1/47

SOFTWARE RELIABILITY ANALYSIS AND REPOSITORY PRECLOSURE SAFETY—AN INITIAL SURVEY

Prepared for

U.S. Nuclear Regulatory Commission
Contract NRC–02–97–009

Prepared by

Norman A. Eisenberg

Center for Nuclear Waste Regulatory Analyses
San Antonio, Texas

September, 2001

2/47

ABSTRACT

An initial investigation is made into the significance of software reliability to the assessment of preclosure repository safety and into the potential application of software reliability methods to preclosure repository safety issues. Basic concepts and selected methodologies for software reliability analysis are described. Research and guidance developed by the U.S. Nuclear Regulatory Commission, the U.S. Department of Energy, the mining industry, and others for software reliability analysis in safety-critical systems are briefly surveyed. Based on this background, issues related to software reliability and preclosure repository safety are identified.

3/47

CONTENTS

4/47

FIGURES

5/47

TABLES

6/47

ACKNOWLEDGMENTS

This report was prepared to document work performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the U.S. Nuclear Regulatory Commission (NRC) under Contract No. NRC–02–97–009. The activities reported here were performed on behalf of the NRC Office of Nuclear Material Safety and Safeguards, Division of Waste Management. The report is an independent product of the CNWRA and does not necessarily reflect the view or regulatory position of the NRC.

The author thanks Dr. Biswajit Dasgupta and Dr. Asadul Chowdhury, CNWRA, for their help, guidance, expertise, and cooperation in developing this report. The author also thanks Professor Jeff Tian, Southern Methodist University, for providing copies of several of his papers, which were instrumental in the development of this report. The author also extends his gratitude to Mr. Jiantao Pan (formerly Carnegie-Mellon University, currently Microsoft Corporation) for granting permission to use his figure on software reliability growth. Mr. Pan developed this figure as part of a paper, Software Reliability, which is an excellent summary of software reliability and was helpful in this effort; he wrote this paper for the graduate course, Dependable Embedded Systems, given by Professor Koopman at the Carnegie-Mellon University. The author also acknowledges the substantial assistance of Ms. Cheryl Patton, CNWRA, for her expertise and effort in placing this report in the approved format, thereby conforming to the latest standards. The author gratefully acknowledges the technical and programmatic reviews provided by Dr. Budhi Sagar and Dr. Wesley Patrick, respectively, CNWRA, and the editorial review of J. Pryor.

QUALITY OF DATA, ANALYSES, AND CODE DEVELOPMENT

DATA: No CNWRA-generated original data are contained in this report.

ANALYSES AND CODES: No analyses, codes, or analyses were used in this report.

7/47

1 INTRODUCTION

Software reliability may be a significant factor in the safe operation of the proposed Yucca Mountain repository. During the preclosure phase of repository development, operations are expected to be directly controlled or assisted by computer systems. The candidate operations and systems for computer control include overhead bridge cranes, trolleys, waste-container transporters, and gantries used to move casks, canisters, bare-fuel assemblies, or waste packages. Although descriptions of computer-controlled operations in the U.S. Department of Energy (DOE) documents provide little detail on the specific hardware and software systems planned to be used, it is clear that the DOE intends to apply software-based digital systems for safety-critical applications during repository operations. There are a great many techniques for analyzing the safety and reliability of software systems. In addition, the U.S. Nuclear Regulatory Commission (NRC) and other organizations have issued guidance on software reliability used in safety-critical applications. This large amount of information is reviewed and suggestions are made for incorporating software reliability considerations into the Center for Nuclear Waste Regulatory Analyses (CNWRA) PCSA Tool (Dasgupta, et al., 2000, 2001). As is the case with human reliability analysis, software reliability analysis has received intense study and continues to benefit from ongoing research activities. An important aspect of applying this large amount of information to the issues of preclosure repository safety review is to select existing methods and guidance with the greatest applicability to the repository safety issues.

Section 2 of this report provides a brief overview of the issues, methods, and approaches to software reliability analysis. This overview includes: (2.1) definition of software reliability; (2.2) unique aspects of software reliability; (2.3) software reliability analysis methodologies and approaches (2.3), including a long, but not comprehensive, list of software reliability models, and a discussion of approaches to software reliability in safety-critical systems. Section 3 provides a summary of NRC guidance and research on software reliability analysis. Section 4 summarizes some of the DOE/Yucca Mountain Project approaches to software reliability analysis for repository operations. Section 5 summarizes important research and standard-setting work related to software safety accomplished in the mining industry and associated regulatory agencies. Section 6 provides conclusions and recommendations, and Section 7 lists the references.

8/47

2 OVERVIEW OF SOFTWARE RELIABILITY ANALYSIS—ISSUES, METHODS, AND APPROACHES

2.1 Definition of Software Reliability

According to the American National Standards Institute, software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment (American National Standards Institute/Institute of Electrical and Electronic Engineers, 1991). Software reliability has also been defined as the probability that a software system will run continuously without failing, given a perfect operating environment (Knight and Littlewood, 1994).

Software reliability engineering is the discipline "concerned with the assessment, prediction, and management of the reliability of software systems" (Tian, 1998).

2.2 Unique Aspects of Software Reliability

To provide a background for describing software reliability, it is necessary to articulate the substantial differences between the approaches for analyzing the reliability of software and the approaches used for mechanical, electrical, and electronic systems. Fundamental differences in the nature of software versus physical systems drive the substantial differences in the approaches to reliability analysis.

The reliability of software has markedly different characteristics from the reliability of hardware. Mechanical, electrical, and electronic hardware devices generally have failure significantly influenced by time. For many mechanical systems, subsystems, and devices, the probability of failure increases markedly with time as components wear out. For many mechanical, electrical, and electronic hardware devices, a bathtub curve for probability of failure is characteristic (see Figure 2-1) because juvenile or burn-in failures occur more frequently at the beginning of service, a low rate of failure occurs after this initial period, and failures rise again later as aging-related failures increase. Software reliability (i.e., the inverse of failure probability) is not a direct function of time in service. Software does not wear out. It does not change with time, unless it is intentionally modified. Faults in the software are present from the time of its creation and may remain undiscovered for the entire service lifetime. Faults discovered in software, if corrected, increase the reliability of the software. Because changes to software have the potential to introduce additional errors, most authors indicate an initial increase in failure rate following the correction of an error, followed by time decay to a lower failure rate, until the next error correction is made (see Figure 2-2). Notice also that the apparent failure rate rises as more faults are discovered in the software. A dramatic example of how an attempted repair to software introduces additional defects is the case in which AT&T attempted to resolve a simple problem in its telephone system software, resulting in failure of the entire system for a client base of millions (Collins, et al., 1994).

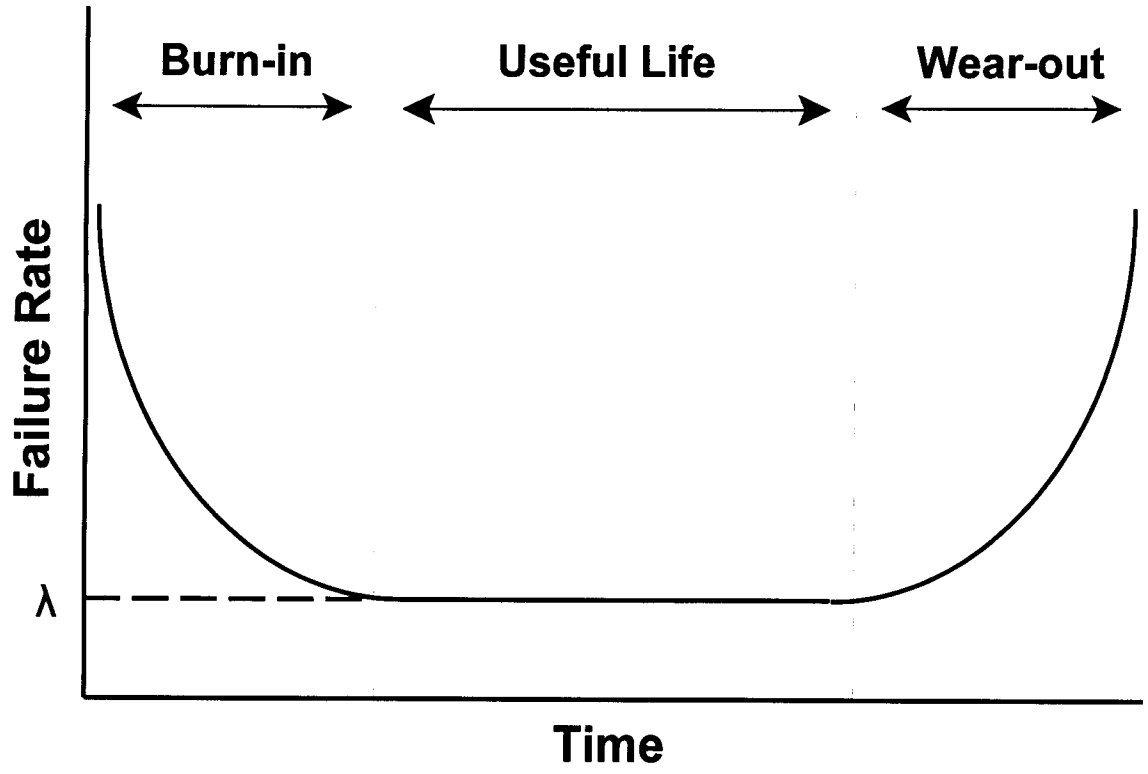

Figure 2-1. Typical Bathtub Curve for Failure Rate Versus Time, Characteristic of Mechanical, Electrical, and Electronic Hardware; Based, with Permission, on a Figure from (Pan, 1999)

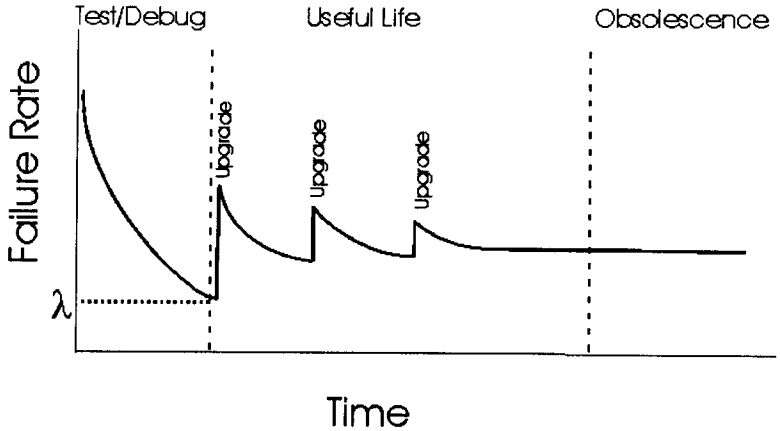

Figure 2-2. Hypothetical Failure Rate for Software (Pan, 1999)

The differences in the reliability behavior between hardware and software are due to the fundamental differences in their natures. Some of these fundamental differences are mentioned in Table 2-1. Hardware reliability decreases in time, after an initial period of juvenile failures, because physical phenomena, such as wear, corrosion, and fatigue, degrade the hardware. There are no counterparts for software to these physical phenomena causing degradation. In fact, the physical existence of software is tenuous. The physical manifestation of software may be a floppy computer disk, compact disk, or other electronic medium. To be used, the software is entered into a computer's memory, where it is represented by binary code. In this sense, the software is physically manifested as the logical state of the computer. There is little concern regarding the physical media used to distribute copies of software, because the means for checking accurate reproduction are excellent and extremely reliable.

