

**NUREG/CR-xxxx
BNL-yyyyy-2011**

**DEVELOPMENT OF QUANTITATIVE SOFTWARE RELIABILITY
MODELS FOR DIGITAL PROTECTION SYSTEMS OF NUCLEAR
POWER PLANTS**

DRAFT REPORT FOR COMMENT

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

Software Failure Quantification

JCN N-6919

May 2011

Prepared for

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

TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
ABSTRACT [To be completed]	
FORWARD [To be completed]	
LIST OF TABLES.....	v
LIST OF FIGURES	v
EXECUTIVE SUMMARY [To be completed]	
ACKNOWLEDGEMENTS [To be completed]	
ACRONYMS AND ABBREVIATIONS.....	vi
1. INTRODUCTION.....	1-1
1.1 Background	1-1
1.2 Objective and Scope	1-2
1.3 Approach.....	1-3
2. SELECTION OF QUANTITATIVE SOFTWARE RELIABILITY METHODS	2-1
2.1 Common Limitations of QSRMs	2-1
2.2 Evaluation of Quantitative Software Reliability Methods	2-3
2.2.1 Software Reliability Growth Models (SRGMs).....	2-3
2.2.2 Bayesian Belief Network (BBN) Method	2-4
2.2.3 Test-Based Methods	2-5
2.2.4 A Correlation Method Using Software Development Practices	2-6
2.2.5 Metrics-Based Methods.....	2-7
2.2.6 Standard-Based Method	2-7
2.2.7 Other Considerations	2-7
2.3 Selected Candidate Methods	2-8
3. SELECTION OF AN EXAMPLE SYSTEM AND SYSTEM DESCRIPTION	3-1
3.1 Example System Identification and Familiarization	3-1
3.2 System Description	3-2
3.2.1 Design and Operation of ATR Loop Operating Control System.....	3-4
3.2.2 Description of Software Development	3-6
4. DEVELOPMENT OF A FAILURE-ON-DEMAND BASED SOFTWARE RELIABILITY GROWTH METHODS.....	4-1
4.1 Introduction	4-1
4.2 Discrete NHPP for Demand Failure Probability	4-2
4.3 Major Issues with Discrete SRGM Applications	4-4

TABLE OF CONTENTS (CONT'D)

<u>Section</u>	<u>Page</u>
5. DEVELOPMENT OF A BBN MODEL TAKING INTO CONSIDERATION QUALITY OF SOFTWARE DEVELOPMENT ACTIVITIES AND ASSOCIATED GUIDANCE	5-1
5.1 Introduction	5-1
5.2 Development of a BBN Model	5-2
5.2.1 Development of a BBN that Captures the Quality of Software Development Activities	5-2
5.2.2 Conversion of Number of Faults to Software Failure Probability.....	5-3
5.2.3 Direct Estimation of Software Failure Probability Using Expert Elicitation	5-4
5.2.4 Summary Discussion of Assumptions and Issues	5-5
5.2.5 Treatment of Uncertainty.....	5-8
5.3 Procedure for Developing a BBN Model of an Example System	5-9
5.4 Evaluation of the BBN Approach Against the Desirable Characteristics	5-12
6. DEVELOPMENT OF A CONTEXT-BASED STATISTICAL TESTING METHOD	6-1
6.1 Introduction	6-1
6.2 A Context-Based Statistical Testing Method	6-2
6.2.1 Contexts Defined by a PRA	6-2
6.2.2 A Risk-Informed Strategy for Performing Statistical Tests	6-4
6.2.2.1 Accounting for the Contribution of Software Failure to the Risk of the Entire NPP	6-5
6.2.2.2 Accounting for the Contribution of Software Failure to the Failure Probability of a Mitigating System or Function of the NPP.....	6-9
6.2.3 Performing Statistical Tests	6-9
6.2.3.1 Specification of Needed Test Configuration.....	6-9
6.2.3.2 Generating Test Cases by Sampling from the Operational Profile and Evaluation Of Results	6-11
6.2.4 Bayesian Reliability Assessment	6-12
6.2.5 Summary Discussion of Assumptions and Issues	6-14
6.2.6 Treatment of Uncertainties.....	6-15
6.3 Demonstration of the Feasibility of the Risk-Informed Statistical Testing	6-16
6.4 Evaluation of the Statistical Testing Approach Against the Desirable Characteristics	6-17
7. CONCLUSION AND INSIGHTS.....	7-1
7.1 Selection of Candidates QSRMs and Common Limitations	7-1
7.2 Selection of an Example System.....	7-2
7.3 Development of QSRMs for PRA Use.....	7-2
8. REFERENCES.....	8-1
Appendix A Development of A Failure-on-Demand Based Software Reliability Growth Method	A-1
Appendix B Description of Tests Performed for Licensing Purposes	B-1

LIST OF TABLES

<u>Table</u>		<u>Page</u>
2-1	Evaluation of QSRRMs against desirable characteristics	2-9
6-1	Examples of conditions requiring reactor trip for a Westinghouse PWR.....	6-3

LIST OF FIGURES

<u>Figure</u>		<u>Page</u>
3-1	Advanced Test Reactor at Idaho National Laboratory	3-2
3-2	Core Cross Section of Advanced Test Reactor	3-3
3-3	Cross Section of the In-core Portion of a Typical Pressurized Water Loop	3-3
3-4	Pressurized Water Loop System	3-4
4-1	Parallel Development of Continuous and Discrete NHPP Models	4-3

ACRONYMS AND ABBREVIATIONS

ABWR	Advanced Boiling Water Reactor
ASME	American Society of Mechanical Engineer
ATR	Advanced Test Reactor
ATWS	Anticipated Transient Without Scram
BBN	Bayesian Belief Network
BEA	Battelle Energy Alliance
BNL	Brookhaven National Laboratory
CCF	Common Cause Failure
CD	Core Damage
CDF	Core Damage Frequency
CPU	Central Processing Unit
CSRM	Context-Based Software Risk Model
DCS	Distributed Control System
DNHPP	Discrete Non-Homogeneous Poisson Process
DPU	Distributed Processing Unit
DSRGM	Discrete Software Reliability Growth Model
EFC	Error-Forcing Context
ESFAS	Engineered Safety Features Actuation System
ESPS	Engineered Safeguard Protection System
FAT	Factory Acceptance Test
FPGA	Field Programmable Gate Array
GT	Grooming Test
I&C	Instrumentation and Control
I/O	Input/Output
IE	Initiating Event
IEC	International Electrotechnical Commission
INL	Idaho National Laboratory
IPT	In-pile Tube
KAERI	Korean Atomic Energy Research Institute
LERF	Large Early-Release Frequency
LOCA	Loss of Coolant Accident
LOCS	Loop Operating Control System
LPI	Low Pressure Injection
MLE	Maximum Likelihood Estimation
MSFIS	Main Steam and Feedwater Isolation System
MTTF	Mean Time to Failure

ACRONYMS AND ABBREVIATIONS (CONT'D)

NASA	National Aeronautics and Space Administration
NHPP	Non-Homogeneous Poisson Process
NPP	Nuclear Power Plant
NPT	Node Probability Tables
NQA	Nuclear Quality Assurance
NRC	Nuclear Regulatory Commission
OEI	Optical-to-electrical Interface
pmf	Probability Mass Function
PRA	Probabilistic Risk Assessment
PWR	Pressurized Water Reactor
QL	Quality Level
QSRM	Quantitative Software Reliability Method
RES	Office of Nuclear Regulatory Research
RG	Regulatory Guides
RPS	Reactor Protection System
RTC	Reactor Technology Complex
SG	Steam Generator
SIL	Safety Integrity Level
SO	System Operability
SRGM	Software Reliability Growth Model
STUK	Finnish Government Authority for the Nuclear Industry
SIVAT	Simulation Validation Test
VTT	Finnish Technical Research Center
V&V	Verification and Validation

1. INTRODUCTION

1.1 Background

The U.S. Nuclear Regulatory Commission's (NRC's) current licensing process for digital systems rests on deterministic engineering criteria. In its 1995 probabilistic risk assessment (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 much has been accomplished in the area of risk-informed regulation, the process of risk-informed analysis for digital systems is not fully developed. Since digital instrumentation and control (I&C) systems are expected to play an increasingly important safety role at nuclear power plants (NPPs), the NRC established a plan for digital system research [USNRC 2001] defining a coherent set of projects to support regulatory needs. One of the projects included in this research plan addresses risk assessment methods and data for digital systems. Digital I&C systems have unique characteristics compared with analog I&C systems, such as using software, and may have different failure causes and/or modes; hence, incorporating them in 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 in NPP PRAs, and (2) incorporating digital systems in the NRC's risk-informed licensing and oversight activities. For several years, Brookhaven National Laboratory (BNL) has worked on NRC projects, investigating 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]. The NRC also sponsored research at the Ohio State University investigating the modeling of digital systems using dynamic PRA methods, as detailed in NUREG/CR-6901 [Aldemir 2006], NUREG/CR-6942 [Aldemir 2007], and NUREG/CR-6985 [Aldemir 2009]. An important identified 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. Accordingly, BNL has been exploring how software failures¹ can be included into these reliability models, so that their contribution to the risk of the associated NPP can be assessed.

Based on a recommendation from the NRC's Advisory Committee on Reactor Safeguards Subcommittee on Digital I&C Systems, the NRC tasked BNL in 2008 with organizing and running an expert panel meeting (workshop) with the goal of establishing a "philosophical basis" for incorporating software failures into digital system reliability models for use in PRAs [Chu 2009b]. The experts were recognized specialists from around the world with knowledge of software reliability and/or PRA. A philosophical basis for incorporating software failures into a PRA was established at the meeting. The panelists also agreed that software failure rates and probabilities can be included in reliability models of digital systems, and identified a few general methods for quantifying software reliability.

Subsequently, BNL reviewed a spectrum of quantitative software reliability methods (QSRMs) to catalog potential methods that can serve to quantify software failure rates and per-demand

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 at some point during the software lifecycle (including faults that result from incomplete or incorrect requirements and specifications).

failure probabilities of digital systems at NPPs, such that the system models can be integrated into a PRA [Chu 2010]. 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 strengths and limitations of QSRMs for PRA applications were categorized, described, and evaluated. In addition, a set of desirable characteristics of a QSRM was established. While these 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 undertaken.

The study documented in this report continued the preceding work on software reliability by selecting candidate QSRMs and further developing them in preparation for a case study of the selected method(s). The actual case study (or studies) will be documented in a separate report.

1.2 Objective and Scope

The following are the objectives of this study: (1) Evaluate the QSRMs reviewed in [Chu 2010] against the desirable characteristics therein, and select a few candidate methods that can potentially be used in PRA modeling of software failures; (2) select a protection system for a case study of the selected method(s); (3) develop approaches for applying the candidate method(s) to the selected protection system; and (4) obtain insights into the feasibility, practicality, and usefulness of developing digital system models to include in PRAs.

In general, for PRA modeling purposes, there are two types of digital systems at an NPP, viz., control and protection systems. During normal operation, a control system, such as a feedwater control system, continuously performs its function (i.e., it continually receives input from the plant and sends out control signals to the plant). In contrast, a protection system, such as a reactor protection system (RPS), monitors the plant's condition during normal operation, but only generates a reactor trip or other appropriate signal when needed. 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, i.e., the starting point of an accident sequence analysis, is characterized by its annual frequency. Therefore, to meet the needs of a PRA, the frequency at which a control system fails causing an initiating event must 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. The latter would be included in a PRA as an initiating event and is modeled in the same way as the failure of a control system, that is, in terms of an annual frequency. On the other hand, after the occurrence of some other initiating event entailing a reactor trip, a failure to generate a reactor trip signal is modeled in the PRA as a demand failure probability. Therefore, even for a single system, different QSRMs may be needed, depending on the failure modes of interest, that is, a failure-rate based method and a failure-on-demand based method.

In this study, the scope of the method development is limited to modeling failures of protection systems to perform their functions (represented by the probability of failure on demand) at an NPP, mainly due to the importance of these safety-related functions.

Presently, there is no consensus method for modeling digital systems in NPP PRAs. The possibility exists 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 modules). However, a review of the literature [Chu 2010] 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 method of reliability modeling used for digital systems in a PRA, and the associated level of detail, different QSRMs may be needed to quantify the contribution of software failure to the digital system's failure probability or rate. 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, software failures are known to be sensitive to the context (environment) in which the software operates [Garrett 1999]. Therefore, it is important to account for software's context when modeling its failure in a PRA (e.g., the specific system function being evaluated, the associated success criteria, and other relevant conditions in the plant).

Due to the above limitations in the state-of-the-art methodologies, this study considered only a system level failure mode, that is, failure to perform the needed function. In addition, since the objective of this study is limited to quantifying the probability of failure of the software of an individual protection system, the potential for common-cause failure due to dependencies between two digital protection systems performing similar functions in the same accident scenarios was considered beyond the study's scope.

1.3 Approach

This study builds on earlier work at BNL (in particular, [Chu 2010]) in selecting candidate QSRMs and developing approaches for applying them to an example system. A set of desirable characteristics also was established and used to identify the example system. The derivation of these characteristics was based on what the candidate methods need, and practical considerations.

In identifying the candidate methods among those reviewed in [Chu 2010], the desirable characteristics for candidate QSRMs developed therein were enhanced through the addition of a characteristic on availability of needed data. The different QSRMs were compared against these enhanced characteristics, and the following three were chosen as candidates: The software reliability growth method, the Bayesian belief network (BBN) method, and the statistical testing method.²

For each of these three, the literature was reviewed further to assess its suitability for estimating demand-failure probabilities of safety-critical protection systems, and to formulate an approach for applying it to an example system. As will be seen later in this report, it was decided that the statistical testing method is the preferred approach. However, due to limitations with this method, and to account for the quality of software-lifecycle activities, it was also decided to first develop a prior distribution via the BBN approach, and then update it by Bayesian inference using the statistical testing results. Since developing and quantifying a detailed BBN model can be very resource intensive, and the evidence needed for quantification may be lacking, both a simplified and detailed BBN model will be developed. This exercise will provide insight into the costs and benefits of developing a detailed model. It is expected that the prior distribution that results from both the simple and detailed BBN models will be fairly broad due to the high level of uncertainty associated with the use of expert opinion (particularly in cases where evidence is

² One may argue that statistical testing is a part of the BBN method because a BBN model that estimates a distribution for software failure on demand can easily be modified to include the test results.

very limited). As such, the statistical testing results will also be used to perform a Bayesian update of a non-informative prior, to provide an additional standard for comparison.

Software reliability experts from the Center for Software Reliability (CSR) at City University London (UK) reviewed and commented upon the draft working material. This research, and the input from CSR, identified issues and limitations of the methods and, in a few cases, possible ways to address them. It should be noted that it was originally intended to also develop a prior distribution using a discrete software reliability growth model (DSRGM) approach. However, following extensive discussions with CSR, it was ultimately decided not to further pursue the DSRGM approach due to the expected lack of data on demand-based debugging tests for highly reliable systems, as well as the other limitations with this approach discussed in Section 4.

Section 2 documents the evaluation of the QSRM methods against the set of desirable characteristics. Section 3 describes the selection of the example system for the case studies. Sections 4 to 6 detail the approach to developing the three candidate methods, that is, software reliability growth, BBN, and statistical testing. Section 7 gives the conclusions and the insights obtained.

2. SELECTION OF QUANTITATIVE SOFTWARE RELIABILITY METHODS

In this section, the quantitative software reliability methods (QSRMs) reviewed in an earlier study [Chu 2010] are evaluated against the desirable characteristics developed therein. The objective is to identify the most promising methods for quantifying software reliability to support formulating reliability models of digital systems for use in probabilistic risk assessments (PRAs) of nuclear power plants (NPPs). The set of desirable characteristics is repeated below. Note, an additional characteristic was added to take into consideration that some of the methods may require data, information, or models that are not available.

1. The description of the method and its application is comprehensive and understandable.
2. The assumptions of the method have reasonable bases.
3. The method allows for consideration of the specific operating conditions of the software.
4. The method takes into consideration the quality of lifecycle activities.
5. The method makes use of available test results and operational experience.
6. The method addresses uncertainty.
7. The method has been verified and validated.
8. The method is capable of demonstrating the high reliability of a safety-critical system.
9. The method should be able to estimate parameters that can be used to account for software common cause failures (CCFs),
10. The data and needed information exist and can be collected.

Section 2.1 describes common limitations of QSRMs. Section 2.2 assesses each method against the desirable characteristics by discussing its strengths and weaknesses associated with the characteristics documented in Chu [2010]; Table 2-1 summarizes the evaluation. Section 2.3 identifies candidate methods for a proof-of-concept case study.

2.1 Common Limitations of QSRMs

The following describes common limitations of current QSRMs. If a QSRM can address any of these limitations, this is further discussed in its evaluation.

Test profile vs. operational profile

Most QSRMs reviewed use available test and/or operational data. For test data, it often is assumed that the test cases were sampled from the software's operational profile, and that standard statistical methods were used in quantifying the software's failure rates or probabilities. Often, there is little description in these papers or reports on how the tests were done, and what was done to ensure that they truly represent the operational profile, which may not be well known. It is commonly known that test profiles may not realistically represent operational profiles [Miller 1992]. This source of uncertainty/inaccuracy should be accounted for in estimating software reliability.

Context specificity

Software failures are sensitive to the context (environment) in which the software is operating [Garrett 1999]. Therefore, it is important to consider this context when modeling software

failures in an NPP PRA (e.g., the specific system function being evaluated, the associated success criteria, and other relevant conditions in the plant).

The QSRMs reviewed for this study mostly consider the failure of a software program to perform its function without specifically considering the contexts in which the software is expected to function. In general, one may argue that a QSRM can be applied to specific contexts. In practice, only the Context-Based Software Risk Model [Guarro 2007] specifically addresses contexts, and it may be very difficult to apply some other methods on a context-specific basis. For example, for an engineered safety features actuation system (ESFAS) that must initiate safety injection in different sequences of different event trees, it probably is not practical for any of the other methods to address the specific system contexts, except that test-based methods might be able to do so via context-specific tests.

Demonstration of high reliability

It is expected that a digital reactor protection system (RPS) should have at least the reliability of the analog RPS it replaces (i.e., a failure probability on demand on the order of 10^{-5}). Statistically it would require undertaking hundreds of thousands of tests without failure to demonstrate this kind of reliability. Automated testing may be a means for performing a large number of tests. However, it may be difficult to demonstrate that the test cases are taken randomly from the operational profile. It is unlikely that any of the QSRMs can confidently demonstrate a failure probability of this order.

Failure-mode-specific modeling

As discussed in Section 1, the system level failure modes of digital systems at an NPP can be easily defined. However, depending on the level of detail of system modeling, quantification of the failure probabilities for lower level failure modes might be needed. For example, it may be desirable to use generic failure modes to model each microprocessor (e.g., a program of a microprocessor continues to run but generates incorrect results). Current QSRMs do not seem to address software failure modes at this level of detail.

CCF of two diverse digital systems

CCF is a very important issue for digital systems. It is a common assumption (though possibly conservative) that if the redundant channels of a safety-related system, that is, an RPS or ESFAS, run identical software, they would all fail given a software failure that leads to loss of channel function. However, an NPP may add a second, diverse digital shutdown system with a different software program, and it would be overly conservative to assume that the alternative shutdown system and the primary shutdown system would fail simultaneously. [Wood 2009] identifies many different diversity strategies.

N-version programming is a software diversity strategy that has been quantitatively assessed in experiments [Lyu 2005] that demonstrated that it can enhance software reliability. N-version programming can potentially be included in a reliability model using the concept of CCF models typically used in a PRA for modeling hardware failures (e.g., the beta factor model). Littlewood and Rushby [2010] developed a mathematical approach for assessing the probability of failure of a system consisting of two diverse software, a complex one and a simple one, that supports consideration of the potential that the simple software is perfect [Littlewood 2010]. The approach uses the probability that the two systems contain common faults as a parameter to be

subjectively assessed. For the complementary case (where there are no common faults), the approach uses statistical testing and the BBN method to estimate the probability that the complex system would fail, and assesses the fallibilities of the formal verification of the simple system to estimate the probability that it is imperfect.

Regardless of the level of diversity, the possibility that two different protection systems serving the same function can fail from the same cause cannot be completely precluded. Hence, methods are needed to account for potential dependencies between diverse systems. As discussed in Section 1, the objective of this study is limited to quantifying the probability of failure of the software of an individual protection system. Therefore, CCF across different systems, whether they are considered to be diverse or not, is beyond the scope of this study.

2.2 Evaluation of Quantitative Software Reliability Methods

In this section, the QSRMs are evaluated against the desirable characteristics. The evaluation is based mainly on the information documented in the review of QSRMs [Chu 2010], and identifies whether the method addresses each characteristic. The answers, “Yes”, “Maybe”, and “No” depend on whether the method conceptually meets the desirable characteristics and whether the method’s implementation is deemed practical. Table 2-1 summarizes the evaluation of the QSRMs against the desirable characteristics. It may be argued that some of the characteristics are more important than others and should, therefore, be given more weight. However, it is difficult to justify the subjective assignment of weights, as well as the resources required should a formal expert elicitation process be used for this purpose.

2.2.1 Software Reliability Growth Methods

Software reliability growth methods³ are the most developed and widely used quantitative software reliability methods, and they and their applications are well described in the literature, meeting Characteristic 1. They are used in determining if a piece of software is ready to be released, or how much more testing is needed before release. Software reliability growth models (SRGMs) have previously been demonstrated with test data from NASA and military applications, meeting Characteristic 7. These models are failure-rate based and applicable to control systems that normally run constantly. Essentially, they assume that the system’s failure rate decreases in time as more software faults are identified and fixed, and use test results to estimate the parameters of the models.

SRGMs depend heavily on availability of test data, that is, data collected during software development, though they also can be used with operational experience. As such, SRGMs meet Characteristic 5. On the other hand, debugging data of the software of a safety-related system is unlikely to be well documented, making it difficult to meet Characteristic 10. The fact that system failure rate is estimated from actual failure data implies that the estimated failure rate may not be very low, making it very difficult to satisfy Characteristic 8. However, assuming that the system’s failure rate decreases with time allows SRGMs to reflect the benefit of removing faults, unlike statistical testing methods that assume the software’s reliability does not change.

³ Many of the references for Sections 2 and 4 and Appendix A 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.

Many different SRGMs have been proposed. They all assume that the system's failure rate decreases with time; however, each SRGM assumes that the rate of decrease follows a different type of curve, expressed in terms of an empirical formula. Using a few different SRGMs with the same set of test data would allow variability among the models to be captured. Although parameter uncertainty of an SRGM is typically not assessed, this is not an inherent limitation of the models. For example, Musa described an approach for handling parameter uncertainty in his 1987 book on software reliability [Musa 1987]. Alternatively, a Bayesian analysis could be used to determine parameter uncertainties. Applying these approaches would satisfy Characteristic 6.

Some assumptions made in individual SRGMs may be incorrect, for example, assuming that a fault is fixed perfectly. However, as discussed in [Chu 2010], many assumptions made in different SRGMs do not need to be scrutinized too closely because, ultimately, for models based on empirical formulas, the most important aspect is how well they predict/fit the data. As such, SRGMs are rated "Maybe" for Characteristic 2. In addition, since SRGMs do not account for the software context or the quality of lifecycle activities, and cannot be used to estimate software CCF parameters, they do not meet Characteristics 3, 4, and 9.

The limitation that the SRGMs are only applicable to control systems potentially might be eliminated by extending discrete SRGMs to modeling software failure on demand. Discrete SRGMs (e.g., see Yamada [1985] and Okamura [2004]) consider test results in the form of the number of software faults discovered in each test, and show that this number follows a discrete non-homogeneous Poisson Process (DNHPP). They have the advantage that the times when the software failures occur do not have to be collected. Many continuous-time SRGMs can be converted into DNHPP models. These models represent software failures on demand in a way similar to how SRGMs model software failure rates. Section 4 describes how to develop a failure-on-demand-based method.

2.2.2 Bayesian Belief Network (BBN) Method

A BBN is a probabilistic graphical model depicting a set of random variables and their conditional independencies⁴ 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 their dependencies. The BBN method has been used successfully in non-nuclear applications and is well-documented in the literature, meeting Characteristic 1. It uses conditional probability tables to represent interdependency among disparate events, and, in PRA applications, can potentially combine qualitative information, such as quality of software lifecycle activities (thereby, meeting Characteristic 4), with quantitative information, such as test and operational data (meeting Characteristic 5). The method also was used in modeling both software failure rates and probabilities [Gran 2000, Eom 2004]. These applications appear to be explorative in nature, even though a few experts have stated that the method is promising [Littlewood 2000] (earning a "Maybe" for Characteristic 7). With a BBN approach, uncertainties associated with the events/nodes are represented explicitly in their probabilities, thereby meeting Characteristic 6. In [Gran 2000], a BBN model was formulated that assesses the quality of the software development activities to obtain a prior distribution for the probability of failure, and test results were used in a standard Bayesian analysis of the probability. While that study did not result in

⁴ In a BBN, a node is independent of its non-descendent nodes given its parent nodes.

estimates of very high reliability, it is possible that for nuclear protection system software a panel of experts could agree upon a very high reliability, potentially meeting Characteristic 8.

An important assumption of a BBN model is the conditional independence implied in its structure. One way to confirm the correctness of the BBN structure is to ask if any two nodes should be connected by an arc representing a “direct” dependency. If two nodes are not connected by an arc, then the only dependency between them should be through intermediate nodes. It may be difficult to demonstrate such implied conditional independence (i.e., showing that the two nodes should not have a direct link/arc), resulting in a ranking of “Maybe” for Characteristic 2.

Applications of the BBN method often use discretized probability distributions to represent the nodes of the models, and subjective expert elicitation in estimating the conditional probability tables representing the impacts that parent nodes have on child nodes. In particular, it is difficult to develop conditional probability tables that convert qualitative information, such as quality of software development activities, into failure probabilities and rates. It is unclear whether this is a task that can be appropriately resolved through expert elicitation, and whether the needed experts exist, resulting in a “Maybe” rating for Characteristic 10. In addition, since the applications of the BBN method do not account for the software context and it would be very difficult to use it to estimate software CCF parameters; it is assumed that the BBN method does not meet Characteristics 3 and 9.

2.2.3 Test-based Methods

The National Research Council committee [1997] recommended employing test results to determine software failure probabilities. For licensing purposes, tests have been used to identify software bugs and to demonstrate that a system performs its intended functions. However, the results are not suitable for quantifying software failure reliability because the test cases are not sampled randomly from the operational profile. Therefore, specific tests are needed to support test-based methods [May 1995]. The test configuration used in performing required tests for licensing can be adopted for statistical testing, thereby satisfying Characteristic 10.

Test-based methods fall into two categories (i.e., black-box and white-box methods) depending on how the tests are performed. These two methods employ different quantification approaches, and both are documented comprehensively, thereby meeting Characteristic 1. Black-box test-based methods do not consider the software structure and use standard statistical methods, including treating parameter uncertainty, to quantify software reliability based on test and operational data, thereby meeting Characteristics 5 and 6. As mentioned in Section 2.1, automated testing may be a means for undertaking a large number of tests, thereby potentially allowing black-box methods to satisfy Characteristic 8. Section 6 describes how the black-box methods can account for the specific contexts defined in a PRA, thereby likely meeting Characteristic 3. Conceptually and mathematically, the black-box method can be used in assessing software reliability, as has been done in non-software applications. Realistic applications of the black-box method to quantify failure probability of a digital protection system remain undemonstrated, so this method is rated “Maybe” for Characteristic 7.

