INL/RPT-22-68656

Light Water Reactor Sustainability Program

An Integrated Framework for Risk Assessment of High Safety-significant Safety-related Digital Instrumentation and Control Systems in Nuclear Power Plants: Methodology and Demonstration

August 2022

U.S. Department of Energy

Office of Nuclear Energy

INL/RPT-22-68656

Light Water Reactor Sustainability Program

An Integrated Framework for Risk Assessment of High Safety-significant Safety-related Digital Instrumentation and Control Systems in Nuclear Power Plants: Methodology and Demonstration

Han Bao[1], Tate Shorthill[2], Edward Chen[3], Jooyoung Park[1], Sai Zhang[1], Athira Varma Jayakumar[4], Carl Elks[4], Nam Dinh[3], Heng Ban[2], Hongbin Zhang[5], Edward Quinn[6], Svetlana Lawrence[1]

August 2022

[1]Idaho National Laboratory
Idaho Falls, ID 83415

[2]University of Pittsburgh
Pittsburgh, PA 152601

[3]North Carolina State University
Raleigh, NC 27695

[4]Virginia Commonwealth University
Richmond, VA 23284

[5]Terrapower
Bellevue, WA 98008

[6]Technology Resources
Dana Point, CA 92629

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

EXECUTIVE SUMMARY

This report documents the activities performed by Idaho National Laboratory (INL) during Fiscal Year (FY) 2022 for the U.S. Department of Energy (DOE) Light Water Reactor Sustainability (LWRS) Program, Risk Informed Systems Analysis (RISA) Pathway, digital instrumentation and control (DI&C) risk assessment project. In FY 2019, the RISA Pathway initiated a project to develop a risk assessment strategy for delivering a technical basis to support effective and secure DI&C technologies for digital upgrades/designs. A framework was proposed for this strategy, which aims to (1) provide a best-estimate, risk-informed capability to quantitatively and accurately estimate the risk impact of plant modernization, considering the introduction of high safety-significant safety-related (HSSSR) DI&C systems, (2) support and supplement existing risk-informed DI&C design guides by providing quantitative risk information and evidence, (3) offer a capability of design architecture evaluation of various DI&C systems, (4) assure the long-term safety and reliability of HSSSR DI&C systems, and (5) reduce uncertainty in costs and support integration of DI&C systems in the plant.

To achieve these technical goals, the framework provides a means to address relevant technical issues by: (1) defining a risk-informed analysis process for DI&C upgrade, that integrates hazard analysis, reliability analysis, and consequence analysis, (2) applying risk-informed tools to address common cause failures (CCFs) and quantify corresponding failure probabilities for DI&C technologies, particularly software CCFs, (3) evaluating the impact of digital failures at the component level, system level, and plant level, and (4) providing insights and suggestions on designs to manage the risks, thus to support the development and deployment of advanced DI&C technologies on nuclear power plant (NPPs).

Adding diversity within system or components is the primary means to eliminate and mitigate CCFs, but diversity also increases system complexity and errors and may not address all sources of systematic failures. Optimizing the diversity and redundancy applications for the safety-critical DI&C systems remains a challenge. To deal with the technical issues in addressing potential software CCFs in HSSSR DI&C systems of NPPs and supporting relevant design optimization, the framework provides:

- A best-estimate, risk-informed capability to address new technical digital issues quantitatively, focusing on software CCFs in HSSSR DI&C systems of NPPs
- A common and a modularized platform for DI&C designers, software developers, cybersecurity analysts, and plant engineers to predict and prevent risk in the early design stage of DI&C systems
- Technical bases and risk-informed insights to assist users to address and fulfill the risk-informed alternatives for evaluation of CCFs in HSSSR DI&C systems of NPPs
- A risk-informed tool that offers a capability of design architecture evaluation of various DI&C systems to support system design decisions in diversity and redundancy applications.

The research and development efforts of this project in FY 2022 are focused on methodology improvement and demonstration of the LWRS-developed framework for the risk assessment and design optimization of safety-critical DI&C systems. This framework was further developed with a capability to trace software failures in digital feedback pathways in highly redundant safety-critical DI&C systems; potential failures to a DI&C system are organized in a fault tree for clear visual and linear traceability. Case studies demonstrated the identification of digital failure mechanisms in key instrumentation, construction of the software fault tree in highly complex DI&C systems, and the identification of software single points of failure and key CCFs. Based on the software failure traceability, an innovative approach was also developed to quantify probabilities of various software failure modes including CCFs in a DI&C system. All these capabilities offer a common and modularized platform to DI&C designers, software developers, cybersecurity analysts, and plant engineers for the evaluation of various design architectures of DI&C systems to support system design decisions in diversity and redundancy applications.

The primary audience of this report are DI&C designers, engineers, and probabilistic risk assessment (PRA) practitioners. This includes stakeholders, such as the nuclear utilities and regulators who consider the deployment and upgrade of DI&C systems, DI&C software developers and reviewers, and cybersecurity specialists.

It should be noted that all the analyses are performed for the demonstration of the proposed framework, not for the evaluation of actual systems. Results are obtained based on very limited design information and testing data.

CONTENTS

FIGURES

TABLES

ACRONYMS

ACTS	Automated Combinatorial Testing for Software Tools
ADC	analog to digital
AFP	auxiliary feedwater pump
AFW	auxiliary feedwater
APR-1400	Advanced Power Reactor 1400 MW
ATF	accident tolerant fuel
ATWS	anticipated transient without scram
BAHAMAS	Bayesian and HRA-Aided Method for the Reliability Analysis of Software
BBN	Bayesian Belief Network
BFM	beta-factor model
BP	bistable processor
BTP	branch technical position
BVA	boundary value analysis
CAP	control/actuation pathway
CCCG	common cause component group
CCF	common cause failure
CDF	core damage frequency
CET	core exit thermocouple
CF	coverage factor
CIM	component interface module
CREAM	Cognitive Reliability and Error Analysis Method
CP	communications module
CPCS	Core Protection Calculator System
CPS	computer-based procedure system
CPU	central processing unit
D3	defense-in-depth and diversity
DCP	defect conditional probability
DI&C	digital instrumentation and control
DIS	diverse indication system
DOE	U.S. Department of Energy
DOE–NE	U.S. Department of Energy–Office of Nuclear Energy
DOM	digital output module
DPS	diverse protection system

EEPROM	Electrically erasable programmable read-only memory
EP	equivalence partitioning
EPRI	Electric Power Research Institute
ERP	Enhanced Resilient Plant
ESF	engineered safety feature
ESFAS	Engineered Safety Features Actuation System
ET	event tree
ETA	event tree analysis
FLEX	Flexible Coping Strategy
FMEA	failure mode effect analysis
FPD	front panel display
FT	fault tree
FTA	fault tree analysis
FY	Fiscal Year
GC	group controller
HAZCADS	Hazards and Consequence Analysis for Digital Systems
HEP	Human-Error Probability
HJTC	heated-junction thermocouple
HPI	high-pressure injection
HRA	human reliability analysis
HSSSR	High Safety-significant Safety-related
IAP	integrated action plan
ICC	inadequate core cooling
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IFPD	information flat panel display
IFP	information/feedback pathway
INL	Idaho National Laboratory
IPS	information processing system
LC	loop controller
LCL	local coincidence logic
LDP	large display panel
LOCA	loss-of-coolant accident
LOSC	loss-of-seal cooling
LP	logic processor

LPCI	low-pressure core injection
LPI	low-pressure injection
LWR	light water reactor
LWRS	Light Water Reactor Sustainability
MCDC	modified condition decision coverage
MCR	main control room
MDAFP	motor-driven auxiliary feedwater pumps
ML	machine learning
NAMAC	Near Autonomous Management and Control System
NEI	Nuclear Energy Institute
NIST	National Institute of Standards and Technology
NPP	nuclear power plant
NRC	U.S. Nuclear Regulatory Commission
ODC	orthogonal-defect classification
ORCAS	Orthogonal-defect Classification for Assessing Software reliability
PBF	partial beta factor
PLC	programmable logic controller
PWR	pressurized-water reactor
PRA	probabilistic risk assessment
QIAS-P	qualified indication and alarm system – safety
QIAS-N	qualified indication and alarm system – non-safety
R&D	research and development
RCCA	rod cluster control assembly
RCS	reactor control system
RCSM	reactor coolant saturation margin
RESHA	Redundancy-guided Systems-theoretic Hazard Analysis
RG	Regulatory Guide
RHR	residual heat removal
RISA	Risk-Informed Systems Analysis
RPS	reactor protection system
RRO	Risk Reduction Objective
RSR	reserve shutdown room
RT	requirements traceability
RTB	reactor trip breaker
RTM	requirement traceability matrix

RTOS	Real-Time Operating System
RTS	reactor trip system
RTSS	reactor trip switchgear system
RVL	reactor vessel level
SAR	safety analysis report
SDLC	software development life cycle
SAPHIRE	Systems Analysis Programs for Hands-on Integrated Reliability Evaluations
SBO	station black-out
SDD	software design description
SDN	safety data network
SFP	software failure probability
SIL	safety integrity level
SODP	shutdown overview display
SP	selective processor
SPADES+	safety parameter display and evaluation system+
SPAR-H	Standardized Plant Analysis Risk-Human Reliability Analysis method
SRGM	software reliability growth method
SRM	staff requirements memorandum
SRP	Standard Review Plan
SRS	software requirements specifications
SSC	system, structure, and component
ST	shunt
STD	systems test document
STPA	systems-theoretic process analysis
TC	test coverage
TCA	trigger coverage assessment
THERP	Technique for Human-Error Rate Prediction
UAV	Unmanned Aerial Vehicle
UCA	unsafe control action
UIF	unsafe information flow
UPM	Unified Partial Method
U.S.	United States
UV	undervoltage
VCU	Virginia Commonwealth University