Table 2-1. Differences Between Hardware and Software That Are Relevant to Reliability (Keene, 1994; Pan, 1999)

Attribute	Hardware	Software
Cause of failure	Wear, corrosion, fatigue, design defect, unanticipated environment	Mainly design defect, unexpected input
Wear-out	Related to energy input	None
Repair	Mechanical repair or maintenance can diminish, delay, or reverse wear	Periodic restart may eliminate problems for a time
Environmental factors	May accelerate or decelerate wear and degradation	No effect, except to the extent that inputs are changed
Reliability models	Predicted by physical models	No physical basis for reliability, depends on design
Redundancy	Parallel subsystems can improve reliability, except for common-cause failure	Parallel subsystems cannot improve reliability, because identical software will have the same errors
Standardized components	Adherence to engineering specifications and codes has greatly increased reliability of mechanical systems	No standardized code; reuse of software can improve reliability

The physical phenomena that reduce hardware reliability in time are the primary cause for other differences in hardware and software reliability. The facts that wear-out is related to energy input and that environmental factors may have a significant impact for hardware are in stark contrast to the behavior of software, which does not wear out and is not directly affected by the environment in which it is used. Other differences between hardware and software reliability relate to the fact that software is typically created with a number of defects. Because software

components (e.g., subroutines), are reproduced flawlessly, any defects are also replicated. This means that building software systems with parallel architecture will not increase reliability, because common-cause failures are essentially built in. Hardware reliability has been enhanced by standardized components and manufacturing codes and specifications. No such standardized components or manufacturing codes exist for software. Existing software standards attempt to control the software creation process, but fall short of specification of standardized components, which is common practice for hardware. Nevertheless, there is some standardization accomplished by reuse of software components, especially for certain military applications.

2.3 Software Reliability Analysis Methodologies and Approaches

Software reliability analysis uses some of the same concepts and methods used in conventional (traditional) reliability analysis. Those concepts and methods are briefly presented, with special attention to the differences between conventional and software reliability and to special terms used in software reliability analysis.

An important concept in software reliability is the distinction between a fault and a failure. One author (Tian, 1998) states

> A failure is a behavioral deviation from the user requirement or specification, a fault is the underlying cause within a software system that causes certain failure(s), while an error refers to human mistakes or misconceptions that result in faults being injected in software systems.

For example, educational software for children promises animation and sound if a certain icon is selected; however, nothing happens when the icon is selected. This condition is a failure. The underlying cause is some defect in the code, either incorrect code or missing code. Failure corresponds to the user's view and depends on the dynamic behavior of the software system; fault corresponds to the developer's view and is a static characteristic of the code. In certain contexts, this distinction is important because, although software may have several faults, these faults may never cause a failure, if the user and the environment of the software are such that the defective portions of the code are never used. By its very definition, failure requires the software to be operating; faults may be present in software, whether it is being utilized or not.

2.3.1 Reliability Modeling

Consider the usual definitions of cumulative distribution function, complementary cumulative distribution function, and probability density function for a continuous random variable (Bowen and Bannett, 1988):

$$F(x) = Pr(X \leq x) \tag{2-1}$$

where

$F(x)$ — cumulative distribution function
$Pr(C)$ — probability that condition C exists

X — random variable
x — a value that X can take

and

$$f(x) = dF(x)/dx \tag{2-2}$$

where f(x) is the probability density function

and

$$F^c(x) = 1 - F(x) \tag{2-3}$$

where $F^c(x)$ is the complementary cumulative distribution function.

Now consider the reliability of software in the time domain. Then the reliability, $R(t)$, is given by (Tian, 1998; Musa, 1998; Friedman and Voas, 1995):

$$R(t) = 1 - G(t) \tag{2-4}$$

where $R(t)$ is the probability that the software will perform without any failure to time t (usually operation or execution time but may be real time, depending on choice of the analyst); $G(t)$ is the failure probability (i.e., the probability that the time of failure is less than or equal to t).

In other words, $G(t)$ is the cumulative distribution function of failure time. The unconditional probability distribution function for failure time, $g(t)$, follows from applying Eq. (2-2) and is given by:

$$g(t) = dG/dt \tag{2-5}$$

The hazard rate (or hazard function), $z(t)$, is the failure density, conditional on no failure occurring prior to time t. Using the rules of conditional probability, the hazard rate is given by:

$$z(t) = g(t)/R(t) \tag{2-6}$$

Differentiating Eq. (2-4) and combining with Eqs. (2-5) and (2-6) yields

$$dR/R = -z(t)dt \tag{2-7}$$

which may be integrated with the boundary condition, $R(0) = 1$, to yield

$$R(t) = \exp\left(- \int_0^t z(\xi)\, d\xi \right) \tag{2-8}$$

From the reliability function, $R(t)$, one can derive the expression for mean time between failures, commonly known as MTBF, an important intuitive measure of reliability (Tian, 1998):

$$\text{MTBF} = \int_0^\infty R(\xi)\,\mathrm{d}\,\xi \tag{2-9}$$

Musa (1998) denotes the integral in Eq. (2-9) as the mean time to failure, also commonly known as MTTF. In conventional reliability the MTBF = MTTF + MTTR (where MTTR is mean time to repair or replace). However, Musa considers execution time to be the operative variable for software reliability, so the processes that challenge the software cease until the fault is repaired. According to this assumption, the MTTR has no influence on the occurrence of additional faults, so MTBF may be considered the same as MTTF.

Additional terminology is employed to describe stochastic processes, which form the basis for some of the software reliability models. A stochastic process may be defined as a sequence of random variables. For example, the number of failures of a software system may be considered to be a random variable, but may take on only an integral value. If the software is tested, a particular value, N_1, may be obtained; in another test, a second value, N_2, may be obtained. This variability is because the exact nature, sequencing, and timing of inputs by users in the two tests are likely to be different. One can then view the number of failures at a given time as a selection of the result from an infinite set of experiments or tests (i.e., a stochastic process). Working with this stochastic modeling viewpoint, one may define the mean value function, $m(t)$:

$$m(t) = E\{N(t)\} \tag{2-10}$$

where

- $m(t)$ — mean value function
- $E\{X(t)\}$ — the expected value of the random variable $X(t)$ resulting from a stochastic process
- $N(t)$ — cumulative number of faults in the software system experienced at time t

Based on these definitions, one may define the instantaneous failure rate, $\lambda(t)$:

$$\lambda(t) = \mathrm{d}m(t)/\mathrm{d}t \tag{2-11}$$

The instantaneous failure rate, $\lambda(t)$, is similar to the failure intensity, $z(t)$. However, the failure intensity, $z(t)$, is defined for a specific software system and its actual failure history; the instantaneous failure rate, $\lambda(t)$, is defined for a stochastic view of a software system, so the actual failure history is considered to be just one realization of a large number of identical software systems. As a consequence of this difference in viewpoint, the failure intensity, $z(t)$, is generally a discontinuous function, while the instantaneous failure rate, $\lambda(t)$, is generally a continuous function.

2.3.2 Software Reliability Models

Several references provide compilations of the software failure models (Tian, 1998; Musa and Okumoto, 1983; Musa, 1998; Smith, 1997; Friedman and Voas, 1995; Smidts and Li, 2000). Smidts and Li propose four categories of software reliability models: (i) reliability growth models; (ii) input domain models; (iii) architectural models; and (iv) early prediction models. These categories are discussed in more detail in Section 3.2.1. Musa (1998) and Tian (1998) propose

14/47

and use classification schemes for software failure models. Musa and Okumoto (1983) propose a classification scheme based on four attributes:

(i) Time domain—whether the model uses calendar or execution time
(ii) Category—whether the number of failures is finite or infinite, considering an infinite operating time
(iii) Type—the mathematical distribution of failures, given the time domain used
(iv) Group—the functional form of the failure intensity (in terms of time, for the finite category; in terms of expected number of failures experienced, for the infinite category)

Some important reliability models have been classified according to this scheme, which are reproduced as Table 2-2.

As discussed in Section 2.2, one way that software reliability is quite different from reliability for mechanical or electrical systems is that there are no physical mechanisms that would cause elapsed time to be linked to the probability of software failure. Software does not have processes like corrosion or wear that can cause failure. Real time is not necessarily the correct variable to model software reliability. One important characteristic of software reliability models is the fundamental variable used to characterize reliability, as indicated in the Musa classification scheme (see Table 2-2). Although the distinction is of theoretical importance, on a practical basis, the distinction tends to be submerged. First, in most situations, there is a rough proportionality between execution time and real time; in those circumstances, a simple constant relates the two approaches. For example, suppose a software system is operated by a business during its normal business hours (8 hours per day, 5 days per week), then the ratio of execution time to real time is 40 hours per week, 168 hours per week. Another reason that the distinction between execution time and real time is less apparent to the users of information about software reliability is that the developers of models for software reliability are well acquainted with the fact that users wish to incorporate the software reliability models into an overall analysis of system reliability. For this reason, the reliability model must operate using real time as the fundamental variable, so the model will be compatible with the remainder of the analysis.

The Tian (1998) classification scheme has as its major division the input domain used by the model. Tian defines two types of software reliability models

- Software reliability growth models—use time as the fundamental modeling variable and make use of failure data recorded as a function of time

- Input domain reliability models—use input sequence as the fundamental modeling variable and analyze the probability of various input states and combine that with the probability that a given input state will cause failure

Table 2-2. Software Reliability Model Classification Scheme Adapted from Musa (1998)

	Class	Type		
Finite Failures Models	Exponential	Poisson	Binomial	Other
		Musa (1975) Moranda (1975) Schneidewind (1975) Goel-Okumoto (1979)	Jelinski-Moranda (1972) Shooman (1972)	Goel-Okumoto (1978) Musa (1979) Keiler, et al. (1983)
	Weibull		Schick-Wolverton (1973) Wagoner (1973)	
	C1		Schick-Wolverton (1978)	
	Pareto		Littlewood (1981)	
	Gamma	Yamada-Ohba-Osahi, (1983)		
Infinite Failure	**Family**	**Type**		
	Geometric	Poisson	Other	
		Musa-Okumoto (1984)	Moranda (1975)	
	Inverse Linear		Littlewood-Verrall (1973)	
	Inverse Polynomial		Littlewood-Verrall (1973)	
	Power	Crow (1974)		

Note: The year in parentheses indicates the publication year of the initial paper articulating the model.

