BROOKHAVEN NATIONAL LABORATORY

BNL-94047-2010

REVIEW OF QUANTITATIVE SOFTWARE RELIABILITY METHODS

Tsong-Lun Chu, Meng Yue, Gerardo Martinez-Guridi, and John Lehner

Brookhaven National Laboratory

Letter Report

Digital System Software PRA

JCN N-6725

September 2010

Prepared for

U.S. Nuclear Regulatory Commission
Office of Nuclear Regulatory Research
Division of Risk Analysis

DISCLAIMER

TABLE OF CONTENTS

Section **Page**

TABLE OF CONTENTS (CONT'D)

Section **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

EXECUTIVE SUMMARY

The current U.S. Nuclear Regulatory Commission (NRC) licensing process for digital systems rests on deterministic engineering criteria. In its 1995 probabilistic risk assessment (PRA) policy statement, the Commission encouraged the use of PRA technology in all regulatory matters to the extent supported by the state-of-the-art in PRA methods and data. Although many activities have been completed in the area of risk-informed regulation, the risk-informed analysis process for digital systems has not yet been satisfactorily developed. Since digital instrumentation and control (I&C) systems are expected to play an increasingly important role in nuclear power plant (NPP) safety, the NRC established a digital system research plan that defines a coherent set of research programs to support its regulatory needs. One of the research programs included in the NRC's digital system research plan addresses risk assessment methods and data for digital systems. Digital I&C systems have some unique characteristics, such as using software, and may have different failure causes and/or modes than analog I&C systems; hence, their incorporation into NPP PRAs entails special challenges.

The objective of the NRC's digital system risk research is to identify and develop methods, analytical tools, and regulatory guidance for (1) including models of digital systems into NPP PRAs, and (2) using information on the risks of digital systems to support the NRC's risk-informed licensing and oversight activities. For several years, Brookhaven National Laboratory (BNL) has worked on NRC projects to investigate methods and tools for the probabilistic modeling of digital systems, as documented mainly in NUREG/CR-6962 and NUREG/CR-6997. However, the scope of this research principally focused on hardware failures, with limited reviews of software failure experience and software reliability methods. NRC also sponsored research at the Ohio State University investigating the modeling of digital systems using dynamic PRA methods. These efforts, documented in NUREG/CR-6901, NUREG/CR-6942, and NUREG/CR-6985, included a functional representation of the system's software but did not explicitly address failure modes caused by software defects or by inadequate design requirements. An important identified research need is to establish a commonly accepted basis for incorporating the behavior of software into digital I&C system reliability models for use in PRAs. To address this need, BNL is exploring the inclusion of software failures[1] into the reliability models of digital I&C systems, such that their contribution to the risk of the associated NPP can be assessed.

Presently, there is no consensus method for modeling digital systems in an NPP PRA. In this study, a review of currently available quantitative software reliability methods (QSRMs) was performed with the objective of cataloging potential methods that can be used to quantify software failure rates and demand failure probabilities of digital systems at NPPs such that the system models can be integrated into a PRA. The QSRMs were identified by reviewing research on digital system modeling methods sponsored by the NRC or by the National Aeronautics and Space Administration, performed by international organizations, and published in journals and conferences. The QSRMs were categorized, described, and evaluated regarding their strengths and limitations for PRA applications. Note that specific

[1] Software failure can be defined as not successfully performing a specified/intended function or performing unintended actions. A software failure occurs when some inputs to the software occur and interact with the internal state of the digital system to trigger a fault that was introduced into the software during the software lifecycle.

recommendations regarding which QSRMs have the most potential for integration into a PRA, and are therefore candidates for further assessment, are beyond the scope of this study.

It is possible that reliability models of digital systems may include software failures representing different software failure modes at different levels of detail (e.g., the software may be modeled as separate software modules). However, a review of the literature revealed that practically all available QSRMs consider the software system as a whole, not as separate modules or broken down by failure mode. Depending on the reliability modeling method used for digital systems in a PRA, and the associated level of modeling detail, different QSRMs may be needed to quantify the digital system reliability model. It may also be necessary to separately model different types of software (e.g., application-specific software and operating system software), using different QSRMs. In addition, It is well recognized that software failures are sensitive to the context (environment) in which the software is operating. Therefore, it is important that the software context be accounted for when modeling software failures in an NPP PRA (e.g., the specific system function being evaluated and the associated success criteria, as well as other relevant conditions in the plant).

In general, for PRA modeling purposes, there are two types of digital systems at an NPP, namely, control and protection systems. A control system, such as a feedwater control system, performs its control function during normal plant operation. In contrast, a protection system, such as a reactor protection system (RPS), monitors the condition of the plant during normal operation, but only generates a reactor trip or other appropriate signal if a need arises. A control system may fail and cause a reactor trip, which would be included in a PRA as an initiating event (e.g., a loss of feedwater). An initiating event in a PRA, which is the starting point of an accident sequence analysis, is characterized by its annual frequency. Therefore, the frequency that a control system fails causing an initiating event needs to be estimated. A protection system may have two different failure modes. For example, an RPS may fail to generate a reactor trip signal when needed or it may generate a spurious trip signal. A spurious trip signal would be included in a PRA as an initiating event and can be modeled in the same way failure of a control system is modeled, that is, in terms of an annual frequency. On the other hand, given the occurrence of some other initiating event that leads to the need for a reactor trip, a failure to generate a reactor trip signal would be modeled in the PRA in terms of a demand failure probability. Therefore, even for a single system, different QSRMs may also need to be used depending on the failure modes of interest, that is, a failure-rate based method and a failure-on-demand based method.

As part of the study, a set of desirable characteristics for QSRMs for modeling the digital systems operating at an NPP was proposed. The desirable characteristics can be used in evaluating available QSRMs and their applications to determine if the characteristics are satisfied. In particular, it is desirable that a method be capable of demonstrating the high reliability of a safety-critical system (e.g., a failure on demand probability on the order of 10^{-5}, commensurate with an analog RPS). Although an itemized evaluation of the reviewed methods against the desirable characteristics is beyond the scope of this study, the information documented in this report is useful for performing such an evaluation.

Only a few publicly available studies that attempted to quantify software failure rates and demand failure probabilities were performed by organizations that are related to the nuclear industry, for example, the Bayesian Belief Network (BBN) studies performed at the Technical Research Center of Finland VTT and Halden Reactor Project. However, these particular studies did not address NPP digital systems. Even fewer publicly available studies were performed specifically to analyze digital systems at an NPP and these studies were explorative

in nature. For example, the BBN study performed at the Korean Atomic Energy Research Institute (KAERI) addressed the quality of the software requirement specification of an RPS. However, even in this latter case, the part of the study involving quantification of the software reliability is not publicly available. Therefore, the majority of methods reviewed in this report are those that have been used in other, non-nuclear industries.

The reviewed QSRMs were separated into the following four major categories:

(1) Software reliability growth methods – Time-based methods that use test data to estimate software failure rates that, in turn, are employed to ascertain whether a particular software can be released, by demonstrating that its failure rate meets the desired level.

(2) BBN methods – Methods that use a probabilistic graphical model depicting a set of random variables (represented by nodes) and their conditional independencies via a directed acyclic graph ("acyclic" means the graph does not form a feedback loop).

(3) Test-based methods - Methods that essentially employ standard statistical methods for the results of software tests ,and possibly for operational data, in the same way as hardware data is analyzed

(4) Other methods – Methods that include (a) a correlation approach that estimates software failure rate at the end of the software testing stage by making use of the software engineering practices of past software development projects, and implemented in a software tool called "Frestimate"; (b) metrics-based methods that estimate software failure rates and probabilities by correlating software engineering measures (SEMs)/metrics and software reliability; and (c) the context-based software risk model (CSRM) that combines traditional PRA approaches (e.g., event trees) with an advanced (dynamic) modeling approach to integrate the contributions of digital hardware and software into a model of overall system risk. In addition, a few studies that considered rule/standard-based methods and software diversities were reviewed.

The principal findings of the QSRM review are provided below.

1. Most of the existing QSRMs were not developed specifically for supporting quantification of software failure rates and demand failure probabilities to be used in reliability models of digital systems. However, they do estimate software failure rates or probabilities, and use the estimates in supporting decision making during software development. Some of these methods only estimate software failure rates (as opposed to demand failure probabilities), and would generally be applicable only to NPP control systems and the spurious actuation of protection systems, and not to the failure of protection systems to initiate their protective function(s), as discussed earlier. In the case of software reliability growth models (SRGMs), it is possible that they could be generalized or extended to model demand-type failures of protection systems.

2. There are many SRGMs in the literature, but none is generally superior to the others, because all are based on assumed empirical formulas that are not applicable to all situations. An SRGM that provides the best fit for one set of data may not provide the best fit for a different data set. Standard goodness-of-fit methods can be used to identify the SRGM which provides the best fit for a given set of data, and techniques for determining which method makes better predictions are available. It should also be noted that

performing the analysis with multiple SRGMs can be used as one means of addressing modeling uncertainty.

The many methods and terminologies in the literature suggest that unifying SRGMs may be desirable, especially those methods in the exponential Non-Homogeneous Poisson Process (NHPP) category. A generalization of SRGMs in terms of an NHPP is provided in this report.

3. BBN methods have the capability to aggregate disparate information about software (e.g., aggregation of software failure data and quality of software lifecycle activities that is assessed using expert elicitation) and to include parameter uncertainties as a part of the modeling. However, there are challenges in developing a BBN that takes full advantage of these capabilities, including the substantial development effort that is needed, the expertise of the BBN developers, the qualification of any experts used to elicit information relevant to estimating model parameters, and the availability of thorough documentation of the software development activities. Another challenge is that qualitative evidence (e.g., the impact of software development quality on software reliability) needs to be quantified. Since there may not be sufficient available data to "anchor" the conversion of the qualitative information to quantitative values, the uncertainty in the resultant quantitative estimates from the experts may be very large, which may make it difficult to demonstrate the small failure probabilities often associated with safety-related systems (a limitation common to many QSRMs).

4. Test-based methods are used in this study to demonstrate that a very large number of tests, $\sim 10^5$, must be conducted (and no failures observed) to demonstrate a mean software failure probability on demand of 10^{-5} (which is expected of an NPP safety-related system like an RPS). Besides the large number of tests required, there is also the concern that the testing environment may not represent the actual operating environment to which the software is exposed during operation in an NPP, which is a serious limitation on the accuracy of the testing results. Note, this issue applies to any QSRMs that use test data (e.g., SRGMs). Lastly, testing may not uncover errors in requirements and specifications of software, which have caused many software failures, though this limitation is common among many of the QSRMs reviewed.

5. The general concept of performing correlation/regression analyses using past software development experience is reasonable. However, because of the unavailability of detailed information on the past software development projects and the correlation/regression analyses used to construct the predictive model contained in Frestimate, this specific method could not be evaluated in detail. Based on a review of the information available, some potential limitations of the predictive model contained in Frestimate include (1) subjectivity in the responses to the survey of software-development practices; (2) large uncertainties associated with the process for determining the ratio between inherent defects and failure rate, because it does not involve the use of information related to the specific software being assessed; and (3) it is not known whether Frestimate was validated or benchmarked by organizations independent from the organization (SoftRel, LLC) that developed it. Also, similar to most QSRMs, Frestimate does not specifically account for the context in which software operates, does not consider specific software failure modes, and does not appear to be capable of estimating probabilistic parameters of software when the expected values of the parameters are small, as would be expected for protection systems in an NPP.

6. NUREG/CR-6848 documents the use of 6 metrics methods for predicting software reliability (referred to as reliability prediction systems [RePSs]), each based on one of the 40 "root"[2] software engineering measures (SEMs) identified and ranked by a set of experts, as documented in NUREG/GR-0019. The RePSs were developed by applying available methods, concepts, and empirical formulas, and do not represent new innovative methods. Some of the methods use engineering insights (as opposed to application-specific information) to develop empirical formulas representing the relationship between software reliability, that is, failure rate and probability, and software engineering measures. Since the empirical formulas are not laws of software reliability, their general applicability and accuracy are limited. Lastly, NUREG/CR-6848 considers that the results of the study validated the overall approach by showing that highly ranked SEMs produce results that are closer to the true answer. However, such a conclusion depends on the quantification methods developed and associated with the SEMs. Alternative quantification methods that may produce very different results can be developed and associated with the SEMs (as indicated in NUREG/GR-0019) and potentially lead to a different conclusion.

7. CSRM is not specifically an approach to estimating the probability or rate of failure modes of a particular software (i.e., it is not technically a QSRM), but is more of an overall integrated risk-modeling approach that incorporates hardware, software, and the static or dynamic interactions between them. It appears reasonable as a means of risk-informing the software testing process in support of assessing software reliability. Aspects of the CSRM approach can also be used to support quantifying software failure rates or demand failure probabilities for inclusion into an existing digital system reliability model. The most unique aspect of the CSRM approach is the context-based, risk-informed testing using a logically defined and partitioned input-parameter space for scenarios that involve off-nominal conditions (i.e., scenarios involving anomalous events, such as one or more component hardware failures). This context-based evaluation has to be carried out for each software-related failure scenario that involves a combination of software and hardware failures. The publicly available reports specifically on CSRM provide only one example of the implementation of the context-based, risk-informed testing approach. The CSRM reports suggest that such testing can be practically accomplished for the potentially large number of software failure modes that may need to be addressed given the level of resolution that CSRM applies in modeling software behavior, but it is not clear from the available information what amount of time and resources would be required. It should also be noted that the context-based, risk-informed testing is not meant to be applied to scenarios that occur under nominal system conditions (e.g., those that do not involve hardware failures), since these types of scenarios can be quantified using existing software reliability estimation models.

[2] As stated in NUREG/CR-6848, "the measure on which RePS construction is based is termed the 'root' of the RePS. Other measures within the RePS are defined as 'support' measures."

ACRONYMS AND ABBREVIATIONS

AIAA	American Institute of Aeronautics and Astronautics
ANSI	American National Standards Institute
BBN	Bayesian Belief Network
BN	Bayesian Network
BNL	Brookhaven National Laboratory
CCF	Common Cause Failure
cdf	Cumulative Distribution Function
CSRM	Context-Based Software Risk Model
DBN	Dynamic Bayesian Network
DD	Defect Density
ESFAS	Engineered Safety Features Actuation System
FSW	Flight Software
I&C	Instrumentation and Control
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
KAERI	Korean Atomic Energy Research Institute
KSLOC	Thousand Source Lines of Code
LLNL	Lawrence Livermore National Laboratory
LOC	Loss of the Crew
M-ADS	A helicopter location identification system
MTTF	Mean Time to Failure
NASA	National Aeronautics and Space Administration
NHPP	Non-Homogeneous Poisson Process
NPP	Nuclear Power Plant
NRC	Nuclear Regulatory Commission
OOBN	Object-Oriented Bayesian Network
PACS	Personnel Access Control System
pdf	Probability Density Function
pfd	Probability of Failure on Demand
PRA	Probabilistic Risk Assessment
PSA	Probabilistic Safety Assessment
QSRM	Quantitative Software Reliability Method

ACRONYMS AND ABBREVIATIONS (CONT'D)

REM 610	A motor-protection relay system
RPS	Reactor Protection System
RSDIMU	Redundant Strapped Down Inertia Measurement Unit
SDLC	Software Development Life Cycle
SEM	Software Engineering Measures
SIL	Safety Integrity Level
SPAM-150-C	A motor-protection relay system
SRGM	Software Reliability Growth Model
SSE	Sum of Square Errors
STS	Space Transportation System
SW	Software
UMD	University of Maryland
V&V	Verification and Validation

1. INTRODUCTION

1.1 Background

Nuclear power plants (NPPs) traditionally relied upon analog instrumentation and control (I&C) systems for monitoring, control, and protection functions. With a shift in technology from analog systems to digital systems with their functional advantages (e.g., fault-tolerance, self-testing, signal validation, and process system diagnostics), plants have begun such replacement, while new plant designs fully incorporate digital I&C systems. However, digital systems have some unique characteristics, such as using software, and may have different failure causes and/or modes than the analog systems; hence, their incorporation into NPP probabilistic risk assessments (PRAs) entails special challenges.

The current U.S. Nuclear Regulatory Commission (NRC) licensing process for digital systems rests on deterministic engineering criteria. In its 1995 PRA policy statement [USNRC 1995], the Commission encouraged the use of PRA technology in all regulatory matters to the extent supported by the state-of-the-art in PRA methods and data. Although many activities have been completed in the area of risk-informed regulation, the risk-informed analysis process for digital systems has not yet been satisfactorily developed. Since digital I&C systems are expected to play an increasingly important role in NPP safety, the NRC established a digital system research plan [USNRC 2001] that defines a coherent set of research programs to support its regulatory needs. One of the research programs included in the NRC's digital system research plan addresses risk assessment methods and data for digital systems.

The objective of the NRC's digital system risk research is to identify and develop methods, analytical tools, and regulatory guidance for (1) including models of digital systems into NPP PRAs, and (2) using information on the risks of digital systems to support the NRC's risk-informed licensing and oversight activities. For several years, BNL has worked on NRC projects to investigate methods and tools for the probabilistic modeling of digital systems, as documented mainly in NUREG/CR-6962 [Chu 2008] and NUREG/CR-6997 [Chu 2009a]. However, the scope of this research principally focused on hardware failures, with limited reviews of software failure experience and software reliability methods. NRC also sponsored research at the Ohio State University investigating the modeling of digital systems using dynamic PRA methods. These efforts, documented in NUREG/CR-6901 [Aldemir 2006], NUREG/CR-6942 [Adlemir 2007], and NUREG/CR-6985 [Aldemir 2009], included a functional representation of the system's software but did not explicitly address failure modes caused by software defects or by inadequate design requirements. An important identified research need is to establish a commonly accepted basis for incorporating the behavior of software into digital I&C system reliability models for use in PRAs. To address this need, BNL is exploring the inclusion of software failures into the reliability models of digital I&C systems, such that their contribution to the risk of the associated NPP can be assessed. Two tasks were undertaken towards this objective.

(1) Establishment of a philosophical basis for incorporating software failures into digital system reliability models for use in PRAs

    On May 5 and 6, 2009, BNL hosted an expert panel meeting (workshop) establishing a "philosophical basis" for incorporating software failures into digital system reliability models. The panelists, who were primarily experts that have performed substantial work and/or

research on the fundamental principles and application of software reliability engineering, agreed that there is a rational basis for modeling and quantifying software failures within the context of a PRA, and quantitative methods can be used to quantify software failure rates and probabilities [Chu 2009b]. Software failure can be defined as "not successfully performing a specified (intended) function or performing unintended actions." Faults (which may include, but are not limited to, coding errors and requirements and specification errors) are introduced into a piece of software during the software lifecycle. A software failure occurs when some inputs to the software interact with the digital system's internal state and trigger a fault. As pointed out in [Chu 2009b], some researchers argue that software failure is a deterministic process and probabilistic descriptions of software behavior are inappropriate. However, because of our incomplete knowledge, we are not able to fully account for and quantify all the variables that define the software failure process. Therefore, we can choose to use probabilistic modeling to describe and characterize it.

(2) Review of quantitative software reliability methods (QSRMs)

A review was undertaken of current methods for quantifying software failure rates and probabilities to catalog those that might be used to support reliability modeling of digital systems of NPPs.

The objective of this report is to document the work accomplished under the second task. This report was peer reviewed by 15 internal and external reviewers.

1.2 Objective and Scope

The objective of reviewing the QSRMs was to gain comprehensive knowledge of available methods, especially those emphasizing the quantification of software failure rates and probabilities that might be employed in reliability models of digital systems used in NPP PRAs. The review was built upon BNL's previous reviews of software reliability methods, and on leveraging earlier work sponsored by the NRC and by the National Aeronautics and Space Administration (NASA).

In general, for PRA modeling purposes, there are two types of digital systems at an NPP, that is, control and protection systems. A control system, such as a feedwater control system, performs its control function during normal plant operation. In contrast, a protection system, such as a reactor protection system (RPS), monitors the condition of the plant during normal operation, but only generates a reactor trip or other appropriate signal if a need arises. A control system may fail and cause a reactor trip, which would be included in a PRA as an initiating event (e.g., a loss of feedwater). An initiating event in a PRA, which is the starting point of an accident sequence analysis, is characterized by its annual frequency. Therefore, the frequency that a control system fails causing an initiating event needs to be estimated. A protection system may have two different failure modes. For example, an RPS may fail to generate a reactor trip signal when needed or may generate a spurious trip signal. A spurious trip signal would be included in a PRA as an initiating event and can be modeled in the same way failure of a control system is modeled, that is, in terms of an annual frequency. On the other hand, given the occurrence of some other initiating event that leads to the need for a reactor trip, a failure to generate a reactor trip signal would be modeled in the PRA in terms of a demand failure probability. Therefore, even for a single system, different QSRMs may need to

be used depending on the failure modes of interest, that is, a failure-rate based method and a failure-on-demand based method.

It is well recognized that software failures are sensitive to the context (environment) in which the software is operating. Therefore, it is important that the software context be accounted for when modeling software failures in an NPP PRA (e.g., the specific system function being evaluated and the associated success criteria, as well as other relevant conditions in the plant). As an example, in a typical PRA, the RPS is usually the first top event in the event trees. Therefore, each initiating event defines a context for this system, that is, different initiating events represent different plant conditions that may generate different input signals to the RPS, and the system software may need to be modeled differently for different initiating events. For other protection (actuation) systems, such as a system which generates an actuation signal of an injection system, different sequences in different event trees define the different contexts for the actuation system and its software. More refined contexts can be determined by the cutsets of those sequences leading to the demand of the actuation system. Each of the cutsets represents a more specific scenario in which the system should function, and typically contains hardware failures and human errors that help determine the possible variations in the input signals to the system. It should be noted that the software context can be more refined than just what is specified by the cutsets, and, therefore, the resolution of the PRA should be considered when choosing and implementing a modeling approach for software failures.

Currently, there is no consensus method for modeling digital systems in an NPP PRA. It is possible that reliability models of digital systems may include software failures representing different software failure modes at different levels of detail (e.g., the software may be modeled as separate software modules). However, a review of the literature revealed that practically all available QSRMs consider the software system as a whole, not as separate modules or broken down by failure mode.[3] Depending on the reliability modeling method used for digital systems in a PRA, and the associated level of modeling detail, different QSRMs may be needed to quantify the digital system reliability model. In addition, it may be necessary to separately model different types of software (e.g., application-specific software and operating system software), using different QSRMs.

Since the software failure rates and/or probabilities are intended to be used in an NPP PRA, the review of QSRMs mainly centered on methods that can be used to quantify the reliability of software once it has been put into service at an NPP. Therefore, it was not BNL's intent to explore how to improve the reliability of software, for example, by formulating methods for identifying software faults and improving software verification and validation (V&V).

Note that specific recommendations regarding which QSRMs have the most potential for integration into a PRA, and are therefore candidates for further assessment, are beyond the scope of this study.

---

[3] In principle, the QSRMs can be applied to separate modules or failure modes (e.g., [Smidts 1999]). In practice, the methods do not appear to have been applied that way. Potential limitations of applying QSRMs to individual modules or failure modes include (1) difficulty in defining module boundary and (2) incomplete knowledge of failure modes.

1.3 Approach

The approach followed in this study is summarized below:

Development of Desirable Characteristics of QSRMs

A set of desirable characteristics of a QSRM was developed and is documented in Section 2. The development of these characteristics greatly benefitted from the opinions of the panelists at the workshop of Task (1), as detailed in Chu [2009b]. While the characteristics are intended to be used to evaluate and select QSRMs for future applications, a structured comparison of the methods against the individual characteristics was not performed as part of this effort. The selection of potential QSRMs for PRA applications is beyond the scope of the current study, but expectedly will rely on these desirable characteristics.

Identification of QSRMs

Multiple sources were searched to identify QSRMs.

(1) NRC-sponsored research

BNL previously reviewed the literature on modeling digital systems and identified the methods for, and the issues associated with, modeling them; Chu [2007] details some of the NRC sponsored research performed at BNL. In addition, NRC sponsored research at the University of Virginia [Kaufman 2001], University of Maryland [Smidts 2000], and Ohio State University [Aldemir 2006] also identified modeling methods for digital systems.

(2) Research performed at International Organizations

Members of the NEA/CSNI/WGRisk (Risk Working Group of the Committee on the Safety of Nuclear Installations of the Nuclear Energy Agency) met in October 2008 to share their experience in digital-system modeling; QSRMs were among the topics discussed [NEA 2009]. In addition, an NEA/CSNI/WGRisk report [Dahll 2007] summarized software reliability methods, and a few papers and reports issued by the Halden Reactor Project, such as Gran [2002a], discussed QSRMs.

(3) NASA-sponsored research

A NASA - NRC technical interchange meeting on software/digital instrumentation and control system reliability analysis took place in March 2009 during which two methods being considered by NASA were presented: a context-based software risk model (CSRM) [ASCA 2007] and a correlation method based on software-development practices [Neufelder 2002].

(4) Open literature research

There is a wealth of papers and books on software reliability. Some discuss the problems in modeling software failure and include summary descriptions of various methods, which were particularly helpful (e.g., [Dahll 2007, Lyu 2007, National Research

Council 1997, Littlewood 2005]. To identify any newly developed methods, several recent conference proceedings were reviewed, for example, conferences of the PSAM (Probabilistic Safety Assessment and Management), PSA (Probabilistic Safety Assessment), and ISSRE (International Symposium on Software Reliability Engineering).

Description and review of QSRMs

Various QSRMs were reviewed for their potential use in quantifying software failure rates and probabilities to support the reliability modeling of digital systems in an NPP PRA. These QSRMs fall into four major categories as discussed in [Dahll 2007], namely, software reliability growth methods, Bayesian belief network (BBN) methods, test-based methods, and other methods including the correlation method based on software engineering practices [Neufelder 2002], metrics-based methods [Smidts 2004], the CSRM method [ASCA 2007], rule/standard based methods [Bloomfield 2007], and software diversity quantification methods [Lyu 2003]. It should be noted that in some taxonomies software reliability growth methods could be thought of as a subset of test-based methods, and as indicated in Section 5, some of the issues associated with test-based methods are applicable to software reliability growth methods.

1.4 Organization of the Report

In Section 2, a set of desirable characteristics is proposed for QSRMs for quantifying software failure rates and probabilities to support the reliability modeling of digital systems in an NPP PRA. Sections 3 to 6 describe the QSRMs identified above and provide comments on their strengths and limitations. Section 7 contains a summary and the principal findings of the review.