AN INTEGRATED FRAMEWORK FOR RISK ASSESSMENT OF HIGH SAFETY-SIGNIFICANT SAFETY-RELATED DIGITAL INSTRUMENTATION AND CONTROL SYSTEMS IN NUCLEAR POWER PLANTS: METHODOLOGY AND DEMONSTRATION

1. INTRODUCTION

This report documents the activities performed by Idaho National Laboratory (INL) during fiscal year (FY) 2022 for the U.S. Department of Energy (DOE) Light Water Reactor Sustainability (LWRS) Program, Risk Informed Systems Analysis (RISA) Pathway, digital instrumentation and control (DI&C) risk assessment project [1] [2] [3]. The LWRS program, sponsored by the U.S. DOE and coordinated through a variety of mechanisms and interactions with industry, vendors, suppliers, regulatory agencies, and other industry research and development (R&D) organizations, conducts research to develop technologies and other solutions to improve economics and reliability, sustain safety, and extend the operation of nation's fleet of nuclear power plants (NPPs). The LWRS program has two objectives to maintain the long-term operations of the existing fleet: (1) to provide science- and technology-based solutions to industry to implement technology to exceed the performance of the current business model and (2) to manage the aging of systems, structures, and components (SSCs) so NPP lifetimes can be extended, and the plants can continue to operate safely, efficiently, and economically.

As one of the LWRS program's R&D pathways, RISA Pathway aims to support decision-making related to economics, reliability, and safety providing integrated-plant systems-analysis solutions through collaborative demonstrations to enhance economic competitiveness of the operating fleet. The RISA Pathway R&D's purpose is to support plant owner-operator decisions with the aim to improve the economics and reliability and maintain the high levels of current NPPs' safety over periods of extended plant operations. The goal of the RISA Pathway is to conduct R&D to optimize safety margins and minimize uncertainties to achieve economic efficiencies while maintaining high levels of safety. This is accomplished in two ways: (1) by providing scientific basis to better represent safety margins and factors that contribute to cost and safety; and (2) by developing new technologies that reduce operating costs.

One of the research efforts under the RISA Pathway is the DI&C Risk Assessment project, which was initiated in FY 2019 to develop a risk assessment strategy for delivering a strong technical basis to support effective and secure DI&C technologies for digital upgrades/designs [1]. An integrated risk assessment framework for the DI&C systems was proposed for this strategy which aims to:

- Provide a best-estimate, risk-informed capability to quantitatively and accurately estimate the safety margin obtained from plant modernization, especially for the high safety-significant safety-related (HSSSR) DI&C systems
- Support and supplement existing advanced risk-informed DI&C design guides by providing quantitative risk information and evidence
- Offer a capability of design architecture evaluation of various DI&C systems to support system design decisions and diversity and redundancy applications
- Assure the long-term safety and reliability of HSSSR DI&C systems
- Reduce uncertainty in costs and support integration of DI&C systems at NPPs.

The R&D efforts of this project in FY 2022 are focused on methodology improvement and demonstration of the LWRS-developed framework for the risk assessment and design optimization of

safety-critical DI&C systems. This framework was further developed with a capability to trace software failures in digital feedback pathways for highly redundant safety-critical DI&C systems, potential failures to a DI&C system are organized in a fault tree for clear visual and linear traceability. Case studies demonstrated methods for identification of digital failure mechanisms in key instrumentation, constructing the software fault tree in highly complex DI&C systems, software single points of failure, and key CCFs. Based on the software failure traceability, an innovative approach was also developed to quantify probabilities of various software failure modes including CCFs in a DI&C system. These capabilities offer an integrated framework for DI&C designers, software developers, cybersecurity analysts, and plant engineers for the evaluation of various design architectures of DI&C systems to support system design decisions.

The remaining sections of the report are organized as follows: Section 2 provides the technical background of identification and quantification of risks associated with HSSSR DI&C systems; Section 3 describes the methodology and demonstration of a hazard analysis method developed in the framework, called redundancy-guided systems-theoretic method for hazard analysis (RESHA); Section 4 introduces the recent efforts in the improvement of a multiscale quantitative reliability analysis process of the LWRS-developed framework for DI&C risk assessment; Section 5 documents the consequence analysis of a generic pressurized-water reactor (PWR) probabilistic risk assessment (PRA) model with improved digital reactor trip system (RTS), engineered safety features actuation system (ESFAS), and human system interface (HSI) fault trees (FTs); Section 6 discusses the feasibility of the proposed framework in the risk assessment and design optimization of potential artificial intelligence (AI)-aided control systems in future reactor designs and upgrades; Section 7 outlines conclusions and future work of this project.

2. TECHNICAL BACKGROUND

This section describes the technical background of identification and quantification of risks associated with HSSSR DI&C systems. Section 2.1 provides background details for relevant efforts from performed in the last few years. Section 2.2 reviews regulatory positions and guidance, especially NRC's current DI&C CCF policy and future extension plan. Section 2.3 briefly introduces the proposed framework for the risk assessment of the HSSSR DI&C systems. Section 2.4 describes the value propositions of the proposed framework.

2.1 Background

Although the current fleet of the U.S. NPPs was originally designed and constructed with analog systems, the U.S. nuclear industry has been working on transitioning from analog to DI&C technologies. DI&C systems have many advantages over analog systems. They are proven to be more reliable, cheaper, and easier to maintain given obsolescence of analog components. However, they also pose new engineering and technical challenges. The U.S. Nuclear Regulatory Commission (NRC) continues to support the research work in developing and improving licensing criteria for the evaluation of new DI&C systems. In 2018, SECY-18-0090 [4] was published to clarify guidance associated with evaluating potential common cause failures (CCFs) of DI&C systems. The SECY-18-0090 identifies these guiding principles: applicants and licensees for production and utilization facilities should continue to assess and address software CCFs for DI&C systems and components. A defense-in-depth and diversity (D3) analysis for RTSs and engineered safety features should continue to be performed to demonstrate that vulnerabilities to a CCF have been identified and adequately addressed. The D3 analysis can be performed using either a design-basis deterministic approach or best-estimate approach [4]. In 2019, the NRC staff developed the integrated action plan (IAP) [5]. Four detailed modernization plans were proposed to resolve regulatory challenges, provide confidence to licensees, and modernize the I&C regulatory infrastructure. One of them—protection against CCF—addresses "developing guidance for using effective qualitative assessments of the likelihood of failures, along with coping and/or bounding analysis for addressing CCFs, use of defensive design measures for eliminating CCF from further consideration, and staff evaluation of the NRC's existing positions on defense against CCF [5]." The current guidance, however, is unclear regarding the applicability of criteria for using coping analysis and other design features (e.g., defensive measures) for eliminating CCFs from further consideration [5]. Meanwhile, the industry stakeholders are seeking clearer NRC staff guidance on methods for analyzing the potential for CCFs in DI&C systems and a more risk-informed, consequence-based regulatory infrastructure that removes uncertainty in requirements and enables technical consistency [5]. CCF not only leads to the loss of function of safety-critical systems but also the spurious activation of redundant safety-instrumented systems [6].

Many efforts from regulatory, industrial, and academic communities have been made for qualitatively addressing CCFs in DI&C Systems, especially software CCFs, given the increased pace of design and deployment of HSSSR DI&C systems in NPPs. To successfully model DI&C systems, the need exists to model both the hardware and software interactions of the system. Traditional methods, such as failure modes and effects analysis (FMEA) and fault tree analysis (FTA), have been used to extensively model analog systems. However, these traditional methods are not fully suitable to identify failures in interactions between digital systems and controlled processes (i.e., Type 1 interactions) as well as interactions between digital systems and their own components or other systems (i.e., Type 2 interactions) [7]. Lessons learned from the NRC's investigation of multiple analysis methods indicate there "may not be one preferable method for modeling all digital systems" [7]. Combining methods may prove beneficial. A recent advancement in hazards analysis, developed jointly by Electric Power Research Institute (EPRI) and Sandia National Laboratory, combines FTA and the systems-theoretic process analysis (STPA) as a portion of their methodology for Hazard and Consequence Analysis for Digital Systems (HAZCADS) [8]. Though STPA may be applied at any stage of system design and review, it is ideally suited for early

applications in the design process before safety features have been incorporated into the design [9]. Then, as more details are incorporated, the STPA method is applied iteratively to further improve the design. However, even when fine details about a system are known, the analysis may remain at a high level, relying on causal factor investigations to provide details of subcomponent failures and interactions. In other words, details, such as redundant subsystems or components, are often ignored in all but the final part of STPA.

In July 2021, Nuclear Energy Institute (NEI) published NEI 20-07, "Guidance for Addressing Software Common Cause Failure in High Safety-Significant Safety-Related Digital I&C Systems" [10]. A two-step process was proposed to address HSSSR systematic CCFs based on STPA: Step 1 is to perform a systematic hazards analysis based on STPA that creates a model of the system control structure, identifies unsafe control actions (UCAs) as software failures, and establishes a risk reduction objective (RRO); Step 2 is to develop STPA loss scenarios and eliminate or mitigate them in an efficient way. A bounding assessment is proposed to calculate the risk change when entire HSSSR systems fail due to software CCFs (assuming system failure probability = 1). The risk change (e.g., Δ core damage frequency [CDF]) is then mapped to the regions described in Regulatory Guide (RG) 1.174 [11] and used to determine the RRO. This process qualitatively addresses potential failures in DI&C based on a bounding assessment; consequently, the real safety margin gained by plant digitalization on HSSSR DI&C systems could be underestimated in this intentionally conservative approach.

