U.S.NRC

United States Nuclear Regulatory Commission

*Protecting People and the Environment*

NUREG/CR-7044
BNL-NUREG-99068-2013

# Development of Quantitative Software Reliability Models for Digital Protection Systems of Nuclear Power Plants

Office of Nuclear Regulatory Research

U.S.NRC
United States Nuclear Regulatory Commission
*Protecting People and the Environment*

NUREG/CR-7044
BNL-NUREG-99068-2013

# Development of Quantitative Software Reliability Models for Digital Protection Systems of Nuclear Power Plants

Manuscript Completed: January 2013
Date Published: October 2013

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

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

NRC Project Manager:
Ming Li
Alan Kuritzky

Office of Nuclear Regulatory Research

# ABSTRACT

The U.S. Nuclear Regulatory Commission is currently performing research on the development of probabilistic models for digital instrumentation and control systems for inclusion in nuclear power plant (NPP) probabilistic risk assessments. As part of this research, Brookhaven National Laboratory (BNL) is exploring the inclusion of software failures into digital system reliability models. A previous BNL technical report, entitled "Review of Quantitative Software Reliability Methods," BNL-94047-2010 (ADAMS Accession No. ML102240566), documented a review of currently available quantitative software reliability methods (QSRMs) that can be used to quantify software failure rates and probabilities of digital systems at NPPs and identified a set of desirable characteristics for QSRMs. In the current report, two candidate QSRMs are selected based on a structured comparison of the previously-identified QSRMs against the set of desirable characteristics. Each selected method is further developed in preparation to be applied in a case study. This report also identifies an example digital protection system for use in the case studies. The actual case studies will be documented in separate reports. Completion of the case studies is expected to provide a much better understanding of the existing capabilities and limitations in treating software failures in digital system reliability models.

# FOREWORD

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

The current U.S. Nuclear Regulatory Commission (NRC) licensing process for digital systems relies on deterministic engineering criteria. In its 1995 PRA Policy Statement, the Commission encouraged the use of PRA technology in all regulatory matters, to the extent supported by the state-of-the-art in PRA methods and data. Though many activities are carried out in the life cycle of digital systems to ensure a high-quality product, there are presently no consensus state-of-the-art methods for quantifying the reliability of digital systems.

To address this limitation, the NRC is currently performing research on the development of probabilistic models for digital I&C systems for inclusion in NPP PRAs. This research is consistent with the recommendations from the 1997 National Research Council report on digital I&C in nuclear power plants and with the Commission staff requirements memorandum (M061108), dated December 6, 2006, which directs the staff to address the deployment of digital systems, including the area of risk-informed digital I&C.

Brookhaven National Laboratory (BNL) is supporting the NRC in this research through a series of projects on digital I&C system reliability modeling and quantification. Previous BNL projects have focused on reliability modeling and quantification of digital system hardware, and on a review of currently available quantitative software reliability methods (QSRMs) that can be used to quantify software failure rates and probabilities of digital systems at NPPs. This previous work identified a set of desirable characteristics for QSRMs. In this current report, two candidate QSRMs are selected based on a structured comparison of the previously-identified QSRMs against the set of desirable characteristics. Each selected method is further developed in preparation to be applied in a case study. The scope of the method development is limited to modeling failures of NPP protection systems to perform their functions when called upon (represented by the probability of failure on demand), due to the risk importance of these functions. This report also identifies an example digital protection system for use in the case studies. The actual case studies will be documented in separate reports. Completion of the case studies is expected to provide a much better understanding of the existing capabilities and limitations in treating software failures in digital system reliability models for use in NPP PRAs.

The NRC is conducting separate research on the analytical assessment of digital I&C systems. The current licensing deterministic approach is heavily dependent on expert judgment. Assuring that safety-related digital I&C systems will perform satisfactorily in service has become more difficult due to uncertainties arising from systemic causes, such as the use of increasingly complex software. In order to provide reasonable assurance for deterministic licensing decisions, multiple approaches must be used to address these uncertainties (including testing and process audits). To further enhance the NRC's licensing reviews, ongoing analytical assessment research is focused on developing improved methods for obtaining reasonable assurance of the performance of safety-critical software and systems.

While these two areas of research (i.e., digital I&C reliability modeling for PRA and the analytical assessment of digital I&C systems in support of the current deterministic regulatory decision making approaches) are complementary in many ways, it should be emphasized that they are intended to support very different applications.  As such, some methods may be appropriate for one of these two areas, but not the other.  In the longer term, if the staff determines that the development and quantification of reliability models of digital I&C systems (including software) is practical and useful, it will consider the development of regulatory guidance for the use of risk information in regulatory decisions regarding digital I&C systems for new and operating reactors to be consistent with the Commission's PRA policy statement.

# TABLE OF CONTENTS

## LIST OF FIGURES

# LIST OF TABLES

# EXECUTIVE SUMMARY

The U.S. Nuclear Regulatory Commission (NRC) encourages the use of probabilistic risk assessment (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 has not been fully developed. The NRC established a plan for digital system research to identify and develop methods, analytical tools, and regulatory guidance for (1) including models of digital systems in nuclear power plant (NPP) PRAs and (2) incorporating digital systems in the NRC's risk-informed licensing and oversight activities.

Brookhaven National Laboratory (BNL) is supporting the NRC in this research through a series of projects on digital instrumentation and control (I&C) system reliability modeling and quantification. In a previous project, BNL surveyed the quantitative software reliability methods (QSRMs) and developed a set of criteria to select the appropriate QSRMs that can be used to quantify software failure rates for NPP digital I&C systems. The current study continues on this research path, with the following objectives:

1. Evaluate the QSRMs identified by 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 to apply the candidate methods to the selected protection system.

The following ten selection criteria were applied to the surveyed QSRMs.

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 software life cycle activities.
5. The method uses 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 is capable of estimating parameters that can be used to account for software common cause failures (CCFs) of diverse protection systems or channels.
10. The data and necessary information exist and can be collected.

Based on this review, the Bayesian Belief Network (BBN) and Statistical Testing Method (STM) were selected as appropriate candidate methods to be applied to an example system.

Ideally, an example system for the case study should be a safety-critical digital I&C system. However, the selection of the example system was based on the availability of the system and the documentation that supports the case study. For this research effort, BNL reached an agreement with the Idaho National Laboratory to apply the BBN and STM to the protection function of a control system of their Advanced Test Reactor.

BNL's BBN method assumes that the number of defects in the software is determined by the quality of the software development activities: in other words, the characteristics of the software development process and the software product determine the software's quality, and, thus, the number of defects in the software. The software failure probability on demand will be derived from the number of defects and the operational environment that can trigger the defects, resulting in the software's failure to perform its intended function. NUREG-0800, Chapter 7, Branch Technical Position 7-14, "Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Systems" (BTP 7-14), provides guidelines for evaluating software life cycle processes, including quality assurance, for digital computer-based I&C systems. Furthermore, industry standards also describe quality attributes for software development activities. The BBN models the causal relationship between quality attributes and the number of defects, and then the failure probability. Subject matter experts are used to build the BBN causal relationships and provide inputs to the BBN model.

As part of this research effort, BNL proposed the STM to quantify the probability of failure on demand. The test cases are produced from a thermal-hydraulic simulation, and are intended to represent the actual reactor operational conditions.

BNL developed a framework to include software failure in the NPP PRAs. In this framework, a segment of cutset was used to represent the accidental conditions (termed the PRA context) under which the software is exercised. The software failure probability obtained through the STM is, therefore, a conditional probability, given the PRA context. This risk-informed approach may reduce the testing required to demonstrate that the reliability of a digital system is sufficient to support a desired plant-level risk target. Three examples were then provided to demonstrate how to estimate the adequate number of test cases to achieve a certain level of software reliability.

In summary, the current study selected the BBN and STM as two methods to be applied to an example system to demonstrate their feasibility and practicality for potential use in NPP PRAs. The actual case studies are being conducted as a follow-up research project, and the results will be presented in future reports.

The following findings/limitations stand out as noteworthy for further investigations:

1. This study assumes that the software failure of a protection system is modeled at the system level. In general, it may be necessary to consider software failures at a more detailed level to fit the NPP's PRA needs.
2. The proposed STM allows PRA contexts to be explicitly identified, but requires that the operational profile for each PRA context be developed. The realism of the operational profiles is a potential limitation of the proposed approach. In addition, the thermal-hydraulic models used to generate the test cases are a rough approximation of the physical processes of an NPP, and their ability to model all events that may affect the inputs to the software (e.g., pump failure) may be limited.
3. This study conservatively assumes that the software in redundant channels fails simultaneously due to common software faults. This assumption might be overly conservative if diverse software or other means of mitigating CCFs are deployed in redundant channels.
4. The BBN method will depend heavily on expert opinion. The uncertainty introduced by the expert opinion process needs to be addressed.

## ACKNOWLEDGEMENTS

The work presented in this report includes work carried out during several years under a U.S. Nuclear Regulatory Commission (NRC) project. Many people helped in one way or another with this work, and we are grateful to them.

In particular, we would like to thank Professors Bev Littlewood and Lorenzo Strigini of City University London for their comments and suggestions on draft versions of the report. We would also like to thank Dr. Hyun Gook Kang of the Korea Atomic Energy Research Institute (KAERI) for his helpful discussions during his visit to Brookhaven National Laboratory (BNL) in 2010.

We are especially grateful to the external peer reviewers from the nuclear industry and international organizations who commented on some early parts of the work, the public reviewers, and those reviewers from the NRC who reviewed the final draft of this report. The peer reviewers include Dr. Ali Azarm of Information Systems Laboratories (ISL), Jan Erik Holmberg of the Technical Research Centre of Finland (VTT), Allen Nikora and Ying Shi of the National Aeronautics and Space Administration (NASA), Dave Blanchard of Applied Reliability Engineering (ARE), Rick Wachowiak of the Electric Power Research Institute (EPRI), and Nguyen Thuy of Électricité de France (EDF). The public reviewers include James A. Ross of GE Hitachi, Daniel Schmid of the Swiss Federal Nuclear Safety Inspectorate (ENSI), and Gassino Jean and Regnier Pascal of the French Institut de Radioprotection et de Sûreté Nucléaire (IRSN). The NRC staff include Steven A. Laur and Steven Arndt of the Office of Nuclear Reactor Regulation (NRR), Malcolm Patterson and Royce D. Beacom of the Office of New Reactors (NRO), and Sushil Birla and Mike Waterman of the Office of Nuclear Regulatory Research (RES).

We are indebted to the NRC Project Manager, Alan Kuritzky, for his technical and managerial support; to the NRC technical lead, Ming Li, for his technical support; and to Kevin Coyne for his technical and managerial support.

We also express our appreciation to Jean Frejka, who put several versions of the report together and helped with the logistical aspects of the project.

# ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| ABWR | Advanced Boiling Water Reactor |
| ASME | American Society of Mechanical Engineers |
| ATR | Advanced Test Reactor |
| ATWS | Anticipated Transient Without Scram |
| BBN | Bayesian Belief Network |
| BEA | Battelle Energy Alliance |
| BNL | Brookhaven National Laboratory |
| BTP | Branch Technical Position |
| CCF | Common-Cause Failure |
| CDF | Core Damage Frequency |
| COMPSIS | Computer Systems Important to Safety |
| CPU | Central Processing Unit |
| CSR | Center for Software Reliability |
| 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 Method[1] |
| EFC | Error-Forcing Context |
| ESFAS | Engineered Safety Features Actuation System |
| ESPS | Engineered Safeguard Protection System |
| FAT | Factory Acceptance Test |
| FLIM | Failure Likelihood Index Method |
| HRA | Human Reliability Analysis |
| I&C | Instrumentation and Control |
| I/O | Input/Output |
| IC | Integrated Circuits |
| IE | Initiating Event |
| IEC | International Electrotechnical Commission |
| INL | Idaho National Laboratory |
| IPTs | In-Pile Tubes |
| JTAG | Joint Test Action Group |
| KAERI | Korea Atomic Energy Research Institute |
| KNICS | Korea Nuclear Instrumentation and Control System |

[1] In the literature, in SRGMs, M stands for "model." However, we think "method" is more appropriate. "Method" and "model" are used interchangeably in this report.

| | |
|---|---|
| LL | Large LOCA |
| LOCA | Loss of Coolant Accident |
| LOCS | Loop Operating Control System |
| LPI | Low-Pressure Injection |
| | |
| MTTF | Mean Time To Failure |
| MUCH | Maximum Useful Capacity Holder |
| MUNIN | MUscle and Nerve Inference Network |
| | |
| NASA | National Aeronautics and Space Administration |
| NEA | Nuclear Energy Agency |
| NPP | Nuclear Power Plant |
| NPT | Node Probability Tables |
| NRC | U.S. Nuclear Regulatory Commission |
| | |
| OECD | Organisation for Economic Co-operation and Development |
| PDF | Probability Density Function |
| PRA | Probabilistic Risk Assessment |
| PWR | Pressurized Water Reactor |
| | |
| QSRM | Quantitative Software Reliability Method |
| | |
| RES | Office of Nuclear Regulatory Research |
| RG | Regulatory Guide |
| RPS | Reactor Protection System |
| | |
| SIL | Safety Integrity Level |
| SL | Small LOCA |
| SO | System Operability |
| SRGM | Software Reliability Growth Model[1] |
| STM | Statistical Testing Method |
| STUK | Radiation and Nuclear Safety Authority, Finland |
| SIVAT | Simulation Validation Test |
| | |
| V&V | Verification and Validation |
| VTT | Technical Research Centre of Finland |
| | |
| WGRisk | Working Group on Risk Assessment |

# 1 INTRODUCTION

## 1.1 Background

The U.S. Nuclear Regulatory Commission's (NRC's) current licensing process for digital systems relies on deterministic engineering criteria. In its 1995 probabilistic risk assessment (PRA) policy statement [NRC 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 [NRC 2010a] defining a coherent set of projects to support regulatory needs. Some of the projects included in this research plan address risk assessment methods and data for digital systems. The objective of the NRC's digital system risk research is to identify and develop methods, analytical tools, and regulatory guidance for (1) including models of digital systems in NPP PRAs and (2) incorporating digital systems in the NRC's risk-informed licensing and oversight activities.

Figure 1-1 graphically depicts the relationships between the various activities associated with the NRC's digital system risk research. The work on developing a digital system reliability modeling approach is being coordinated with several other related research efforts being carried out by the NRC. As indicated in Figure 1-1, these other areas include failure mode identification and analysis [Chu 2008 and 2009a], operating experience analysis [Korsah 2010], and digital system inventory and classification [Wood 2012]. In addition, this research has benefited from interactions with the Electric Power Research Institute and the National Aeronautics and Space Administration (NASA) under separate memoranda of understanding, and with the Organisation for Economic Co-operation and Development (OECD) Nuclear Energy Agency (NEA), more specifically the Working Group on Risk Assessment (WGRisk) and the OECD/NEA activity on Computer Systems Important to Safety (COMPSIS).

An important insight from the initial digital system reliability research is the need to establish a commonly accepted basis for incorporating software behavior into digital I&C system reliability models that is compatible with existing NPP PRAs[2]. 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].

[2] Existing NPP PRAs are assumed to be developed using traditional (static) event tree and fault tree methods. To address software failures in the current PRA framework, software failures need to be captured in PRA sequences. In other words, software functions or components need to be modeled as event tree top events or fault tree basic events and quantified using one or more quantitative software reliability methods, which are the primary interest of this study.