White-box test-based methods were developed only in a few studies (e.g., [Zhang 2004] and May [1995]). Zhang [2004] considers the internal structure of the software but requires

additional resources, such as those for assessing the coverage of different structural elements (e.g., estimating the frequency of visitation at a path or node of a software program). May [1995] considers partitioning the input space of a software with associated partitioning of the area of the code that is executed, and estimates a failure probability for each partition. Both of these methods advantageously afford information about the internal structure of the software and may detect faults that would otherwise remain undetected. However, it may be difficult to force the execution of a software program to follow a specific path. Also, these white-box test-based methods derived expressions for the mean software failure probability in terms of the failure probabilities of individual nodes of the software or individual partitions of the input space. The expressions show that the number of tests without failure for each individual node or partition has to be equal to the number of tests without failure using a black-box method in order to produce the same mean software failure probability. Therefore, from the perspective of estimating software failure probability (as opposed to testing for licensing purposes), it is not obvious that the white-box methods would give better statistical results than black-box test-based methods. In addition, Zhang indicated that, due to the many possible paths, she was unable to evaluate all of them using her method. Due to the potentially much greater number of tests required for white-box testing, it is assigned a rating for Characteristic 7 (“No”) that is lower than that for black-box testing (“Maybe”). The ability (or inability) of white-box testing methods to meet all of the other desirable characteristics essentially is the same as that for black-box testing methods.

In general, any method that uses test data can be considered a test-based method and is subject to the limitations of test-based methods discussed in Section 2.1. In particular, the common assumption that the software’s testing profile is representative of its operational profile often is not well supported. Accordingly, all test-based methods are rated as “Maybe” for Characteristic 2. In addition, since test-based methods do not account for the quality of lifecycle activities, and would be difficult to use for estimating software CCF parameters, they do not meet Characteristics 4 and 9.

2.2.4 A Correlation Method Using Software Development Practices

Neufelder’s [2002] correlation approach provides a point estimate for the software failure rate at the end of the software testing stage, by making use of the software engineering practices of past software development projects. Similar to the BBN methods discussed in Section 2.2.1, the method attempts to account for the quality of software development activities, thereby meeting Characteristic 4. The general concept of undertaking correlation/regression analyses using past software development experience is reasonable and, in general, the resulting estimate can be further updated with test data and operating experience, thereby meeting Characteristic 5. However, Neufelder fails to give a solid technical basis for using her simple equation to convert the number of defects to a failure rate, thereby limiting the method’s ability to meet Characteristic 2. Because of the unavailability of detailed information on the past software development projects and the correlation/regression analyses used to construct its predictive model, Neufelder’s method could not be evaluated in detail, and, therefore, does not meet Characteristic 1. Information on these previous projects is an essential part of the method, and its proprietary nature leads to a rating of “Maybe” for Characteristic 10. Furthermore, since independent organizations have not validated or benchmarked the method, it only partially meets Characteristic 7. Neufelder indicated that an ultra-high reliability with a failure rate of 10^{-4} cannot be supported by the state of the art [Lahey 1997]; hence, the method does not meet Characteristic 8. Further, since it does not account for the software context, only provides point

estimates, and cannot be used to estimate software CCF parameters, it does not meet Characteristics 3, 6, and 9.

2.2.5 Metrics-Based Methods

Two of the methods described in NUREG/CR-6848 [Smidts 2004] are associated with mean time to failure (MTTF) and defect density, respectively. The former is a standard black-box test-based method, while the latter is considered a white-box test-based method that does not treat uncertainty (therefore, does not meet Characteristic 6). In addition, instead of testing the actual software, the defect density method uses the defects identified by inspection, and the frequency of visits to the paths of the software by running a finite-state machine model of the software. As such, this method is rated “Maybe” for Characteristic 5. For all other desirable characteristics, the MTTF method and defect-density method have the same answers as the black-box and white-box testing methods, respectively,

2.2.6 Standard-Based Method

IEC Standard 61508 [IEC 61508] defines safety integrity levels (SILs) for software and specifies recommended strategies/measures for each SIL; the strategies and measures are related to the quality of the software development activities (meeting Characteristics 1 and 4). For example, using formal methods in developing system requirement specifications is highly recommended for a SIL of 4. In addition, the IEC Standard 61508 assigns target failure rates and probabilities to the SILs. This may be considered a way of estimating software failure rate and probability at the system level based on lower level qualitative requirements, for example, the lowest software failure probability is 10^{-4} for a SIL of 4. The assignment, done by expert opinion (members of a committee), does not seem to have a strong basis, that is, it was not validated by case studies. Hence, this method does not meet Characteristics 2, 5, 7, 8, and 10. The standard allows different ways (i.e., different strategies and measures) of meeting the requirements of the same SIL, but the assignment of software failure rate and probability does not account for this variability, nor does the standard address other forms of uncertainty, thereby failing to meet Characteristic 6. Also, since IEC Standard 61508 does not account for the software context nor address the issue of software CCF parameters, it does not meet Characteristics 3 and 9.

2.2.7 Other Considerations

The Context-Based Software Risk Model (CSRМ) [Guarro 2007] is an overall integrated risk-modeling approach that incorporates hardware, software, and their static or dynamic interactions. As discussed in [Chu 2010], CSRМ focuses on context-based, risk-informed testing for scenarios that involve off-nominal conditions. It appears reasonable as a means of risk-informing the software testing process to support assessing software reliability. While aspects of the CSRМ approach can aid in quantifying software failure rates or demand failure probabilities, it is not specifically a means of estimating the probability or rate of failure modes of a particular software, that is, it is not technically a QSRМ. Accordingly the CSRМ approach is not evaluated against the set of QSRМ desirable characteristics.

N-version programming is a diversity strategy to reduce the likelihood of potential CCFs of software and is not a QSRМ. Accordingly, it is not evaluated against the set of QSRМ desirable characteristics. Nonetheless, it offers some interesting concepts for addressing software CCFs, as described here. N-version programming is accomplished by using different software

development teams to develop software using the same specifications and performing tests to determine if different versions fail together. N-version programming experiments, for example, Lyu [2005], demonstrated that it can improve software reliability, that is, the software do not fail together at all times. On the other hand, failures of the software are correlated, that is, the software does fail together in some cases. This points to the feasibility of using a beta factor to account for correlated software failures, similar to that used for modeling CCFs of hardware. Employing experimental results to determine a beta factor appears reasonable. However, different experiments seem to provide significantly different beta factors, which probably must be addressed by considering parameter uncertainty. In addition, the beta factor only would account for the benefits of N-version programming, not any other diversity measures.

2.3 Selected Candidate Methods

Based on the evaluation of Section 2.2, summarized in Table 2-1, it can be seen that none of the methods reviewed meets the complete set of desirable characteristics, and no single method clearly stands out as the most appropriate. Based on the insights from Section 2.2, the BBN method and the test-based methods were selected as candidates for further assessment, mainly because of the former's ability to account for the quality of the software lifecycle activities and the latter's use of standard statistical methods, including treatment of parameter uncertainties. These methods also appear to have the greatest potential for demonstrating high software reliability. (It should be noted that the Metrics-MTTF method, which also compares well to the characteristics, is a test-based method.) This finding is consistent with the results of the expert panel meeting on modeling software failures in PRA [Chu 2009b], where the panelists specifically identified testing and BBNs as general methods with potential for quantifying software failure rates and probabilities. One concern with the BBN method is that it requires converting qualitative information into failure probabilities while there may be no needed experts to accomplish this. A concern with the test-based methods is the availability of the equipment (hardware and software) needed to undertake the tests.

To advance the state of the art, the QSRMs were not considered solely as individual, stand-alone methods, that is, consideration was given to combining the best features of different methods. Based on the more detailed assessment of the candidate methods documented in this report, it was decided that the black-box statistical testing method is the preferred approach for the case study. The white-box method was not selected because it has not been well demonstrated, and extra effort is needed in its implementation with no obvious benefits, as pointed out earlier and further discussed in Section 6.1. In conjunction with the black-box method, a risk-informed testing profile will be explored to help account for the software context and limit the testing burden. Also, due to limitations of test-based methods, and to account for the quality of software lifecycle activities, it was decided to first develop a prior distribution via the BBN approach, and then use the black-box test-based method to generate the data needed for a Bayesian analysis. As stated in Section 1.3, since developing and quantifying a detailed BBN model can be very resource intensive, and the evidence needed for quantification may be lacking, both a simplified and detailed BBN model will be developed as part of the case study. This exercise will provide insight into the costs and benefits of developing a detailed model. It is expected that the prior distribution that results from both the simple and detailed BBN models will be fairly broad due to the high level of uncertainty associated with the use of expert opinion (particularly in cases where evidence is very limited). As such, the statistical testing results will also be used to perform a Bayesian update of a non-informative prior distribution following

existing guidance in the handbook of parameter estimation [Atwood 2002], to provide an additional standard for comparison.

Table 2-1 Evaluation of QSRLMs against desirable characteristics.

	1	2	3	4	5	6	7	8	9	10
SRGM	■	■	□	□	■	■	■	□	□	□
BBN	■	■	□	■	■	■	■	■	□	■
Test-based (black box)	■	■	■	□	■	■	■	■	□	■
Test-based (white box)	■	■	■	□	■	■	□	■	□	■
Frestimate	□	■	□	■	■	□	■	□	□	■
Metrics-MTTF	■	■	■	□	■	■	■	■	□	■
Metrics- Defect Density	■	■	■	□	■	□	□	■	□	■
Standard-based	■	□	□	■	□	□	□	□	□	□

Desirable Characteristics

1. The description of the method and its application is comprehensive and understandable.
2. The assumptions of the method have reasonable bases.
3. The method should allow for considering the specific operating conditions of the software.
4. The method considers the quality of lifecycle activities.
5. The method uses test results and operational experience.
6. The method addresses uncertainty.
7. The method was verified and validated for software reliability applications.
8. The method can demonstrate the high reliability of a safety-critical system (e.g., a failure on demand probability of $\sim 10^{-5}$, commensurate with an analog reactor protection system).
9. The method should be able to estimate parameters that can be used to account for software common-cause failures.
10. The data and needed information exist and can be collected.

Legend

■ **YES** – Method conceptually meets the characteristic and its implementation is deemed practical

■ **MAYBE** – Method may conceptually meet the characteristic and/or its implementation may not be practical

□ **NO** – Method does not conceptually meet the characteristic

SRGMs were also found to meet several of the desirable characteristics, as seen in Table 2-1. They employ information collected during software development, i.e., the results from debugging tests, and are the most often used software reliability methods. As part of this study, a discrete SRGM (DSRGM) was explored as an alternative to the BBN method for generating a prior distribution for software failure probability. Since this study only developed a DSRGM approach in parallel to the continuous approach in the literature, a separate specific evaluation of DSRGMs was not undertaken. The results of the comparison of the SRGM approach to the desirable characteristics are assumed to apply to the DSRGM approach. The most likely limiting factor in using a DSRGM is that data on demand-based debugging tests may not be available, particularly for highly reliable safety systems. Based on this and other limitations associated with this method, as discussed in Section 4, it was ultimately decided not to develop a prior distribution using the DSRGM approach.

Sections 4 to 6 describe how a QSRM can be developed for a digital protection system using each of the above 3 methods, keeping in mind the common limitations of current methods described in Section 2.1.

3. SELECTION OF AN EXAMPLE SYSTEM AND SYSTEM DESCRIPTION

3.1 Example System Identification and Familiarization

Since the objective of the study is to develop methods for quantifying the probability of failure on demand of digital protections systems at nuclear power plants (NPPs), it is preferable that the example system be such a system. In order to take into account the quality of software development activities using the Bayesian Belief Network (BBN) method, it is desirable to obtain information associated with the activities, for example, verification and validation reports. In addition, due to the desire to release this report publicly, it is preferable to use a system with publicly available documentation (recognizing that the study can be documented, if necessary, so as to omit or mask proprietary information). On the other hand, very few protection systems at U.S. NPPs have been digitally upgraded.

The Oconee Nuclear Station recently received regulatory approval from the Nuclear Regulatory Commission (NRC) for digital upgrades to its reactor protection system (RPS) and engineered safeguard protection system (ESPS) [NRC 2010] and the Wolf Creek Generating Station received NRC approval to replace its Main Steam and Feedwater Isolation System (MSFIS) controls [NRC 2009]. Systems such as these would be ideal example systems for the study of this report, because a large amount of design information, including development of application software, is potentially available. However, since many of the documents associated with these digital upgrades are proprietary, and due to expected difficulties in obtaining all of the required system and software information from the plant or vendor, particularly for systems for which the licensee is currently seeking regulatory approval, it was concluded that alternative systems would need to be pursued. Digital protection systems either currently installed or being planned at several foreign nuclear power plants were also considered, but it was concluded that the logistics of establishing the necessary international cooperation and potential difficulties in obtaining and using the necessary information, particularly if it is documented in a foreign language, were too prohibitive.

Due to the expected difficulties associated with obtaining the necessary information and data for an NPP protection system, the search for an example system was expanded to include systems at other types of facilities. Through the NRC, a contact with NASA was established to explore the feasibility of using a NASA system. However, as with the NPP protection systems, the proprietary nature of the supporting information and data for most NASA systems complicated this effort. BNL also explored the protection systems at BNL accelerator facilities, that is, the personnel access control systems. Due to a lack of detailed documentation for these systems, they were found to not be suitable.

Availability of system information became the most limiting factor in the search for an example system. The NRC contacted Idaho National Laboratory (INL) for assistance, and obtained INL agreement to supply the information for a control system of a test facility of the Advanced Test Reactor (ATR). Though it is a control system, it does generate a reactor trip signal when certain conditions occur at the test facility. In this sense, it can be considered a protection system, since it serves a protection function. The remainder of this section describes this example system in more detail.

Due to the difficulties in finding an example system for the case studies, the following system information did not become available until very late in the project. As such, the candidate quantitative software reliability methods were illustrated without the benefit of this information. Additional system details will be included as part of the documentation of the case studies.

3.2 System Description

The ATR, see Figure 3-1, is the third generation of test reactors built at the Reactor Technology Complex (RTC), located at INL, to study the effects of intense neutron and gamma radiation on reactor materials and fuels [Grover 2005]. As shown in Figure 3-2, the ATR core consists of 40 curved plate fuel elements in a serpentine arrangement around a 3 x 3 array of primary testing locations, including nine large high-intensity neutron flux traps. Five of the nine flux traps in the ATR are equipped with pressurized water loops, which are used for materials and fuels testing. While there are variations in the designs of the various loops, Figure 3-3 shows a typical loop cross section. Three concentric tubes form the piping assembly for each water loop in the ATR. The assembly penetrates the vessel's bottom closure plate and has an inlet and an outlet below the vessel. Coolant comes up through the innermost tube, the flow tube, and passes the sample. Near the top of the vessel, on four of the five loops, the coolant passes through positions in the flow tube into the annulus enclosed by the pressure tube and returns down that annulus to the outlet. On the fifth loop, the water passes only one way, up through the in-pile tube and out.

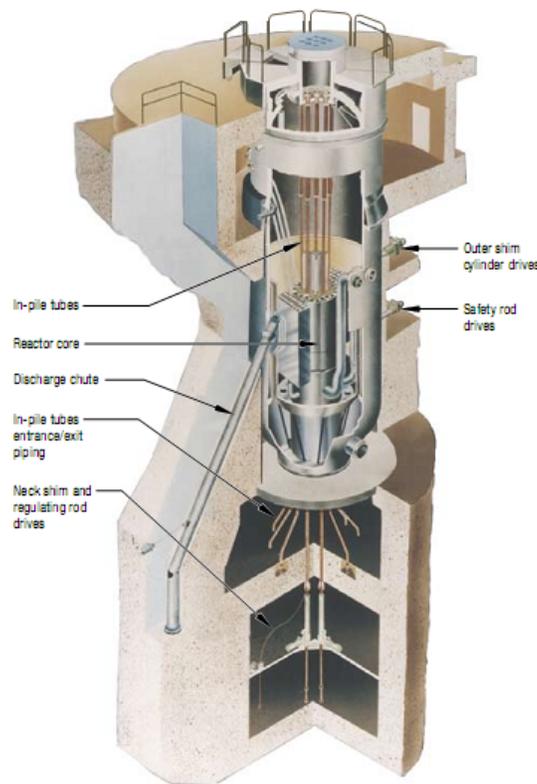


Figure 3-1 Advanced Test Reactor at Idaho National Laboratory

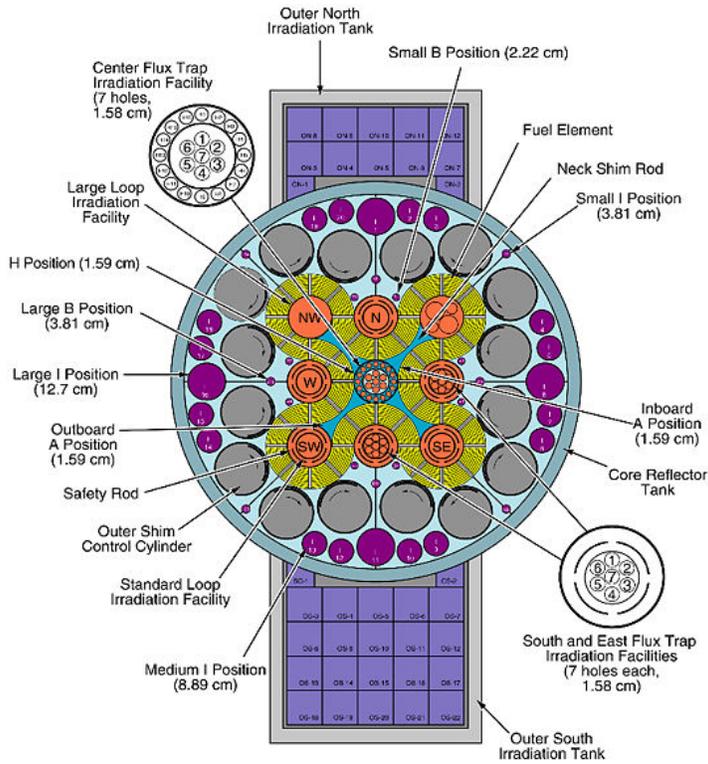


Figure 3-2 Core Cross Section of Advanced Test Reactor

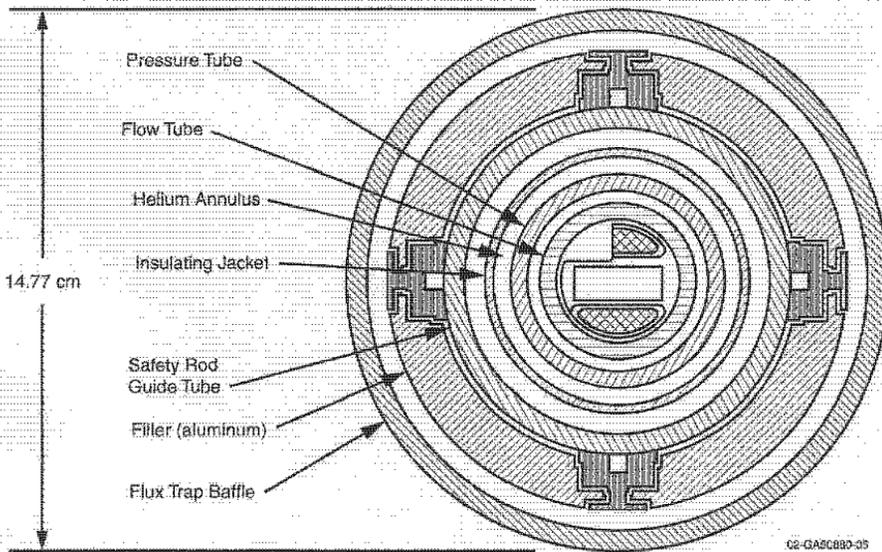


Figure 3-3 Cross Section of the In-core Portion of a Typical Pressurized Water Loop

Loop cubicles and equipment occupy the space around the reactor on two basement floors. The pressurized water loop equipment includes piping within the reactor vessel and pumps, heat exchangers, a pressurizer, and demineralizers within a shielded cubicle (Figure 3-4).

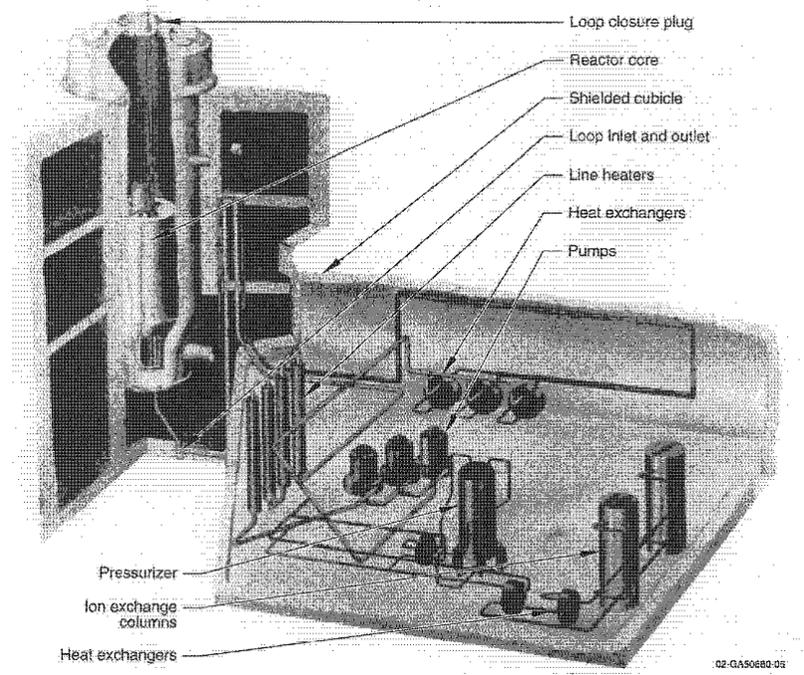


Figure 3-4 Pressurized Water Loop System

3.2.1 Design and Operation of ATR Loop Operating Control System

Five in-pile tubes (IPTs) installed within the ATR are supported by loop systems that are installed within shielded cubicles in one of the two basement levels of the ATR facility [INL 2008]. The designed loops are: 1C-W, 1D-N, 2B-SE, 2D-SW, and 2E-NW. The primary function of each loop system is to circulate water through IPTs at the proper pressure, temperature, and flow conditions for irradiation of experimental material and nuclear fuel specimens. The purpose of the Loop Operation Control System (LOCS) is to detect abnormal conditions within the IPTs and the supporting loop systems that can lead to possible experiment or hardware damage, to control the loop parameters per sponsor experiment requirements, and to provide loop equipment protective interlocks.

The major portion of the LOCS is a MAX 1000 Distributed Control System (DCS) that consists of five remote processing units and eight computer workstations. Each of the five remote processing units is used to control the equipment for a specific loop facility. The eight workstations are used by operations and engineering personnel to control and monitor the system. The communication between the eight workstations and the remote processing units is through a dual-directional fiber optic highway. The remote processing units contain input/output (I/O) modules, two pairs of redundant distributed processing units (DPUs), two pairs of redundant power supplies, and two optical-to-electrical interface (OEI) modules. The input

modules interface with the loop equipment to convert field signals to digital signals (for example, converting a 4-20 mA signal to a digital signal). The output modules convert digital signals to interface with loop equipment (for example, stopping and starting equipment). The two pairs of redundant DPUs, designated the A/B and C/D pair, interface with the I/O modules to operate the loop equipment at operator requested values. Two pairs of DPUs are necessary as a single DPU cannot process all the information necessary to control a loop facility. One of the two pairs of redundant power supplies are used to provide power for the electric equipment in the remote processing unit, transmitters, and to supply power to field electrical signal OEI units for communication over the fiber highway. The other pair of redundant powers supplies is used to provide power to field electrical equipment.

The LOCS controls the following equipment:

- A. Primary coolant pumps
- B. Loop line heaters
- C. Loop pressurizer heaters
- D. Makeup pumps
- E. Purification flow control valve
- F. Makeup system recirculation pump control
- G. Conductivity flow control,

and performs the following process functions:

- Primary Coolant Flow Control Function
- Primary Coolant Temperature Control Function
- Primary Coolant Pressure Control Function
- Pressurizer Level Control Function
- Loop Degassing Flow Control Function
- Ion Exchange Column Flow Control Function
- Makeup System Storage Tank Level Control Function.

The objective of the loop protective functions is to detect abnormal conditions and initiate required mitigating actions to preclude damage to the IPTs, experiments, and associated loop hardware. For each experiment IPT loop there are seven loop protective functions provided by this system:

1. IPT coolant temperature (inlet)
2. IPT coolant flow (inlet)
3. IPT coolant pressure (inlet)
4. Test specimen temperatures
5. IPT coolant temperature (outlet)
6. IPT coolant temperature (ΔT)
7. Loop coolant pump power supply.

IPT protection is provided by low IPT inlet flow, high IPT inlet temperature, high IPT outlet temperature, low IPT inlet pressure, high IPT pressure tube differential pressure for 1C-W, and low Maximum Use Capacity Holder (MUCH) experiment flow (when a MUCH experiment is installed). These functions will result in an alarm and a reactor scram. The MUCH experiment flow protective function can be disabled when a MUCH experiment is not installed. High specimen temperature and high IPT differential temperature protective actions will, upon reaching a predetermined condition, result in an alarm and either a reactor scram signal or a

reactor power setback signal which initiates insertion of the controlling regulating rod. These signals can be individually disabled as prescribed by experiment operating documentation. The pump power protective function has an alarm and scram (that is, no power setback option) or can be disabled.

Chapter 15 of the ATR SAR-153 [SAR-153] contains loop facility accident sequence analysis which estimated the sequence frequencies using selected protective functions of the LOCS.

3.2.2 Description of Software Development

This section summarizes the initial information obtained from INL on the development of the ATR Loop DCS software. This information supports development of the Bayesian Belief Network model which evaluates the quality of software development activities. The ATR Loop DCS software is commercial software provided by Metso Automation Max Controls, Inc. The software documentation stipulates that Metso Automation will have and maintain a quality program in compliance with ASME (American Society of Mechanical Engineers) NQA-1 (Nuclear Quality Assurance) [ASME NQA]. The quality level (QL) determination for the ATR loop DCS software was made per LWP-13014, "Determining Quality Levels," [LWP-13014] by the QL analyst and documented in RTC-0070, "ATR Distributed Control System Equipment - Excluding Human-Machine-Interface Equipment," [RTC-0070] as Quality Level 2. The software was determined to be non-safety, configurable software per LWP-13620 [PLN-2766].

The software-related documentation provided by INL does not contain information describing the software development activities of the recent upgrade of the ATR Loop DCS that was originally installed in 1993 [PLN-2776]. The software quality assurance plan of the LOCS upgrade is described in [PLN-2766]. The review activities of the software development cycle are briefly discussed in the documentation provided. Document [PLN-2768] illustrates the software verification and validation (V&V) plan of the LOCS upgrade project. A Configuration Management Plan is documented in [PLN-3137] and contains some information about the software development activities. Document [PG-T-94] details a previous upgrade project of the LOCS and defines the phases in the software development cycle. The information extracted from these documents is briefly summarized below as a high-level introduction of the software development activities for the upgraded LOCS. Note, information about guidance adopted in each development phase is not included in the documents provided so far. In some cases, the descriptions below reflect assumptions made about the activities completed in each phase for the ATR Loop DCS. This information will be confirmed or updated as more information on the software development activities of the Loop DCS upgrade is obtained. Standards for the design/development of the LOCS-DCS and CAP-DCS are found in LWP-10400 [LWP-10400], LWP-13620 [LWP-13620], and PLN-2293 [PLN-2293]. Additional standards that are applicable to design and development are provided in [PLN-3137].