2. DESIRABLE CHARACTERISTICS OF QUANTITATIVE SOFTWARE RELIABILITY METHODS

A set of desirable characteristics for quantitative software reliability methods (QSRMs) for quantifying software failure rates and probabilities to support the reliability modeling of digital systems in a nuclear power plant (NPP) probabilistic risk assessment (PRA) is proposed below. The desirable characteristics were developed based on the perceived need for reliability models of digital systems in a PRA as discussed in Section 1 and the knowledge and experience of the study team in performing research and a literature review on modeling of digital systems. These characteristics are expected to address the general guidelines provided in the American Society of Mechanical Engineers (ASME) standard for PRA for NPP applications. The internal and external peer review of this report served as an independent review of the proposed set of desirable characteristics. It is emphasized that these characteristics are only "desirable," and are not necessarily pre-requisites for using a QSRM to support an NPP PRA. The desirable characteristics can be used in evaluating available QSRMs and their applications to determine if the characteristics are satisfied. Although an itemized evaluation of the methods against the desirable characteristics is beyond the scope of this study and is planned to be included in the next phase of the research, the information in Sections 3-7 that reviewed different QSRMs is useful in performing such an evaluation.

1. The description of the method and its application is comprehensive and understandable.

   There should be adequate documentation of the method and its applications such that the method and its applications can be understood and evaluated. It should include a description of the intended uses (i.e., the applicability) of the method (e.g., is it only intended as a means for deciding if a software can be released) and all important assumptions, including their bases. The information in the documentation should allow a reader to use the method in applications. For applications of the method, the results and their use in accomplishing the objective of the application should be documented.

2. The assumptions of the method have reasonable bases.

   The assumptions of a method can significantly limit the usefulness and applicability of the method. The bases of the assumptions should be provided, their significance should be discussed, and the limitations on the applicability of the method due to the assumptions should be discussed. For example, some QSRMs assume that a discovered fault is fixed perfectly; however, the "fix" may not completely resolve the problem and may even introduce new faults. Nonetheless, this assumption is used in deriving empirical formulas that in some cases have been shown to produce good results. Another example is the assumption that automated software tests are independently drawn from the software's actual operational profile, which may be difficult to verify and may make test-based methods inadequate for demonstrating a low failure probability. For example, it may not be possible to demonstrate a failure probability of 10^{-5} (typical for an analog reactor protection system [RPS]), even if millions of test cases can be performed, due to the epistemic (i.e., state-of-knowledge) uncertainty regarding the representativeness of the test profile [Chu 2009b].

3. The method allows for consideration of the specific operating conditions of the software.

   It is well recognized that software failures are sensitive to the context (environment) in which the software is operating [Garrett 1999]. This context can include previous failures or other influential events or conditions. For example, a specific failure mode of a digital feedwater control system may apply during full-power operation, but not during low-power operation. An application of a QSRM should take into consideration the specific operating conditions (context) of the software. The QSRM and the reliability model in which it is used should also be compatible with the NPP PRA framework.

4. The method takes into consideration the quality of lifecycle activities.

   High quality software results from the use of good software engineering practices during development to minimize the probability of introducing errors into the software, and a rigorous verification process to maximize the probability of detecting errors [National Research Council 1997]. The quality of life cycle activities of software is expected to significantly impact the reliability of the software. For example, a certified, well-experienced software development team is expected to produce better software than that of a less qualified team. Due to recognized weaknesses in the state of the art of QSRMs, that is, lack of adequate data to demonstrate high software reliability using test-based methods, the ability of a QSRM to account for the quality of life cycle activities is desirable, particularly through the systematic elicitation of expert judgment.

5. The method makes use of available test results and operational experience.

   Operational experience is the most direct evidence of software system reliability. Due to sensitivity of software to its operating environment (context), the operational experience should be collected from actual operation of the software being analyzed. When test results are used in the same way operational experience is used, it should be demonstrated that the test inputs are sampled independently from the operational profile or a profile which is a good representation of the operational profile. It is well recognized that different types of tests have different capabilities in detecting faults [Frank 1997]. For example, debug testing can be superior for identifying bugs because its detection rate may be greater than that of operational testing, but this does not mean that it is necessarily better for estimating software failure rate or probability. If sufficient test results and operational experience are not available for the software being analyzed, there may be some benefit to using data for similar software from other applications.

6. The method addresses uncertainty.

   The method should estimate uncertainty of software failure rates and probabilities, and provide discussion on the significance of important assumptions. For example, in the case of using operating experience and test data in a QSRM, applicability of the data has to be verified, that is, the test profile from which the data are collected should be representative of the actual operational profile of the software being analyzed.

7. The method has been verified and validated.

   The method should be verified and validated. Successful application of a method in supporting decision-making and achieving the objectives of the applications (e.g., using system failure rate or probability in a reliability model or demonstrating that the reliability is satisfactory for a particular purpose) can serve as a means of verification and validation. Also, consistency with operational experience should be demonstrated.

8. The method is capable of demonstrating the high reliability of a safety-critical system (e.g., a failure on demand probability on the order of 10^{-5}, commensurate with an analog RPS).

   High reliability is considered typical of the analog RPS of current NPPs. It is recognized that current methods are not capable of demonstrating such a high reliability with confidence for a digital protection system [Chu 2009b]. Therefore, improvements to the current methods are needed.

9. The method should be able to estimate parameters that can be used to account for software common cause failures (CCFs),

   It is probably a reasonable assumption that multiple redundant channels of a digital system that use identical software would fail together. For channels or systems that are not identical, dependent failure may also take place. For example, N-version programming is a diversity strategy, but does not guarantee that software developed by different development teams will not fail dependently. Effectiveness of diversity strategies in preventing software CCFs should be accounted for when modeling software CCFs.

In Sections 3-6, summary descriptions of QSRMs are provided along with discussions of their strengths and limitations. In addition, Section 7 provides summary comments on the QSRMs. The information in these sections is useful in evaluating the methods against the desirable characteristics.

3. SOFTWARE RELIABILITY GROWTH METHODS

3.1 Introduction

Software reliability growth methods,[4] that is, as described in Institute of Electrical and Electronics Engineers (IEEE) Standard 1633 [2008], Lyu [1996], and American Institute of Aeronautics and Astronautics (AIAA) [1992], use test data to estimate software failure rates that, in turn, are employed to ascertain whether a particular software can be released, by demonstrating that its failure rate meets the desired level. Software reliability growth methods are time-based and require as input the times between successive failures during tests, or number of failures during different intervals. Typically, they assume that the software is fixed perfectly and instantaneously after failures, so that its reliability "grows" with time, that is, the software failure rate declines with time. Software reliability growth methods are known so widely that models using these methods are often termed software reliability models, omitting the word "growth." Figure 3-1, from AIAA [1992], illustrates the use of software failure rate to determine by how much testing must be prolonged before reaching a failure-rate objective. After the findings from a reliability-growth model suggest a particular software is ready for release, the failure rate assessed using this model can be considered as the failure rate of the software during operation, which is the interest of this study.

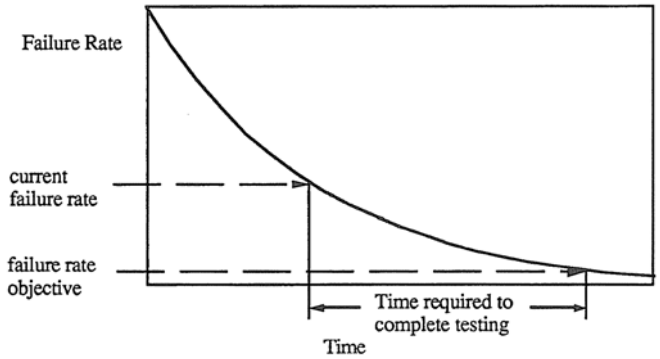


Figure 3-1 Using reliability growth methods to determine required test duration before achieving failure rate objective

A large number of software reliability growth models (SRGMs) have been developed over the years, and each model has its strengths and limitations. There are different assumptions in different SRGMs on how the failure rate decreases with time; that is, the models specify different empirical formulas, and use test data to estimate their parameters. The empirical formulas and test data are employed to decide if the failure rate objective has been reached. Musa proposed a categorization scheme [1984 and 1987] that has been widely referenced, for example in Lyu [1996]. In general, the large number of SRGMs that were formulated separately should be unified because many of them are similar, although their terminologies are inconsistent. More recently, IEEE Standard 1633 [2008] grouped SRGMs into three high-level

[4] Many of the references for Section 3 of this report refer to reliability growth *models*, not methods. While the authors of this report believe that it is more appropriate to characterize them as methods, they are often described in this report as models to maintain consistent terminology with the referenced documents.

categories: exponential Non-Homogeneous Poisson Process (NHPP), non-exponential NHPP, and Bayesian, and briefly described many of the models.

An NHPP is a Poisson process wherein failure rate changes with time; it is needed because SRGMs basically assume the failure rate of software declines with time, unlike the supposedly constant hardware failure rate over time considered in a PRA. The "exponential" and "non-exponential" NHPPs represent processes whose failure rate decreases in time exponentially or non-exponentially, respectively. A homogeneous Poisson process can be considered a special case of an NHPP in which the failure rate is constant in time; it typically is used for failure-rate-based events in a PRA.

The SRGMs discussed in this report are failure-rate based. To quantify the failure of a protection system to initiate its protective function(s), it would be more appropriate to use a failure-on-demand-based method, as discussed previously. It may be possible to generate demand-based results by including the frequency of demands in the failure rate estimation of an SRGM, or re-interpret the time-based failure data used in an SRGM as demand-based data. In, particular, two groups of methods that are related to the SRGMs, and can potentially be used to model software failure on demand, are discrete reliability growth methods and discrete software reliability growth methods. Evaluation of these methods is beyond the scope of this study, but they are briefly summarized below:

(1) In a recent study, Hall [2008] developed a mathematical formulation of a discrete reliability growth method, which models system reliability improvement due to changes in system design after failure modes are identified during tests. The method is a failure-on-demand-based method, and its applicability to modeling software failure remains to be evaluated.

(2) Yamada [1985] proposed a discrete software reliability growth method using a discrete NHPP, which is based on the number of failures in a sequence of tests without considering the times when the tests are performed, and is intended to overcome the issue of the SRGMs associated with selecting calendar time or execution time for estimating software failure rates (i.e., in some cases, it may not be possible to collect data based on execution time, and calendar time has to be used to estimate the execution time). Okamura [2004] presented additional discrete software reliability growth models such as the cumulative binomial process developed by Hoare and Rahman [1983], and developed an approach for estimating model parameters. It may be possible to develop a failure-on-demand-based method building upon the discrete NHPP methods.

Section 3.2 defines and details the properties of an NHPP, and categorizes and discusses some SRGMs according to the IEEE standard's categorization. Using a NHPP in characterizing SRGMs provides the mathematical foundation of the latter, and offers a means of unifying the many SRGMs in the literature. Section 3.3 compares some of the SRGMs, while Section 3.4 provides comments on them.

3.2 Description and Categorization of SRGMs

In this section, the SRGM categorization of the IEEE standard is adopted, and some of the methods in each of the categories are briefly discussed with references to the original publications provided. Section 3.2.1 defines some of the terms used in SRGMs, including some basic properties of an NHPP. Sections 3.2.2 to 3.2.4 offer short descriptions of different types

of SRGMs. Due to the widely varying notations and assumptions of the SRGMs, the details of individual models are not provided. As described in Section 3.2.1, a way of defining an SRGM is in terms of the functional form of how the failure rate changes with time. Section 3.2.5 briefly describes standard methods of estimating the parameters of the SRGMs.

3.2.1 Definition of Terms

The lifetime (time to failure) of a system is represented by a probability distribution, *F(t)*, and a probability density function, *f(t)*, with failure rate defined as

$$\lambda(t) = f(t)/\{1-F(t)\}.$$

Reliability of the system is related to the failure rate by

$$R(t) = 1-F(t) = \exp[-\int_0^t \lambda(x)dx] \quad (3\text{-}1)$$

In SRGMs, the occurrence of software failures can be modeled by an NHPP. The following is a brief summary of an NHPP model extracted from Beichelt and Fatti [2002] and Cinlar [1975].

Let $N(t)$ be the number of failures up to time t, $\mu(t)$ the expected number of failures at time t (also called the mean value function), that is, $\mu(t) = E\{N(t)\}$, and the intensity function[5] $\lambda(t) = \frac{d\mu(t)}{dt}$. The assumptions of NHPP include

(1) $N(0) = 0$,
(2) $\{N(t), t \geq 0\}$ has independent increments,
(3) $P[(N(t+h) - N(t)) = 1] = \lambda(t)h + O(h)$, and
(4) $P[(N(t+h) - N(t)) > 1] = O(h)$

for all t, and where h is a small value, and $O(h)$ represents the second and higher order terms of h such that $O(h)/h \to 0$ as $h \to 0$.

Some of the properties of an NHPP include:

(1) Let $t_0(=0), t_1, t_2, t_3, \cdots, t_n(=t)$ be a partition of the interval 0 to t, and f_i the number of failures between t_{i-1} and t_i, then the f_i's are independent Poisson random variables with $E\{f_i\} = \mu(t_i) - \mu(t_{i-1})$, and

$$P[f_i = n] = \frac{[\mu(t_i) - \mu(t_{i-1})]^n}{n!} e^{-[\mu(t_i)-\mu(t_{i-1})]}. \quad (3\text{-}2)$$

[5] In this report, the terms "failure rate" and "intensity function" are used interchangeably.

(2) The successive times to failure, $T_1, T_2, T_3, \cdots$, are random variables with distribution $F_i(t)$ representing the time to the i[th] failure, For time to the first failure, $F_1(t) = 1 - \exp[-\int_0^t \lambda(x)dx]$, and

$f_1(t) = \lambda(t)\exp[-\int_0^t \lambda(x)dx]$.

[Comparing $F_1(t)$ with Equation (3-1), note that the intensity function $\lambda(t)$ of the NHPP is the same as the failure rate of T_1.]

Similarly,

$$F_i(t \mid T_1, T_2, --T_{i-1}) = \Pr ob(T_i \le t \mid T_1, T_2, --T_{i-1}) = 1 - \exp[-\int_{T_{i-1}}^t \lambda(x)dx], \text{ and}$$

$$f_i(t \mid T_1, T_2, --T_{i-1}) = \lambda(t)\exp[-\int_{T_{i-1}}^t \lambda(x)dx]. \quad (3\text{-}3)$$

As discussed later, Equations (3-2) and (3-3) often are used in SRGMs in estimating the models' parameters.

Historically, many different SRGMs were developed somewhat independently, making different assumptions and defining different terminologies and parameters. Nevertheless, some of the models are very similar, if not identical [AIAA 1992]. Attempts were made to unify SRGMs, for example, Pham [2000] represented a few SRGMs in terms of different functional forms of $\mu(t)$, Grottke [2001] represented a few SRGMs in terms of differential equations for $\mu(t)$ and Huang [2003] derived a few SRGMs in terms of different ways in which $\mu(t)$ is calculated as the average of earlier values. This thought is also reflected in the categorization scheme of IEEE Standard 1633 [2008].

Different SRGMs have dissimilar assumptions about how various parameters change with time, such as expected number of failures $\mu(t)$. Categorizing the methods in terms of how $\mu(t)$ changes with time is seemingly reasonable, and all the historical SRGMs can be grouped and standardized within this framework.

3.2.2 Exponential NHPP Models

The SRGMs in this category assume that the failure rate decreases exponentially with time, which is the consequence of the widely used assumption that the failure rate is proportional to the number of current faults in the software. The decline with time is similar to the decay of radioactive isotopes, that is, the rate of radioactive decay of an isotope is proportional to the inventory of the isotope, which decreases exponentially with time. The methods include Musa's Basic model [Musa 1975], Schneidewind's model [Schneidewind 1975], Goel's NHPP model [Goel 1979], the Generalized Exponential model [IEEE 2008, AIAA 1992], Shooman's Exponential model [Shooman 1972], and Jelinski-Moranda's model [Jelinski 1972 and Lyu 1996]. These methods follow different formulations and define some parameters differently. It is demonstrated below that they all end up with a failure rate that decreases exponentially. A frequent assumption in estimating the parameters for these models, and others, is that the failure rate remains constant over the intervals between failures. Effectively, the integration of Equation (3-3) is replaced with $\lambda_i \times (t_i - t_{i-1})$, where λ_i is the failure rate of the i^{th} failure.

For example, in Sections 10.2.2 and 10.2.3, Musa [1987] essentially derived Equations (3-2) and (3-3) using somewhat different notations. Musa's Basic model [1975, 2004] assumes that the failure rate is proportional to the current fault content and showed that:

$\mu(t) = \nu_0[1 - \exp(-\frac{\lambda_0}{\nu_0}t)]$, and

$$\lambda(t) = \lambda_0 \exp(-\frac{\lambda_0}{\nu_0}t), \quad (3\text{-}4)$$

where the two parameters to be estimated using failure data are ν_0, the initial total number of faults, and λ_0, the initial failure rate. The Basic Model is related to a few other models, namely, Shooman's Exponential model [Shooman 1972], Jelinski-Moranda's model [Jelinksi 1972], and the Generalized Exponential model [IEEE 2008, AIAA 1992], via simple transformations of the models' parameters [IEEE 2008, AIAA 1992].

Shooman's Exponential model [1972] considered software failure rate on a per software instruction and per debugging man-month basis. It assumes that the number of initial errors per instruction is a constant, and the failure rate is proportional to the number of errors present.

Jelinski [1972] assumed (1) the failure data take the form of successive, independent times between failures, (2) a constant failure rate between failures, and (3) a failure rate that is proportional to the software's current fault content; and derived an equation equivalent to Equation (3-3).

The American Institute of Aeronautics and Astronautics/American National Standards Institute (AIAA/ANSI) standard [2002] and IEEE Standard 1633 [2008] highlighted simple transformations between the parameters of the above models, and suggested a Generalized Exponential model that could represent them.[6]

Goel and Okumoto [1979] started with several assumptions, such as (1) the numbers of failures in non-overlapping intervals are independent and (2) the expected number of failures is proportional to the expected number of undetected errors, and demonstrated the underlying process is that of an NHPP. Their model can be considered a good representative of all the exponential NHPP models. It since has been mistakenly called the NHPP model, a misnomer because all other SRGMs are NHPPs, including those that are not exponential.

The Schneidewind model [Schneidewind 1975], which is based on discrete intervals, has a failure rate defined by:

$d(i) = \alpha \exp(-\beta i)$,

[6] In the standards, Musa's Logarithmic model [Musa 1984] is in the table of the transformations. However, this model does not have an exponentially decreasing failure rate.

where $d(i)$ is the failure rate during interval i, and α and β are, respectively, the rate at time zero and the decay constant.

Xie [1992] showed that the Goel and Okumoto model is a continuous time version of the Schneidewind model. Xie also showed that the Schneidewind model basically is the same as a few other models. Note that the exponential decaying function of Equation (3-4) is the same as the equation for failure rate in Goel and Okumoto's model [Goel 1979], that is,

$$\lambda(t) = ab\exp(-bt), \quad \text{(3-5)}$$

if $a = \nu_0$ and $b = \frac{\lambda_0}{\nu_0}$.

Therefore, the models in the exponential NHPP category are all essentially the same. In Musa's categorization scheme [Musa 1984], some SRGMs were classified into Poisson and binomial "types." He later demonstrated [Musa 1987] that a generalization of the binomial-type model leads to the Poisson model.

Most of the above models presume that when a failure occurs, the fault immediately is fixed perfectly, so that failure rate decreases with time. In attempting to address the potential for ineffectively removing faults, Musa's Basic model introduces an additional fault-reduction factor representing the efficacy of that process. The Schneidewind model [1975] has three variations and allows discarding older data and modeling of a delay in correcting faults detected by tests. Xie [1992] offers a generalization of the Schneidewind model. The models of delayed fault correction incorporate an additional model parameter, making the failure rate non-exponential. In the overall scheme, it probably is not important whether the failure rate decreases exponentially. Ultimately, the evaluation of how good a model is rests upon whether it fits the data well.

3.2.3 Non-Exponential NHPP Models

The reliability growth models of this category assume that the way failure rate changes/decreases with time does not follow an exponential function. For example, it may follow the shape of the probability density function of a gamma or Weibull distribution [Duane 1964]. The distinction between exponential and non-exponential is somewhat artificial. Different reliability growth models use different functional forms for how *λ(t)* and *μ(t)* change with time. Some of the models in this category are briefly described below.

In Musa's Logarithmic Poisson Execution Time Method [Musa 1984], instead of the exponential NHPP assumption that failure rate decreases exponentially with time (proportional to the software's current fault content), it is assumed that the failure rate falls exponentially with the expected number of failures, that is:

$$\lambda(t) = \lambda_0 \exp[-\theta\mu(t)], \quad \text{(3-6)}$$

where θ is a proportionality constant, and $\mu(t)$ the expected number of failures. It can be shown [Musa 2004] that:

$$\lambda(t) = \frac{\lambda_0}{\lambda_0 \theta t + 1}.$$

Duane's model [Duane 1964 and AIAA 1996] assumes that $\mu(t) = \alpha t^{\beta}$, or equivalently, $\lambda(t) = \alpha\beta t^{\beta-1}$, where α and β are larger than 0, and are the model parameters to be estimated. Note that λ is Weibull-distributed, and is decreasing with t if β is less than 1.

The basis for the S-shaped reliability growth models [Yamada 1983 and 1984] is the observation that, in many software-development practices, the change in the cumulative number of removed faults with the testing time is represented by an S-shaped curve. The S-shape of the mean value function (of the cumulative number of removed faults) reflects the following: (1) the fault detection/removal rate is relatively flat in the beginning of software testing, (2) it increases exponentially after the testers become familiar with the software, and (3) it flattens thereafter because the remaining faults are more difficult to uncover [IEEE 2008]. The S-shaped reliability growth models are categorized as either "delayed" [Yamada 1983] or "inflection" [Yamada 1984] according to the causes of the S-shapedness, that is, either the delay between failure observation and fault removal, or the mutual dependency of some faults in the software [Kapur 1995]. Yamada's delayed S-shaped model [1983] assumes that testing software involves two processes, fault observation and fault removal.

In Yamada's S-shaped model [Yamada 1983], the expected number of failures is assumed to be S-shaped, that is,

$$\mu(t) = a[1-(1+bt)\exp(-bt)],$$

where *a* is the total number of faults, and *b* is the error-detection rate per error in the steady state.

Usually, an SRGM assumes that detection and removal of faults are independent. The inflection S-shaped model [Yamada 1984] does not model the fault dependency directly; rather, it models the manifestation of this dependency assuming the rate of removing faults is a function of the cumulative number of faults removed until time t [Kapur 1995], that is, some faults are not detectable before others [Chillarege 1991]. The mean value function of the inflection model is

$$\mu(t) = a\,\frac{1-\exp(-bt)}{1+\frac{1-r}{r}\exp(-bt)}$$

where r $(0 < r \le 1)$ is the inflection parameter, indicating the ratio of the number of detectable faults to the total number of faults in the software. An inflection parameter of $r = 1$ implies that the faults are independent, and so the inflection growth model is reduced to the Goel and Okumoto model [Kapur 1995], and the growth curve becomes exponential.

There are many more published SRGMs, and it is likely that all can be expressed in terms of function forms of $\mu(t)$ and $\lambda(t)$. For example, Williams [2006] compared a few SRGMs,

including some methods briefly discussed above, and additional ones with three parameters instead of two. The third parameter accounts for imperfect debugging, detectability of faults, and the like.

3.2.4 Bayesian Models

The two NHPP categories of reliability growth models assume that failure rate decreases with time (either exponentially or non-exponentially), while the Bayesian model developed by Littlewood and Verrall [Littlewood 1974, IEEE 2008] assumes that the failure rate decreases probabilistically/stochastically with time and uses Bayes' theorem in its derivations. The Bayesian model essentially is an exponential NHPP model that explicitly includes the uncertainty of the failure rate in the model. This model considers successive times to failure, $T_1, T_2, \cdots, T_n$, to be exponentially distributed with failure rates $\lambda_1, \lambda_2, \cdots, \lambda_n$, with the failure rates being gamma distributed, that is:

$$g(\lambda_i) = \frac{[\psi(i)]^{\alpha} \lambda_i^{\alpha-1} e^{-\psi(i)\lambda_i}}{\Gamma(\alpha)},$$

where α and $\psi(i)$ are the shape and scale parameters of the gamma distribution, and the expected (mean) value $E\{\lambda_i\} = \frac{\alpha}{\psi(i)}$. Therefore, the mean of λ_i decreases with i if $\psi(i)$ increases with i, that is, the failure rate decreases probabilistically. This also reflects the likelihood, but not certainty, that correcting a fault is effective [IEEE 2008]. By assuming $\psi(i)$ equal to either $\beta_0 + \beta_1 i$ or $\beta_0 + \beta_1 i^2$ and eliminating α from the likelihood function, β_0 add β_1 can be assessed by the maximum likelihood method.

3.2.5 Estimation of Parameters

As discussed previously, the SRGMs use test data to estimate the parameters of assumed empirical formulas, and the empirical formulas, in turn, can be expressed in terms of failure rate or expected number of failures. In general, standard ways of estimating parameters, such as the maximum likelihood method, least-squares method, and moment-matching methods, can be used. These methods are not inherent parts of the SRGMs.

Maximum likelihood and least-squares error methods often are used in estimating parameters of SRGMs. For the maximum likelihood method, the likelihood function usually is based on Equation (3-2) or (3-3), depending on the data's format, that is, number of failures in successive intervals or successive times to failure. Sometimes, graphical methods are employed, for example, Duane [1964] and Shooman [1983]. The specific solutions of the original SRGMs are set out in the respective publications; they are not duplicated here.

For the least-squares error method, depending on the parameters, different objective functions may be used. Schneidewind [1975] used as the objective function the expected number of failures; IEEE Standard 1633 [2008] suggests evaluating the failure rate of the Generalized Exponential model with data from two different times to solve for the two model parameters.

In general, the SRGMs only consider point estimates of the model parameters. However, there is no inherent difficulty in assessing the uncertainties in them. A chapter on parameter estimation in Musa's book [1987] considers uncertainties. Also, one possible way of accounting for modeling uncertainty associated with SRGMs is to use different SRGMs to demonstrate the potential variability among the estimates obtained using the SRGMs.