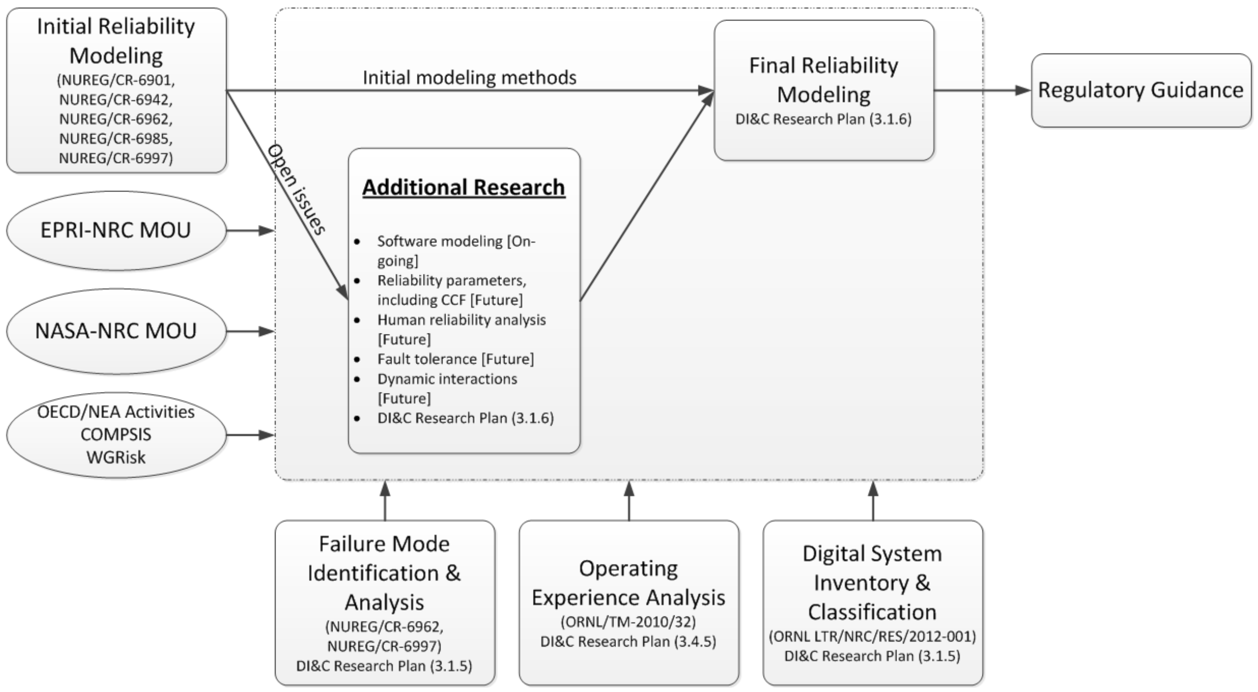

Initial Reliability Modeling
(NUREG/CR-6901, NUREG/CR-6942, NUREG/CR-6962, NUREG/CR-6985, NUREG/CR-6997)
Initial modeling methods
Open issues
Final Reliability Modeling
DI&C Research Plan (3.1.6)
Regulatory Guidance
EPRI-NRC MOU
NASA-NRC MOU
OECD/NEA Activities
COMPSIS
WGRisk
Additional Research
Software modeling [On-going]
Reliability parameters, including CCF [Future]
Human reliability analysis [Future]
Fault tolerance [Future]
Dynamic interactions [Future]
DI&C Research Plan (3.1.6)
Failure Mode Identification & Analysis
(NUREG/CR-6962, NUREG/CR-6997)
DI&C Research Plan (3.1.5)
Operating Experience Analysis
(ORNL/TM-2010/32)
DI&C Research Plan (3.4.5)
Digital System Inventory & Classification
(ORNL LTR/NRC/RES/2012-001)
DI&C Research Plan (3.1.5)

**Figure 1-1 NRC research activities on digital system reliability**

Software failure has been defined in the literature differently [IEEE 610, Lyu 1996], and there is no consensus on the definition. In this study, software failure is defined as the triggering of a fault of the software, introduced during its development life cycle, which results in, or contributes to, the host (digital) system failing to accomplish its intended function or initiating an unwanted action. The triggering includes the generation of particular inputs to the software due to the state of the operating environment (i.e., of the NPP), in combination with the internal state of the digital system.

BNL has been exploring how software failures 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. The following philosophical basis for incorporating software failures into a PRA was established at the meeting [Chu 2009b]:

> "Software failure is basically a deterministic process. However, because of our incomplete knowledge, we are not able to fully account for and quantify all the variables that define the software failure process. Therefore, we use probabilistic modeling to describe and characterize it."

They also agree that:

1. Software fails.
2. The occurrence of software failures can be treated probabilistically.
3. It is meaningful to use software failure rates and probabilities.

4. Software failure rates and probabilities can be included in reliability models of digital systems.

Subsequently, BNL reviewed a spectrum of quantitative software reliability methods (QSRMs) to catalog potential methods that may serve to quantify software failure rates and per-demand 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 NASA, performed by international organizations, and published in journals and presented at conferences. The strengths and limitations of QSRMs for PRA applications were categorized, described, and evaluated. In addition, a set of desirable QSRM characteristics was established. While these characteristics are intended for use in evaluating and selecting QSRMs for future applications, a structured comparison between the methods and the individual characteristics was not undertaken at that time.

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

This study was conducted with the following objectives:

1. Evaluate the QSRMs reviewed by 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.

The study involves general development of the modeling methods. Potential applications to plant-specific decision making are out-of–scope of this work. After the actual case studies are completed and a fuller digital system model is developed, insights can be obtained into the feasibility, practicality, and usefulness of developing digital system models to include in PRAs.

Digital protection systems modeled in a PRA may have multiple failure modes. For example, a reactor protection system (RPS) may fail to generate a reactor trip signal when a trip condition occurs, or may generate a spurious trip signal. In this study, the scope of the method development is limited to modeling software failures of protection systems to perform their functions (represented by the probability of failure on demand[3]) at an NPP, mainly due to the importance of these safety-related functions.

Spurious trips would be included in a PRA as an initiating event (i.e., the starting point of an accident sequence analysis), and its failure rate (in terms of annual frequency of occurrence) would typically be estimated using operating experience. If sufficient operating experience was not available, then a failure-rate-based modeling approach would be necessary. On the other hand, after the occurrence of some other initiating event necessitating a reactor trip, a failure to generate a reactor trip signal would be modeled in the PRA as a demand failure probability,

[3] By “demand,” we mean a plant condition that requires actuation of safety systems (e.g., the reactor trip system).

necessitating the use of a failure-on-demand-based method. Therefore, even for a single system, different QSRMs (e.g., one failure-rate-based and one failure-on-demand-based) may be needed, depending on the failure modes of interest.

There is presently no consensus method for modeling digital systems in NPP PRAs [NRC 2008, NEA 2009]. Different methods have been proposed, including the fault tree method. However, whether or not fault tree models adequately capture the dependencies of digital systems has not been adequately demonstrated. Digital system reliability models may include software failures representing different software failure modes[4] at different levels of detail (e.g., the software may be modeled at a system, subsystem, or module level). 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 modes. That is, the software system is a collection of software, including applications, the operating system, and platform software implemented in a digital system consisting of multiple microprocessors. 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. These considerations notwithstanding, for practicality, this study considered only a system-level failure mode for the protection system to fail to perform its necessary function, consistent with most previous QSRM applications.

The inclusion of software failures in a PRA requires that:

1. The failure modes of the software component or function need to be defined for the specific scenarios defined in a PRA[5].
2. The software failure modes for the specific PRA scenarios have to be quantified using a QSRM.
3. The software failure modes are included in a reliability model of the overall system (i.e., with hardware failure modes included), which is integrated into a PRA.

"Context" is a confusing term in the literature, and has been used interchangeably with "software input space" and "operational profile." For this study, these terms are distinguished as defined below:

- Software input space: The set of all possible input values or ranges formulated in the software requirements specifications for a specific software function.

- Operational profile: The set of distributions of occurrence likelihood (or frequency) for the inputs over values[6]. For instance, a binary input has two possible values, "0" and "1"; its operational profile can be defined (just a numerical example) as Pr("0") = 0.3 and Pr("1") = 0.7, which means that the value of "1" will appear 70% of the time and the value of "0" will appear 30% of the time.

[4] Software failure modes in this report are defined as the ways in which software fails from the output perspective. This definition differs from that found in software failure modes and effects analysis, which is the root cause of a software failure.
[5] Software failures are known to be sensitive to the context in which the software operates [Garrett 1999].
[6] In a real NPP environment, different inputs might not be independent. Such correlations and dependencies might be captured as joint distributions. The following example implies an independent individual input, and is for demonstration purposes only.

- Context: Also termed the “PRA context” in this report. *Context* is defined as plant conditions determined by the PRA scenarios under which the software operates. For instance, if an RPS appears as a top event after the small loss of coolant accident (LOCA) as an initiating event (IE), the condition (represented as input values for the software) associated with this IE is identified as the “PRA context” for this RPS software. An operational profile can be defined for each context.

Many protection systems, including the one anticipated to be the subject of the example case study, are designed with identical redundant channels that run the same software. As such, it is assumed that these channels would fail together, due to common software faults when the same input signals are encountered. Therefore, this type of common-cause failure (CCF) can be quantified using the methods discussed in this study. The potential for CCF between diverse channels of the same system or due to dependencies between two digital protection systems performing similar functions in the same accident scenarios was considered beyond the scope of this study. Similarly, any CCFs that can affect other plant systems modeled in the PRA are beyond the scope of the study. However, due to the potential importance of software CCF to plant risk, this is an area that is recommended for further research.

## 1.3 Approach

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

In identifying the candidate methods among those reviewed by Chu [2010], the set of desirable characteristics for candidate QSRMs developed therein was expanded through the addition of a characteristic on the availability of necessary data. The different QSRMs were compared against the expanded list of characteristics, and the Bayesian Belief Network (BBN) and statistical testing were chosen as candidates[7].

For each of these two candidate methods, the literature was reviewed to assess the method’s 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 in carrying out software-life cycle activities, it was also decided to first develop a prior distribution with the BBN approach, and then update it by Bayesian inference using the statistical testing results. The BBN approach is based on the understanding that the quality in carrying out the software development activities determines the reliability of the software. Since developing and quantifying a detailed BBN can be very resource-intensive and the evidence needed for quantification may be lacking, both a simplified and a detailed BBN will be developed. This exercise will provide insight into the costs and benefits of developing a detailed BBN model. It is expected that results from BBNs will have large uncertainties due to their reliance on expert opinion, particularly in cases where evidence is very limited. The results obtained from the BBN will be updated using evidence

[7] 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.

obtained from the statistical testing. 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 reviewed and commented upon the draft working material of this project. 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 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.

Figure 1-2 summarizes the structure of the report. Section 2 documents the evaluation of the QSRM methods against the set of desirable characteristics and the selection of the two candidate methods, that is, the BBN and statistical testing methods. Section 3 describes the selection of the example system for the case study. Sections 4 and 5 detail the approach to developing the two candidate methods, respectively. Figure 1-2 shows that for each context defined in a PRA for a protection system, a Bayesian analysis is performed using either a non-informative prior distribution or a prior distribution derived from the BBN, and the test results of the statistical testing method. Note that since this study only further developed an overall approach for the case study, Section 6 gives the conclusions and the insights obtained from the study. Additional insights and lessons learned can be obtained only after the case studies are completed.

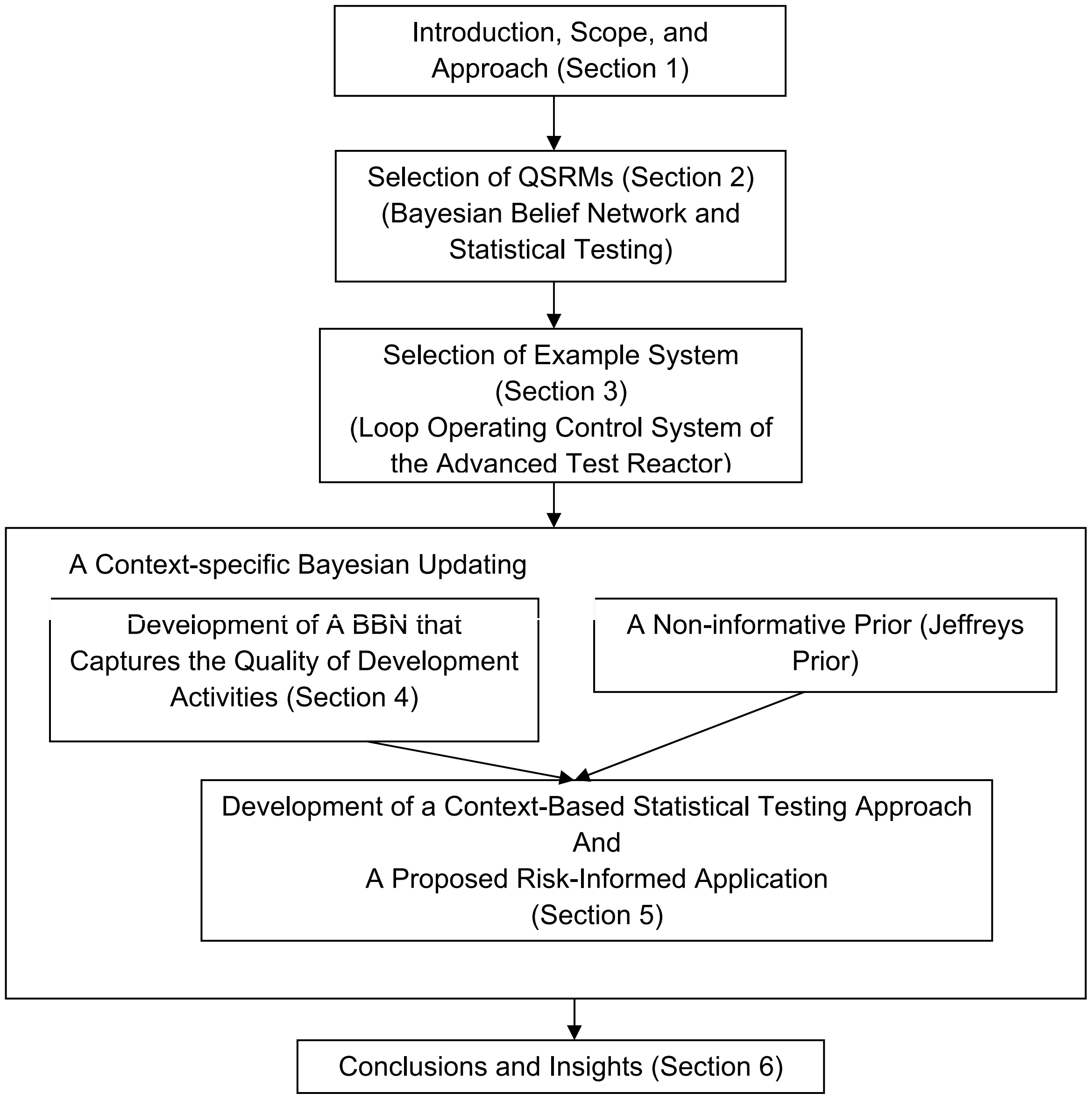

Introduction, Scope, and Approach (Section 1)
Selection of QSRMs (Section 2) (Bayesian Belief Network and Statistical Testing)
Selection of Example System (Section 3) (Loop Operating Control System of the Advanced Test Reactor)
A Context-specific Bayesian Updating
Development of A BBN that Captures the Quality of Development Activities (Section 4)
A Non-informative Prior (Jeffreys Prior)
Development of a Context-Based Statistical Testing Approach And A Proposed Risk-Informed Application (Section 5)
Conclusions and Insights (Section 6)

**Figure 1-2 Development of quantitative software reliability models for digital protection systems of nuclear power plants**

# 2 SELECTION OF QUANTITATIVE SOFTWARE RELIABILITY METHODS

In this section, the quantitative software reliability methods (QSRMs) described in detail and reviewed in an earlier study [Chu 2010] are evaluated against the desirable characteristics developed therein. The objective of this evaluation is to identify the most promising methods for quantifying software reliability to help formulate digital system reliability models for use in probabilistic risk assessments (PRAs) of nuclear power plants. The set of desirable characteristics is repeated below, with a shortened description provided in parentheses. Note that a tenth characteristic was added to account for the fact 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. *(method description)*
2. The assumptions of the method have reasonable bases. *(reasonable assumptions)*
3. The method allows for consideration of the specific operating conditions of the software. *(consideration of operating conditions)*
4. The method considers the quality in carrying out life cycle activities. *(consideration of life cycle quality)*
5. The method uses available test results and operational experience. *(use of data)*
6. The method addresses uncertainty. *(addressing uncertainty)*
7. The method has been verified and validated. *(verification and validation)*
8. The method is capable of demonstrating the high reliability of a safety-critical system. *(demonstrating high reliability)*
9. The method is capable of estimating parameters that can be used to account for software common-cause failures (CCFs) of diverse protection systems or channels. *(software CCF)*
10. The data and necessary information exist and can be collected. *(availability of data)*

Section 2.1 describes QSRM common limitations. Section 2.2 assesses each method against these characteristics by discussing its strengths and weaknesses; Table 2-1 summarizes the evaluations. Section 2.3 identifies candidate methods for a future proof-of-concept case study. Sections 4 and 5 describe the approaches for developing the BBN and statistical testing methods, respectively. Candidate methods developed BNL are further evaluated against the characteristics and the corresponding results, and discussions are presented at the end of Sections 4 and 5.

## 2.1 Common Limitations of Quantitative Software Reliability Methods

The following describes common limitations applicable to the set of reviewed QSRMs. A detailed evaluation per characteristic per method is given in the following sections.

*Test profile vs. operational profile*

Most of the QSRMs reviewed use available test and/or operational data. For test data, the QSRM descriptions did not always specify whether the test cases were sampled from the software's operational profile. There is often little description in these papers or reports of how the tests were performed, or what was done to ensure that they truly represent the operational profile. Nonetheless, standard statistical methods were used in quantifying the software's failure rates or probabilities. 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* (Characteristic 3)

Software failures are dependent on the context (e.g., the specific system function being evaluated, the associated success criteria, and other relevant conditions in the plant) in which the software is operating [Garrett 1999].

In general, the QSRMs reviewed for this study consider the failure of a software program to perform its function without specifically considering the external context in which the software is expected to function (here, *external context* refers to contextual factors external to the digital system of interest). In general, one may argue that a QSRM can be applied to a specific external context (e.g., actuation of a safety function, given the occurrence of a specific accident scenario). For example, Guarro [2007] successfully demonstrated a Context-Based Software Risk Model in a PRA study for a space application. Application of this concept is described in more detail in Sections 5.1 and 5.2.

*Demonstration of high reliability* (Characteristic 8)

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 level of reliability. Automated testing, in which test cases are randomly generated from the operational profile, could demonstrate this high level of reliability, but it may be difficult to verify and validate the operational profile. To BNL's best knowledge, no studies in the literature have demonstrated this level of probability.