2.3.2.1 Software Reliability Growth Models

Some of the important software reliability growth models identified by Tian (1998) and Musa (1998) are described in the next sections. It is important to understand that these methods for quantifying reliability of software are subject to challenge. In particular, the ability to predict the reliability of software for safety-critical applications may not be amenable to such straightforward reliability estimation. Section 2.3.4 discusses methods applicable to safety-critical applications.

2.3.2.1.1 De-eutrophication Models

De-eutrophication (literally nutrient-removal) models are so named because the failure probability is functionally related to the number of defects remaining in the software system at a given time; this model presumes that the reliability of the software is growing, because software faults are observed and removed. The Jelinski-Moranda model assumes that the hazard rate (probability of failure per unit time) is proportional to the number of defects remaining in the software system. Thus,

$$z_i = \varphi(N-[i-1]) \tag{2-11}$$

where

z_i — hazard rate for the i^{th} failure (i.e., after the discovery of the $i-1$ failure)
N — initial number of faults in the software
φ — constant of proportionality

Between observation and correction of failures, the hazard rate remains constant; the hazard rate is reduced by φ at each fault observation and removal. Thus, hazard rate as a function of time is a sequence of decreasing steps. The Shooman model is similar to the Jelinski-Moranda model, but uses slightly different parameters and counts failures over defined periods.

Moranda proposed a slight variation on Eq. (2-11)

$$z_i = z_0\varphi^{[i-1]} \tag{2-12}$$

where

z_0 — initial hazard rate
φ — a constant, as before

and it is required that $\varphi < 1$. The time history for this model is also a sequence of steps, with each step initiated by the discovery and correction of a fault. However, unlike the Jelinski-Moranda model [Eq. (2-11)], the steps in this model is of decreasing magnitude. This decrease in hazard rate reduction with successive fault observations accounts for the increased difficulty in observing and removing the remaining faults.

2.3.2.1.2 Nonhomogeneous Poisson Process Models

The stochastic view of software reliability modeling is discussed in Section 2.3.1. For this concept of reliability, a straightforward application of a Poisson process yields

$$P\{X(t) = n\} = \frac{[m(t)]^n e^{-m(t)}}{n!} \tag{2-13}$$

where

$X(t)$ — number of failures for the time interval $(0,t)$
n — an integer
$m(t)$ — mean value function defined in Eq. (2-9)

Some of the named models in Table 2-2 may be obtained by specifying the functional form of $m(t)$ in Eq. (2-13). This choice is equivalent to specifying the distribution of failures as a function of time, i.e., specifying the functional form of $\lambda(t)$, the instantaneous failure rate, as defined in Eq. (2-10). The Goel-Okumoto model (exponential model) is obtained by the following choice

$$m(t) = N(1 - e^{-bt}) \tag{2-14}$$

and the Yamada-Ohba-Osaki model (S-shaped model) is obtained by the following choice

$$m(t) = N[1 - (1 + bt)e^{-bt}] \tag{2-15}$$

where

- N — estimated number of total defects
- b — a constant

The S-shaped model is useful for describing a failure rate history that starts small, rises rapidly in a middle phase, and then slows down again when many of the faults have been detected and repaired.

2.3.2.1.3 Generalized Poisson Model

The Jelinski-Moranda model [Eq. (2-11)] may be generalized (i) by considering failures of a sequence of intervals of varying length and (ii) by considering the problem in the context of a stochastic process, rather than a deterministic process. A Poisson distribution is then used to describe the number of failures in each interval, as follows

$$m_i(t_i) = \varphi\,[N - M_{i-1}] \cdot g_i(t_1, t_2, \ldots, t_i) \tag{2-16}$$

where

- $m_i(t_i)$ — mean value function described in Eq. (2-9) and is given here as the expected value of f_i, the number of failures in interval i
- t_i — test length of interval i
- N — estimated total number of defects
- M_i — defects discovered and removed in the i^{th} interval
- φ — a constant of proportionality, as before
- g_i — a function of t_i that has a specific form for a particular named model

Named models result when specific functional forms are chosen for the functions f_i and g_i. In particular

- The Jelinski-Moranda model [Eq. (2-11)] results for the choices: $f_i = 1$ and $g_i = t_i$. Note that for these choices, $m_i(t_i) = \varphi[N - M_{i-1}]t_i$, so $dm_i/dt = \varphi[N - M_{i-1}] = \lambda(t) \approx z(t)$, which is the same result as in Eq. (2-11), except this expression is on a per-interval basis, while Eq. (2-11) is on a per-failure basis. As before, the precise mathematical distinctions between a deterministic versus a stochastic viewpoint are neglected.

- The Schick-Wolverton model results for the choices: $f_i = 1$ and $g_i = t_i^2/2$. For these choices, the instantaneous failure rate becomes $\lambda(t) = \varphi[N - M_{i-1}]t_i$, which, unlike the

result for the Jelinski-Moranda model is not constant, but increases linearly with time. The probability of observing a failure increases as the square of time.

2.3.2.1.4 Execution Time Models

Musa (1998) argues strongly that execution time, rather than real time, is the correct variable to use for estimating software failure and failure rates; however, he acknowledges that real time is a more natural measure, is better understood by the user community, and can usually be converted to execution time by multiplying by a computer utilization factor. The Musa basic execution time model (Poisson/exponential/finite failure model, see Table 2-2) is

$$\lambda(\tau) = \lambda_0 \cdot \exp(-\lambda_0/v_0)\,\tau \qquad (2\text{-}17)$$

where,

- $\lambda(\tau)$ — instantaneous failure rate (= $dm/d\tau$)
- τ — execution time (not real time t)
- λ_0 — initial failure intensity at the beginning of execution
- v_0 — expected number of failures for an infinite time

By assuming a logarithmic Poisson model, the Musa-Okumoto model (Poisson/geometric/infinite failure model) was obtained as shown in Eq. (2-18)

$$m(\tau) = \theta^{-1}\log(\lambda_0\theta\,\tau + 1) \qquad (2\text{-}18)$$

where

- τ — execution time
- $m(\tau)$ — mean value function described in Eq. (2-9)
- θ — failure intensity decay parameter, which is the fractional decrement of failure intensity with time
- λ_0 — initial failure intensity at the beginning of execution

Because of their different time dependencies, the Musa basic model and the Musa-Okumoto model may be used to bound the actual failure history from above and below, respectively.

2.3.2.1.5 Schneidewind Model

The Schneidewind model provides the following estimate for the mean value function

$$m_i = (\alpha/\beta)[e^{-\beta(i-1)} - e^{-\beta i}] \qquad (2\text{-}19)$$

where

m_i is the expected number of failures in the i^{th} interval, where all intervals have equal lengths and α and β are model-fitting parameters, which permit variable weighting of recent and older failures.

2.3.2.1.6 Littlewood-Varrell Model

The Littlewood-Varrell model assumes that the faults are independent and their arrival time is described by an exponential distribution. No execution time is allocated to repair, because, generally, the software will be taken out of service until the fault is repaired. The instantaneous failure rate is given by

$$\lambda(\tau) = [(n-i)\cdot\alpha]/[\beta + \tau_i + \tau] \quad \text{for } \tau_i < \tau < \tau_{i+1} \tag{2-20}$$

where

- τ — execution time (not the real time t)
- n — initial number of faults
- τ_i — time of occurrence of the i^{th} fault
- α — constants determined by fitting data
- β — constants determined by fitting data

Using this model for instantaneous failure rate gives the following form for the mean value function

$$m(\tau) = [(n - i)\cdot\alpha] \cdot \ln\{[\beta + \tau_i + \tau]/[\beta + \tau_i]\} \tag{2-21}$$

where the variables are defined as before. The values provided by Eqs. (2-20) and (2-21) are mean values. Because the Littlewood-Varrell model is based on Bayesian updating at each sequence, the instantaneous failure rate is described by a gamma distribution.

2.3.2.2 Input Domain Reliability Models

Input domain reliability models are substantially different from the time domain reliability models described in Sections 2.3.2.1.1 to 2.3.2.1.6. Unlike those models, the input domain models do not attempt to tie software reliability to time, neither execution time nor real time. Instead, the variable determining reliability is the input state of the software. Reliability, in this context, is defined as the probability of no failures, given a specific input state for the software (Tian, 1998).

The input domain for a software model is defined by the set of inputs for the software and the probabilities that various inputs will occur. These probabilities are based on anticipated or actual usage patterns for the software. Suppose N different input variables, E_j, must be specified in order for the software to execute a run. Each model input may be considered to be an N-component vector, E_{ij}, where $j = 1,2,...N$ indicates the particular input variable and $i = 1,2,...M$ indicates the enumeration (label) for each distinct input vector. Associated with each input vector, i, is a scalar, P_i, which is the probability that the particular input vector will be used (the P_i conforms to the three postulates of probability theory). The P_i comprises the operational profile for the software system.

As one example of an input domain reliability model, consider the Nelson model. An estimate of the reliability is obtained from a sample of n out of the M possible input vectors. A key assumption for the Nelson model is that no repair of faults is undertaken during the test. These n samples are randomly selected according to the input domain probabilities, P_i. The estimated reliability is then given by:

$$\check{R} = (n - f)/n \qquad (2\text{-}22)$$

where

Ř — input domain reliability, as defined previously;
f — number of failures encountered for a randomly selected sample of *n* inputs

Note that exhaustive testing of all *M* input vectors is usually not an option for anything but the simplest software; this inability to test complexity is one reason why the value obtained from Eq. (2-22) is only estimate of reliability.

Brown and Lipow provided a generalization of the Nelson model by partitioning the inputs into separate domains, D_k. In this case, a separate failure rate is calculated for each domain, and the reliability is based on a probability-weighted sum

$$\check{R} = 1 - \sum_{k=1}^{N} \frac{f_k}{n_k} P(D_k) \qquad (2\text{-}23)$$

where

f_k — number of failures for testing domain *k*
n_k — number of domains
$P(D_k)$ — probability that inputs in domain, D_k, will be used in the application of the software

Each domain, D_k, contains a subset of the input vectors, E_{ij}, which are constrained in some fashion by the particular domain definition. Once again, since only a representative sample will be used from each domain, the value provided by Eq. (2-23) is only an estimate of the reliability.