3.3 Applications of SRGMs

SRGMs are widely used in determining/predicting the quality and reliability of software, and in deciding when to release it. Williams [2006] compared six different SRGMs by how well they fit the first 80% of the data, and how well they predicted the rest of it. Among other differences, the six models differed in their number of parameters. The two-parameter models included the delayed S-shaped growth model, the exponential model, and the logarithmic Poisson model. The three-parameter models encompassed the imperfect debugging model, the inflection S-shaped growth model, and the logistic growth model. Four sets of data from the literature [Yamada 1983, Kapur 1990, Musa 1984, and Obha 1984], in the format of numbers of failures in various time intervals, were applied to each of the models. The total number of failures from the four data sets was, respectively, 19, 20, 59, and 109. The four data sets were of different sizes and from different periods (1980s and 1990s). The comparison was straightforward; the models were ranked according to their goodness-of-fit and prediction capability. Williams used the maximum likelihood method and one of the four sets of data to estimate a set of parameters for each model. For this, he employed data in the format of the cumulative number of failures up to the i^{th} time interval t_i. The fitness and the prediction capability were measured in terms of respective "sum of square errors" (SSE) values for each data set, that is, the sum of the square of the deviation of the failure numbers calculated/predicted using the models and the actual failure data at different times. Williams concluded that (1) the inflection S-shaped model fits the data best, and (2) the three-parameter logistic model shows the best overall ability to predict failure data, with the two-parameter delayed S-shaped model a close second. He also noticed that the performance of many of the six models varies as the volume of the data rises, and more data does not necessarily improve model performance.

It should be pointed out that there exists no SRGM that is universally superior to the other models in all applications [Lyu 1996], as evident from Williams' comparison [2006]. Stringfellow [2002] pointed out that the assumptions made for individual SRGMs often are violated in real applications, although many models empirically were deemed robust. He proposed a method for selecting the "best model(s)" to estimate the total number of failures in the software as testing progresses, and when to stop testing. However, he cautioned that the "best" reliability model can vary across systems, and even across releases, recognizing that the underlying problem is the complexity of the interactions of the factors that influence the software's reliability.

Chapter 4 of Lyu's book [1996] discusses techniques for selecting a method/model that best predicts a *specific* set of test data, and making corrections to obtain better predictions. Essentially, older data of the data set are used to predict newer data, and measures are defined to determine which SRGM makes better predictions. In addition, biases of the predictions can be identified and used to make adjustments to the predictions so that the accuracy can be improved. The limitation of these techniques is that their benefits are applicable only to the specific set of data being analyzed, not universally (i.e., an SRGM that provides the best fit for one set of data may not provide the best fit for a different data set).

3.4 Review of SRGMs

SRGMs successfully have supported decisions about testing; for example, deciding if the current version of the software meets the reliability goal, or how long should testing continue before releasing the software. The many models and terminologies in the literature suggest that unifying them is desirable, especially models in the exponential NHPP category. Unification of SRGMs in terms of the empirical formulas of an NHPP, that is, expressing the expected number of failures as a function of time, $\mu(t)$, is a trend in software reliability, as exemplified in recent publications, such as IEEE Standard 1633 [2008], Son [2009], Huang [2003], and Williams [2006]. The formulation described in Section 3.2.1 of this report is a generalization of SRGMs in terms of an NHPP.

Different SRGMs contain different assumptions about how variables, such as failure rate and expected number of failures, change with time. The SRGMs also express the assumptions in terms of empirical formulas associated with these variables, and employ test data to assess the parameters of the assumed empirical formulas. The accuracy of these models is reflected in how well their respective formulas fit the test data. While there are published comparisons of applications of different methods, often they are intended to demonstrate that a new model produces better results, that is, it fits the data better. However, consistent with the comparison in Williams [2006], no one model is universally considered to be better than the others, because all are based on assumed empirical formulas that are not applicable to all situations.

Different SRGMs define different variables and parameters, and specify different physical meanings to them. These physical meanings then are incorporated into assumptions and derivations of empirical formulas. Often, it is difficult to judge if, and how well, the assumptions for a model are satisfied. As stated by Stringfellow [2002], the assumptions for individual models frequently were violated in real applications, even though many models were demonstrated empirically to be robust. For example, an exponential NHPP model assumes that failure rate is proportional to the number of faults remaining in the software, and that faults are fixed perfectly. The former assumption is a reasonable one, leading to an exponentially decreasing failure rate. The latter assumption probably is non-conservative. However, the assumption may not be too critical because, ultimately, what counts with models based on empirical formulas is how well the model fits the data. For the same reason, many of the assumptions that have been made in different SRGMs may not need to be scrutinized too critically.

The failure rates estimated using SRGMs are generated by fitting actual test data, and therefore, should fit the data reasonably well. The goodness of the fit reflects upon the choice of the model and its associated empirical formula. Because these are empirical formulas, not laws of software reliability, a good fit in one application does not guarantee a good fit in other applications. In addition, it remains to be demonstrated that the estimated failure rates fit actual operational experience well, since it is commonly recognized that test inputs do not necessarily reflect operational environment well. In addition to finding a method/model that fits a set of test data well, it may be also useful to use multiple SRGMs to help address modeling uncertainty, as suggested in Section 3.2.

The nature of SRGMs is to estimate or predict the software reliability based on the failure data. As alternative approaches to estimating the software reliability growth, both artificial neuron

networks (ANNs) and genetic programming have been explored by making use of their historical-data-based learning and predicting capabilities [Sitte 1999 and Costa 2005].

As mentioned earlier, a review of the literature revealed that practically all available QSRMs consider the software system as a whole, not as separate modules or broken down by failure mode. However, as described in [Musa 1987], failure rates and probabilities for different categories (e.g., severities or failure types) can be modeled by allocating the estimated failure rate/failure probability for each category according to the proportion of the total number of failures represented by that category. This requires that failures reported during testing be categorized according to an appropriate scheme, which may represent additional unplanned effort for the testing, development, and assurance staff.

An intuitive use of SRGMs in supporting PRA modeling would be taking the estimated software failure rate at the end of the test period as the failure rate for the software when it is in actual operation. There are a few limitations of this use.

1. The SRGMs discussed in the preceding sections provide failure rates. Accordingly, they may be appropriate for modeling nuclear power plant (NPP) control systems, such as a digital feedwater control system, and spurious actuations of protection systems. To quantify the failure of a protection system to initiate its protective function(s), it would be more appropriate to use a failure-on-demand-based method.[7]

2. Since SRGMs are driven by test-failure data, this implies that the reliability of the software is not very high or the testing burden is significant. Therefore, it may not be practical to use these models to demonstrate very high reliability, such as that required in the nuclear industry (with the possible exception of spurious actuations of protection systems).

3. SRGMs share many of the limitations of test-based methods, as discussed in Section 5.4 (e.g., the testing environment may not represent the actual "operational profile" to which the software is exposed during operation).

[7] As mentioned previously, it may be possible to extend the SRGMs to model demand-type failures. In particular, there are two groups of methods, referred to as discrete software reliability growth methods and discrete reliability methods (which remain to be reviewed), that may be more appropriate for modeling the failure of an NPP protection system to initiate its protective function(s).

4. BAYESIAN BELIEF NETWORK (BBN) METHODS

Section 4.1 describes the BBN method, including its mathematical formulation and an example of a BBN. Section 4.2 discusses a few BBN applications to safety-critical systems. Comments on the method and its application are presented in Section 4.3.

4.1 Description of Bayesian Belief Networks

4.1.1 Mathematical Description

Pearl first introduced the BBN method in his book on "Probabilistic Reasoning in Intelligent Systems" [1988]. A BBN is a probabilistic graphical model depicting a set of random variables and their conditional independencies via a directed acyclic graph. Here, "acyclic" means the graph does not form a feedback loop. In a BBN, the nodes represent random variables, and the arcs signify dependency among the nodes. The random variables often are assumed to have discrete probability distributions [Jensen 2002], and may portray disparate information, as will be illustrated in this section through examples and applications of BBNs. Shachter [2001] suggested that, in general, continuous distributions can be used. The word "belief" is a subjective interpretation of probability often used in BBN modeling, that is, subjective probability describes one's belief or confidence in the occurrence of a particular outcome. In this report, BBN and Bayesian network (BN) are used interchangeably. A simple example BBN is used in Section 4.1.3 to demonstrate the use of BBNs in making inferences.

In probability theory, a joint distribution of *n* random variables in a graph always can be developed by using the chain rule, that is,

$$P(V_1, V_2, \cdots, V_n) = P(V_1)P(V_2 \mid V_1)P(V_3 \mid V_2, V_1)\cdots P(V_n \mid V_{n-1}, \cdots, V_1)$$

A BBN encodes a basic assumption that a node is conditionally independent of its non-descendent nodes, given its parent nodes. The basic premise represents a judgmental reasoning that provided there are known values for the parents of a node whose value is unknown currently, then no other knowledge (except that concerning the descendants of the node) will affect one's opinion about the true value of the node [Lauritzen 1988]. Each node of a BBN can be described by a local conditional probability distribution function given its parents in the graph, that is, $P(V_i \mid parents(V_i))$, where $parents(V_i)$ indicates the parent nodes of node V_i. Note that there is no specific constraint on how a variable depends on its parents. If node V_i does not have a parent node, then its local probability function is termed unconditional.

For a BBN, the joint distribution of all variables is

$$P(V_1, V_2, \cdots, V_n) = \prod_{i=1}^{n} P(V_i \mid parents(V_i)) \tag{4-1}$$

Equation (4-1) is the chain rule that is employed in making Bayesian inferences for BBNs. A BBN is defined completely by specifying every term on the right-hand side of the equation. Should additional information about the nodes become available, the joint probability distribution can be Bayesian updated, and used in making inferences.

The conditional independence encoded by the BBNs supports a more compact development of the joint distribution by minimizing the number of required parameters. In general, if there are n binary nodes, the number of parameters for constructing the full joint distribution, $P(V_1, V_2, \cdots, V_n)$, is on the order of 2^n. Similarly, the number of parameters needed for the factored form of the joint distribution is $\sum_{i=1}^{n} 2^{parents(V_i)}$, that is, not invariably larger than $(2^n - 1)$ [Langseth 2007]. The more compact joint distribution represents the modeler's understanding of the system being modeled, and reduces data and calculation needs.

The conditional independence of a BBN can be represented by the graphical property of d-separation or d-connection in graph theory. If two sets of nodes X and Y are d-separated in the graph by a third set, Z, then the corresponding variable sets X and Y are independent conditionally given the variables in Z. Jensen's book on Bayesian Networks and Decision Graphs [2002] details the definitions of d-separation and d-connection, and conditional dependence. The following examples illustrate the concept of d-separation.

In Figures 4-1(a) and 4-1(b) nodes A and B are all d-separated (conditionally independent) given C (i.e., C is in Z or is known) while in Figure 4-1(c) nodes A and B are d-connected (conditionally dependent) given C.

A and B are conditionally independent of each other given C (1) in Figure 4-1(a), that is, provided C is known, knowing A does not add more information about B; and, (2) in Figure 4-1(b) because information about A and B can all be inferred separately by knowing C, the common parent of A and B.

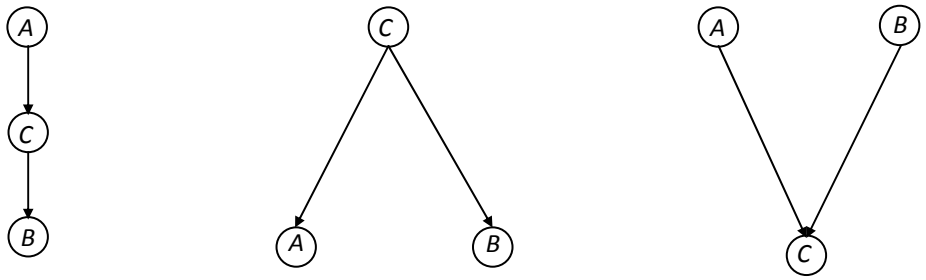


Figure 4-1 Types of network fragments in BBNs

In Figure 4-1(c), nodes A and B are mutually independent because they have no common parents; this is equivalent to saying that nodes A and B are d-separated by the empty conditioning set [Langseth 2007]. On the other hand, nodes A and B are conditionally dependent given C because knowing C supports inferring information on A and B. Interestingly, the network fragment in Figure 4-1(c) represents the case where independent causes (denoted by A and B) become dependent by conditioning on a common effect (of A and B, represented by C). Figure 4-1(c) also shows that the conditioning set determines whether two sets of nodes are d-separated. For example, A and B are dependent on each other in Figure 4-1(a), that is, if nothing is known about C (the conditioning set becomes empty in this case), A provides some

information about B and they become dependent. In Figure 4-1(b), lacking any knowledge of C (the conditioning set again is empty), some information about B can be inferred from knowledge of A, so making A and B dependent. On the other hand, if something is known about C in Figure 4-1(c), then information on A may disclose some knowledge about B.

4.1.2 Bayesian Inference

Bayesian inference using BBN can be done in the standard way, as in Helminen [2001]. In general, once evidence is obtained, the joint distribution of the nodes, that is, Equation (4-1), is updated. In particular, the distribution of a particular node, for example, one representing the rate or probability of software failure, is generated by integrating or summing (marginalizing) the irrelevant variables. Marginalizing irrelevant variables can be time-consuming, and methods were developed for performing the task efficiently. In particular, a “variable elimination” method is illustrated in the example BBN described in Section 4.1.3.

4.1.3 An Example BBN

The example demonstrates that a BBN allows for updating "beliefs" in light of evidence by calculating the conditional probabilities of the nodes given that some nodes have been observed and some evidence is available.

Murphy [1998] presented an example wherein all variables represented by the nodes are binary ones, that is, each node has two values; true (T) or false (F), as shown in Figure 4-2. An event, the grass is wet (W=T) may reflect two causes, that is, (1) the sprinkler is on (S=T), or (2) it is raining (R=T). Also, if it is cloudy, the chance of raining (R=T) may increase; if not, the chance of turning on the sprinkler may rise to water the grass. Thus, each node has an associated conditional probability table (CPT), as depicted in Figure 4-2. The sum of probabilities for each column is 1.0.

Using the chain rule of probability, the joint probability distribution of all nodes is

$$P(C,S,R,W) = P(C) \times P(S \mid C) \times P(R \mid C,S) \times P(W \mid C,S,R) \tag{4-2}$$

Using the conditional independence encoded in the BBN, that is, the chain rule for BBN, the joint probability distribution of all nodes becomes

$$P(C,S,R,W) = P(C) \times P(S \mid C) \times P(R \mid C) \times P(W \mid S,R) \tag{4-3}$$

namely, a more compact expression of joint probability, even in this simple BBN.

One major task in studying the BBNs is to undertake probabilistic queries or inferences using Bayes' rule, and some techniques such as marginalizing, that is, summing out over "irrelevant" variables. Two types of inferences or reasoning can be completed using a BBN; inductive inference that diagnoses a cause for a given effect (called bottom-up reasoning), and, deductive inference that predicts an effect for a given cause (called top-down reasoning).

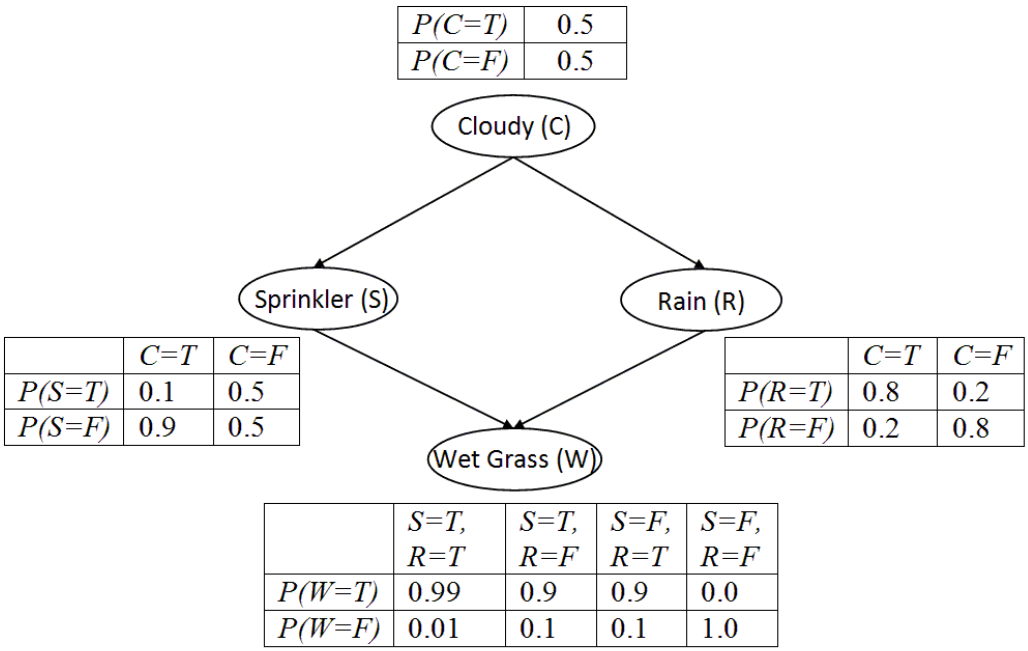


Figure 4-2 An example for a BBN inference

An example for the former inference for the BBN in Figure 4-2 is to question that if the grass is wet, is it more likely to be caused by the rain or the sprinkler? This query is solved by comparing the probability $P(S=T \mid W=T)$ to $P(R=T \mid W=T)$. Since the joint probability of occurrences of A and B is

$$P(A,B) = P(A \mid B) \times P(B)$$

then $P(A \mid B) = \dfrac{P(A,B)}{P(B)}$. Therefore,

$$P(S=T \mid W=T) = \frac{P(S=T, W=T)}{P(W=T)} = \frac{\sum_{c\in[T,F]} \sum_{r\in[T,F]} P(C=c, S=T, R=r, W=T)}{\sum_{c\in[T,F]} \sum_{s\in[T,F]} \sum_{r\in[T,F]} P(C=c, S=s, R=r, W=T)}$$

From Equation (4-3), using the associated CPTs shown in Figure 4-2, we obtain

$$P(S=T \mid W=T)$$

$$= \frac{\sum_{c\in[T,F]}\sum_{r\in[T,F]} P(C=c)\times P(S=T\mid C=c)\times P(R=r\mid C=c)\times P(W=T\mid S=T,R=r)}{\sum_{c\in[T,F]}\sum_{s\in[T,F]}\sum_{r\in[T,F]} P(C=c)\times P(S=s\mid C=c)\times P(R=r\mid C=c)\times P(W=T\mid S=s,R=r)}$$

$$= \frac{\sum_{c\in[T,F]}(P(C=c)\times P(S=T\mid C=c)\sum_{r\in[T,F]}(P(R=r\mid C=c)\times P(W=T\mid S=T,R=r)))}{\sum_{c\in[T,F]}(P(C=c)\sum_{s\in[T,F]}(P(S=s\mid C=c)\sum_{r\in[T,F]}(P(R=r\mid C=c)\times P(W=T\mid S=s,R=r))))}$$

$$= \frac{0.2781}{0.6471} = 0.4298$$

Note that the order of marginalizing the variables does not affect the results, but may alter the efficiency of the calculation. In this example, the calculation involves the so-called "variable-elimination" method [Zhang 1996]. The key idea therein is to "push sums in" (to the right) as far as possible when summing (marginalizing) out the irrelevant terms. For example, in the above equation, the summation over values of the variable *R* is moved to the right of those terms that do not include *R*, such that the summation over *R* does not need to be repeated for those terms. The example also demonstrates how tedious and time-consuming the calculation can be, even when using the chain rule for BBNs.

Similarly, $P(R=T\mid W=T)$ can be calculated, as shown below.

$$P(R=T\mid W=T) = \frac{P(R=T,W=T)}{P(W=T)}$$

$$= \frac{\sum_{c\in[T,F]}\sum_{s\in[T,F]} P(C=c,S=s,R=T,W=T)}{\sum_{c\in[T,F]}\sum_{s\in[T,F]}\sum_{r\in[T,F]} P(C=c,S=s,R=r,W=T)}$$

$$= \frac{0.4581}{0.6471} = 0.7079$$

Comparing the calculated $P(S=T\mid W=T)$ and $P(R=T\mid W=T)$, it can be concluded that the grass is more likely wet from rain.

An example of deductive inference is to solve the likelihood that the grass is wet, given that the sky is cloudy, that is, $P(W=T\mid C=T)$, which is approached in the same way.

$$P(W=T\mid C=T) = \frac{P(W=T,C=T)}{P(C=T)}$$

$$= \frac{\sum_{s\in[T,F]}\sum_{r\in[T,F]} P(C=T,S=s,R=r,W=T)}{\sum_{s\in[T,F]}\sum_{r\in[T,F]}\sum_{w\in[T,F]} P(C=T,S=s,R=r,W=w)}$$

In these inferences, the "variable-elimination" method is used to generate the "exact" inference, that is, the full summation (or integration) over discrete (continuous) variables. Murphy [2005] gives other exact inference methods as well as approximate ones.

4.1.4 Software Tools for Bayesian Inference

The complexity of the BBN inference is apparent from the simple example given above, and from Littlewood's [2007] two-legged BBN study. Murphy [2005] lists BBN software packages that can perform the inference; some major ones are described briefly by Korb [2004] in his Appendix B, available at http://www.csse.monash.edu.au/bai/book/appendix_b.pdf. Note that most of the commercial software packages have free versions that are restricted in different ways, for example, the model's size is limited or the models cannot be saved.

4.1.5 Building BBNs

In general, three tasks are involved in BBN applications, that is, elicitation and construction of a BBN model, elicitation of the probabilities of a model, and computation. The computation of BBNs, that is, inferences, has been covered in the above discussion. Building Bayesian Networks requires information or evidence from two main sources: expert knowledge and statistical data [Langseth 2007]. For example, the main sources of reliability evidence for a safety critical system may include design features, development processes, testing, and operational experience, as Gran [2002b] suggested.

Usually, a group of experts in the domains of BBN and the specific application build the BBN. The structure of the BBN and assumptions encoded in it are decided by the BBN experts using information solicited from experts in the application domain.

Dahll [2002] described the process of building a BBN for a specific application. Some nodes are denoted as "observable" ones because they represent the different observable properties about the application. The target node of the BBN is the node of interest. Other nodes are referred to as "intermediate" nodes. Starting from the target node, edges can be drawn to other nodes that affect the target node. Then, new nodes can be connected to these nodes by combining the relevant information for the application. Littlewood [2007] employed a two-legged[8] (testing and verification) approach to build a simplified BBN for software-based system without considering other factors that might significantly affect system development, such as quality assurance and the development process itself.

Although building the BBN usually starts from developing nodes representing high-level information, that is, an arrow goes from a higher abstraction to a lower one, or from a more general concept to a more detailed one (as discussed in Section 4.2.2), there is no unique way of doing this. No general guideline can be followed that will guarantee the correctness of dependencies in the BBN. The details of constructing BBNs are application-specific; some examples of applications using BBNs are discussed in the following section.

[8] In this context, "legs" refer to sources of reliability evidence to be used in constructing the BBN, such as those discussed earlier in this section.

4.2 Applications of BBN Methods

4.2.1 Multi-Legged Arguments

Littlewood [2007] modeled a multi-legged argument representing the impact of diversity (i.e., diverse arguments) on confidence in safety claims for software-based systems using a Bayesian belief network that combined the disparate evidence and assumptions forming the different legs. Each leg supports different reliability claims to be made at different levels of confidence in terms of different (reliability, confidence) pairs.

Littlewood interpreted confidence as probability in his study, that is, the Bayesian subjective strength of belief in the claim. The nodes (variables) of the BBN included the following:

(1) the system's unknown, true probability of failure on demand (a continuous variable represented by S), or a probabilistic claim about it;
(2) the system's specification (symbolized by a discrete variable Z with a value of "correct" or "incorrect");
(3) the conclusion from verifying the system against its specification (represented by a discrete variable V with a value of "verified" or "not verified");
(4) the oracle used in testing the system (denoted by a discrete variable O that is either "correct" or "incorrect");
(5) the system's test results (typified by a discrete variable T indicating whether "no failures" or "failures" were uncovered); and,
(6) the acceptance (or otherwise) of the final claim (represented by C) on whether the system is suitable for use, that is, the claim is "accepted" or "rejected."

In Littlewood's study, the goal of the BBN was to obtain the posterior distribution of (a) the probability of failure on demand (pfd) of the system or a probabilistic claim about the pfd, that is, the S node, and (b) whether the claim on the system's suitability should be accepted, that is, the C node, based on observations of the other variables in the BBN. The two legs of argument, namely, the verification and testing nodes, were considered with other nodes. These two nodes are the BBN's observable nodes. The updated joint probability distribution $P(C, S \mid observations)$, in particular, the value of $P(C = accepted,\ S > 10^{-3} \mid observations)$ was of primary interest. The BBN model was also used to consider the two legs of argument separately to investigate their individual influences on the claims, effectively producing two separate single-legged BBNs.

From a set of simplifying and conservative assumptions, node probability tables with analytical expressions with parameters were developed representing the conditional relationships of the nodes. For example, a beta distribution was assumed for the probability of failure on demand (the S node), and different sets of parameters of the distribution were assigned depending on whether or not the specification is correct (the Z node). In this way, analytical solutions were obtained for the BBN, offering further insights than just the numerical solutions using the defined node probability tables. In particular, an analytical expression for $P(\ S > 10^{-3} \mid ideal\ \ observations)$ was derived, that is, the probability that the failure probability is higher than 10^{-3} given that testing and verification succeed.

The study also revealed that adding a diverse second argument leg (e.g., having both a testing argument and a verification argument) can increase confidence in a dependability claim. However, it also generated counterintuitive results for some numerical values used for the parameters in the model. In particular, for the two-legged model, for some specific parameter values, it is possible that (1) successfully verifying a system against its specification may reduce the confidence that was obtained from failure-free testing, and (2) an increase in the number of successful tests may slightly lower confidence in claims about pfd. For the single testing leg BBN, evidence that no failures were observed in the test cases decreases the confidence in claims about pfd for some parameter values. Similarly, for the single verification leg BBN, successful verification lowered the confidence in system failure probability when using some specific parameter values. How these particular cases led to counterintuitive results was attributed by Littlewood [2007] to "subtle interplay between (simplifying and conservative) assumptions and evidence, both within and between legs." The authors also suggested that "the counterintuitive results may be not believable when real experts assess real systems." They acknowledged that the model is oversimplified, and warned against naïvely trusting in the results of a numerical analysis of a BBN.