2.2 Review of the Regulatory Positions and Guidance

The NRC's current DI&C CCF policy is expressed in various documents, including SRM-SECY-93-087 [12], SECY-18-0090 [4], and branch technical position (BTP) 7-19 [13]. The NRC documented its four positions with respect to CCF in DI&C systems and D3 as Item 18, II.Q, in SECY-93-087, which was subsequently modified in the associated staff requirements memorandum (SRM) [12]. In accordance with the SRM on SECY-93-087, the NRC published the BTP 7-19 of NUREG- 0800 [13] to provide guidance for a D3 assessment of DI&C systems and confirm the vulnerabilities to CCF. The BTP 7-19 provides various acceptance criteria and requires a D3 analysis to ensure conformance with the regulatory positions on D3 for DI&C systems.

SRM-SECY-93-087 [12] directs that, if the D3 assessment shows a postulated CCF could disable a safety function, then a diverse means could be provided to perform that safety function or a different function. The current policy does not allow for the use of a risk-informed approach to determine specific circumstances that would not require a diverse means for addressing DI&C CCFs. Recently, the NRC is working on expanding the current policy regarding potential CCFs in DI&C systems. The NRC staff are developing a SECY paper that will provide recommended language for an extended policy, which allows greater use of risk-informed approaches to address DI&C CCFs. The expanded policy will encompass the current points of SRM-SECY-93-087 [12] with clarifications and expand the use of risk-informed approaches. Any use of risk-informed approaches will be expected to be consistent with the safety goal policy statement, PRA policy statement, and SRM-SECY-98-0144 [14]. The current DI&C policy will continue to remain a valid option for licensees and applicants.

Figure 1 shows the NRC-proposed expanded policy to address DI&C CCFs. The "Current Path" allows for the use of best-estimate analysis and diverse means to address a potential DI&C CCF, while the "Risk-Informed Path" allows for the use of risk-informed approaches and other design techniques or measures other than diversity to address a potential DI&C CCF. The current policy continues to be a viable option to address DI&C CCFs, and the four points in SRM-SECY-93-87 will remain a viable path to licensees and applicants:

- "The applicant shall assess the defense-in-depth and diversity of the proposed instrumentation and control system to demonstrate that vulnerabilities to common-mode failures have adequately been addressed."
- "In performing the assessment, the vendor or applicant shall analyze each postulated common-mode failure for each event that is evaluated in the accident analysis section of the safety analysis report (SAR) using best-estimate methods. The vendor or applicant shall demonstrate adequate diversity within the design for each of these events."
- "If a postulated common-mode failure could disable a safety function, then a diverse means, with a documented basis that the diverse means is unlikely to be subject to the same common-mode failure, shall be required to perform either the same function or a different function."
- "A set of displays and controls located in the main control room shall be provided for manual, system-level actuation of critical safety functions and monitoring of parameters that support the safety functions."

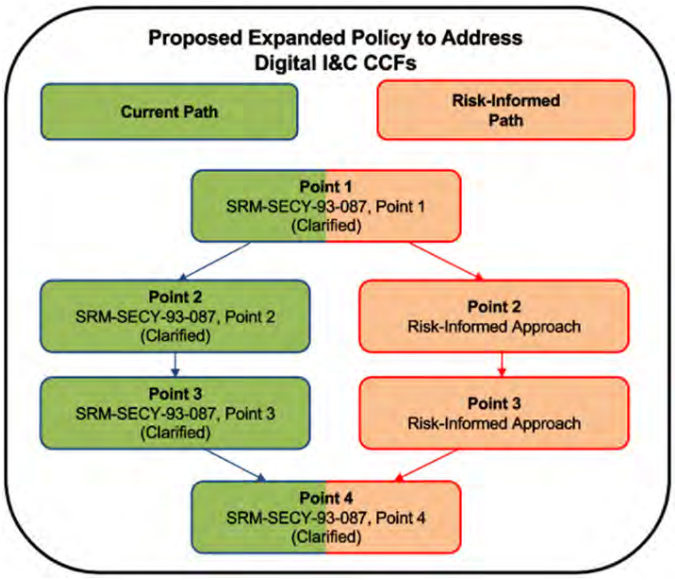

Figure 1. An expanded policy to address DI&C CCFs proposed by the NRC (from Digital I&C Subcommittee Meeting on Outline for Draft SECY Paper to Allow for Consideration of Risk- Informed Alternatives for Addressing DI&C CCF, held on May 20th, 2022).

According to the Digital I&C Subcommittee Meeting on Outline for Draft SECY Paper to Allow for Consideration of Risk- Informed Alternatives for Addressing DI&C CCF:

- "Point 1 doesn't preclude the use of risk-informed approaches for the D3 assessment, existing policy and guidance support a graded approach and applying a level of rigor for the D3 assessment commensurate with the safety significance of the proposed DI&C systems or component."
- "Current approach focuses on consequences and is considered as an appropriate area for risk-informing the evaluation of postulated DI&C CCFs. The staff's goal is that risk-informed approaches will be consistent with all five principles of risk-informed decision-making."
- "Current approach only provides one way of addressing undesirable outcomes (i.e., diverse means), the staff considers this an appropriate area for evaluating design measures other than diversity to

reduce the risk from a DI&C CCF. The staff's goal is to apply a graded approach for the level of justification needed for design techniques or measures other than diversity."

- "Point 4 is consistent with current regulations that effectively require diverse and independent displays and controls, risk-informed approach to Point 4 would not provide appreciable benefits."

The meeting also illustrated the benefits for risk-informed approaches:

- Risk-informed approaches can provide flexibility to address DI&C CCFs and are consistent with the PRA policy statement
- Risk-informed approaches can have different levels of PRA use
- Risk-informed approaches could support a graded approach for addressing DI&C CCFs in high safety-significant systems
- PRA models could be used to systematically assess the need to reduce the risk introduced by the DI&C systems
- Risk-informed approaches can identify initiators or scenarios where lack of DI&C diversity does not compromise safety.

The use of risk-informed approaches will be consistent with all five principles of risk-informed decision-making, as listed in RG 1.174 [11]. PRAs used for risk-informed approaches will be technically acceptable (e.g., meet the guidance in RG1.200 [15]) and include an effective PRA configuration control and feedback mechanism.

Currently, adding diversity within system or components using existing systems, manual operator actions, or new diverse systems is the main means to eliminate and mitigate CCFs. Diversity may be useful in addressing hazards like CCFs, but it also increases plant complexity and errors and may not address all sources of systematic failures. Most systematic failures are a result of either latent design defects due to inadequate requirements or uncontrolled system interactions. How to optimize the diversity and redundancy applications for the HSSSR DI&C systems remains a challenge.

2.3 The Proposed Framework

Previous research efforts provide a technical basis for dealing with potential software CCFs in the HSSSR DI&C systems of NPPs; however, some technical challenges and questions remain:

1. Is qualitative evaluation sufficient for addressing software CCFs in HSSSR DI&C systems? Most of the STPA-based approaches mentioned above focus on the identification of software failures but not the quantification of their probabilities. Although these software failures are added into an integrated fault tree (FT), their probabilities are not calculated. Instead, a conservative bounding assessment is performed to evaluate their impacts to plant safety, which may lead to an underestimation of safety margins gained by plant digitalization and/or skewed risk metrics.
2. How to quantitatively evaluate CCF-related impacts to HSSSR DI&C systems and entire plant response? This proposes a need in developing an integrated strategy to include both qualitative hazard analysis and quantitative reliability and consequence analysis for addressing software CCF issues in HSSSR DI&C systems. Inputs and outputs of each analysis process should be consistently connected.
3. How to efficiently identify the most significant CCFs, especially software CCFs? Existing STPA-based approaches represents good performance in capturing systematic failures in digital interactions; however, there is no clear representation of how to create a control structure for a complicated system containing multiple layers of redundancy and diversity.
4. How to perform a complete reliability analysis for large-scale HSSSR DI&C systems with small-scale software/digital units? Currently, there is no consensus method for the software reliability modeling

of digital systems in NPPs. A reliability analysis approach is needed, especially for the quantification of UCAs from STPA analyses.

5. How to evaluate different system architectures from perspectives of both risk and cost? A DI&C system could be designed using several options for system architecture (e.g., redundancy and diversity at different system levels), and a comprehensive, consistent, integrated approach is needed to support evaluation of various design architectures to ensure the most optimal one is selected for implementation. This approach should be able to support vendors and utilities with optimization of design solutions from economical perspectives given the constrain of meeting risk-informed safety requirements.
6. Lastly, a need clearly exists to develop a risk assessment strategy to support quantitative D3 analyses for assuring the long-term safety and reliability of vital digital systems and reducing uncertainties in costs, time, and supporting integration of digital systems during the design stage of the plant.

To address the above-mentioned challenges, an integrated risk assessment strategy is needed to include both qualitative hazard analysis and quantitative reliability and consequence analysis for addressing software CCF in the HSSSR DI&C systems. To fulfill this need and deal with the technical barriers in identifying potential software CCF issues in HSSSR DI&C systems of NPPs, the LWRS-RISA Pathway initiated this project to develop a risk assessment strategy [1] to:

- Provide a best-estimate, risk-informed capability to quantitatively and accurately estimate the NPP safety margin gained from the modernization of HSSSR DI&C systems
- Support and supplement existing advanced risk-informed DI&C design guides by providing quantitative risk information and evidence
- Offer a capability of design architecture evaluation of various DI&C systems to support system design decisions in diversity and redundancy applications
- Assure the long-term safety and reliability of HSSSR DI&C systems
- Reduce uncertainty in costs and support integration of DI&C systems at NPPs.