2.3.3 Difficulties in Quantifying Software Reliability

Given these descriptions of various methods to quantify the reliability of software, it is important to point out that such methods have been seriously questioned and criticized by experts in the field of software reliability. These criticisms tend to fall into two categories: (i) theoretical issues regarding the applicability of various assumptions to quantifying software reliability (e.g., that a suitable variable to compute reliability is time) and (ii) practical issues regarding the feasibility of obtaining data for reliability models (e.g., obtaining failure data in a reasonable time with a reasonable amount of testing for highly reliable software). Smidts and Li (2000) indicate that "identification of a complete set of software engineering measures from which software reliability can be predicted" is not feasible at this time. Pan (1999) indicates that determination of software reliability is difficult because (i) understanding the nature of software is limited, (ii) there is no clear identification of what aspects of software are determinants of reliability, (iii) suitable measures for software reliability are not available, and (iv) there is lack of agreement on the definition of basic software characteristics (e.g., software size). Butler and Finelli have written several papers on the infeasibility of quantifying software reliability (Butler and Finelli, 1991,1993). Work by Butler and Finelli (1991) identifies the theoretical problem in assuming that separately programmed subsets of code will be statistically independent.

Evidence is presented that coding performed by separate groups suffered from common faults and failures. A practical problem identified is that to give software a high level of reliability, using a standard reliability growth model (these are discussed in Section 2.3.2), an infeasible amount of testing is required. For example, in order to reach a range of ultra-reliability, with failure rates of 10^{-7} to 10^{-9} per hour, accelerated testing of the software would need to proceed for decades. The impracticality of determining reliability by testing, for highly reliable software, is repeated in a paper by Butler and Johnson (1993). It is asserted that to certify a probability of failure of 10^{-9} for a 1-hour mission, testing must be done for 10^{9} hours (i.e., 114,000 years). Similarly, Leveson (1991) indicates that although testing software to ascertain high reliability is currently infeasible, it may be possible with technological advances.

The difficulty of quantifying software reliability implies that for repository preclosure safety, the issue needs to be approached with caution. Although it would be desirable to have a simple and accepted method for quantifying software reliability so the implications for repository safety could be quantified, such a method is not currently available. A practical approach of limited testing of various reliability quantification methods seems appropriate, while the software engineering field continues research on reliability quantification methods and while DOE develops more details on the exact nature of the software systems it plans to use. In addition, procedural approaches, such as those contained in various standards, provide an additional yardstick by which to judge the effectiveness of the DOE effort.

2.3.4 Software Reliability in Safety-Critical Systems

Although software reliability may be estimated by a variety of methods, as discussed in Section 2.3.2, to use such methods in a straightforward manner in a risk analysis may, at best, substantially overestimate the risks or, at worst, obfuscate the true risks and system vulnerabilities needing improvement. This potential for over-estimation results from the fundamental definition of failure used in software reliability analysis. Any behavior produced by the software that does not meet the user's specification is considered a failure. For example, consider the possible failures of a hypothetical software system used for repository operations.

1. To provide input to the system, each user must sign on with an identification and password. The software fails to obscure the password on the user's monitor, when a user types it in.

2. To maintain the computer system, including hardware, operating system, and firmware updates, the software execution must be stopped periodically. The software fails to exit normally, and the computer must be manually turned off to end the program execution.

3. For security purposes and to ensure worker safety, the software tracks the entry and exit of personnel into certain critical areas, so that sources of high radiation fields (e.g., waste containers, spent nuclear fuel assemblies) are not collocated with personnel. The software tracks each person with a unique identification number input to the system by a card reader at the entry point to the critical area. The software successfully tracks the entry and exit to the critical area of each personal identification number, but incorrectly associates the number with the person's name.

4. As part of the operations in the cask carrier-handling system, the shipping cask is upended on the transport vehicle, so it can be placed on a cart in the Waste Handling Building (CRWMS M&O, 2000b). To minimize exposure of workers to radiation, the upending operation is controlled by the software system, after the lifting yoke is attached to the cask trunnions. Because of a software fault, the software may begin the upending operation before both sides of the yoke have been fastened to the cask or before the cask hold-downs have been released. This software failure may result in injury to workers or damage to the crane, the Waste Handling Building cart, the cask, or critical components of the Waste Handling Building with an associated possibility of radiological hazard.

Although Failure 1 is technically a failure and may pose a slight security risk, it is unlikely to be safety related. Failure 2 is technically a failure, but is probably only an annoyance to the system operators. Failure 3 is also technically a failure, but is not likely to be safety related as long as the software consistently misidentifies workers. Failure 4 is not only a technical failure, but presents a potentially serious safety hazard. Clearly Failure 4 is the type to focus on. The issue, then, is what system of analysis will (i) identify critical parts of the software system that are safety related and (ii) provide a method for correcting or avoiding such software faults.

The following definitions taken from Lawrence (1995b) help to clarify the distinctions between software reliability and safety-critical software reliability:

> **Safety-critical software** is software whose inadvertent response to stimuli, failure to respond when required, response out-of-sequence, or response in unplanned combination with others can result in an accident or the exacerbation of an accident. This includes software whose operation or failure to operate can lead to a hazardous state, software intended to recover from equipment malfunctions or external insults, and software intended to mitigate the severity of, or recover from, an accident.
>
> **A critical system** is a system whose failure may lead to unacceptable consequences. The results of failure may affect the developers of the system, its direct users, their customers, or the general public. The consequences may involve loss of life or property, financial loss, legal liability, regulatory actions, or even the loss of good will if that is extremely important. The term **safety critical** refers to a system whose failure could lead to loss of life, injury, or damage to the environment. For nuclear reactors, this includes radiation releases or exposure to the public or operators and reactor workers. [Note: a generalized version would say facility workers instead of reactor workers].
>
> **Hazard analysis** is the process of identifying and evaluating the hazards of a system, and then either eliminating the hazard or reducing its risk to an acceptable level (National Institute of Standards and Technology, 1993).
>
> **Software hazard analysis** ". . . eliminates or controls software hazards and hazards related to interfaces between the software and the system (including hardware and human components) it includes analyzing the requirements,

design, code, user interfaces and changes" (National Institute of Standards and Technology, 1993).

Several authors have proposed approaches to software reliability in safety-critical systems. For example, over a period of decades, Leveson and her associates have developed a methodology, Safeware, to provide modeling and hazard analysis for complex systems comprised of a mix of hardware, software, and human elements (Leveson, 1995; Leveson, et al.,1997; Reese and Leveson, 1997). This methodology extends to computer controlled systems, the hazard analysis techniques developed to address safety and reliability for the hardware aspects of electromechanical systems. This methodology identifies system hazards and uses a process of software hazard analysis and control while software is being developed. Analysis continues during the software development process, rather than being performed at the end in an attempt to demonstrate that a particular level of safety has been achieved or to otherwise qualify the software. To accomplish this analysis, a variety of analytical, management, and software development techniques are deployed, including fault-tree analysis, human-error analysis, completeness and consistency analysis, deviation analysis, test data coverage analysis, system engineering analyses, and operator training.

Smith (1999) provides a brief summary of approaches for safety-critical software. He points out that software fault-tree analysis (as proposed by Leveson) is an effective means of identifying safety-critical aspects of software. Formal methods, including model-, logic-, and net-based (e.g., Petri net) approaches, may improve the success of safety-critical software. Some programming languages, such as Ada, have been especially designed to produce fail-safe software. Finally, use of standards for software development helps to achieve safe system operation.

It is clear from this modest examination of safety-critical software development methods, practices, and analytical tools that both software development and safety analysis must be performed in an integrated manner; separation of software considerations, from the remainder of a safety-critical system, will produce erroneous results and perhaps unsafe conditions. An integrated analysis is required.

24/47

3 SUMMARY OF NRC GUIDANCE AND RESEARCH ON SOFTWARE RELIABILITY ANALYSIS

3.1 Existing NRC Guidance on Software Reliability Analysis

Existing U.S. Nuclear Regulatory Commission (NRC) guidance on software reliability analysis and software quality assurance is largely contained in the NRC Regulatory Guide series. Additional guidance may also be found in other forms, such as standard review plans, branch technical positions, and other documents. The NRC regulatory guides are placed in 10 divisions; all the guides related to computer software are in Division 1—Power Reactors. Table 3-1 lists those guides related to computer software. Subsequent sections provide a brief description of these regulatory guides.

All existing NRC guidance on software reliability analysis and computer software is written for nuclear power reactors. This limitation raises the question of whether the NRC staff will develop specific guidance for repository operations or whether existing guidance can be used.

Table 3-1. NRC Regulatory Guides Related to Software Reliability

Regulatory Guide Number	Title
1.152	Criteria for Digital Computers in Safety Systems of Nuclear Power Plants
1.168	Verification, Validation, Reviews, and audits for Digital Computer Software Used in Safety Systems of Nuclear Power Plants
1.169	Configuration Management Plans for Digital Computer Software Used in Safety Systems of Nuclear Power Plants
1.170	Software Test Documentation for Digital Computer Software Used in Safety Systems of Nuclear Power Plants
1.171	Software Unit Testing for Digital Computer Software Used in Safety Systems of Nuclear Power Plants
1.172	Software Requirements Specifications for Digital Computer Software Used in Safety Systems of Nuclear Power Plants
1.173	Developing Software Life Cycle Processes for Digital Computer Software Used in Safety Systems of Nuclear Power Plants

The regulatory guides in Table 3-1 are briefly described in the following sections.

- Regulatory Guide 1.152 describes a method acceptable to the NRC staff for complying with the Commission's regulations for promoting high functional reliability and design quality for the use of digital computers in safety systems of nuclear power plants. The term computer refers to a system that includes computer hardware, software, firmware, and interfaces. This regulatory guide endorses, with exceptions, IEEE Standard 7–4.3.2–1993. An important regulatory position articulated is that "The staff does not endorse the concept of quantitative reliability goals as a sole means of meeting the Commission's regulations for reliability of the digital computers used in safety systems."

- Regulatory Guide 1.168 describes an acceptable means to demonstrate compliance with the quality assurance requirements of 10 CFR Part 50 Appendix B. It endorses, with exceptions, IEEE Standard 1012–1986. Some important exceptions to the Institute of Electrical and Electronics Engineers standard, which reveal the NRC approach to software safety issues, include the following:

  — The definition of critical software is narrowed to apply only to safety-related systems within the context of the regulations and guidance

  — The staff does not endorse the concept of quantitative reliability goals as a sole means of meeting the Commission's regulations for reliability of the digital computers used in safety systems.

  — Although the Institute of Electrical and Electronics Engineers standard does not require independence in performing software verification and validation, NRC requires it

  — The requirement in the Institute of Electrical and Electronics Engineers standard, for reperforming verification and validation tasks if the software is changed, is tied into the regulatory requirement for reasonable assurance that operations can be conducted without endangering public health and safety

  — Post-development verification of off-the-shelf software is not allowed

  — Records of verification and validation activities must be maintained as quality assurance records

  — Certain steps in verification and validation, listed as optional in the Institute of Electrical and Electronics Engineers standard, are determined to be mandatory in this NRC guidance

- Regulatory Guide 1.169 endorses IEEE Standard 828–1990 on software configuration management plans, with exceptions.