4.2.2 BBN Applications to Software Reliability of M-ADS and Motor-Protection Relay Systems

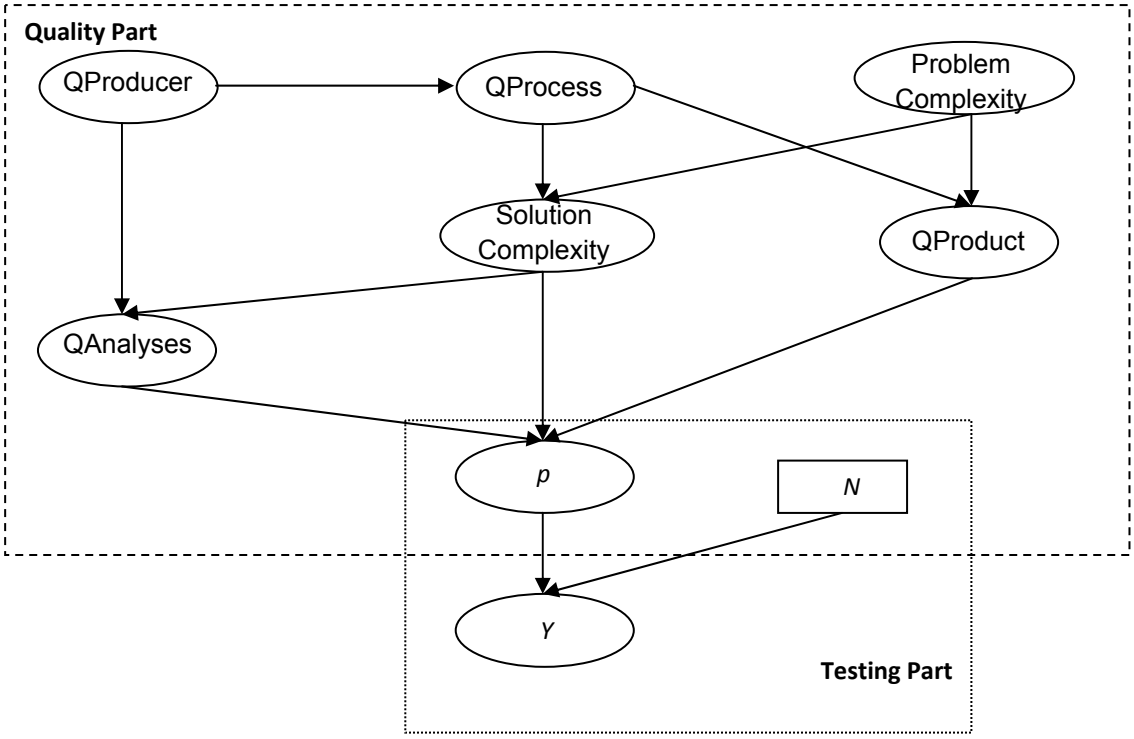
Applying the BBN method to assessing software reliability of different systems was discussed in several papers and reports representing the work at the VTT Technical Research Center of Finland [Helminen 2001, 2003a, 2003b, 2005, and 2007] and the Halden Reactor Project [Gran 2002a and 2002b]. Gran [2002a] used different approaches in studies of the software reliability of two different systems, a helicopter location identification system (M-ADS) and a motor-protection relay system (SPAM 150 C) produced by ABB Substation Automation. These studies encompass both expert judgment and quantitative evidence, such as test data. The qualitative evidence (soft evidence) represented the experts' judgment on the quality of the development process and the system's design features, while the quantitative evidence (hard evidence) included both testing and operational data that are directly measurable statistical evidence [Helminen 2001]. The M-ADS was developed following avionic standard DO-178B [RTCA 1999], and the BBN model considered the 10 lifecycle stages defined in the standard using system demand failure probability as the measure. Gran [2002b] gives details about the study. In the study of SPAM 150 C [Helminen 2003a and b], expert judgment was applied in a six-step process to estimate failure parameters, such as the percentiles of the lognormal distribution representing the frequency of system failure. The experts produced a prior distribution that was further Bayesian updated using test and operational data, as detailed in Helminen [2003a] and [2003b].

A more recent study of a different motor-protection relay system, that is, REM 610 [Helminen 2005 and 2007], used a similar approach as for the SPAM 150 C. The studies of the M-ADS, SPAM 150 C, and REM 610 are further described below.

- Study of the M-ADS [Gran 2002a and b]

The BBN was built in three stages: (1) eliciting and constructing a BBN model, (2) elicitation of the probabilities of the BBN model, and (3) carrying out the computations. Separate pieces of information were collected and represented by the BBN nodes that were connected by edges expressing the causality between the nodes. A directed edge from node *A* to node *B* means that a "belief" in *A* implies expectations on *B*. The BBN was built starting with nodes representing high-level information and then moving down to nodes with lower level information, that is, an arrow goes from a higher abstraction to lower one, or from a general concept to a more detailed one.

A high-level BBN was built first that included the "testing"-part and the "quality"-part (Figure 4-3). The testing-part described the connection between observing "0 failures in *N* tests," and the "failure probability *p* of the system." For a defined number of demands *N* with a constant failure probability *p*, the random number of failures *Y* was assumed binomially distributed. The quality-part consisted of four high-level quality nodes (i.e., quality of producer, process, analyses, and the product); this part also encompassed the complexity of the problem and its solution. Each of the quality nodes was a top-node in the lower-level BBN, and influenced the 10 lifecycle stages of the avionic standard DO-178B, which subsequently were linked to other nodes that represent the objectives of the lifecycle stages. The higher-level and lower-level BBNs then were merged together.



(1) QProducer: Quality of producer; (2) QProcess: Quality of production process; (3) QProduct: Quality of product; and, (4) QAnalysis: Quality of analysis.

Figure 4-3 Quality part and testing part of the high-level BBN [Gran 2002a]

To compute the probabilities of variables in the BBN model, prior probability density functions (pdf's) were assigned to the parent nodes, and then conditional pdf's were ascribed to descendant nodes to reflect the influence of the parent nodes. The variables represented by the nodes in a BBN are either continuous or discrete, accordingly eliciting continuous or discrete pdf's. However, conceptually it is easier to elicit pdf's for discrete variables, and less effort is needed in the associated computation to generate the inference.

Qualitative evidence mainly was used in deriving the prior estimation for the reliability of the system, and quantitative evidence was employed to update this judgment. Note, the difference between soft evidence and hard evidence sometimes is not clear because testing and/or operational experience may have qualitative features, as suggested in Helminen [2001]. The prior estimate of the failure parameter of the system was derived from soft evidence during the system's development, pre-testing, and design evaluation phases before the system was deployed. It then was updated, incorporating the hard evidence obtained after the system came into operation [Helminen 2001]. [Gran 2002b] undertook a sensitivity study for the failure probability based on the uncertainty in the observations.

In Gran [2002b], a high-level BBN (Figure 4-4) was developed, which differs from that in Figure 4-3 in that the quality-part in Figure 4-4 includes a node, *N*-hypothetical, intended to express the equivalence between the information in the quality-part and that of the system tested with *N* randomly selected inputs without failure. The *N-hypothetical* node was modeled in terms of a subjectively developed conditional probability table given its three parent nodes as shown in Figure 4-4 [Gran 2002b].

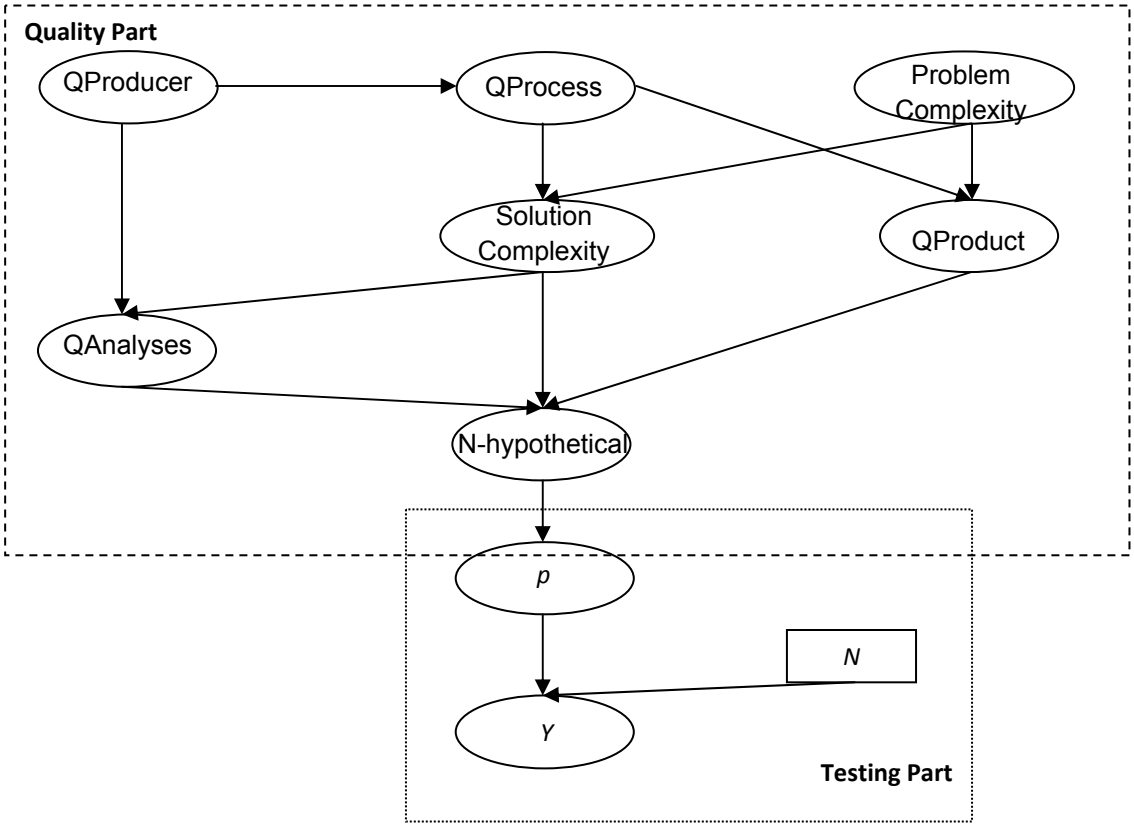


Figure 4-4 Quality part and testing part of the high level BBN with N-hypothetical node [Gran 2002b]

- Studies of Motor-Protection Relay SPAM 150 C [Gran 2002a], [Helminen 2003a and 2003b]

The case studies in Gran [2002a], that are further described in Helminen [2003a and 2003b], considered revisions to the software of a motor-protection relay, SPAM 150C, manufactured by ABB Substation Automation. The studies considered the software going through different revisions throughout its operating life. Helminen [2003b] describes a six-step process for using expert judgment to formulate a prior estimate of the motor-protection relay failure rate for the first version of the software. In the first two steps, the experts estimated score and weight values for the different software development phases of the system. In the third step, a weighted total score value for each expert was calculated. In order to convert the total score value of the expert to a failure frequency distribution, in step four, each expert was asked to estimate failure distributions for two or three different score values. More specifically, the expert was asked to give failure frequency percentiles of given score values. Failure frequency distributions for the score values were then estimated by fitting lognormal distributions to the percentiles given by the expert. In step five, a failure frequency distribution was estimated for each expert using the method of least square with the total score value calculated in step three and the lognormal distributions estimated in step four, for different score values. In step six, a combined failure frequency distribution, that is, the prior distribution for the system failure frequency, was calculated by averaging the frequency distributions estimated for individual experts. This prior distribution was Bayesian updated using information gleaned during the operation of this version of the software, represented as the approximated amount of working years and the number and types of software faults encountered.

The underlying model for the Bayesian network was a lognormal-Poisson model because (1) the combined prior (p) was assumed to be a mixture of lognormal distributions and (2) the number of software faults found in a certain time period (Y) followed a Poisson distribution. Four scenarios were explored, that is, a neutral approach and a conservative approach with two different data sets. In the neutral approach, the prior distribution of a new version of the software was assumed to be equal to the posterior distribution of the previous version. In the conservative approach, the assumption was that a software change always negatively affects reliability. In this case, the prior distribution of a new version of the software was assumed to be equal to the posterior distribution of the previous version plus a subjectively assessed change. This change in the system's failure rate after each software revision was modeled explicitly using a lognormally distributed random variable, the parameters of which were estimated by expert judgment [Helminen 2003b]. The two different data sets used included one that contained the number of clear software faults encountered in different versions of the software, and a second that contained the clear software faults plus the so-called "inconveniences" (i.e., software faults that caused no trouble to the customer or caused inconvenience to a small fraction of the customers) [Helminen 2003b].

The BBNs of the studies were modeled in the format of the WinBUGS program. Note, that a WinBUGS model also is a BBN because it encodes the same "conditional independence" as do the general BBNs [Spiegelhalter 2003].

- Study of Motor-Protection Relay REM 610 [Helminen 2005 and 2007]

Helminen [2005 and 2007] assessed the software reliability of a different motor-protection relay system, REM 610, using a similar approach to that employed in studying the SPAM 150C system. Four protection functions were chosen for the study. The failure rates of two different failure modes were evaluated, that is, failure to trip and spurious trip of the system. The study encompassed four different phases of software development, and experts estimated the percentiles of the lognormal distribution representing the failure rate for each failure mode. The estimates then were used to derive a joint distribution for the failure mode. Ten designers of the system participated, forming four assessment groups that represented the system's different developmental stages. The experts were assigned to these groups according to their position and involvement in the process. Before taking part in the quantitative session, the experts first underwent training on the assessment process, and then took part in a qualitative session developing simple logical models of the protection functions, identifying dependencies of the functions, and highlighting potential uncertainties or inconsistencies in the design information. Equal weight was given to prior estimates from individual experts. These priors for the same failure mode of the four protection functions were merged into a joint estimate using WinBUGS. The operational data, which was relatively substantial, was expressed in terms of the estimated operating years and the reported software faults of REM 610. The number of REM 610 operating years and the reported number of software faults were, respectively, 140 years and zero faults (for both failure modes of the software). The priors were updated with these operating data to generate the posterior distributions of the failure modes.

4.2.3 BBN Applications to Reactor Protection System Software

Two papers [Eom 2004 and 2009], documenting the BBN work done by the Korean Atomic Energy Research Institute, are described below.

The first paper [Eom 2004] suggests that obtaining a precise quantitative estimate of the software reliability of safety-critical systems is nearly impossible because of the many qualitative characteristics of the software that cannot be measured directly. Hence, a BBN method was adopted for evaluating the reliability of the software embedded in a safety-critical system, combining both qualitative and quantitative evidence. This included evaluating the quality of the software products (such as the requirement specifications, design specifications, code, and the final software product in the binary form) throughout the phases of the development life cycle and considering the test results (i.e., number of tests without faults or testing time without faults). To illustrate the feasibility of the BBN method, this case study [Eom 2004] was limited to assessing the quality of the software requirement specification for an RPS in a nuclear power plant. The identified variables for the BBN characterized the software function (accuracy, functionality, reliability, robustness, safety, security, and timing) and development process (completeness, consistency, correctness, style, traceability, unambiguity, and verifiability). A set of questions was designed to assess the quality of the software requirement specification for each variable. The target node of the BBN indicated whether "T100_Software_Requirement_Specification" is acceptable or not, and the target node's variable had two states, that is, "acceptable" and "unacceptable". The BBN encompasses 166 nodes connected mainly according to causal relationship. The Sherman Kent rating scale [Kent 1964], was used to convert some qualitative evidence to quantitative inputs to the BBN, for example, "likely" would represent a probability of 0.6. A software tool, HUGIN [HUGIN

2010], was used to perform the Bayesian update. This paper does not describe how software failure probability was estimated using the BBN model.[9]

In [Eom 2009], the researchers proposed the concept of a generalized BN template that is not restricted to a particular software development environment, thereby reducing the effort to build BNs for software developed under different environments. The recently formulated object-oriented BN (OOBN) and dynamic BN (DBN) can be used to implement the general BN template. An OOBN simplifies the work to create large and repetitive BNs by creating BN subnets which are called a "class" in Object-Oriented methodology terms [Koller 1997]. A DBN allows time-indexed variables [Bangs 2000]. In Eom [2009], a general phase BN (an OOBN) is a BN subnet that models a single software development phase, and consists of the following six classes (i.e., nodes):

1. Residual defects in the final product of a previous phase, representing the number of remaining defects therein;
2. Inspection quality in a current phase, denoting the number of defects removed by inspections and traceability analyses;
3. Residual defects in a current phase, representing the number of remaining defects in the final product of the current phase;
4. Process quality in a current phase including seven process characteristics, that is, consistency, verifiability, unambiguity, traceability, style, correctness, and completeness;
5. Functional quality in a current phase that again includes seven characteristics, that is, accuracy, timing, reliability, robustness, security, safety, and functionality; and
6. Defect prediction class that models a causal relationship predicting defect numbers in the current development phase.

Thus, a DBN for the software development life cycle (SDLC) was created by linking BNs for the individual software development phases in terms of the causal relationship of the features represented by classes (nodes) in the BNs (e.g., the node "Residual defects in a current phase" of phase i and the node "Residual defects in the previous phase" of phase (i+1) are connected by a directed edge).

A case study [Eom 2009] was performed for the software of a digital RPS following the approach described above. The inputs for the model mainly were derived from the verification and validation (V&V) results of the software. As an example, a node "Functional quality in a current phase" for phase "Coding" was calculated from its descendant nodes representing the original node's properties. Inputs to other nodes in the BN were calculated similarly from the V&V results. Then, quantitative results and associated findings for each end node in the BN (e.g., the residual defects in the last phase of the SDLC of the RPS software) are obtained via Bayesian inferences.

[9] Through private communication with one of the authors of the paper, Dr. Kang, it was confirmed that the model was used in quantifying software failure probability, but the documentation is not publicly available.

4.3 Review of BBN Methods and Applications

The BBN method is a general method, accounting for the influence or effects of one event on another. Often, it is employed to model the subjective opinions of experts and combine disparate information. Another advantage is that uncertainties in parameters can be accounted for automatically within the scope of the model. The BBN method is considered as a tool that affords structure in modeling, though the analyst needs to be aware of the underlying assumptions of the BBN method when applying it to a particular model. A few software tools are available to facilitate the analysis (as identified in Section 4.1.4).

Conditional independence or dependence is the fundamental concept of the BBN method. Deciding on how to characterize the dependencies between nodes is probably the most important and difficult part of developing a BBN model, and it is heavily dependent on the analyst's judgment and knowledge. As described in Section 4.1.1, an advantage of the BBN method is reflected in the relative simplicity of the BBN chain rule, compared with full dependency among the random variables; that is, the dependencies are localized and represented by conditional probability distributions. However, the analyst must decide upon such dependencies. For example, the analyst must verify that a node in one corner of a BBN is related to another node in a different corner only through some intermediate nodes of the BBN, and there is no direct dependency between the two. Such judgments may be difficult to verify.

The BBN method is promising as a potential way to quantify software reliability, as illustrated in the applications performed at the VTT Technical Research Center of Finland and the Halden Reactor Project. A strength of the method is its ability to aggregate disparate information about software (e.g., aggregation of software failure data and quality of software lifecycle activities that is assessed using expert elicitation [Gran 2000a and 2000b]) and to include parameter uncertainties as a part of the modeling. However, there are challenges in developing a BBN that takes full advantage of these capabilities, including the substantial development effort that is needed, the expertise of the BBN developers, the qualification of any experts used to elicit information, and the availability of thorough documentation of the software development activities. Another challenge is that qualitative evidence (e.g., the impact of software development quality on software reliability) needs to be quantified. Since there may not be sufficient available data to "anchor" the conversion of the qualitative information to quantitative values, the uncertainty in the resultant quantitative estimates from the experts may be very large, which may make it difficult to demonstrate the small failure probabilities often associated with safety-related systems (a limitation common to many QSRMs).

The BBN method also can be used in a more traditional way. For example, its use in performing Bayesian inferences [Helminen 2003a] using WinBUGs essentially is a standard Bayesian updating of a prior distribution obtained via expert elicitation with available data. Similarly, traditional Bayesian analyses undertaken as a part of a PRA can be considered applications of the BBN method.

Some comments on a few applications of the BBN method are given below. Note, no comments are provided on the Korean Atomic Energy Research Institute (KAERI) application of BBN to an RPS because no publicly available information exists regarding quantification of software failure probability/rate (i.e., [Eom 2004] only describes the calculation of the probability of whether the software requirement specification is acceptable or not).

1. Multi-legged Arguments [Littlewood 2007]

   The consideration of multi-legged arguments in Littlewood [2007] demonstrated, in terms of a seemingly simple BBN, a potentially complex interplay among the nodes. It identified a few counterintuitive results for some numerical values used for the parameters in the BBN model, and offered tentative explanations for them. It is desirable to have a way of systematically identifying the conditions under which counterintuitive results are generated, so they can be avoided.

2. Analysis of a helicopter location identification system (M-ADS) [Gran 2002a and b]

   The study is a representative/prototype BBN analysis of a safety-critical software system. Essentially, it uses expert judgment on the quality of the software development process and system design to estimate a prior distribution of the probability of system failure, and combines it with test and operational data in a traditional Bayesian analysis. In [Gran 2002a], the expert judgment was assessed to be equivalent to observing no failures in 1000 trials. While a prior distribution of system failure based on data of this magnitude might not be appropriate for assessing the probability of failure for an RPS in an NPP (which is expected to have a much lower failure probability), there is no reason why this method should not be able to account for any type of expert opinion, which potentially could justify much lower failure probabilities (i.e., result in an equivalent data set of no failures in a much larger number of trials).

3. Motor-protection relay systems [Helminen 2003a, 2003b, 2005, and 2007]

   The studies of two motor-protection systems followed almost the same approach to perform a Bayesian update using operational data. Expert opinion elicitation was used to estimate the prior distribution for the Bayesian analysis. Unique to SPAM 150 C study is the way in which the experts arrived at an estimate of the prior failure rate distribution for the initial software version, and the changes in failure rate due to revising the software. In this approach, it is unclear how the failure rate distribution was estimated using the least square method, as described in Step 5 of the process (see Section 4.2.2). Another issue is that for a motor-protection relay system, which is a system that takes protective actions when an abnormal condition occurs, it may be more appropriate to model the system in terms of a failure probability on demand. The use of failure frequency in these studies seems to include the frequency with which demands to the system occur. Therefore, the results depend on the specific demand frequency, and are not applicable in cases where the demand frequency differs significantly.

5. TEST-BASED METHODS

5.1 Introduction

Test-based methods essentially are those employing standard statistical methods to the results of software tests, and possibly to operational data, in the same way as hardware data is analyzed. The data is assumed to represent random samples from the software's operational environment. In this report, test-based methods are mainly are used to demonstrate that a great amount of data is needed to demonstrate the high reliability of a safety-critical software program. Other QSRMs considered in this report employ the same type of software failure data, but include more software-specific information in their modeling.

Test-based methods for both demand and time-based failures can be formulated. Due to the importance of demand failures for safety-related protection systems of a nuclear power plant, this section concentrates on this type of failure. For control systems, a time-based approach is more appropriate; details are available in Littlewood [1997] and Butler [1993].

Dahll, et al. [2007] state, "Testing means to execute a program with selected data and check the answer against an oracle." An oracle is a decision mechanism for judging the outcome of executing a program as successful (acceptable) or failed. Here, the term "program" signifies a procedure within a complete software, the software of a major component of a digital system in an NPP, or the software of the entire system.

Using a test-based method to quantify a program's probability of failure requires completing the following main tasks: (1) generating test cases via the program's expected "operational profile,"[10] (2) carrying out the tests, and (3) applying the method to quantify the program's probability of failure. Purposely, this method draws statistical conclusions about this probability from the test results.

The test-based methods described in this section are divided into two types according to how testing is done. The two main types of testing are white-box testing and black-box testing, defined by Pressman [2001] as follows:

1. "White-box testing, sometimes termed glass-box testing (or gray-box testing), uses the control structure of the procedural design[11] to derive test cases. Test cases so derived (1) guarantee that all independent paths within a module are exercised at least once, (2) exercise all logical decisions on their true- and false-sides, (3) execute all loops at their boundaries and within their operational bounds, and, (4) exercise internal-data structures to ensure their validity.

---

[10] Chapter 5 of Lyu [1996] defines operational profiles, and describes their development and use in testing. An operational profile consists of disjoint "operations" that the software can execute, along with the probability with which the operations will occur. The operations of an operational profile are partitions of its input space. An operation is a group of runs that typically involve similar processing. A run represents the smallest division of software processing that is started by external demand.

[11] This type of testing makes use of information on the internal structure of the software in deriving test cases, as exemplified by the methods described in Section 5.3.

2. Black-box testing, also called behavioral testing, focuses on the software's functional requirements. Using this kind of testing will deliver sets of input conditions that will exercise fully all the functional requirements for a program."

Subsection 5.2 describes two black-box test-based approaches for estimating the probability of failure of a program, that is, the frequentist and Bayesian approaches, which are based on two different interpretations of probability. Singpurwalla [1999] defines the frequentist approach as the relative frequency of an event after indefinitely repeated trials under "almost identical conditions," and the Bayesian approach as the degree of belief that a person (or a group) has about the occurrence of an event. Both approaches can give estimates of the value of the probability of failure of a program; they are briefly discussed is Section 5.2 for the case in which there are no failures after *n* tests. As described therein, an upper confidence bound can be derived for this probability at any confidence level. Haapanen, et al. [2000] present the frequentist and Bayesian upper-confidence bounds for the case wherein failures occurred during testing.

Subsection 5.3 briefly discusses two methods that use data obtained from tests, but also consider the internal structure of a software program, that is, white-box-testing methods.

A few unique issues are associated with using standard statistical methods with software test data. Their significance and the resulting limitations on estimating software failure probabilities are discussed in Section 5.4. In general, these issues apply to every QSRM that utilizes test data, that is, BBN methods and reliability growth methods.

5.2 Black-Box Test-Based Methods

As mentioned previously, test-based methods essentially are those employing standard statistical methods to the results of software tests, and possibly to operational data, in the same way as hardware data is analyzed. Black-box test-based methods consider a software program as a single entity, take random samples from its input space, determine if the outputs are correct, and use the results in standard statistical analyses. The statistical analyses based on the two different interpretations of probability (i.e., the frequentist approach and Bayesian approach) are summarized below.

5.2.1 Frequentist Approach

The basic frequentist approach considers that testing a program follows a Bernoulli process, namely, a series of independent, dichotomous trials, where the two events at each trial can be either success or failure. The parameter θ is the probability that a test case randomly selected from the operational profile entails a failure, and θ is constant (i.e., remains unchanged from trial to trial). In sampling from a Bernoulli process, the binomial model gives the probability of observing exactly x failures in n independent trials. Accordingly, if X is a discrete random variable that counts the number of failures in this process, and the possible values of X are $x = 0, 1, 2, \ldots, n$, then the associated probability density function (pdf) of X is

$$g(x \mid \theta) = \Pr\{X = x\} = \binom{n}{x} \theta^x (1-\theta)^{n-x} \tag{5-1}$$

where θ is an unknown constant. Actually, the notation of the conditional probability $g(x|\theta)$ signifies that the probabilities related to X are conditional upon the unknown value of θ.