To achieve these technical, the proposed framework provides a means to address relevant technical issues by:

- Defining a risk-informed analysis process for a DI&C upgrade that integrates hazard analysis, reliability analysis, and consequence analysis
- Applying systematic and risk-informed tools to address CCFs and quantify corresponding failure probabilities for DI&C technologies, particularly software CCFs
- Evaluating the impact of digital failures at the component level, system level, and plant level
- Providing insights and suggestions on designs to manage the risks, thus, to support the development and deployment of advanced DI&C technologies on NPPs.

Figure 2 displays the schematic of the proposed risk assessment framework for HSSSR DI&C systems. The proposed framework provides a systematic, verifiable, and reproducible approach based on technically sound methodologies. The framework successively implements qualitative hazard analysis, quantitative reliability analysis, and consequence analysis to obtain quantitative risk metrics. The quantified risks are then compared with respective acceptance criteria which allows the identification of vulnerabilities as well as provides suggestions for risk reduction and design optimization.

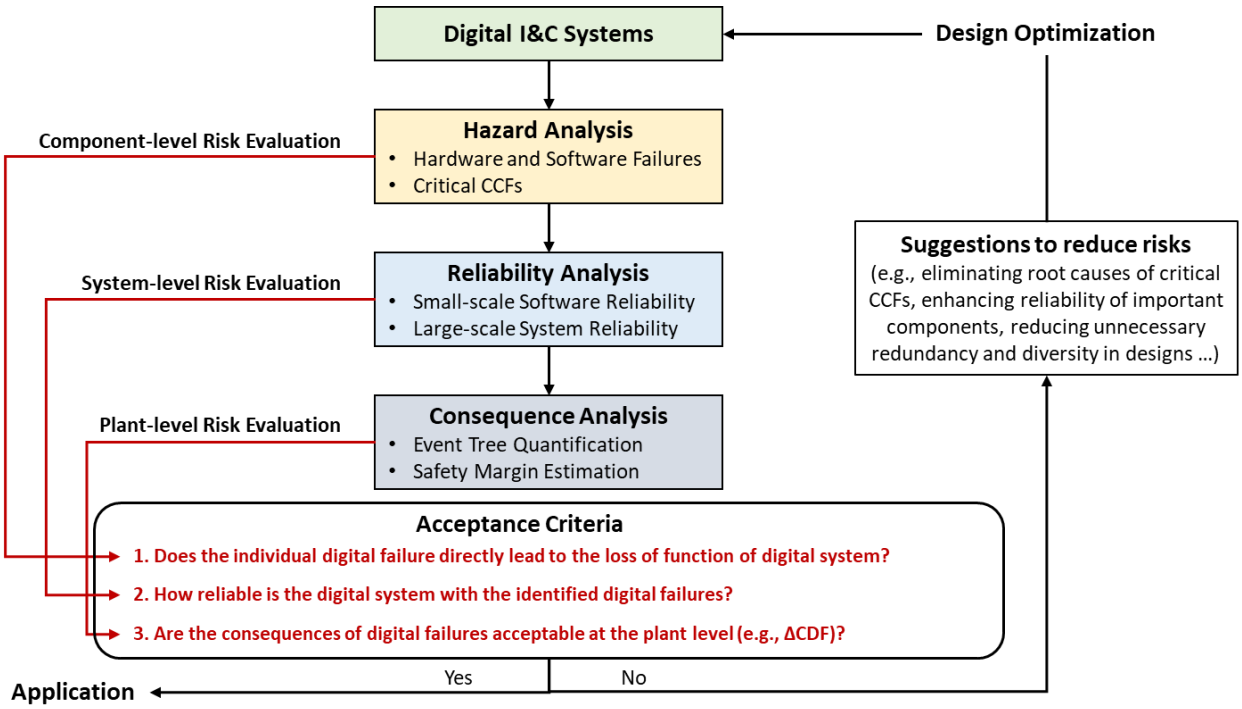

Figure 2. Schematic of the proposed risk assessment framework for HSSSR DI&C systems.

2.4 Value Proposition

To deal with the technical issues in addressing potential software CCF issues in HSSSR DI&C systems of NPPs, this proposed framework is expected to provide:

1. A best-estimate, risk-informed capability to address new technical digital issues quantitatively, focusing on software CCFs in HSSSR DI&C systems of NPPs.

Existing qualitative approach for addressing CCFs in HSSSR DI&C systems may significantly underestimate the real safety margin introduced by plant digitalization. The framework is developed and demonstrated in an integrated way including both qualitative hazard analysis and quantitative reliability and consequence analyses. The framework aims to provide a best-estimate, risk-informed capability to accurately estimate the safety margin increase obtained from plant modernization, especially for the HSSSR DI&C systems.

In this framework, a redundancy-guided systems-theoretic method for hazard analysis (RESHA) was developed for HSSSR DI&C systems to support I&C designers and engineers to address both hardware and software CCFs and qualitatively analyze their effects on system availability [16] [17]. It also provides a technical basis for implementing reliability and consequence analyses of unexpected software failures, and supporting the optimization of D3 applications in a cost-efficient way. Targeting the complexity of redundant designs in HSSSR DI&C systems integrates STPA [9], FTA, and HAZCADS [8] to effectively identify software CCFs by reframing STPA in a redundancy-guided way, such as (1) depicting a redundant and diverse system via a detailed representation; (2) refining different redundancy levels based on the structure of DI&C systems; (3) constructing a redundancy-guided multilayer control structure; and (4) identifying potential CCFs in different redundancy levels. This approach has been demonstrated and applied for the hazard analysis of a four-division digital RTS [16] and a four-division, digital, ESFAS [17]. These efforts have been included in the LWRS-RISA milestone report for FY-20 [2] and FY-21 [3].

The second part in risk analysis is reliability analysis with the tasks of (1) quantifying the probabilities of basic events of the integrated FT from the hazard analysis; (2) estimating the probabilities

of the consequences of digital system failures. In the proposed framework, two methods have been developed for different application conditions: the Bayesian and human-reliability-analysis-aided method for the reliability analysis of software (BAHAMAS) [18] for limited-data conditions and orthogonal-defect classification for assessing software reliability (ORCAS) for data-rich analysis. Finally, consequence analysis is conducted to quantitatively evaluate the consequence impact of digital failures on plant behaviors and responses. Some digital-based failures may initiate an event or scenario that may not be analyzed before, which brings in a big challenge to plant safety.

In February 2022, the NRC organized a public meeting to inform the industry and solicit external stakeholders' feedback on the NRC's plan to potentially expand the current policy for addressing CCFs for DI&C systems to allow consideration of risk-informed alternatives. The LWRS Program's RISA team presented on providing capabilities to address and fulfill the risk-informed alternatives for the evaluation of CCFs in DI&C systems. The NRC staff found the framework interesting from the regulatory point of view since it may be useful to evaluate the impacts of various DI&C design architectures to the overall plant safety.

2. A common and a modularized platform to DI&C designers, software developers, cybersecurity analysts, and plant engineers to predict and prevent risk in the early design stage of DI&C systems.

As shown in Figure 3, the proposed framework, as a modularized platform, aims to have good communication with various small-scale unit-level software reliability analysis methods (e.g., quantitative software reliability methods) and large-scale system-level reliability analysis frameworks (e.g., probabilistic risk assessment [PRA]). RESHA, as a top-down approach, can identify the digital or software failures in the unit-level interactions inside of a digital system, then BAHAMAS and ORCAS can be used to quantify the probability of the STPA-identified software failures based on suitable existing quantitative software reliability methods such as Bayesian networks, test-based, or metric-based methods.

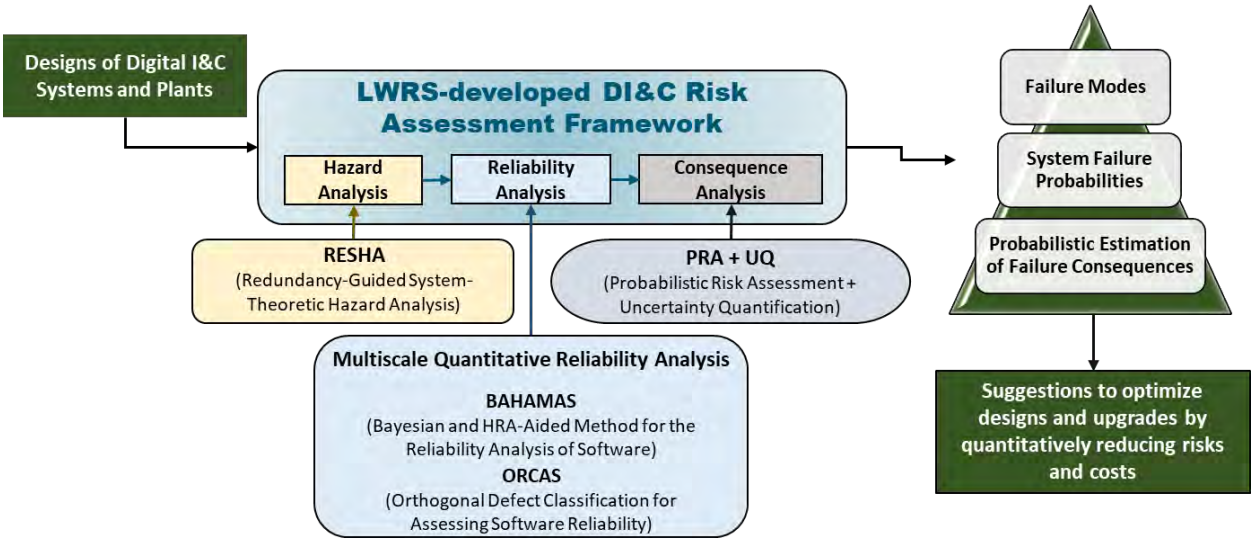

Figure 3. The flexible and modularized structure of the proposed risk assessment framework for HSSSR DI&C systems.