- Regulatory Guide 1.170 endorses ANSI/IEEE Standard 829–1983, Institute of Electrical and Electronics Engineers Standard for Software Test Documentation, with exceptions.

- Regulatory Guide 1.171 endorses ANSI/IEEE Standard 1008–1987, Institute of Electrical and Electronics Engineers Standard for Software Unit Testing, with exceptions. The exceptions relate to the independence of the software unit testing and requirements for record keeping.

- Regulatory guide 1.172 endorses IEEE Standard 830–1993, Institute of Electrical and Electronics Engineers Recommended Practice for Software Requirements Specifications, with exceptions. An important exception states: For safety system software, unnecessary requirements should not be imposed. There may be documented variations in essential requirements, but the variations must be linked in the software requirements specifications either to site and equipment variations or to specific plant design bases and regulatory provisions.

- Regulatory Guide 1.173 endorses IEEE Standard 1074–1995, Institute of Electrical and Electronics Engineers Standard for Developing Software Life Cycle Processes, with exceptions. An important exception is the additional requirement that: Planned and documented software safety analysis activities should be conducted for each phase of the software development life cycle. The regulatory guide then lists the analyses that should be identified in the applicant life cycle model related to inputs, activity descriptions, and outputs.

3.2 NRC Research on Software Reliability Analysis

NRC has sponsored some significant research on software reliability over the last decade, including significant efforts at The University of Maryland, Center for Technology Risk Studies and Lawrence Livermore National Laboratory, Fission Energy and Systems Safety Program.

3.2.1 University of Maryland Research

A recent report, Software Engineering Measures for Predicting Software Reliability in Safety Critical Digital Systems (Smidts and Li, 2000), was prepared under a joint research agreement between the NRC and the University of Maryland. The focus of this report is the evaluation of various software reliability measures. Software reliability measures are defined in this report as "the degree to which a software system, component, or process possesses a given software attribute. For instance, the measure Line of Code assesses the physical size of a code. . ." Ultimately an expert elicitation is used to rate various reliability measures for various phases of the software life cycle. Although the report attempts to determine the best "reliability predictions system—a complete set of software engineering measures which can be used to predict software reliability" for estimating the reliability of instrument and control software, it is found that the question cannot be definitively answered at this time. The report identifies several software reliability measures that are good candidates for inclusion in an effective software reliability prediction methodology.

The report first defines reliability as the probability that the digital system will successfully perform its intended safety function (for all conditions under which it is expected to respond) upon demand with no unintended functions that might affect system safety. Note that this definition departs from the standard definition of software reliability by introducing the concept of failure on demand and relating failure to performance of a safety function.

Four categories of models are considered for modeling the reliability of software (note that the Smith classification scheme is different from that used in Section 2.3.2)

- Reliability growth models—extrapolate failure data and trends to predict subsequent reliability behavior
- Input domain models—derive a probability of success from the properties of the input domain of the software and the results of test cases designed to cover the input domain
- Architectural models—derive reliability estimates based on the architecture of the software and the reliability behavior of its constituent modules
- Early prediction models—bases reliability estimates on the characteristics of the software development process extrapolated to the operational phase

For each of these categories, a problem with its use for reliability prediction was identified.

- Reliability growth models—for highly reliable software, the amount of testing required to quantify the reliability is excessive and impractical
- Input domain models—a problematic assumption is that a finite set of equivalence classes can represent and test for the infinite input domain
- Architectural models—this approach requires the failure rates or probability of transition to a failure state for components of the software system, information that is generally not available or readily obtainable
- Early prediction models—a problematic assumption is that empirical relationships derived from a subset of software development applies to safety-critical systems

Based on these concerns, this report extends an evaluation begun by Lawrence Livermore National Laboratory (Lawrence, et al., 1998). Of the 78 measures identified by the Lawrence Livermore National Laboratory team, the University of Maryland team reduced these to 30 by eliminating models and other measures on the basis of importance. Then 10 additional measures, not yet in use, were suggested by the group of experts. Next each member of the team of 10 experts was elicited to rank the various measures. The experts' inputs were ranked using multiattribute utility theory, the ranks were aggregated, and the results analyzed. This process identified the top-ranked measures for both object-oriented software systems and nonobject-oriented software systems; because the measures have strengths and weaknesses that depend on the phase of software development, the rankings were obtained for four phases of development: (i) requirements, (ii) design, (iii) implementation, and (iv) testing. In addition, families, which relate to the primary attributes of a measure, such as functional size, were also ranked, as for the measures.

3.2.2 Lawrence Livermore National Laboratory Research

The Lawrence Livermore National Laboratory, Fission Energy and Systems Safety Program, produced a series of reports for the NRC related to software reliability and the use of software in safety systems. The following briefly summarizes significant elements of this body of work:

- Techniques, Processes, and Measures for Software Safety and Reliability (Sparkman, 1992) reviews domestic and international standards related to software reliability and safety and identifies their similarities and differences. The field of software reliability is reviewed with a conclusion that two elements are required for highly reliable software: (i) good software engineering practices based in a life-cycle approach and (ii) assessing the reliability of software using reliability measurement techniques. Good practices for software engineering include using plans for software quality assurance, configuration management, verification, and validation; using coding standards; performing technical reviews and audits; using well-defined error-reporting procedures; employing risk management; and conducting safety analyses. Several groups of standards are identified. Those standards focusing on safety planning and analysis include IEEE 1228, MIL–STD–882B, MoD 00–55; those focusing on quality assurance include DoD 2167A and DoD 2168, FAA 13B, 16A and 18A, IEC 65A(Secretariat)122, IEC 880, and MoD 00–55; those that suggest schemes for relating hazard severity to software integrity levels include: MIL–STD–882B, IEC 65A(Secretariat)122, and MoD 00–55 and MoD 00-56; two standards identifying software reliability measures and application guidance are IEEE 982.1 and IEEE 982.2.

- Survey of Industry Methods for Producing Highly Reliable Software (Lawrence and Warren, 1994) presents the result of an effort to identify current methods for software reliability by surveying a group of companies and organizations using or developing such methods. Methods were sought for measurement, estimation, error detection, and prediction of software reliability, which can be used during the software life cycle as an input to a risk assessment for nuclear power reactor safety systems. The companies surveyed included Computer Sciences Corporation, International Business Machines, Federal Systems Company, and Thompson Ramo Wooldridge. The organizations surveyed included National Aeronautics and Space Administration Software Engineering Lab/University of Maryland and American Institute of Aeronautics and Astronautics Software Reliability Project. The results of the interviews and discussion with the various entities are reported; however, no evaluation of this material is provided.

- Reviewing Real-Time Performance of Nuclear Reactor Safety Systems (Preckshot, 1993) develops guidance for the performance review of real-time computer systems used in nuclear powerplant safety systems. Three aspects of guidance are considered: (i) prototyping, (ii) timing analysis, and (iii) methods of estimating reliability of systems. This guidance was developed to avoid problems in this type of safety-critical software, such as late delivery of the system, operation that is too slow to perform critical safety functions, and defective software that cannot be economically rectified. Recommended prototypes include human factors, early prototypes to resolve technical questions, and validation prototypes to estimate software performance. Timing analysis should consider upper and lower bounds for the time of code execution, so that the software will behave predictably. In addition, code execution and

communication times should be demonstrated to be correct and within limits. Final performance should be consistent with earlier indications from prototyping. Although the body of the report is short, two large appendices provide detail on methods of timing analysis, scheduling real-time computations, prototyping, real-time software development approaches, modeling and measurement, and real-time operating systems.

- Software Reliability and Safety in Nuclear Reactor Protection Systems (Lawrence, 1993) reviews software reliability methods and discusses the software life cycle and how actions taken during the life cycle may enhance the reliability of the resulting software product. Two important points made in the report include (i) software reliability should not be a separate activity from the reliability of the rest of the system (hardware and humans) and (ii) assuring the safety and reliability of software must be accomplished by activities during all phases of the life cycle. Reliability-enhancing activities for various phases of the software life cycle are identified and discussed.

- Software Safety Hazard Analysis (Lawrence, 1995b) proposes a method for performing software safety hazard analysis, based on an evaluation of published methods, approaches, and standards. The approach to software safety hazard analysis retains the generic goals of hazard analysis: (i) encouragement of design changes to reduce or eliminate hazards and (ii) performance of analyses and tests to enhance confidence in the reliability of critical system elements. The methodology focuses on the potential effects on system hazards from the correct functioning of the software, as well as the incorrect functioning of the software. Methods for estimating the reliability of software are not considered part of the methodology, but an extensive appendix reviews tools for software reliability analysis. An important point articulated is that the "software hazard analysis should be performed within the context of the overall system design." Two aspects of the overall system are important for analyzing software hazards: (i) those attributes generated by the safety requirements for the overall system (e.g., limits on public doses offsite at the repository) and (ii) those attributes generated by the primary mission of the overall system that could affect safety (e.g., the disposal of waste for the repository). The analysis must consider the role that software plays in accomplishing both the system safety function and its overall mission, through system control and monitoring functions. A description of the relationship between software safety analysis and overall system safety analysis is provided in IEEE Standard 1228.

- An Overview of Software Safety Standards (Lawrence, 1995a) is a short paper providing an overview of software safety standards available at the time the paper was written. No claim is made that the compilation is complete. An interesting aspect of the paper is the following list of organizations developing software safety standards:

  —Multidisciplinary societies
  - –British Computer Society
  - –International Electrotechnical Commission

–Institute of Electrical Engineers
–Institute of Electrical and Electronics Engineers

—Government organizations
–Canadian Standards Association
–Deutsches Institut fur Nomrung e.V.
–United States Department of Defense
–United Kingdom Ministry of Defence
–National Aeronautics and Space Administration

—Industry sector societies
–American Nuclear Society
–American Society of Mechanical Engineers
–Electronics Industry Association
–Instrument Society of America
–Requirements and Technical Concepts for Aviation
–Underwriter's Laboratories

3.2.3 National Institute of Standards and Technology Research

In the early 1990s the NRC sponsored some work by the National Institute of Standards and Technology. One product of this research (Wallace, 1992) is a study of standards, draft standards, and guidelines establishing requirements to assure the safety and quality of software operated in association with safety systems in nuclear powerplants. This study (i) articulated attributes of a standard needed to provide assurance of safety, (ii) found a high degree of variability in the requirements and precision of existing standards, and (iii) recommended that a standard be developed specifically for software deployed in nuclearpower plant safety systems. A second product by the National Institute of Standards and Technology (Wallace, 1994) was the proceedings of a workshop held jointly by the NRC and the National Institute of Standards and Technology on September 13–14, 1993, entitled, Digital Systems Reliability and Nuclear Safety. The workshop had participants from industry, including the nuclear power industry, academia, and the regulatory community. Topics discussed at this workshop included possible inadequacies in current software engineering practices for safety-critical systems, methods for reducing risk in safety-critical software systems, proposed regulatory positions for nuclear powerplant systems, and proposed research associated with the use of digital systems in nuclear powerplants.