1. Requirements Phase

The requirements phase is the period of time during which the requirements of functional and performance capabilities for a software product are defined and documented [PG-T-94]. At the beginning of the software development, requirements were included as a baseline⁵ as part of the design document, addressing both the technical and functional requirements of the software

⁵ A baseline is a specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.

application. The requirements specifically addressed operating system interface, performance, installation, design input, and design constraint issues [PG-T-94]. The requirements were used in conjunction with subsequent requirement and design documents to determine if the functionality that has been implemented satisfies the requirements. A traceability matrix of requirements was created and contained as part of the verification and validation plan [PLN-2766].

2. Design Phase

In the design phase, the designs for architecture, components, interfaces of the software were created, documented, and verified to satisfy the requirements [PG-T-94].

3. Implementation Phase

During this time period, the software product was created and debugged [PG-T-94].

4. Test Phase

In test phase, the components of a software produce were evaluated and integrated, and the software product was evaluated to determine whether or not all of the requirements have been satisfied [PG-T-94].

For the ATR, Loop DCS V&V was accomplished by performing a series of tests including the factory acceptance test (FAT), the grooming test (GT), and the system operability (SO) test [PLN-2768].

Factory acceptance testing (FAC) occurs at the end of the software development cycle. It is used to provide confidence that the software satisfies the general requirements. For the ATR Loop DCS software, the FAC was performed at Metso and witnessed by the Battelle Energy Alliance (BEA) after BEA approved the test procedure. In general, the FAC environment should be comparable to the environment where the equipment will be installed and used.

Prior to system operability (SO) testing, "grooming" tests were performed by the development engineers to evaluate the fully integrated system in the field (the ATR site), and make the appropriate modifications to ensure the implemented installation meets the engineering design requirements. This phase was performed by thoroughly testing the completed loops after the system was installed at the ATR facility and connected to the experiment loop instrumentation. As part of the documentation of the grooming tests, logs were created to record deficiencies and corrections and reviewed to ensure that the modifications made during the grooming tests also satisfy the requirements. If the system does not pass the grooming tests, the errors must be corrected and the grooming test process repeated to obtain a corrected baseline release.

The purpose of SO testing, which is performed at the site, is to gain confidence that the software functions correctly in its operating environment. The SO test is a stand-alone controlled system test procedure prepared, approved, and performed by ATR Operations. Both functional and operational tests were undertaken to thoroughly exercise and verify the system functions. In those cases where the SO test was not passed, the errors were corrected and verified within the scope of the SO test work order and documented. If the errors could not be corrected and verified within the scope of the work order, then the previous DCS baseline was restored and the V&V process repeated for the corrected baseline release [PLN-2768].

5. Installation and Checkout

During this time period, the software product was integrated into its operational environment and tested in the environment to ensure that it performs as required.

4. DEVELOPMENT OF A FAILURE-ON-DEMAND-BASED SOFTWARE RELIABILITY GROWTH METHOD

4.1 Introduction

In NPP PRA studies, the reliability of a control system and a protection system in performing their designed functions are typically characterized by their failure rate and demand failure probability, respectively. As discussed in Section 1, this study is mainly interested in demand failures of protection systems. The continuous-time non-homogenous Poisson process (NHPP) based software reliability growth models (SRGMs) [Chu 2010] can be directly used to estimate the failure rate of control system software using the test results that are usually recorded in terms of the number failure occurrences within a time period (software execution time). However, software testing of protection systems is often performed by counting the number of faults triggered and repairing the triggered faults at the end of each test run. At the end of each trial, the manifested failure(s) are fixed by removing the fault(s) that caused these failures, i.e., each failure can only occur once. SRGMs assume that the failure probability of the software decreases with the number of test runs due to the decrease in the number of faults remaining in the software. The debugging data is usually recorded in the format of the number of successful runs between failures or the number of failures in a certain number of runs. A continuous-time SRGM cannot be directly applied or at least has to be tailored to cope with such testing results.

On the other hand, discrete models (e.g., Yamada [1985]) are expected to be more straightforward for such applications. A discrete non-homogenous Poisson process (D-NHPP) can naturally model the fault manifest/removal process in safety related software testing. A few discrete time SRGMs have been developed (see e.g., [Shanthikumar 1983, Yamada 1985, Satoh 2001]). Similar to development of the continuous SRGMs that calculate the software failure rate, the result obtained from a discrete SRGM (DSRGM) is the failure probability per demand. In this study, an analytical framework is established for estimating the demand failure probability using DSRGMs based on the previous studies, which is best illustrated by the description in Section 4.2 about the parallel development of basic SRGMs in both continuous and discrete time domains. One possible way to account for modeling uncertainty, would be to use a number of DSRGMs to capture the variability of software reliability among different SRGM models based on the same set of debugging data. A set of point estimates of the predicted failure probabilities can be obtained to generate a prior distribution of the software demand failure probability. The details of this approach can be found in Appendix A.

The D-NHPP approach developed here is not recommended to estimate the failure probability of safety-related system software mainly due to the unavailability of the needed failure data. As discussed in Section 4.3, it was decided that the DSRGM approach would not be further pursued for the case study. This, however, does not prevent its application to a piece of software that does not have a very high reliability requirement, or for applications that do not involve failure probability estimation.

4.2 Discrete NHPP for Demand Failure Probability

In this study, the D-NHPP is formally introduced and used to define software demand failure probability. Appendix A provides the details. A cumulative Bernoulli trial (CBT) process [Hoare 1983] is used to model the software debugging process in terms of detecting and removing faults at the end of each test run. It is shown that such a CBT model is characterized by a D-NHPP if the initial number of faults is of a Poisson distribution. For many continuous-time SRGMs, it is straightforward to obtain their discrete counterparts via discretization. The advantage of doing this is that a spectrum of continuous-time SRGMs and the associated development, including how to select the best models according to their prediction accuracy (e.g., see Chapter 4 of Lyu [1996]), which does not exist for discrete models, can be completely transplanted to the discrete-time domain. More insights about the D-NHPP framework can be obtained by inspecting the parallel development of the basic continuous-time SGRMs [Musa 1987] and the discrete SRGMs, as discussed below.

A homogeneous Poisson process is a counting process with a constant occurrence rate. For failure processes with non-constant failure rates, a non-homogeneous Poisson process is applicable, as assumed in the continuous-time SRGMs. A discrete NHPP is a discrete counterpart of the continuous NHPP for the purpose of accommodating the debugging data (e.g., of an RPS software) recorded in the format of the number of successful runs between failures or the number of failures in a certain number of runs. A continuous or discrete NHPP is characterized by the expected number of failures (mean value function) observed. Note, how the expected number of failures changes as the testing progresses is defined by individual SRGMs based on dissimilar assumptions about fault detection and removal, for example, the assumption that failure occurrence rate is proportional to the remaining fault content in the continuous time Goel and Okumoto model [Goel 1979]. As a result, in the continuous time domain, the change in the expected failure number (or failure rate decrease) is governed by assumption-based differential equations. In general, each continuous time model can be discretized and represented by a difference equation. The expected failure number can be obtained by solving the difference equation and used in the D-NHPP model to obtain the probability mass function.

A clear picture of parallel paths in deriving the basic continuous and discrete NHPP models that are based on the following assumptions can be seen in Figure 4-1:

- Assumption A1: Software faults occur at independent and identically distributed random times (i.e., all random variables are mutually independent and each random variable has the same probability distribution as the others).
- Assumption A2: The initial number of software faults is finite, implying that the total number of failure occurrences is also finite.
- Assumption A3: The initial number of software faults is of a Poisson distribution.

In the continuous time domain, the software debugging process can be described by an NHPP with the number of failures in a time interval following a Poisson distribution; while in the discrete time domain, a Bernoulli trial process is suitable for modeling the outcomes of test runs of the software (i.e., a success or failure), and the number of experienced (and removed) failures after a certain number of test runs should obey a binomial distribution. Referring to the dashed rectangle on the left in Figure 4-1, Okamura [2004] demonstrated that representing the debugging process with a cumulative Bernoulli trial process or a binomial process (Box D1)

leads to a discrete NHPP model, in parallel to the Poisson process representation of software debugging (in continuous time domain) (Box C1) leading to a continuous time NHPP model.

Another parallel path between the continuous and discrete time NHPPs can also be observed in the dashed rectangle on the right in Figure 4-1. In continuous time domain, the NHPP model (the Poisson type model) obtained based on certain assumptions [Musa 1987] can be equivalently derived by using a binomial distribution to represent the number of failures occurred in a time interval (the binomial type model) under the assumption of a Poisson distribution for the initial number of faults in the software (Box C2). A discrete time NHPP model can be derived in exactly the same manner [Okamura 2004], i.e., assuming the number of failures in a number of tests is binomial distributed, and the only difference is that the failure probability at a specific run is used in the derivation instead of hazard rate or failure rate at a specific time of the testing (Box D2).

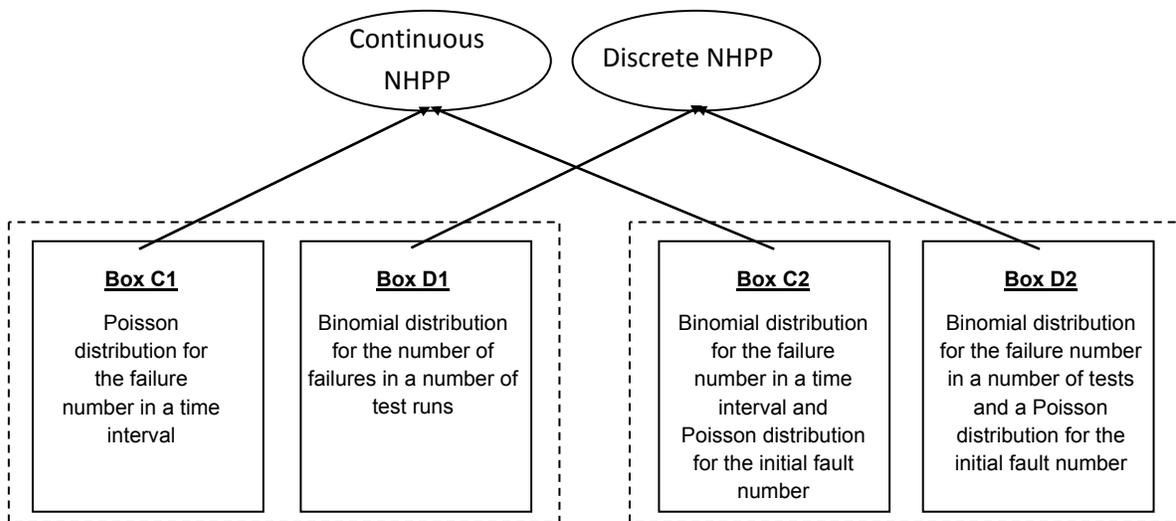


Figure 4-1: Parallel Development of Continuous and Discrete NHPP Models

The assumptions were adopted here only for the purpose of deriving the basic discrete SRGM models as the counterparts of the basic continuous time SRGMs [Musa 1987]. This does not mean that these basic discrete models are preferred or endorsed to be used. In fact, different assumptions have been used to obtain a variety of continuous-time SRGMs [Chu 2010], with some of them being more realistic (e.g., non-equal contribution of faults to the software failure occurrence for Duane's model or the model presented in [Littlewood 1981]). Discretization of such SRGMs can be performed to acquire their discrete counterparts.

A straightforward way of generating discrete SRGMs is to replace the differential equations characterizing the continuous SGRM with difference equations. The probability mass function (pmf) of a discrete NHPP model can then be obtained by solving the resulting difference equations for the mean value function of the accumulated number of failures [Okamura 2004]. A difference equation becomes a differential one when the time interval tends to zero, and therefore, preserves the characteristics of the differential equation.

A more straightforward way of discretizing the continuous-time SGRMs is to simply replace the time with the number of runs [Littlewood 2011] because for a safety-related software the probability of failure per demand is small, so a Bernoulli process will closely resemble a Poisson process. The advantage of doing this is that a spectrum of continuous-time SRGMs and the associated development, including how to select the best models according to their prediction accuracy, which does not exist for discrete models, can be completely transplanted to the discrete-time domain.

Standard parameter estimation methods such as maximum likelihood estimation or least-square estimation have been used in NHPP models and also can be used to estimate the parameters in a D-NHPP model and parameter uncertainties, as discussed in Appendix A. Alternatively, a Bayesian approach can be used, i.e., to perform a Bayesian update on the pre-selected priors of the parameters based on the new observation data. By using the Bayesian approach the uncertainty as well as the point estimate of parameters can be estimated directly. A probability mass function of a DSRGM can be used to predict the failure probability per demand after the software is released (See Appendix A for details).

4.3 Major Issues with DSRGM Applications

In order to predict software reliability using an SRGM (or DSRGM), debugging data based on execution of the software with operationally representative test inputs are needed (e.g., for an RPS software, such inputs are generated from the NPP responses to individual PRA initiating events that will cause the RPS to generate a reactor trip signal). It is also recognized that a large number of failure records in the debugging data are needed in order to demonstrate the reliability growth of the software as debugging progresses. The major issues, which drastically limit the application of SRGMs to safety-related software, related to the two requirements are elaborated below.

- (1) Software vendors are unlikely to perform debugging using operationally representative inputs. Therefore, the software test profile will not match its operational profile. In addition, some test inputs may be unrealistic or even represent conditions that are not possible during NPP operation. Such testing results cannot be used to predict software reliability using an SRGM or any other software reliability quantification method.
- (2) If the vendors perform testing using operational profile generated inputs (at least partially), the documentation of the results may not relate the test outcomes to the inputs caused by specific initiating events, or may mix the test outcomes of operationally representative inputs and other inputs. In such situations, it will not be possible to estimate context-specific (e.g., initiating-event-specific) software failure probabilities.
- (3) Even if the vendors perform the testing and document the results appropriately, there may not be sufficient success records for exercising SRGMs to show the anticipated reliability growth. In addition, if there are a few failure records in the debugging data, the predicted reliability by using SRGMs will not be very high, which may not be acceptable for a safety-critical system such as an RPS.

The concerns expressed above pose significant limitations on application of SRGM methods. As such SRGMs (including DSRGMs) are not considered suitable for estimating software failure probabilities, and will not be pursued as part of the case study.

5. DEVELOPMENT OF A BBN MODEL TAKING INTO CONSIDERATION QUALITY OF SOFTWARE DEVELOPMENT ACTIVITIES AND ASSOCIATED GUIDANCE

The Bayesian belief networks (BBN) method has been a topic of research in many different areas, for example, medical diagnosis [Lauritzen 1988], risk assessment of natural hazards [Regamey 2006, Bensi 2010], and in supporting tsunami-warning decisions [Blaster 2009]. Lauritzen [1988] provided an example causal network using probabilistic reasoning in a prototype expert system, MUNIN (MUScle and Nerve Inference Network) by structuring the medical knowledge needed for diagnosing neurological disease. In [Regamey 2006], a BBN model is developed for assessing avalanche risk using an avalanche model and a geographical information system. An example of the commercial employment of the BBN method is its usage in the Office Assistant of Microsoft Office '97 [Horvits 1998].

Some experts consider the BBN method to be a promising way for quantifying software reliability [Johnson 2000, Littlewood 2000, Dahll 2007, Eom 2009]. However, only in a few studies by the Halden Reactor Project, the VTT Technical Research Center of Finland, and the Finnish government authority for the nuclear industry (STUK) [Helminen 2001, 2003a, 2003b, 2005, and 2007, Gran 2000, 2002a and b] has a BBN been used to quantify software reliability. A recent study by the Korea Atomic Energy Research Institute (KAERI) [Eom 2009] developed an approach for estimating the number of software faults from the quality of software development activities, without quantifying software failure probabilities. In this section, possible methods are described for quantifying software-failure probabilities using BBNs. The issues and limitations of the approaches are discussed. The proposed BBN approach would generate a prior distribution for the probability of software failure of a protection system that could be further Bayesian-updated using the results of the statistical testing described in Section 6. In general, the statistical testing can be considered a part of the overall BBN approach, because the Bayesian updating can be included as nodes in the BBN model. In this study, BBN and statistical testing methods are discussed in separate sections because they can be considered distinct methods.

5.1 Introduction

An earlier BNL study [Chu 2010], gave an introduction to the BBN method, and described and reviewed some BBN studies undertaken by a few organizations related to the nuclear industry. A BBN model is a special mathematical representation, in terms of nodes and directed arcs, of an underlying model and its associated data. The dependencies between parent and child nodes and the probabilities of root nodes⁶ might be based on theoretical models, empirical relations, expert opinion, engineering judgment, or any mixture of them. Based on the BBN models reviewed in the BNL study, developing BBN models for quantifying software reliability essentially involves evaluating the quality of software development activities against the standards the development must follow and other factors that may affect software reliability, and then quantifying the impacts of the quality of software development and other factors on the software failure rate or failure probability. The results of the BBN model can then be used as the prior distribution of the software failure rate or probability, and a Bayesian update of the prior

⁶ A parent node has an arc pointing to its child node(s), and a node without a parent is a root node.

can be performed using testing/operating data to obtain a posterior distribution of the software's reliability. As mentioned previously, the Bayesian update process can be incorporated directly into the BBN model or can be applied as a separate, follow-on analysis.

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 2002a and 2002b, Eom 2004 and 2009]), and include parameter uncertainties as a part of the modeling. However, there are challenges in developing a BBN that takes full advantage of these capabilities, that is, the substantial development effort needed, the expertise of the BBN developers, the qualification of 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 the quality of software development on its reliability) must be quantified. Since there may be insufficient data to "anchor" the conversion of qualitative information to quantitative values, the resulting uncertainty in the quantitative data from the experts may be very large, so that it is difficult to demonstrate the small failure probabilities often associated with safety-related systems. Overconfidence is one of the most common (and potentially severe) problems in expert judgment [Lin 2008]. For example, in the BBN model developed for a helicopter-location identification system by the Halden Reactor Project [Gran 2002a], experts had to estimate the distributions of the software failure probability conditional on the Quality of Product, Solution Complexity, and Quality of Analysis. The accuracy with which experts can make such estimations is highly uncertain. No benchmark studies have been performed to validate the results of the expert elicitation for converting qualitative evidence to software reliability, which is the biggest challenge in BBN development. Due to the weakness in this kind of expert elicitation, in a study performed by KAERI [Eom 2009], a BBN was used to estimate the numbers of faults injected, and removed, in each stage of software development, based on the quality of carrying out the associated development; no attempts were made to estimate the probabilities of software failure.

In this study, the above two BBN methods that assessed the quality of software development activities are summarized and additional published information was collected and used in proposing possible enhancements in the approaches such that they might possibly be applied to modeling protection systems of a nuclear power plant. Section 5.2 describes an approach for developing a BBN model to quantify the probability of failure on demand, with its associated issues and limitations. Section 5.3 details the steps in implementing the approach. Section 5.4 summarizes the evaluation of the BBN approach against the desirable characteristics developed in Section 2, in terms of the approach's potential in better addressing these characteristics.

5.2 Development of a BBN Model

5.2.1 Development of a BBN that Captures the Quality of Software Development Activities

Guidance on how to carry out software development activities is available in different applications. In the case of the helicopter-location identification system [Gran 2002a], the avionic standard DO-178B [RTCA 1999] was used, and the BBN model considered the 10 lifecycle stages it defines. For the Korea Nuclear Instrumentation and Control System (KNICS) [Eom 2004 and 2009], the guidance was the KNICS procedure, developed with information from NRC Branch Technical Position (BTP) 14 [USNRC 2007]. The BBN models in both studies are

examples of how to formulate a BBN to evaluate the quality of software development activities. In general, the qualitative requirements in the standards are used to define the nodes in the BBN models which represent the quality of software development activities [Gran 2002a and Eom 2009], and the nodes are used as factors that affect the addition, detection and removal of faults which, in turn, determine the number of faults remaining in the system [Eom 2009]. Since guidance, such as BTP-14, is used in licensing, the vendors, the nuclear power plants, and the NRC need to demonstrate and verify that the relevant guidance is followed. Therefore, useful information should be available from these associated evaluations, for example, verification and validation reports. Further, since the people involved in the evaluations are familiar with the specific digital system and the guidance, they potentially can serve as experts in developing the BBN model and participate in expert elicitation in quantifying node probability tables (NPTs).

The next two subsections describe two methods of obtaining the software failure probability, depending upon the information provided in the BBN models. Implementing them is a very important part of the entire BBN model for the specific system being assessed.

5.2.2 Conversion of Number of Faults to Software Failure Probability

In the BBN studies [Eom 2004 and 2009] of the Korea Atomic Energy Research Institute (KAERI) there was no attempt to compute the failure rate or probability of the RPS software. In particular, the Eom [2009] study focused on using the BBN to estimate the remaining number of faults at the end of the software life-cycle by modeling the mechanisms of fault insertion at each development phase, and of fault removal in the current and the previous phases. This approach is similar to that of Fenton [2007]. The number of inserted and removed faults is affected by the quality of software development work, which is estimated by comparing the development activities against the standards used in the development cycle. In addition, some data on the number of faults detected at each development phase were available, and were used in this study to support estimations of the numbers of undetected ones. Further, KAERI adopted the waterfall model of software development, and the interconnection of the previous phase BBN to the current one passes undetected faults from phase to phase. KAERI did not explore how to calculate the software reliability using the BBN.

A few studies/methods in the literature convert the number of faults in a program into software failure rates or probabilities. They are summarized below:

- (1) Delic et al. [1997] developed a simple BBN model for estimating the probability of failure-on-demand of a software program by introducing a “size distribution” of faults, and the number of estimated faults multiplied by this size distribution produces the distribution of the probability of such failure. The size distribution was estimated from experience with similar software-development projects of the same vendor. A difficulty here is that vendor-specific data must be available; the size distribution developed using data from one vendor’s software cannot be used for that of others.
- (2) Musa [1987] introduced the concept of the fault-exposure ratio that effectively converts the number of faults into software failure rates. This might be considered as a less detailed version of Delic’s model [1997] by integrating over the size distribution of the faults. The same idea was used in more recent studies/methods, i.e., Smidts [2004] in her metrics-based methods and Neufelder [2002] in her “Frestimate” method. A concern with such methods is that software projects differ from each other in many ways (e.g., in terms of complexity), and the approach does not account for these differences (i.e., it assumes that

two programs with the same estimated number of faults have the same failure rate/probability). In that sense, the differences between the software are "averaged out," and assumed to be reflected solely by the number of faults. It is questionable whether such a simplified relationship adequately estimates the failure probability, since it does not account for variations in many other factors that can affect this probability (e.g., the software's complexity). Another concern is the general applicability to current software of the factors or formulas derived with information from outdated software, as software development has changed over time (e.g., automated code-generation is now prevalent for digital systems).

Since vendor/software-specific data are more applicable, they are preferred over the generic fault-exposure ratio.

A more detailed conceptual model for converting the estimated number of software faults into a software-failure probability can be developed by considering faults to be distributed in different paths or parts of software (similar to the defect density method in [Smidts 2004]). This model assumes the following: (1) Different challenges to the software (e.g., different initiating events requiring a reactor trip) may follow different paths or parts of the software, and (2) the estimated number of faults are distributed among the different paths/parts according to some rules, for example, the complexity of the paths or parts (the faults may also be evenly distributed over them). Given a demand to the system, the path or part of the software first must be determined with an estimated number of faults for the path or part. Then, the path/part specific failure probability is estimated using the concept of (1) fault size distribution obtained from a specific vendor (preferred, if available) [Delic 1997], or (2) fault exposure ratio obtained from various software projects (if vendor/software specific data is unavailable) [Musa 1987].

This conceptual model focuses on fault(s) located in the code for the protective logic, the major concern that may fail the RPS. Commonly, a safety-critical software, such as an RPS software, also performs several auxiliary functions, e.g., for display and interface. The possibility may not be ruled out that a fault in the code related to these functions, once triggered, could cause the RPS software to fail. The impact of auxiliary function failure on the entire software must be evaluated for a complete reliability assessment of the software.

5.2.3 Direct Estimation of Software Failure Probability Using Expert Elicitation

A few studies that estimated software failure probability/rate using expert elicitation were published in a series of papers and reports [Helminen 2001, 2003a, 2003b, 2005, and 2007] and [Gran 2000, Gran 2002a and 2002b]. In particular, Gran [2002a and b] developed models accounting for the quality of software-development activities and used the results of the quality evaluation in estimating software failure probability by expert elicitation. He used expert elicitations to build BBN models, including node probability tables, to convert qualitative- to quantitative-evidence, and to collect evidence for inferences [Gran 2002a and 2002b]. After collecting evidence of a specific application software development, a prior distribution of the software failure probability can be calculated and then updated via testing and/or operational data. An important part of these studies includes specifying the qualifications of the experts needed in different elicitations.

As discussed in Section 5.1, the accuracy with which experts can make estimations of software failure probability is highly uncertain; no benchmark studies have been undertaken to validate the findings from expert elicitation. On the other hand, directly estimating failure probability by

expert elicitation is somewhat similar to what is done in human reliability analysis (HRA) (e.g., the failure likelihood index method (FLIM) used in [Chu 1995]), in which factors that affect operator performance (Performance-Shaping Factors) are used in estimating human failure probabilities, even though the two approaches have different mathematical formats. Similarly, the correlation method of “Frestimate” [Neufelder 2002] converts qualitative information into failure rates. It uses proprietary data on past software projects in a regression analysis to assess the number of faults left in a software program. Regression analysis is another mathematical representation relating qualitative- to quantitative-information. The idea might be extended to directly estimating software failure probability. The limitations here are the lack of data on past software projects (the data of “Frestimate” are proprietary) and the method’s applicability to specific software.

In recent human reliability studies, for example, [Boring 2010], benchmark studies were undertaken using data from simulator training of operators to compare HRA methods. The same idea might serve to validate software reliability assessments, that is, by either collecting operating experience of software projects, or by applying statistical tests. It should be noted that HRA research progressed for many years before reaching the point of performing benchmarking exercises and still no HRA methods have been validated (they have only been compared). Therefore, it is expected that similar exercises on software reliability analysis will not occur until the state-of-the-art is further advanced and becomes more commonly accepted in the technical community.

5.2.4 Summary Discussion of Assumptions and Issues

The BBN method offers the advantage of supporting the combination of disparate information, such as quality of software-development activities and statistical test data. However, this also is a disadvantage because as yet there is no established route for converting the qualitative information obtained from assessing software development activities into software reliability (i.e., probability of failure on demand). This section discusses some of the key assumptions and issues associated with using the BBN approach for estimating software reliability. The assumptions and issues relate to (1) converting qualitative information to quantitative estimates, (2) the BBN structure, and (3) other potential limitations of the approach.

Converting Qualitative Information to Quantitative Estimates

1. The existing methods for converting the qualitative information obtained from assessing software development activities into software reliability (e.g., see Gran [2002a] and [Delic 1997]) have severe limitations/difficulties essentially because of lack of data relating quality of software development and the number of faults to reliability. Converting the number of faults to the software failure rate via the fault exposure ratio [Musa 1987] assumes a very simple relationship between the number of faults and software reliability and depends on very old software projects, calling into question its general applicability. Also, employing the “fault size distribution” [Delic 1997] is limited to software developed for a particular vendor that has operational data from similar software projects. This problem parallels the dearth of data for HRA. In light of recent benchmark analyses of human reliability (e.g., [Boring 2010]), it would be beneficial to perform benchmark exercises, as mentioned previously, to validate or calibrate a BBN model by collecting operational data on past software-development projects and/or performing the statistical testing described in Section 6.

