heuristic search, the theory of machine game-playing, and the performance analysis of replicated databases.

Mutchler received a BS and an MS in mathematics from the University of Virginia and a PhD in computer science from Duke University. He is a member of the IEEE Computer Society and ACM.

Address questions about this article to Poore at University of Tennessee, CS Dept., 107 Ayres Hall, Knoxville, TN 37996-1301; Internet poore@cs.utk.edu.