4 DOE YUCCA MOUNTAIN PROJECT APPROACHES TO SOFTWARE RELIABILITY

The DOE Yucca Mountain project has described some plans for software utilization during preclosure operations in the document Instrumentation and Controls for Waste Emplacement (CRWMS M&O, 2000a). Although the document does not provide complete descriptions of the repository preclosure safety operations (e.g., it only covers the Waste Emplacement/Retrieval System) or the plans for deployment of computer-controlled equipment, some important and interesting aspects of the planned use of software in safety-related repository operations are provided.

Section 4.2 of this Department of Energy (DOE) report provides a list of design criteria for the Waste Emplacement/Retrieval System. These criteria are as follows (CRWMS M&O, 2000a):

- 4.2.1—The waste system shall ensure that the possibility of an uncontrolled descent down the North or South Ramp of system equipment carrying a waste package is limited to less than 1×10^{-6} events/year (Criterion 1.2.2.1.1).

- 4.2.2—The structures, systems, and components important to safety shall be designed to permit prompt termination of operations and maintain waste packages in a safe and sustainable position during an emergency (Criterion 1.2.2.1.5).

- 4.2.3—The waste emplacement system shall be designed in accordance with the Project as low as is reasonably achievable program goals (TBD–406) and the applicable guidelines in "Information Relevant to Ensuring that Occupational Radiation Exposures at Nuclear Power Stations Will Be As Low As Is Reasonably Achievable" (Regulatory Guide 8.8) (Criterion 1.2.2.1.9).

- 4.2.4—The system shall receive electrical power from the Subsurface Emplacement Transportation System (Criterion 1.2.4.11).

- 4.2.5—The system shall receive and provide the operational information, status, and control data as outlined in the following Table 4-1 to the Monitored Geologic Repository Operations Monitoring and Control System (Criterion 1.2.4.13).

- 4.2.6—The system shall include provisions for the inspection, testing, and maintenance of system equipment (Criterion 1.2.5.1).

- 4.2.7—The inherent availability for the system shall be greater than 0.9485 (Criterion 1.2.5.2).

The numbered criteria listed previously refer to Waste Emplacement/Retrieval System Description document (CRWMS M&O, 2000c).

Table 4-1. System Inputs/Outputs (CRWMS M&O, 2000a)	
Inputs	**Outputs**
Radiation monitoring data and status	Equipment status and status of operations
Subsurface electrical distribution system data and status monitoring	Equipment alarm status
Subsurface fire suppression system data and status monitoring	Control equipment, status, and alarms
Waste package identification and tracking data	Interlock status
Operational message advisory	Video signals
Activity plans and procedures	Communications equipment status
Emergency response commands	Timeout warnings for handling equipment
Mined geologic repository operational alarm status	Control loads left in improper states (suspended loads, unattended controls)
Remote control of system equipment	Control equipment, status, and alarms

These design criteria raise some concerns regarding the DOE approach to software reliability and safety in this context

1. Criterion 4.2.4 requires that electrical power be received from the Subsurface Emplacement Transportation System. Safety may be enhanced by requiring the computer system, which may have safety-critical functions, to be powered by a source independent of that used to power the Subsurface Emplacement Transportation System. By introducing this increased redundancy, certain safety-critical monitoring and control functions can continue, even if the main power is lost.

2. Criterion 4.2.7 requires an overall availability for the Waste Emplacement/Retrieval System. This is a system operability requirement, which is not necessarily related to safety; e.g., requiring that an assembly line operate 95 percent of the time does not necessarily assure safety of the workers or the product and may compromise safety. Consideration should be given to demonstrating that achievement of this criterion will not compromise the safety of the repository operations.

The Instrumentation and Controls for Waste Emplacement document (CRWMS M&O, 2000a) lists a set of standards that will be used or adhered to for this part of the repository. The list is comprised solely of Institute of Electrical and Electronics Engineers standards.

- ANSI/IEEE Std 352–1987. Institute of Electrical and Electronics Engineers Guide for General Principles of Reliability Analysis of Nuclear Power Generating Station Safety Systems.

- ANSI/IEEE Std 577–1976. Institute of Electrical and Electronics Engineers Standard Requirements for Reliability Analysis in the Design and Operation of Safety Systems for Nuclear Power Generating Stations.

- ANSI/IEEE Std 1008–1987. Institute of Electrical and Electronics Engineers Standard for Software Unit Testing.

- IEEE Std 7–4.3.2–1993. Institute of Electrical and Electronics Engineers Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations.

- IEEE Std 379–1994. Institute of Electrical and Electronics Engineers Standard Application of the Single-Failure Criterion to Nuclear Power Generating Station Safety Systems.

- IEEE Std 603–1998. Institute of Electrical and Electronics Engineers Standard Criteria for Safety Systems for Nuclear Power Generating Stations.

- IEEE Std 730–1998. Institute of Electrical and Electronics Engineers Standard for Software Quality Assurance Plans.

- IEEE Std 828–1998. Institute of Electrical and Electronics Engineers Standard for Software Configuration Management Plans.

- IEEE Std 829–1998. Institute of Electrical and Electronics Engineers Standard for Software Test Documentation.

- IEEE Std 1028–1997. Institute of Electrical and Electronics Engineers Standard for Software Reviews.

Some comments on the selection of these standards by DOE for repository operations include

1. The criteria for selection of these standards has not been stated; statement of the criteria are an important aspect of judging the adequacy of the standards selected.

2. A primary goal using a set of standards in this context is to assure safe operation of the repository; a secondary, but very important goal, is demonstrating to the NRC with reasonable assurance that the repository will be safely operated. It is not clear that this subset of standards will assure that these important goals are met.

3. Guidance issued by NRC for software used in nuclear safety applications (mostly nuclear power plant applications) endorses certain industry standards, in many cases with exceptions. This DOE selection of standards is different and does not address the exceptions noted in NRC guidance (e.g., R.G. 1.152 endorses IEEE Standard 7-4.3.2-1993 with exceptions). It may be important to resolve the inconsistency between the standards adopted by DOE and those adopted with exceptions by NRC.

4. Some standards adopted by NRC for software used in nuclear safety applications are not among those adopted by DOE for the repository. It may be important to determine whether these omitted standards should be included by DOE to assure safe operation.

Section 6 of the DOE report (CRWMS M&O, 2000a) provides a broad description of the possible use of computer automation for emplacement operations. No definitive statements are made regarding the degree of automation for emplacement operations, but an entire range of options from a little to complete automation is discussed. In those cases in which automation would be increased, more sensing devices, such as position sensors, would need to be included in the system design. In those cases in which automation would be lessened, remote observation and control devices (e.g., television cameras and monitors and joysticks to control transporters) would need to be included in the system design. In addition to emplacement operations, digital control systems are expected to operate other safety-related functions, such as surface waste handling system, subsurface ventilation system, subsurface transportation system, power distribution system, repository environmental (temperature, humidity) monitoring, radiation monitoring systems, fire detection and suppression systems, personnel and process monitoring systems, emergency response, security and access control systems, warning and alarm systems (CRWMS M&O, 2000a).

The DOE has taken a cautious approach to adopting digital technology for preclosure repository operations. Although the advantages of digital control systems are recognized, so are their potential liabilities. The report by the National Research Council (1997) prepared for the NRC is cited for some of this cautious attitude. Four issues articulated in the National Research Council report are discussed by DOE

- Systems Aspects of Digital Instrumentation and Control
- Software Quality Assurance
- Common-Mode Software Failure Potential
- Safety and Reliability Assessment Methods

The concern stated regarding the first two issues is that suitable programs for (i) system integration and (ii) software quality assurance may be so costly and require so much time to accomplish, that the use of digital systems for repository preclosure operations may be impractical. An approach to resolving the vulnerabilities that may arise, when a single computer controls many repository functions, is to use a defense-in-depth strategy (CRWMS M&O, 2000a).

> Defense-in-depth design strategies include use of redundant systems, use of diverse technologies and systems, physical separation of redundant or backup systems, and incorporation of fault-tolerant design features.

Most of the defense-in-depth strategies cited by DOE are more applicable to hardware than software systems. Only fault-tolerant (fail-safe) design is commonly applied, while introduction of true redundancies in software has proven to be difficult. The DOE also states that another issue with adopting digital controls systems is that use of commercial off-the-shelf hardware and software is attractive only if a suitable and cost-effective process can be formulated for the technology's dedication (i.e., approval and acceptance) by the NRC (CRWMS M&O, 2000a). The approach adopted is to assume that the qualification process will preserve a significant part of the cost advantages of using off-the-shelf hardware and software. The report briefly

discusses NRC involvement in software reliability and cites many of the Lawrence Livermore National Laboratory documents described in Section 4.2.2; however, the NRC Regulatory Guides on this topic do not seem to be acknowledged.

Section 6.5 of the report (CRWMS M&O, 2000a) provides some detail for the instrumentation, controllers, and communication devices being considered for the repository. The DOE has undertaken survey of available equipment and practices and includes a broad range of devices and system sophistication. Among the devices considered were programmable logic controllers; multiple, distributed, and embedded microcontrollers, and microprocessing units. Although some detail included in Section 6.5 is informative, the level of detail is conceptual and mainly addresses the functions of various subsystems. The nature of the components comprising the subsystems, much less the software systems that control and connect the subsystems, are not specified because these design choices have not yet been made.

5 SOFTWARE RELIABILITY IN THE MINING INDUSTRY

There are significant differences between conventional mining operations and preclosure operations at the proposed Yucca Mountain repository. The substantial investigations of software reliability and safety in mining operations should be considered in the evaluation of safety in preclosure operations, especially those underground. Especially pertinent in these evaluations of software reliability and mining safety is the consideration of computer-controlled heavy machinery in operations in which human life is at risk.

Extensive work has been performed on computer reliability and mine safety as a joint effort of the Mine Safety and Health Administration and National Institute for Occupational Safety and Health (Dransite, 2000; Sammarco et al., 1997, 2001). A major product of these activities has been the issuance of the report, Programmable Electronic Mining Systems: Best Practice Recommendations (in Nine Parts), which has developed recommendations for safety of processor-controlled mining equipment. The audience for these recommendations includes mining companies, original equipment manufacturers, and aftermarket suppliers to mining companies. The major elements of these recommendations include the following:

- Introduction to Safety—This introductory discussion for the benefit of the mining industry presents the basic concepts of system and software safety, indicates the need within the mining industry to address the safety of programmable electronic controllers used for mining equipment, and asserts the benefits of implementing a program of system and software safety.

- System Safety—This discussion draws heavily on IEC Standard 61508 (International Electrotechnical Commission, 1998) and other standards to develop a set of recommendations for the safe, life-cycle development of both surface and underground mining systems that use embedded programmable electronic devices, both networked and non-networked.