*Level of modeling detail*

As discussed in Section 1, the system-level failure modes of digital systems at an NPP can be defined. For example, an expert workshop [Chu 2009b] identified three system-level failure modes: failure to generate signal in time, spurious signal, and adverse effects on other functions. 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 software failures for each microprocessor (e.g., a microprocessor program continues to run but generates incorrect results), as is done in the reliability model of a digital feedwater control system [Chu 2009a], where software failure modes are included at the microprocessor level as placeholders. Current QSRMs do not seem to address software failure modes at a more detailed level. However, it may be possible to use the statistical testing method at this level of detail.

*CCF of two diverse digital systems* (Characteristic 9)

CCF is a very important issue for digital systems. It is commonly assumed that if the redundant channels of a safety-related system (e.g., an RPS or engineered safety feature actuation system) run identical software, they would all fail given a software failure that leads to loss of a channel function. This assumption may be too conservative, since some digital protection systems may be designed with two or more diverse channels [Wood 2009]. 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.

N-version programming is a software diversity strategy that has been quantitatively assessed in experiments[8] [Lyu 2005] that demonstrated that it can enhance software reliability. It 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 alpha or beta factor model). Littlewood and Rushby [2010] developed a mathematical approach for assessing the failure probability of two diverse software systems, a complex one and a simple one, that considers the possibility that the simple software is “perfect” [Littlewood 2011]. The approach provides considerations of diverse software systems in reliability modeling, and can serve as a starting point for assessing the potential dependencies between two diverse systems.

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 scope 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 the systems are considered diverse or not—is beyond the scope of this study. However, due to its potential importance to plant risk, the need for further research into software CCF is noted in Section 6.4.

## 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 the QSRMs [Chu 2010], which provides more description of the methods 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, or to justify the resources required, should a formal expert elicitation process be used for this purpose.

[8] N-version programming is a diversity strategy to reduce the likelihood of potential software CCFs, and is accomplished by using different software development teams to develop software using the same specifications.

### 2.2.1 Software Reliability Growth Methods

*Characteristic 1 – Method description*

Software reliability growth methods[9] are used in determining whether a piece of software is ready to be released, or how much more testing is needed before release. These models are failure-rate-based, and are not directly applicable to protection systems for which a failure-on-demand probability needs to be estimated. Essentially, these models assume that the system failure rate decreases in time as more software faults are identified and fixed, and use test results to estimate the parameters of the models. They are the most developed and widely used QSRMs, and they and their applications are well described in the literature. As such, they fully meet Characteristic 1.

One limitation, in which the software reliability growth model (SRGMs) are not directly applicable to protection systems for which a failure-on-demand probability needs to be estimated, 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). One of their advantages is that the software failure occurrence times 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.

*Characteristic 2 – Reasonable assumptions*

Some assumptions made in individual SRGMs may be incorrect, for example, assuming that a fault is fixed perfectly. However, as discussed by Chu [2010], for models based on empirical formulas, the focus is primarily on how well they predict/fit the data. As such, SRGMs are rated "Maybe" for Characteristic 2.

*Characteristic 3 – Consideration of operating conditions*

As discussed in Section 2.1, this characteristic is a limitation common to many QSRMs. Since SRGMs do not account for the software context, they do not meet Characteristic 3.

*Characteristic 4 – Consideration of life cycle quality*

Since SRGMs do not explicitly account for the quality in carrying out life cycle activities, they do not meet Characteristic 4.

*Characteristic 5 – Use of data*

SRGMs depend heavily on the 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.

[9] 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.

*Characteristic 6 – Addressing uncertainty*

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 an SRGM's parameter uncertainty is not typically assessed, this is not an inherent model limitation. For example, Musa described an approach for handling parameter uncertainty for software reliability [Musa 1987]. Alternatively, a Bayesian analysis could be used to determine parameter uncertainties. Applying these approaches would satisfy Characteristic 6.

*Characteristic 7 – Verification and validation*

SRGMs have previously been demonstrated with test data from the National Aeronautics and Space Administration (NASA) and military applications (see Chapter 7 of Lyu [1996]), meeting Characteristic 7.

*Characteristic 8 – Demonstrating high reliability*

SRGMs require failure data on debugging tests or operating experience. In order to demonstrate a high reliability, a few failures in a very large number of tests and/or operational demands are needed. However, it is not likely that the large number of tests or operational demands is available for a safety-related nuclear power plant system. This will make it difficult to satisfy Characteristic 8. A "No" ranking is assigned.

*Characteristic 9 – Software CCF*

As discussed in Section 2.1, SRGMs cannot be used to estimate software CCF parameters. They do not meet Characteristic 9.

*Characteristic 10 – Availability of data*

SRGMs require software debugging/testing data to estimate model parameters. Such data is normally proprietary, and is therefore not publically available. This makes it difficult to meet Characteristic 10.

### 2.2.2 Bayesian Belief Network Method

*Characteristic 1 – Method description*

A BBN is a probabilistic graphical model depicting a set of random variables and their conditional independencies[10] via a directed acyclic graph. Here, "acyclic" means that 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. A few studies have been performed by the nuclear industry, for instance, by Gran [2002a] and Eom [2009]. A basic idea from these studies is that the quality in carrying out the software development

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

activities, which includes both software product quality and software development process quality, determines the reliability of the software.

*Characteristic 2 – Reasonable assumptions*

An important assumption of a BBN is the conditional independence implied in its structure [Jensen 2002, Chu 2010]. One way to confirm the correctness of the BBN structure is to ask whether 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.

*Characteristic 3 – Consideration of operating conditions*

Since the applications of the BBN method do not account for the software context, it is assumed that the BBN method does not meet Characteristic 3.

*Characteristic 4 – Consideration of life cycle quality*

A BBN uses conditional probability tables to represent interdependency among disparate events, and, in PRA applications, can potentially combine qualitative information, such as quality in carrying out software life cycle activities (thereby meeting Characteristic 4), with quantitative information, such as test and operational data (meeting Characteristic 5).

*Characteristic 5 – Use of data*

See the preceding discussion.

*Characteristic 6 – Addressing uncertainty*

With a BBN approach, uncertainties associated with the events/nodes are represented explicitly in their probabilities, thereby meeting Characteristic 6.

*Characteristic 7 – Verification and validation*

The BBN method was used in modeling both software failure rates and probabilities [Gran 2000, Eom 2004]. These applications appear to be exploratory in nature, even though a few experts have stated that the method is promising [Littlewood 2000], earning a “Maybe” for Characteristic 7.

*Characteristic 8 – Demonstrating high reliability*

Gran [2000] formulated a BBN 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 estimates of very high reliability, restrictions that kept it from applying to highly reliable nuclear protection system software were not identified either. A “Maybe” rating was assigned.

*Characteristic 9 – Software CCF*

As was discussed in Section 2.1, the inability to account for software CCF between diverse systems is a common limitation of the current QSRMs.

*Characteristic 10 – Availability of data*

Applications of the BBN method often use discretized probability distributions to represent the nodes, and subjective expert elicitation in estimating the conditional probability tables representing the impacts that parent nodes have on child nodes. In practice, efforts used to develop conditional probability tables are very significant, with expert elicitation as a resource-intensive part. It is expected that the qualifications of the experts and the validity and credibility of the inputs they provide have to be scrutinized carefully. All of these concerns result in a "Maybe" rating for Characteristic 10 for BBN.

### 2.2.3 Test-Based Methods

*Characteristic 1 – Method description*

Test-based methods fall into two categories (i.e., black-box and white-box methods[11]), depending on how the tests are performed. These two methods employ different quantification approaches, and both are documented comprehensively, thereby meeting Characteristic 1. A few studies have been performed for the nuclear industry, for example, by Miller [1992], May [1995], Littlewood [1997a], and Zhang [2004].

*Characteristic 2 – Reasonable assumptions*

In general, any method that uses test data can be considered a test-based method, and is subject to the QSRM limitations discussed in Section 2.1. In particular, the common assumption that the software's testing profile is representative of its operational profile is often not well supported. Accordingly, all test-based methods are rated as "Maybe" for Characteristic 2.

*Characteristic 3 – Consideration of operating conditions*

A basic requirement of statistical testing is that the test cases be sampled from its operational profile, which is expected to be an approximation of the operating condition. Therefore, in general, statistical testing methods satisfy this characteristic. A "Maybe" was assigned because the methods have not been applied to specific contexts defined in an NPP PRA. Section 5 describes how the black-box methods can account for the specific contexts defined in a PRA: that is, an operational profile is developed for each context defined by the PRA model, thereby likely meeting Characteristic 3.

*Characteristic 4 – Consideration of lifecycle quality*

Since test-based methods do not specifically account for the quality of life cycle activities, they do not meet Characteristic 4.

[11] The definitions of these two methods are vague in the literature. Many times knowledge gained from white-box testing is used to guide the black-box testing. In literature, such testing practices are called "gray-box."

*Characteristic 5 – Use of data*

Test-based methods perform tests and use the results in quantifying software failure probability, thereby meeting Characteristic 5.

*Characteristic 6 – Addressing uncertainty*

Black-box test-based methods use standard statistical methods, including parameter uncertainty analysis, to quantify software reliability, thereby meeting Characteristic 6. White-box methods develop their own quantification methods, which take into consideration the structure of the software (e.g., Zhang [2004] derived formulas for calculating an averaged system failure probability over different nodes and paths in the software) and parameter uncertainty.

*Characteristic 7 – Verification and validation*

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 the 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 in only a few studies (e.g., Zhang [2004] and May [1995]). Zhang [2004] considers the internal structure of the software. These methods require 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 account for information about the internal structure of the software, and may detect faults that would otherwise remain undetected. However, it may be difficult to determine the set of inputs (that is, the partition) that would force the execution of a software program to follow a specific path. Also, these white-box test-based methods estimate the software failure probability from the failure probabilities of individual paths 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 [Chu 2010]. This suggests that a much larger number of tests would be needed, as compared with the black-box testing method. 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 the 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 is essentially the same as that for black-box testing methods.

*Characteristic 8 – Demonstrating high reliability*

As was mentioned in Section 2.1, automated testing may be a means of undertaking a large number of tests, thereby potentially allowing black-box methods to satisfy Characteristic 8. A rating of “maybe” was assigned.

*Characteristic 9 – Software CCF*

As was discussed in Section 2.1, the inability to account for software CCF between diverse systems is a common limitation of the current QSRMs.

*Characteristic 10 – Availability of data*

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.

### 2.2.4 A Correlation Method Using Software Development Practices

*Characteristic 1 – Method description*

Neufelder's correlation approach [2002], which was implemented in a computer code called "Frestimate" [SoftRel 2009], provides a point estimate for the software failure rate by making use of the software engineering practices of past software development projects. Because of the unavailability of detailed information on the past software development projects and the correlation/regression analyses used to construct its predictive model, the method could not be evaluated in detail, and therefore does not meet Characteristic 1.

*Characteristic 2 – Reasonable assumptions*

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. A rating of "Maybe" was assigned.

*Characteristic 3 – Consideration of operating conditions*

Since it does not explicitly account for the software context, it does not meet Characteristic 3.

*Characteristic 4 – Consideration of life cycle quality*

Similar to the BBN methods discussed in Section 2.2.1, the method attempts to account for the quality in carrying out software development activities, thereby meeting Characteristic 4.

*Characteristic 5 – Use of data*

The general concept of undertaking correlation/regression analyses using past software development experience/data is reasonable. The resulting estimate can be further updated with test data and operating experience, thereby meeting Characteristic 5.

*Characteristic 6 – Addressing uncertainty*

Since it only provides point estimates, it does not meet Characteristic 6.

*Characteristic 7 – Verification and validation*

Since independent organizations have not validated or benchmarked the method, it only partially meets Characteristic 7. A rating of "Maybe" was assigned.

*Characteristic 8 – Demonstrating high reliability*

The authors are unaware of any peer-reviewed, publicly available literature that provides a technical basis for calculating ultra-high reliability failure probabilities better than $10^{-4}$ using the Frestimate method. Therefore, the method does not meet Characteristic 8.

*Characteristic 9 – Software CCF*

As was discussed in Section 2.1, the inability to account for software CCF between diverse systems is a common limitation of the current QSRMs.

*Characteristic 10 – Availability of data*

Information on the previous projects is an essential part of the method, and its proprietary nature leads to a rating of "Maybe" for Characteristic 10.

### 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 (and 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.[12]

[12] The NRC has sponsored additional work on metrics-based methods. While this work was not completed in time to be included in this study, it may be considered as part of the follow-up case studies.

### 2.2.6 Standard-Based Method

International Electrotechnical Commission (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, thereby meeting Characteristics 1 and 4 (the only characteristics satisfied). For example, using formal methods to develop 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 rates and probabilities 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, made 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, or 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 rates and probabilities does not account for this variability, nor does the standard address other forms of uncertainty; thus, this method fails to meet Characteristic 6. Also, since IEC Standard 61508 neither accounts for the software context nor addresses 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 (CSRM) [Guarro 2007] is an overall integrated risk-modeling approach that incorporates hardware, software, and their static or dynamic interactions, and has been demonstrated by a NASA application. The probability of software failures is quantified using a test method in the NASA application. BNL does not consider it a candidate QSRM for the following reasons:

1. The dynamic scenario modeling does not fit the immediate needs of current NPP PRAs, which are static scenario (event tree/fault tree)-based.
2. CRSM's software reliability component is essentially a black-box testing method and is covered in this study in Section 5, where the underlying concept of CSRM is used with PRA-defined contexts and thermal hydraulic modeling instead of the dynamic modeling of the CSRM.

N-version programming is a diversity strategy to reduce the likelihood of potential software CCFs, and is not a QSRM. Accordingly, it is not evaluated against the set of desirable QSRM 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 whether different versions fail concurrently. N-version programming experiments, such as those performed by Lyu [2005], demonstrated that multi-versions can improve software reliability, that is, multi-versions do not fail concurrently at all times. On the other hand, software failures are correlated, that is, the software does fail concurrently in some cases.

## 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 has all of the desired 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. The BBN method can account for the quality in carrying out software life cycle activities while the test-based methods use standard statistical methods, including treatment of parameter uncertainties. These methods also appear to have great potential for demonstrating high software reliability. (It should be noted that the Metrics-MTTF method is a type of 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 BBN method requires significant support from experts to build the BBN network and provide inputs to the network. The effort of soliciting expert opinions needs to be estimated, and the expert opinion elicitation process needs to be structured and engineered. One 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 its implementation requires extra effort and has no obvious benefits, as pointed out earlier and further discussed in Section 5.1. 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. Due to the limitations of the test-based methods, and to account for the quality in carrying out software life cycle 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 can be very resource-intensive and the evidence needed for quantification may be lacking, both a simplified and a detailed BBN 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 the detailed BBNs 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 examine the usefulness of the BBN method.

SRGMs were also found to meet several of the desirable characteristics, as seen in Table 2-1. They employ information collected during software development, that is, the results from debugging, and are the most commonly 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 comparison results of the SRGM approach against the desirable characteristics are assumed to be applicable to the DSRGMs. 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 on other limitations associated with this method, it was ultimately decided not to develop a prior distribution using the DSRGM approach.

Sections 4 and 5 describe how a QSRM can be developed for a digital protection system using each of the two selected methods, in light of the common limitations of the current methods described in Section 2.1.

**Table 2-1 Evaluation of QSRMs against desirable characteristics.**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **SRGM** | Y | M | N | N | Y | Y | Y | N | N | N |
| **BBN** | Y | M | N | Y | Y | Y | M | M | N | M |
| **Test-based (black-box)** | Y | M | M | N | Y | Y | M | M | N | Y |
| **Test-based (white-box)** | Y | M | M | N | Y | Y | N | M | N | Y |
| **Frestimate** | N | M | N | Y | Y | N | M | N | N | M |
| **Metrics-MTTF** | Y | M | M | N | Y | Y | M | M | N | Y |
| **Metrics-Defect Density** | Y | M | M | N | M | N | N | M | N | Y |
| **Standard-based** | Y | N | N | Y | N | N | N | N | N | N |

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 considers the specific operating conditions of the software.
4. The method considers the quality in carrying out life cycle 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 ~$10^{-5}$, commensurate with an analog reactor protection system).
9. The method can estimate parameters that can be used to account for software common-cause failures of diverse protection systems or channels.
10. The data and necessary information exist and can be collected.

Legend

| | |
|---|---|
| Y | **YES** – Method conceptually meets the characteristic and its implementation is deemed practical |
| M | **MAYBE** – Method may conceptually meet the characteristic and/or its implementation may not be practical |
| N | **NO** – Method does not conceptually meet the characteristic |

# 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, it is preferable that the example system be such a system. To account for the quality in carrying out software development activities using the BBN method, it is desirable to obtain information associated with the activities, such as verification and validation reports. In addition, due to the desire to release the details of the case study to the public, 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 protection systems for many of the advanced reactors may only be designs on paper, and those that have been built may be proprietary.

The Oconee Nuclear Station recently received regulatory approval from the U.S. Nuclear Regulatory Commission (NRC) for digital upgrades to its reactor protection system (RPS) and engineered safeguard protection system (ESPS) [NRC 2010b], and the Wolf Creek Generating Station received NRC approval to replace its Main Steam and Feedwater Isolation System controls [NRC 2009]. Systems such as these would be ideal example systems for the study in this report because a large amount of design information, including the 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 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 the National Aeronautics and Space Administration (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. Brookhaven National Laboratory also explored the protection systems at BNL accelerator facilities (i.e., the personnel access control systems). Due to a lack of detailed documentation for these systems, they were found to be unsuitable.