Using Equation (5-1), the probability of no failures in n tests is

$$g(0 \mid \theta) = \Pr\{X = 0\} = (1-\theta)^n \tag{5-2}$$

The upper confidence bound for θ, θ_u, at any confidence level is derived from Equation (5-2) [Haapanen, et al. 2000], and is given by

$$\theta_u = 1 - (1 - \gamma)^{1/n} \tag{5-3}$$

where γ is the confidence level (expressed as a decimal less than 1.0). This upper confidence bound means that if a large number of test cases are undertaken, then 100γ% of the test cases are such that the true failure probability is covered by the interval [0, θ_u] and, accordingly, in 100(1-γ)% of test cases the true failure probability is larger than the upper confidence bound. Thus, there is a 100(1-γ)% chance to erroneously judge the failure probability. The choice of confidence level depends on the possible consequences of such mistaken conclusions, which rest upon the outcomes of system failure.

The number of successful tests (i.e., tests whose outcomes were judged successful) required to demonstrate that the failure probability is bounded by θ_u at confidence level γ, is obtained from Equation (5-3) as

$$n = \frac{\ln(1-\gamma)}{\ln(1-\theta_u)} \tag{5-4}$$

This equation shows that 3 x 10^5 tests must be successful (no failures observed) to demonstrate a probability of failure of the software per demand of 10^{-5} with a confidence level of 0.95 (i.e., 95% confidence).

5.2.2 Bayesian Approach

The Bayesian approach is a straightforward application of Bayes' theorem. Miller, et al. [1992] derived the Bayesian methodology described here. The likelihood function is the binomial model (Equation [5-1]), and a conjugate beta prior distribution is used to obtain a beta posterior distribution.

Let Θ be the random variable representing an analyst's knowledge of the unknown probability θ before testing. The prior distribution of Θ is assumed to follow a Beta(a,b) distribution. Thus, the pdf of Θ is

$$f(\theta) = \frac{\theta^{a-1}(1-\theta)^{b-1}}{B(a,b)} \tag{5-5}$$

where $0 \le \theta \le 1$, $a > 0$, $b > 0$, and the normalizing constant $B(a,b)$ is the complete beta function. The expected value of Θ is $a/(a+b)$.

Several authors discussed the choice of the prior distribution and the parameters characterizing it in textbooks on Bayesian statistical inference. The handbook of parameter estimation,

NUREG/CR-6823 [Atwood 2002], offers general guidance on selecting prior distributions. Some authors broached this subject within the context of the Bayesian estimation of the failure probability of a program; for example, Haapanen, et al. [2000] and Miller, et al. [1992]. In particular, Miller, et al. suggested using the results of reliability growth methods to estimate a prior distribution via the moment matching method. In addition, quality of software lifecycle activities can generate a subjective prior distribution, as was done in some BBN studies, for example, Gran [2002a] and Helminen [2003a].

In Bayesian terminology, $f(\theta)$ is the prior pdf of Θ, and $g(x|\theta)$ (given by Equation [5-1]) is the likelihood function of X conditioned on the value of Θ. The posterior pdf of Θ conditioned on the observed (after testing) value of X is denoted by $f(\theta|x)$. According to Bayes' theorem, the posterior pdf of Θ, given the observed value x, is:

$$f(\theta \mid x) = \frac{g(x \mid \theta) f(\theta)}{\int_0^1 g(x \mid \theta) f(\theta) d\theta} \tag{5-6}$$

Accordingly,

$$f(\theta \mid x) = \frac{\theta^{x+a-1}(1-\theta)^{n-x+b-1}}{B(x+a, n-x+b)} \tag{5-7}$$

where $x = 0, 1, \ldots, n$, and $0 \le \theta \le 1$.

In other words, the posterior (after testing) distribution of Θ is Beta($x+a, n-x+b$), where x is the number of failures observed in n tests, and a and b are the parameters of the prior Θ distribution. The posterior distribution has a mean of

$$\frac{(a+x)}{(a+b+n)}. \tag{5-8}$$

The Bayesian approach also can generate an upper bound of θ, θ_u. To do so, a confidence level γ is specified that implicitly defines the upper bound of θ_u, such that

$$\Pr\{\Theta \le \theta_u \mid x\} = \gamma \tag{5-9}$$

Solving this equation for θ_u determines an interval $0 \le \Theta \le \theta_u$ in which Θ lies with confidence γ. For example, if $\gamma = 0.95$, an analyst is 95% confident that the value of Θ is in the interval $0 \le \Theta \le \theta_u$.

Simplifying the mathematics by assuming that $a = 1$ and $x = 0$, the posterior cumulative distribution function (cdf) is expressed as:

$$F(\theta_u \mid x) = \Pr\{\Theta \le \theta_u \mid x\} = \int_0^{\theta_u} f(\theta \mid x) d\theta \tag{5-10}$$

which, reduces to

$$F(\theta_u \mid 0) = 1-(1-\theta_u)^{n+b} = \gamma \tag{5-11}$$

Solving this equation for θ_u

$$\theta_u = 1-(1-\gamma)^{1/(n+b)} \tag{5-12}$$

The number of successful tests required to show that the failure probability is bounded by θ_u at confidence level γ is obtained from Equation (5-12) as:

$$n = \frac{\ln(1-\gamma)}{\ln(1-\theta_u)} - b \tag{5-13}$$

With regard to the confidence interval, note that Equation (5-13) differs from Equation (5-4) only in the term of *b*, which is equal to 1 if a uniformed prior distribution is selected. With regard to the mean value of the posterior distribution given in Equation (5-8), for a small value of *b*, the number of tests without failure needed to achieve a mean value of 10^{-5} is 10^5.

5.3 White-Box Methods

The white box (or gray box) test methods take into consideration the internal structure of the software and have the advantage that they provide information on internal components of the software and may ensure more internal components be covered by tests. The associated QSRMs have to account for the internal structure accordingly. The following two studies are applications of the white-box methods to a reactor protection system.

May, et al. [1995] described gray box testing as a partition-based method developed for assessing the reactor protection system of a nuclear power plant. It partitions software into subsets, each represented by a partitioning factor α_i, that is, the proportion of the code covered by the partition; and performs tests on individual subsets. For each partition, a simulation model of the operating environment was used in generating test sets allowing random variations in the software input space. Using a Bayesian approach, May, et al., showed that the mean failure probability of the software is approximately $\frac{1}{\Sigma(n_i \alpha_i)}$, where n_i is the number of successful tests of subset *i*.

The gray box reliability method of Zhang [2004] uses a Monte Carlo method to sample from the software's operational profile and counts the number of times a node of the software system is visited. The method does not require visiting every path of the system, but every node must be visited. If a node has not been visited, a manually identified set of inputs is used to make sure the node is visited. Bayesian analysis is used to estimate the probability that the system would fail given that a node is visited. The overall probability of system failure is calculated in terms of the failure probabilities and visit frequencies of the nodes. In an enhanced model, Zhang [2004] gives the system failure probability in terms of the probabilities of the visited and unvisited paths of the system.

In addition, the defect-density based method described in Section 6.2 can be considered a white-box method, even though a finite-state machine model of the software was used instead of the software itself.

5.4 Review of Test-Based Methods

This section discusses the use of findings from software tests for estimating the probability of software failure. The following discussion benefitted from a workshop on software reliability held at Brookhaven National Laboratory in May 2009 [Chu 2009]. In general, the discussion applies to any QSRM, including SRGMs, employing test results in assessing software failure rates and probabilities.

1. Testing hardware and software is different. Hardware usually is evaluated by repeating similar tests many times, and counting the number of failures over a number of demands, or a specified period. In contrast, applying the same test to software invariably gives the same result, either success or failure. To address this issue, the selection of test cases should be based on the software's operational profile, which reflects the software's input space. In order to cover the input space as completely as possible, the test cases should all be selected to be different from each other.[12]

   On the other hand, the operational profile may not be well defined/known, and the profile from which the tests are generated may not be the same as the actual operational profile. Hence, the testing environment may not represent the actual "operational profile" to which the software is exposed during operation in an NPP. Littlewood [1991] stated that, "most software testing is unlike operational use, and any reliability predictions based on this kind of classical testing will not give an accurate picture of operational reliability." For this reason, a review panel for the Nuclear Installations Inspectorate of England concluded that the test results of the plant-protection system of Sizewell B could not be used to determine the probability of failure on demand of the system [May 1995]. This problem is widely recognized, as discussed by Bastani and Pasquini [1993] and Miller, et al. [1992]. In general, there is epistemic uncertainty about such aspects as representativeness of the test environment and the accuracy of the test oracle. Nevertheless, the software's testing profile should approximate, as closely as possible, the software's operational profile.

2. If a failure occurs during a test, and the associated software fault is identified and corrected, and the failure is no longer applicable to the corrected software. However, since the software was modified, it may be inappropriate to use directly the test results from older versions of the current software in estimating software failure probabilities. Furthermore, revising the software may have introduced new faults.

   A basic way to resolve this issue is to consider the revised software as a *new* software, as suggested by Parnas, et al. [1990]; Haapanen, et al. [2000]; and Littlewood [Chu 2009]. Therefore, any tests performed with the older version are ignored, and the revised software will be tested from the beginning again. Hence, evidence obtained from testing before

[12] It should be noted that some systems (e.g., multi-threaded systems or event-driven systems) may not exhibit the same behavior each time the same test is run. Also, initialization errors and memory (aging) related errors may cause anomalous test results. The definition of software failure in [Chu 2009b] also implies that the occurrence of a software failure depends not only on the input state (triggering event), but also on the internal state of the digital system. This indicates that, in addition to selecting test cases to represent different input states, test cases may also need to be selected to characterize this type of non-deterministic behavior, if appropriate and feasible.

modifying the software (to resolve the detected failure) is disregarded. Since a safety-critical system may be required to have a very low probability of failure, such an approach may be essential for this kind of system.

3. Errors in requirements and specifications of software have caused many software failures [ASCA 2007, Lutz 2004]. Testing may not uncover incorrect requirement specifications of a program, since this is not the goal of software testing.[13] However, since testing typically is planned and carried out carefully, some errors in the specifications may be discovered by chance [Lutz 2004]. For example, if the requirement specifications are incomplete, such that a function that the resulting software must perform is missing, then the "fault" in this example is that the software lacks the function. Its absence may be manifest under testing if the function is required during a test case, and the software's result is wrong. Kang, et al. [2009] suggested that it might be possible to exhaustively test the software of a reactor protection system, which, if true, potentially would demonstrate that the software does not fail. However, because exhaustive testing can only verify that the requirement specifications are satisfied, the problem of incorrect requirements would remain unresolved. Failure to account for requirement and specification errors is a limitation common among many of the QSRMs reviewed.

4. Different types of tests have different capabilities for identifying different types of faults. Therefore, these differences should be considered when using the tests to statistically estimate software failure rates and probabilities. For example, Ciupa, et al. [2008] compared the results of identifying faults three different ways, that is, through manual testing, random testing, and user reports that reflect actual operating experience. They compared the distributions of the faults, identified using the three ways of identifying faults, over the categories/sub-categories of faults (e.g., specification faults and implementation faults). They found significantly different distributions from the three distinct ways of identifying faults. Frank, et al. [1997] pointed out that (1) if test cases are selected by sampling according to the operational profile, then direct estimates of the failure probability may be obtained, and (2) if a test case is selected in any other way, then the probability of encountering a failure bears no necessary relationship to the failure probability in operational use. Popov and Littlewood [2004] addressed this aspect of testing considering, "Test suites are drawn in accord with the testing goal. If operational reliability is targeted the test suites are generated using the expected operational profile … of software. If debugging is targeted the test suite is generated according to what the debugger believes maximizes the chances of finding faults…" They define a test suite as "…a sequence of demands on which software is executed." Since the objective in a PRA is to assess reliability (rather than identifying different types of faults), tests incorporating the expected operational profile appear to be the most appropriate. However, the difference between the expected operational profile and actual operational profile may impose limitations on the accuracy of the test-based methods and any other QSRMs that make use of test data.

5. Assuming that test results are suitable, very many tests must be carried out to gain statistical confidence in a probabilistic parameter with a low value. For example, per Equation (5-8), to demonstrate a mean software failure probability on demand of 10^{-5} requires 10^5 successful tests to be conducted. To demonstrate this same failure probability

[13] Methods other than testing can potentially find requirement defects, for example, inspections and model checking.

with a confidence level of 0.95 (i.e., 95% confidence), per Equation (5-13), 3 x 10^5 successful tests must be conducted. In both of these cases, there can be no observed failures during the testing. If a failure occurs during testing, and the fault causing it is fixed, then testing must be repeated with the updated software.

This issue is particularly acute for safety-critical software because it usually demands a very low probability of failure. Butler and Finelli [1993] discussed this problem in detail. However, with increasing computing power and advances in testing techniques, conducting a large number of test cases may be achievable, as discussed in Section 7.2 of [Lyu 1996] and in [Cukic 1998]. There are many publications on accelerated testing, for example, [Cukic 1998] and [Bastani 1993]. Their applicability to safety critical systems remains to be investigated.

6. The White-box methods have the advantage that they take into consideration the internal structures, such as paths and nodes of software, so that tests can possibly be performed to ensure that certain parts of the software will be tested. While these methods provide information about the internal structure of the software and may detect faults that would otherwise be undetected, they require additional resources, such as those needed to estimate the coverage of different structural elements (e.g., estimating the frequency at which a path or node is visited). From the perspective of estimating software failure probability, it is not obvious that they would provide better statistical results as compared with black-box test-based methods, assuming that the same number of tests is performed in both cases. For example, the white-box method of [May 1995] provided an example study indicating that given the same number of tests, the white-box method provides the same results as the black-box method only if the partitioning factors, α_i's, are equal to 100%, and providing worse results (i.e., higher failure probabilities) if the factors are less than 100%. This can be observed by examining the expression for the mean failure probability specified in Section 5.3, in which the number of tests performed for each partition is multiplied by the partitioning factor, making the tests not as effective in achieving a lower failure probability. This observation appears counterintuitive, because the white-box method uses more information about the software and should provide better results. A possible explanation is that the mathematical model does not fully capture the benefits of white-box testing.

6. OTHER QSRMs

The methods considered in this section represent those methods that do not belong to the major categories of methods discussed in the three preceding sections. These methods include (a) a correlation approach that estimates software failure rate at the end of the software testing stage by making use of the software engineering practices of past software development projects, and implemented in a software tool called "Frestimate"; (b) metrics-based methods that estimate software failure rates and probabilities by correlating software engineering measures/metrics and software reliability; and (c) the context-based software risk model that combines traditional probabilistic risk assessment approaches (e.g., event trees) with an advanced (dynamic) modeling approach to integrate the contributions of digital hardware and software into a model of overall system risk. In addition, a few studies that considered rule/standard-based methods and software diversities were reviewed.

6.1 A Correlation Method Using Software-Development Practices

6.1.1 Introduction

Neufelder [2000a, 2000b, 2002] developed approaches for estimating the probabilistic parameters of software, such as its failure rate, and then implemented them in a computer code called Frestimate [SoftRel 2009b]. The review which follows is based on gathering documents on these approaches and this code from the public literature and SoftRel's website [SoftRel 2009b]. The focus of this review is on those features of the computer code related to assessing the probabilistic parameters of a particular software. Frestimate calculates the probabilistic parameters of the software being evaluated, such as the failure rate and the mean time to failure (MTTF), during three "phases" of the life of the software:

1. Early development. This capability usually is employed before the software is developed, or during its development, before starting testing. The main type of data used in this phase is the software-development practices.

2. Systems testing. This capability typically is employed when testing the software. Estimates of the probabilistic parameters of the tested software are obtained during or at the end of testing, and are based on testing failure data.

3. Operational measurement. After deploying the software for operation in the field, operational failure data are gathered and entered into the code to assess the associated failure rate and MTTF.

Frestimate's users determine which phase the software is in, collect the data for that phase, and enter them into Frestimate. Only the first function, "Early Development," is reviewed here; in many cases, when people refer to Frestimate, they mean this function. When a user employs the second function, "Systems Testing," Frestimate applies some software reliability growth models to estimate parameters, such as failure rate. Software reliability growth models are detailed in Section 3 of this report. The third function, "Operational Measurement," seemingly uses simple formulas for assessing failure rate and mean time to failure (MTTF). Neufelder [2009a] proposes that the actual failure rate and MTTF are simply functions of how many units have been deployed, the average usage time of each unit, and the number of failures reported to date on those units." She gives the numerical results of the following example: If 2 units

(with the same software) are deployed, the average "duty cycle" for each unit is 730 hours, and the total number of software failures, including multiple instances of the same defect, since deployment is 17, then the MTTF = 85.9 hours. Apparently, the MTTF was calculated as (2*730)/17. When the MTTF is determined in this way, that is, accounting for every instance of every defect, and for all duty on all deployed units, Neufelder calls it an "external MTTF." This approach is not discussed here due to its simplicity.

6.1.2 Description of the Predictive Model

Neufelder [2002] states that "A predictive model is used when the software has not been developed or tested yet. It is based on empirical data…As with any predictive model, one needs to have some trained data. In this case, that includes observed software defect counts and the practices employed during the development of that software. The author has mined data from 48 organizations. Of those, the actual fielded defects were known for 22 organizations. This set of data constitutes the 'trained' data for which a predictive model was developed. The software-development practices of the organizations were correlated to the observed defects." The development of the predictive model (i.e., the "Early Development" function in Frestimate) is described in [Neufelder 2000a], [Neufelder 2000b], and [Neufelder 2002]. The approach, which was derived from a Rome Air Development Center (RADC) method [McCall 1992], essentially correlates the failure rate of a software project and software-development practices, using a database of software-development practices and past software projects, including fault-detection experiences.

Frestimate predicts (latent) "defect density" (DD), namely, a measure of the number of defects per thousand lines of code of the software, as an intermediate step to assess failure rate. A (latent) defect is the same as a fault; this review uses the terminology employed in Neufelder's documents. Frestimate uses DD to obtain the total number of defects of any type, N_0, expected at the end of testing, that is, on the delivery date.

Neufelder's approach [2000a, 2000b, 2002] for estimating the failure rate of software consists of two main steps, described below:

1. Develop a "predictive model" of DD based on empirical data.

2. Obtain DD using this model, and transform it into the failure rate of the software.

Develop a "predictive model," based on empirical data

Figure 6-1 is an overview of the process for developing the predictive model. Assigning a consecutive number to each box in the figure in the direction of the arrows (i.e., from the top and from the left), each step is briefly explained from the information given by Neufelder [2000a, 2000b, 2002].

1. Determine the universe of software-development practices and create a survey. Select those software-development practices (e.g., consistent and documented formal and informal reviews of the software and system requirements prior to design and code, the language and operating system is well supported by industry, and existence and use of test beds) that are likely to be correlated to defect density for any organization. McCall, et al.'s work [1992] was a starting point for selecting these parameters.

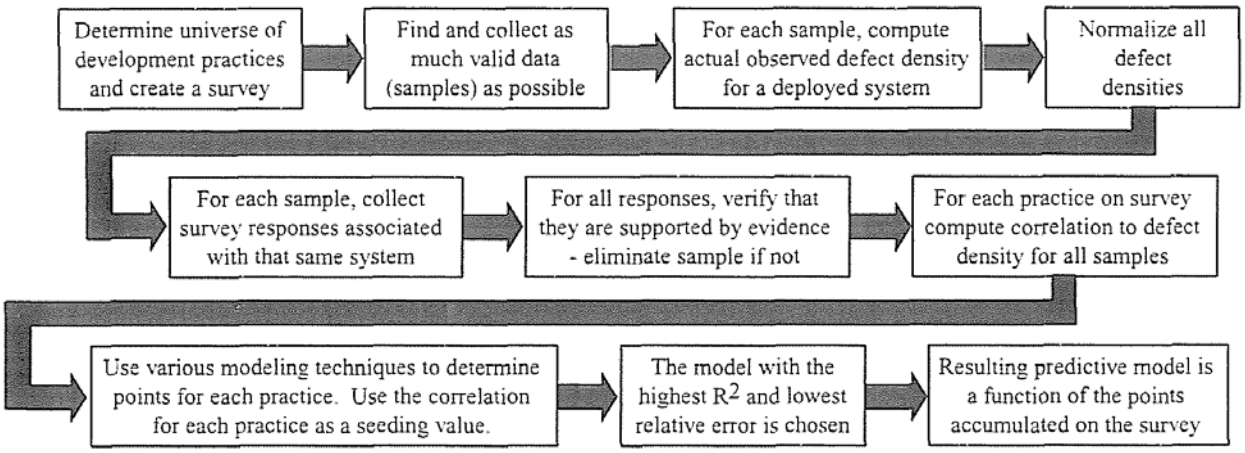


Figure 6-1 Development of predictive model [Neufelder 2002]

2. Find and collect as much valid data (samples) as possible. Neufelder [2006] states that she benchmarked the software-development practices at more than 90 organizations. Accordingly, the database consists of information collected from them.[14] Her breakdown of the 90 organizations was semiconductor fabrication 57%, defense/aerospace/space 29%, firmware 7%, medical 4%, energy 1%, telecomm 1%, and, commercial software 1%.

3. For each sample, compute actual observed defect density for a deployed system. This was accomplished by 1) plotting, in a scatter plot of data collected at different times, failure intensity (cumulative failures per cumulative time) on the x-axis, and cumulative failures on the y-axis; 2) using a Least Squares Estimate to find the best straight line through these points; and, 3) determining the y intercept and setting it to the theoretical number of inherent defects which is the numerator in defect density. This number is theoretical because it is impossible to be completely certain when no defects remain in the software.

4. Normalize all defect densities. The number of thousands of source lines of code (KSLOC) of the software for each sample were counted. Since each software may be in a different programming language, the KSLOC for each software was converted into assembler KSLOC. The defect density for each software was normalized by dividing the inherent number of defects identified in the previous step by its assembler KSLOC. Defect density was used, rather than absolute value, to support comparisons of defect densities of programs developed in different languages.

5. For each sample, collect survey responses associated with that same system. Neufelder collected responses to the software-development practices as written in the

[14] Apparently, only data from a fraction of the organizations was used in building the model. For example, Neufelder [2002] stated that actual fielded defects were known for only 22 out of the 48 organizations for which data was mined.

surveys mentioned in step 1. She states [2000a] that she "… evaluated the practices for each of the organizations using the same criteria," and that she "…was intimately familiar with each organization and their 'actual' versus 'wish list' practices." To avoid documentation of overly optimistic or pessimistic responses, she "…required inputs from a wide cross section of employees such as managers, lead engineers, quality engineers, test engineers, engineers with an average level of experience and new hires…"

6. For all responses, verify that they are supported by evidence – eliminate sample if not. Neufelder required each organization to provide physical proof of all questions answered positively.

7. For each software-development practice on the survey, compute correlation to defect density for all samples. The response to a practice (also called parameter) was given 1 for yes, 0.5 for yes but not consistently, and 0 for never. Regression analyses were applied to demonstrate a correlation between each practice and the defect density obtained previously. A negative correlation is expected when correlating practices to defect density. Correlations of -1 and +1, respectively, signify that the practice perfectly correlates to lower- and higher-defect density. A correlation of 0 means no correlation at all; such practices are dropped.

8. Use various modeling techniques to determine points for each software-development practice. Estimate each practice's points (also called weights) by using the relative correlation of the practice (from step 7) as a seeding value, and then "…maximizing the correlation between the score that results from the sum of the weighted yes answers and the actual defect densities observed" [Neufelder 2000b].

9. Choose the model with the highest R^2 and lowest relative error. Neufelder does not define the term R^2, but, seemingly, it is the *coefficient of determination* of the regression. A polynomial regression model was selected as the best model for expressing the relationship between empirical scores and empirical defect density.

10. The resulting predictive model is a function of the points accumulated on the survey. In other words, the resulting polynomial (model) is a function of the total score from a completed survey, represented by the variable X. The actual polynomial varies in Neufelder's different publications, presumably because it changes slightly as new samples are incorporated into the model. The polynomial from her most recent publication [Neufelder 2002] is

$$Predicted\ DD = 1.7x10^{-7}X^2 - 1.00439x10^{-3}X + 1.58463875 \qquad (6\text{-}1)$$

The units of DD from this formula are the number of defects in a thousand source lines of assembler code, and it is the expected DD when the developers deliver the software to be used in the field. Frestimate includes the resulting predictive model (polynomial). Note, it is unclear that the raw data and process used to arrive at the predictive model (i.e., the polynomial in Equation (6-1) justify the number of significant digits of the coefficients in this equation.

Obtain DD using this model, and transform it into the failure rate of the software

The specific software-development practices of an organization that is developing or developed a software can be used to estimate the software's failure rate via the predictive model. Neufelder proposes using the steps in Figure 6-2 to do so. Again, assigning a consecutive number to each box in the figure in the direction of the arrows (i.e., from top and from the left), each step is briefly explained from the information provided in Neufelder [2000a, 2000b, 2002].

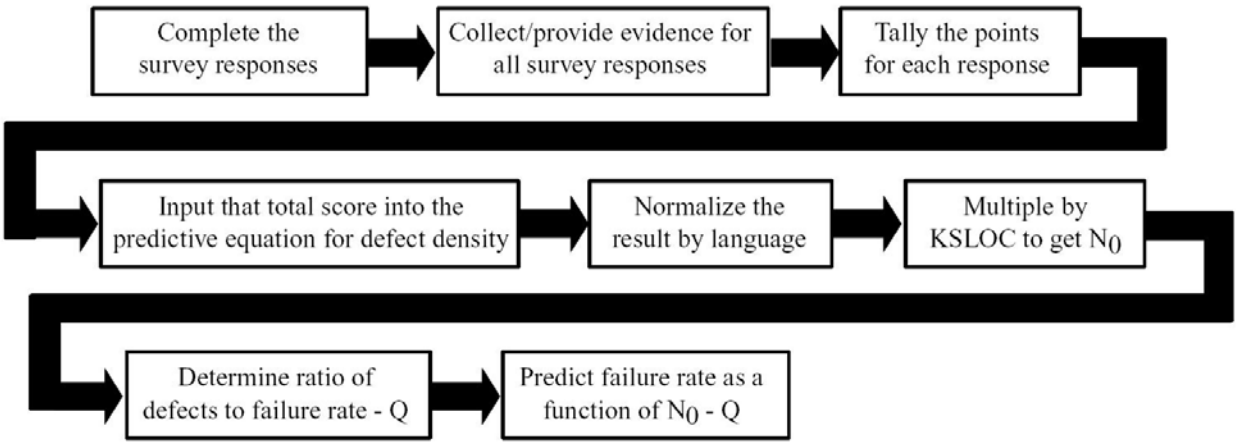


Figure 6-2 Using the predictive model to estimate the failure rate [Neufelder 2002]

1. Complete the survey responses. Frestimate offers the following four types of surveys (implemented in its modules): Basic (1 question); Shortcut (15 questions); Rome Labs (7 "factors," such as a "Development" factor that incorporates many development practices planned or used for the particular software project); and, Full-scale modules (117 questions). A user selects one of the four modules and completes a survey of the software-development practices used by a particular organization for generating software. The survey depends on the user's choice of module; for example, the surveys of the Basic and Full-scale modules have 1 and 117 questions, respectively. Neufelder states that the accuracy of the results generally increases with more detailed modules.