- Software Safety—This discussion draws heavily on IEC Standard 61508 (International Electrotechnical Commission, 1998) and other standards to develop a set of recommendations for the safe, life-cycle development of the software subsystem of mining systems using programmable electronic devices.

- Safety File—This discussion describes the documentation required to demonstrate that the system will meet a prescribed level of safety for the intended application. This documentation identifies the limitations on the safe use of the system, is begun at the design stage, and is maintained throughout the entire life-cycle of the equipment, thereby providing the administrative records for the safety program.

- Safety Assessment—These recommendations provide for an independent evaluation of the information contained in the Safety File, to determine its completeness, suitability, and justification.

- Safety Framework Guidance—These recommendations provide a lower level of guidance that is more application oriented. The document reinforces safety and analytical concepts, describes a variety of methods that are applicable, provides

examples, and references. Although this guidance does not claim to be exhaustive in its content, it does provide references to additional material that can be used, depending on the capabilities of the user and the issues confronted.

This guidance is voluntary guidance developed by these government agencies, because a nonregulatory, cooperative approach was chosen.

The issue of software safety arose in the mining industry in the late 1980s. More mining equipment with electronic programmable controllers, was being marketed and used as the practice of longwall mining grew (Dransite, 2000). In a longwall mining operation, which is highly automated, coal is cut from the face of a seam and carried away by a conveyor system. While the coal is being cut and removed, a set of hydraulically operated shields support the mine roof that is over the equipment and miners. As the mine face is completely mined, the hydraulic roof supports are sequentially advanced. This complex operation, Shearer-Initiated Roof Support Advance is controlled by programmable electronics. When the support is removed from sections of the roof, it subsides onto the mine floor. Clearly, the advance of the roof supports is a safety-critical operation. Industry and government agencies received many reports of accidents and near misses involving or attributable to the programmable electronics. The term ghosting was commonly used to depict the unexpected behavior of the electronically controlled devices. Although the new mining machinery provided considerable improvements in the safety of the working environment for the miners, the introduction of these electronic devices also introduced new and unfamiliar hazards. The following quote from Dransite (Dransite, 2000) graphically illustrates the type of safety-critical, life-threatening problems caused by a combination of malfunctioning equipment, improper maintenance, and inadequate operator training:

> A shield unexpectedly lowered and injured a face foreman (foreman controlling operations at mine faces) at a longwall mine. There were also complaints from the same mine that shields were "ghosting" and creating a serious personnel safety hazard. The problems were found to be due primarily to failure of the mine operator to conduct timely maintenance. Additionally, operator training was inadequate for following proper operational procedures and sequences, and in providing guidance for recognizing improper system operation. In the above mine example, 24 shields out of 186 had defective magnetic position transducers. These transducers signaled the controller that a shield had properly advanced. Without a proper advance signal from the transducer, the system was programmed to attempt four advance retries and then go into a drop and drag mode where the shield canopy was lowered and the shield dragged into position by the face conveyor advance. This advance attempt would occur even if a shield had properly advanced. With so many shields giving errant advance indications, the PE (programmable electronic) controller fell minutes behind in attempting the shield advances. Someone on or near a shield undergoing this delayed advance attempt would perceive the movement to be ghosting. Additionally, the movement could pose a physical injury threat to someone positioned on the shield. Further compounding the problems, the operators were inputting manual commands to advance the face conveyor when it was still under automatic control. This produced additional unplanned movement when the automatic conveyor advance was later executed.

Additional incidents and accidents described by Dransite (2000) include the following graphic illustrations of dangerous interactions between heavy machinery, electronically programmable controllers, and human operators.

- Problem: Shield advances out of sequence in the automatic mode, appearing to be ghosting
  Cause: Inadequate power supply current capacity

- Problem: Unexpected shield advance causes foot injury
  Cause: Lack of maintenance and a lack of understanding of system programming

- Problem: Foreman lost all toes of right foot while attempting to pass through the active shield advance area
  Cause: Failure to follow proper procedure by pausing the automatic system for safe passage

- Problem: Operator pinned by unexpected shield advance
  Cause: Sticking hydraulic valve

- Problem: System Emergency Stop Function worked intermittently
  Cause: A firmware change caused pulse width modulation of the drive signal to motor valves controlling the shields; this change allowed a 100-microsecond window in which an emergency stop command would not be executed, if the controller found the motor valve signal in an off state

- Problem: Unplanned shield movement
  Cause: An intermittent hardware fault in the shearer erroneous location information to be transmitted from the shearer controller to the shield advance system controller. In addition, a programming change in the shield advance system controller inadvertently deleted some code that rejected shearer location information outside reasonable parameters

- Problem: An unanticipated lowering of the shield canopy caused head and neck injury to a shearer operator
  Cause: Lack of maintenance on a defective position transducer caused the shield to go into a programmed drop and drag mode under this controller input condition

Overall, four areas have been identified for improvement in automated machinery used in mines. They include: (i) operator training, (ii) timely maintenance, (iii) maintaining the integrity of enclosure sealing, and (iv) being alert to detect abnormal operating sequences, which might indicate software problems.

An important conclusion from this literature is that significant effort has been applied to the interaction of humans, heavy machinery, and electronic controllers, including their software. However, the basic standards adopted by this industry and government collaboration are different from those adopted by DOE or NRC. The key standards used as the basis for the National Institute of Occupational Safety and Health/Mine Safety and Health Administration recommendations (Sammarco, et al., 2001) are

- IEC 61508 Parts 1–7, Functional Safety: Safety Related Systems (International Electrotechnical Commission, 1998)
- MIL–STD–882C, System Safety Program Requirements (U.S. Military Standard, 1993)
- Software in Programmable Components (Underwriter Laboratories, 1998)

This raises the issue of the compatibility of these standards for heavy equipment controllers, which may be applicable to the Yucca Mountain operations, with the set of standards adopted by DOE, and with the set of standards adopted, with exceptions, by the NRC staff.

6 CONCLUSIONS, RECOMMENDATIONS, AND ISSUES

6.1 Conclusions

Specific, definitive conclusions about software reliability and (i) its role in preclosure repository safety and (ii) how it should be incorporated into the analysis of preclosure safety require additional investigation, further development of the PCSA Tool (Dasgupta, et al., 2001) and, most importantly, further details about repository operations from DOE. A few general conclusions can be made.

- Software reliability, particularly in the context of safety-critical systems, has received considerable study over the past 30 to 40 years. There is no need to develop new software reliability methodologies for specific application to the problems of repository preclosure safety. Instead, an appropriate method(s) needs to be selected for use in each phase of repository development and each review of the safety analysis. Because of the large amount of information available on this subject, selection of a methodology based on a logical rationale is not likely to be simple, straightforward, or effortless.

- Software reliability could be a significant aspect of preclosure repository safety and should be addressed in the safety analysis review.

- Before a quantitative capability for software reliability is incorporated into the PCSA Tool, further evaluation is needed to assure that (i) quantification will provide meaningful results; (ii) the capability will be compatible with the remainder of the analysis; and (iii) an integrated approach to safety evaluation will be achieved.

- Because software reliability has strong interactions with the hardware system and human operators, an integrated analysis for safety evaluation is needed.

- The effort and detail of the software reliability analysis used as part of the PCSA Tool should be consistent with the level of detail in the remaining aspects of the analysis and the availability of information about the repository design and operational characteristics, especially the hardware, software, and human components.

- The level of detail that appears to be available from DOE on the use of electronic programmable controllers and computer systems, including the embedded software systems, is insufficient to provide a basis for incorporating a quantitative capability into the PCSA Tool. Furthermore, the level of detail makes a qualitative evaluation of the safety aspects of the software for preclosure operations infeasible, except at a generic and superficial level.

- The NRC has been among the leaders in applying software reliability analysis methods to problems of nuclear safety; adoption of methodologies developed or used by the NRC is advocated for the benefits that include: ready acceptance by the regulatory staff, NRC staff familiarity and experience in use of these methods, and familiarity and acceptance by the broader nuclear safety community. Currently, the NRC staff have not considered any methodology for quantifying software reliability to be sufficiently accurate to use quantification as the sole determinant of system safety.

- The NRC has issued a large body of guidance related to software reliability analysis and procedures for enhancing software reliability. The NRC staff should determine which part of this guidance, if any, may be applicable to preclosure safety. Specifying which guidance is applicable will help to constrain the issues that need to be addressed by software reliability analysis incorporated into the PCSA Tool.

6.2 Recommendations for Further Study

- Further evaluation of various software reliability models may be warranted as more detail regarding the hardware, software, and operations of the repository become available. With more detail available, possibly including failure data on the specific software to be deployed in the repository, an evaluation could be made to determine whether it is advisable to attempt quantification of the software reliability and if quantification is attempted, what specific models would be most appropriate.

- This report skimmed the surface on the literature, methods, and procedures for software reliability in the context of safety-critical systems. It would appear that these methods are extremely pertinent to preclosure repository safety. More investigation of these methods should be directed at enhancing the NRC/CNWRA review capability and providing tools for identifying vulnerabilities in DOE approaches and designs.

- Because software reliability analysis and software reliability in the context of safety-critical systems are subjects of current research, a modest effort should be devoted to monitoring developments in this dynamic field to assure that the most appropriate methods are available to evaluate repository safety.

- It is premature to attempt to put a quantitative capability into the CNWRA PCSA Tool. Further work is advised to develop a qualitative capability; in addition, an approach to quantifying a software reliability in the PCSA Tool should be investigated.

- Further evaluation of DOE documents related to software reliability and repository design and operations is advised.

- Additional investigation of experiences in the mining industry with software reliability is recommended, because this industry appears to be a good analogue for some repository operations and systems.

6.3 Issues Arising from this Study

6.3.1 Technical and Policy Issues for Consideration by the NRC Staff

The information reviewed in this initial survey raises some technical/policy issues that the NRC staff may wish to address.

- The NRC staff may wish to consider whether additional guidance should be articulated for software systems used in safety critical aspects of the repository. In particular, the existing regulatory guides related to software reliability are designed for nuclear power plants and, in some cases, gaseous diffusion plants; it is not clear that this guidance is suitable or sufficient for repository operations.

- The NRC guidance, the guidance ascribed to DOE, and the guidance recommended by the mining industry are not the same. The NRC may wish to consider to what degree these different guidance documents and principles should be reconciled.

6.3.2 Informational Issues Related to DOE

The information reviewed in this initial survey appears to identify some information needs to related to the DOE designs and plans.

6.3.2.1 Questions about DOE Operational Plans for the Repository

Some of these information needs relate to DOE operational plans.

- More mature approaches to software safety advocate an integrated approach that includes hardware, software, and human components of the system. DOE documents indicate, correctly, that there is some flexibility in assigning functions to humans or to automated systems with embedded software. It may be important for DOE to articulate how these assignments will be made, what principles or rationales will be used to make them, and the schedule for making these assignments.