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's 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. Additional system details will be included as part of the subsequent documentation of the case studies.

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. Additional system details will be included as part of the subsequent documentation of the case studies.

## 3.2 System Description

The ATR (see Figure 3-1) is the third generation of test reactors built at INL's Reactor Technology Complex 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 and 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.

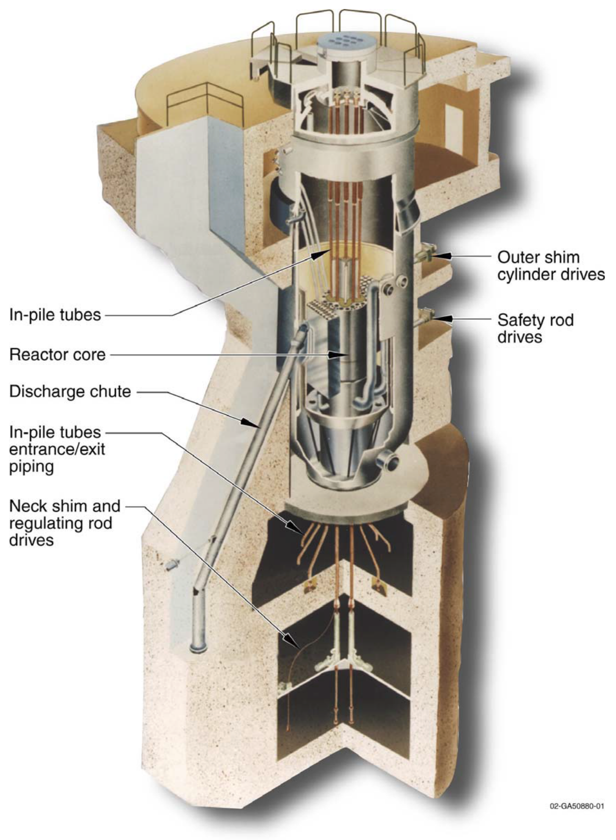
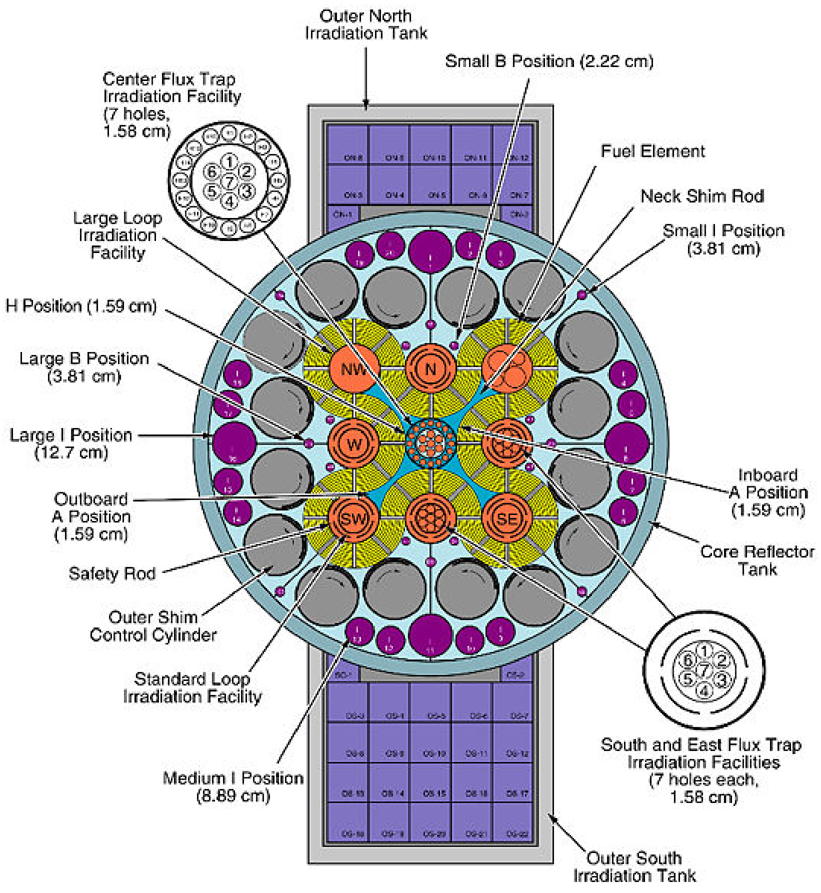

Outer shim cylinder drives
Safety rod drives
In-pile tubes
Reactor core
Discharge chute
In-pile tubes entrance/exit piping
Neck shim and regulating rod drives
02-GA50880-01

**Figure 3-1 Advanced Test Reactor at Idaho National Laboratory**



Outer North Irradiation Tank
Small B Position (2.22 cm)
Center Flux Trap Irradiation Facility (7 holes, 1.58 cm)
Fuel Element
Neck Shim Rod
Small I Position (3.81 cm)
Large Loop Irradiation Facility
H Position (1.59 cm)
Large B Position (3.81 cm)
Large I Position (12.7 cm)
Outboard A Position (1.59 cm)
Safety Rod
Outer Shim Control Cylinder
Standard Loop Irradiation Facility
Medium I Position (8.89 cm)
Inboard A Position (1.59 cm)
Core Reflector Tank
South and East Flux Trap Irradiation Facilities (7 holes each, 1.58 cm)
Outer South Irradiation Tank
NW
N
W
SW
SE

**Figure 3-2 Core cross section of Advanced Test Reactor**

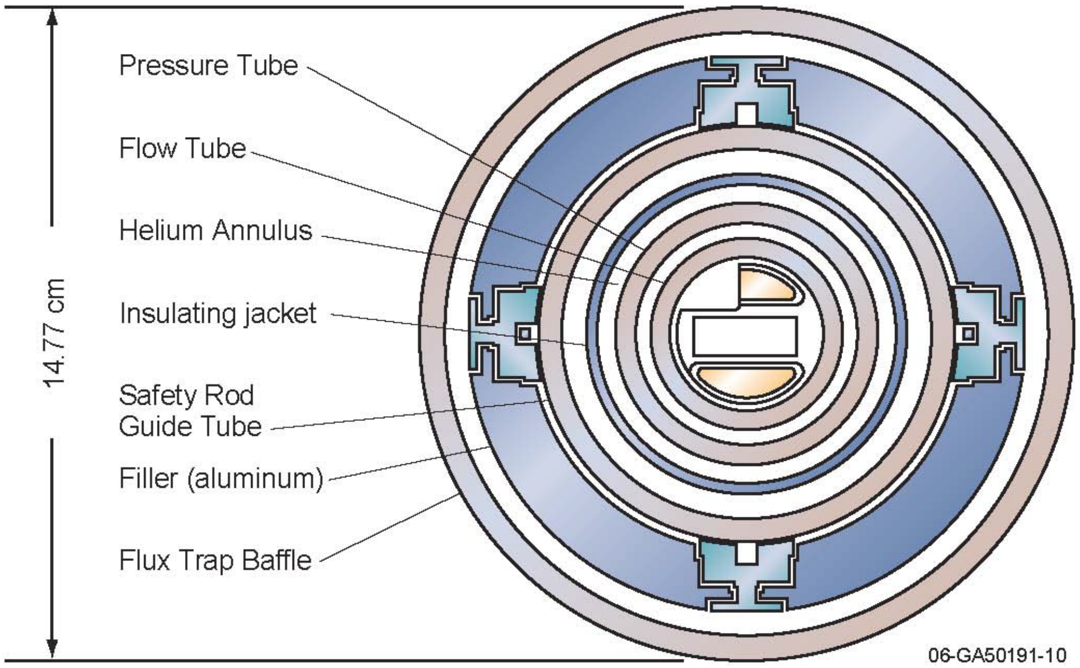


**Figure 3-3 Cross section of the in-core portion of a typical pressurized water loop**

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).

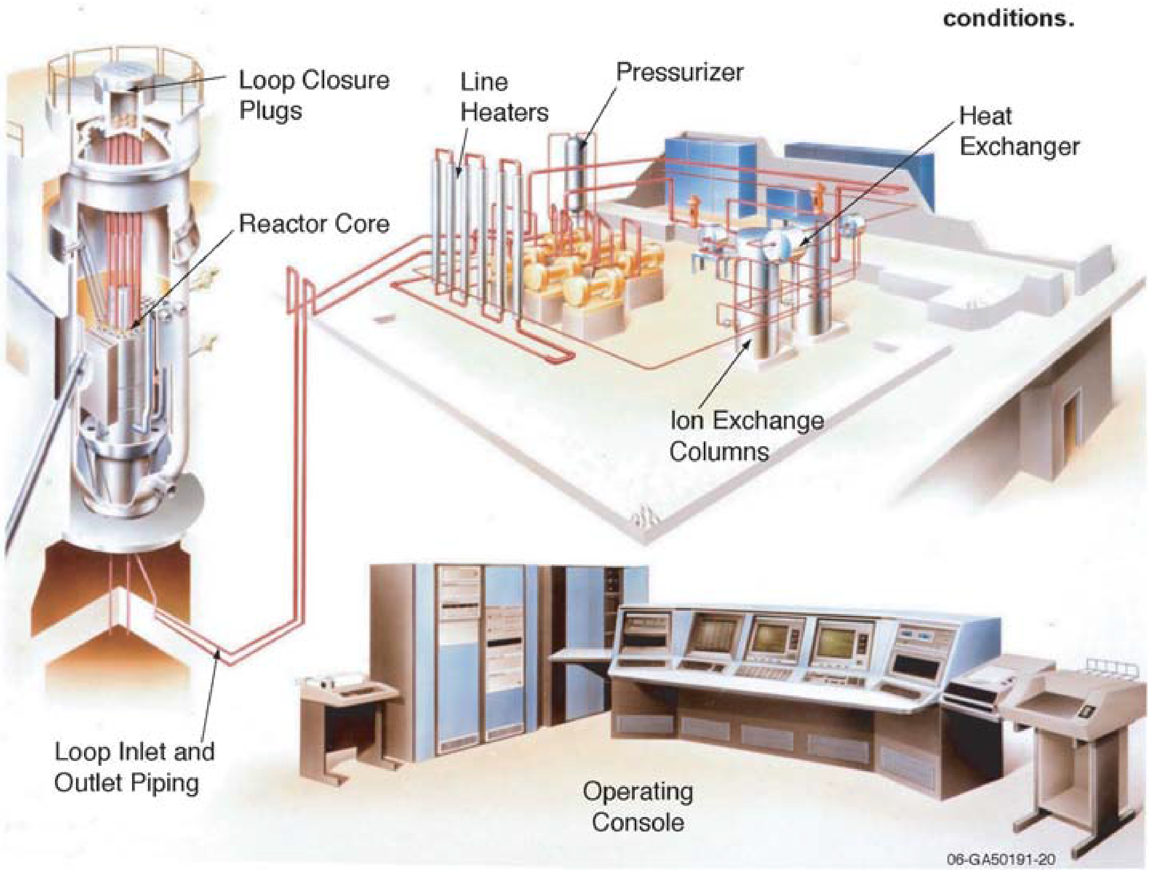


**Figure 3-4 Pressurized water loop system**

## 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 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 Operating 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) consisting 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 eight workstations and the remote processing units communicate 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 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 of the information necessary to control a loop facility. One of the two pairs of redundant power supplies is used to provide power to the electrical equipment in the remote processing unit and transmitters, and to provide power to the field optical-to-electrical interface units for communicating 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:

1. primary coolant pumps
2. loop line heaters
3. loop pressurizer heaters
4. makeup pumps
5. purification flow control valve
6. makeup system recirculation pump control
7. conductivity flow control

and performs the following process functions:

1. primary coolant flow control function
2. primary coolant temperature control function
3. primary coolant pressure control function
4. pressurizer level control function
5. loop degassing flow control function
6. ion exchange column flow control function
7. 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's IPT loop, there are seven loop protective functions provided by this system by monitoring the following parameters:

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 ($\Delta T$)
7. Loop coolant pump power supply

In addition, there is a maximum useful capacity holder (MUCH) experiment flow (low) protection function applicable to loops 1C-W and 1D-N. 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 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 the 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.

As indicated by INL [2008], Chapter 15 of the ATR safety analysis report SAR-153 contains the 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 the development of the BBN model, which evaluates the quality in carrying out 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 the American Society of Mechanical Engineers's NQA-1 [ASME NQA]. The quality level determination for the ATR loop DCS software was made as Quality Level 2, and the software was determined to be non-safety, configurable software[13].

The software-related documentation provided by INL does not contain information describing the software development activities for the recent upgrade of the ATR Loop DCS that was originally installed in 1993. 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[14], and contains some information on 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 that information on 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. PLN-3137 provides a list of references in which standards for the design/development of the LOCS-DCS can be found. Additional standards that are applicable to design and development are also provided in PLN-3137.

[13] Idaho National Laboratory, "Software Quality Assurance Plan (SQAP): ATR Loop Distributed Control System Upgrade," Revision 0, 2009. Not publicly available.
14 Idaho National Laboratory, "Configuration Management Plan Advanced Test Reactor Loop Operating Control System Distributed Control System," Revision 0, 2009. Not publicly available.

### *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[15]. At the beginning of the software development, requirements were included as a baseline[16] as part of the design document, addressing both the technical and functional requirements of the software application. The requirements specifically addressed operating system interface, performance, installation, design input, and design constraint issues[15]. The requirements were used in conjunction with subsequent requirement and design documents to determine whether 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[13].

### *2. Design Phase*

In the design phase, the designs for the architecture, components, and interfaces of the software were created, documented, and verified to satisfy the requirements[15].

### *3. Implementation Phase*

During this time period, the software product was created and debugged[15].

### *4. Test Phase*

In test phase, the components of a software product were evaluated and integrated, and the software product was evaluated to determine whether or not all of the requirements had been satisfied[15].

For the ATR, Loop DCS V&V was accomplished by performing a series of tests, including the factory acceptance test (FAT), the grooming test, and the system operability (SO) test[17].

FAT 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 FAT was performed at Metso and witnessed by the Battelle Energy Alliance (BEA) after BEA approved the test procedure. In general, the FAT environment should be comparable to the environment in which the equipment would be installed and used.

Prior to 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 that the implemented installation met 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 satisfied the

[15] TRA Construction Projects Management, "Software Verification and Validation Report, ATR Loop Operating Control System (LOCS) Upgrade Project," Revision 0.0, 1997. Not publicly available.
[16] A baseline is a specification or product that has been formally reviewed and agreed upon, that serves thereafter as the basis for further development, and that can only be changed through formal change control procedures.
[17] Idaho National Laboratory, "Verification and Validation (V&V) Plan: ATR Loop Distributed Control System Upgrade," Revision 1, 2009. Not publicly available.

requirements. If the system did not pass the grooming tests, the errors had to 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[17].

### *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 performed as required.

# 4 DEVELOPMENT OF A BBN METHOD

The BBN method has been a topic of research in many different areas, such as medical diagnosis [Lauritzen 1988], risk assessment of natural hazards [Regamey 2006, Bensi 2010], and 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 necessary for diagnosing neurological disease. Regamey [2006] developed a BBN 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 [Horvitz 1998].

Some experts consider the BBN method to be a promising way of quantifying software reliability [Johnson 2000, Littlewood 2000, Dahll 2007, Eom 2009]. The basic idea is that the quality in carrying out the software development activities determines the reliability of the software [Neil 1996, Johnson 2000, Littlewood, 2000, and Dahll 2007]. However, only a few studies by the Halden Reactor Project, the VTT Technical Research Centre of Finland, and the Finnish government authority for the nuclear industry (STUK) [Helminen 2001, 2003a, 2003b, 2005, and 2007; Gran 2000, 2002a, and b] have used a BBN to quantify software reliability. A recent study by the Korea Atomic Energy Research Institute (KAERI) [Eom 2009] developed an approach to estimating the number of software faults based on the quality in carrying out software development activities, without quantifying software failure probabilities. The basic idea behind the BBN application is that better quality results in fewer faults in the software.

Brookhaven National Laboratory proposes using a BBN approach to generate a prior distribution for the probability of software failure of a protection system. Then the context-specific statistical testing results (as described in Section 5) can be used in a Bayesian update for this prior. Note that the Bayesian update component can easily be integrated into the BBN framework as a "grand" BBN model. In this study, BBN and statistical testing methods are discussed in separate sections because they can be used as distinct methods.

In this section, possible methods are examined for quantifying software-failure probabilities using BBNs. Issues and limitations of each approach are discussed. As applications of BBN might heavily depend on expert opinion or other evidence, the BBN output uncertainty is expected to be significant. Discussion of the sources of uncertainties is provided throughout this section, and possible treatment of the uncertainties is discussed in Section 4.2.5.