2. Collect/provide supporting evidence for all survey responses.

3. Tally the points for each response. The points for each software-development practice were obtained during the development of the predictive model (Step 8, above). For each practice, some points are assigned to a "yes" answer, and, frequently, a different number of points is given to a "sometimes" response. Based on the analyst's responses to the questions, a point value is assigned to the corresponding practice.

4. Enter the total score into the predictive equation for defect density. The number of points for all software-development practices are summed to obtain the value of the variable X, which then is used in Equation (6-1) to obtain the DD.

5. Normalize the result by language. If the software being evaluated employs a programming language other than assembler, then the DD in terms of assembler code obtained in the previous step must be converted to DD in the other language. Accordingly, the value of the former DD must be multiplied by the "code expansion ratio" of the other language. Neufelder [2000b] cites Table 7-9 of Lakey and Neufelder [1997] as a source of the ratios for several languages. She further notes, "The code expansion ratios were developed by Capers Jones of the Software Productivity Research, Inc. Table 7-9 is a short summary of a comprehensive listing."

6. Multiply the DD by KSLOC to get N_0. The DD from the previous step, expressed in terms of the software's language then is multiplied by the software's size (in thousands of source lines of code [KSLOC]) to estimate the inherent number of defects in the software, N_0.

7. Determine ratio of defects to failure rate – Q. Neufelder [2002] indicates that the parameter Q is "…the ratio … between inherent defects and failures per time based on historical data." Frestimate offers three ways to assess Q:

   a. Using Frestimate's built-in default values. Apparently, these values come from [McCall, et al. 1992] because Neufelder [2000b] notes "If no historical data is available, [McCall, et al. 1992] provides a lookup chart of average values for Q…"

   b. Calculating the parameter using all fielded software defects for a similar software previously deployed.

   c. Computing the parameter by employing observed MTTF values for a similar project previously deployed.

8. Predict the failure rate as a function of N_0 and Q using a simple exponential formula:

$$\lambda(t) = N_0 * \exp(-Q*t/N_0) / t \qquad (6\text{-}2)$$

   where t is the time of interest. No formal derivation of Equation (6-2) was found.

   In the function "Early development," Frestimate calculates a "nominal" value and lower- and upper-bounds of the failure rate, based on a user-specified confidence interval.

6.1.3 Applications of the Predictive Model

The only application of the predictive model for assessing the probability of failure or failure rate of a software program that was found was to NASA's Space Transportation System (STS) flight software (FSW).[15] Unfortunately, the details of that study are not publicly available due to confidential and/or proprietary concerns, so a review of the study was not performed as part of this project. In the STS FSW study, the predictive model was applied for estimating the

[15] Thompson, N., "Using Frestimate for a Phase-Specific PRA of STS Flight SW," NASA - NRC Technical Interchange Meeting on Software/Digital Instrumentation and Control System Reliability Analysis, March 4, 2009, Rockville, MD.

probability of "critical" failure, which was considered to be the loss of the crew (LOC). This probability was estimated as follows:

$$P = 1 - \exp(-\lambda T)$$

where λ is the failure rate generated by Frestimate, and T is the mission time of the software.

6.1.4 Review of the Correlation Method

The general concept of performing correlation/regression analyses using past software development experience is reasonable. However, because of the unavailability of detailed information on the raw samples and the regression analyses used to construct the predictive model (i.e., the "Early Development" function in Frestimate), this methodology cannot be evaluated in detail, and may not be appropriate for use where transparency and understanding of both data and modeling assumptions are required to permit sufficient levels of peer review and quality assurance. The Frestimate method is a failure-rate based method which is not suitable for modeling demand failures of protection systems, and may only be useful in evaluating the frequency of spurious actuation of protection systems. The potential limitations of this code for NRC's purposes are discussed next, based on current information.

1. Neufelder examined the failure data of many software projects, and used these empirical data for building the predictive model. Her basic assumption was that she could get information about all the faults in the software projects. However, the relationship between faults and failures is complex because of the unpredictability of the number of faults in the software, and the samples used from the software's input space. For example, Kanoun and Laprie (in Lyu [1996]) point out that the data in Adams [1984] from nine large software products revealed the activation of only 5% of the faults in a program with a mean lifetime of 15 years. Hence, it is unclear that the empirical data underlying Neufelder's predictive model encompass all the faults of every software project that she reviewed. Thus, it may be incorrect to assume that she obtained information about all the faults in the software projects. Accordingly, the predictive model might under count the number of faults in a particular software, thus underestimating the associated parameters, such as the failure rate.

2. Neufelder's predictive model is based on failure data from past software projects. If many or most of these past projects involved software for normally operating control systems, the associated data used to obtain estimates of defect density may not be applicable to the types of software used in NPP protection systems.

3. A user must complete a subjective survey of software-development practices. Different analysts, having different perceptions of the ways the organization developing the software implements these practices, may generate different parameters.

4. Frestimate assumes that the operational period is a continuation of the testing period because it considers that failures will happen during the operational period, defects will be discovered and removed, the failure rate will decline, and reliability will increase. In practice, this assumption may be unrealistic because new defects can be, and have been, introduced into software when removing known ones. Hence, the failure rate may not decrease.

5. Applying Equation (6-2) requires estimating the parameter "Q." Assessing it via one of the three approaches described above seems to be a subjective process involving large uncertainties, because none of these approaches use information related to the specific software being assessed.

6. Apparently, similar to most QSRMs, Frestimate does not specifically account for the context in which software operates, does not consider specific software failure modes, and cannot be used for estimating probabilistic parameters of software when the expected values of the parameters are small. Lakey and Neufelder [1997] recognize this latter limitation when they discuss "Ultra High Reliability Prediction" as follows: "It is essential to consider achievability and testability when predicting reliability for software systems that must be relatively high. Demands for perfection should be avoided as they are not testable or demonstrable. For example, if the demand for the failure rate is 10^{-4} then there must be sufficient resources for extensive validation and verification to demonstrate this level. The current state of the art is limited in providing any help in assessing the software reliability at this level…"

7. It is not known whether Frestimate was validated or benchmarked by organizations independent from the organization (SoftRel, LLC) that developed it.

6.2 Metrics Methods

6.2.1 Description of Metrics Methods

The metrics methods, developed by Smidts and Li at the University of Maryland (UMD) [2000, 2004], estimate software failure rates and probabilities by correlating software engineering measures (SEMs)/metrics and software reliability. Development includes (1) using expert opinion to identify and rank SEMs important to software reliability, and (2) developing methods for calculating software reliability using system specific SEM information. As a test case, the methods were applied to a personnel access control system (PACS) and the findings compared with test results generated by sampling inputs from an operational profile representing the actual operating condition of the system. The development of the methods is further described below.

1. As detailed in NUREG/GR-0019, Smidts and Li [2000] screened by order of importance 78 SEMs earlier identified and ranked at the Lawrence Livermore National Laboratory (LLNL) [Lawrence 1998]; they reduced them to 30 measures constituting the basis of the UMD study. That study introduced an important concept, namely, that a "reliability prediction system" (RePS), that is, a method/system for estimating software reliability, is a *complete* set of SEMs from which software reliability can be predicted. The criteria for ranking the SEMs were established by revising those of the LLNL study; the criteria constituted an important part of a questionnaire sent to a panel of experts who ranked the SEMs. These experts from the nuclear and aerospace industries covered the following areas: Software development, software engineering, software reliability, software safety, and digital instrumentation and control system design. Later, a workshop was held in which the experts summarized their evaluations, and offered feedback on the ranking method. The data from this elicitation, including 10 missing measures identified, were aggregated and used in ranking a total of 40 measures.

2. In NUREG/CR-6848, Smidts and Li [2004] detail their selection of six "root"[16] SEMs as the measures used to develop methods for quantifying probability of failure on demand. The objectives are to validate the RePSs in terms of their accuracy, and to validate the ranking of SEMs in [NUREG/GR-0019]. The criteria used in selecting the SEMs were: (a) ranking level determined in NUREG/GR-0019, (b) ease of data collection which calls for object-oriented measures identified in NUREG/GR-0019, (c) relevance to reliability determined in NUREG/GR-0019, and (d) coverages of different SEM families defined in NUREG/GR-0019. The selected root SEMs are:

   - mean time to failure,
   - defect density,
   - test coverage,
   - bugs per line of code,
   - function points (a measure of the system's functional size), and
   - requirement traceability.

   For each measure, typically supplemented by other support measures, a method for quantifying software failure probability was developed using available methods, concepts, and empirical formulas. The RePS based on the SEM ranked highest by the experts (i.e., defect density) is briefly described below and reviewed in Section 6.2.2.

   For the SEM of defect density, defined as the number of defects per thousand lines of code, the estimation of the probability of system failure-on-demand was derived from a concept formulated by Voas [1992] on estimating testability of software. The system's failure probability is calculated as the sum of the probabilities of execution of all those input/output paths with identified but unresolved defects in a finite-state machine model of the system. The probability that such a path is executed is calculated as the product of the probabilities of the transitions in the input/output path. To carry out the calculations, the method requires that a finite-state machine representing the users' requirements and the associated operational profile be developed, and some defects of the system be identified during inspection of the code. This method can be considered a white-box testing method of the type that is discussed in Section 5.3, which takes into consideration the internal structure of the software.

3. In NUREG/CR-6864, Smidts and Li [2004] discuss six different methods used in estimating system failure probability, one for each SEM. These methods were applied to the PACS and the results compared with the "real" failure probability estimated using test results of 42 failures in 499 trials. The defect density SEM produced the best result.

6.2.2 Review of Metrics Methods

The metrics methods were applied to the PACS and some of the methods were demonstrated to provide reasonably good estimates of the actual system failure probability. However, the example system is a relatively simple system that does not have the very high reliability that is

[16] As stated in NUREG/CR-6848, "the measure on which RePS construction is based is termed the 'root' of the RePS. Other measures within the RePS are defined as 'support' measures."

expected of a safety related system at a nuclear power plant. The proposed approach appears to suggest that one RePS be developed for each of the 40 SEMs to determine the best method(s). It is desirable that the expert opinion in ranking the SEMs be used to reduce the number of RePSs to be considered, as suggested by one of the peer reviewer comments included in NUREG/CR-6848.

The metrics methods were developed by applying available methods, concepts, and empirical formulas and do not represent new innovative methods. Some of the methods use engineering insights (as opposed to application-specific information) to develop empirical formulas representing the relationship between software reliability, that is, failure rate and probability, and software engineering measures. The empirical formulas are not laws of software reliability; hence, their general applicability and accuracy are limited. An exception is the previously described method associated with defect density.

As suggested in Smidts and Li [2000], a method (system) for estimating software reliability (i.e., an RePS,) should be a *complete* set of SEMs from which software reliability can be predicted. Each of the 6 methods described in NUREG/CR-6848 is capable of producing an estimated failure probability using supporting measures that are not among the 40 root SEMs identified by the experts, and thus consists of a complete set of SEMs according to the definition of NUREG/GR-0019. The root SEMs were applied in an orthogonal, independent manner and the available documentation does not indicate that any work has been done on development of metrics methods that combine some or all of the highly ranked SEMs.

For the example method based on defect density, two important assumptions were implicit:

1. The identified and unresolved defects are the only defects remaining, and
2. The finite-state machine model of the system is a realistic representation of the actual software in terms of the likelihood that different branches of different paths are actually executed.

These assumptions may not be valid. In the case of the first assumption, there may be unidentified faults which the method does not account for. NUREG/CR-6848 proposes to use Capture/Recapture models to estimate the unidentified defects. It is not clear if this approach has ever been validated. In the case of the second assumption, the probability of an execution path of the finite-state machine may not truly represent that of the actual system even if the operational profile were known perfectly. In addition to concern over these two assumptions, it is also not clear from Section 5.2.4 of NUREG/CR-6848 how the defect density RePS actually relates to the calculated defect density.

NUREG/CR-6848 considers that the results of the study validated the overall approach by showing that highly ranked SEMs produce results that are closer to the true answer. However, such a conclusion depends on the quantification methods developed and associated with the SEMs. Alternative quantification methods that may produce very different results can be developed and associated with the SEMs (as indicated in NUREG/GR-0019) and potentially lead to a different conclusion. For example, it would be equally valid if one associates a software reliability growth model (SRGM) to the defect density SEM, because the SRGM does address defects, and this might lead to very different results.

6.3 The Context-based Software Risk Model (CSRM) Method

6.3.1 Introduction

The Context-based Software Risk Model (CSRM) method, as described in [ASCA 2007], combines traditional probabilistic risk assessment (PRA) approaches (e.g., event trees) with an advanced modeling approach (e.g., the Dynamic Flowgraph Methodology [DFM]), to integrate the contributions of digital hardware and software into a model of overall system risk. To a large extent, the CSRM method apparently is an implementation and extension of the approach proposed by Garrett and Apostolakis [1999] using DFM. It should be noted that, as indicated in [ASCA 2009], the context-based, risk-informed testing associated with this approach is not meant to be applied to scenarios that occur under nominal system conditions, since these types of scenarios can be quantified using existing software reliability estimation models. Rather, as stated in [ASCA 2007], CSRM focuses on identification, assessment, and prevention of digital system failures that are due to incorrect or logically incomplete software design and specifications, which are likely to manifest themselves only under off-nominal conditions (e.g., in conjunction with hardware failures or other system conditions that might not have been envisioned when the software requirements were specified).

6.3.2 Description

Due to the novel approach involved, the descriptions of the CSRM method given below are quoted directly from [ASCA 2007] to preserve clarity. This subsection also includes some information from [ASCA 2009] on using CSRM for quantifying a probabilistic parameter (i.e., demand failure probability or failure rate) of software failure. The equations presented were re-numbered for simplicity, and some clarifications are offered in square brackets.

> The basic concept of the Context-based SW [software] Risk Modeling (CSRM) method is that software failures are highly context dependent, therefore, the probability of success or failure of software-driven and software-controlled operations can change drastically when the system of which the software is a part enters a different "context." The boundary conditions and inputs to the software produced by the "balance-of-system" are ultimately what determines a proper or faulty response by the software itself. Unlike the majority of hardware failures that are characterized as being random in time with a certain type of failure rate (typically assumed to actually be constant in time), software failures are driven by a conditional cause-effect mechanism, by which faults in software logic design, or faults introduced accidentally at the time of coding or compilation, become actual failures when a specific logic path of instruction execution is traversed and activated because of a certain input condition – i.e., by a certain specific combination of inputs received from the balance-of-system and / or external environment.
>
> The CSRM method is executed by carrying out the following steps. These steps correspond to the standard execution steps of a typical PRA, as referred to in parentheses.
>
> 1. Identify all mission-critical and safety-critical system functions supported by software (System Familiarization Step of PRA). The objective of this step is to identify all the critical system functions executed directly or supported indirectly by the software. This ensures that the analytical steps implemented later will cover all potential risk

contribution from the software, either by inclusion in the model or exclusion from the model with appropriate justification.

2. Identify all major logic conditions of mission-critical system function execution that may induce or trigger software errors (Model Development Step of PRA). … [The goal of this step is to] identify all major logic conditions of mission-critical system function execution that may induce or trigger software errors. The objective is to decompose the software-related system risk into a disjunction of N context-dependent terms, as expressed by Equations 6.3-1 and 6.3-2 below:

$$SSWR = \bigcup_{i=1}^{N} CSWR_i \qquad (6.3\text{-}1)$$

$$CSWR_i = SC_i \cap (SWR / SC_i) \qquad (6.3\text{-}2)$$

where:

SSWR ≡ overall system-level SW-related risk
$CSWR_i$ ≡ i-th context-related SW risk-contributing item
SC_i ≡ i-th context-forcing system condition
SWR / SC_i ≡ SW response given the i-th context-forcing system condition

…logic PRA models will be used to analyze the context-dependent terms in Equations 6.3-1 and 6.3-2…such as the Dynamic Flowgraph Methodology (DFM).

A DFM model is a graphic network that links key process parameters to represent the cause-and-effect and the time-dependent relationships. In particular, for a digital control system, both the controlled/monitored process and the controlling software itself are represented in the DFM model.

Key controlled/monitored process parameters and software variables that capture the essential behavior of these components and software/firmware functions are identified and represented as process variable nodes. The variable represented by a process variable node is discretized into a number of states. The reason for discretization is to simplify the description of the relations between different variables. The choice of the states for a process variable node is often dictated by the logic of the system. The number of states for each variable must be chosen on the basis of careful consideration to balance the accuracy of the model with the complexity introduced by higher numbers of variable states.

These process variable nodes are then linked together through transfer boxes or transition boxes for instantaneous actions or time-delayed actions, respectively. A decision table is used to represent the relationships between input and output process variable nodes for [either type of] box. This table is a mapping between the combinatorial states of box inputs to the outputs. Decision tables allow each variable to be represented by any number of states.

When traversed inductively, [DFM models] generate scenario representations conceptually similar to ET [event tree] or ESD [Event Sequence Diagram] model

representations, although of course not limited to binary variable and state representations like the latter. When traversed deductively, they produce the multi-state variable equivalent of binary FT [fault tree] models and generate failure "prime implicants," which are the multi-valued logic equivalent of binary logic "cut-sets" (e.g., FT cut-sets).

3. Quantify or bound the probability of erroneous software behavior in response to system conditions identified per Step 2 (Risk Quantification Step of PRA). … [The purpose of this step is] to quantify or "bound" the probability of erroneous software behavior in response to system conditions previously identified. ...The prime implicants obtained in a DFM analysis can be generally classified into 3 types:

    1. Prime implicants that are conjunction of hardware states. This class of prime implicants represents hardware only fault conditions.
    2. Prime implicants that are conjunction of software states. This class of prime implicants represents software only fault conditions.
    3. Prime implicants that are conjunction of hardware states and software states. This class of prime implicants represents software fault conditions that are triggered by some specific hardware conditions.

    The DFM top events are quantified in fashion similar to fault-tree top events. For a particular top event i, the set of n prime implicants … is first converted to a set of m mutually exclusive implicants (MEIs) … [and] the probability of the top event can be calculated as the sum of the probabilities for these mutually exclusive implicants.

    [For the MEIs of the third type, i.e., those that are conjunction of hardware states and software states, the] $CSWR_i$ context-related SW risk-contributing items that appear in Equations 6.3-1 and 6.3-2 can usually be considered as being independent and mutually exclusive, so that in terms of associated probabilities, the quantitative risk formulation below can be applied:

$$SSWR = \sum_{i=1}^{N} [P(SC_i) \times P(SWR/SC_i)] \qquad (6.3\text{-}3)$$

    Using the context decomposition (Equation 6.3-3) as a road map, SW functional analysis can be carried out and the risk scenario modeling results can be used to guide the SW test process, which in turn drives risk scenario quantification. In essence the analysis permits the partitioning of quantification and associated testing between:

    A. A subset of "normal conditions" that have high values for the $P(SC_i)$ terms (order of magnitude 10^{-1} to 1) and thus need to be shown to have low values (typically order of magnitude < 10^{-3}) for the conditional probabilities of erroneous SW response, $P(SWR/SC_i)$.

    B. Subsets of "off-normal" conditions, triggered by balance-of-system failure or anomaly events with relatively low value probabilities $P(SC_i)$ (order of magnitude < 10^{-2}), and thus only need to be shown to have "low-enough" values for the conditional probabilities of erroneous SW response, $P(SWR/SC_i)$ (typically order of magnitude 10^{-2} or lower).

    In general, the recommended quantification of Equation 6.3-3 terms can proceed by choosing for each $CSWR_i$ scenario the best applicable SW reliability estimation model. For example, quantification of "type A" normal condition $CSWR_i$ scenarios, where $P(SC_i)$ values are by definition high and thus $P(SWR/SC_i)$ values should be demonstrated to be very low, one could

use a Bayesian belief network approach that accounts for generic and process-related information… to generate a prior for the $P(SWR/SC_i)$ values. Each such prior could then be "updated" (in the Bayesian statistical estimation sense) via the appropriate type of testing process that would be feasible in terms of time and resources.

The "type B" anomaly or exception driven $CSWR_i$ scenarios, on the other hand, can be successfully quantified, regardless of whether the statistical method preferred is classical or non-informative-prior-Bayesian, with a low number of tests, logically partitioned as discussed earlier to prove $P(SWR/SC_i)$ values to be below an upper bound that doesn't need to be any lower than order-of-10^{-2}.

Since in testing, the SW may not be subjected to the same environment as in the actual system application. If a specific function cannot be tested in its true "mission-configuration," either because it is not well defined, or because it is complex, or because of any other reasons, then the conditional SW reliability estimation may have to be modified with an appropriate adjustment factor, which will usually act in the conservative direction, i.e., pushing towards a lower reliability estimate than what is produced by the form of testing that is possible.

The compilation of a table [containing such adjustment factors or alternative conditional probability estimations] … permits an expert judgment of to what degree a software reliability estimation obtained for a particular software module can be applied to specific functions that are relevant in a conditional probability formulation like Equation 6.3-3. The objective is to judge whether the software reliability model may have been applied to a software module containing the function but exerting it under conditions substantially different from those that may be encountered in the actual mission, or, in cases of more extreme divergence between test and mission conditions, whether the actual function may have not been exerted at all in testing.

The different values of the factor that can be used in such a tabulation should be arrived at via a process of expert elicitation, which should include software design engineers and software test experts as well as PRA experts. This process may be carried out in three steps:

1. Define the table structure, i.e., identify, structure and characterize, in qualitative and/or discrete terms, all the factors that are believed to have determinant influence on the value of the adjustment for the originally obtained direct estimation of the conditional software POF [probability of failure];

2. Define the ranges of magnitude of adjustment of the direct estimation believed appropriate for each combination of qualitative or discrete characterizations of the determining influence factors;

3. Select a specific value, or distribution within the defined range, for the adjustment factor to be applied to the originally estimated conditional probability.

The CSRM method can be used to support quantifying the probability of software failure, but it is not a quantification method or approach for this quantification. This point was clarified in [ASCA 2009] as follows:

A. The CSRM process provides a modeling roadmap by which software events that are important to mission success or failure can be systematically identified and analyzed in PRA-compatible and consistent fashion;

B. CSRM does not mandate or force any single particular way of quantifying the basic software failure modes that may contribute to the mission relevant software-related events referred to in A above.

The above points are made because some misinterpretations have occurred, by which the CSRM process was being looked at not as a PRA modeling process, but as if it provided a specific formula to generate quantitative risk values for software. This is not the case. CSRM is a PRA modeling tool, and, like the PRA event-tree / fault-tree modeling paradigm, does not mandate which specific failure rate databases or estimation formulas are to be used to quantify cut-sets and basic events. Of course this does not make it inappropriate to consider the question of what quantitative estimation processes can be effectively used in conjunction with the CSRM modeling approach…

…The method by which software failure modes and states can be quantified is highly dependent on the state of development of the system and the software itself. In the application of CSRM, or any other PRA process to a mission and system at the design concept or early design phase of development, where little or no system-specific software test results are available, only "surrogate data" or "parametric" estimation methods can be used to quantify the software states, and this includes any data that may have been developed at lower levels of analysis (i.e., subsystem PRA models inclusive of software).

In later stages of system development, the PRA model quantification can be updated with system-specific software test data, after extensive tests of the software have been performed. To best support PRA model quantification, these tests should be "risk-informed," i.e., directed at resolving questions and uncertainties pointed out by the earlier software PRA models and analyses…

To summarize, CSRM quantification can be in principle executed using any of the following approaches, alone or in combination:

Method A: Use of "surrogate data," i.e., software failure and anomaly historical records from missions using software functions of similar nature, to estimate overall software failure probability and relative frequency of basic software failure modes. …This method uses surrogate data to quantify the software failure states in the CSRM logic models. This method is generally applied in the early design phase, where no direct system-specific software test data are available. The surrogate data can be in the form of failure and anomaly history of software used in similar contexts.

Method B: Use of a parametric regression model, such as "FREstimate," to estimate an overall software residual (i.e., projected to post development and testing) failure rate. …This method uses parametric regression models to estimate residual software failure rates. Factors that go into those regression models include the Software Engineering Institute (SEI) Capability Maturity Model (CMM) level of the software development organization, predicted size of the code, and scale of the development and quality assurance/quality control effort...

Method C: Use of expert opinion to sub-allocate an overall software failure probability or failure rate to a sub-set of software failure modes of interest. …This method assumes that the overall software failure probability estimates are derived by some means such as [Methods A and B]. Expert opinion is then utilized to sub-allocate these overall estimates into software failure modes that are represented in the CSRM logic models.

Method D: Use of system-specific software test data to estimate the probability or failure rate of specific software functions and failure modes of interest. …This method…assumes a certain degree of communication and co-operation between the PRA team and the software testing team in identifying, setting up and running the tests to quantify…specific software failure states.

…Methods A, B and C are applicable in early system development and design stages, whereas Method D becomes applicable only after system software has become available and testable, at

least in prototype form. …[O]verall software failure probability derived with either Method A or Method B can be sub-allocated with Method C.