Currently, there is little publicly available information on operational experience for nuclear power plant protection system software. It is expected that vendors will have collections of such information on the software they developed. However, most likely these data are proprietary. If adequate operating experience of a software program of a protection system and the associated detailed information needed for BBN modeling are available, a BBN model could be evaluated against the software's performance so that the parameters and structure could be tuned and the model validated. The benchmarking can be undertaken at the system level and the level of the node probability tables. However, based on previous experience with HRA, it may require completion of many such studies before confidence is developed in applying the BBN modeling method to estimate probability distributions for software failure.

The difficulties in carrying out benchmark studies should not be under-estimated: (1) The scarcity and/or unavailability of supporting data to characterize the relationship between the qualitative features of the development work and the quantities (e.g., the number of remaining faults or operational data on demand failures) that are suitable for quantifying software reliability, and (2) the applicability of the data available if it were not carefully collected and classified to reflect the differences (mainly in quality) between software projects. To overcome the first difficulty, statistical testing data (discussed in Section 6) can serve as supporting data, in addition to any operating experience. Furthermore, the quality of the software must be assessed rigorously and documented, thereby linking the quality information to the quantities needed for calculating software reliability. A planned case study on the example system using statistical testing will provide the information needed to determine if the BBN model's prediction is consistent with the statistical testing results. To establish the applicability of the available data necessitates systematically and consistently evaluating different software projects (mainly in terms of software quality) and categorizing their collected data accordingly.

The BBN method is a mathematical method that uses a BBN to represent or model the causal relationships of random variables. Similarly, a probabilistic risk assessment (PRA) uses a logic model to represent the logical relationships among basic events and top events, and the validity of the logical relationships and the associated data are very important. The usefulness of the BBN method depends upon the availability of an underlying model on the causal relationships among the variables and associated data. As such, employing a BBN for estimating software reliability differs from those BBN applications in other areas discussed earlier in this section. For estimating software reliability, both the underlying model and data are lacking, so they need to be developed as part of the BBN model. For the other applications discussed earlier, each had at least an underlying model or data, so their BBN models were used only as a mathematical representation of the model and/or data.

2. The BBN method of Gran [2002a] is similar to the failure-likelihood index method of human reliability analysis in that both convert qualitative information into probabilities when actual data are sparse, though they differ in their mathematical formats. In addition, the correlation method of "Frestimate" [Neufelder 2002] uses proprietary data on past software projects in a regression analysis to estimate the number of faults remaining in a software program. Regression analysis is another mathematical representation relating qualitative- to quantitative-information. It represents a more general approach of using past software development practices and the associated operational experience. However, the regression analysis used by Neufelder [2002] was limited to estimating the number of remaining faults

in the software, not to software failure rates and probabilities, and her use of fault exposure ratio suffers from the limitations discussed previously. Also, the information on past software development projects used by Neufelder is proprietary, and the projects may be outdated. Conceptually, the idea of regression analysis can be extended to directly estimating software failure probability. Again, this idea is limited by lack of data on previous software projects (the data of “Frestimate” are proprietary) and its applicability.

BBN Structure

3. To capture the quality of software-development activities, it is desirable to develop a BBN model using the structure of a standard on software development, e.g., BTP-14 [NRC 2007] for U.S. NPPs. The existing BBN models of Gran [2002a] and Eom [2009] use the standard DO-178B [RTCA 1999] and the KNICS procedure, respectively; they convert the requirements in the standards into nodes of the BBN models. Consequently, the BBN models are very complex, with many node probability tables having to be estimated. In general, the possible states of the nodes and the relationships between them should be defined precisely, and guidance given on how to estimate the node probability tables to maintain consistency. Bloomfield [2002] offered a criticism of the detailed models suggesting that they require a detailed assessment where potentially little data are available. However, since the standards are used to decide whether a piece of software is acceptable, people involved in approving the system (i.e., the staff of the plant, vendor, and regulator) should know how the software being evaluated meets the requirements of the standard, even though they will not necessarily know its reliability.

Alternatively, it would be simpler to develop a higher level model, for example, representing each stage of the software-development cycle by only one or two nodes. Further exploration is needed to demonstrate how to establish the relationship between the simplified model and the associated standard, and to determine the feasibility of quantifying such a model.

4. A BBN model can be built based on the causal relationship of the relevant events in a specific application; the directions of the arrows represent the influence of the parent nodes on the child nodes. If a single child node has multiple parent nodes, a high dimensional NPT or conditional probability density function must be formulated to characterize the impacts of the many parent nodes on the child node. While generally it is not easy to formulate any NPTs, it is even more difficult building high-dimension NPTs or distributions due to the need for carefully considering the subtle interplays between the parent nodes on their impacts on the child nodes. A typical case in a software development cycle can be evaluating the quality of inspection (a high level node) [Eom 2009] that may be affected by factors such as the completeness, correctness, and traceability of inspections (the low level nodes). In all of the studies mentioned above, the arrows pointed from the high-level node to the low level nodes in the BBN. That is, a one-dimensional NPT table is used for each lower level node to represent the influence the higher level node has on it. This makes building the BBN NPTs easier comparing with the case in which arrows are pointing from the lower level nodes to the higher level node, which requires use of a multi-dimensional NPT that has to consider all combinations of lower level states in assessing the NPT. However, the causality can be more naturally represented by the latter, and it would provide a more detailed model. Therefore, it is desirable that the two types of modeling be compared using simple examples to further evaluate their differences.

Other Potential Limitations

5. Unexpected or controversial results may surface in applying the collected software evidences to a BBN model [Littlewood 2007]. That is, the parameters and structure of a BBN may require revisiting and possibly revising when counter-intuitive findings emerge.
6. Two types of evidence might be collected for a BBN model; hard evidence representing a realization of a particular state of a node, and soft evidence representing the discrete probability distribution of a node. Jenson [2002] terms soft evidence likelihood evidence, and indicates that it is not clear what the evidence means. Soft evidence is likely to contradict the BBN model (i.e., it is very unlikely that an expert will provide a distribution for a node that exactly matches the one obtained through exercising the BBN model). On the other hand, some analysts [Pan 2006] use the Jeffrey's rule that essentially uses the discrete probability distribution (soft evidence) as the weight in calculating a weighted average of the probabilities of the nodes of the BBN, which may be a reasonable approach. Additional research is needed to understand the meaningfulness and value of using soft evidence.

5.2.5 Treatment of Uncertainty

Parameter uncertainties are intrinsic to the distributions representing the nodes and node-probability tables; this can be considered an advantage of the BBN method. However, assessing the distributions requires the availability of data or of knowledgeable experts, which may be an inherent limitation in applying BBN methods to quantifying software reliability. This is exemplified by the issue of converting qualitative information on software-development activities into software failure rates/probabilities. Regarding the node probability tables that assess the quality of software development activities, it is desirable to use software engineering experts to develop guidance based on engineering considerations. Regarding the node probability table that converts the qualitative information into software failure probabilities, collecting the experience from applicable software development projects would be useful, but very difficult.

Drouin [2009] states that, in general, modeling uncertainties are addressed by determining the sensitivity of the PRA results to different assumptions or models. For a BBN, modeling uncertainties can be addressed in a similar manner. For example, different expert teams can work independently in expert elicitation on node probability tables. To consider the effect of the complexity of BBN models on the results, two BBN models can be developed, a simpler one and a more complex one. As discussed earlier, benchmark studies can validate or invalidate the BBN model. That is, for this purpose, information on software-development activities and any available operational data from past software development projects can be incorporated in applying the BBN model. The benchmark exercises can be completed at different levels of detail of the BBN model, for example, at the overall level and the level of the node-probability tables. Alternative methods also can be developed to afford comparisons with the BBN method. For example, it might be worthwhile to try an extension of the failure likelihood method of human reliability analysis to assess software reliability because it uses similar information to that in the BBN method, and may be understood more easily by the PRA community. The correlation method of "Frestimate" can be extended to estimating software failure probability if there are data from earlier projects developing protection-system software.

Implementing any of the above approaches for addressing modeling uncertainty may entail significant difficulties, particularly in terms of level of effort and availability of information.

Consideration will be given to which, if any, of these approaches are practical for the planned case studies.

5.3 Procedure for Developing a BBN Model of an Example System

In this section, a process is proposed for developing a BBN model for an example system. The basic structure of the model, if developed appropriately, should be usable for any NPP digital protection system. To apply the model to a specific software, the evidence needs to be collected for that software to specialize the model. The proposed process for developing the BBN model for the example system involves the following steps.

(1) System familiarization

Besides getting an in-depth understanding of the design of the example system, its functions, and the systems that it interacts with, it is very important to understand the software-development work and the associated guidance. The needed documentation includes information on system design and operation, test- and operational-data, and a description of software development activities, such as debugging records, and validation and verification reports. In addition to reviewing the documents, it is desirable to visit the plant site to observe the system in operation, and to meet the software developers and plant personnel to resolve questions. To facilitate later tasks on expert elicitation, experts should be identified who are familiar with the software development of the example system. It is preferable to use a system that was developed following NRC guidance, that is, BTP-14 [NRC 2007], such that the model is more applicable to systems at U.S. nuclear power plants. The major issue associated with this step is the potential unavailability of detailed information/documents of the example system, and of its software and the associated development. A significant effort may be needed to collect all relevant information.

(2) Resolution of high-level issues

The most important issue to resolve is how to convert qualitative information on software development work into the software's probability of failure-on-demand. This probably also is the most difficult task in applying BBN to software reliability assessments. Software is the product of a software-development life cycle (SDLC). Each of the SDLC-related activities, which generally can only be characterized and evaluated qualitatively, has a certain impact on the software's reliability. While generally the trend is known of the impact of these activities on the software reliability or the number of remaining faults in the software, quantifying the impact is very difficult. More importantly, the subtle interplays between different activities also directly affect the software reliability, but probably are even more difficult to capture. As discussed in Sections 5.2.2 and 5.2.3, the state-of-the-art in this area is very weak mainly due to lack of operational experience of similar software development projects; there is a need for additional research, such as a benchmarking study of BBN applications to software reliability or data collection.

(3) Development of high-level BBN model

The high-level structure depends on the resolution of the issue of converting qualitative information to software failure-on-demand. If a direct conversion is used, the high-level

structure of Gran [2002a] might be considered a starting point. This is similar to how human reliability is assessed from performance-shaping factors. If an indirect conversion is used, the high-level BBN discussed in [Eom 2009] can be adopted first, to convert the qualitative information to the number of remaining faults, then employing the high-level structure of Delic [1997] to obtain the software's reliability. In either case, the high-level issues indicated previously first need to be resolved.

(4) Development of expert-elicitation approaches

Two types of application expert elicitation should be used. The first type employs experts (Type I experts) who are familiar with software-development activities and associated guidance and requirements. The experts will participate in developing the BBN model, including its structure, and quantifying the node probability tables. Since the products of the experts are based on generic information, the BBN model is a generic one, applicable to any software developed following the guidance. Two kinds of NPTs must be quantified; those that evaluate the quality of development activities, and those that convert the quality or the number of faults into software failure probabilities. The latter is the most difficult to do. The second type of experts (Type II experts) requires people familiar with the development of the specific system under evaluation. They are responsible for providing system-specific answers associated with the observable nodes of the BBN model. It is questionable whether adequate Type I experts can be identified. Type II experts should be more readily available, since the software developers should qualify as this type of expert.

Investigations are needed on how to perform an expert elicitation. [Tregoning 2008] gives an example of implementing such an elicitation. The issues for experts to consider should be carefully selected and clearly identified. The technical issues to be solved dictate the identification and selection of the panel experts. Those chosen need training. Background information and material on the issues is required for the panel experts. Individual responses that are highly uncertain or significantly different are allowed in the elicitation, but significant systematic biases must be avoided. Specific questions need developing, based on the technical issues. To better formulate elicitation questions, breaking down or refining the issues may be needed and helpful. The responses from individual experts are analyzed, aggregated, and documented. Any conflicts between experts' responses may need reconciliation.

The next three steps describe the activities that will be supported by experts, possibly through expert elicitation.

(5) Development of the complete BBN model

The low-level BBN model should account for the requirements specified in the guidance for developing software, that is, BTP-14 for U.S. NPPs. The PRA experts who are responsible for developing the model should undertake this task to ensure the lower level model is consistent with the higher level model. They will be assisted by experts in developing software and the associated guidance. It is preferable that the model be developed according to BTP-14, so that it is more applicable to systems at nuclear power plants. Note, the example system may not have been developed from the guidance of BTP-14 and, therefore, cannot be evaluated directly against it. However, the standard(s) on which the example system development was based on might be

interpreted and the interpretation compared to the corresponding requirements in BTP-14.

It is desirable to keep the overall BBN structure as simple as possible because, even if a complicated BBN is built that correctly reflects the causal relationship between different nodes, not much can be gained from the model without sufficient supporting data. Obtaining supporting data could be easier for a simpler BBN structure. Each node of the BBN model should be defined clearly including its possible states. Causal relationships between connected nodes should be described precisely. The conditional independence of the BBN model should be verified.

In addition to a detailed BBN model, it is desirable to develop a simplified one, building upon the experience of developing the former. While the simplified model may not be based on a specific guidance, it should cover the general requirements of guidance or standards, such that it might be used to evaluate software developed using different standards.

(6) Expert elicitation to quantify node-probability tables

This part of the study would quantify the generic part of the BBN model formulated in the preceding step with system-specific nodes and questions to be quantified in the next step. PRA experts should undertake this task by eliciting the Type I experts with experience and knowledge in software development and its associated guidance. In this step, the entire BBN should be structured completely via a pre-selected approach, e.g., a "top-down" approach that builds subnets starting from a high-level BBN after properly determining the level of detail of the BBN (e.g., the number of BBN nodes and the number of states for a node). In addition, a critical aspect of this step is eliciting experts to populate the NPTs for individual nodes.

(7) Collection of evidence on observable nodes

This step assesses the quality of software development activities of the specific software system under evaluation by collecting evidence associated with the observable nodes of the BBN model using available documentation on the system (e.g., debugging data). It also encompasses eliciting Type II experts for answers to questions about the observable nodes of the BBN model (e.g., is the software specification verifiable and its implementation traceable?). Such knowledge can be provided by developers of the software or a third party that is familiar with the software after being given the relevant information.

(8) Performing BBN analyses and generating results

Standard software tools such as AgenaRisk[®] [AgenaRisk 2008] can be used in developing the BBN model, and performing the needed analyses, such as generating a probability distribution for the probability of failure-on-demand. This step is time-consuming but straightforward.

- (9) Performing sensitivity calculations to address modeling assumptions and issues

The BBN model can be used to perform the sensitivity calculations discussed in Section 5.2.5 to determine the importance of different assumptions and issues. In addition, sensitivity calculations can be used in assessing the importance of specific guidance/requirements for developing software and their implementation. Generating alternative models/methods as suggested in Section 5.2.5 would require a significant amount of resources, and have to be considered as beyond the scope of the current plan for the case studies.

- (10) Performing benchmark studies to evaluate the BBN model

As discussed in Section 5.2.4, the statistical testing results of the example system can determine if the prediction of the BBN model is consistent with the test results. This will be done in the planned case studies. However, to validate and calibrate the BBN model, many additional benchmarking studies must be completed. The benchmarking studies effectively would be applications of the BBN model to other software projects, and they would evaluate whether the BBN model produces reasonable results.

5.4 Evaluation of the BBN Approach against the Desirable Characteristics

The BBN method was initially evaluated in Section 2 against the 10 desirable characteristics for a QSRM. An updated evaluation that compares the BBN approach proposed in the preceding subsections against the desirable characteristics is provided below.

- (1) *The description of the method and its application is comprehensive and understandable.*

The BBN method has been used successfully in non-nuclear applications and is well-documented in the literature, meeting this characteristic.

- (2) *The assumptions of the method have reasonable bases.*

The conditional independence assumption of a BBN model needs to be verified when the BBN structure is developed, and is an issue for consideration in applying the BBN method. In some cases, unexpected results might reflect incorrect assumptions on conditional independence when the model is exercised. A “Maybe” is assigned.

- (3) *The method allows for consideration of the specific operating conditions of the software.*

An inherent weakness of the proposed BBN model (though not necessarily of the BBN approach, in general) is that it does not account for different operational conditions. One potential way of addressing this characteristic is to use context-specific statistical testing in a Bayesian updating process (as discussed in Section 6.2.4).

- (4) *The method takes into consideration the quality of lifecycle activities.*

The BBN method uses conditional probability tables to represent interdependency among disparate events, and, in PRA applications, can potentially combine qualitative

information, such as quality of software lifecycle activities, thereby meeting this characteristic.

- (5) *The method makes use of available test results and operational experience.*

The BBN method uses test and operational data in a Bayesian analysis, thereby meeting this characteristic.

- (6) *The method addresses uncertainty.*

With a BBN approach, uncertainties associated with the events/nodes are represented explicitly in their probabilities, thereby meeting this characteristic.

- (7) *The method has been verified and validated.*

Since the existing models reviewed are explorative ones, they are not considered validated models. A few experts have stated that the method is promising [Littlewood 2000], earning a “Maybe” for this characteristic. The performance of benchmark studies recommended in this study would serve as a validity check for the BBN model.

- (8) *The method is capable of demonstrating the high reliability of a safety-critical system.*

While previous applications of the BBN approach have not produced estimates of very high reliability, it is possible that for nuclear protection system software a panel of experts could agree upon a very high reliability, potentially meeting this characteristic. Since the objective of this study is only to use a BBN model to estimate a prior distribution that will be Bayesian updated with statistical test data, this characteristic is of lesser importance.

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

It would be very difficult to use the BBN method to estimate CCF parameters. Therefore, the issue of CCFs would probably need to be addressed external to the BBN model (as discussed in Section 2.1).

- (10) *The data and needed information exist and can be collected.*

Applications of the BBN method often use discretized probability distributions to represent the nodes of the models, and subjective expert elicitation in estimating the conditional probability tables representing the impacts that parent nodes have on child nodes. In particular, it is difficult to develop conditional probability tables that convert qualitative information, such as quality of software development activities, into failure probabilities and rates. It is unclear whether this is a task that can be appropriately resolved through expert elicitation, and whether the needed experts exist, resulting in a “Maybe” rating for this characteristic. In addition, as discussed previously, data of past software development projects are not available or are proprietary, but statistical testing may provide some data for benchmarking.

6. DEVELOPMENT OF A CONTEXT-BASED STATISTICAL TESTING METHOD

6.1 Introduction

Software usually is tested to detect bugs, such that those identified be removed, and also to demonstrate that the software can perform its intended functions, possibly for licensing purposes. Different test strategies have been developed, for example, black-box and white-box testing. However, they are not designed for quantifying software reliability, that is, the failure probability on demand, and thus, cannot be used for that purpose [May 1995, Hamlet 1994, Kuball 2004]. The main reason is that the inputs to the software in these tests are not random samples from the operational profile. Therefore, separate operationally representative tests must be undertaken. Such testing performed to support quantifying software reliability, i.e., the probability of failure on demand, is called statistical testing [IEC 1986].

Statistical black-box testing can be done by random sampling according to the operational profile. Some researchers also developed statistical methods based on white-box testing [May 1995, Zhang 2004, Yang 2010] to evaluate the overall probability of system failure taking into consideration the internal structure of the software. The white-box test-based methods offer additional reliability information about the internal structure of the software, but require more resources for the analyses. In addition, as discussed in Section 2, the number of tests needed using white-box testing may be much larger than that needed for black-box testing in order to demonstrate the same software failure probability. Equivalently, using the same number of tests (without failure), the white-box method of [May 1995] would generate a higher (i.e., unnecessarily conservative) estimated mean failure probability than that from the black-box method [Chu 2010]. As indicated in [Zhang 2004], a white-box method that attempts to encompass all possible paths may be impractical. For these reasons, only black-box testing is considered in this study.

In a probabilistic risk assessment (PRA) of a nuclear power plant (NPP), different accident scenarios represent different challenges to the safety systems actuated by digital protection systems, such as the reactor protection system (RPS) and engineered safety feature actuation system (ESFAS). Therefore, each challenge represents a unique condition/context for the protection system, as part of the RPS operational profile. The existing statistical methods for quantifying the probability of failure on demand need to be applied in a way that accounts for the accident scenarios defined in a PRA.

In this section, an approach is proposed for statistical testing of digital protection systems, such that the results can be directly used in supporting PRA modeling of these systems. In particular, the testing considers the contexts or boundary conditions defined in a PRA (statistical testing can be combined with other quantitative software reliability methods that consider earlier phases of software development and estimate a generic prior distribution for the probability of failure on demand). In fact, the proposed approach can be considered a white-box method (Similar to May [1995]) in which the input space is partitioned according to PRA scenarios. Appendix B describes different types of tests for licensing purposes, and the configurations used in testing some NPP systems. Statistical testing would ideally be employed to a system once it's in its operational configuration, but it may not be feasible to test the system after its installation at the NPP. Alternatively, the testing should be performed after or as a part of the factory acceptance test. Section 6.2 describes the strategy for undertaking the black-box testing, including the

needed test configuration, proposes a risk informed approach for statistical testing, provides the mathematical formulas for calculating the system's failure probabilities on demand, and discusses assumptions and issues associated with such testing, including treatment of uncertainty. Section 6.3 proposes a demonstration of the feasibility of using the statistical testing method to support PRA modeling of software failures by estimating the number of test cases needed to ensure that the overall risk from software failure is below an established level. Section 6.4 discusses the benefits of the proposed approach in terms of the desirable characteristics of Section 2.

6.2 A Context-Based Statistical Testing Method

6.2.1 Contexts Defined by a PRA

At any particular time, the software, the device(s) controlled by it, the system wherein the software is embedded, and the NPP are in a certain state. In general, the NPP's state provides the overall context for their operation. For example, the input to the software depends on the plant's state. Some inputs to the software resulting from the context (states) of the NPP may trigger software faults, leading to a software failure. Garrett and Apostolakis [1999] used the term "error-forcing context" (EFC) for denoting a context in which a software failure may happen. This concept is very similar to modeling human performance in an NPP; an unsafe human act is considered meaningful only in the system context within which it occurs. This observation led the Office of Nuclear Regulatory Research (RES) of the U.S. Nuclear Regulatory Commission (NRC) to develop the concept of the EFC [Cooper 1996]. According to this model, a software failure or a human failure happens due to the occurrence of an EFC; the impact of either one on the NPP can range from minor to severe.

As such, it is important to account for the software's context in modeling software failures in the PRA of an NPP. For example, in a typical PRA, the RPS usually is the first heading in the event trees after an initiating event. Therefore, each initiating event (IE) defines, at a high level, a context for this system, that is, different initiating events represent dissimilar plant conditions that may generate distinct input signals to the RPS. Accordingly, if the RPS contains software, the context for this software may differ for various initiating events because the plant conditions under each IE are different, and hence, the inputs to the software differ.

Table 6-1 gives some examples of plant conditions requiring the RPS to trip a typical pressurized water reactor (PWR) during full-power operation. The right-hand column of this table lists an example of the type of IE generating the condition necessitating reactor trip. Each plant condition requiring a reactor trip, such as each row in Table 6-1, represents the context under which the RPS will be operating. If the functioning of an RPS includes software, then this software's context also is defined by this condition.

In addition, failure of the software after an IE may be due, at least in part, to inputs to the software before the IE; for example, the condition of the plant before the IE may fluctuate and produce varying inputs to the software. Hence, the software's context may not only encompass its latest inputs (due to the IE itself), but may also contain inputs received before the IE. Accordingly, the context may consist of a trajectory of inputs (a series of successive values for the input variables of a program that occur during the operation of the software over time); this is further discussed in Section 6.2.5.

Table 6-1 Examples of conditions requiring reactor trip for a typical PWR.

Plant condition	Trip	Example of initiating event causing this condition
2/4 pressurizer pressure channels are below the low-pressure trip setpoint.	Low pressurizer pressure	Loss of coolant accident (LOCA).
2/4 pressurizer pressure channels exceed the high-pressure trip setpoint.	High pressurizer pressure	Turbine trip.
2/3 narrow range sensors on any steam generator (SG) indicate a low-low level.	Low-low steam generator level	Loss of main feedwater.
2/3 loop flow indicators are below the low-flow trip setpoint.	Low reactor coolant flow	Loss of offsite power.
Trip signal generated by the operators.	Manual	Manual trip.

Similarly, for other protection systems, such as a system (e.g., an ESFAS) that generates an actuation signal of a safety system, the context of operation can be identified by the plant's condition at the time that the system is required to operate. For instance, the ESFAS of a typical PWR will actuate when certain plant conditions occur, such as a "low pressurizer pressure" indicating that a LOCA might have happened. Depending on the conditions of each accident sequence, the operators may manually start a safety system if the ESFAS fails to produce an actuation.

Since the ESFAS generates an actuation signal of a safety system, and the latter is the one that performs a mitigating function, a PRA usually considers the latter to be a "frontline" system, and includes it as a node (i.e., heading) in an event tree (an event tree is a binary model of the successes and failures of mitigating systems and high-level operator actions that can influence the outcome of an IE).⁷ The ESFAS, deemed a "support" system, typically is not included as a heading in the event tree. Nevertheless, since the ESFAS "supports" the safety system's operation, the PRA model of the mitigating system (commonly a fault tree) can explicitly include failures of the ESFAS. In this way, the ESFAS implicitly is included in an event tree, and hence, is in the applicable event tree sequences leading to an undesirable event, such as core damage; for simplicity, these sequences simply are named here as sequences, and the undesirable event is core damage.

Each sequence differs from the other sequences in a PRA model, and represents an accident scenario depicting the progression of the accident from the IE to the conditions resulting in an undesirable event. Accordingly, the progression of the accident from the IE to the point in the sequence where the system(s) initiated by the ESFAS are used defines the context, at a high

⁷ More information on PRA modeling can be found in the PRA Procedures Guide [Hickman 1983].

level, for the actuation of the ESFAS. If the ESFAS includes software to perform its function, then this software's context also will be defined by this part of the sequence. In general, different sequences in the same or different event trees define different contexts for the ESFAS and its software.

Solving a PRA model qualitatively yields the cutsets of each sequence. Each cutset represents a more detailed way in which the sequence may occur, that is, instead of representing the sequence at the system level, it is represented at the basic event level (generally, individual component failure modes, such as a pump failing to start). Hence, those basic events associated with the hardware failures and human errors (and possibly also with software failures) that occur before the ESFAS-actuated systems in a cutset define a more detailed context for the operation of the ESFAS software. Since a sequence can contain both failures and successes of mitigating systems and human actions, the successes in a cutset also are part of the context. Here, for simplicity, cutsets are discussed only in terms of failures.

The resolution of a PRA should be considered when choosing a modeling and quantification approach for software failures because the finer the resolution, the more detailed is the context for the software of a digital system. The relationship between the level of resolution of a PRA and the level of detail of the software's context is illustrated with the following example. It is a common practice in PRAs to gather similar IEs into a single group, so reducing the number of IEs to be addressed from a relatively large to a more manageable number. Assume, hypothetically, that an analyst put LOCAs of all sizes into a single IE group (such coarse lumping is unlikely to happen in practice). Then, the context for the software of a LOCA-mitigating system probably will be given by an "overall" context that hopefully would represent all LOCAs. On the other hand, grouping the LOCAs in a more detailed way (such as large, medium, and small LOCAs) would generate a more detailed context for the software within each IE LOCA group; hence, it would reflect more realistically the plant conditions resulting from the occurrence of a LOCA of such size. In addition, each LOCA size leads to different timing in terms of the time required for mitigating systems to actuate (including software-driven systems). Such timing also can be considered part of the software's context. Software testing then would be organized according to the level of detail defined for each case; in this example, the software's context would be coarse for the former case, and be more detailed for the latter. The level of resolution of a PRA at which testing is done can be iteratively increased (i.e., made more detailed) to achieve a balance between this level and the detail of the software's context considered adequate for carrying out the testing. The relationship between a PRA's level of resolution and a software's context not only applies to IEs, as illustrated here, but also to the level of resolution of other elements of a PRA model, such as the resolution of system-level models.