## 4.1 Introduction

A BBN 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 nodes[18] might be based on theoretical models, empirical relations, expert opinion, engineering judgment, or any mixture of them. As discussed earlier, developing BBNs for quantifying software reliability essentially involves (1) constructing a BBN that models the causal relationship between the software failure probability and the quality in carrying out software development activities and (2) evaluating the quality in carrying out software development activities against the guidance/requirements in such standards as the NRC's

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

Branch Technical Position (BTP 7-14) [NRC 2007] and other factors that may affect software reliability, and applying the results to the BBN model. The results of the BBN can then be used as the prior distribution of the software failure rate or probability, and a Bayesian update of the prior can be performed using testing/operating data to produce a posterior distribution of the software's reliability. As mentioned previously, the Bayesian update process can be incorporated directly into the BBN, or can be applied as a separate, follow-up analysis.

BBN methods are capable of aggregating disparate information about software (e.g., aggregation of software failure data and quality in carrying out software life cycle activities, which 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, including 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 in carrying out software development on the software's 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 significant. Therefore, the BBN model's application to safety-related systems that are expected to have a very small failure probability may be overwhelmed by the uncertainty (for instance, a lognormal distribution with the mean value of $10^{-5}$ but an error factor of 100). In the BBN 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. Finding experts capable of estimating this type of information and willing to participate is a challenge. The accuracy with which experts can make such estimations is highly uncertain. However, a structured elicitation process with well-trained experts might be an effective solution to reduce the uncertainty. BNL also recognizes that the estimation of the number of faults poses less uncertainty than the direct estimation of the failure probability, because some data on the number of faults may be available. However, estimating failure probability for a given number of faults is a different challenge.

In this study, two BBN studies (i.e., [Gran 2002a] and [Eom 2009]) that assessed the quality in carrying out software development activities are summarized, and additional published information was collected and used in proposing possible enhancements to the approaches, such that they might possibly be applied to modeling protection systems of a nuclear power plant. Section 4.2 describes an approach for developing a BBN to quantify the probability of failure on demand, along with its associated issues and limitations. Section 4.3 details the steps in implementing the approach. Section 4.4 summarizes the evaluation of this enhanced BBN approach against the desirable characteristics developed in Section 2, highlighting the fact that the newly developed BBN method performs better than the ones in the literature with respect to the 10 characteristics.

## 4.2 The BNL BBN Method

### 4.2.1 Development of a BBN that Captures the Quality of Software Development Activities

Guidance on carrying 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 considered the 10 life cycle 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 the NRC BTP 7-14 [NRC 2007]. The BBNs in both studies are examples of how to formulate a BBN to evaluate the quality in carrying out software development activities. In general, the qualitative guidance/requirements in the standards are used to define those nodes in the BBNs that represent the quality in carrying out software development activities [Gran 2002a and Eom 2009], and these 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 software program [Eom 2009]. Since guidance, such as BTP 7-14, is used in licensing, the vendors and licensees need to demonstrate, and the U.S. Nuclear Regulatory Commission (NRC) needs to verify, that the relevant guidance is followed. Therefore, useful information should be available from these associated evaluations, such as verification and validation reports. Furthermore, since the people involved in the evaluations are familiar with the guidance, they can potentially serve as experts for developing the BBN and participate in expert elicitation in quantifying node probability tables (NPTs).

The next two subsections describe two methods for obtaining the software failure probability, depending upon the information provided in the BBNs. The number of faults is the intermediate result from the first method.

### 4.2.2 Conversion of Number of Faults to Software Failure Probability

In the BBN studies [Eom 2004 and 2009] by KAERI, there was no attempt to compute the RPS software's failure rate or probability of failure. In particular, the Eom study [2009] focused on using the BBN to estimate the number of faults remaining 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 previous phases. This approach is similar to that used by Fenton [2007], which is based on the understanding that a poor-quality development increases the number of defects likely to be present, and high-quality testing increases the proportion of defects found. 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 estimates of the number of undetected ones. Furthermore, KAERI adopted the waterfall model of software development, and the interconnection between the previous-phase BBN and the current one passes undetected faults from phase to phase [Eom 2009].

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

1. Delic et al. [1997] developed a simple BBN for estimating the probability of failure-on-demand for a software program by using a "size distribution" of faults [Littlewood 1980],

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 by the same vendor. One 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 other projects.

2. Musa [1987] introduced the concept of a fault exposure ratio that converts the number of faults into software failure rates. It is consistent with the assumption commonly used in software reliability growth models (SRGMs) that software failure rate is proportional to the number of faults in a software program. In addition to providing a formula and a procedure to calculate this fault exposure ratio for a specific software, Musa provided a representative constant fault exposure ratio based on a number of Bell Lab software projects [Musa 1987]. This might be considered a less detailed version of Delic's model [Delic 1997] by integrating over the size distribution of the faults. The same idea was used in more recent studies/methods (i.e., Smidts [Smidts 2004] in her metrics-based methods and Neufelder [Neufelder 2002] in her correlation method). A direct use of Musa's constant may be problematic, since software projects differ from each other in many ways (e.g., in terms of complexity), and the constant 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 are assumed to be reflected solely by the number of faults. It is questionable whether such a simplified constant adequately estimates the failure probability, since it does not account for many other factor variations that can affect this probability (e.g., the software's complexity). Another concern is the general applicability to current software when the constant is derived with data obtained from the 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 to the software being evaluated, they are preferred to the generic constant fault exposure ratio. In addition, data from safety-related software should not be mixed with those from non-safety-related software.

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 developed by 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 is distributed among the different paths/parts according to some rules, such as 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 must first be determined with an estimated number of faults for the path or part. The path-/part-specific failure probability is then 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].

BNL's proposed case study focuses on fault(s) located in the code for the protective logic that may cause the reactor protection system (RPS) to fail. A safety-related software program, such as the RPS software, also commonly performs several auxiliary functions, such as self-testing. There is still a possibility that a fault relating 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.

### 4.2.3 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, 2002a, and 2002b]. In particular, Gran [2002a and b] developed models accounting for the quality in carrying out software development activities and the problem complexity, and used the results of the quality evaluation to estimate software failure probability by expert elicitation. He used expert elicitations to build BBNs, including node probability tables, to convert qualitative evidence to quantitative evidence, and to collect evidence for inferences [Gran 2002a and 2002b]. After collecting evidence for a specific application's 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 4.1, the accuracy with which experts can estimate software failure probability is likely to be highly uncertain; no known benchmark studies have been undertaken to validate the findings from expert elicitation. The potential use of the failure likelihood index method (FLIM) [Chu 1995] to estimate software failure probability is under BNL's investigation. In FLIM, factors that affect software performance are used to estimate software failure probabilities. Similarly, Neufelder's correlation method [2002], which uses the Frestimate computer code, converts process/quality information into the number of faults, which is then converted to a failure rate. 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 quantitative information. The same idea might be extended to directly estimating software failure probability. The limitations here are the lack of data on past software projects and the method's applicability to specific software. In addition, a benchmark system, with operational data available for the analysis, is necessary to validate the software reliability assessments.

### 4.2.4 Summary of Assumptions and Issues

The BBN method can combine disparate information, such as quality in carrying out software development activities and statistical test data. In practice, converting the qualitative information obtained from assessing software development activities into quantitative information is often difficult, and introduces significant uncertainties. This section discusses some of the key assumptions and issues associated with using the BBN approach to estimate 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 the lack of justification for relating quality in carrying out software development and the number of faults with reliability. Converting the number of faults to the software failure rate via the constant 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. It should be pointed out that it is widely assumed in SRGMs that the failure rate is proportional to the remaining fault content of the software. Such a simple relationship also implies that the fault contributions to the software failure are equal, and that the contribution from each fault can be characterized by a constant similar to the fault exposure ratio. Employing the “fault size distribution” [Littlewood 1980, Delic 1997] is limited to software that was developed for a particular vendor, and that has operational data from similar software projects. The problem of lacking failure experience parallels the dearth of data for human reliability analysis (HRA). In light of recent benchmark analyses of human reliability (e.g., NRC [2011]), it would be beneficial to perform benchmark exercises to validate or calibrate a BBN by collecting operational data on past software development projects and/or performing the statistical testing described in Section 5.

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, these data are most likely proprietary. If adequate operating experience for a software program of a protection system and the associated detailed information needed for developing a BBN are available, the BBN 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 recent experience with HRA [NRC 2011], it may require the completion of many such studies before confidence is developed in applying the BBN method to estimate probability distributions for software failure.

The difficulties in carrying out benchmark studies should not be underestimated: (1) the lack 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 for on-demand failures) that are suitable for quantifying software reliability, and (2) the applicability of the data available if it was 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 5) 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 whether the BBN’s prediction is consistent with the statistical testing results. Establishing 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.

Any mathematical model (for example, a fault tree or event tree) can be considered a mathematical representation of the underlying physical, logical, and causal relationship/phenomena. The BBN method is a mathematical method that uses a BBN to represent or model the causal relationships of random variables, such as how the quality in carrying out the software development activities affects the number of faults inserted. 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 based on engineering knowledge of the nuclear power plant and failure data of similar equipment, respectively. The usefulness of the BBN method also 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. For the other applications discussed earlier, each had at least an underlying model or data, so their BBNs were used only as a mathematical representation of the model and/or data.

2. As was discussed in Section 4.2.3, Neufelder's correlation method [2002] uses proprietary data on past software projects in a regression analysis to estimate the number of faults remaining in a software program and the software failure rate. 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 faults remaining in the software program and did not include software failure rates and probabilities, and her use of fault exposure ratio suffers from the limitations previously discussed. Also, the information she used regarding past software development projects is proprietary, the projects may be outdated, and the nuclear operation data is lacking. Regression analysis, in general, is a potential method that relates software development practices and software failure probability. It may suffer from the same limitations as Neufelder's method.

*BBN Structure*

1. To capture the quality in carrying out software development activities, it is desirable to develop a BBN using the structure of a software development standard, such as BTP 7-14 [NRC 2007] for U.S. nuclear power plants. The existing BBNs by Gran [2002a] and Eom [2009] use the standard DO-178B [RTCA 1999] and the KNICS procedure, respectively; they convert the guidance/requirements in the standards into nodes of the BBNs. Consequently, the BBNs are very complex, with many NPTs 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 critique of detailed models, suggesting that they require detailed assessments where potentially little data are available. However, since the standards are used to decide whether a software program 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 guidance/requirements of the standard, even though they will not necessarily know its reliability.

   Alternatively, it would be simpler to develop a higher-level model, representing, for example, 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.

2. A BBN can be built based on the causal relationship between 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 it is generally not easy to formulate any NPTs, it is even more difficult to build high-dimension NPTs or distributions,

due to the need to carefully consider the subtle interplays between the parent nodes on their impacts on the child nodes. A typical case in a software development cycle can be to evaluate the quality of inspection (a high-level node) [Eom 2009], which 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 that the higher-level node has on it. This makes building the BBN NPTs easier, compared with the case in which arrows are pointing from the lower-level nodes to the higher-level node, which requires the 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, the two types of modeling should be compared using simple examples to further evaluate their differences.

*Other Potential Limitations*

1. Unexpected or controversial results may surface in applying the collected software evidences to a BBN [Littlewood 2007]: that is, the parameters and structure of a BBN may require revisiting and possibly revising when counterintuitive findings emerge.

2. Two types of evidence might be collected for a BBN: 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 (i.e., it is very unlikely that an expert will provide a distribution for a node that exactly matches the one obtained by exercising the BBN). On the other hand, some analysts [Pan 2006] use the Jeffreys rule, which 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.

### 4.2.5 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 distributions that assess the quality in carrying out software development activities, it is desirable to use software engineering experts to develop evidence 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 BBN complexity on the results, two BBNs can be developed, a simpler one and a more complex one. As was discussed earlier, benchmark studies can validate or invalidate the BBN: that is, for this purpose, information on software development activities and the available relevant operational data from past software development projects can be incorporated in applying the BBN. The benchmark exercises can be completed at different levels of detail of the BBN, such as at the overall level and the level of the node probability tables. Alternative methods can also be developed to allow for comparisons with the BBN method. For example, it might be worthwhile to try an extension of the failure likelihood method from HRA 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. Neufelder's correlation method 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 to addressing model 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.

## 4.3 Procedure for Developing the BBN Approach to an Example System

In general, the steps involved in building a BBN include (1) identifying the nodes of interest, (2) constructing the topology of the arcs connecting (some of) these nodes (this is where strong assumptions on conditional independence are made), and (3) constructing the NPTs. In this section, a process is proposed for applying the BBN approach to an example system to estimate its probability of failure on demand. The basic structure of the BBN, if developed appropriately, should be applicable to current NPP digital protection systems. To apply the BBN to a specific piece of software, the evidence needs to be collected for that software to specialize the BBN. The proposed process for developing the BBN for the example system involves the following steps:

1. *System familiarization*

   In addition to gaining an in-depth understanding of the design of the example system, its functions, and the systems with which it interacts, it is very important to understand the software development work and the associated guidance. The necessary documentation includes information on system design and operation, test and operational data, and a description of software development activities, such as debugging records and verification and validation 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 (i.e., BTP 7-14 [NRC 2007]), so that the model is more applicable to systems at U.S. NPPs. 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 records. 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 is probably also the most difficult task in applying BBN to software reliability assessments. Software is the product of a software development life cycle. Each of the activities related to this cycle, which are generally characterized and evaluated qualitatively, has a certain impact on the software's reliability. While there is a general trend in the impacts of these activities on the software reliability or on 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 are probably 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 the lack of operational experience in 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*

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 used by Gran [2002a] might be considered a starting point. If an indirect conversion is used, the high-level BBN discussed by Eom [2009] can be adopted first to convert the qualitative information to the number of remaining faults, after which the high-level structure used by Delic [1997] can be employed to obtain the software's reliability. In either case, the high-level issues indicated previously need to be resolved first.

4. *Development of expert elicitation approaches*

Two types of expert elicitation application should be used. The first type employs generic experts (Type I experts) familiar with software development activities and associated guidance and requirements. The second type (Type II experts) requires people familiar with the development of the specific system under evaluation. The Type I experts will participate in developing the BBN, including its structure, and quantifying the node probability tables. Since the products of the Type I experts are based on generic information, the BBN is a generic one in the sense that this BBN is applicable to any software developed from the guidance. The Type II experts are responsible for providing system-specific answers associated with the observable nodes of the BBN. Two kinds of NPTs must be quantified: those that convert the quality or the number of faults into software failure probabilities (Type I experts), and those that evaluate the quality in carrying out development activities (Type II experts).

There are a number of aspects that need to be considered in performing 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 addressed dictate the identification and selection of the experts. Those chosen need to be trained to make sure they understand the specific issues and questions they are asked to address, which cover the expert opinion elicitation process, the software development life cycle under study, the system under study, the BBN fundamentals, etc. Background information and material on the issues need to be prepared for the experts. Specific questions need to be developed, based on the technical issues. To better formulate elicitation questions, breaking down or refining the issues may be necessary and helpful. Individual responses that are highly uncertain or significantly different are allowed in the elicitation, but significant systematic biases must be identified and avoided. The responses from individual experts are analyzed, aggregated, and documented. Conflicts between experts' responses may need reconciliation.

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

5. *Development of the complete BBN*

The low-level BBN should cover the software reliability-relevant attributes specified in the guidance for developing software (i.e., BTP 7-14) for U.S. NPPs and provide a means of evaluating the quality in carrying out software development activities. Such attributes are abstracted from various software standards, such as Capability Maturity Model Integration, the International Organization for Standardization's standards, the International Electrotechnical Commission's standards, the Institute of Electrical and Electronics Engineers' standards, etc. The PRA experts responsible for developing the model should undertake this task to ensure that 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 desirable to keep the overall BBN structure as simple as possible because, even if a complicated BBN is built that adequately 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 should be defined clearly, including its possible states. Causal relationships between connected nodes should be described precisely. The conditional independence assumptions of the BBN should be verified.

In addition to a detailed BBN, 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 set of guidance, it should cover the general recommendations of the 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 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, such as 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). 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 in carrying out software development activities for the specific software system under evaluation by collecting evidence associated with the observable nodes of the BBN, using available documentation on the system (e.g., debugging data). It also involves eliciting Type II experts for answers to questions about the observable nodes of the BBN (e.g., is the software specification verifiable and its implementation traceable?). Such knowledge can be provided by developers of the software, or by a third party familiar with the software, after they are given the relevant information.

8. *Performing BBN analyses and generating results*

   Standard software tools, such as AgenaRisk® [Agena 2008] and HUGIN [Hugin 2010], can be used in developing the BBN and performing the needed analyses, such as by generating a probability distribution for the probability of failure on demand.

9. *Performing sensitivity calculations to address modeling assumptions and issues*

   The BBN can be used to perform the sensitivity calculations discussed in Section 4.2.5 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 discussed in Section 4.2.5, would require significant additional resources, and is considered beyond the scope of the current plan for the case studies.

10. *Performing benchmark studies to evaluate the BBN*

    As discussed in Section 4.2.4, the statistical testing results of the example system can determine whether the prediction of the BBN is consistent with the test results: that is, the test results do not contradict the BBN prediction (e.g., test results show that the failure probability is in the tails of the distribution obtained from the BBN model). This will be done in the planned case study. Furthermore, benchmarking studies are better able to validate and calibrate the BBN. The benchmarking studies could be applications of the BBN to other software projects, and would evaluate whether the BBN produces reasonable results. However, the extensive benchmark studies are not within the scope of the case study.