3. Technical bases and risk-informed insights to assist users to address and fulfill the risk-informed alternatives for evaluation of CCFs in HSSSR DI&C systems of NPPs.

Figure 4 illustrates how the proposed framework can support the potentially expanded DI&C CCF policy and licensing of a HSSSR DI&C design or upgrade. NRC BTP 7-19, "Guidance for Evaluation of Diversity and Defense-In-Depth in Digital Computer-Based Instrumentation and Control Systems Review Responsibilities," [13] clarifies the requirement for acceptable methods for addressing CCFs, including

identifying CCFs, reducing CCF likelihood, and evaluating CCF impacts in design-basis events. The capabilities of the proposed framework in hazard, reliability, and consequence analysis matches well with these requirements.

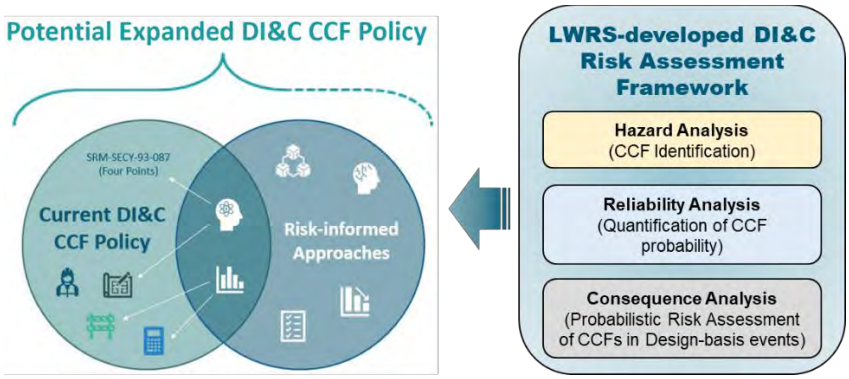

Figure 4. NRC potential expanded DI&C CCF policy vs. the LWRS-developed framework in CCF evaluation (derived based on the NRC's presentation in NRC Public Meeting on February 15, 2022).

This framework also aims to support and supplement existing advanced risk-informed DI&C design guides by providing quantitative risk information and evidence (e.g., failure probabilities, quantitative consequence impacts). The proposed framework can support and supplement EPRI's digital engineering process framework. For HAZCADS, the LWRS-developed DI&C risk assessment framework can provide detailed CCFs in different redundancy levels, quantitative evidence to support the risk importance analysis and the ranking of risk reduction targets, and a quantitative consequence analysis to trace the impacts of individual failures. For DRAM (Digital Reliability Analysis Methodology), the proposed framework can use PRA model to check if the control methods are qualified to mitigate consequences and reduce risks.

4. A risk-informed tool that offers a capability of design architecture evaluation of various DI&C systems to support system design decisions in diversity and redundancy applications.

The proposed framework can be beneficial for the design of HSSSR DI&C systems in plant modernization process; the safety improvements of these new digital architectures are expected to be significant and can be presented more clearly. Currently, it is thought after qualitatively addressing CCFs, all of them need to be fixed by adding diversity, which costs a lot. In fact, some of CCFs do not have significant impacts on the change of CDF or large release frequency. The framework can evaluate the impacts of single software CCFs to the HSSSR DI&C systems and even the plant safety, based on which suggestions can be provided to optimize the D3 application in the early design stage of HSSSR DI&C systems. For instance, it can support the determination of the level of redundancy (e.g., a four-division ESFAS vs. a two-division ESFAS) or the level of diversity (e.g., deployment of software/design/equipment diversity at a division level vs. unit level). By conducting risk-benefit analyses of different redundant and diverse designs, costs can be saved if some CCFs are proved to be insignificant to plant safety. Based on the current analysis results, failure probability of the HSSSR DI&C system due to software CCFs is quite low, and the CDF is also significantly reduced compared to the one with traditional analog systems. The proposed framework was also suggested to deal with the software risk analysis for machine-learning-based digital twins in the nearly autonomous management and control systems [19].

3. REDUNDANCY-GUIDED SYSTEM-THEORETIC HAZARD ANALYSIS (RESHA)

This section documents the methodology development and demonstration of a hazard analysis method incorporated in the framework, called redundancy-guided systems-theoretic method for hazard analysis (RESHA). Section 3.1 provides an overview of RESHA; some basic concepts and terms are introduced. Section 3.2 discusses RESHA's capability in tracing software failures in digital feedback pathways in highly redundant DI&C systems. Section 3.3 describes the methodology of RESHA with details. Results of case studies in the hazard analysis of a representative digital RTS, ESFAS, and safety-related HSI are discussed in Section 3.4.

3.1 Overview

The RESHA method is an FTA-based method that incorporates STPA to identify inner software failures and digital-based failures in Type II interactions. Figure 5 illustrates the concepts of Type I and Type II interactions: Type I, the interactions of a DI&C system (and/or its components) with a controlled process (e.g., NPPs), and Type II, the interactions of a DI&C system (and/or its components) with itself and/or other digital systems and components [20]. Software should not be analyzed in isolation from the complete digital system. In addition to the inner failures of software, failures in Type II interactions should also be considered in the risk analysis of a DI&C system.

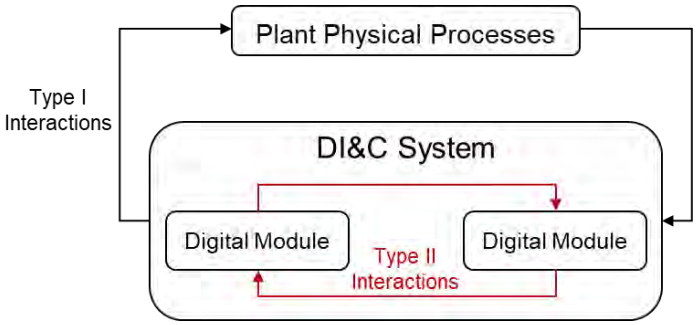

Figure 5. Illustration of Type I and Type II interactions.

According to the difference in functionality, there are normally two types of digital modules in DI&C systems: digital controller and intermediate digital module. The physical elements of a digital controller include a central processing unit along with its associated microcode, memory, and I&C program [20]. A digital controller may be connected to other controllers or intermediate modules (e.g., sensors, actuators, or even input/output modules). STPA handbook [9] provides another definition of a controller in an I&C system; a controller provides control actions on the system and gets feedback to determine the impact of the control actions, as shown in Figure 6. The HSSSR DI&C systems in NPPs normally include multiple controllers to ensure the availability and reliability of the actuation of safety controls and features. For example, the rod cluster control assembly can be dropped by manual control from operators or automated control of RTS. In this case, both operators and RTS can be considered as a controller based on the controller definition from STPA. Digital controllers can be defined in different scales for a different analysis target. For example, a digital processor inside of a RTS can be treated as a controller when details about Type II interactions inside of RTS are needed, while RTS itself can be a digital controller when details about Type I interactions are needed.

In STPA, process models represent the controller's internal beliefs that are used by the control algorithm to determine control actions. Control algorithms specify how control actions are determined

based on the controller's process model, inputs, and feedbacks from previous control processes and intermediate modules. In [18], a process model is thought as the diagnosis portion of a controller, whereas the control algorithm provides actions based on the model's diagnosis.

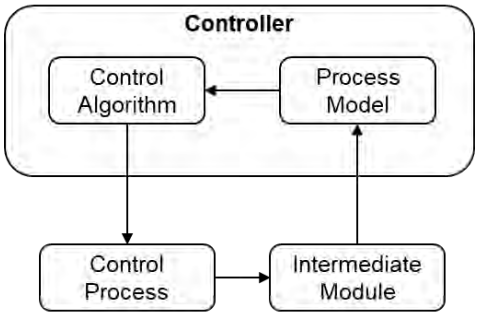

Figure 6. A generic control loop with a controller (derived from STPA handbook [9]).

RESHA identifies UCAs as the digital failures resulting from two categories of causal factors:

- Category 1: Inner software failure
  - Software design defects (mainly due to inconsistent process model)
  - Software implementation defects (mainly due to inadequate control algorithm)
- Category 2: Misleading, inadequate, or incorrect input conditions
  - Failures in Type II interactions.

Regarding Category 2 causal factors, RESHA recognizes these as unsafe information flows (UIFs) from dependent intermediate modules. These are explicit indications that there exist digital-based failures from inappropriate Type II interactions. For clarification, an unsafe information flow is regarded as any received signal external to the controller that is erroneous, falsified, or incorrect. In contrast to a UCA, UIFs are failures in the feedback pathway of the control structure, whereas the former is a failure in the controller pathway. They are considered one class of casual factors for UCAs under the STPA methodology. A causality relationship diagram is provided below to show how UIFs are related to UCAs and associated hazards/losses. Some of the more relevant Type II interactions are listed below but is not exhaustive:

- Data transmitted by intermediate module is correct but received/interpreted incorrectly at controller
  - Excessive noise distortion
  - Data transmission pathway degraded (e.g., loose connecting wire)
  - Data feedback to controller has incorrect timing/order
- Data transmitted by intermediate module is not correct
  - Internal software failure related to design or implementation defects
  - Data received by the intermediate module from other intermediate modules is incorrect resulting in the output also being incorrect.