6.2.2 A Risk-informed Strategy for Performing Statistical Tests

Statistical testing of software typically involves testing the software over many possible inputs to it as defined by the software's input space. Since the number of inputs can be very large, testing the software in this way entails undertaking an extremely large number of tests, with the consequent costs in resources, that is, time, money, and effort. This poses one of the most difficult problems in assessing the probability of failure of a software program. This subsection proposes a risk-informed strategy for performing statistical tests that is intended to help resolve this issue. The idea is similar to that used in a National Aeronautics and Space Administration (NASA) study [ASCA 2007]. The strategy essentially consists of accounting for the contribution of software failure to the risk of an NPP. Its objective is to bound the maximum value of the

probability of a software failure, so to meet a risk or reliability goal. For example, if it is shown that the maximum value is, say, 10^{-3} , to reach such goal, then software testing only would have to demonstrate this number, thus reducing (possibly substantially) the number of tests needed to be carried out (with no observable failure). Accordingly, applying this strategy will only demonstrate whether the software reliability meets the established goal; it will not generate the exact or true failure probability. In addition, it should be made clear that this approach is only intended to make it more practical to include a bounding value for software failure probability in a plant PRA, by reducing the testing burden, and is not presently intended to be used to support software licensing reviews.

This subsection presents two approaches for implementing this strategy: (1) Accounting for the contribution of software failure to the risk of the entire NPP, and (2) accounting for the contribution of software failure to the failure probability of a mitigating system or function of the NPP. Each approach is discussed in terms of mean values of probabilities and frequencies since mean values commonly are used in NRC regulatory guides on risk-informed regulation, such as the risk-acceptance guidelines of Regulatory Guide (RG) 1.174 [NRC 2002]. At the same time, when assessing a risk metric, such as core damage frequency (CDF), a probability distribution for each basic event in a PRA is expected to be used to propagate the parameter uncertainty throughout the PRA model. Section 6.2.4, "Bayesian Reliability Assessment," addresses determining a probability distribution for a software failure event. NUREG-1855 [Drouin 2009] discusses uncertainty in a PRA model, including parameter uncertainty and its propagation in the model.

6.2.2.1 Accounting for the Contribution of Software Failure to the Risk of the Entire NPP

Since the progression of an accident, defined by a cutset, specifies the context for the operation of a software embedded in a system, a cutset containing failures of this software can be expressed as

$$SC \cdot SF \cdot OF$$

where the symbol "•" means Boolean "AND," and SC is the context for the operation of the software, defined by the failure(s) in the cutset that occur before the software failure in the accident progression as defined in the associated event tree. SC can represent one or more failures; for example, it may be just the IE, or can include additional failures. Though the context is defined in these terms, in theory it may be different because either (1) the software failure may happen before an IE, or (2) some input to the software before the IE happens may contribute to a software failure after an IE. This issue is considered in Section 6.2.5.

SF is the software failure appearing in the cutset. This failure occurs given the context SC in the same cutset.

OF is a placeholder for other failures necessary for the cutset to entail core damage, but that occur after the failures represented by SC and SF. Therefore, the failures within OF, if any, are not part of the software's context in the cutset.

The failures that occur prior to and after a software failure in a cutset can be identified by examining the accident progression defined by the cutset, and the associated sequence. SC quantitatively is characterized by a frequency since it contains the IE, while each SF and OF is

characterized by a conditional probability. In general, SF is conditional on the occurrence of SC, and OF is conditional on the occurrence of SC and SF.

The contribution of a cutset, i , to CDF is expressed as

$$\text{CDF}_i = [F(\text{SC}) * P(\text{SF} | \text{SC}) * P(\text{OF} | \text{SC and SF})]_i$$

where “*” means multiplication, $F(\text{SC})$ is the frequency of the software context, $P(\text{SF} | \text{SC})$ is the conditional probability of SF given SC, and $P(\text{OF} | \text{SC and SF})$ is the conditional probability of OF given SC and SF. For simplicity, $P(\text{SF} | \text{SC})$ and $P(\text{OF} | \text{SC and SF})$ are shown, respectively as $P(\text{SF})$ and $P(\text{OF})$, even though they are conditional probabilities.

This equation can be used to bound the value of the probability of a software failure to keep the value of CDF of each cutset equal to, or less than, a certain value. For example, if it is desirable to keep the contribution to CDF of each individual cutset involving software failure to less than or equal to $10^{-7}/\text{year}$ ⁸, then for cutset i ,

$$10^{-7}/\text{year} \geq [F(\text{SC}) * P(\text{SF}) * P(\text{OF})]_i$$

Solving for $P(\text{SF})_i$,

$$P(\text{SF})_i \leq 10^{-7}/[F(\text{SC}) * P(\text{OF})]_i$$

where $P(\text{SF})_i$ is dimensionless because it was assumed that $F(\text{SC})$ is expressed in units of per year since an IE typically is expressed in these units.

The bound for the probability of software failure [$P(\text{SF})$] appearing in the cutset then is obtained from the last equation using numerical values for $F(\text{SC})$ and $P(\text{OF})$. For instance, in the internal events PRA of the Surry NPP completed as part of NUREG-1150 [NRC 1990], there is a sequence consisting of a Large LOCA (LL, the IE) and failure of low pressure injection (LPI). Assuming that the ESFAS of this plant employs software, and that a cutset is being evaluated wherein the failure of LPI is caused by failure of the software to generate a signal to start this system, then SC is the occurrence of the LL, expressed in terms of the input to the ESFAS' software indicating that a LOCA happened. As discussed above, the software's context may not only consist of the latest inputs to the software (due to the IE itself), but may contain inputs received before the IE. Accordingly, the context may encompass a trajectory of inputs; this is further discussed in Section 6.2.5. For this example, it can be considered that the context is the occurrence of the LL, so $F(\text{SC}) = F(\text{LL}) = 5.0 \times 10^{-4} / \text{year}$ (from NUREG-1150 [NRC 1990]). Since only the occurrence of LL and failure of LPI (due to failure of the hypothetical software) is required to cause core damage, OF does not exist in this cutset.⁹ Hence, the last equation becomes,

⁸ This value is for the purpose of illustration only, and does not imply a desired threshold. Also, it should be noted that different thresholds may be desired for existing and new reactors, since new reactors are expected to have a lower risk profile than existing reactors.

⁹ During the Oconee digital upgrade [NRC 2010], a diverse actuation of the low-pressure injection system was added that can further lower the needed software reliability. It is expected that providing diverse means of fulfilling a mitigating function will generally decrease the required software reliability. Investigating the approaches needed for assessing the reliability of a diverse configuration (i.e., a configuration involving software and other equipment not driven by software or driven by diverse software), including evaluating the potential impact of common failure, is a relevant field of study; it is considered beyond the scope of this project.

$$P(SF)_i \leq 10^{-7}/5.0 \times 10^{-4} = 2 \times 10^{-4}$$

Accordingly, for this cutset, as long as the failure probability per demand of the ESFAS' software is no greater than 2×10^{-4} , then the contribution of this sequence to CDF will be no greater than 10^{-7} /year. This implies that when testing the software to assess its failure probability per demand, it would only be necessary to carry out testing to demonstrate that this probability is no greater than 2×10^{-4} . This approach can be considered risk-informed software testing and, in this case, reduces the required number of tests to 4998.¹⁰ Clearly, for different proposed values of a cutset's CDF contribution (i.e., the term CDF_i), different values will be obtained for the bound of the software's failure probability $[P(SF)_i]$.

Cutsets containing other failures that are not a part of the software context (i.e., the term "OF") may lead to larger bounds of software failure probabilities than those associated with cutsets that do not contain this term; an instance of the latter is the cutset of the large LOCA mentioned above. This distinction is illustrated using a cutset from the NUREG-1150 Surry PRA [NRC 1990] involving a small LOCA (SL, the IE), failure of the RPS (i.e., an Anticipated Transient Without Scram (ATWS) event), and failure to mitigate the ATWS that consisted of two conditions: (1) Failure of the operator to manually trip the reactor (R); and, (2) the presence of an unfavorable moderator temperature coefficient (Z). Assuming that the RPS is implemented using software, and that the cutset being evaluated includes the failure of the RPS to generate a reactor trip (scram) signal due to a software failure, then the sequence may be represented as $SL \cdot SF \cdot R \cdot Z$. Considering again that it is desirable to keep the contribution to CDF of this cutset to less than or equal to 10^{-7} /year, then

$$10^{-7}/\text{year} \geq [F(SL) \cdot P(SF) \cdot P(R) \cdot P(Z)] = 10^{-3}/\text{year} \cdot P(SF) \cdot 1.7 \times 10^{-1} \cdot 1.4 \times 10^{-2}$$

where the numerical values on the right-hand side of the inequality are from NUREG-1150. In this example, SF is only conditional on the initiating event (SL), and the other failures (i.e., "OF") are R and Z. The event R is deemed to also be conditional on SL, so $P(R)$ is a conditional probability, and $P(Z)$ is an unconditional probability.

Solving for $P(SF)$,

$$P(SF) \leq 10^{-7}/2.4 \times 10^{-6} = 4.2 \times 10^{-2}$$

As described above, using this result should substantially reduce the effort required when applying statistical testing because it only would have to show that the software failure probability is less than 4.2×10^{-2} , necessitating only 22 tests.

The approach described for a single cutset can be extended to an entire sequence. Since the contribution of a sequence j to CDF (CDF_S) usually is estimated by summing up the contribution from each cutset in the sequence, then this contribution can be assessed as

¹⁰ It is reiterated that this approach is only intended to make it more practical to include a bounding value for software failure probability in a plant PRA, by reducing the testing burden, and is not presently intended to be used to support software licensing reviews.

$$CDF_s = \sum_i [F(SC)*P(SF)*P(OF)]_i$$

Then, the probabilities of software failure P(SF) of all the cutsets should be such that the right term of this equation is equal to a maximum sequence frequency. For instance, if it is desirable to keep the contribution to CDF of a sequence involving software failures to less than or equal to 10^{-6} /year, then

$$10^{-6}/\text{year} \geq \sum_i [F(SC)*P(SF)*P(OF)]_i$$

In general, the context for the software's operation [F(SC)] differs for each cutset of a sequence, though, in practice, the context may be the same for several cutsets. For example, the context for some cutsets of the same sequence may simply be the IE of the sequence.

The value used for CDF in the previous inequalities (e.g., 10^{-6} /year in the last inequality) can be related to actual regulatory practices or guidance, such as the risk-acceptance guidelines of Regulatory Guide (RG) 1.174 [NRC 2002]. A simplified example illustrates using these guidelines by assuming that (1) the relevant risk metric is the CDF, (2) an NPP is replacing an analog system with a digital system, and the evaluation is related to the impact on risk of the digital system's software, (3) the baseline CDF of the NPP is 10^{-5} /year, (4) it is desirable that the change in the NPP's CDF resulting from this replacement is very small, such that the ΔCDF is less than 10^{-6} /year, and, (5) the NPP's CDF is calculated using a single sequence. Then, the $\Delta CDF \geq CDF_s - 10^{-5}$ /year, or

$$10^{-6}/\text{year} \geq \sum_i [F(SC)*P(SF)*P(OF)]_i - 10^{-5}/\text{year}$$

Solving this inequality for $P(SF)_i$ yields the values of these probabilities that satisfy the RG 1.174 guideline related to a change in the NPP's CDF. Simplifying this inequality further for explaining its application, if the sequence only contains the single cutset from the NUREG-1150 Surry PRA mentioned above (i.e., $F(SC) = F(LL) = 5.0*10^{-4}$ / year, and OF does not exist) then

$$P(SF) \leq [(10^{-5} + 10^{-6})/\text{year}]/F(SC) = 1.1*10^{-5}/5.0*10^{-4} = 2.2*10^{-2}$$

This result from a simplified analysis means that the software failure probability only needs to be demonstrated to be less than $2.2*10^{-2}$ in order to meet the regulatory guideline assumed in this example, thereby requiring only 44 tests. In practice a sequence may have many cutsets containing software failures, so the cumulative frequency from all of them must meet the regulatory guideline; this implies that each software failure probability would likely have to be significantly smaller than the numeric bound obtained in this simplified example (i.e., $2.2*10^{-2}$). In addition, the potentially large number of sequences and cutsets in the plant PRA that contain software failures may lead to a prohibitive number of contexts that need to be tested, as further discussed in Section 6.2.5.

An analogous process can be carried out for all sequences in a PRA, and for the risk-acceptance guideline of RG 1.174 associated with large early-release frequency (LERF).

6.2.2.2 Accounting for the Contribution of Software Failure to the Failure Probability of a Mitigating System or Function of the NPP

Another approach for bounding the software failure probability is by requiring that the contribution of software failure be a small fraction (e.g., 10%) of the probability of failure of a function or system. One way to implement this is by first assessing the failure probability of the function or system due to hardware failures, and then demonstrating that the probability of software failure is a small fraction of that of hardware failure. Accordingly, this involves evaluating the latter, and deciding upon the fraction of this probability that should bound the probability of software failure.

This approach is illustrated by considering the failure of a mitigating function, such as providing coolant to the reactor vessel after an IE involving loss of primary coolant. For example, in the scenario discussed earlier of a large LOCA in the Surry NPP, water had to be injected into the vessel to avoid core damage; the LPI system supports this function. Assuming that (1) the LPI's failure dominates the failure of this function, (2) the LPI is started by a digital ESFAS containing software, (3) the hardware failure probability of LPI is 10^{-3} , and (4) the contribution of the probability of software failure to the total probability of failure of the LPI should be 10% or less, then the bound for software failure probability is 10^{-4} :

$$P(\text{SF}) \leq 10^{-4}$$

This result means that statistical testing only would have to demonstrate that the failure probability of the hypothetical ESFAS software is not greater than 10^{-4} , thereby reducing the resources needed for testing. The 10% maximum contribution of the probability of software failure to the total probability of failure of the LPI was chosen simply to illustrate this approach. In general, this contribution can be selected as a small percentage of the total probability of failure of a system or function to bound the contribution of software failure to this total probability.

6.2.3 Performing Statistical Tests

Statistical testing is intended to assess the probability of failure of the software of a protection system in the different scenarios/contexts in which it is needed. Therefore, the tests should be performed to simulate the system under these conditions. As discussed, a PRA model defines, at a high level, such scenarios/contexts. An operational profile for each of the scenarios must be defined and used in creating test cases. The inputs representing the test cases are fed to the protection system and the correctness of the system's outputs are evaluated.

Implementing this conceptual approach is described in this subsection as follows: (1) Specifying the test configuration, (2) generating test cases by sampling from the operational profile, and (3) evaluating the findings from each test with an oracle to determine if the test is successful. As an example, a medium LOCA in which the RPS is needed is used.

6.2.3.1 Specifying Needed Test Configuration

Different test configurations may represent different degrees of realism in simulating the operation of a protection system, and may have dissimilar capabilities in capturing potential faults in the software. It is agreed generally that the original application software should be used in testing the software. Some insist in using only the original hardware because its

performance (e.g., speed) may be critical in determining the interaction between software and hardware; a different piece of hardware thus might change the triggering event of a software fault and the software's responses. Others even argue for using exactly the same compiler and the running environment for the software in the tests because different compilations and execution environments may affect the software's behavior. However, employing the original hardware may pose extra difficulties and costs in testing. Limited accessibility is likely to the internal states of the software and hardware because the original hardware may not be designed to accommodate this purpose. Research is needed on how realistic the test configuration must be and on the tradeoff between realism and the accessibility of internal information.

A more practical limitation is the availability of the equipment needed for undertaking the tests. Vendor developed tools for testing are more readily available, and it may be impractical to develop new tools for a specific application. The vendor's tools may need modifications to facilitate statistical testing. For the Oconee digital upgrade [NRC 2010], a test machine, ERBUS, developed by AREVA, was employed in the factory acceptance tests. It is an open-loop test configuration without physical connections to the NPP's thermal-hydraulic simulator. Appendix B details this test configuration and the automatic test tool of a Japanese ABWR used to validate and verify a safety-related digital system software [Fukumoto 1998]. At a high level, the two test configurations are the same. That is, a test computer generates test inputs and analyzes outputs that are sent through the interfaces between the computer and the digital protection system (including hardware and software) being tested. The test cases are generated offline employing a thermal-hydraulic model of the plant (i.e., without connecting the thermal-hydraulic model of the plant directly to the system under test). A test configuration consistent with this high-level description is recommended for the statistical testing approach in the case study.

Alternatively, a test tool can be designed for software validation in a simulated environment, that is, without using the original hardware, such that a class of digital I&C systems can be tested. Not using the original hardware means that either a software model must be created or reconfigurable hardware produced (e.g., a field programmable gate array [FPGA]) to mimic the original hardware; either way would increase flexibility and lower cost from the view point of people performing the tests. A limitation with this approach is that the software model or the hardware reconfiguration would only approximate, and not duplicate, the original hardware. A tool of this kind is the SIVAT (Simulation Validation Test) developed by Areva NP [2009] for TELEPERM XS systems, wherein the original hardware is represented by models. The SIVAT tool is briefly described below.

In SIVAT, a model actually is a software function written or generated in a higher-level programming language (e.g., C or Fortran). A model represents the CPU of a TELEPERM XS, ensuring that all internal signals and variables of the model are stored in the simulator database. An input or output model also can be formulated and used in SIVAT for the measuring and actuating periphery of the TELEPERM XS system. Linking separate models for the physical process to SIVAT enables a partial or complete closed-loop simulation of realistic events or disturbances. In addition, SIVAT models communication between individual TELEPERM XS CPUs. It also encompasses a simulator control system (the same as that used in the ERBUS test machine) to support the successive execution of the application software code and the visualization and modification of software variables. In summary, SIVAT performs TELEPERM XS system simulation based on the models of CPUs and/or input and output models and the original code of the application software. Clearly, the accuracy of the simulation depends on

how well the model(s) represent the system or the process. Although SIVAT models the hardware and process as realistically as possible, it is preferable to use a test configuration that consists of the actual hardware and software. Without a detailed assessment of the tool, it is difficult to justify whether, or by how much, the hardware/software interaction can be captured in a simulated testing environment.

6.2.3.2 Generating Test Cases by Sampling from the Operational Profile and Evaluation of Results

This part of statistical testing describes how to generate the test cases for each of the scenarios/contexts detailed in Section 6.2.2 using the test configuration described in Section 6.2.3.1.

There are a few publications on estimating the software failure probabilities of safety-critical systems using statistical testing [Kuball 2007, Littlewood 1997, May 1995] that provide useful theoretical guidance on operational profiles and generating test cases. An important concept of the guidance is that the test cases represent 'trajectories' (a series of successive values for the input variables of a program that occur during the operation of the software over time) in the space of inputs to the software. That is, a set of single values of the input signals cannot represent the input of a test case. This proviso is related to the possible memory of the protection system, that is, its performance may depend on the history of inputs leading to the demand on the system. Other key concepts of the guidance cover the interdependencies of the input signals because they may represent related physical parameters, and the assumption that the test cases are independent, such that the results can be used in statistical analyses. The following is a proposed approach for generating test cases addressing the above issues.

For each context (accident sequence), for example, a medium LOCA requiring a reactor trip, a thermal-hydraulic simulator of the plant should be included to simulate the context, and generate a trajectory of inputs to the protection system under test. The simulation starts from the steady-state operation of an NPP and then simulates the occurrence of the accident, in this example a medium LOCA. The thermal-hydraulic simulator accounts for the dependency among the physical parameters. The steady operation of the NPP that might be characterized by fluctuations in its parameters can be represented by probability distributions. For a medium LOCA, more test cases can be simulated by considering the fluctuations in the initial conditions and in different medium LOCA sizes and locations. Additionally, consideration should be given to noise in the sensor signals and inaccuracies and time delays in the sensors. For example, modifying the sensor inputs using the inaccuracy specified by the sensor manufacturer can account for the variability of the sensor's precision. To ensure that the inputs from redundant sensors of the same type are considered independently, one can assume that the inaccuracy of the first sensor follows a normal distribution, and the difference between the signal of the second sensor and the first one is represented by a second normal distribution, and so on. Noise in the sensor signals and time delays can be treated similarly.

The probability distributions representing the fluctuations in the initial conditions, the frequencies of different LOCA sizes and locations, and the sensor's inaccuracies can be estimated and used in random sampling of the test cases. They define the operational profile of the context.

A problem in the above approach is the long time that each simulation run of the thermal-hydraulic model of the plant may take, especially when a particular protection function is not needed until late in an accident scenario, for example, automatic actuation of recirculation after

successful injection. That is, the simulation may need to cover 24 hours in accident time. Section 6.2.5 discussed the possibility of employing a shorter trajectory. Another issue is that the number of needed tests could be very large, considering the many sequences or cutsets that must be evaluated. Section 6.3 proposes an approach to estimate the required number of tests for demonstrating that the contribution of software failures to the risks of an NPP is small.

The outcome of each individual test is either that the protection system generates an actuation signal when needed or it does not. The oracle for determining if the test results are correct can be simply accomplished using the test computer to decide whether the actuation signal was generated in a timely manner. The timeliness is determined by the physical condition of the plant and the necessary information is sometimes available in design basis accident analyses. For example, for a medium LOCA, the expected time of reactor trip is determined by the plant's thermal-hydraulic model that generates the test inputs, and this can be compared to the latest time when the reactor must be tripped to prevent damage to the plant as determined by physical constraints, such as the onset of damage to the cladding.

6.2.4 Bayesian Reliability Assessment

As mentioned previously, it is proposed for the case study that the statistical testing results be used to perform a Bayesian updating of a prior distribution for the probability of software failure-on-demand. The prior distribution may be derived from either a detailed or simplified Bayesian belief network (BBN) approach, based on factors such as the quality of software lifecycle activities, or it may be non-informative, e.g., following existing guidance in the handbook of parameter estimation [Atwood 2002].

The Bayesian approach is a straightforward application of Bayes' theorem. Miller et al. [1992] derived the Bayesian methodology described here. The likelihood function is a binomial distribution, 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 probability density function (pdf) of Θ is

$$f(\theta) = \frac{\theta^{a-1}(1-\theta)^{b-1}}{B(a,b)} \quad (6-1)$$

where $0 \leq \theta \leq 1$, $a > 0$, $b > 0$, 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. Other 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, the quality of a software's lifecycle activities can be used to 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 $g(x|\theta)$ 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

$$f(\theta|x) = \frac{g(x|\theta)f(\theta)}{\int_0^1 g(x|\theta)f(\theta)d\theta} \quad (6-2)$$

Accordingly,

$$f(\theta|x) = \frac{\theta^{x+a-1}(1-\theta)^{n-x+b-1}}{B(x+a, n-x+b)} \quad (6-3)$$

where $x = 0, 1, \dots, n$, and $0 \leq \theta \leq 1$.

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

$$\frac{(a+x)}{(a+b+n)}. \quad (6-4)$$

Once the posterior distribution of Θ is determined, it is entered into one of the PRA codes that implement a method for propagating parameter uncertainty through a PRA model. In general, a different distribution of Θ must be obtained for each context defined for the operation of a software, as discussed in Subsection 6.2.1.

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

$$\Pr\{\Theta \leq \theta_u | x\} = \gamma \quad (6-5)$$

Solving this equation for θ_u determines an interval $0 \leq \Theta \leq \theta_u$ 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 $0 \leq \Theta \leq \theta_u$.

An interesting application of this approach is setting the parameters $a = b = 1$ for the prior probability density function because this function becomes the uniform distribution (i.e., $f(\theta)$ is a constant) that can be interpreted as a non-informative prior distribution [Martz 1982]. (Another choice of prior distribution is possible; for example, the handbook on parameter estimation [Atwood 2002] recommended employing a Jeffrey's prior distribution.) In addition, by making $x = 0$, as often is the case for testing of safety critical software, the posterior cumulative distribution function (cdf) is expressed as

$$F(\theta_u | x) = \Pr\{\Theta \leq \theta_u | x\} = \int_0^{\theta_u} f(\theta|x)d\theta \quad (6-6)$$

which reduces to

$$F(\theta_u | 0) = 1 - (1 - \theta_u)^{n+1} = \gamma \quad (6-7)$$

Solving this equation for θ_u

$$\theta_u = 1 - (1 - \gamma)^{1/(n+1)} \quad (6-8)$$

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 (6-8) as

$$n = \frac{\ln(1 - \gamma)}{\ln(1 - \theta_u)} - 1 \quad (6-9)$$

The above derivations assume that there is no observable failure during the tests. It is expected that, for a safety-related protection system, if a fault is discovered the software will be revised to remove the fault; then, the revised software is treated as a new software that will be re-tested after discarding the test results of the earlier version. Littlewood and Wright [2007b] contend that this approach may be too optimistic because it ignores the failure of the earlier version that is evidence of poor quality. They consider that it would be more conservative to assume that removing the fault does not improve the system reliability, and suggested combining the test results (including failures) of the earlier version with those of the new version. Consequently, a larger number of tests would be needed to demonstrate that a reliability goal is met if faults are detected, compared to the number needed based on assuming that the results of the earlier version are ignored.

6.2.5 Summary Discussion of Assumptions and Issues

Generally, in statistical testing of software, random samples are taken from the operational profile. However, the operational profile may not be well known, nor accurately represented. A NASA-sponsored study [ASCA 2007] suggested using adjustment factors estimated by experts as a means to account for differences between testing and operating conditions. As noted in [Chu 2010], this approach appears to be a possible way of addressing this issue, but due to the reliance on expert elicitation, the resulting factors will entail significant subjectivity and uncertainty, and may not be bounding.

As discussed in Section 6.2.3.1, different test configurations represent different degrees of realism. Thus, the effectiveness of different test configurations in representing the actual operating system should be researched further. In this study, it is suggested using a vendor-developed test configuration consisting of (1) the actual protection system hardware and software, (2) a test system, and (3) test inputs generated via a thermal-hydraulic simulator of the plant. This represents the most realistic test configuration readily available to an NPP. A less realistic configuration that does not use the actual protection system hardware is not desirable, and may not realistically capture the interactions between the application software and platform software. For example, the speed of the hardware may affect the software's responses.

Often, software failures occur when certain inputs trigger a fault in the system; these inputs can be identified and used to reproduce the failure. However, some experts suggest that failures

caused by certain software bugs (i.e., Mandel bugs) [Grottke 2007] are difficult to reproduce and are considered non-deterministic. Hence, testing may not be able to identify these bugs. Probably, these failures can be explained in terms of the changing internal states of the digital system that remain unaccounted for in attempting to reproduce the failures. How tests can ensure the bugs are found is an open question.

A specific issue associated with the internal states of digital systems is that their distribution may not be stationary (constant in time), such that representative samples of the internal states from the starting point of the test cases cannot be taken. For example, the problem with the Patriot missile [GAO 1992] resulted from running the system too long, such that a small numerical error grew in time and caused system failure. Tests with a short duration may never identify the error. This can be considered as an issue in the “memory” of the system, that is, how system states depend on past inputs. For an RPS or ESFAS, one may argue that the trajectory constituting a demand on the system needs not be very long because the application software may not have a long “memory.” Thus, the software may calculate a filtered input for each sensor using a few successive readings of input, and a rate of change using the current and preceding inputs. Reviewing the application software will verify this. To assume a short “memory” for the platform software and operating system may necessitate an evaluation based on their complexity and how often the system is reset. In common practice, the operating systems of safety-critical systems are simple; for example, the real-time operating system Kernel of TXS is only 1 k byte, and does not dynamically allocate resources [NRC 2000]. These features tend to make the “memory” short, supporting the hope that short trajectories can be used in testing. It is desirable that deterministic analysis be performed to determine that the platform software and operating system do not have a long memory. In the absence of this analysis, the issue of input trajectory length represents a potential limitation of the statistical testing approach.