## 4.4 Evaluation of the BNL BBN Approach

The BBN method was initially evaluated in Section 2 against the 10 desirable characteristics for a quantitative software reliability method. An evaluation of the new BNL-proposed BBN approach is provided below. The use of the BBN results with statistical testing, as suggested in Section 2, makes it possible to account for the contexts specified in a PRA (Characteristic 3). The benchmark studies suggested in this section can potentially be used to verify a BBN model and support any improvements that may be identified (Characteristic 7).

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. A "Yes" is assigned.

2. *The assumptions of the method have reasonable bases.*

   The conditional independence assumption of a BBN needs to be verified when the BBN structure is developed, and is an issue to consider 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 considers the specific operating conditions of the software.*

   An inherent weakness of the proposed BBN (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 4.2.4. A rating of "Yes" is assigned.

4. *The method considers the quality of life cycle 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 in carrying out software life cycle activities. A "Yes" is assigned.

5. *The method uses available test results and operational experience.*

   The BBN method uses test and operational data in a Bayesian analysis, thereby meeting this characteristic. A "Yes" is assigned.

6. *The method addresses uncertainty*.

   With a BBN approach, uncertainties associated with the events/nodes are represented explicitly in their probabilities. A "Yes" is assigned.

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. After some benchmark studies are done, the BBN can potentially be validated or improved. This would change the rating to "Yes."

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, as the BBN model by Gran [2002b] for a non-nuclear system has been used to demonstrate a failure probability range of lower than $10^{-4}$. A "Maybe" is assigned. Since the objective of this study is only to use a BBN to estimate a prior distribution that will be Bayesian-updated with statistical test data, this characteristic is of lesser importance (because the results of the BBN are expected to have large uncertainties).

9. *The method is capable of estimating parameters that can be used to account for software common-cause failures (CCFs) of diverse protection systems or channels*.

   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 externally to the BBN, as discussed in Section 2.1. A "No" is assigned.

10. *The data and necessary 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 in carrying out 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 necessary experts exist, resulting in a "Maybe" rating for this characteristic. In addition, as was discussed previously, data from past software development projects are not available or are proprietary, but statistical testing may provide some data for benchmarking.

# 5 DEVELOPMENT OF A CONTEXT-BASED STATISTICAL TESTING METHOD

## 5.1 Introduction

Software is usually tested to detect and remove bugs, and also to demonstrate that the software can perform its intended functions, possibly for licensing purposes. Different test strategies have been developed, such as black-box and white-box testing. However, they are not designed for quantifying software reliability (i.e., the probability of failure on demand), and thus cannot be used for that purpose [May 1995, Hamlet 1994, and 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. Testing performed to quantify 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[19], Zhang 2004, and 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 on 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 developed by May [1995] would generate a higher (i.e., unnecessarily conservative) estimated mean failure probability than would the black-box method [Chu 2010]. As indicated by Zhang [2004], a white-box method that attempts to encompass all possible paths may be impractical. The method requires that "instruments" (modifications to the software) be added to the software being tested, making the test less realistic. In addition, white-box testing may have to choose some inputs in order to cover some paths, thus violating the use of the operational profile in sampling and invalidating the reliability result. For these reasons, only black-box testing is considered in this study.

In a probabilistic risk assessment (PRA) of a nuclear power plant, different accident scenarios represent different challenges to the safety systems actuated by digital protection systems, such as the reactor protection system (RPS) and the engineered safety features actuation system (ESFAS). Therefore, each challenge represents a unique condition/context for the protection system as part of the RPS's 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, that is, the accident sequences and cutsets of a PRA specify the conditions of the plant in which a protection's function is needed and should be tested. Statistical testing can be combined with the BBN method discussed in Section 4, which considers earlier phases of software development and

[19] The method, strictly speaking, is a "white-box-flavored" black-box method, because it only partitions the input space and the partitions map to different internal structures of the software. In this report, it is described as a white-box method.

estimates a generic prior distribution for the probability of failure on demand. The context-based idea is similar to that used in a National Aeronautics and Space Administration (NASA) study [ASCA 2007], except that the context-based method makes use of the contexts defined in existing PRA models and uses a thermal-hydraulic model of the plant in simulating the contexts. Appendix A describes different types of tests for licensing purposes, and the configurations used in testing some NPP systems. Statistical testing would ideally be employed for a system once it is 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 5.2 describes the strategy for undertaking the black-box testing, including configuring the test, defining the operational profiles of the accident scenarios, generating test cases by sampling from the profile, and determining the correctness of the test results. In general, the operational profile of the software should include everything that can affect the inputs to the software, including the physical processes with which the system interacts and any equipment failures, as defined in PRA scenarios, that indirectly affect the physical processes, or even digital system hardware failures that directly or indirectly affect the input signals to the software. Section 5.2 provides the mathematical expressions for calculating the system's failure probabilities on demand and discusses assumptions and issues associated with such testing, including treatment of uncertainty. The statistical testing involves testing the software over many possible inputs by sampling from the operational profiles of the PRA accident scenarios. Since the number of test cases can be very large with the consequent costs in resources, that is, time, money, and effort, this poses one of the most difficult problems in assessing the failure probability of a software program. Section 5.3 proposes a risk-informed application of statistical testing to demonstrate that the overall risk from software failure is below an established level, and an approach for determining the feasibility of the application. It first provides some simplified examples to illustrate the concept, then describes the application and how to estimate the total number of test cases needed. Section 5.4 discusses the benefits of the proposed statistical testing approach in terms of the desirable characteristics of Section 2.

## 5.2 A Context-Based Statistical Testing Method

### 5.2.1 PRA Contexts

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 [Garrett 1999] adopted the concept of an "error-forcing context" (EFC) for human reliability analysis [Cooper 1996] for denoting the context in which a software failure may happen. Therefore, a software failure happens due to the occurrence of an EFC that is unanticipated and is difficult to predict; and the impact on the NPP can range from minor to severe.

Given that the PRA context influences the likelihood that a software fault will manifest into a software failure, it is important to account for this context when modeling software failures in a NPP PRA. For example, in a typical PRA, the RPS (with software embedded) usually is the first heading (also called a "top event") in the event trees after an initiating event. Therefore, in this case, such an initiating event (IE) defines the PRA context for this system, that is, different initiating events represent dissimilar plant conditions that may generate distinct input signals to the RPS.

Table 5-1 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 examples of the types of IE generating the condition necessitating reactor trip. Each plant condition requiring a reactor trip, such as each row in Table 5-1, represents the PRA context under which the RPS will be operating.

In addition, failure of the software after an IE may be due, at least in part, to inputs to the software before the IE in the timeline; for example, the condition of the plant before the IE may fluctuate and produce varying inputs to the software. Hence, the PRA context may not only encompass its latest inputs, but may also contain inputs received before the IE, or sometimes represented by software internal states. 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 5.2.4.

**Table 5-1 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 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 that generates an actuation signal of a safety system (e.g., an ESFAS), the PRA context 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.

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) are initiated by the ESFAS, is used to define the PRA context, at a high level, for the actuation of the ESFAS. In general, different sequences in the same or different event trees lead to different PRA contexts for the ESFAS software. If the software fails an earlier function in a sequence, it would be conservative to assume it would fail all later functions in the sequence. If it is successful in performing an early function in a sequence, then the testing would still be useful in assessing later functions.

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 (hardware failures, human errors, and software failures) that occur before the ESFAS-actuated systems in a minimal cutset define a more detailed PRA 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 the complete cutset also are part of the context. Similarly, cutsets are discussed only in terms of failure events, with success events implied.

The resolution (level of detail) of a PRA should be considered when choosing a modeling and quantification approach for software failures because the finer the resolution, the more detailed 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 further illustrated with the following example. It is a common practice in PRAs to gather similar IEs into a single group, thus 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). 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.

Note that while PRA model-defined contexts may appear to be coarse, there are associated/implied details (e.g., timing) that should be taken into consideration when testing software, as long as the details could affect the inputs to the software. The details should be captured in defining/characterizing the operational profile used in generating the test cases. Section 5.2.2.2 describes how the operational profile can be defined, and test cases generated.

The relationship between a PRA's level of resolution and a software program'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. Typically, a PRA's level of detail is at the cutset level. Thus, software testing should be organized according to the level of detail within the PRA context defined for each case. This is especially important for ESFAS functions, which typically are preceded by other failures in the PRA model. On the other hand, considering each cutset separately would be resource-intensive, and some of the cutsets may impose similar, if not identical, contexts, and can probably be treated together. It may be possible to perform tests at the event tree sequence level by grouping the cutsets representing the sequence. This consideration will be further elaborated in the case study by applying the statistical testing method to the example system.

### 5.2.2 Performing Statistical Tests

Statistical testing is used to assess the failure probability of the software of a protection system in the different scenarios. The test cases need to capture the PRA context corresponding to the scenarios. As discussed previously, a PRA context is the accident environment defined by a PRA scenario/sequence/cutset. An operational profile for each of the scenarios, which is conditioned by the PRA context, must be defined and used in creating test cases. The inputs representing the test cases are fed to the protection system, and the system's outputs are evaluated and the test cases' success determined. Since the test cases are generated for conditions in which a protection system is demanded, whether or not an actuation signal is generated in a timely manner determines whether the test result is successful.

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 whether the test is successful. As an example, a medium LOCA in which the RPS is needed is used.

#### 5.2.2.1 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 on 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 might change the triggering event of a software fault and the software's responses. Others even argue for using exactly the same compiler and running environment for the software in the tests because different compilations and execution environments may affect the software's behavior. Employing the original hardware may pose extra difficulties and costs in testing and limited accessibility to the internal states of the software and hardware is likely because the original hardware may not be designed to accommodate this purpose. With advent of JTAGs[20] and in-circuit emulator, this difficulty may be alleviated. Still, generating and feeding test cases automatically need to be further developed for our purpose. In general, it is desirable that a test configuration is able to simulate possible internal states of the digital system and use probabilistic models to account for the occurrence of the internal states, for example, using the probability distributions of the internal hardware failure modes to capture the likelihood of the occurrence of the failure modes. 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 2010b], 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

[20] JTAG was an industry group (Joint Test Action Group) formed in 1985 to develop a method to test populated circuit boards after manufacture. At the time, multi-layer boards and non-lead-frame integrated circuits (ICs) were becoming standard, and connections were being made between ICs which were not available to probes. The majority of manufacturing and field faults in circuit boards were due to solder joints on the boards, imperfections in board connections, or the bonds and bond wires from IC pads to pin lead frames. JTAG was meant to provide a pins-out view from one IC pad to another so that all these faults could be discovered. Today JTAG is also widely used for IC debug ports. In the embedded processor market, essentially all modern processors support JTAG when they have enough pins. Embedded systems development relies on debuggers talking to chips with JTAG to perform operations like single stepping and break pointing. Electronic products, such as cell phones or wireless access points, generally have no other debug or test interfaces.

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 instrumentation and control 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 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 [AREVA 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 central processing unit (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.

#### 5.2.2.2 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 or PRA contexts detailed in Section 5.2.1 using the test configuration described in Section 5.2.2.1.

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. By defining an operational profile for an accident scenario, all possible inputs are theoretically captured in terms of the probability distributions that characterize the operational profile. Since the test cases would be sampled from the operational profile, whether or not certain inputs would be sampled or the number of samples of certain inputs is determined by the probability distributions.

For each PRA context (accident scenario/sequence), such as a medium LOCA requiring a reactor trip, a thermal-hydraulic simulation 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, and should be able to calculate the physical parameters at the specific locations of the sensors that generate inputs to the software. Each test case is generated by sampling from the probability distributions developed for the operational profile of the scenario, and is given in the form of a trajectory on inputs.

Development of the operational profile of an accident scenario is a very important part of the statistical testing approach. The operational profile represents the actual operating condition and determines the possible trajectories of inputs. An approach for defining the operational profile of an accident scenario is described below. It essentially tries to identify all possible causes of variations of software inputs and represent them in terms of probability distributions.

The steady-state 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, as discussed in Section 5.2.1. 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. For the inputs from redundant sensors of the same type, 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 a random sampling of the test cases. They are used in defining the operational profile of the accident scenario.

In general, the inputs to a piece of software are determined by the plant's physical processes, and any failures or events that can affect the physical processes should be taken into consideration in defining the operational profile of a scenario. For example, for an accident scenario requiring the activation of a safety injection system, some other failures may have taken place, leading to the need for safety injection. These failures may be represented by cutsets of the PRA model, and each may impart a different impact on the physical processes and their timing. The impacts have to be accounted for using the thermal-hydraulic model. Potentially, a large number of the cutsets need to be considered. They will have to be grouped in terms of their impacts on the physical processes, and test cases have to be sampled based on the relative frequencies of the groups of cutsets.

It may happen that failures or events not explicitly modeled in a PRA can affect the physical processes.  Sensor inaccuracy and variation in initial plant conditions are only examples of them.  In addition, the occurrence of non-minimal cutsets may affect the physical condition of the plant, and generates inputs that would otherwise not be included.  The identification and inclusion of these events in defining the operational profile of an accident scenario are a completeness issue within the approach.  In general, each such event can be modeled by its probability of occurrence, and, when generating a test case, a probabilistic sampling can be done to determine whether any of the events will occur.  A practical limitation is the ability of a thermal-hydraulic model of the plant to realistically capture everything that can affect the inputs to the software.  For example, a reactor trip is needed upon loss of feedwater, and there may be many different ways loss of feedwater can occur (e.g., different failures of pumps and valves).  Realistic modeling of the different ways loss of feedwater may occur imposes requirements on the level of detail of modeling of the thermal-hydraulic model.

A problem with 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 5.2.4 discusses 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 5.3 proposes an approach to estimating 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 whether 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 for reactor trip is determined by the plant’s thermal-hydraulic model, which generates the test inputs, and this can be compared to the latest time by which 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.  Since the oracle is determined by the scenarios defined in a PRA, not from the requirement specifications of the software, the statistical testing may serve as an independent check of the requirement specifications, and has the potential to identify weaknesses of requirement specifications if failures are observed during the tests.

### 5.2.3 Bayesian Reliability Assessment

As mentioned previously, it is proposed for the case study that the statistical testing results be used to perform a Bayesian update of a prior distribution for the probability of software failure-on-demand.  The prior distribution may be derived from either a detailed or a simplified BBN approach, as described in Section 4, based on factors such as the quality in carrying out software life cycle activities, or it may be non-informative, for example, following existing guidance in the handbook of parameter estimation [Atwood 2002].

The Bayesian approach is a straightforward application of Bayes’ theorem.  Miller et al. [Miller 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:

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

where 0 ≤ *θ* ≤ 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. [Haapanen 2000] and Miller et al. [Miller 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 in carrying out a software's life cycle 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(θ)* is the prior pdf of *Θ*, and *g(x|θ)* 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(θ|x)*. According to Bayes' theorem, the posterior pdf of *Θ*, given the observed value *x*, is:

$$f(\theta \mid x) = \frac{g(x \mid \theta) f(\theta)}{\int_0^1 g(x \mid \theta) f(\theta) d\theta} \tag{5-2}$$

Accordingly:

$$f(\theta \mid x) = \frac{\theta^{x+a-1}(1-\theta)^{n-x+b-1}}{B(x+a, n-x+b)} \tag{5-3}$$

where *x* = 0, 1, …, *n*, and 0 ≤ *θ* ≤ 1.

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

$$\frac{(a+x)}{(a+b+n)} \tag{5-4}$$

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 posterior distribution of Θ must be obtained for each PRA context defined for the operation of a software program, as discussed in Subsection 5.2.1.

The Bayesian approach also can generate an upper bound of the software failure probability $\theta$, $\theta_u$. To do so, a confidence level $\gamma$ is specified that implicitly defines the upper bound of $\theta_u$, such that:

$$Pr\{\Theta \leq \theta_u \mid x\} = \gamma. \tag{5-5}$$

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

An interesting application of this upper bound 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 Jeffreys prior distribution.) In addition, by making $x = 0$ (i.e., assuming there is no observed failure), as often is the case for testing safety-critical software, the posterior cumulative distribution function is expressed as:

$$F(\theta_u \mid x) = \Pr\{\Theta \leq \theta_u \mid x\} = \int_0^{\theta_u} f(\theta \mid x) d\theta \tag{5-6}$$

which reduces to:

$$F(\theta_u \mid 0) = 1 - (1-\theta_u)^{n+1} = \gamma \tag{5-7}$$

Solving this equation for $\theta_u$:

$$\theta_u = 1 - (1-\gamma)^{1/(n+1)} \tag{5-8}$$

The number of successful tests required to show that the failure probability is bounded by $\theta_u$ at confidence level $\gamma$ is obtained from Equation 5-8 as:

$$n = \frac{\ln(1-\gamma)}{\ln(1-\theta_u)} - 1 \tag{5-9}$$

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 [Littlewood 1997b] contend that this approach may be too optimistic because it ignores the failure of the earlier version that is evidence of poor quality. They feel that it would be more conservative to assume that removing the fault does not improve 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 the assumption that the results of the earlier version are ignored.

### 5.2.4 Inclusion of Software Failures in a PRA Model

This section describes how software failures can be included in a NPP PRA. It is assumed that a PRA model is available, with the failures of the protection functions performed by the software (and the digital system) under study included as basic events (designated as SF below) to be quantified using statistical testing. In addition, the PRA model has been solved, such that the core damage cutsets are available.