6.3.3 Applications of CSRM

ASCA [2007] describes the CSRM method and demonstrates it by applying it to part of the software of a Miniature Autonomous Extravehicular Robotic Camera (Mini AERCam), a free-flying satellite that affords flexible remote-viewing capabilities to support manned-space missions. As described in [ASCA 2007], the principal steps in the analysis of the Mini AERCam included:

1. Development of a top-level event tree that identifies the key events of the mission phases being modeled
2. Development of DFM models for the pivotal events (top events) in the event tree
3. Deductive analysis of the DFM models to obtain prime implicants (i.e., minimum combinations of states of variables that can cause occurrence of the top event)
4. Conversion of the prime implicants (PIs) to a set of mutually exclusive implicants (MEIs)
5. Quantification of the prime implicants using surrogate (generic) hardware failure data and testing results using a full system simulator package

The DFM approach has been used previously to evaluate the risk associated with software failures in the nuclear and aerospace industries, as demonstrated by Yau, et al. [1995] and Yau, et al. [1998]. As mentioned previously, the prime implicants obtained through deductive DFM analysis of software-based systems can be classified into those that contain (1) only hardware conditions, (2) only software conditions, and (3) combinations of hardware and software conditions. An excerpt from [ASCA 2007] describing how a PI (of the Mini AERCam) consisting of hardware and software conditions (i.e., of Type 3) was quantified is provided below.

> The hardware condition is a small leak in one of the propellant lines. The software condition … causes drifting of the attitude control given a sub-nominal thrust caused by a line leak. If only one of the two conditions exists, the Mini AERCam does not fail... This PI example shows how this type of analysis identifies SW [software] entry conditions for which the SW needs to be tested, which do not correspond to normal states of the system and may not be otherwise identified and tested for.
>
> …[The PI] was quantified by considering the "entry condition" (i.e. small propellant line leak) [represented in the report as condition "C_3"] and the conditional probability that the software causes an attitude shift under this … condition. The former entry condition was quantified with data from a HW [hardware] failure rate database (NPRD [Nonelectronic Parts Reliability Data]). The failure rate associated with the entry condition was determined to be 6.0E-06/hr. Given a mission duration of 5 hours, the probability is:
>
> $P(C_3) = 3.0E-05$.
>
> Testing of the software conditional failure rate using real hardware in a test-as-you-fly configuration is ideal, but was considered unrealistic for this project. Instead, the software conditional failure probability [$P_{F|C3}$] was estimated by testing the attitude control function under the simulated presence of the entry condition using a Virtual System Integration Laboratory (VSIL) simulation of the hardware provided by Triakis Corporation. The VSIL simulation uses hardware documentation to produce a realistic software model of the hardware that can interface with the actual GN&C [Guidance, Navigation and Control] software … To quantify the conditional

software failure contribution [for this PI], an unfaulted baseline mission profile was simulated and it was confirmed that the GN&C software functioned correctly.

…A function was then added to simulate the presence of the gas leak and a series of test runs were performed to determine how the GN&C behaved under the faulted conditions. The function allowed the gas leak to occur at any time during the simulation, in any direction, and with any combination of force and torque.

A combination of intelligent partitioning and randomization was used to ensure that the test cases covered the mission space as completely as possible. Testing showed that each attitude maneuver consisted of 3 phases:

1. Initial movement
2. Oscillation
3. Stabilization

The three phases take a disproportionate amount of time, so having the leak start at a random time during the mission would have tested the stabilization phase much more rigorously than the movement phase, even though the stabilization phase was considered the least likely to manifest a failure during a leak. To prevent this, an equal number of tests were performed for each phase, with the leak starting at a random time during that phase. The direction of each leak was randomized, with pitch, yaw, and roll equally represented. The force of the leak was randomized to be uniformly distributed between 5% and 40% of the Mini AER Cam's thrust in that direction. The upper limit was selected in accordance with the definition of a small leak (0-40% of thrust). The lower limit was selected because leaks with very small force were unlikely to produce results more severe than larger leaks and testing time was limited.

A total of 351 test cases were run. This number was determined by the time limits on the project; more cases would have been desirable. State data was recorded at 1 second intervals. A test case was considered a success if the Mini AER Cam's actual orientation stabilized within the time allotted, and stabilized to within 8º per axis of the commanded orientation. 8º per axis was chosen because this is NASA's [National Aeronautics and Space Administration's] official success criterion for Mini AER Cam attitude accuracy. Other state data was recorded to clarify the decision tables in the GN&C DFM sub-model.

None of the test cases produced an error, so a straight Bayesian estimation was used to establish a preliminary failure probability estimate of:

$P_{F|C3}$ = 2.83E-3

Because the real hardware was not used for the testing, an adjustment to the preliminary $P_{F|C3}$ was made using the Probability Adjustment table in Table 2-1 of [ASCA 2007].

Type of Entry Condition = Exception
Type of SW Function = Well Specified - Complex
Type of Testing = Formal in Simulated System Configuration

After expert elicitation from the VSIL team and examination of the quality of documentation used to create the simulation, an adjustment factor of 15 was chosen to reflect the departure from "test as you fly" principle introduced by using a simulation of the Mini AERCam hardware.

After applying the adjustment:

$P'_{F|C3}$ = 2.83E-3 x 15
$P'_{F|C3}$ = 4.2E-2

Hence, the probability estimate for this mutually exclusive implicant is found to be:

$P(MEI_3) = P(C_3) \times P'_{F|C3}$
$P(MEI_3)$ = 3.0E-5 x 4.2E-2
$P(MEI_3)$ = 1.26E-6

One of the commonly cited limitations with using test-based methods is that they are not practical for demonstrating very low failure rates or demand failure probabilities [Butler 1993]. As stated in [ASCA 2007], the risk-informed and CSRM-based testing shows how a risk level on the order of 10^{-5}/demand for a specific risk scenario and context can be demonstrated with a test effort on the order of a few hundred hours of computer and simulator time.

6.3.4 Review of CSRM

As emphasized in [ASCA 2009], CSRM is not specifically an approach to estimating the probability or rate of failure modes of a particular software (i.e., it is not technically a QSRM), but is more of an overall integrated risk-modeling approach that incorporates hardware, software, and the static or dynamic interactions between them. The principal focus of CSRM is on uncovering the type of problems that traditional software reliability approaches typically overlook, that is, logic errors triggered by off-nominal system conditions, since ASCA believes these errors are the dominant contributors to system risk from software errors and require a more targeted approach for quantification.

The CSRM approach appears promising in its ability to risk-inform the software testing process for software-related failure scenarios that involve a combination of software and hardware failures. In such an approach, the software testing process does not need to prove that the software is error-free through tens or hundreds of thousands of demands or hours of operation, but needs only to prove that the software correctly responds to randomly selected combinations of inputs associated with system conditions (contexts) that may not have been thought of by system and software designers. However, the context-based evaluation has to be carried out for each software-related failure scenario that involves a combination of software and hardware failures, and based on the example for one software-related failure scenario described in [ASCA 2007], the quantification approach appears to be relatively resource intensive.

As discussed previously, ASCA [2009] proposes four methods for assessing a probabilistic parameter of a particular software:

1. Use of “surrogate data”
2. Use of a parametric regression model
3. Use of expert opinion
4. Use of system-specific software test data

While the first three of these methods (surrogate data, parametric regression model, and expert opinion) appear to be applicable to a software function, in general, it is not clear that they can be effectively applied at the level of resolution needed to differentiate between different contexts

(i.e., different conditions of the system in which the software is embedded). The remaining approach (use of system-specific software test data) is the most innovative and promising, though it does require the availability of a full system simulator package for performing the context-based, risk-informed testing. Without such testing capability, it appears that the CSRM approach would need to rely on the same type of QSRMs addressed elsewhere in this report.

Lastly, a key issue for all QSRMs that rely on test data involves how well a software's testing profile matches its operational profile. As described in [ASCA 2007], when performing risk-informed testing (essentially by applying method 4, above), one way this issue can be addressed is by applying an adjustment factor obtained through expert elicitation. Based on the information and examples provided in [ASCA 2007], it appears that adjustment factors of up to a factor of 50, or assumed conditional probabilities of failure up to 1.0, may be applied. Since these adjustment factors can significantly influence the estimation of the software failure probability (or rate), and are determined and applied based on expert elicitation, the application of this method is likely to entail significant subjectivity and uncertainty, and the resulting factors may not be bounding. Nonetheless, the application of adjustment factors appears to be one possible way of addressing the software testing conditions.

In summary, CSRM is not specifically an approach to estimating the probability or rate of failure modes of a particular software (i.e., it is not technically a QSRM), but is more of an overall integrated risk-modeling approach that incorporates hardware, software, and the static or dynamic interactions between them. It appears reasonable as a means of risk-informing the software testing process in support of assessing software reliability. Aspects of the CSRM approach can also be used to support quantifying software failure rates or demand failure probabilities for inclusion into an existing digital system reliability model. The most unique aspect of the CSRM approach is the context-based, risk-informed testing using a logically defined and partitioned input-parameter space for scenarios that involve off-nominal conditions (i.e., scenarios involving anomalous events such as one or more component hardware failures). This context-based evaluation has to be carried out for each software-related failure scenario that involves a combination of software and hardware failures. The publicly available reports specifically on CSRM [ASCA 2007, ASCA 2009] provide only one example of the implementation of the context-based, risk-informed testing approach. The CSRM reports suggest that such testing can be practically accomplished for the potentially large number of software failure modes that may need to be addressed given the level of resolution that CSRM applies in modeling software behavior, but it is not clear from the available information what amount of time and resources would be required. It should also be noted that, as indicated in [ASCA 2009], the context-based, risk-informed testing is not meant to be applied to scenarios that occur under nominal system conditions (e.g., those that do not involve anomalous failures), since these types of scenarios can be quantified using existing software reliability estimation models.

6.4 Rule/Standard Based Methods

IEC (International Electrotechnical Commission) Standard 61508 [IEC 61508] is a general standard for safety-related systems that specifies the requirements of the systems and provides guidance on assigning safety integrity levels (SILs). Part 1 of the standard designates the target failure probability for failure on demand and the target failure rates for different SILs. For example, the highest SIL is 4, with a target failure on demand probability of less than 10^{-4} and a target failure rate of less than 10^{-8} per hour. The recommended techniques and measures for

each software's SIL are given in the tables of Annex A and B of Part 3 of the standard; that is, to achieve a certain SIL, techniques/measures are recommended for the life-cycle activities of software. For example, for SIL 4, formal proof is highly recommended. Bloomfield [2007] suggested that uncertainty should be considered. In an example study, experts estimated the probability of failure of a safety critical system; the resulting distribution was used in a discussion on how uncertainty should be accounted for in assigning SILs.

The relationship between the SILs' qualitative requirements and the associated quantitative requirements/targets was assigned subjectively, and is not validated. Therefore, great care should be used in deciding whether use of the standard's software failure probabilities and rates is appropriate. Studies are needed to quantify the failure probabilities or rates of a few safety-related systems that meet the SIL requirements, using either operational experience or reliability modeling, in order to verify the validity of the reliability targets.

6.5 Quantification Methods for Software Diversities

The review of the previous methods was focused on their capability to estimate a failure rate or probability for a particular software. This section discusses the issue of software diversity, in particular N-version programming, which is important to consider when estimating the failure rate or probability of systems that include redundant software that is designed to accomplish the same function. For a nuclear power plant, one situation in which software diversity might be considered is that of a diverse digital system for reactor shutdown. A recent NRC-sponsored study on diversity strategies for digital instrumentation and control systems [Wood 2010] considers software diversity as a part of the overall digital system's diversity, that is, software diversity is considered jointly in hardware-related diversity strategies. It identifies a number of software diversities, such as usage of different algorithms, logic, program architectures, operating systems, and computer languages. It is desirable to quantify the benefits of different software diversity strategies so that they can be accounted for in reliability models.

N-version programming refers to use of multiple software development teams to develop different versions of software according to the same specification. It is an important software diversity strategy, and the feasibility of quantifying its benefits has previously been demonstrated. Eckhardt [1985] initially developed, and later Littlewood [1989] generalized, a mathematical theory that may afford a theoretical basis for considering N-version programming. The following four experiments of N-version programming were completed on different systems, each using a few programming teams to develop the application software. The findings were analyzed to determine if the software developed by different teams fails together.

1. Knight and Leveson's experiment of an anti-missile system [Knight 1986].

2. UCLA Six-Language project on an automatic landing system for commercial airliners [Avizienis 1988 and Lyu 1995].

3. NASA 4-University project of a redundant, strapped down inertia measurement unit (RSDIMU) of an inertia navigation system [Eckhardt 1991].

4. University of Iowa and Rockwell International's study of a computerized airplane landing system [Lyu 1993].

In general, the results of the four N-version programming experiments show that N-version programming can improve reliability. However, different versions of software do not fail completely independently. That is, a set of inputs may cause multiple versions to fail. The possible causes of this dependency include the following [Knight 1986]: (1) certain parts of a problem are more difficult to resolve than others, entailing the same faults despite different programmers, (2) the specification contains too much information on implementation, leading to limited diversity, and (3) the programmers' lack of understanding of key points in the specification. The degree of dependency identified in the studies varied. The information in the following table was taken from Lyu [1993]. It indicates the variability in the reduction factor of failure probability due to 3-version programming, compared with a single version, for each of the experiments identified previously.

In a more recent experiment, Lyu [2003] used a mutation testing technique in an N-version programming project consisting of 34 programming teams. The faults identified in the development stage were injected into versions of the final program to create mutants, each containing a fault. The experiment investigated the effectiveness of data flow coverage, mutation coverage, and design diversity for fault coverage. In another study Lyu [2005] undertook over 900,000 operational tests on the final versions of the 4-University project, and found that a 3-version system achieves a reduction factor of 80 to 330 compared with a single version.

Table 6-1 Factor of reduction in failure probability of 3-Version in comparison with 1-Version.

	Knight and Leveson	UCLA	NASA 4-University	University of Iowa/Rockwell International
3-Version Improvement	Not available	2 to 5	2.3	13

As documented above (for N-version programming), diversity strategies can significantly impact the overall system failure rate or probability if redundant software is implemented to accomplish a particular function. Therefore, the impact of these strategies should be accounted for when performing system reliability analysis.

7. SUMMARY AND PRINCIPAL FINDINGS

In this study, a review of currently available quantitative software reliability methods (QSRMs) was performed with the objective of cataloging potential methods that can be used to quantify software failure rates and demand failure probabilities of digital systems at nuclear power plants (NPPs) such that the system models can be integrated into a probabilistic risk assessment (PRA). The QSRMs were identified by reviewing research on digital system modeling methods sponsored by the Nuclear Regulatory Commission (NRC) and by the National Aeronautics and Space Administration, performed by international organizations, and published in journals and conferences. The QSRMs were categorized, described, and evaluated regarding their strengths and limitations for PRA applications. Note, as mentioned previously, specific recommendations regarding which QSRMs have the most potential for integration into a PRA, and are therefore candidates for further assessment, are beyond the scope of this study.

In general, for PRA modeling purposes, there are two types of digital systems at an NPP, that is, control and protection systems. A control system, such as a feedwater control system, performs its control function during normal plant operation. In contrast, a protection system, such as a reactor protection system (RPS), monitors the condition of the plant during normal operation, but only generates a reactor trip signal if a need arises. A control system may fail and cause a reactor trip, which would be included in a PRA as an initiating event (e.g., a loss of feedwater). An initiating event in a PRA, which is the starting point of an accident sequence analysis, is characterized by its annual frequency. Therefore, the frequency that a control system fails causing an initiating event needs to be estimated. A protection system may have two different failure modes. For example, an RPS may fail to generate a reactor trip signal when needed or may generate a spurious trip signal. A spurious trip signal would be included in a PRA as an initiating event and can be modeled in the same way failure of a control system is modeled, that is, in terms of an annual frequency. On the other hand, given the occurrence of some other initiating event that leads to the need for a reactor trip, a failure to generate a reactor trip signal would be modeled in the PRA in terms of a demand failure probability. Therefore, even for a single system, different QSRMs may need to be used depending on the failure modes of interest, that is, a failure-rate based method and a failure-on-demand based method.

It is well recognized that software failures are sensitive to the context (environment) in which the software is operating. Therefore, it is important that the software context be accounted for when modeling software failures in an NPP PRA (e.g., the specific system function being evaluated and the associated success criteria, as well as other relevant conditions in the plant). As an example, in a typical PRA, the RPS is usually the first top event in the event trees. Therefore, each initiating event defines a context for this system, that is, different initiating events represent different plant conditions that may generate different input signals to the RPS, and the system software may need to be modeled differently for different initiating events. For other protection (actuation) systems, such as a system which generates an actuation signal of an injection system, different sequences in different event trees define the different contexts for the actuation system and its software. More refined contexts can be determined by the cutsets of those sequences leading to the demand of the actuation system. Each of the cutsets represents a more specific scenario in which the system should function, and typically contains hardware failures and human errors that help determine the possible variations in the input signals to the system. It should be noted that the software context can be more refined than just what is

specified by the cutsets, and, therefore, the resolution of the PRA should be considered when choosing and implementing a modeling approach for software failures.

Currently, there is no consensus method for modeling digital systems in an NPP PRA. It is possible that reliability models of digital systems may include software failures representing different software failure modes at different levels of detail (e.g., the software may be modeled as separate software modules). However, a review of the literature revealed that practically all available QSRMs consider the software system as a whole, not as separate modules or broken down by failure mode. Depending on the reliability modeling method used for digital systems in a PRA, and the associated level of modeling detail, different QSRMs may be needed to quantify the digital system reliability model. In addition, it may be necessary to separately model different types of software (e.g., application-specific software and operating system software), using different QSRMs.

As part of the study, a set of desirable characteristics for QSRMs for modeling the digital systems operating at an NPP was proposed. The desirable characteristics can be used in evaluating available QSRMs and their applications to determine if the characteristics are satisfied. In particular, it is desirable that a method be capable of demonstrating the high reliability of a safety-critical system (e.g., a failure on demand probability on the order of 10^{-5}, commensurate with an analog RPS). Although an itemized evaluation of the reviewed methods against the desirable characteristics is beyond the scope of this study, the information documented in this report is useful for performing such an evaluation.

Only a few publicly available studies that attempted to quantify software failure rates and demand failure probabilities were performed by organizations that are related to the nuclear industry, for example, the Bayesian Belief Network (BBN) studies performed at the Technical Research Center of Finland VTT and Halden Reactor Project [Gran 2002a, Helminen 2007]. However, these particular studies did not address NPP digital systems. Even fewer publicly available studies were performed specifically to analyze digital systems at an NPP and these studies were explorative in nature. For example, the BBN study [EOM 2004] performed at the Korean Atomic Energy Research Institute (KAERI) addressed the quality of the software requirement specification of an RPS. However, even in this latter case, the part of the study involving quantification of the software reliability is not publicly available. Therefore, the majority of methods reviewed in this report are those that have been used in other, non-nuclear industries.

The reviewed QSRMs were separated into four major categories: (1) software reliability growth methods (e.g., see [IEEE 2008]), (2) BBN methods (e.g., see [Gran 2002a]), (3) test-based methods (e.g., see [Dahll 2007a]), and (4) other methods, which include a method based on software-development practices of past software development projects [Neufelder 2002], metrics-based methods [Smidts 2004] , and the context-based software risk model (CSRM) [ASCA 2007]. In addition, a few studies that considered rule/standard-based methods [Bloomfield 2007] and software diversities [Lyu 2003] were reviewed.

The principal findings of the QSRM review are provided below. More detailed discussions and comments on each of the individual QSRM categories are provided in previous sections of the report.

1. Most of the existing QSRMs were not developed specifically for supporting quantification of software failure rates and demand failure probabilities to be used in reliability models of

digital systems. However, they do estimate software failure rates or probabilities, and use the estimates in supporting decision making during software development. For example, software reliability growth models (SRGMs) are used in supporting decisions on whether a particular software can be release based on its estimated failure rate, and the correlation approach developed by Neufelder [2002] is used to determine the software engineering practices that are most likely to reduce defects and to estimate warranty staffing levels. Since both of these methods use information obtained prior to placing the software into operation, the resultant estimates of software failure rate can be further updated with operating experience using a standard statistical Bayesian approach. However, since both of these methods only estimate software failure rates (as opposed to demand failure probabilities), they would generally be applicable only to NPP control systems and the spurious actuation of protection systems, and not to the failure of protection systems to initiate their protective function(s).[17]

2. Most of the SRGMs, if not all, can be expressed in terms of empirical formulas of the expected number of failures, μ(t), as a function of time, and parameter uncertainty associated with an SRGM model can be assessed. There are many SRGMs in the literature, but none is generally superior to the others, because all are based on assumed empirical formulas that are not applicable to all situations. An SRGM that provides the best fit for one set of data may not provide the best fit for a different data set. Standard goodness-of-fit methods can be used to identify the SRGM which provides the best fit for a given set of data, and techniques for determining which method makes better predictions are available. It should also be noted that performing the analysis with multiple SRGMs can be used as one means of addressing modeling uncertainty.

   The many methods and terminologies in the literature suggest that unifying SRGMs may be desirable, especially those methods in the exponential Non-Homogeneous Poisson Process (NHPP) category. The formulation described in Section 3.2.1 of this report is a generalization of SRGMs in terms of an NHPP.

3. BBN methods have the capability to aggregate disparate information about software (e.g., aggregation of software failure data and quality of software lifecycle activities that is assessed using expert elicitation [Gran 2000a and 2000b]) and to include parameter uncertainties as a part of the modeling. However, there are challenges in developing a BBN that takes full advantage of these capabilities, including the substantial development effort that is needed, the expertise of the BBN developers, the qualification of any experts used to elicit information, and the availability of thorough documentation of the software development activities. Another challenge is that qualitative evidence (e.g., the impact of software development quality on software reliability) needs to be quantified. Since there may not be sufficient available data to "anchor" the conversion of the qualitative information to quantitative values, the uncertainty in the resultant quantitative estimates from the experts may be very large, which may make it difficult to demonstrate the small failure probabilities often associated with safety-related systems (a limitation common to many QSRMs).

---

[17] As mentioned previously, it is possible that SRGMs could be extended to modeling demand-type failures. In particular, there are two groups of methods, referred to as discrete SRGMs and discrete reliability growth methods (which remain to be reviewed), that may be more appropriate for modeling NPP protection systems.

4. The test-based methods use standard statistical methods with software testing and, conceivably, with operating data, if it is available. These methods are used in this study to demonstrate that a very large number of tests, ~10^5, must be conducted (and no failures observed) to demonstrate a mean software failure probability on demand of 10^{-5} (which is expected of an NPP safety-related system like an RPS). Besides the large number of tests required, there is also the concern that the testing environment may not represent the actual operating environment to which the software is exposed during operation in an NPP, which is a serious limitation on the accuracy of the testing results. Note, this issue applies to any QSRMs that use test data (e.g., SRGMs). Lastly, testing may not uncover errors in requirements and specifications of software, which have caused many software failures [ASCA 2007, Lutz 2004], though this limitation is common among many of the QSRMs reviewed.

5. The general concept of performing correlation/regression analyses using past software development experience is reasonable. However, because of the unavailability of detailed information on the past software development projects and the correlation/regression analyses used to construct the predictive model in [Neufelder 2002], this particular method could not be evaluated in detail. Based on a review of the information available, some potential limitations of Neufelder's predictive model include (1) subjectivity in the responses to the survey of software-development practices; (2) large uncertainties associated with the process for determining the ratio between inherent defects and failure rate, because it does not involve the use of information related to the specific software being assessed; and (3) it is not known whether Frestimate was validated or benchmarked by organizations independent from the organization (SoftRel, LLC) that developed it. Also, similar to most QSRMs, Frestimate does not specifically account for the context in which software operates, does not consider specific software failure modes, and does not appear to be capable of estimating probabilistic parameters of software when the expected values of the parameters are small, as would be expected for protection systems in an NPP.

6. NUREG/CR-6848 documents the use of 6 metrics methods for predicting software reliability (referred to as reliability prediction systems [RePSs]), each based on one of the 40 "root"[18] software engineering measures (SEMs) identified and ranked by a set of experts, as documented in NUREG/GR-0019. The RePSs were developed by applying available methods, concepts, and empirical formulas, and do not represent new innovative methods. The root SEMs were applied in an orthogonal, independent manner and the available documentation does not indicate that any work has been done on development of metrics methods that combine some or all of the highly ranked SEMs. Also, some of the metrics methods use engineering insights (as opposed to application-specific information) to develop empirical formulas representing the relationship between software reliability, that is, failure rate and probability, and software engineering measures. Since the empirical formulas are not laws of software reliability, their general applicability and accuracy are limited. Lastly, NUREG/CR-6848 considers that the results of the study validated the overall approach by showing that highly ranked SEMs produce results that are closer to the true answer. However, such a conclusion depends on the quantification methods developed and associated with the SEMs. Alternative quantification methods that may produce very

[18] As stated in NUREG/CR-6848, "the measure on which RePS construction is based is termed the 'root' of the RePS. Other measures within the RePS are defined as 'support' measures."

different results can be developed and associated with the SEMs (as indicated in NUREG/GR-0019) and potentially lead to a different conclusion.

7. CSRM is not specifically an approach to estimating the probability or rate of failure modes of a particular software (i.e., it is not technically a QSRM), but is more of an overall integrated risk-modeling approach that incorporates hardware, software, and the static or dynamic interactions between them. It appears reasonable as a means of risk-informing the software testing process in support of assessing software reliability. Aspects of the CSRM approach can also be used to support quantifying software failure rates or demand failure probabilities for inclusion into an existing digital system reliability model. The most unique aspect of the CSRM approach is the context-based, risk-informed testing using a logically defined and partitioned input-parameter space for scenarios that involve off-nominal conditions (i.e., scenarios involving anomalous events, such as one or more component hardware failures). This context-based evaluation has to be carried out for each software-related failure scenario that involves a combination of software and hardware failures. The publicly available reports specifically on CSRM [ASCA 2007, ASCA 2009] provide only one example of the implementation of the context-based, risk-informed testing approach. The CSRM reports suggest that such testing can be practically accomplished for the potentially large number of software failure modes that may need to be addressed given the level of resolution that CSRM applies in modeling software behavior, but it is not clear from the available information what amount of time and resources would be required. It should also be noted that, as indicated in [ASCA 2009], the context-based, risk-informed testing is not meant to be applied to scenarios that occur under nominal system conditions (e.g., those that do not involve hardware failures), since these types of scenarios can be quantified using existing software reliability estimation models.

8. REFERENCES

[Adams 1984] Adams, E.N., "Optimizing preventive service of software products," IBM Journal of Research and Development, vol. 28, no. 1, pp. 2-14, January 1984.

[Aldemir 2006] Aldemir, T., et al., "Current State of Reliability Modeling Methodologies for Digital Systems and Their Acceptance Criteria for Nuclear Power Plant Assessments," NUREG/CR-6901, February 2006.