RESHA relies on UCAs, UIFs, and an integrated FT to achieve its primary outputs. There are different categories of UCAs (e.g., control action not provided, or control actions provided spuriously) which are integrated with the FT to account for software failures. Depending on the goals of the risk assessment, the integrated FT may also be combined with an event tree (ET) as part of a larger comprehensive risk assessment. For such assessments, the ET links with FT top events which, in turn, guide the selection of UCA/UIF types to be used within the FT itself. For example, a top event pertaining to failure on demand will contain UCAs/UIFs that match that same category.

3.2 Failure Mechanism Traceability

3.2.1 Challenges in Tracing Software Failures

The versatility and capability of software on digital platforms is especially attractive to the nuclear industry, which has traditionally relied on analog counterparts. The prospect of advanced capabilities, increased economic viability through greater automation, and improved overall safety are all examples of the benefits of modernization. However, this also introduces significant challenges for predictable digital systems availability and reliability. Imagine if the operating system for a NPP were to suddenly experience a "blue screen of death", the consequences would be non-trivial. Therefore, software failures[a] in DI&C systems have become an increasing issue for mission success [13]. Both single failures and CCFs may impact the overall reliability of a system. For instance, while diverse control trains can successfully mitigate hardware CCFs, diverse software implementation has not been shown to offer the same level of defense-in-depth and diversity. The main argument is that coding education is too homogenous such that the functional difference between software is insufficient to be considered diverse [21]. The impact is that software systems are highly susceptible to single failures and CCFs overall lowering mission success.

However, adequate identification of these failure modes is also a significant challenge. Unlike conventional hardware systems, where a failure is simply the inability to perform a prescribed action, software failures are not intuitive, may emerge from complex interactions, and/or may be the result of inadequate or misinformed design choices. STPA identifies potential UCAs that software control systems can cause due to intentional or unintentional design and the corresponding undesirable loss scenarios. A key assumption is that the system assessed must have "authority" over the physical process to enact control actions (e.g., sensors do not have authority over the process they are monitoring but can influence controllers). Control actions become unsafe when they occur within select contexts or conditions resulting in a defined loss. The STPA manual also discusses the failure mechanisms behind UCAs, including an inadequate control and process model, unsafe control input, and inadequate feedback and information [9]. In Figure 7, information flows in a counterclockwise manner, where the failure mechanisms of both the control/actuation pathway (CAP) and information/feedback pathway (IFP) can be seen. UCAs, in this diagram, are the outputs from the controller that can cause an undesirable loss. There exists another type of unsafe failure in this system, specifically failures of the upstream dependencies to the controller. This is especially prevalent in control-free systems such as those for monitoring and processing.

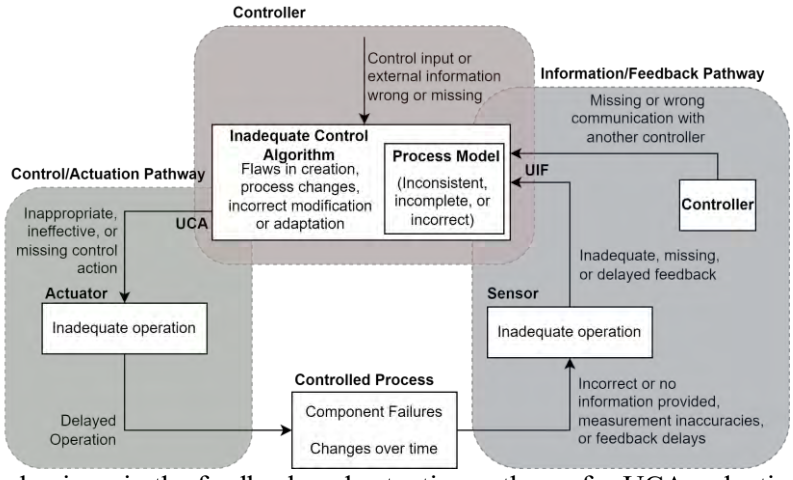

Figure 7. Failure mechanisms in the feedback and actuation pathway for UCAs adaptive from [9].

[a] In this work, "failure" is used generally as an undesirable outcome by the system, which includes unintended design consequences or "misbehaviors" as well as inability to perform necessary intended functionality.

For example, in the Advanced Power Reactor 1400 (APR-1400) HSI [22], monitoring of key reactor variables is conducted by the qualified indication and alarm system for safety (QIAS-P). This system is defined as a safety-critical system but has no capability to change the reactor state. Within a division of the QIAS-P, there are multiple intermediate feedback systems that parse, validate, and augment the data before forwarding. Expressing UCA failure modes in such a control-free system would thus be difficult and would exclude the majority of the software failures in the system. These problems also extend to conventional nuclear control systems, such as the RTS and ESFAS where control is dependent on various smart sensors and feedback systems. We introduce the idea of UIFs as software failures along the IFP as a solution to this problem. A UIF occurs whenever a failure occurs in a digital component that does not have authority over the physical process but rather augments, transforms, or parses data along the IFP. These components are referred as to intermediate processors. Identified UCA/UIFs can also be included in fault trees to establish causality relationships between software failure modes and corresponding loss scenarios. Fault trees are useful as graphical tools to assess how the failure of one component can lead to other downstream failures and their impacts. In this report, we further develop and extend the concepts in RESHA to capture software failures in upstream controller dependencies, specifically in the intermediate processors of the IFP. This addition captures a previously overlooked portion of the DI&C system.

3.2.2 The Concept of Unsafe Information Flow

Before presenting the approach, the theory of software failure is discussed. There are generally three terms related to failure used widely in standards such as IEC 61508 [23] or IEC 62740 [24]. They are root cause, failure mechanism, and failure mode. For consistency, terms developed from the IEC standards are utilized. Root causes are the most basic; they are defined as causal factors with no predecessor relevant for the purpose of the analysis such that if corrected, would prevent recurrence of failure [24]. Determining relevancy is defined by a stopping rule that is a "reasoned and explicit means of determining when a causal factor is defined as being a root cause" [24]. A failure mechanism is the "process that leads to failure" and may be physical, chemical, logical, psychological, or a combination thereof [24]. A failure mode is the "manner in which a component fails functionally" [25]. A relationship between root causes, failure mechanisms, and failure modes can be seen in Figure 8 where each element leads to or triggers the next.

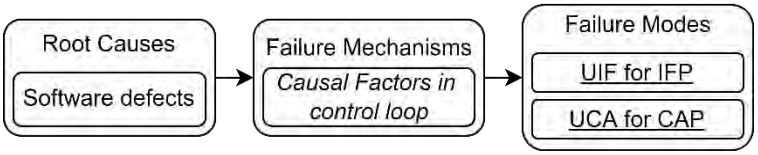

Figure 8. Rudimentary relationship between root cause, failure mechanisms, and UCA/UIF software failure modes.

In HAZCADS, UCAs are added to fault trees as software failure modes and actions that can cause defined losses under worst-case scenarios if it occurs. Under the same loss scenarios, the failure mechanisms describe the cause of UCAs and represent hazardous states. Importantly, failure mechanisms describe state processes and do not explicitly cause failures themselves. Therefore, it is inappropriate to include them into the fault trees. Rather a change in perspective is necessary when discussing failures of intermediate processors in the IFP. Consider a simple doorbell mechanism seen in Figure 9. From the viewpoint of the doorbell, the system failure modes could be (1) fails to ring when button pushed and (2) inadvertently rings when button not pushed. The potential failure mechanisms could be (1) switch fails to make contact when pressed, and (2) switch fails to break contact when not pressed. However, from the perspective of the switch, the corresponding failure modes to the above mechanisms could be (1) high contact resistance buildup, and (2) contact spring is broken.

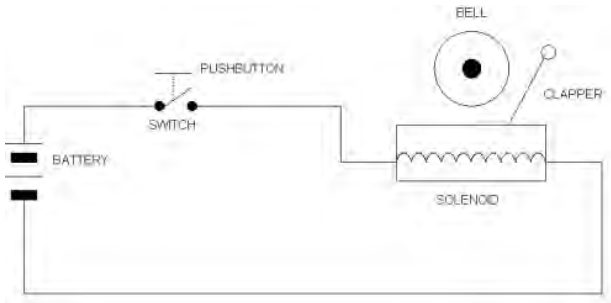

Figure 9. Doorbell and associated circuitry [26].

The importance to draw from this example is that software failure mechanisms in the IFP can be modified by a change in perspective to the intermediate processor upstream to the controller and represented as failure modes in the fault tree. The condition is the corresponding system loss scenario must be consistent throughout the fault tree. The failure modes of intermediate processors in the IFP and related to the top event are thus known as UIFs.

The UIFs are organized into four categories based on failures of intermediate processors in the IFP: (1) failure to provide information when "needed" (UIF-A); (2) providing information when not "needed' (UIF-B); (3) providing information but is either early, late, not in sync, or out of order (UIF-C); and (4) providing information that is too low or too high, not-a-number (NaN), or infinity (inf) (UIF-D). The context is vital to all UIFs, and the term "needed" above must be replaced with the appropriate hazard contextual conditions that can lead to a loss. For instance, a UIF-A, "stove is hot" indication light is off when a person touches the stove top (causing a burn). The contextual information, 'touches the stove top' defines when the loss manifests and 'causing a burn' defines the consequence of the loss. Missing this contextual information results in superfluous failure information that can complicate the assessment.