- It may be important for DOE to articulate whether monitoring and control functions during the preclosure phase of repository development be centralized or distributed.

6.3.2.2 Questions about DOE Designs.

Some of these information needs relate to DOE designs.

- DOE should indicate what measures it employs to assure that software reliability considerations are being incorporated into the design of preclosure repository facilities. DOE should indicate how it is assuring that repository designs reflect good software safety principles, including: (i) an integrated approach incorporating hardware, software, and human performance considerations, (ii) a fail-safe approach for safety-critical software, (iii) a quantification of safety-related software reliability to demonstrate compliance with the applicable regulations, and (iv) a quantification of safety-related software reliability to demonstrate safe design and operation.

- DOE may wish to replace some of the current vagueness regarding the proposed software systems for the repository preclosure operations. Some of the information needed includes: (i) what is the software; (ii) what functions does it perform; (iii) what process was used to control the development of the software; (iv) what is the hardware; (v) what is the firmware; (vi) what is the operating system, if any; (vii) what is the history of faults and failures for this software; (viii) what is the revision history of the software; (ix) what failsafe protocols or requirements were used in developing the software; (x) what are the human, instrumentation, and communication interfaces for the software system.

6.3.2.3 Information Needs Related to Questions About How DOE Will Incorporate Software Reliability Analysis Considerations Into Operational Planning and Facility Design

Some of these information needs relate to how DOE will incorporate SRA considerations into operational planning and facility design.

- DOE may wish to articulate what strategy it will use to develop preclosure designs and operational plans in a consistent, unified fashion. In particular, DOE may wish to state what strategy will be used to integrate hardware, software, and human elements, for both design and operations.

- If operational planning continues to lag behind facility design, DOE may wish to consider and state how it will demonstrate that designs meet the applicable regulatory requirements and assure that software requirements and operational characteristics have sufficient reliability and safety features to compensate for potential human error during operations.

7 REFERENCES

American National Standards Institute/Institute of Electrical and Electronics Engineers. *Standard Glossary of Software Engineering Terminology.* Washington, DC: American National Standards Institute/Institute of Electrical and Electronics Engineers. 1991.

Bowen, W.M and C.A. Bannett, eds. NUREG/CR-4604, "Statistical Method for Nuclear Material Management." Washington, DC: NRC. December 1988.

Butler, R.W. and G.B. Finelli. "The Infeasibility of Experimental Quantifications of Life Critical Software Reliability." ACM SIGSOFT'91. Conference on Software Critical Systems, New Orleans, December 4–6, 1991. New York: Association for Computing Machinery. 1991.

Butler, R.W. and G.B. Finelli. "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software." Institute of Electrical and Electronics Engineers Transactions on Software Engineering. Vol. 19, No. 1. pp 3–12. 1993.

Butler, R.W. and S.C. Johnson. "Formal Methods for Life-Critical Software." AIAA Computing in Aerospace 9 Conference, San Diego, California, October 19–21, 1993. pp. 319–29. Washington: American Institute for Aeronautics and Astronautics. 1993.

Civilian Radioactive Waste Management System Management and Operating Contractor. "Instrumentation and Controls for Waste Emplacement." ANL–WES–CS–000001. Revision 00. Las Vegas, Nevada: TRW Environmental Safety Systems Inc. 2000a.

———. "Preliminary Preclosure Safety Assessment for Monitored Geologic Repository—Site Recommendation." TDR–MGR–SE–000009. Revision 00. ICN 01. Las Vegas, Nevada: TRW Environmental Safety Systems Inc. 2000b.

———. "Waste Emplacement/Retrieval System Description Document." SDD–WES–SE–000001. Revision 00. Volume I. Las Vegas, Nevada: CRWMS M&O. 2000c.

Collins, W.R., K.W. Miller, B.J. Spielman, P. Wherry. "How Good is Good Enough? An Ethical Analysis of Software Construction and Use." *Communications of the ACM.* Vol. 37, No. 1. pp. 81–91. 1994.

Dasgupta, B., D. Daruwalla, R. Benke, and A.H. Chowdhury. "Development of a Tool and Review Methodology for Assessment of Preclosure Safety Analysis Progress Report." CNWRA. San Antonio, Texas: Center for Nuclear Waste Regulatory Analyses. 2000.

Dasgupta, B., D. Daruwalla, R. Benke, A.H. Chowdhury, and B. Jagannath. "Methodology for Assessment of Preclosure Safety of Yucca Mountain Project." Proceedings of the Ninth International Conference on High-Level Radioactive Waste Management, 2001.

Dransite, G.D. "System Safety Applications in Mining." 18th International System Safety Conference, September 15, 2000. Fort Worth, Texas. 2000.

Friedman, M.A. and J.M. Voas. *Software Assessment: Reliability, Safety, Testability.* New York City, New York: John Wiley & Sons, Inc. 1995.

International Electrotechnical Commission. "Functional Safety: Safety-Related Systems." Draft IEC–61508, Parts 1–7, Version 4. International Electrotechnical Commission. May 12, 1998.

Institute of Electrical and Electronics Engineers. "Standard for Software Safety Plans." IEEE 1228. New York: Institute of Electronic and Electrical Engineers. 1994.

Keene, S.J. "Comparing Hardware and Software Reliability." *Reliability Review.* Vol. 14, No.4. pp. 5–7. December 1994.

Knight, J.C. and B. Littlewood. "Critical Task of Writing Dependable Software." Institute of Electrical and Electronics Engineers Software. pp. 16–20. 1994.

Lawrence, J.D. NUREG/CR–6101."Software Reliability and Safety in Nuclear Reactor Protection Systems." Washington, DC: June 11, 1993.

Lawrence, J.D. and L.P. Warren. NUREG/CR/6278, "Survey of Industry Methods for Producing Highly Reliable Software." Washington, DC: August 29, 1994.

Lawrence, J.D. NUREG/CP–0145, "Workshop on Developing Safe Software." Washington, DC: NRC. August 30, 1994.

Lawrence, J.D. "An Overview of Software Safety Standards." UCRL–JC–122249. Livermore, California: FESSP, Lawrence Livermore National Laboratory. 1995a.

Lawrence, J.D. NUREG/CR–6430. "Software Safety Hazard Analysis." Version 2.0. Washington, DC: NRC. October 1995b.

Lawrence, J.D., W.L. Parsons, A. Sicherman, and G.L. Johnson. "Assessment of Software Reliability Measurement Methods for Use in Probabilistic Risk Assessment." Version 1. UCRL–ID–136035. Livermore, California: FESSP, Lawrence Livermore National Laboratory. 1998.

Leveson, N. *Safeware: System Safety and Computers.* New York City, New York: Addison Wesley. 1995.

Leveson, N., L. Alfaro, T. Alvarado, M. Brown, E.B. Hunt, M. Jaffe, S. Joslyn, D. Tinner, J. Reese, J. Samarzya, S. Sandys, A. Shaw, and Z. Zabinsky. "Demonstration of a Safety Analysis on a Complex System." Software Engineering Laboratory Workshop, NASA Goddard, December 1997.

Leveson, N.G. "Software Safety in Embedded Computer Systems." Communications of the ACM. Vol. 34, No. 2. pp. 35–46. 1991.

Musa, J.D. and K. Okumoto. "Software Reliability Models: Concepts, Classification, Comparisons, and Practice". *Electronic Systems Effectiveness and Life Cycle Costing*. J. K. Skwirzynski, ed. Series F3 pp. 395–424. Heidelberg: Springer-Verlag: Nato AFI 1983.

Musa, J.D. *Software Reliability Engineering*. New York: McGraw-Hill. 1998.

National Research Council. "Digital Instrumentation and Control Systems in Nuclear Power Plants, Safety and Reliability Issues." TIC 235537. Washington, D.C.: National Academy Press. 1997.

National Institute of Standards and Technology. "Review of Software Hazard Analyses." Draft. Washington, DC: National Institute of Standards and Technology. June 4 1993.

NRC. NUREG-1489. "A Review of NRC Staff Uses of Probabilistic Risk Assessment." Washington, DC: U.S. Nuclear Regulatory Commission. March 1994.

Pan, J. Software Reliability. Student Paper, Carnegie-Mellon University, Topics in Dependable Embedded Systems, Spring 1999. Also: Software Reliability http://www.cs.cmu.edu/~koopman/des_s99/sw_reliability/index.html .

Preckshot, G.G. "Reviewing Real-Time Performance of Nuclear Reactor Safety Systems." Washington, DC: NRC. NUREG/CR–6083, May 28, 1993.

———. NUREG/CR–6083, "Reviewing Real-Time Performance of Nuclear Reactor Safety Systems." Washington, DC: NRC. May 28, 1993.

Reese, J.D. and N.G. Leveson. "Software Deviation Analysis: A Safeware Technique." AIChE 31st Annual Loss Prevention Symposium, Houston, Texas. New York: American Institute of Chemical Engineers. March 1997.

Sammarco, J.J., J.L. Kohler, T. Novak, and L.A. Morley. "Safety Issues and the Use of Software-Controlled Equipment in the Mining Industry." Institute of Electrical and Electronics Engineers Industry Applications Society Annual Meeting, New Orleans, Louisiana, October 5–9. 1997.

Sammarco, J.J., T.J. Fisher, J.H. Welsh, and M.J. Pazuchanics. "Programmable Electronic Mining Systems: Best Practice Recommendations (In Nine Parts)." DHHS-(NIOSH) Publication No. 2001–132. Pittsburgh, PA: National Institute for Occupational Safety and Health, Pittsburgh Research Laboratory. April 2001.

Smidts, C. and M. Li. NUREG/GR–0019. "Software Engineering Measures for Predicting Software Reliability in Safety Critical Digital Systems." Washington, DC: NRC. August 8, 2000.

Smith, D.J. *Reliability, Maintainability and Risk*. 5th Edition. Oxford: Butterworth-Heinemann. 1997.

Smith, R.W. "Safety and Reliability." <http://www.computing.edu.au/~roger/sr_html/sr_html.shtml> August 3, 1999.

Sparkman, D. Techniques, Processes, and Measures for Software Safety and Reliability. UCRL–ID 108725. Livermore, CA: FESSP, Lawrence Livermore National Laboratory. May 30, 1992.

Tian, J. "Reliability Measurement, Analysis, and Improvement for Large Software Systems." *Advances in Computers*. Volume 46, pp. 159–235. 1998.

Underwriter Laboratories. "Software in Programmable Components." UL–1998, 2nd Edition. Northbrook, Illinois: Underwriter Laboratories. May 1998.

U.S. Military Standard. "System Safety Program Requirements." MIL–STD–882C. 1993.

Wallace, D.R., B.B. Cuthill, L.M. Ippolito, and L. Beltracchi. *Proceedings of the Digital Systems Reliability and Nuclear Safety Workshop*. NUREG/CP–0136 (also NIST SP 500–216). Washington, DC: Nuclear Regulatory Commission. 1994.