The generation of test cases described in Section 6.2.3 attempts to account for the operational profiles of specific PRA contexts. It does not represent abnormal conditions caused by failures internal to the digital system. For example, a part of the software may include responses to internal hardware failures, e.g., detection of a failure of a CPU, implying that internal hardware failures be included in statistical testing. This may be considered as the internal states of the digital system. Accounting for internal hardware failures is related to how an overall reliability model of both the hardware and software of a digital system should be developed, a recognized open issue. Excluding such failures in statistical testing is a significant omission. A recent BNL study on a digital feedwater control system [Chu 2010] represents a possible way to encompass them. In that study, hardware failures of components internal to the digital system are modeled and define the additional contexts to which the system software would otherwise never be subjected.

The proposed statistical testing has to take into consideration many different conditions/contexts defined in a PRA, which may make the approach impractical. Section 6.3 proposes a feasibility study to estimate the number of test cases that need to be performed in order to demonstrate that the software’s contribution to core damage frequency is sufficiently small.

6.2.6 Treatment of Uncertainties

The Bayesian approach described in Section 6.2.4 automatically accounts for parameter uncertainty. The prior distribution may be derived from a BBN as described in Section 5, or by using a non-informative prior distribution. When using a non-informative distribution, guidance

on selecting a prior distribution detailed in the handbook for parameter estimation [Atwood 2002] can be used (i.e., use of the Jeffrey's prior). One key contributor to modeling uncertainty is the assumption on selecting a prior distribution. This uncertainty can be addressed by performing sensitivity calculations using different prior distributions.

Section 6.2.5 discussed the key issues and limitations in statistical testing, which contribute to modeling uncertainty. Presently, quantitative measures of their importance are difficult to develop, though their importance can be demonstrated qualitatively by examples, as discussed below.

This study recommended utilizing vendor-developed test tools consisting of (1) the actual protection system's hardware and software, (2) a test system supplying inputs to the system and evaluating its outputs, and (3) test inputs generated using a thermal-hydraulic simulator of the plant. Based on engineering knowledge, it is easy to conceive of possible failures that cannot be identified with less realistic test configurations. For example, if the system's actual hardware is not part of the software testing, then the interactions between hardware and software, including their timing, will not be realistically accounted for. In general, further research is needed to explore the effectiveness of different test configurations in representing the actual system in its operational environment.

Another unresolved problem is how to consider internal states to ensure that the operational profile is realistically represented. The Patriot missile failure [GAO 2002] is an example of failures related to internal states of digital systems. The applicability of the issue of the "memory" to a specific system possibly might be assessed by reviewing the application software, and evaluating the design of the software of the platform and operating system. Verifying an assumption of a short memory would allow shorter trajectories to be used in testing.

Failure to account for internal hardware failures would be an obvious error of omission. The issue should be evaluated by developing models of both the hardware and software of digital systems (e.g., the BNL model of a digital feedwater control system [Chu 2009]) and use the modeled hardware failures to help define the operational profile for testing software.

6.3 Demonstration of the Feasibility of Risk-informed Statistical Testing

One concern about the strategy for risk-informed statistical testing proposed in Section 6.2.2 is that a very large number of tests might be needed. To investigate this issue, a PRA model, for example, the PRA of one of the NUREG-1150 plants, could serve in determining the number of test cases without failure required to demonstrate that, for an RPS or ESFAS, the contribution of the system's software is limited to an acceptable threshold (e.g., is in accordance with the guidelines in RG 1.174). The following is a possible approach for determining the total number of test cases for a plant planning to replace its analog reactor protection system with a digital one.

1. Based on the total CDF of a plant PRA, determine a generally acceptable increase in total CDF due to software failure of the system according to RG 1.174 [NRC 2002] (taking into consideration cumulative impacts of previous changes to licensing basis). Identify the PRA sequences in which the system is failed due to failure of the actuation signal and determine their percentage contribution to the total CDF. Assuming the generally acceptable increase

in total CDF is due to these sequences and each of the sequences is allowed the same percentage increase in frequency, determine the acceptable increases in the sequence frequencies.

2. For each sequence in which the protection system is failed due to loss of the actuation signal, determine an upper bound on the mean software failure probability according to Section 6.2.2, such that the sequence frequency is below the acceptable value. Determine the number of tests without failure that would be needed to demonstrate that the estimated mean software failure probability is below the upper bound.

It should be noted that it might be plausible to optimize the application of the generally acceptable increase in total CDF to the sequences that involve protection system failure such that the total number of tests required is minimized. The premise would be that in sequences of lower frequency, the software failure probability would not have to be demonstrated as low as in higher frequency sequences. This can be explored as part of the planned case study.

3. The total number of tests is the sum of the number of tests over all sequences.

The above approach generally is also applicable at the cutset level (i.e., the cutsets in which the actuation signal is lost), if the PRA model of the original system is at the component level of the system, though the number of cutsets would be much larger than the number of sequences.

6.4 Evaluation of the Statistical Testing Approach Against the Desirable Characteristics

- (1) *The description of the method and its application is comprehensive and understandable.*

The black-box statistical testing approach employs standard statistical methods that are documented comprehensively, thereby meeting this characteristic.

- (2) *The assumptions of the method have reasonable bases.*

In general, the assumptions associated with the statistical testing approach have reasonable bases. However, there are some assumptions that are more difficult to justify. For example, the assumption that the test profile is the same as the operational profile is an inherent limitation common to all test-based methods, and cannot be easily demonstrated as otherwise.

- (3) *The method allows for consideration of the specific operating conditions of the software.*

The proposed risk-informed testing strategy of this study does take into consideration specific contexts/accident scenarios, thereby meeting this characteristic if the approach is determined to be practical.

- (4) *The method takes into consideration the quality of lifecycle activities.*

The test-based methods do not account for the quality of lifecycle activities.

- (5) *The method makes use of available test results and operational experience.*

The black-box test-based method uses standard statistical methods to quantify software reliability based on test and operational data, thereby meeting this characteristic.

- (6) *The method addresses uncertainty.*

The black-box test-based method uses standard statistical methods, including treating parameter uncertainty, to quantify software reliability, thereby meeting this characteristic.

- (7) *The method has been verified and validated.*

Because the current black-box method is theoretical and the few applications were explorative ones, the method is not considered substantiated. Conceptually, there is no reason why this method cannot be verified and validated. The successful application of the proposed statistical testing approach of this study to an example system would help demonstrate the method.

- (8) *The method is capable of demonstrating the high reliability of a safety-critical system.*

Automated testing may be a means for undertaking a large number of tests, thereby potentially allowing black-box methods to meet this characteristic. Also, the risk-informed testing strategy proposed in this study might show that high reliability may not be necessary in demonstrating that the software's contribution to the plant's core damage frequency is small enough. A feasibility study is recommended that estimates the total number of tests that must be performed to determine the feasibility of the approach.

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

It would be difficult to use black-box statistical testing to estimate software CCF parameters.

- (10) *The data and needed information exist and can be collected.*

The black-box testing method generates test data that is needed in statistical analysis satisfying this characteristic.

7. CONCLUSIONS AND INSIGHTS

In this study, the QSRMs reviewed in an earlier study [Chu 2010] were evaluated against the desirable characteristics, first developed therein and enhanced in this study by adding a characteristic on the availability of needed data, to identify candidate methods to apply in a case study. The candidate methods selected were the software reliability growth method (SRGM), the Bayesian belief network (BBN) method, and the statistical testing method. All three were developed further to evaluate their use in estimating the probability of failure-on-demand of the software of a protection system, including examining their issues and limitations.

Based on this extended work, the statistical testing method was deemed the preferred approach. However, facing the limitations of the statistical testing method, and to account for the quality of software-lifecycle activities, it was decided to first develop a prior distribution (using either the BBN or SRGM method), and then undertake a Bayesian update to this distribution using the results of statistical testing.

Using the SRGM approach for obtaining the prior distribution would entail formulating a discrete SRGM (DSRGM) to model the demand failure of the software. A critical limiting factor in using a DSRGM is that demand-based debugging test data from a safety-related system, such as a reactor protection system (RPS) are unlikely to be available; even if they were, it would be likely the data would be of insufficient quantity and not representative of the system's operating profile. Accordingly, the DSRGM approach will not be further pursued, and only the BBN approach will be used to obtain the prior distribution. Since development and quantification of a detailed BBN model can be very resource-intensive, and the desired evidence for quantification may be lacking, both a simplified and detailed BBN model will be developed. This exercise will provide valuable insight into the relative costs and benefits of developing a detailed model.

Further, as an additional method for comparison, a simple prior distribution will be estimated based on the Nuclear Regulatory Commission's (NRC's) guidance on parameter estimation [Atwood 2002], that is, using a Jeffrey's prior as the prior distribution, and then employing statistical testing results to perform a Bayesian update of this noninformative prior. A disadvantage with this approach is that the model does not account for the quality of software development.

The following subsections summarize the findings and insights.

7.1 Selection of Candidate QSRMs and Common Limitations

The evaluation of the QSRMs against the desirable characteristics revealed that none of the methods reviewed meets all of the characteristics, and no single method clearly stands out as the most appropriate. Based on the insights from Section 2.2, the BBN method and the test-based methods were selected as candidates for further assessment, mainly because of the former's ability to account for the quality of the software lifecycle activities and the latter's use of standard statistical methods, including treatment of parameter uncertainties. In addition, these methods appear to have the greatest potential for demonstrating high software reliability. (It should be noted that the Metrics-MTTF method, which also compares well to the characteristics, is a test-based method.) This finding is consistent with the results of the expert panel meeting on modeling software failures in PRA [Chu 2009b], where the panelists specifically identified

testing and BBNs as general methods with potential for quantifying software failure rates and probabilities. The SRGM/DSRGM approach was also initially selected as a candidate method in this study since it employs information collected during software development and is the most often used software reliability method.

In the evaluation of QSRMs, some limitations common to all of the reviewed methods were identified. They include

1. The test data used in the quantification may not truly represent the software's operational profile, thereby questioning the applicability of the estimated software reliability.
2. The methods quantify software reliability at the overall system level and do not account for the different contexts or boundary conditions in which the software must perform, nor differentiate between the different failure modes that the software may have.
3. It may be difficult to demonstrate with confidence the expected high reliability of digital systems.
4. No methods are available that assess the potential dependencies between redundant digital protection systems (i.e., between a primary and a backup system or between redundant channels that use diverse software).

Section 7.3 summarizes how the three candidate methods (DSRGM, BBN, and statistical testing) can be used in developing models for estimating the probability of failure on demand of a protection system, covering their associated issues and limitations and suggesting possible resolutions. As discussed previously, only the BBN and statistical testing methods will be applied in case studies.

7.2 Selection of an Example System

Since the objective of the study is to develop methods for quantifying the probability of failure on demand of digital protection systems at nuclear power plants (NPPs), it is preferable that the example used is such a system. To account for the quality of software development activities via the BBN method, it is desirable to obtain information about these activities, for example, verification and validation reports. In addition, it is desirable that the report produced for the case study is available publicly. Therefore, to the extent practical, it is best to base the study on public information (recognizing that the study can be documented, if necessary, so as to omit or mask proprietary information).

In searching for an example system satisfying the above characteristics, protection systems at NPPs first were considered, and then ruled out mainly due to expected difficulty in obtaining all of the necessary system information from the plant or vendor. A further hindrance was the proprietary nature of the supporting information and data, which also complicated attempts to identify a NASA system. Eventually, it became evident that it would be very difficult to identify an example system that meets all of the desired characteristics. Therefore, the search focused on systems that serve a protective "on-demand" function. The personnel-access control systems at some Brookhaven National Laboratory (BNL) accelerator facilities were investigated, but the documentation was inadequate. Eventually, the NRC obtained agreement from Idaho National Laboratory (INL) to supply information on a control system of a test facility of their Advanced Test Reactor. While this system is not strictly a protection system, part of it performs a protective function. It is expected that this system can be used as the example system in the case studies.

Due to the difficulties in finding an example system for the case studies, the system information did not become available until very late in the project. As such, the candidate QSRMs were illustrated without the benefit of this information.

7.3 Development of QSRMs for PRA Use

The following summarizes the findings and insights of this study on the three candidate methods.

Discrete Software Reliability Growth Method

In this study, an analytical framework was developed for estimating the probability of demand failure using DSRGMs based on previous studies [Shanthikumar 1983, Yamada 1985, Satoh 2001, Cai 2000]. A discrete NHPP first was defined. Drawing upon the parallelism between the continuous and discrete NHPP methods, many SRGMs can be transformed into DSRGMs by converting the differential equations of the former into difference equations of the latter. Thereafter, the model parameters can be solved with available data. A more straightforward way of obtaining a DSRGM is by simply replacing the time with the number of runs in a continuous-time SGRM [Littlewood 2011], because, for safety-related software, the probability of failure per demand is small, so a Bernoulli process will closely resemble a Poisson process.

A major limitation with employing DSRGMs is the dearth of debugging data needed in applying the methods. The developer may not record how many times the software was tested, nor how many failures have occurred. If records of testing are kept, but there are many occurrences of failure, it will be difficult to demonstrate the desired high reliability of the software for a safety-critical system. Another significant limitation is that to estimate accurately software failure probabilities, testing results based on test cases generated from the operational profile must be used. However, it is unlikely that the debugging test cases were based on the operational profile. Therefore, the estimated reliability of the software may be biased. Due to these reasons, DSRGMs are not considered suitable for estimating software-failure probabilities or for providing a prior distribution.

Bayesian Belief Network Method

This study formulated a process for developing a BBN model for the case study founded on existing BBN models for quantifying software reliability, as in Gran [2002a], Eom [2009], and Delic [1997]. Important issues and limitations requiring resolution were identified. Advantageously, the BBN method is able to combine disparate information, such as the quality of software development activities and statistical test data. However, this advantage is muted due to the lack of an established approach for converting the qualitative information from the quality of software development work into software reliability (i.e., probability of failure on demand). The existing methods used for this conversion (e.g., Gran [2002a] and [Delic 1997]) have severe limitations, essentially due to lack of data relating the quality of software development and the number of faults to reliability. This difficulty is somewhat similar to that associated with human reliability analysis (HRA), in which different Performance Shaping Factors affect human failure probability differently. Benchmark studies of HRA recently were performed (e.g., [Boring 2010]) to validate and compare the predictions of different HRA methods. For the BBN method, benchmark studies would involve collecting information on the

quality of software development activities of past projects, and collecting operational data or the performance of the statistical tests described in Section 6. The benchmark studies can validate or calibrate a BBN model used for quantifying software failure probability. The benchmarking can be undertaken both at the system level and the node probability table level. It is expected that vendors have collected operational data for the software they developed; however, most likely the information is proprietary and the benchmarking has to be done by the vendors.

The BBN method of Gran [2002a] is similar to the failure-likelihood-index method of human reliability analysis in that both convert qualitative information into probabilities when actual data are sparse, though they differ in their mathematical formats. In addition, the correlation method of “Frestimate” [Neufelder 2002] uses proprietary data on past software projects in a regression analysis to assess the number of faults left in a software program. Regression analysis is another mathematical representation that relates qualitative- to quantitative-information. In principle, if operating experience from past software projects is available, the regression analysis can be extended to directly estimating software-failure probability, instead of stopping at estimating the number of remaining faults. Again, a limitation of this is the paucity of data on past software projects (e.g., the information from “Frestimate” is proprietary), including operating experience.

To capture the quality of software-development activities, it is desirable to develop the BBN model using the structure of a standard on software development, for example, BTP-14 [USNRC 2007] for U.S. NPPs. Gran’s [2002a] and Eom’s [2009] models use standards DO-178B [RTCA 1999] and the Korea Nuclear Instrumentation and Control System (KNICS) procedure, respectively, and convert the requirements in the standards into the nodes of their models. Consequently, the BBN models are very complex, and many node-probability tables have to be populated with estimated values. In general, the possible states of the nodes and the relationships between them must be defined precisely, and, for consistency, guidance should be given on how to estimate the values to be input in the node-probability tables. A criticism of very detailed models is that they require detailed assessment where potentially little data or information is available [Bloomfield 2002]. However, since the standards are used in determining the acceptability of a piece of software, people involved in approving of the system (i.e., the staff of the plant, vendor, and regulator) should know how the software being evaluated meets the requirements. A simpler alternative would be to develop a higher-level model, for example, wherein only one or two nodes represent each stage of the software development cycle. One challenge with this alternative would be establishing the relationship between the simplified model and the associated standard. As discussed previously, in the case study a simplified and a detailed BBN model will be developed.

Statistical Testing Method

Statistical testing is the most practical approach for quantifying the probability of failure on demand of a protection system. In this study, risk-informed statistical testing is proposed. It employs the contexts/accident scenarios of a PRA model to define the operational/test profile from which test cases should be generated. Basic test inputs should be obtained in the form of a trajectory in the input space, i.e., a series of successive values for the input variables of a program occurring during the operation of the software, via a thermal hydraulic model of the plant that simulates accident conditions. The variability in the plant’s initial condition (e.g., small variations in physical conditions, such as power level) should be included. Additionally, the model should encompass the variability due to noise in the signals, and inaccuracies and time delays of the sensors. The most practical way of performing statistical testing is to use vendor-

supplied test systems because developing separate test equipment is too costly. A test configuration is recommended that uses the actual hardware as well as software (e.g., the ERBUS-based test configuration used in the Oconee digital upgrade).

The most significant limitation with statistical testing involves how realistically the tests represent the actual operational conditions of the software. For example, the operational profile of each sequence or cutset that requires the protective function may not be well known or characterized; further, the test configuration may not truly represent the configuration that may occur during operation. Capturing the internal state of the digital system is another important problem relating to how past inputs affect the system's behavior, and how the system's internal hardware failures are accounted for. For application software of protection systems at an NPP, one might argue the software's memory may not be very long, that is, the software may calculate a filtered input for each sensor via a few successive readings of input and a rate of change using current and preceding inputs. For platform software and for operating system software, an evaluation of the short "memory" assumption may be needed based on the software's complexity and how often the system is reset. The proposed statistical testing approach makes use of a thermal hydraulic model of the plant to simulate the test trajectories starting from an initial steady state condition. Using this approach, for many scenarios the trajectories may not be short, though there is no assurance that they are long enough to fully capture the software memory.

The proposed statistical testing approach does not account for abnormal conditions caused by the system's internal failures, for example, detecting a failure of a central processing unit. Considering internal hardware failures relates to how an overall reliability model of both the hardware and software of a digital system should be developed, recognizably an open issue. A recent BNL study on a digital feedwater control system [Chu 2009a] presents a possibly way to model internal hardware failures. Including such modeling would make the testing more complete, though adding to the resources needed to implement the tests.

Statistical testing of software typically involves testing the software over many possible inputs to it, defined by the software's operational profile. Since the number of inputs can be very large, testing the software in this way entails undertaking an extremely large number of tests, with the consequent costs in time, money, and effort. This is among the most difficult problems in assessing the probability of failure of a software program. The difficulty is exacerbated by a possible need to consider the impact of auxiliary function failures on software reliability, as discussed at the end of Section 5.2.2. This study proposes a risk-informed strategy for statistical tests that expectedly will resolve this issue. The strategy is similar to that used in a National Aeronautics and Space Administration (NASA) study [ASCA 2007], that is, essentially bounding the contribution of software failure to the risk of an NPP. The objective is to bound the maximum value of the probability of a software failure so that a risk or reliability goal can be met. If it can be shown that the maximum value is, say, 10^{-3} , to reach the goal, then software testing would only have to demonstrate this number, thus reducing (possibly substantially) the number of tests needed to be carried out. Accordingly, applying this strategy will only demonstrate whether the software reliability meets the established goal; it will not generate the exact or true failure probability. A procedure is offered in Section 6.3 for determining the total number of tests required to demonstrate the feasibility of the risk-informed statistical testing.

8. REFERENCES

- [AgenaRisk 2008] Agena Ltd., "AgenaRisk 5.0 User Manual," www.agenarisk.com, 2008.
- [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., et al., "Dynamic Reliability Modeling of Digital Instrumentation and Control Systems for Nuclear Reactor Probabilistic Risk Assessments," NUREG/CR-6942, October 2007.
- [Aldemir 2009] Aldemir, T., et al., "A Benchmark Implementation of Two Dynamic Methodologies for the Reliability Modeling of Digital Instrumentation and Control Systems," NUREG/CR-6985, February 2009.
- [AREVA 2009] AREVA, "SIVAT: TELEPERM XS™ Simulation Validation Test Tool," Topical Report, ANP-10303NP, Revision 0, June 2009.
- [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.
- [ASME NQA] American Society of Mechanical Engineers, "Quality Assurance Requirements for Nuclear Facility Applications," 2000.
- [Atwood 2002] Atwood, C.L., et al., "Handbook of Parameter Estimation for Probabilistic Risk Assessment," NUREG/CR-6823, November 2002.
- [Bensi 2010] Bensi, M.T., "A Bayesian Network Methodology for Infrastructure Seismic Risk Assessment and Decision Support," Ph.D. Dissertation, University of California, Berkeley, 2010.
- [Blaser 2009] Blaser, L., Ohrnberger et al., "Bayesian Belief Network for Tsunami Warning Decision Support," Proceedings of the 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, pp 757-768, 2009.
- [Bloomfield] Bloomfield, R., et al., Letter to the Editor, *Nuclear Engineering International*, January 2002.
- [Boring 2010] Boring, R.L., et al., "Lessons Learned on Benchmarking from the International Human Reliability Analysis Empirical Study," PSAM 10, Seattle, Washington, June 11-17, 2010.
- [Cai 2000] Cai, K.-Yuan, "Towards a Conceptual Framework of Software Run Reliability Modeling," *Information Sciences*, Vol. 126, pp. 137-163, 2000.

- [Chu 1995] Chu, T.L., et al., "Evaluation of Potential Severe Accidents during Low Power and Shutdown Operations at Surry, Unit 1," NUREG/CR-6144, Volume 2, October 1995.
- [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.
- [Chu 2010] Chu, T.L., et al., "Review Of Quantitative Software Reliability Methods," Brookhaven National Laboratory, BNL-94047-2010, September 2010.
- [Cooper 1996] Cooper, S. E., et al., "A Technique for Human Error Analysis (ATHEANA)," NUREG/CR-6350. U.S. Nuclear Regulatory Commission, Washington, D.C., May 1996.
- [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.
- [Delic 1997] Delic, K.A., Mazzanti, F., and Strigini, L., "Formalizing Engineering Judgment on Software Dependability via Belief Networks," Sixth IFIP International Working Conference on Dependable Computing for Critical Applications, "Can We Rely on Computers?" Garmisch-Partenkirchen, Germany, IEEE Computer Society Press, pp. 291-305, 1997.
- [Drouin 2009] Drouin, M., et al., "Guidance on the Treatment of Uncertainties Associated with PRAs in Risk-Informed Decision Making - Main Report," NUREG-1855, Vol. 1, March 2009.
- [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.

- [Fenton 2007] Fenton, N., et al., "Predicting Software Defects in Varying Development Lifecycles Using Bayesian Nets," *Information and Software Technology*, Vol. 49, pp. 32-43, 2007.
- [Fukumoto 1998] Fukumoto, A., et al., "A Verification and Validation Method and Its Application to Digital Safety Systems in ABWR Nuclear Power Plants," *Nuclear Engineering and Design*, Issue 183, pp. 117-132, 1998.
- [GAO 1992] General Accounting Office, Online Document at <http://www.fas.org/spp/starwars/gao/im92026.htm>, Feb. 4, 1992.
- [Garrett 1999] Garrett, C., and Apostolakis, G., "Context in the Risk Assessment of Digital Systems," *Risk Analysis*, Vol. 19, pp. 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 2000] Gran, B.A., and Dahll, G., "Estimating Dependability of Programmable Systems Using Bayesian Belief Nets," OECD Halden Reactor Project, HWR-627, May 2000.
- [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*, vol.40, pp. 797-812, 2002.
- [Grottke 2007] Grottke, M., and Trivedi, K., "Fighting bugs: Remove, Retry, Replicate, and Rejuvenate," *IEEE Computer*, 40(2):107–109, 2007.
- [Grover 2005] Grover, S.B., and Furstenau, R.V., "Capabilities and Facilities Available at the Advanced Test Reactor to Support Development of the Next Generation Reactors," *Proceedings of GLOBAL 2005*, Tsukuba, Japan, October 9-13, 2005.
- [Guarro 2007] Guarro, S., Dixon, S., and Yau, M., "Risk-Informed Safety Assurance and Probabilistic Risk Assessment of Mission-Critical Software-Intensive Systems," ASCA Inc., June 15, 2007.
- [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.
- [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.
- [Hamlet 1994] Hamlet, D., "Connecting Test Coverage to Software Dependability," 5th International Symposium on Software Reliability Engineering, 1994.
- [Hickman 1983] Hickman, J.W., et al., "PRA Procedures Guide, A guide to the Performance of Probabilistic Risk Assessments for Nuclear Power Plants," NUREG/CR 2300, Vol. 1 and 2, January 1983.
- [Horvitz 1998] Horvitz, E., et al., "The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users," *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, Madison, Wisconsin, July 1998.
- [IEC 1986] International Electrotechnical Commission, "Software for Computers in the Safety Systems of Nuclear Power Stations," IEC 880, First Edition, 1986.
- [IEC 61508] International Electrotechnical Commission, "Function Safety of Electrical/Electronic/Programmable Safety-Related Systems," Parts 1-7, IEC 61508, various dates.
- [IEEE 1987] IEEE, "IEEE Standard for Software Unit Testing," ANSI/IEEE Std 1008-1987.
- [IEEE 2008] Institute of Electrical and Electronics Engineers (IEEE), "IEEE Recommended Practice on Software Reliability," IEEE Standard 1633-2008, March 27, 2008.
- [INEEL] Idaho National Engineering and Environmental Laboratory, "User's Handbook for the Advanced Test Reactor."
- [INL 2008] Idaho National Laboratory (INL), "ATR Loop Operating Control System," System Design Description, SDD-7.9.20, Rev. 8, April 22, 2008.
- [Jensen 2002] Jensen, F.V., *Bayesian Networks and Decision Graphs*, Springer, 2002.
- [Johnson 2000] Johnson, G., and Yu, X., "Conceptual Software Reliability Prediction Models for Nuclear Power Plant Safety Systems," Lawrence Livermore National Laboratory, UCRL-ID-138577, April 3, 2000.
- [Kuball 2004] Kuball, S., et al., "The Effectiveness of Statistical Testing When Applied to Logic Systems," *Safety Science*, Vol. 42, pp. 369-383, 2004.