A cutset containing a failure of software can be expressed as:

$$SC \bullet SF \bullet OF$$

where the symbol "•" means Boolean "AND," and SC is the PRA software context for the operation of the software, determined by the failure(s) in the cutset that occur before the software failure in the accident progression. For statistical testing purposes, the SC defines all software inputs and is characterized as an operational profile, which has been discussed in Section 5.2.2.2. An SC can include one or more failure events; for example, it may be just the IE, or it can include additional failure events.

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

OF is a placeholder for other failure events necessary for the cutset to incur 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 failure events 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 is quantified 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.

Figure 5-1 utilizes a simplified small LOCA example to demonstrate the SC, SF, and OF concepts.

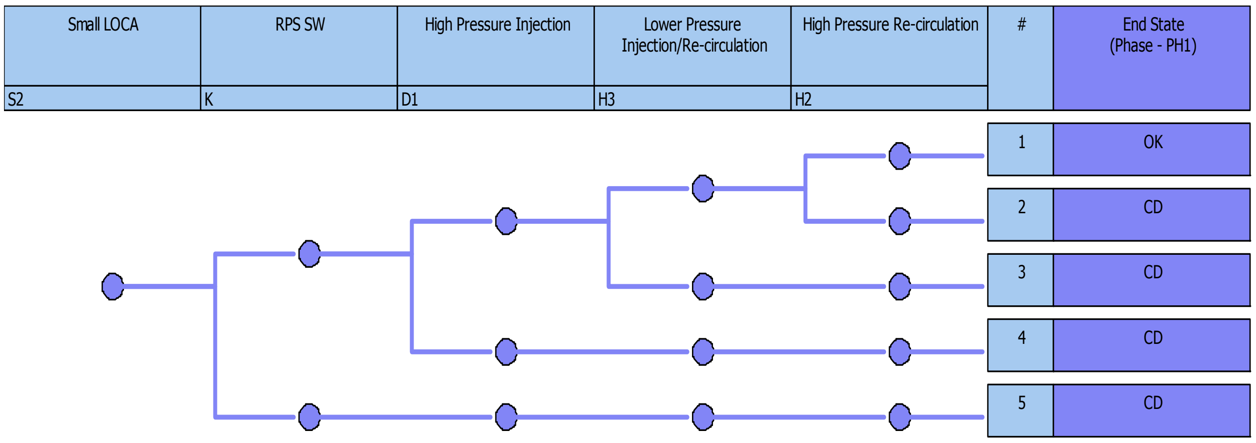


**Figure 5-1 Integrating software into PRA: a small LOCA example**

In this example, the IE is the Small LOCA event S2. The top events in order of being exercised are RPS software (event K), High-Pressure Injection (event D1), Lower-Pressure Injection/Re-circulation (event H3), and High-Pressure Re-circulation (event H2). The end states are OK and CD (core damage). We further assume that the frequency of S2 is known, and that the probabilities of D1, H3, and H2 are estimated using the fault tree method. The estimate of the probability of RPS software failure given the small LOCA is of interest to this study.

The PRA context SC of the RPS software is then identified as the IE S2. It is worth noting, as discussed in Section 5.2.1, that such a PRA context includes additional plant conditions that serve as inputs to the RPS software given the event S2. In other words, although the PRA context is represented by S2, it includes other parameters, such as the power of the reactor, the temperatures of the reactor/hot leg/cold leg, and all other inputs to this software. In summary, the PRA context in this case is a snapshot of all plant parameters relevant to the RPS software, including both nominal and off-nominal conditions, given the small LOCA event. Furthermore, the PRA context can be seen as a conditional operational profile, given the small LOCA. This conditional operational profile can be generated from the thermal-hydraulic simulation code by setting its parameters to reflect the small LOCA situations. Section 5.2.2.2 provides more discussions on the conditional operational profile.

The concept of SF is now the failure event of the RPS software given the small LOCA situation, which is event K in the Figure 5-1 event tree.

**Table 5-2 Sequences and cutsets for the example event tree**

| Sequence # | Cutset | End State |
|---|---|---|
| **2** | S2 H2 | CD |
| **3** | S2 H3 | CD |
| **4** | S2 D1 | CD |
| **5** | S2 K | CD |

Table 5-2 lists all core damage sequences and the corresponding cutsets. It is clear that only sequence #5 (highlighted in Table 5-2) contains software failure contribution. In this simplified example the OF is empty, as the sequence containing the RPS software failure assumes core damage without considering further the anticipated transient without scram (ATWS). In reality, the #5 sequence would transfer to an ATWS event tree where mitigation of the ATWS is modeled and additional failures are required to result in core damage. The additional failures constitute the events in OF.

The contribution of the i^th cutset that includes software failure events to core damage frequency (CDF) is expressed as:

$$\mathrm{CDF(SW)}_i = [\mathrm{F(SC)} * \mathrm{P(SF \mid SC)} * \mathrm{P(OF \mid SC\ and\ SF)}]_i$$

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

The total CDF can be calculated using the rare-event approximation as:

$$\text{CDF} = \sum_i \text{CDF(SW)}_i + \text{CDF(nonSW)}$$

where CDF(nonSW) is the contributions from cutsets that do not include software failures.

As discussed in Section 5.2.1, when performing statistical testing, it may be possible to combine those cutsets with the same impacts on the plant and thus the same inputs to the software, to reduce the number of tests that need to be performed.

### 5.2.5 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 or 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 by 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 conservative.

As discussed in Section 5.2.2.1, 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, BNL proposes 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 in the form of trajectories in the space of the physical parameters. 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., Mandelbugs) [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. One principle of the safety-critical software is to keep the design as simple as possible. For instance, the Teleperm TXS design [NRC 2000] does not allow the "dynamic memory allocation." This simple-design philosophy minimizes such Mandelbugs.

A specific issue associated with the internal states of digital systems is that 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 and without consideration of the internal states may never identify the error. This can be considered an issue in the "memory" of the system, that is, how system states depend on the software past behaviors. The case study addresses this "long memory" issue by extending the testing duration to its practical maximum range. The "practical maximum range" will be defined based on engineering judgment, and will be justified in the future case study.

It is desirable that deterministic analysis be performed to determine that the platform software and operating system do not have a long memory. For example, if a platform's software does not use calendar time, then the Y2K issue does not need to be taken into consideration in generating test cases. In the absence of this type of analysis, the issue of input trajectory length represents a potential limitation in the statistical testing approach.

Section 5.2.2.2, in defining the operational profile, discusses the need to consider all possible events that may directly and indirectly affect the inputs to a software program, including those that are not typically modeled in a PRA (e.g., non-minimal cutsets), and proposes that the events be represented by their occurrence probability and sampling from the probability distributions. An issue is how far one should search for such events. One may argue that some seemingly unrelated events may indirectly affect the inputs. This can be considered a completeness issue. An associated issue is how the effects of the events can be captured, that is, whether or not the effects can be easily included in the thermal-hydraulic model of the plant.

The generation of test cases described in Section 5.2.2 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, such as the detection of a failure of a CPU, implying that internal hardware failures are included in statistical testing. This represents a type of hardware–software interaction within a digital system, and may be considered the internal state 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 way hardware failures are accounted for can be included in the proposed statistical approach by including the hardware failures in the definition of the operational profile, that is, each such hardware failure is represented by a distribution for its probability of failure, and taking samples from this type of distribution to determine whether any of the hardware failures actually take place is a particular test case.

The proposed statistical testing has to take into consideration many different conditions/contexts defined in a PRA, which may make the approach impractical. Section 5.3 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.

### 5.2.6 Treatment of Uncertainties

The Bayesian approach described in Section 5.2.3 automatically accounts for parameter uncertainty. The prior distribution may be derived from a BBN, as described in Section 4, 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 a Jeffreys prior). One key contributor to model uncertainty is the assumption in selecting a prior distribution. This uncertainty can be addressed by performing sensitivity calculations using different prior distributions.

Section 5.2.4 discussed the key issues and limitations in statistical testing, which contribute to model uncertainty. In general, the uncertainties can be treated by performing sensitivity calculations by making alternative assumptions. 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 using the modeled hardware failures to help define the operational profile for testing software.

## 5.3 A Proposed Risk-Informed Application of Context-Based Statistical Testing Approach

This section formulates the integration of testing results into the NPP PRA and estimates the number of test cases required to achieve the given level of reliability. One concern about the method for the context-based statistical testing proposed in Section 5.2 is that a very large number of tests might be needed. To investigate this issue, an application of the method 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 Regulatory Guide 1.174 [RG 1.174] [NRC 2002]) is proposed. The proposed application essentially consists of accounting for the contribution of software failure to the risk of an NPP by estimating the software failure probability for each of the accident scenarios in which the system is needed. A PRA model, such as the PRA of one of the NUREG-1150 plants, could be used in the application.

Section 5.3.1 provides some simplified examples to illustrate some risk-informed concepts associated with software failure's contribution to overall risk or CDF in an NPP. In Section 5.3.2, a proposed approach is described for demonstrating that the contribution of software failure is low enough according to RG 1.174. It also describes how the total number of tests that need to be performed can be estimated to demonstrate the application's feasibility.

### 5.3.1 Simplified Examples to Illustrate Risk-Informed Concepts for Statistical Tests

This subsection presents a few simplified examples to illustrate the concepts associated with a risk-informed application of statistical testing that accounts for the contribution of software failure to the risk or CDF for the entire NPP. The examples are discussed in terms of mean values of probabilities and frequencies, since mean values commonly are used in RG 1.174. Due to different simplifying assumptions used in the examples, the numerical results should not be used in drawing conclusions.

As described in Section 5.2.4, the contribution of the $i^{th}$ cutset, along with software failure, to CDF is expressed as:

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

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

<u>Example 1- An individual cutset</u>

This equation can be used to bound the value of the probability of a software failure to keep the value of the CDF of each cutset equal to, or less than, a certain value. For example, if it is desirable to limit the contribution to CDF of each individual cutset involving software failure to less than or equal to $10^{-7}$/year[21], then for the $i^{th}$ cutset:

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

Solving for $P(SF)_i$:

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

where $P(SF)_i$ is dimensionless because both F(SC) and the initiating event frequency are expressed in units of per year.

The bound for the probability of software failure [P(SF)] appearing in the cutset then is obtained from the last equation, using numerical values for F(SC) and P(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, as discussed in Section 5.2.4. For this example, it can be considered that the context is the occurrence of the LL, so F(SC) = F(LL) = $5.0*10^{-4}$/year (from NUREG-1150 [NRC 1990]). Since

[21] The value presented here is for the purposes of illustration only, and does not imply a desired threshold.

only the occurrence of LL and failure of LPI (due to failure of the hypothetical software) are required to cause core damage, OF does not exist in this cutset. Hence, the last equation becomes:

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

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 (Equation 5-4, with *a* and *b* set to 1). 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 [$P(SF)_i$].

<u>Example 2 – An example cutset/sequence from a PRA</u>

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 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), that is, the fraction of the operating cycle where MTC is not negative enough to prevent potentially excessive reactor coolant system over-pressure during an ATWS. 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 • SF • R • 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:

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

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):

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

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 (Equation 5-4, with *a* and *b* set to 1).

<u>Example 3 – A conceptual risk-informed application</u>

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, this contribution can be assessed as:

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

Then the probabilities of software failure P(SF) for 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, although in practice the context may be the same for several cutsets. For example, the PRA context for some cutsets of the same sequence may be represented by the IE of the sequence, that is, the SC is the IE, with different OFs for different cutsets.

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 be 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 result in a small change in the NPP's CDF. Section 5.3.2 describes a way of using the idea of this inequality in an actual application and how to determine the feasibility of the application by estimating the number of tests that need to be performed.

Simplifying this inequality further to explain 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 quantitative criterion defined in this example assumed in this example, thereby requiring only 44 tests (Equation 5-4, with *a* and *b* set to 1). In practice, a sequence may have many cutsets containing software failures (as is the case if the initiating event is a small LOCA), so the cumulative frequency from all of them must meet the quantitative criterion; this implies that some of the software failure probabilities of the cutsets 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. Section 5.3.2 considers a realistic application and proposes to perform a feasibility study to estimate the total number of tests that need to be performed by allocating the contribution of software failure to all of the sequences and cutsets with software failures.

### 5.3.2 Demonstration of the Feasibility of an Application of Context-Based Statistical Testing

One concern about the strategy for the risk-informed statistical testing method proposed in Section 5.2 is that a very large number of tests might be needed. To investigate this issue, a realistic application that considers a plant's replacement of its analog reactor protection system with a digital one is considered, using the idea in Section 5.3.1. A PRA model, such as 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 an 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). It is assumed that the PRA includes a model of the analog protection system, and that the hardware failure probability of the digital system is the same as that of the analog one. The following is a possible approach for determining the total number of test cases that need to be performed without failure.

1. Based on the total CDF of a plant's 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 the 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 that the generally acceptable increase in total CDF is due to these sequences and that 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 5.2.4, 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.

## 5.4 Evaluation of the Statistical Testing Approach

Test-based methods were initially evaluated in Section 2 against the 10 desirable characteristics for a quantitative software reliability method (QSRM). An updated evaluation of the statistical testing approach proposed in the preceding subsections is provided below. The proposed approach improves statistical testing methods by considering PRA-specified contexts, thus meeting Characteristic 3. A successful application of the proposed approach would serve as a validation of the approach (Characteristic 7) and possibly demonstrate the high reliability of a safety-related software (Characteristic 8). In addition, the approach, in general, can be extended to consider failures of diverse software, giving it a "Maybe" ranking under Characteristic 9.

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. A "Yes" is assigned.

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 demonstrated as easily as otherwise. Therefore, a "Maybe" is assigned.

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. Since the approach has not yet been determined to be practical, a "Maybe" is assigned. A successful application of the method would change the rating to a "Yes."

4. *The method takes into consideration the quality of life cycle activities*.

   The test-based methods do not account for the quality in carrying out life cycle activities. A "No" is assigned.

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. A "Yes" is assigned.

6. *The method addresses uncertainty*.

   The black-box test-based method uses standard statistical methods, including treating parameter uncertainty, to quantify software reliability. A "Yes" is assigned.

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. Therefore, a "Maybe" is assigned. The successful application of the proposed statistical testing approach of this study to an example system would help demonstrate the method and change the ranking to "Yes."

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. The successful application of the method would change the rating from “Maybe” to “Yes.” 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. A successful demonstration of the feasibility would effectively demonstrate the usefulness of the method, effectively meeting the objective of the *high reliability* characteristic.

9. *The method should be able to estimate parameters that can be used to account for software common cause failures (CCFs) of diverse protection systems or channels.*

   As discussed in Section 2.1, the current QSRMs do not consider CCF of two diverse systems. It is beyond the scope of this study to consider this kind of CCF. A “No” is assigned. However, in general, the proposed statistical testing approach can be extended to test two diverse systems, changing the rating from “No” to “Maybe.”

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. A “Yes” is assigned.

# 6 CONCLUSIONS AND INSIGHTS

In this study, the quantitative software reliability methods (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 Bayesian Belief Network (BBN) method and the statistical testing method. Both 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, given the limitations of the statistical testing method, and to account for the quality in carrying out software life cycle activities, it was decided to first develop a prior distribution (using the BBN method), and then undertake a Bayesian update to this distribution, using the results of statistical testing. Therefore, the combination of the two methods may have the benefits of both methods, that is, being able to capture the quality in carrying out the software development activities and to take into consideration the contexts defined by the accident scenarios of a probabilistic risk assessment (PRA).

Since development and quantification of a detailed BBN can be very resource-intensive and the desired evidence for quantification may be lacking, both a simplified and a detailed BBN will be developed. This exercise will provide valuable insight into the relative costs and benefits of developing a detailed model.

Furthermore, 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 Jeffreys prior as the prior distribution, and then employing statistical testing results to perform a Bayesian update of this non-informative 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.

## 6.1 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 in carrying out the software life cycle activities and the latter's use of standard statistical methods, including treatment of parameter uncertainties. In addition, these methods appear to have great potential for demonstrating high software reliability. (It should be noted that the Metrics-Mean Time To Failure 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.

In the evaluation of the QSRMs, some limitations common to all of the reviewed methods were identified. These 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 neither 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 6.3 summarizes how the two candidate methods (BBN and statistical testing) can be enhanced and 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. How the proposed approaches would better address some of the characteristics is also discussed.

## 6.2 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 be such a system. To account for the quality in carrying out software development activities via the BBN method, it is desirable to obtain information about these activities, such as verification and validation reports. In addition, it is desirable that the report produced for the case study be 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 were considered first and then ruled out, mainly due to the expected difficulty of 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 National Aeronautics and Space Administration (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 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.

## 6.3 Development of QSRMs for PRA Use

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

Bayesian Belief Network Method

This study formulated a process for developing a BBN for the case study founded on existing BBNs 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 in carrying out software development activities and statistical test data. However, this advantage is muted by 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 the 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., NRC [2011]) 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 5. The benchmark studies can validate or calibrate a BBN used for quantifying software failure probability. The benchmarking can be undertaken both at the system level and at 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, Neufelder’s correlation method [2002] uses proprietary data on past software projects in a regression analysis to assess the number of faults remaining 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 used by Neufelder is proprietary), including operating experience.