UIFs are separated into four categories related to the type of intermediate processor assessed. For continuous or high-demand systems (e.g., real-time monitoring), only UIF-A, C, and D are applicable. There is rarely a context where information is provided when not needed (erroneous information provided falls under UIF-D). For low-demand systems (e.g., alarms, notification, polling/probes), all categories are applicable. Chaining of UIF events is also a valid construction as failures in one information system can lead to failures in other information systems. For a particular component, either UIFs or UCAs failure modes are possible, but not a combination of both as the component is either a controller or an intermediate processor. If such a scenario arises, then the component must be further decomposed into its subcomponents. Recall that UIFs are failure modes from the perspective of intermediate processors but also failure mechanisms for UCAs. Therefore, if a UCA exists for a controller, then there also exists UIFs that can be decomposed from the controller dependencies. The inverse is not always true; if a UIF exists for an intermediate processor, a UCA failure mode does not necessarily exist.

Lastly, there are generally two categories of software failures separated by where the failure mechanism is located: internal factors, such as inadequacies in the control algorithm or process model, and external factors, such as inadequate feedback and information input from dependencies. In this first category, failure modes manifest due to two primary deficiencies. The first is an incomplete design process where not all relevant constraints, requirements, or conditions needed for intended operation are considered. The second deficiency is an incomplete engineering/implementation process, where hidden assumptions or human errors in the software coding process results in residual defects. Hardware failures of the software platform are also included in the first category (i.e., integrated circuit burnout). The second category of failure modes manifest in the faulty transference of knowledge between systems. Specifically, these include failures in the data transmission infrastructure, such as input-output ports, wires, sensors, excess noise, malicious spoofing, etc., as well as failures in dependent hardware and software devices. While mechanisms are identified in [9], the tracing of these failures to specific DI&C components are not considered in the scope of their work. The primary difference between the failure

categories is the location of the inadequacy. In category one, the failure is from an internal defect; while in category two, the defect is external. This is used later to organize basic events in our fault tree.

A simple example is used to illustrate the concepts presented below. The function of the system is to turn the lights on when the sun sets. A luminosity sensor measures the light level and outputs an analog voltage level. The smart processor coverts the voltage value to digital logic for use by the controller. It also combines values from a clock to account for seasonal changes. The controller has a single condition; if the light level and time of day reach a prescribed value, activate the relay. A relay is a digital switch that when activated, provides power to the light bulb. The loss is defined as unnecessary power usage (i.e., lights are on when the sun has not set). The blocks are colored to show which parts of the control loop each belong to.

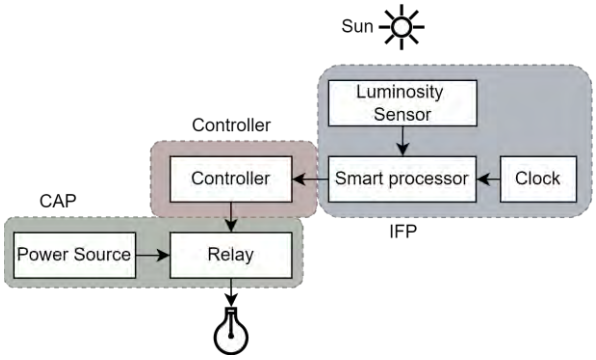

Figure 10. Basic light control system.

In this work, the smart processor is classified to as an intermediate processor. A UIF failure mode for the smart processor could be that the light-level-output signal suggests that sun has already set when it may be midday. The root cause of this UIF could be that the logic and time zone were tuned incorrectly outputting a low level regardless of the actual light level. The controller will turn on the light as programmed perfectly; however, the light will always be on regardless of light level. A UCA failure mode of the controller could be that the relay activation signal is always on regardless of the proper functionality of components in the IFP. The root cause of this UCA could be that the condition for activation was set too low such that the controller always believes it is dark. Importantly, notice that failure of the system has occurred in two separate locations but resulted in the same loss scenario. The UIF failure was exterior to the controller while the UCA was due to an internal defect in the software. Performing mitigation strategies only on the controller would not have resolved the UIF as the controller was not inherently at fault. Therefore, it is highly relevant to locate exactly which aspects of the system fail such that the proper mitigation strategy can be applied at the correct location. While this simplistic example has only one intermediate processor and controller, in real nuclear DI&C systems (e.g., QIAS-P), there may exist multiple redundant and diverse intermediate processors.

Lastly, there are a couple of key constraints to the definition of intermediate processors. First, they cannot be purely software and must have an associated hardware platform. An AD41111 is an example of an analog to digital (ADC) component that can interpreted as an intermediate processor. Without this constraint, the tracing of information feedback failures can become overly convoluted—especially for software variables. This is because software may have hundreds of variables stored in its memory. The relevance of each parameter is difficult to assess and would not contribute meaningful information to the safety assessment. By extension, only the output information for physically separated components can have associated UIFs. Variables passed to each other within the same software are not relevant enough to the analysis to be considered. Generally speaking, intermediate processors are separated by either a communication protocol (e.g., I2C, SPI, and UART) transmitted by a physical wire or by a wireless network protocol (e.g., Wi-Fi 802.11). Components on the same printed circuit board (i.e., integrated ADC on a programmable logic controller) separated by traces can be included but are highly discouraged as the assessment is too granular and will not have a meaningful impact on mission success.

3.3 Methodology

RESHA incorporates the concept of combining FTA and STPA from HAZCADS. STPA is reframed in a redundancy-guided way to address CCF concerns in highly redundant DI&C systems. The main outcomes of RESHA are (1) the identification of CCFs and potential single points of failure (SPOFs) in the DI&C design; (2) an integrated FT including both hardware and software failures, individual failures, and CCFs; and (3) hazard preventive strategies. The acceptance criterion of risk evaluation for the RESHA is "does the individual digital-based failure lead to the loss of function of the DI&C system?" In another words, is there any SPOF existing in the system that may lead to the failure of DI&C system? A seven-step process, shown in Figure 11, illustrates the workflow of RESHA in the proposed framework for the hazard analysis of DI&C systems, especially for CCF analysis of highly redundant HSSSR DI&C systems. Previous versions of RESHA can be found at [16, 27].

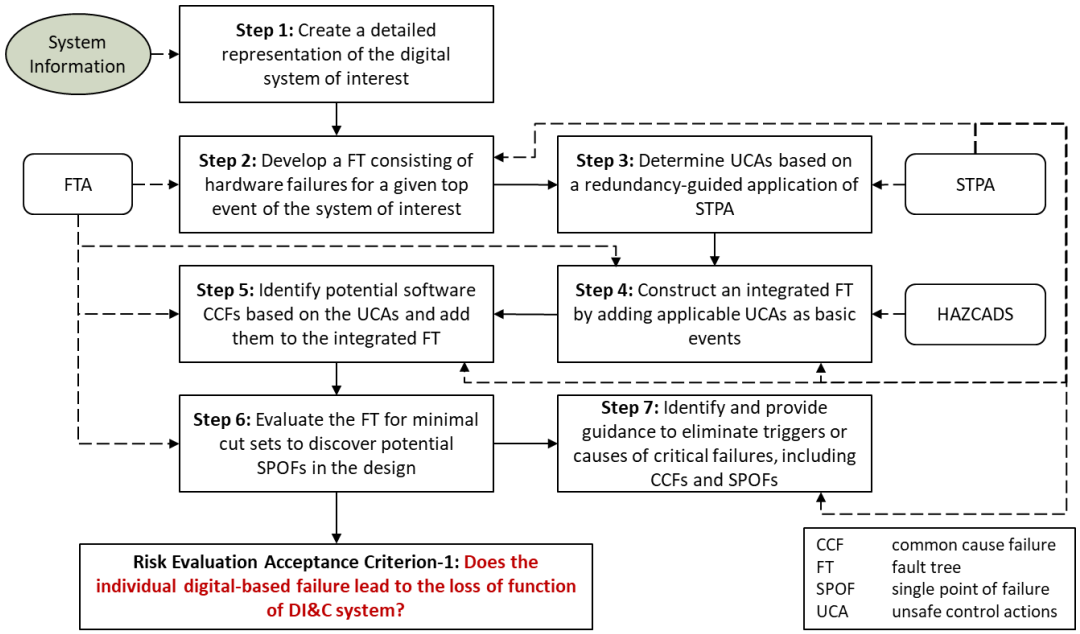

Figure 11. Workflow of the Redundant-guided System-theoretic Hazard Analysis (RESHA) in the proposed framework for the hazard analysis of DI&C systems (derived from [28]).

Step 1: Create a detailed hardware representation of the digital system of interest.

The purpose of Step 1 is to establish a system sketch to serve as a blueprint for the hazard analysis. This is done by gathering system design information, including wiring, piping, and instrumentation diagrams, existing logic diagrams, etc. This information is then used to create a system sketch with the main goal of mapping out the processors, sensors, controllers, components, interactions, and connections of the system. The point of this step is not to necessarily fit everything into one diagram but to gain a sufficient understanding of the system in order to complete the hazard analysis; the level of detail provided in this step lays out the foundation for the work.

In this step, detailed information on the structure and functions of the digital system of interest should be collected, gathered, and classified. Normally, a HSSSL DI&C system has a three-level hierarchical architecture [29]: (1) divisions that process the signal path from sensor to actuator level (e.g., the four-division design in APR-1400 ESFAS), (2) units that perform a specific task by using several modules (e.g., an acquisition and processing unit or a voter unit), and (3) modules that realize a specific part of the function processing (e.g., input/output modules and processors). The representation should contain information on hardware structure and be created to a detailed level that captures sufficient design information affecting system function and reliability. In this work, most efforts on hazard identification and reliability modeling reach to the level of modules, which is the smallest hardware component to

implement a specific part of the entire function processing independently. In addition, based on the requirements and purposes of the risk analysis phase, practical assumptions and reasonable simplifications of the hardware representation should be stated and explained in this step. The representation figure should clearly display the information flow between different divisions, units, and modules. For the analysis on digital systems with redundancy designs, the complexity of redundancy should be illustrated in the hardware representation. It builds the basis for the construction of hardware FTs and redundancy-guided multilayer control structure.