- [Kuball 2007] Kuball, S., and May, J.H.R., "A Discussion of Statistical Testing on A Safety-Related Application," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, vol. 221, no. 2, pp. 121-132, 2007.
- [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>).
- [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 (Methodological)*, Vol. 50, No. 2, pp. 157–224, 1988.
- [Lin 2008] Lin, S.W., and Bier, V.M., "A Study of Expert Overconfidence," *Reliability Engineering and System Safety*, Vol. 93, pp. 711–721, 2008.
- [Littlewood 1997] Littlewood, B., and Strigini, L., "Guidelines for Statistical Testing," Center for Software Reliability, City University London, PASCON/WO6-CCN2/TN12, October 8, 1997.
- [Littlewood 2000] Littlewood, B., and Strigini, L., "Software Reliability and Dependability: A Roadmap," International Conference on Software Engineering, *Proceedings of the Conference on The Future of Software Engineering*, Limerick, Ireland, June 2000.
- [Littlewood 2007a] 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.
- [Littlewood 2007b] Littlewood, B., and Wright, D., "Some Conservative Stopping Rules for the Operational Testing of Safet-Critical Software," *IEEE Transactions on Software Engineering*, Vol. 23, No. 11, November 1997.
- [Littlewood 2010] Littlewood, B., and Rushby, J., "Reasoning about the Reliability of Diverse Two-Channel Systems in Which One Channel Is "Possibly Perfect," SRI International, Technical Report SRI-CSL-09-02, updated January 2010.
- [Littlewood 2011] Littlewood, B., Email to Chu, T. L., dated January 25, 2011.
- [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.
- [LWP-10400] Idaho National Laboratory, "Laboratory-Wide Procedure for Design Control."
- [LWP-13014] Idaho National Laboratory, "Laboratory-Wide Procedure for Determining Quality Levels."

- [LWP-13620] Idaho National Laboratory, "Laboratory-Wide Procedure for Software Quality Assurance."
- [Martz 1982] Martz, H. F., and Waller, R.A., *Bayesian Reliability Analysis*, John Wiley & Sons, Inc., 1982.
- [May 1995] May, J., Hughes, G., and Lunn, A.D., "Reliability Estimation from Appropriate Testing of Plant Protection Software," *Software Engineering Journal*, November 1995.
- [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.
- [Musa 1987] Musa, J.D., Iannino, A., and Okumoto, K., *Software Reliability: Measurement, Prediction, Application*, New York: McGraw-Hill, 1987.
- [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.
- [Neufelder 2002] Neufelder, A.M., "The Facts about Predicting Software Defects and Reliability," *The Journal of the Reliability Analysis Center (RAC)*, Second Quarter – 2002.
- [NRC 1990] United States Nuclear Regulatory Commission (USNRC), "Severe Accident Risks: An Assessment for Five U.S. Nuclear Power Plants," NUREG-1150, Volumes 1 and 2, December 1990.
- [NRC 1995] USNRC, "Use of Probabilistic Risk Assessment Methods in Nuclear Regulatory Activities," Final Policy Statement, August 16, 1995.
- [NRC 1997] USNRC, "Developing Software Life Cycle Processes for Digital Computer Software Used in Safety Systems of Nuclear Power Plants," Regulatory Guide 1.173, Revision 1, September 1997.
- [NRC 2000] USNRC, "Safety Evaluation by the Office of Nuclear Reactor Regulation Siemens Power Corporation Topical Report EMF-2110(NP), 'TELEPERM XS: A Digital Reactor Protection System', Project No. 702," May 4, 2000.
- [NRC 2001] USNRC, "NRC Research Plan for Digital Instrumentation and Control," SECY-01-0155, August 15, 2001.
- [NRC 2002] USNRC, "An Approach for Using Probabilistic Risk Assessment in Risk-Informed Decisions on Plant-Specific Changes to the Licensing Basis," Regulatory Guide 1.174, Revision 1, November 2002.
- [NRC 2007] USNRC, "Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Systems," NUREG-800, Standard Review Plan, Branch Technical Position 7-14, Revision 5, March 2007.

- [NRC 2009] USNRC, "Wolf Creek Generating Station Amendment to Renewed Facility Operating License," Docket No. 50-482, Amendment No. 181, March 15, 2009.
- [NRC 2010] USNRC, "Duke Energy Carolinas, LLC Docket No. 50-269, Oconee Nuclear Station, Unit 1, Amendment To Renewed Facility Operating License, Amendment 1\10. 366, Renewed License No. DPR-38," January 28, 2010.
- [Okamura 2004] Okamura, H., Murayama, A., and Dohi, T., "EM Algorithm for Discrete Software Reliability Models: A Unified Parameter Estimation Method," Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE'04), 2004.
- [Pan 2006] Pan, R., Peng, Y., and Ding, Z., "Belief Update in Bayesian Networks Using Uncertain Evidence," *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2006)*, November 6, 2006.
- [PG-T-94] TRA Construction Projects Management, "Software Verification and Validation Report, ATR Loop Operating Control System (LOCS) Upgrade Project," Revision 0.0, 1997.
- [PLN-2293] Idaho National Laboratory, "Configuration Management Plan for ATR Distributed Control Systems."
- [PLN-2766] Idaho National Laboratory, "Software Quality Assurance Plan (SQAP): ATR Loop Distributed Control System Upgrade," Revision 0, 2009.
- [PLN-2768] Idaho National Laboratory, "Verification and Validation (V&V) Plan: ATR Loop Distributed Control System Upgrade," Revision 1, 2009.
- [PLN-3137] Idaho National Laboratory, "Configuration Management Plan Advanced Test Reactor Loop Operating Control System Distributed Control System," Revision 0, 2009.
- [Regamey 2006] Gret-Regamey, A. and Straub, D., "Spatially Explicit Avalanche Risk Assessment Linking Bayesian Networks to a GIS," *Natural Hazards and Earth System Sciences*, Vol. 6, pp 911-926, 2006.
- [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.
- [SAR-153] Idaho National Laboratory, "Upgraded Final Safety Analysis Report for the Advanced Test Reactor."
- [Satoh 2001] Satoh, D., and Yamada, S., "Discrete Equations and Software Reliability Growth Models," *Proceedings of 12th International Symposium on Software Reliability Engineering (ISSRE'01)*, pp. 176-184, 2001.

- [Shanthikumar 1983] Shanthikuma, J.G., "Software Reliability Models: A Review," *Microelectronics Reliability*, Vol. 23, No. 5, pp. 903-943, 1983.
- [Siemens 1999] Siemens, "TELEPERM XS: A Digital Reactor Protection System," EMF-2110(NP) (A), Revision 1, September 1999.
- [Smidts 2004] Smidts, C.S. and Li, M., "Preliminary Validation of a Methodology for Assessing Software Quality," NUREG/CR-6848, July 2004.
- [Tregoining 2008] Tregoining, R., et al., "Estimating Loss-of-Coolant Accident (LOCA) Frequencies Through the Elicitation Process," NUREG-1829, Volume 1, April 2008.
- [Weiss 1988] Weiss, S.N., and Weyuker, E.J., "An Extended Domain-Based Model of Software Reliability," *IEEE Transactions on Software Engineering*, Vol. 14, No. 10, October 1988.
- [Weiss 1989] Weiss, S.N., "What to Compare When Comparing Test Data Adequacy Criteria," *Association of Computing Machinery (ACM) Special Interest Group on Software Engineering (SIGSOFT), Software Engineering Notes*, Vol. 14, No. 6, page 42, October 1989.
- [Wood 2009] Wood, R.T., et al., "Diversity Strategies for Nuclear Power Plant Instrumentation and Control Systems," NUREG/CR-7007, February 2010.
- [Yamada 1985] Yamada, S., and Osaki, S., "Discrete Software Reliability Growth Models," *Applied Stochastic Models and Data Analysis*, Vol. 1, pp. 65-77, 1985.
- [Yang 2010] Yang Y., and Sydnor, R., "Multi-thread Software Reliability Estimation Based on Test Results and Software Structure," PSAM 2010, Seattle Washington, June 7-11, 2010.
- [Zhang 2004] Zhang, Y., "Reliability Quantification of Nuclear Safety-Related Software," Ph. D. Thesis, Department of Nuclear Engineering, Massachusetts Institute of Technology, February 2004.

Appendix A

DEVELOPMENT OF A FAILURE-ON-DEMAND BASED SOFTWARE RELIABILITY GROWTH METHOD

A.1 Introduction

In probabilistic risk assessment (PRA) studies of nuclear power plants (NPPs), the failure rate and the demand-failure probability, respectively, are generally used to characterize the reliability in performing the intended functions of a control system and a protection system. The continuous-time non-homogenous Poisson process (NHPP) based software reliability growth models¹¹ (SRGMs) [Chu 2010] can be used directly to calculate the failure rate of software, using test results that usually are recorded as the number of failures within a specified time (i.e., the software's execution time). However, software testing often involves counting the number of errors or failures that occurred and repairing them at the end of each test run. This is particularly true for testing safety-related-system software. When doing so (e.g., when testing reactor protection system [RPS] software), a number of trials are performed by challenging the software using demand-to-trip as input. Fault(s) may or may not manifest in each trial. At the end of each, the failure(s) are fixed by removing the fault(s) that caused them (i.e., each failure can only occur once). SRGMs assume that the failure probability of the software decreases with the number of test runs due to a decline in the remaining number of faults in the software. The debugging data usually is recorded as the number of successful runs between failures, or the number of failures in a certain number of runs. A continuous-time SRGM cannot be applied directly to this type of data, although the SRGM could be extended to cope with such testing results.

On the other hand, discrete models (e.g., Yamada [1985]) are expected to be more straightforward in their application to debugging data for safety system software. A discrete non-homogenous Poisson process (D-NHPP) can naturally model the fault manifesting/removal process in testing safety-related software. Unlike the continuous SRGMs that calculate the software failure rate, the result from a discrete SRGM (DSRGM) is the failure probability per demand, which is the parameter of most interest for an RPS (and most other safety systems) in PRA applications.

There are a few discrete-time software reliability growth models [Shanthikumar 1983] for estimating software reliability defined as the probability that a software program is successful in a number of tests. Yamada and Osaki [Yamada 1985] developed the first D-NHPP model to overcome the issue of the meaning of time in a failure-rate based software reliability growth model. They pointed out that the appropriate unit of the software failure detection period sometimes is the number of test runs or the number of executed test cases, other than the software's execution time or calendar time. A D-NHPP model is characterized by its probability mass function (pmf). Yamada and Osaki [Yamada 1985] also demonstrated that the pmf can be derived by assuming the expected number of failures experienced per test run is proportional to the current number of faults, and the failure occurrence rate is constant. Satoh and Yamada

¹¹ Many of the references in this appendix 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.

[Satoh 2001] obtained the same pmf by discretizing the differential equation for the mean value function of the Goel and Okumoto NHPP model [Goel 1979].

Cai [2000] introduced “run reliability” that is equivalent to $(1 - \text{the probability of failure on demand})$, and developed the full mathematical notations for the D-NHPP. Similar to other authors, [Satoh 2001, Huang 2003], Cai showed that many NHPP models are convertible into D-NHPP models.

Okamura et al. [2004] analytically demonstrated that a general ordered-statistics model [Miller 1986] and a cumulative binomial trial model [Hoare 1983, Dohi 2003] are D-NHPP models with derived probability mass functions. In addition, Okamura et al. [2004] also indicated that the discrete Jelinski and Moranda model [Jelinski 1972] is the same as the discrete Goel and Okumoto model [Goel 1979].

This appendix summarizes these studies and attempts to present systematically the D-NHPP based reliability growth methods, so that they can be used in estimating the probability of failure-on-demand of a protection system. The D-NHPP first is introduced, and, from it, the probability of software-demand failure is defined in Section A.2.1. In Section A.2.2, a cumulative Bernoulli trial (CBT) process is employed to model the software debugging process in terms of detecting and removing faults at the end of each test run. Further, it is shown that a D-NHPP will characterize such a CBT model if the initial number of faults has a Poisson distribution. In general, the CBT model considers multiple observation points during testing and, hence, is “multi-term.” However, by selecting a different form of the per fault triggering probability, it reduces to the so-called “single-term” that is derived directly based on the General Order Statistics (GOS) method, as indicated in Section A.2.3. A more straightforward way of obtaining the D-NHPP-based SRGMs is to discretize the continuous counterparts, as briefly discussed in Section A.2.4. Section A.2.5 gives an approximate way of calculating demand failure probability. A discussion of the parallel development of the continuous and discrete time SRGMs presented in Section A.2.6 provides more insights on the SRGMs. The issues and questions related to applying DSRGMs are discussed in Section A.2.7. Estimating parameters and modeling their uncertainties are discussed in Section A.3, while in Section A.4 application of a simple example of a D-NHPP model is illustrated.

Note, the D-NHPP study performed here does not endorse its applicability for estimating software reliability, which is mainly limited by the unavailability of appropriate failure data on safety-related software. This, however, does not prevent applying it to a piece of software that does not have a very high reliability requirement. The intention is to use several D-SRGMs to capture the variability of software reliability among different SRGM models, based on the same set of debugging data. A set of point-estimates of the predicted failure probabilities can be obtained to generate a prior distribution of the software’s demand-failure probability, as a means to addressing the uncertainty in modeling.

A.2 D-NHPP-Based Software Reliability Modeling

A.2.1 Discrete Non-Homogenous Poisson Process

Yamada and Osaki [1985] first introduced the discrete non-homogenous Poisson process (D-NHPP) based on the number of failures in a sequence of tests without considering the times the tests were performed. Their intent was to overcome the issue of those SRGMs selecting the

calendar time or execution time for estimating software failure rates (i.e., sometimes, it may not be possible to collect data on execution time, and calendar time then must be used to estimate the execution time).

The D-NHPP is defined here, and its use in modeling software failure on demand is formulated.

Let $\{N_i, i = 0, 1, 2, \dots\}$ denote a discrete counting process with N_i representing the cumulative number of software faults manifested by the i^{th} test run ($i = 0, 1, 2, \dots$) which is a random variable. Suppose that the counting process follows

$$(1) N_0 = 0,$$

(2) $\{N_i, i = 0, 1, 2, \dots\}$ has independent increments, that is, for any collection of the numbers of test runs i_1, i_2, \dots, i_k ($0 < i_1 < i_2 < \dots < i_k$), k random variables $\{N_{i_1}, N_{i_2} - N_{i_1}, \dots, N_{i_k} - N_{i_{k-1}}\}$ are statistically independent from each other, and,

(3) For any of the numbers of test runs, i_s and i_k , ($0 < i_s < i_k$), $s < k$,

$\Pr\{N_{i_k} - N_{i_s} = n\} = \frac{(\Lambda_{i_k} - \Lambda_{i_s})^n}{n!} e^{-(\Lambda_{i_k} - \Lambda_{i_s})}$, where $\Lambda_i = E\{N_i\}$, the mean value function, represents the mean value of accumulated number of failures up to the i th test run.

Then, the counting process in discrete time, $\{N_i, i = 0, 1, 2, \dots\}$, is called a discrete NHPP, with the probability mass function given by

$$\Pr\{N_i = n\} = \frac{\Lambda_i^n}{n!} e^{-\Lambda_i}, n = 0, 1, 2, \dots, \quad (\text{A-1})$$

For each discrete NHPP model, there is a mean value function defined as the expected number of failures up to a given number of test runs. Each mean value function Λ_i is specifically determined by an individual D-NHPP model, similar to cases of continuous NHPP [Xie 1991].

A D-NHPP is obtainable directly from the continuous NHPP by replacing the time variables with the numbers of test runs. That is, the number of failures occurring within the interval taken to finish any test run follows a Poisson distribution. The D-NHPP is simply a change of notation of NHPP, i.e., with test runs corresponding to the time at which the failures are counted. As indicated in Figure A-1, t_s , t_k , and t_l respectively correspond to the time at which the i_s^{th} , i_k^{th} , and i_l^{th} test runs are finished, that is, the end of these test runs.

Based on the definition for D-NHPP, assuming that i_F tests were run and will be released, the probability of software failure per demand after its release can be defined as

$$\sum_{n=1}^{\Lambda_{\infty} - \Lambda_{i_F}} \Pr\{N_{i_F+1} - N_{i_F} = n\} = \sum_{n=1}^{\Lambda_{\infty} - \Lambda_{i_F}} \frac{(\Lambda_{i_F+1} - \Lambda_{i_F})^n}{n!} e^{-(\Lambda_{i_F+1} - \Lambda_{i_F})} \quad (\text{A-2})$$

where Λ_{∞} represents the initial number of hidden faults in the software.

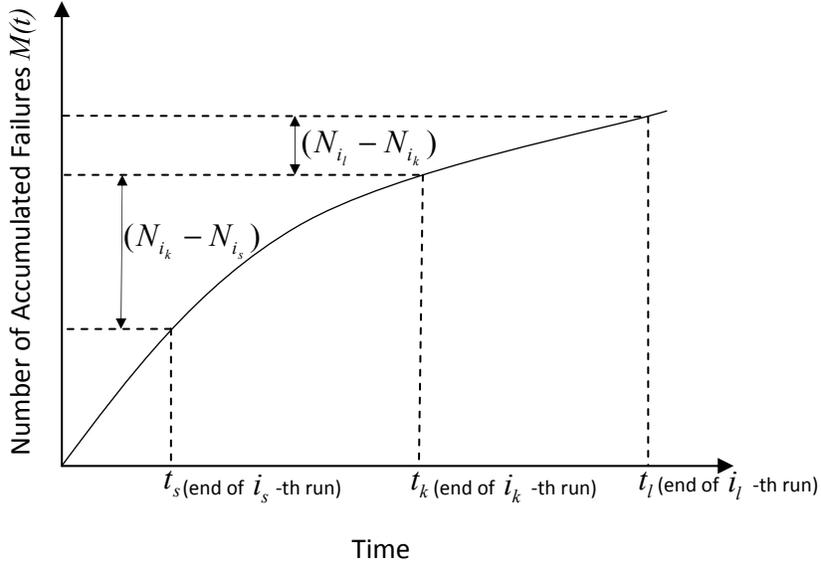


Figure A-1: Accumulated Failure Number vs. Continuous Time and Discrete Time

A.2.2 A Cumulative Bernoulli Trial Process for Modeling Software Testing/ Debugging

A concept of cumulative Bernoulli trials (CBTs) was introduced in Hoare [1983] and used to model the probability of an accumulated number of successes in a certain number of throws of dice, wherein the successful dice are discarded. Assuming a total number of N dice, and defining a success as the outcome of a certain score with a probability α , e.g., $\alpha = \frac{1}{6}$, then the probability that k successes were achieved is represented by a binomial distribution,

$$B(i, N, \alpha) = \binom{N}{i} \alpha^i (1 - \alpha)^{N-i}.$$

Since a successful die is discarded, k dice will be removed. In the second throwing, the distribution function for an outcome of achieving a cumulative number of i successes is given as (the success probability of each additional throw is assumed to be β that may differ from α)

$$\begin{aligned}
B^{(2)}(i, N; \alpha, \beta) &= \sum_{k=0}^i B(k, N; \alpha) B(i-k, N-k, \beta) \\
&= B(k, N; 1 - (1 - \alpha)(1 - \beta))
\end{aligned}$$

More generally, the probability of a total number of i successes in N throwings is given by

$B^{(n)}(i, N; \alpha_1, \alpha_2, \dots, \alpha_n) = B(i, N, 1 - \prod_{v=1}^n (1 - \alpha_v))$, where α_v , $v = 1, 2, \dots, n$, represents the success probability in each attempt [Hoare 1983].

Reversing success and failure in the dice-throwing process conceptually is the same as the software testing/debugging process. Of interest in software testing is the probability of the accumulated number of failures. In each test run, failure(s) may occur; they are assumed to be fixed immediately and perfectly and will not occur again. This is similar to the gambling process where the dice with success are discarded. This result was applied directly to modeling the software debugging process in [Okamura 2004 and Shibata 2006].

A software failure is due to the triggering of hidden fault(s) in the software. Due to the dearth of knowledge of the impacts of, and dependency between, software faults, it generally is assumed that software failures occur at independent, identically distributed random times, that is, all random variables are mutually independent and each random variable has the same probability distribution as the others (Assumption A1). It also is assumed that the initial number of software faults, N , that is, the total number of the failure occurrences when time tends to infinity, is finite (Assumption A2).

Let random variable X_i denote the number of faults that are triggered or are manifest in the i^{th} software testing run, and r_i denote the triggering probability of a fault in the i^{th} run (i.e., the per fault triggering probability). Then, the pmf of the CBT process is given in exactly the same way as in [Hoare 1983, Okamura 2004, and Shibata 2006], that is,

$$\Pr\{N_i = n\} = B(n; N, 1 - \prod_{k=1}^i (1 - r_k)) \quad (\text{A-3})$$

where r_k denotes the triggering probability of a software fault in the k^{th} test run, and

$N_i = \sum_{j=1}^i X_j$ denotes the cumulative total number of failures up to the i^{th} test run.

Note, Equation (A-3) based on the CBT modeling is called a multi-termed model in [Shibata 2006] because, in this way, r_k , $k = 1, 2, \dots, i$ is explicitly considered at multiple observation points of the testing process. A binomial process modeling of software testing produces a single-term model, as derived from order statistics at a single observation point [Okamura 2004], where the pmf is given by simply lumping the triggering probability of a fault in each test run, that is, $\Pr\{N_i = n\} = B(n; N, \sum_{k=1}^i r_k)$. As is shown in Section A.2.3, by selecting a specific

form for the triggering probability of a fault, r_k , the multi-term model can be reduced to the single-term one.

With the additional assumption (Assumption A3) that the initial number of software faults is Poisson distributed with a parameter ω , that is, $\Pr\{N = x\} = \frac{\omega^x}{x!} e^{-\omega}$, Equation (A-3) can be rewritten as

$$\begin{aligned} \Pr\{N_i = n\} &= \sum_{x=0}^{\infty} B(n; x, 1 - \prod_{k=1}^i (1 - r_k)) \times \frac{\omega^x}{x!} e^{-\omega} \\ &= \frac{(\omega [1 - \prod_{k=1}^i (1 - r_k)])^n}{n!} e^{-\omega [1 - \prod_{k=1}^i (1 - r_k)]} \end{aligned} \quad (\text{A-4})$$

Equation (A-4) shows that under assumptions (A1) - (A3), method DSRGM is, in fact, characterized by the D-NHPP in Equation (A-1) with the mean value function $\omega [1 - \prod_{k=1}^i (1 - r_k)]$, which parallels the representation of a continuous SRGM by the continuous time NHPP. Note, A1 and A2 are very basic assumptions that were in many continuous-time SGRMs, such as the Poisson-type and binomial-type models, where both are necessary for showing that, under Assumption A3, a Poisson-type model becomes equivalent to a binomial-type one in the continuous-time domain (see [Musa 1987] for more details). Assumptions A1 and A2 also are employed here for the same reason in the discrete-time domain. Note, this does not mean that this study endorses the binomial model or the CBT model. The parallelism between continuous and discrete NHPPs is further detailed in Section A.2.6.

A.2.3 Selection of Per Fault Triggering Probability

The per fault triggering probability may take different forms, that is, an unconditional probability or conditional probability, as discussed below. [Shibata 2006] showed that selecting the conditional per fault triggering probability reduces the multi-term D-NHPP Equation (A-4) model to a single-term one.

For a continuous NHPP model, a per-fault hazard rate is defined in [Musa 1987] as the contribution made by each fault in a program to the overall program's failure rate, assuming that all faults contribute equally. For a D-NHPP model, a discrete hazard function λ_k is defined similarly, that is,

$$\lambda_k = \Pr\{T_a = k \mid T_a \leq k - 1\} = \frac{\Pr\{T_a = k\}}{1 - \Pr\{T_a \leq k - 1\}} \quad (\text{A-5})$$

where T_a indicates the mean number of test runs it takes for a fault to manifest. λ_k represents the conditional triggering probability of a single fault in the k^{th} run that did not manifest before the k^{th} run.

Under assumption A1, let $P(i)$ indicate the triggering probability of a fault by the i^{th} test run.

The density function of $P(i)$ is denoted as p_k and $P(i) = \sum_{k=1}^i p_k$. Therefore, the physical meaning of p_k is that it denotes the (unconditional) triggering probability of a fault in the k^{th} test run that did not manifest previous to the k^{th} one, that is, $p_k = \Pr\{T_a = k\}$, while implicitly assuming that the fault will be removed after its manifesting, and therefore, manifests only once.

A natural way of selecting the per fault triggering probability is to let $r_k = \lambda_k$ in Equation (A-4). As shown in [Shibata 2006], if the per fault hazard function λ_k (the conditional probability) is chosen as r_k , the multi-term model [Equation (A-4)] becomes

$$\Pr\{N_i = n\} = \frac{(\omega P(i))^n}{n!} e^{-\omega P(i)} = \frac{(\omega \sum_{k=1}^i p_k)^n}{n!} e^{-\omega \sum_{k=1}^i p_k} \quad (\text{A-6})$$

and is reduced to a single-term model [Shibata 2006]. Note, Equation (A-6) still is a D-NHPP model. This is equivalent to representing the pmf of the accumulated number of faults manifesting up to the i^{th} test run using a binominal distribution $\Pr\{N_i = n \mid N\} = B(n, N, P(i)) = \binom{N}{n} P^n(i) (1 - P(i))^{N-n}$ with assumption A3, derived via the General Order Statistics (GOS) method [Okamura 2004].

On the other hand, Equation (A-5) can be rewritten as $\lambda_k = \frac{p_k}{1 - P_{k-1}}$. Therefore,

$$p_k = \lambda_k (1 - P(k-1)) = \lambda_k \prod_{j=1}^{k-1} (1 - \lambda_j) \quad (\text{A-7})$$

If we select $r_k = p_k$ (i.e., the unconditional probability) in Equation (A-4), then the multi-term model (A-4) is retained [Shibata 2006].

One of the three assumptions made in Section A.2.2, Assumption A1, frequently is criticized for not being realistic because, intuitively, all hidden faults of a software program should not contribute equally to the probability of software failure occurrence. A class of continuous-time SRGMs does not rely on this assumption, and these SRGMs generate a continuously decreasing failure rate, for example, Duane types of models [Duane 1964] and a model developed in [Littlewood 1981]. In the latter, each of the remaining faults was assumed to have

the same contribution to software failure. However, the failure rate distribution of the remaining faults is updated based on the observation of manifested faults, that is, the failure rates of the remaining faults tend to decrease compared to those of the faults that manifested earlier. As discussed below, such continuous-time models easily are discretized to obtain the discrete version.

A.2.4 Discretization of Continuous NHPP Models

A discrete counterpart of a continuous NHPP SRGM that is founded upon assumptions different from those adopted in the previous section is obtained by discretizing it based on the concept of discrete NHPPs. This is a straightforward way of generating discrete SRGMs by replacing with difference equations the differential equations characterizing the continuous SGRMs. The pmf of a discrete NHPP model then is obtained by solving the resulting difference equations for the mean value function of the accumulated number of failures [Okamura 2004]. A difference equation becomes a differential one when the time interval approaches zero, and therefore, preserves the characteristics of the differential equation.

More examples of obtaining discrete software reliability growth versions by discretizing the continuous counterparts are given in [Sato 2001, Huang 2003, and Cai 2000]. This approach aims at obtaining the mean value function mathematically and does not explain the underlying physical process of software debugging (or that of reliability growth), while the CBT or the binomial process model provides more insight into the D-NHPP-based SRGMs, as is discussed in Section A.2.6.

A more straightforward option of discretizing the continuous-time SGRMs is by simply replacing the time with the number of runs [Littlewood 2011] because, for a safety-related software, the probability of failure per demand is small, so a Bernoulli process will closely resemble a Poisson process. The advantage of doing this is that a spectrum of continuous-time SRGMs and their associated development, including how to select the best models according to their prediction accuracy, which is nonexistent for discrete models, is completely transplanted to the discrete-time domain.