To capture the quality in carrying out software development activities, it is desirable to develop the BBN using the structure of a standard on software development, such as BTP 7-14 [NRC 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 procedure, respectively, and convert the guidance/requirements in the standards into the nodes of their models. Consequently, the BBNs 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, the vendor, and the regulator) should know how the software being evaluated meets the guidance/requirements. A simpler alternative would be to develop a higher-level model, for example, one 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 will be developed.

The use of the BBN results with statistical testing, as suggested in Section 2, makes it possible to account for the contexts specified in a PRA (Characteristic 3). The benchmark studies suggested can potentially be used to verify a BBN model and support any improvements that may be identified (Characteristic 7).

<u>Statistical Testing Method</u>

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, that is, 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 state (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. In general, events that are typically not modeled in a PRA but may affect the plant condition, which in turn affects the inputs to the software program, also need to be considered (e.g., non-minimal cutset). It is proposed that each such event be modeled with the occurrence probability that is used in defining the operational profile. 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; furthermore, 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 that 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 possible way to model internal hardware failures. Including such modeling would make the testing more complete, though it would add 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 to meet the requirement for high reliability of, for instance, a protection system, testing the software in this way entails undertaking an extremely large number of tests, with 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 4.2.2. This study proposes a risk-informed strategy for statistical tests that is expected to resolve this issue. The strategy is similar to that used in a 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 needing to be carried out. Accordingly, applying this strategy will demonstrate whether the software reliability meets a certain goal. Better estimates of software failure probability can be obtained by performing more tests. A procedure is offered in Section 5.3 for determining the total number of tests required to demonstrate the feasibility of risk-informed statistical testing.

The proposed approach improves statistical testing methods by considering PRA-specified contexts, thus meeting Characteristic 3. A successful application of the proposed approach would serve as a validation of the approach (Characteristic 7) and possibly demonstrate the high reliability of a safety-related software program (Characteristic 8). In addition, the approach, in general, can be extended to consider failures of diverse software, giving it a "Maybe" ranking under Characteristic 9.

## 6.4 Key Limitations to Estimating Software Failure Rates and Probabilities for Use in an NPP PRA

This study proposed the approach for developing the BBN and statistical testing models for quantifying the software failure probability of a protection system. Its application to an example system has yet to be performed. The key limitations of the approach are summarized below:

- Modeling of software failures at different levels of detail

  In this study, it is assumed that the software failure of a protection system is modeled at the system level, that is, the software on all microprocessors of the digital system is treated as a single system. In general, it may be necessary to consider software failures at a more detailed level, for example, at the microprocessor level, where hardware-software interactions take place. The limitation is related to how an overall model of the protections system, including both hardware and software, is modeled, for which there is no consensus method.

- Separately modeling different types of software (e.g., application-specific software and operating system software)

  In this study, different types of software, that is, application, platform, and operating system software, are not considered separately. If statistical testing is done with actual software and hardware, then the different types of software are tested together, including their interactions. In the case of the BBN model, only application software is considered. One may argue that platform and operating system software may have been widely used in other applications with a significant amount of operating experience that can be used in quantifying their reliability. Such information is not used in the proposed approach.

- Modeling software context

  The proposed statistical testing method allows PRA contexts to be explicitly accounted for, and requires that the operational profile for each PRA context be developed. How realistic the operational profiles are is a potential limitation of the proposed approach. In particular, how internal states of the protection system can be captured is an unresolved issue. For example, the potential exists that the inputs to the software prior to the need for the protection system may contribute to a software failure, and it is not obvious how far back in time the testing needs to consider. Hardware component failures internal to the protection system also represent an internal state issue that requires much more elaborate test methods to be developed. In addition, thermal-hydraulic models that are used to generate the test cases are a rough approximation of the physical processes of an NPP, and their ability to model all events that may affect the inputs to the software (e.g., a pump failure) may be limited.

  An associated issue is the potentially large number of tests that need to be performed in order to demonstrate the desired reliability. A feasibility study is proposed in Section 5.3 to estimate the total number of tests needed, using a PRA model of an NPP.

- Software CCF

  Many protection systems are designed with identical redundant channels that run the same software. As such, it is assumed that these channels would fail together, due to common software faults. Therefore, this type of common-cause failure (CCF) can be quantified using the methods discussed in this study. This assumption may be too conservative, since some digital protection systems may be designed with two or more diverse channels [Wood 2009]. 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. The potential for CCF between diverse channels of the same system or due to dependencies between two digital protection systems performing similar functions in the same accident scenarios was considered beyond the scope of this study. Similarly, any CCFs that can affect other plant systems modeled in the PRA are beyond the scope of the study. However, due to the potential importance of software CCF to plant risk, this is an area that is recommended for further research.

- Expert Elicitation

  The proposed BBN approach depends heavily on the availability of experts familiar with the software development activities, particularly verification and validation, of the specific software being evaluated, as well as similar software for other protection systems.  There is no cookbook type of guidance on expert opinion elicitation.  An expert opinion elicitation process has to be developed specifically for the proposed BBN approach.

# 7 REFERENCES

[Agena 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., Ohrnbergeret 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.

[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.

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

[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.

[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.

[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.

[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.

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

[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.

[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.

[Korsah 2010] Korsah, K., et al., “An Investigation of Digital Instrumentation and Control System Failure Modes,” Oak Ridge National Laboratory, ORNL/TM-2010/32, March 2010.

[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.

[Littlewood 1980] Littlewood, B., "Theories of Software Reliability: How Good Are They and How Can They Be Improved?", IEEE Transactions on Software Engineering, Vol., SE-6, No. 5, September 1980.

[Littlewood 1997a] Littlewood, B., and Strigini, L., "Guidelines for Statistical Testing," Center for Software Reliability, City University London, PASCON/WO6-CCN2/TN12, October 8, 1997.

[Littlewood 1997b] 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 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 2011] Littlewood, B., and Rushby, J., "Reasoning about the Reliability of Diverse Two-Channel Systems in Which One Channel Is "Possibly Perfect," IEEE Transactions on Software Engineering, August 01, 2011.

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

[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.

[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.

[NEA 2009] Nuclear Energy Agency, "Recommendations On Assessing Digital System Reliability In Probabilistic Risk Assessments Of Nuclear Power Plants," NEA/CSNI/R(2009)18, December 17, 2009.

[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.

[Neil 1996] M. Neil, B. Littlewood, and N. Fenton, "Applying Bayesian Belief Networks to System Dependability Assessment," *Proceedings of 4th Safety Critical Systems Club Symposium*, 1996 (Springer-Verlag).

[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 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 2008] USNRC, "Review of New Reactor Digital Instrumentation and Control Probabilistic Risk Assessment," Interim Staff Guidance, DI&C-ISG-03, August 11, 2008.

[NRC 2009] USNRC, "Wolf Creek Generating Station Amendment to Renewed Facility Operating License," Docket No. 50-482, Amendment No. 181, March 15, 2009.

[NRC 2010a] USNRC, "NRC Digital System Research Plan FY2010-FY2014," February 2010.

[NRC 2010b] 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.

[NRC 2011] USNRC, "International HRA Empirical Study – Phase 2 Report, Results from Comparing HRA Method Predictions to Simulator Data from SGTR Scenarios," NUREG/IA-0216, Vol. 2, August 2011.

[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.

[Regamey 2006] Gret-Regamey, A. and Straub, D., "Spatially Explicit Avalanche Risk Assessment Linking Bayesian Networks to a GIS," *Natural Hazards and Earth System Science*s, 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.

[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.

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

[Tregoining 2008] Tregoning, 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.

[Wood 2012] Wood, R. T., et al., “Classification Approach for Digital I&C Systems at U.S. Nuclear Power Plants,” Oak Ridge National Laboratory, Letter Report LTR/NRC/RES/2012-001, February 2012.

[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 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.

## A.1 Descriptions 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 are tested and modified if needed until the entire integrated system demonstrably complies with the program design, capabilities, and requirements of the system. The test often is 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, that 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.

## A.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 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 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 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 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 in 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 A-1, which is based on the documentation by AREVA [2009], shows the test configuration.

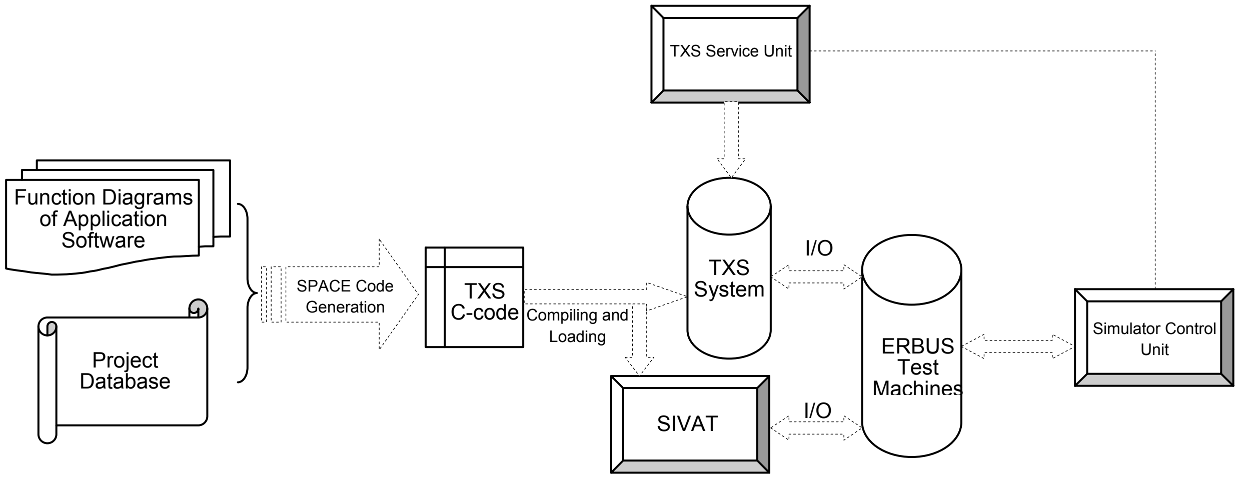


**Figure A-1 ERBUS TELEPERM XS concept**

After the TXS system is manufactured, 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. A Simulation Validation Test (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) running on the computer where SIVAT is being executed, and serves as an oracle for 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, which 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 by 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 the TXS inputs and outputs also can be triggered 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.

## A.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 A-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 module connected to a channel of the digital safety system being tested. 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 on the results by checking them on a cathode ray tube display. This tool was used in system-logic tests and dynamic-transient tests. The transient tests confirm system response against simulated transient data for 665 test scenarios for RPS, and 232 test scenarios for engineered safety features are chosen, covering the design-based transients and the experienced transients in existing plants. The ability to run dynamic tests indicates that the test configuration is suitable for the statistical testing proposed in this report. It is desirable to automate the execution of the test cases and the verification of the accuracy of the results.

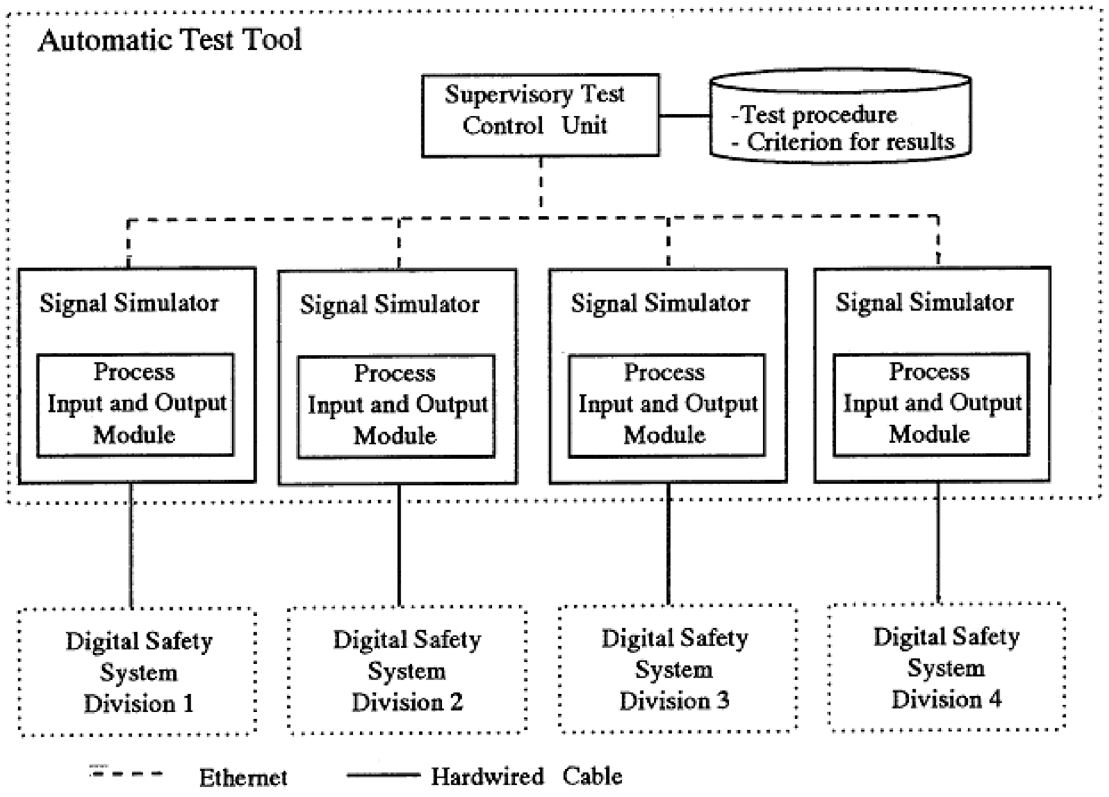

Automatic Test Tool
Supervisory Test Control Unit
-Test procedure
- Criterion for results
Signal Simulator
Process Input and Output Module
Signal Simulator
Process Input and Output Module
Signal Simulator
Process Input and Output Module
Signal Simulator
Process Input and Output Module
Digital Safety System Division 1
Digital Safety System Division 2
Digital Safety System Division 3
Digital Safety System Division 4
Ethernet
Hardwired Cable

**Figure A-2 Test configuration of an ABWR**

## A.4 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. |

| | |
|---|---|
| **NRC FORM 335** (12-2010) NRCMD 3.7 — U.S. NUCLEAR REGULATORY COMMISSION<br>**BIBLIOGRAPHIC DATA SHEET**<br>*(See instructions on the reverse)* | 1. REPORT NUMBER (Assigned by NRC, Add Vol., Supp., Rev., and Addendum Numbers, if any.)<br>NUREG/CR-7044 |
| 2. TITLE AND SUBTITLE<br>Development of Quantitative Software Reliability Models for Digital Protection Systems of Nuclear Power Plants | 3. DATE REPORT PUBLISHED<br>MONTH: October — YEAR: 2013 |
| | 4. FIN OR GRANT NUMBER |
| 5. AUTHOR(S)<br>Tsong-Lun Chu, Meng Yue, Gerardo Martinez-Guridi, and John Lehner | 6. TYPE OF REPORT<br>Final |
| | 7. PERIOD COVERED (Inclusive Dates) |

8. PERFORMING ORGANIZATION - NAME AND ADDRESS (If NRC, provide Division, Office or Region, U. S. Nuclear Regulatory Commission, and mailing address; if contractor, provide name and mailing address.)
Brookhaven National Laboratory
P.O. Box 5000
Upton, NY 11973

9. SPONSORING ORGANIZATION - NAME AND ADDRESS (If NRC, type "Same as above", if contractor, provide NRC Division, Office or Region, U. S. Nuclear Regulatory Commission, and mailing address.)
Division of Risk Analysis
Office of Research
U.S. Nuclear Regulatory Commission
21 Church St, Rockville, MD 20852

10. SUPPLEMENTARY NOTES

11. ABSTRACT (200 words or less)
The U.S. Nuclear Regulatory Commission is currently performing research on the development of probabilistic models for digital instrumentation and control systems for inclusion in nuclear power plant (NPP) probabilistic risk assessments. As part of this research, Brookhaven National Laboratory (BNL) is exploring the inclusion of software failures into digital system reliability models. A previous BNL technical report, entitled “Review of Quantitative Software Reliability Methods,” BNL-94047-2010 (ADAMS Accession No. ML102240566), documented a review of currently available quantitative software reliability methods (QSRMs) that can be used to quantify software failure rates and probabilities of digital systems at NPPs and identified a set of desirable characteristics for QSRMs. In the current report, two candidate QSRMs are selected based on a structured comparison of the previously-identified QSRMs against the set of desirable characteristics. Each selected method is further developed in preparation to be applied in a case study. This report also identifies an example digital protection system for use in the case studies. The actual case studies will be documented in separate reports. Completion of the case studies is expected to provide a much better understanding of the existing capabilities and limitations in treating software failures in digital system reliability models.

| | |
|---|---|
| 12. KEY WORDS/DESCRIPTORS (List words or phrases that will assist researchers in locating the report.)<br>Digital Instrumentation and Controls, PRA, Software Reliability | 13. AVAILABILITY STATEMENT<br>unlimited |
| | 14. SECURITY CLASSIFICATION<br>(This Page) unclassified<br>(This Report) unclassified |
| | 15. NUMBER OF PAGES |
| | 16. PRICE |

Printed on recycled paper

Federal Recycling Program

NUREG/CR-7044

**Development of Quantitative Software Reliability Models for Digital Protection Systems of Nuclear Power Plants**

October 2013