[Aldemir 2007] Aldemir, T., Stovsky, M. P., Kirschenbaum, J., Mandelli, D., Bucci, P., Mangan, L. A., Miller, D. W., Sun, X., Ekici, E., Guarro, S., Yau, M., Johnson, B. W., Elks, C., and Arndt, S. A., "Dynamic Reliability Modeling of Digital Instrumentation and Control Systems for Nuclear Reactor Probabilistic Risk Assessments," NUREG/CR-6942, October 2007.

[Aldemir 2009] Aldemir, T., Gurro, S., Kirschenbaum, J., Mandelli, D., Mangan, L.A., Bucci, P., Yau, M., Johnson, B., Elks, C., Ekici, E., Stovsky, M.P., Miller, D.W., Sun, X., Arndt, S.A., Nguyen, Q., and Dion, J., "A Benchmark Implementation of Two Dynamic Methodologies for the Reliability Modeling of Digital Instrumentation and Control Systems," NUREG/CR-6985, February 2009.

[ANSI/AIAA 1992] American National Standards Institute (ANSI)/ The American Institute of Aeronautics and Astronautics (AIAA), "Recommended Practice for Software Reliability," American National Standards Institute, ANSI/AIAA R-013-1992, February 23, 1993.

[ASCA 2007] ASCA, Inc., "Risk-Informed Safety Assurance and Probabilistic Risk Assessment of Mission-Critical Software-Intensive Systems," Report AR 07-01, June 15, 2007.

[ASCA 2009] ASCA, Inc., "Instruction Guide for Integration of the Context-Based Software Risk Model (CSRM) with a Cx Probabilistic Risk Assessment (PRA)," Report AR 09-03, September 15, 2009.

[Atwood 2002] Atwood, C.L., et al., "Handbook of Parameter Estimation for Probabilistic Risk Assessment," NUREG/CR-6823, November 2002.

[Avizienis 1988] Avizienis, A., Lyu, M.R., and Schutz, W., "In Search of Effective Diversity: A Sic-Language Study of Fault-Tolerant Flight Control Software," *Proceedings of 25th IEEE International Symposium on Fault-Tolerant Computing (IEEE FTCS-25)*, Volume III, 1988.

[Bangs 2000] Bangs, O., "Top-Down Construction and Repetitive Structures Representation on Bayesian Networks," *Proceedings of the 13th International Artificial Intelligence Research Symposium*, Florida 2000.

[Bastani 1993] Bastani, F., and Pasquini, A., "Assessment of a Sampling Method for Measuring Safet-Critical Software Reliability," *Proceedings, 5th International Symposium on Software Engineering*, May 1993.

[Beichelt 2002] Beichelt, F.E., and Fatti, L.P., *Stochastic Processes and Their Applications, Taylor & Francis*, London and New York, 2002.

[Bloomfield 2007] Bloomfield R.E., Littlewood, B., and Wright D., "Confidence: Its Role in Dependability Cases for Risk Assessment," *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN"07)*, Edinburgh, UK**,** June 2007.

[Butler 1993] Butler, R.W. and Finelli, G.B., "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," *IEEE Transactions on Software Engineering*. Vol.19, No. 1, January 1993.

[Chillarege 1991] Chillarege, R., Kao, W.-L., and Condit, R. G., "Defect Type and Its Impact on the Growth Curve," *Proceedings of the 13th International Conference on Software Engineering,* 1991.

[Chu 2007] Chu, T.L., et al., "Basis for Using Software Failure Rates and Probabilities in Probabilistic Failure Modeling of Digital Systems of a Nuclear Power Plant," *IAEA Technical Meeting on "Common Cause Failures in Digital Instrumentation and Control Systems of Nuclear Power Plants*," June 19-21, 2007, Bethesda, Maryland.

[Chu 2008] Chu, T.L., et al., "Traditional Probabilistic Risk Assessment Methods for Digital Systems," NUREG/CR-6962, October 2008.

[Chu 2009a] Chu, T.L., et al., "Modeling a Digital Feedwater Control System Using Traditional Probabilistic Risk Assessment Methods," NUREG/CR-6997, September 2009.

[Chu 2009b] Chu, T.L., et al., "Workshop on Philosophical Basis for Incorporating Software Failures into a Probabilistic Risk Assessment," Brookhaven National Laboratory, Technical Report, BNL-90571-2009-IR, November 2009.

[Cinlar 1975] Cinlar, E., *Introduction to Stochastic Processed,* Prentice-Hall, 1975.

[Ciupa 2008] Ciupa, I., et al., "Finding Faults: Manual Testing vs. Random+ Testing vs. User Reports,**"** *19th International Symposium on Software Reliability Engineering*, pp.157 – 166, November 2008.

[Costa 2005] Costa, E.O., et al., "Modeling Software Reliability Growth with Genetic Programming," *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, (2005).

[Cukic 1998] Cukic, B., "Accelerated Testing for Software Reliability Assessment," *2nd Annual International Conference MIPRO'98*, Opatija, Croatia, May 1998.

[Dahll 2002] Dahll, G., Gran, B.A., and Liwang, B., "Decision Support for Approval of Safety Critical Programmable Systems," NEA/CSNI/R, 2002.

[Dahll 2007] Dahll, G., Liwang, B., and Pulkkinen, U., "Software-Based System Reliability," Technical Note, NEA/SEN/SIN/WGRISK(2007)1, Working Group on Risk Assessment (WGRISK) of the Nuclear Energy Agency, January 26, 2007.

[Duane 1964] Duane, J.T., "Learning curve approach to reliability monitoring," *IEEE Transactions on Aerospace*, vol. 2, no. 2, pp. 563–566, April 1964.

[Eckhardt 1985] Eckhardt, D.E., Jr., and Lee, L.D., "A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors," *IEEE Transactions on Software Engineering,* Volume SE-11, Issue 12, pp. 1511 – 1517, December 1985.

[Eckhardt 1991] Eckhardt, D.E., et al., "*An experimental evaluation of software redundancy as a strategy for improving reliability*", IEEE Transactions on Software Engineering, 17 (7), pp. 692-702, July 1991.

[Eom 2004] Eom, H-S., et.al., "A Study of the Quantitative Reliability Estimation of Safety-Critical Software for Probabilistic Safety Assessment," *4th ANS Topical Meeting on Nuclear Plant Instrumentation, Controls and Human-Machine Interfaces Technologies (NPIC&HMIT 2004)*, Columbus Ohio, September 2004.

[Eom 2009] Eom, H-S., et.al., "Reliability Assessment of a Safety-Critical Software by Using Generalized Bayesian Nets," ," *6th ANS Topical Meeting on Nuclear Plant Instrumentation, Controls and Human-Machine Interfaces Technologies (NPIC&HMIT 2009)*, Knoxville, Tennessee, April 5-9, 2009.

[Frank 1997] Frank, P., Hamlet, D., and Littlewood, B., "Choosing a Testing Method to Deliver Reliability," *International Conference on Software Engineering 1997*, Boston, Massachusetts.

[Garrett 1999] Garrett, C. and Apostolakis, G., "*Context in the Risk Assessment of Digital Systems," Risk Analysis*, 19:23-32, 1999.

[Goel 1979] Goel, A.L., and Okumoto, K., "Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures," *IEEE Transactions on Reliability*, Vol. R-28, No. 3, August 1979.

[Gran 2002a] Gran, B.A., and Helminen, A., "The BBN Methodology: Progress Report and Future Work," OECD Halden Reactor Project, HWR-693, August 2002.

[Gran 2002b] Gran, B.A., "Assessment of Programmable Systems Using Bayesian Belief Nets," *Safety Science* 40, pp. 797-812, 2002.

[Grottke 2001] Grottke, M., "Software Reliability Model Study," Information Society Technologies, IST-1999-55017, January 06, 2001.

[Guarro 1996] Guarro, S., Yau, M., and Motamed, M., "Development of Tools for Safety Analysis of Control Software in Advanced Reactors," NUREG/CR-6465, April 1996.

[Haapanen 2000] Haapanen, P., Korhonen, J., and Pulkkinen, U., "Licensing Process for Safety-Critical Software-Based Systems," Radiation and Nuclear Safety Authority (STUK) Report STUK-YTO-TR 171, Finland, December 2000.

[Hall 2008] Hall, J.B., and Mosleh, A., "A Reliability Growth Projection Model for One-Shot Systems," *IEEE Transactions on Reliability*, Vol. 57, Issue 1, March 2008.

[Helminen 2001] Helminen, A., "Reliability Estimation of Safety-Critical Software-Based Systems Using Bayesian Networks," STUK-YTO-TR178, June 2001.

[Helminen 2003a] Helminen, A., and Pulkkinen, U., "Quantitative Reliability Estimation of a Computer-Based Motor Protection Relay Using Bayesian Networks," SAFECOMP 2003, pp. 92-102, 2003.

[Helminen 2003b] Helminen, A. and Pulkkinen, U., "Reliability Assessment Using Bayesian Networks – Case Study on Quantitative Reliability Estimation of a Software-Based Motor Protection Relay," STUK-YTO-TR 198, 2003.

[Helminen 2005] Helminen, A., "Case Study on Reliability Estimation of Computer-based Device for Probabilistic Safety Assessment," VTT Technical Research Center of Finland, Research Report No. BTUO-051375, February 11, 2005.

[Helminen 2007] Helminen, A., "Case Study on Bayesian Reliability Estimation of Software Design of Motor Protection Relay," SAFECOMP, pp. 384-396, 2007.

[Huang 2003] Huang, C.-Y., Lyu, M.R., and Kuo, S.-Y., "A Unified Scheme of Some Nonhomogeneous Poisson Process Models for Software Reliability Estimation," *IEEE Transactions on Software Engineering*, Vol. 29, No. 3, March 2003.

[HUGIN 2010] HUGINEXPERT, "HUGIN", A Software Tool for Bayesian Network Analysis Developed by HUGINEXPERT, Denmark, http://www.hugin.com/, accessed February 5, 2010.

[IEC 61508] International Electrotechnical Commission, "Function Safety of Electrical/Electronic/Programmable Safety-Related Systems," Parts 1-7, IEC 61508, various dates.

[IEEE 2008] Institute of Electrical and Electronics Engineers (IEEE), "IEEE Recommended Practice on Software Reliability," IEEE Standard 1633-2008, March 27, 2008.

[Jelinski 1972] Jelinski, A. and Moranda, P., "Software Reliability Research," in W. Freiberfer, ed., *Statistical Computer Performance Evaluation*, Academic Press, New York, NY, pp. 465-484, 1972.

[Jensen 2002] Jensen, F.V., *Bayesian Networks and Decision Graphs*, Springer, 2002.

[Jordan 1997] Jordan, M.I., et al., "An Introduction to Variational Methods for Graphical Models," 1997.

[Kang 2009] Kang, H.G., et al., "Input-profile-based software failure probability quantification for safety signal generation systems," *Reliability Engineering and System Safety*, Vol. 94, Issue 10, doi:10.1016/j.ress.2009.02.018, pp. 1542-1546, October 2009.

[Kapur 1990] Kapur, P.K., et al., "Optimal Software Release Policies for Software Reliability Growth Models under Imperfect Debugging," RAIRO: *Operation Research*, 24: pp. 295 – 305, 1990.

[Kapur 1995] Kapur, P.K. and Younes, S., "Software Reliability Growth Model with Error Dependency," *Microelectron Reliability*, Vol. 35, No. 2, pp. 273 - 278, 1995.

[Kaufman 2001] Kaufman, L.M. and B.W. Johnson, "Embedded Digital System Reliability and Safety Analyses," NUREG/GR-0020, February 2001.

[Kent 1964] Kent, S., *Definition of Some Estimative Expressions*, Central Intelligence Agency, Center for the Study of Intelligence, 1964, available at https://www.cia.gov/library/center-for-the-study-of-intelligence/csi-publications/books-and-monographs/sherman-kent-and-the-board-of-national-estimates-collected-essays/6words.html, retrieved August 30, 2010.

[Knight 1986] Knight, J.C. and N.G. Leveson, "An Experimental Evaluation of the Assumption ofIndependence in Multi-Version Programming," *IEEE Transactions of Software Engineering*, 12 (1), pp. 96-109, 1986.

[Koller 1997] Koller, D., and Pjeffer, A., "Object-Oriented Bayesian Networks," *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, Providence, Rhode Island, 1997.

[Korb 2004] Korb, K.B. and Nicholson, A.E., *Bayesian Artificial Intelligence*, CRC Press, 2004.

[Lakey 1997] Lakey, P.B. and Neufelder, A.M., "System and Software Reliability Assurance Notebook," Rome Laboratory, Rome, NY, 1997. (This document does not have a report number, but it is available from http://www.softrel.com/publicat.htm).

[Langseth 2007] Langseth, H. and Portinale, L., "Bayesian Networks in Reliability," *Reliability Engineering and System Safety* 92, pp. 92 - 108, 2007.

[Lauritzen 1988] Lauritzen, S.L. and Spiegelhalter, D.J., "Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems," *Journal of the Royal Statistical Society*, Series B 50 (2), 1988.

[Lauritzen 1992] Lauritzen, S., "Propagration of Probabilities, Means, and Variances in Mixed Graphical Association Models," *Journal of the American Statistical Association*, 87, pp.1098-1108, 1992.

[Lawrence 1998] Lawrence, J.D., et al., “Assessment of Software Reliability Measurement Methods for Use in Probabilistic Risk Assessment,” Technical Report UCRL-ID-136035, Lawrence Livermore National Laboratory, 1998.

[Littlewood 1974] Littlewood, B. and Verrall, J.L., “A Bayesian Reliabiliity Model with a Stochastically Monotone Failure Rate,” III Transactions on Reliability, Vol. R-23, No. 2, June 1974.

[Littlewood 1989] Littlewood, B. and Miller, D.R., “Conceptual modeling of coincident failures in multiversion software, “ Software Engineering, *IEEE Transactions on Software Engineering*, Volume 15, Issue 12, pp.1596 – 1614, December 1989.

[Littlewood 1997] Littlewood, B., and Wright, D., “Some Conservative Stopping Rules for the Operational Testing of Safety-Critical Software,” *IEEE Transactions on Software Engineering*, Vol. 23, No. 11, November 1997.

[Littlewood 2000] Littlewood, B., et al., “Modeling the Effects of Combining Diverse Software Fault Detection Techniques,” *IEEE Transactions on Software Engineering*, Vol. 26, No.12, pp. 1157-1167, 2000.

[Littlewood 2005] Littlewood, B., “Dependability Assessment of Software-based Systems: State of the Art,” *27th International Conference on Software Engineering (ICSE'05)*, St. Louis, Missouri, USA, May 15-21, 2005.

[Littlewood 2007] Littlewood, B. and Wright, D., “The use of Multilegged Arguments to increase Confidence in Safety Claims for Software based Systems: A Study based on a BBN Analysis of an Idealized Example,” *IEEE Transactions on Software Engineering*, Vol. 33, No.5, May, 2007.

[Lutz 2004] Lutz, R., and Mikulski, I.C., “Ongoing Requirements Discovery in High-Integrity Systems,” *IEEE Software*, Vol 21, Issue 2, pp. 19-25, 2004.

[Lyu 1993] Lyu, M.R. and He, Y.-T., “Improving the N-Version Programming Process Through the Evolution of a Design Paradigm,” *IEEE Transactions on Reliability*, Vol. 42, No. 2, June 1993.

[Lyu 1995] Lyu, M.R., Editor, *Software Fault Tolerance*, John Wiley & Sons, 1995.

[Lyu 1996] Lyu, M.R., Editor-in-Chief, *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.

[Lyu 2003] Lyu, M.R., et al., “An Empirical Study on Testing and Fault Tolerance for Software Reliability Engineering,” *Proceeding of the 14th International Symposium on Software Reliability Engineering (ISSRE’03)*, 2003.

[Lyu 2005] Lyu, M.R. and Vouk, M.A., "An Experimental Evaluation on Reliability Features of N-Version Programming," *Proceedings of the 16th International Symposium on Software Reliability Engineering (ISSRE/05)*, 2005.

[Lyu 2007] Lyu, M.R., "Software Reliability Engineering: A Roadmap," *Proceedings of Future of Software Engineering*, pp. 153-170, 2007.

[May 1995] May, J., Hughes, G., and Lunn, A.D., "Reliability Estimation from Appropriate Testing of Plant Protection Software," *Software Engineering Journal*, November 1995.

[McCall 1992] McCall, J., et al., "Software Reliability, Measurement, and Testing - Software Reliability and Test Integration," Report RL-TR-92-52, Rome Laboratory, Rome, NY, April 1992.

[Miller 1992] Miller, K.W., et al., "Estimating the Probability of Failure When Testing Reveals No Failures," *IEEE Transactions on Software Engineering*, Vol. 18, No. 1, January 1992.

[Murphy 1998] Murphy, K., "A Brief Introduction to Graphical Models and Bayesian Networks," 1998, available online at http://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html.

[Murphy 2005] Murphy, K., "Software Packages for Graphical Models / Bayesian Networks," http://www.cs.ubc.ca/~murphyk/Bayes/bnsoft.html, last updated in 2005.

[Musa 1975] Musa, J., "A Theory of Software Reliability and Its Application," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 3, September 1975, pp. 312-327.

[Musa 1984] Musa, J.D. and Okumoto, K., "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement, " *Proceedings of the 7th International Conference on Software Engineering*, Orlando, Fl, pp. 230-28, March 1984.

[Musa 1987] Musa, J.D., Iannino, A., and Okumoto, K., *Software Reliability: Measurement, Prediction, Application*, New York: McGraw-Hill, 1987.

[Musa 2004] Musa, J.D., *Software Reliability Engineering: More Reliable Software Faster and Cheaper*, 2nd Edition, Author House, 2004.

[National Research Council 1997] National Research Council, "Digital Instrumentation and Control Systems in Nuclear Power Plants: Safety and Reliability Issues," National Academy Press, Washington, DC, 1997.

[NEA 2009] Nuclear Energy Agency, Committee on the Safety of Nuclear Installations, "Recommendations on Assessing Digital System Reliability in Probabilistic Risk Assessments of Nuclear Power Plants," NEA/CSNI/R(2009)18, December 17, 2009.

[Neil 2005] Neil M., Fenton, N., and Tailor, M., "Using Bayesian Networks to Model Expected and Unexpected Operational Losses," *Risk Analysis, Vol. 25, No. 4, 2005.*

[Neufelder 2000a] Neufelder, A.M., "How to Measure the Impact of Specific Development Practices on Fielded Defect Density," ISSRE 2000. *Proceedings of the 11th International Symposium on Software Reliability Engineering*, pp. 148-160, October 8-11, 2000.

[Neufelder 2000b] Neufelder, A.M., "How to Predict Software Defect Density during Proposal Phase," *Proceedings of the IEEE 2000 National Aerospace and Electronics Conference, (NAECON) 2000*, pp. 71 – 76, October 10-12, 2000.

[Neufelder 2002] Neufelder, A.M., "The Facts about Predicting Software Defects and Reliability," *The Journal of the Reliability Analysis Center (RAC)*, Second Quarter – 2002.

[Neufelder 2006] Neufelder, A.M., "Reducing Software Defects Efficiently - Facts and Fiction," http://www.softrel.com/myths.htm (PDF file), accessed on March 20, 2009.

[Obha 1984] Obha, M., Software Reliability Analysis Models, IBM,. *Journal of Research and Development*, Vol. 28: pp. 428 - 478.

[Okamura 2004] Okamura, H., Murayama, A., and Dohi, T., "EM Algorithm for Discrete Software Reliability Models: A Unified Parameter Estimation Method," IEEE International Symposium on High Assurance Systems Engineering (HASE'04), 2004.

[Parnas 1990] Parnas, D., Van Schouwen, A., and Kwan, P., "Evaluation of Safety-critical Software," *Communications of the ACM*, Vol. 33 (6), pp.636–648, 1990.

[Pearl 1988] Pearl, J. *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, San Francisco, 1988.

[Popov 2004] Popov, P. and Littlewood, B., "The Effect of Testing on Reliability of Fault-Tolerant Software," *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, 2004.

[Pressman 2001] Pressman, R.S., *Software Engineering, A Practitioner's Approach*, 5th edition, McGraw-Hill Higher Education, 2001.

[RTCA 1999] Radio Technical Commission for Aeronautics, "Software Considerations in Airborne Systems and Equipment Certification," RTCA/DO-178B, prepared by Special Committee 167(SC-167), 1999.

[Saul 1996] Saul, L., Jaakkola, T., and Jordan, M., "Mean Field Theory for Sigmoid Belief Networks," *Journal of Artificial Intelligence Research*, Vol. 4, pp. 61-76, 1996.

[Scheines] Scheines, R., "D-separation," online document available at http://www.andrew.cmu.edu/user/scheines/tutor/d-sep.html.

[Shachter 1989] Shachter, R. and Kenley, C., "Gaussian Influence Diagrams," *Management Science*, 35, pp. 527-550, 1989.

[Schneidewind 1975] Schneidewind, N.F., "Analysis of Error Processes in Computer Software," *Proceedings of International Conference on Reliable Software*, Los Angeles, pp. 337-346, 1975.

[Shooman 1972] Shooman, M.L., "Probabilistic Models for Software Reliability Prediction," in W. Freiberfer, ed., *Statistical Computer Performance Evaluation*, Academic Press, New York, NY, pp. 485-502, 1972.

[Shooman 1983a] Shooman, M.L., *Software Engineering: Design, Reliability, and Management,* New York McGraw-Hill Book Co., 1983.

[Shooman 1983b] Shooman, M. L. and Richeson, G.L., "Reliability of Shuttle Mission Control Center Software," *Proceedings of Annual Reliability and Maintainability Symposium*, 1983.

[Singpurwalla 1999] Singpurwalla, N.D. and Wilson, S.P., *Statistical Methods in Software Engineering – Reliability and Risk*, Springer-Verlag, New York, 1999.

[Sitte 1999] Sitte, R., "Comparison of Software Reliability Growth Predictions: Neural Networks vs. Parametric Recalibration," *IEEE Transactions on Software Engineering*, Vol. 48(3), pp. 285 - 291, (1999).

[Smidts 1999] Smidts, C. and Sova, D., "An Architectural Model for Software Reliability Quantification: Sources of Data," *Reliability Engineering and System Safety*, Vol. 64, pp. 279-290, 1999.

[Smidts 2000] Smidts, C. and Li, M., "Software Engineering Measures for Predicting Software Reliability in Safety Critical Digital Systems," NUREG/GR-0019, November 2000.

[Smidts 2004] Smidts, C.S. and Li, M., "Preliminary Validation of a Methodology for Assessing Software Quality,"NUREG/CR-6848, July 2004.

[SoftRel2006] SoftRel, LLC, "Frestimate Tutorial Version 3.6," http://www.softrel.com/downloads/tutorial36.pdf, accessed on March 20, 2009.

[SoftRel 2009a] SoftRel, LLC, "A tour through Frestimate," http://www.softrel.com/downloads/walkthru.pdf, accessed on March 20, 2009.

[SoftRel 2009b] SoftRel, LLC, "Frestimate Software," http://www.softrel.com/prod01.htm, accessed on March 20, 2009.

[Son 2009] Son, H.S., Kang, H.G., and Change, S.C., "*Procedure for Application of Software Reliability Growth Models to NPP PSA*," *Nuclear Engineering and Technology*, Vol. 41, No. 8, October 2009.

[Spiegelhalter 2003] Spiegelhalter, D., et al., "WinBUGS User Manual," Version 1.4, January 2003.

[Tregoning 2008] Tregoning, R., Abramson, L., Scott, P., "Estimating Loss-of-Coolant Accident (LOCA) Frequencies Through the Elicitation Process", NUREG-1829, April 2008.

[Stringfellow 2002] Stringfellow, C. and Andrews, A. A., "*An Empirical Method for Selecting Software Reliability Growth Models*," Empirical Software Engineering 7, pp. 319 - 343, 2002.

[USNRC 1995] United States Nuclear Regulatory Commission (USNRC), "Use of Probabilistic Risk Assessment Methods in Nuclear Regulatory Activities," Final Policy Statement, August 16, 1995.

[USNRC 2001] USNRC, "NRC Research Plan for Digital Instrumentation and Control," SECY-01-0155, August 15, 2001.

[Voas 1992] Voas, J.M., "PIE: A Dynamic Failure-Based Techinque," *IEEE Transactions on Software Engineering*, Vol. 18, pp. 717-27, 1992.

[Vouk 1990] Vouk, M.A., et al., "Analysis of Faults in a Large-Scale Multi-Version Software Development Experiment," *Proceedings of Digital Avionics Systems Conference*, 1990.

[Williams 2006] Williams, D.R., "Prediction Capability Analysis of Two and Three Parameters Software Reliability Growth Models," *Information Technology Journal*, 5 (6): pp. 1048 - 1052, (2006).

[Wood 2010] Wood, R.T., et al., "Diversity Strategies for Nuclear Power Plant Instrumentation and Control Systems," NUREG/CR-7007, February 2010.

[Yamada 1983] Yamada, S., Ohba, M., and Osaki, S., "S-Shaped Reliability Growth Modeling for Software Reliability Growth Modeling for Software Error Detection," *IEEE Transactions on Reliability*, Vol. R-32, No. 5, December 1983.

[Yamada 1984] Yamada, S., Ohba, M., and Osaki, S., "S-shaped Software Reliability Growth Models and Application, *IEEE Transactions on Reliability*, Vol. 11, No. 12, pp. 289-292, 1984.

[Yamada 1985] Yamada, S. and Osaki, S., "Discrete Software Reliability Growth Models," *Applied Stochastic Models and Data Analysis*, Vol. 1, 65-77, 1985.

[Yau 1995] Yau, M., Guarro, S., and Apostolakis, G., "Demonstration of the Dynamic Flowgraph Methodology using the Titan II Space Launch Vehicle Digital

Flight Control System," *Reliability Engineering and System Safety*, Vol. 49, pp. 335-353, 1995.

[Yau 1998] Yau, M., Apostolakis, G., and Guarro, S., "The use of prime implicants in dependability analysis of software controlled systems," *Reliability Engineering and System Safety*, Vol. 62, pp. 23-32, 1998.

[Xie 1992] Xie, M. and Zhao, M, "The Schneidewind Software Reliability Model Revisited," *Proceedings of 3rd International Symposium on Software Reliability Engineering*, 1992.

[Zhang 1996] Zhang, N. L. and Poole, D., "Exploiting Causal Independence in Bayesian Network Inference," *Journal of Artificial Intelligence Research 5*, 1996.

[Zhang 2004] Zhang, Y., "Reliability Quantification of Nuclear Safety-Related Software," Ph. D. Thesis, Department of Nuclear Engineering, Massachusetts Institute of Technology, February 2004.