A.2.5 Approximation of Demand Failure Probability

The current D-NHPP models were not developed specifically to estimate software failure probability on demand (except maybe Cai's conceptual work [2000]), but can be modified or interpreted to become a failure-on-demand model. This is accomplished by recognizing that the rate of failure occurrence (equation 3 of [Okamura 2004] without the assumption of constant rate of failure occurrence) of a D-NHPP is the probability of failure-on-demand on a per fault basis. To calculate the failure probability per demand, software failure can be defined as at least one failure occurring whilst assuming that failure occurrences are independent, as shown in Equation (A-2). Calculating the demand failure probability can be complicated using Equation (A-2). An approximation of this probability is proposed here.

As testing progresses, manifested faults are removed and the number of remaining faults declines, lowering the failure probability of the software. If a total number of $(i - 1)$ test runs have been performed, the triggered faults removed, and a decision is made to release the software, then r_i , the triggering probability of a single fault in the i th run, becomes the per fault demand failure probability of the software that was released and is running. The software's

demand failure probability thus can be approximated by multiplying the number of remaining faults in the software after it is released (estimated by subtracting the number of triggered faults from the total number of faults in the software) by the per fault triggering probability of software, that is, r_k in Equation (A-4).

A.2.6 More Insights on Continuous and Discrete NHPP SRGMs

A homogeneous Poisson process is a counting process with a constant occurrence rate. For failure processes with non-constant failure rates, a non-homogeneous Poisson process is applicable, as assumed in the continuous-time SRGMs. A discrete NHPP serves as a discrete counterpart of the continuous NHPP for accommodating the debugging data (e.g., of an RPS software) recorded in the format of the number of successful runs between failures, or as the number of failures in a certain number of runs, referred as Type I and Type II data, respectively, in [Cai 2000]. A continuous or discrete NHPP is characterized by the expected number of observed failures (mean value function). Note that how the expected number of failures changes as the testing progresses is defined by individual SRGMs, based on dissimilar assumptions about fault detection and removal, for example, the assumption that the rate of failure occurrence is proportional to the remaining fault content in the continuous time Goel and Okumoto model [Goel 1979]. Therefore, in the continuous time domain, the change of expected failure number (or failure rate decrease) is governed by assumption-based differential equations. In general, each continuous time model can be discretized and represented by a difference equation. Then, the expected failure number is obtained by solving the difference equation and using it in the D-NHPP model to generate the probability mass function.

The parallel paths in Figure A-2 are depicted clearly in deriving the continuous and discrete NHPP models. This offers more insights on how the continuous- and the discrete-time NHPPs are related. In the continuous time domain, the software debugging process can be described by an NHPP with the number of failures in a time interval following a Poisson distribution; while in the discrete time domain, a Bernoulli trial process is suitable for modeling the outcomes of test runs of the software (i.e., a success or failure), and the number of experienced (and removed) failures after a certain number of test runs should obey a binomial distribution. Referring to the dashed rectangle on the left in Figure A-2, Okamura [2004] demonstrated that representing the debugging process with a cumulative Bernoulli trial process or a binomial process (Box D1) leads to a discrete NHPP model, in parallel to the Poisson process representation of software debugging (in continuous time domain) (Box C1) leading to a continuous time NHPP model.

Another parallel path between the continuous and discrete time NHPPs can also be observed in the dashed rectangle on the right in Figure A-2. In continuous time domain, the NHPP model (the Poisson type model) obtained based on certain assumptions [Musa 1987] can be equivalently derived by using a binomial distribution to represent the number of failures occurred in a time interval (the binomial type model) under the assumption of a Poisson distribution for the initial number of faults in the software (Box C2). A discrete time NHPP model can be derived in exactly the same manner [Okamura 2004], i.e., assuming the number of failures in a number of tests is binomial distributed, and the only difference is that the failure probability at a specific run is used in the derivation instead of hazard rate or failure rate at a specific time of the testing (Box D2).

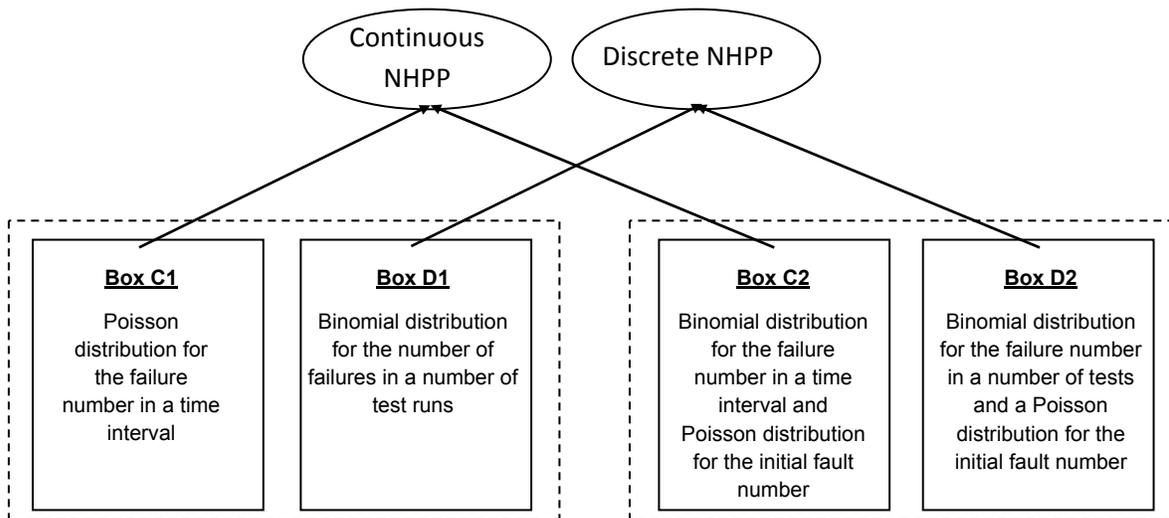


Figure A-2: Parallel Development of Continuous and Discrete NHPP Models

A.2.7 Issues and Questions for DSRGMs

Since SRGMs use test data, some issues about testing discussed in Section 6 are relevant, and are described briefly here. They are unique to SRGMs.

To predict the software reliability via an SRGM, the debugging data must be based on the execution of the software with operational representative inputs (e.g., for an RPS software, such inputs are generated from the NPP's responses to individual PRA initiating events that trigger the RPS to trip the NPP). It also is recognized that the debugging data must encompass several failure records so to demonstrate the growth in reliability of the software as debugging progresses. The major issues related to these two requirements, which drastically limit the application of SRGMs to safety-related software, are elaborated below.

1. One concern is that software vendors are unlikely to perform debugging using operationally representative inputs. Therefore, the software test profile will not match its operational profile. In addition, some test inputs may be unrealistic or even represent conditions that are not possible during NPP operation. Such testing results cannot be used to predict software reliability using an SRGM or any other software reliability quantification method.
2. The second concern is that if the vendors do perform testing using operational profile generated inputs (at least partially), the documentation of the results may not relate the test outcomes to the inputs caused by specific initiating events, or may mix the test outcomes of operationally representative inputs and other inputs. In such situations, it will not be possible to estimate context-specific (e.g., initiating-event-specific) software failure probabilities.
3. The third concern is that even if the vendors perform the testing and document the results appropriately, there may not be sufficient success records for exercising SRGMs to show the anticipated reliability growth. In addition, if there are a few failure records in the debugging data, the predicted reliability by using SRGMs will not be very high, which may not be acceptable for a

safety-critical system such as an RPS. In addition, to estimate parameter uncertainty using the maximum likelihood estimation (MLE), as is discussed in Section A.3, it is difficult to obtain sufficient failure data to assure that the parameter uncertainty converges asymptotically.

4. The acquisition of testing data always is problematic. It invariably is difficult to acquire testing data from a real system. For a proof-of-concept study, it might be sufficient to artificially generate some testing data to serve for estimating the reliability of an example system. However, without real testing data from the example system, the reliability study will be of limited value.

5. Selecting D-NHPP models and the methods of estimating parameters also is difficult. Chapter 4 of Lyu's book [1996] discusses techniques for selecting a method/model that best predicts a *specific* set of test data, and then making corrections to obtain better predictions. Essentially, the older data of the data set are employed to predict newer data, and measures are defined to determine which SRGM makes better predictions. In addition, the biases of the predictions could be identified and used to adjust the predictions so to improve their accuracy. The applicability of these techniques to D-NHPP models must be demonstrated.

As indicated in Section A.2.4, continuous-time SRGMs can be directly used as discrete models by replacing the time with the number of test runs. By doing this, the discretized version of continuous-time SRGMs with better prediction accuracy, selected using the methods described in [Lyu 1995], will offer better predictive capability in the discrete-time domain.

A.3 Parameter Estimation, Parameter Uncertainty, and Modeling Uncertainty of D-NHPP Models

Similar to the parameter estimation for continuous SRGMs discussed in [Chu 2010], the maximum likelihood method, least-squares method, and moment-matching methods can be used for discrete SRGMs. For the maximum likelihood method, the likelihood function usually is based on Equation (A-1) and is solved using iterative algorithms, such as Newton's method. For the least-squares error method, depending on the parameters, different objective functions may be used. Newton's method is sensitive to the selection of the initial values, and so the estimates might not converge to the desired solution satisfying the constraints on the parameters. An expectation-maximization (EM) algorithm for maximum likelihood function based parameter estimation of D-NHPP models was proposed in [Okamura 2004] to overcome this problem of local convergence. The EM algorithm consists of E-steps and M-steps. In the E-step, the data is used to calculate the expected log-likelihood function (LLF), and in the M-step, the parameters are assessed to maximize the expected LLF and then used to determine the expected LLF in the next E-step. The E-step and the M-step are repeated until the parameters converge.

In general, the SRGMs only consider point estimates of the model parameters. However, there is no inherent difficulty in assessing the uncertainties in them. [Musa 1987] discussed treating parameter uncertainties. For estimating maximum likelihood, the asymptotic normality property can be used to measure the error of the estimated parameters thereby giving some level of confidence on the estimation. It can be proven that maximum likelihood estimators of the parameters are normally asymptotically distributed with the expected parameter mean values and covariance matrix, or the Fisher information. The confidence level for the parameters can be approximated based on this information, as shown in [Musa 1987]. For a least-squares

estimation, the logarithm of the ratio of a data point to its model value is defined. The logarithms of all such ratios are assumed as independent, identically normally distributed with zero mean and a common variance. By calculating the variance of each parameter of the model (for intensity functions), the confidence interval for each parameter, assumed to be distributed normally, can be calculated.

Alternatively, a Bayesian approach can be used, that is, a Bayesian update performed on the pre-selected priors of the parameters, based on new observational data. By using the Bayesian approach, the uncertainty is estimated directly, as is the point estimate of the parameters.

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 with them, as suggested in [Chu 2010]. There are hundreds of continuous-time SRGMs. It is feasible to choose some of the well-known ones, and discretize them to obtain the discrete models. By feeding the same set of debugging data to different discrete models, a set of point estimates is generated of the predicted failure probabilities, yielding a distribution of the software's demand-failure probability.

A.4 An Illustration of the Method

A simple example of a D-NHPP model from Okamura [2004] is described here. Consider that a software program under test initially has a total of N software faults. Let b be the probability that each fault is triggered during a specific test, that is, T_a (the mean number of test runs it takes for a fault to manifest) obeys a geometric distribution. The probability that the fault manifests on or before the i^{th} test is $P(i) = 1 - (1 - b)^i$, and the probability that n failures occur by the i^{th} test is

$B(n; N, P(i)) = \binom{N}{n} P^n(i) [1 - P(i)]^{N-n}$. It can be shown that if N is Poisson-distributed with a

mean of ω , then the number of failures by the i^{th} test follows a DNHPP with mean value function of $\omega P(i)$, and the probability of failure on demand in the i^{th} test is $\omega b(1 - b)^{i-1}$, which actually is the discrete Goel and Okumoto model [Okamura 2004]. Similar to an NHPP model, this model has two parameters, that is, b and ω , to be estimated using test results. Alternative assumptions about $P(i)$ would lead to other failure on demand models.

For this simple example of the discretized Goel and Okumoto model, the EM-algorithm can solve the parameters b and ω by using the datasets from, e.g., Lyu's book on software reliability [1996]. It should be possible to build a simulator with the model's postulated parameters to generate artificial test data that can be employed in the model to estimate the parameters. That is, make assumptions about the parameters and their uncertainty, and generate samples from the D-NHPP. In this way, the validity of the method and its numerical solution can be demonstrated.

References:

- [Cai 2000] Cai, K.-Yuan, "Towards a Conceptual Framework of Software Run Reliability Modeling," *Information Sciences*, Vol. 126, pp. 137-163, 2000.
- [Chu 2010] Chu, T.L., et al., "Review of Quantitative Software Reliability Methods," Brookhaven National Laboratory, BNL-94047-2010, September 2010.
- [Dohi 2003] Dohi, T., and Yasui, K., "Software Reliability Assessment Models Based on Cumulative Bernoulli Trial Processes," *Mathematical and Computer Modeling*, Vol. 38, pp. 1177-1184, 2003.
- [Duane 1964] Duane, J.T., "Learning curve approach to reliability monitoring," *IEEE Transactions on Aerospace*, Vol. 2, No. 2, pp. 563-566, April 1964.
- [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.
- [Hoare 1983] Hoare, M.R., and Rahman, M., "Cumulative Bernoulli Trials and Krawtchouk Processes," *Stochastic Processes and Their Applications*, Vol. 16, pp. 113-139, 1983.
- [Huang 2003] Huang, C.Y., Lyu, M.R., and Kuo, S.Y., "A United Scheme of Some Nonhomogeneous Poisson Process Models for Software Reliability Estimation," *IEEE Transactions on Software Engineering*, Vol. 29, No. 2, March 2003.
- [Jelinski 1972] Jelinski, A., and Moranda, P., "Software Reliability Research," in W. Freiberger, ed., *Statistical Computer Performance Evaluation*, Academic Press, New York, NY, pp. 465-484, 1972.
- [Littlewood 2011] Littlewood, B., E-mail to Chu, T.L., dated January 25, 2011.
- [Lyu 1996] Lyu, M.R., Editor-in-Chief, *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.
- [Miller 1986] Miller, D.R., "Exponential Order Statistic Models of Software Reliability Growth," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 1, January 1986.
- [Okamura 2004] Okamura, H., Murayama, A., and Dohi, T., "EM Algorithm for Discrete Software Reliability Models: A United Parameter Estimation Method," *Proceedings of the Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE'04)*, March 2004.
- [Sato 2001] Sato, D., and Yamada, S., "Discrete Equations and Software Reliability Growth Models," *Proceedings of 12th International Symposium on Software Reliability Engineering (ISSRE'01)*, pp. 176-184, November 2001.

- [Shanthikumar 1983] Shanthikuma, J.G., "Software Reliability Models: A Review," *Microelectronics Reliability*, Vol. 23, No. 5, pp. 903-943, 1983.
- [Shibata 2006] Shibata, K, Rinsaka, K., and Dohi, T., "Metrics-Based Software Reliability Models Using Non-homogeneous Poisson Processed," 17th International Symposium on Software Reliability Engineering (ISSRE'06), November 2006.
- [Xie 1991] Xie, M., [Software Reliability Modeling](#), World Scientific Publisher, Singapore, 1991.
- [Yamada 1985] Yamada, S., and Osaki, S., "Discrete Software Reliability Growth Models," *Applied Stochastic Models and Data Analysis*, Vol. 1, pp. 65-77, 1985.

Appendix B Tests Performed for Licensing

This appendix describes/defines the different kinds of tests undertaken to satisfy regulatory requirements. In addition, there are brief descriptions of the test configurations used in the factory acceptance test (FAT) of the Oconee digital upgrade [NRC 2010], and the verification and validation (V&V) of the digital-safety systems of an Advanced Boiling Water Reactor (ABWR) [Fukumoto 1998].

Typically, software used for protection systems of nuclear power plants includes both platform software and application software. In general, different types of tests are applicable to these two kinds of software. For example, there were separate processes for qualifying the TELEPERM platform software [NRC 2000] and approving the application-specific software for the Oconee digital upgrade [NRC 2010]. In this appendix, only the tests on application software are described.

B.1 Description of Different Types of Tests

The following briefly describes the tests carried out as a part of the development of a protection system at a nuclear power plant [IEEE 1998].

1. Unit testing – Unit testing also is called module- or component-testing. A unit is a set of one or more computer program modules [IEEE 1008 1987], and represents the smallest piece(s) being tested. Testing verifies the correct implementation of the design and compliance with program requirements for one software element (e.g., a unit or module) or a collection of them. Software development tools often aid in the tests.
2. Integration testing – Integration testing is an orderly progression of testing incremental pieces of the software program wherein software elements, hardware elements, or both together are tested and modified if needed until the entire integrated system demonstrably complies with the program design, and the capabilities and requirements of the system. The test often is both done on the interfaces between the components and on the integrated system.
3. System testing – This testing is on the end-to-end of the entire integrated (protection) system, including hardware and software. The purpose of testing the entire system is to verify and validate whether it meets its original objectives. Testing often is based on the system's functional/requirement specifications.
4. Factory acceptance test – The vendor performs this test before handing over the system to the customer. The purpose is to ensure that the system is working correctly.
5. Site acceptance test – This testing is conducted in the system's operational environment to determine whether the system satisfies its acceptance criteria (i.e., the initial requirements and current needs of its user), and to enable the customer to decide whether to accept the system. The test takes place after the system is installed at the plant site; its purpose is to verify that the installation was done correctly, that is, all connections are connected properly.

Since the aim of statistical testing is to verify the system during its operation, but it might not be feasible to do so after it is installed at the plant, such testing should be undertaken after or as a part of the factory-acceptance test.

B.2 Factory Acceptance Tests Performed to Support Oconee Digital Upgrade

The Oconee digital upgrade [NRC 2010] replaces the reactor protection system (RPS) and the engineered safeguard protection system (ESPS) with TELEPERM-based digital systems. In addition, the plant also will install two diverse actuation systems, that is, the diverse low pressure actuation system and the diverse high pressure injection system. The NRC reviewed the TELEPERM system in two stages. First, the TELEPERM platform was certified in 2000 [NRC 2000], and the specific application to Oconee was approved in 2010 [NRC 2010]. The following summary description of the software testing is taken from information in publicly available documents.

The TELEPERM platform software includes both the operating system and the platform software [NRC 2000]. The latter encompasses the Run Time Environment (RTE) and its modules, the input/output (I/O) drivers for the input/output module interface, the exception handler, and the self-test software. To control and facilitate the development of application software, the TELEPERM system includes a specification and coding environment (SPACE) tool for designing and assembling safety-related applications. Software specifications are prepared as functional diagrams using a graphical user interface, the TELEPERM XS editor. Using the SPACE tool, the application software for the safety I&C system is completely specified in graphical form. It consists of diagrams of interconnected hardware blocks representing the hardware architecture of the safety system, and diagrams of interconnected function blocks (FBs) representing the software-implemented safety functions. From these specifications, the SPACE tool automatically generates the configuration data for the system software and application software as a composition of interconnected pre-existing- and type-tested- software components. In addition, The SPACE system also produces the documents required for manufacturing the hardware.

The Oconee digital RPS/ESPS application software is implemented using function blocks that are entered into the SPACE tool [NRC 2010]. Application developers follow procedures that direct the usage of approved and validated library components to build Function Diagrams (FDs). After their completion and verification by an independent V&V engineer, the approved FDs are converted into a C-code-based object file using the SPACE tool. This file then is compiled and converted into a platform-compatible executable file to be downloaded into the digital RPS/ESPS. Once installed into the TELEPERM XS (TXS) hardware, the integrated hardware/software system is exercised via a test platform named ERBUS for FAT or troubleshooting. The ERBUS system generates analog and digital signals, which are wired directly into the TXS hardware during factory testing. In addition, the system's output analog- and digital-signals are wired to input channels of the ERBUS for monitoring the system's outputs during tests. Figure B-1, which is based on the documentation in [AREVA 2009], shows the test configuration.

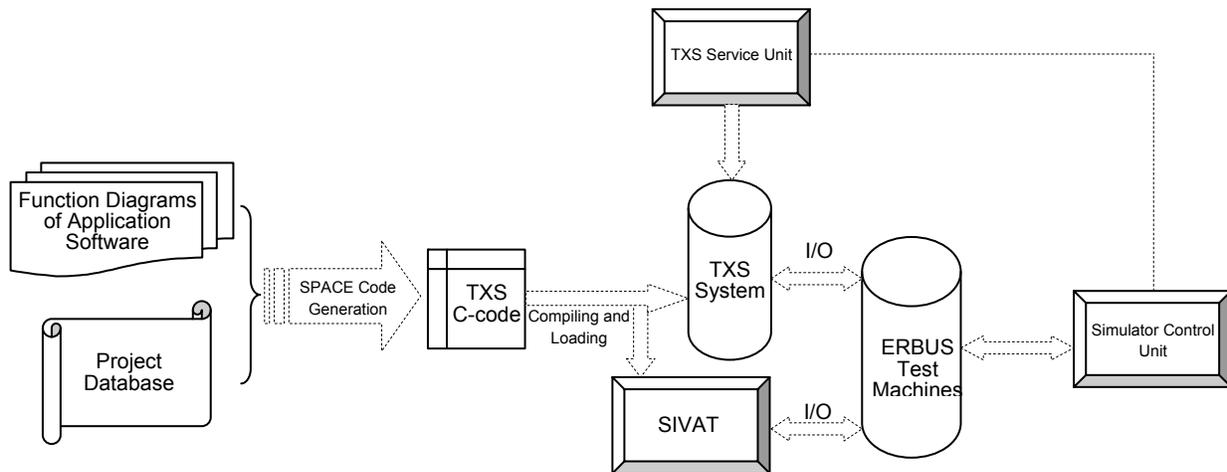


Figure B-1 ERBUS TELEPERM XS Concept

After manufacturing the TXS system, it is set up in the test field, to commission and test the overall system. To do this, the TXS Application Software is loaded onto the CPU modules in the TXS system, and the TXS inputs and outputs are linked with those of the ERBUS TXS test machines. SIVAT is used in validating the functional requirements of the software [AREVA 2009], generates an I/O interface file saved in the simulator database (SimDB), and serves as an oracle of the tests using the test machines. All test machines are connected to a central computer, the Simulator Control Unit (SCU). The SCU is the main component of the test-field simulator. The test-field simulator uses the list of I/O signals that contains an assignment of the TXS signals to the ERBUS TXS channels. There is one communications model for each connected test machine and Service Unit in the simulator. These models send or receive their respective assigned signals.

With the simulator operating, the TXS inputs are cyclically stimulated with the values in the SimDB via the ERBUS outputs, and the values at the TXS outputs are cyclically entered into the SimDB.

Furthermore, the TXS Service Unit (SU) and the SCU can be linked. Then, triggering of the TXS inputs and outputs also can be implemented at the SU. The main task of the test field simulator is to stimulate and measure all inputs and outputs of a TXS system. Depending on the system's size, this can involve several hundred or even several thousand signals.

B.3 Test Configuration of a Japanese ABWR

Fukumoto [1998] describes the usage of an automated tool in testing the reactor protection system and the engineered safety feature system of an ABWR, that is, Kashiwazaki-Kariwa Unit 6 of the Tokyo Electric Power Company. Figure B-2 shows the test configuration. It includes a personal-computer-based supervisory test control (STC) unit and four signal simulators (SSs), each with a process input output (PIO) module connected to a channel of the digital safety

system under test. Test signals are simulated sensor signals. A test procedure defines the names of the input signals of the digital safety system, their values, their time tags, and the names of the output signals from the digital safety system being monitored. The STC controls the SSs via the Ethernet, based on data defined in the test procedure. Each SS generates and feeds test signals to its respective division of the digital safety system, monitors the corresponding output signals of its division, and sends their values to the STC. A series of tests can be run automatically. Test personnel make the final judgment of the results by checking them on a cathode ray tube display. This tool was used in system-logic tests and dynamic-transient tests.

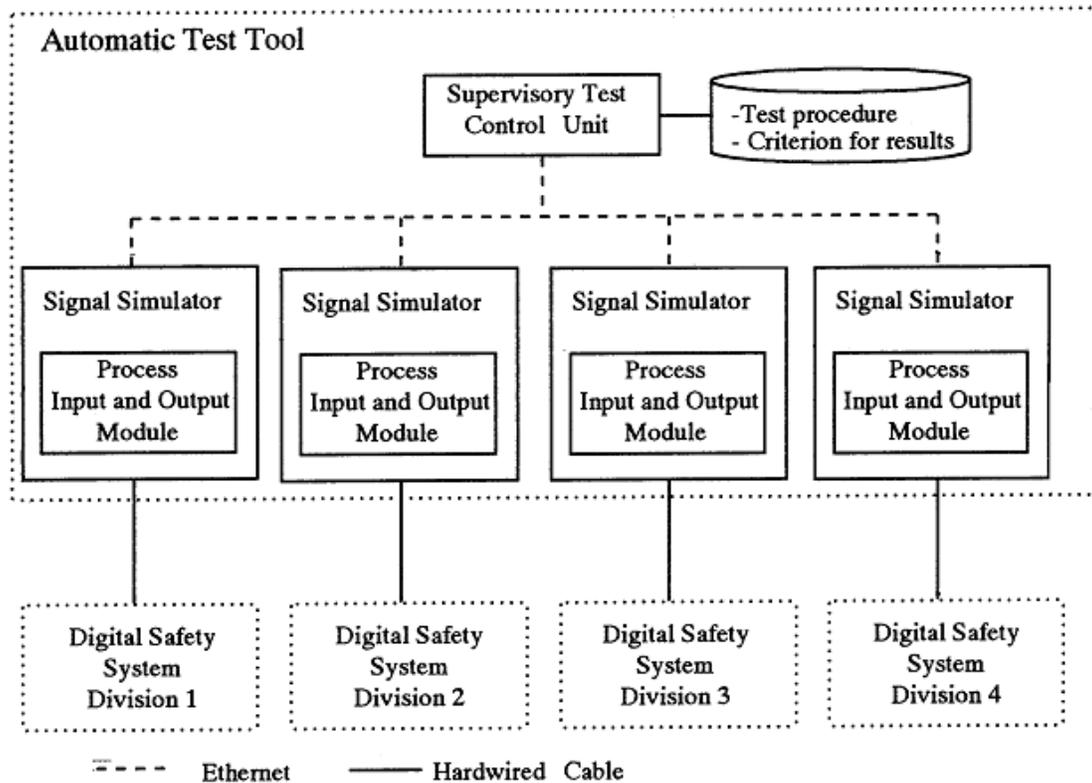


Figure B-2 Test Configuration of an ABWR

References:

- [AREVA 2009] AREVA, "SIVAT: TELEPERM XS™ Simulation Validation Test Tool," ANP-Topical Report, ANP-10303NP, Rev. 0, June 2009.
- [Fukumoto 1998] Fukumoto, A., et al., "A Verification and Validation Method and Its Application to Digital Safety Systems in ABWR Nuclear Power Plants," *Nuclear Engineering and Design*, Issue 183, pp. 117-132, 1998.
- [IEEE 1987] IEEE, "IEEE Standard for Software Unit Testing," ANSI/IEEE Std 1008-1987.
- [IEEE 1998] IEEE, "IEEE Standard for Software Verification and Validation," IEEE Standard 1012-1998.
- [NRC 2000] U.S., Nuclear Regulatory Commission, "Safety Evaluation by the Office of Nuclear Reactor Regulation Siemens Power Corporation Topical Report EMF-2110(NP), 'TELEPERM XS: A Digital Reactor Protection System,' Project No. 702," May 4, 2000.
- [NRC 2010] U.S. Nuclear Regulatory Commission, "Duke Energy Carolinas, LLC Docket No. 50-269, Oconee Nuclear Station, Unit 1, Amendment To Renewed Facility Operating License, Amendment No. 366, Renewed License No. DPR-38," January 28, 2010.