The construction of the block diagram follows a top-down approach starting at the controller then branching along the CAP and IFP. It is not necessary and undesirable to create a perfect representation of every single element in the control system this would over complicate the assessment. Rather an iterate approach should be adopted, adding detail to the diagram as necessary to sufficiently understand the sources of risk to the mission objective. In STPA, there are great examples of how top-down decomposition should be conducted to maximize efficacy while minimizing tedious and arbitrary levels of detail. However, what is important in the block diagram is to clearly identify the flow of information from various systems, subsystems, and devices in both the CAP and IFP. The initial block diagram should emphasize the scope of the analysis and the boundaries.

Several key points are considered for Step 1:

- Step 1 emphasizes the boundary conditions and scope of the analysis. These should be clearly understood as they will be revisited in Step 2 and Step 3.
- Though the RESHA has been developed to analyze digital systems, this system sketch should also include the hardware structural arrangement (i.e., the components of the system in addition to details collected for the digital structural arrangement).
- The level of detail included in a hazard analysis can extend beyond module level failures to the components and sensors providing input to process modules. The level of detail to be included in the hazard analysis depends on the scope of the investigation.

Step 2: Develop a FT consisting of hardware failures for a given top event of the system.

Based on the hardware representation created in Step 1, a FT is developed in this step to include hardware failures to the detailed level required for representing the loss of functions. For analysis of a digital system with redundancy designs, the structure of a hardware FT should follow the levels of redundancy from the division to the unit and to the module levels. This kind of redundancy-guided structure makes it convenient to add in a software failure identified in the next step. The probability quantification of each basic event is not required in this step and will be performed in the integrated reliability analysis. The main assumption for this step is that all software failures will be captured using STPA in Step 3. Therefore, only hardware failures will be included in FT, which is created using the two-part process adapted from the U.S. National Aeronautics and Space Administration (NASA) Handbook [26].

Step 2A: Define the boundary of the analysis (revisited from Step 1). This includes selecting a top event and resolution for the analysis. Top events are based on the purpose of the system of interest (SOI). The failure of the SOI's most significant function is a priority event to be analyzed by the FT—a top event. Step 2A may also be visited briefly to ensure the proper selection of top events. Step 2B: Construct the FT. Starting from the top event, construction proceeds by determining the "necessary and sufficient immediate events" contributing to failure of the top event [26]. This process is repeated down the tree by analyzing each subsequent event step-by-step, ending with an event that can be resolved no further (either by way of fact, or by the discretion of the analyst).

Software failures are added in Step 3, so a placeholder is inserted in the fault tree temporarily. To assist in the organization of the fault tree, three branches for each top event related to digital devices are

recommended: a hardware failure branch, a software failure branch due to internal failure mechanisms, and a dependency failure branch due to all other external failure mechanisms as seen in Figure 12.

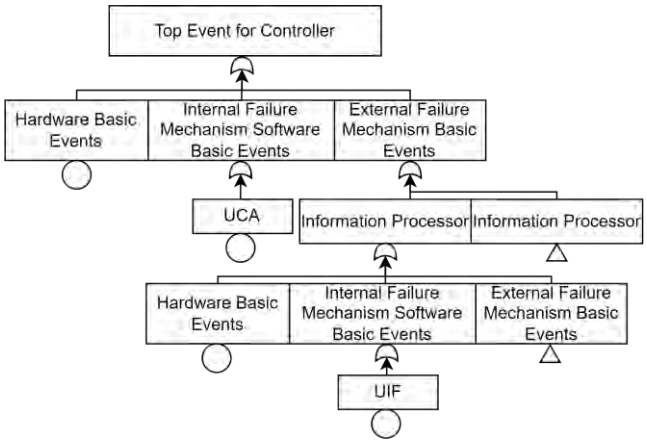

Figure 12. Example of fault tree organization with one controller and two information processors.

Internal and external basic events are organized from the perspective of the current top event. Under the hardware basic event branch, relevant random failure modes of the hardware platform are included such as component burnout, frayed contacts, etc. Under the internal failure mechanism branch, software failure modes due to internal failure mechanisms in the control algorithm or process model are added (in Step 3). Under the external failure mechanism branch, the failures in the upstream dependencies are added and can include both hardware and software failures. This typically will include other controllers, information processors, or sensors. Following an iterative approach to fault tree construction, eventually the dependency branch will be empty as there will be no other dependencies to account for. Using Figure 10 as an example, the Step 2 fault tree is constructed and seen in Figure 13. The events are colored to show which part of the control loop they belong to, but this is only for illustrative purposes. The top event is defined as “lights are always on resulting in unnecessary energy loss.” From the perspective of the controller, the immediate hardware failure occurs if the controller to relay connection is shorted, causing a constant ON state. Under the controller software branch, a placeholder is inserted for software failure modes and will be added in Step 3. Under the dependencies branch, all upstream controller dependencies are listed. From the perspective of the smart processor, three additional branches are added. The only difference is that under the dependency branch, there are no additional dependencies aside from the peripheral sensors.

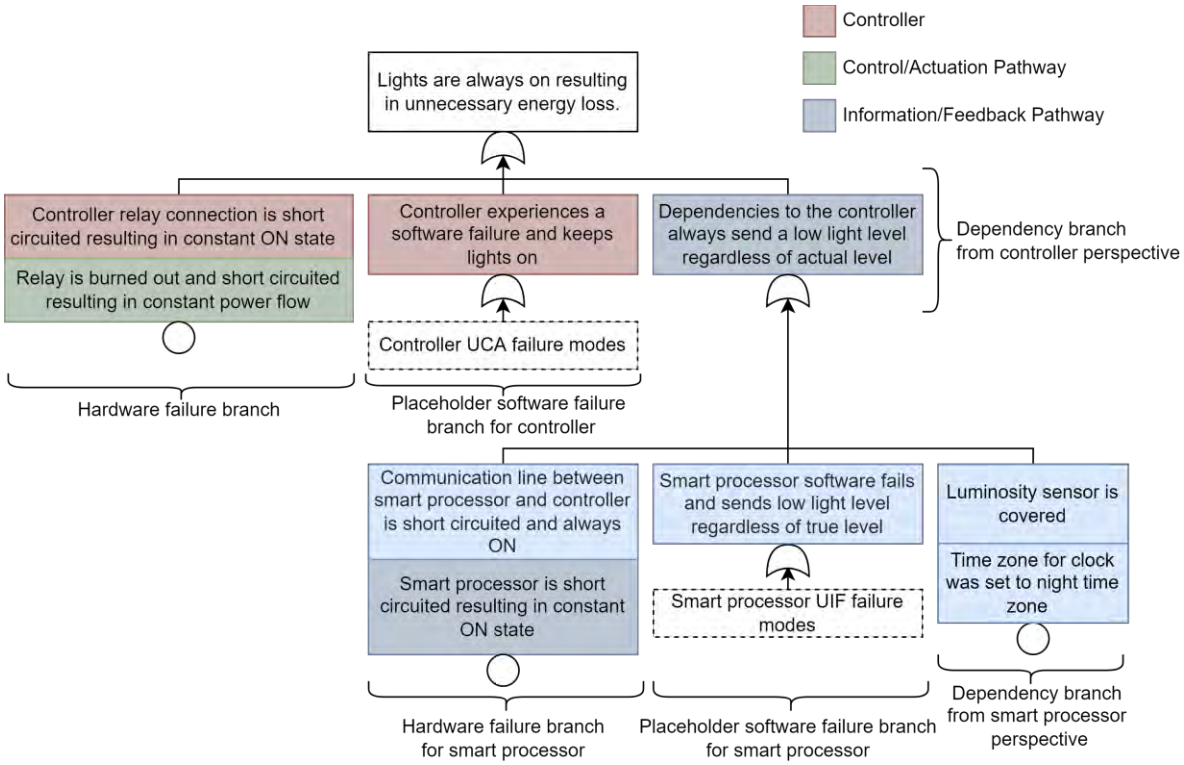

Figure 13. Organization of fault tree aligned in with failures due to internal and external mechanisms.

This separation of internal and external failure modes is derived directly from the failure groups discussed in Section 3.2.2 and is necessary to prevent confusion on where and in what ways systems can fail. Furthermore, this clarification of tree construction organizes failures in a logical manner and is traceable from the control block diagram. Lastly, redundant and diverse divisions should also be included in the hardware fault tree as they will later become possible sources of software CCF. Hardware CCF should be included in this stage and can be identified whenever a duplication of hardware among redundant divisions exists.

Step 3: Determine UCAs/UIFs based on a redundancy-guided application of STPA.

In this step relevant UIFs and UCAs are defined and inserted under the "internal failure mechanism software basic event" branch. In a digital system, all information exchanges—including the decision-making process of the controllers, control and implementation of control actions, performance of controlled process, and feedbacks from controlled process—have a potential to fail the function of the digital system when it is needed or send spurious signals that are not needed. These systematic failures can be initiated by the UCAs or UIFs as a result of an unrealistic process model, an inappropriate control algorithm, incorrect feedback, or outside information. Therefore, the potential software failures can be understood and analyzed by identifying these UCAs. To deal with the complexity problem of redundancy and to identify software CCFs effectively, the control structure is built in a redundancy-guided way. The redundancy-guided multilayer control structure zooms in on systematic information exchanges on each redundancy level because CCFs are tightly connected with redundancy designs. Each control structure layer is created with numbered control actions and feedback signals until a final, redundancy-guided, multilayer control structure is created for the complete system of interest, as shown in Figure 14.

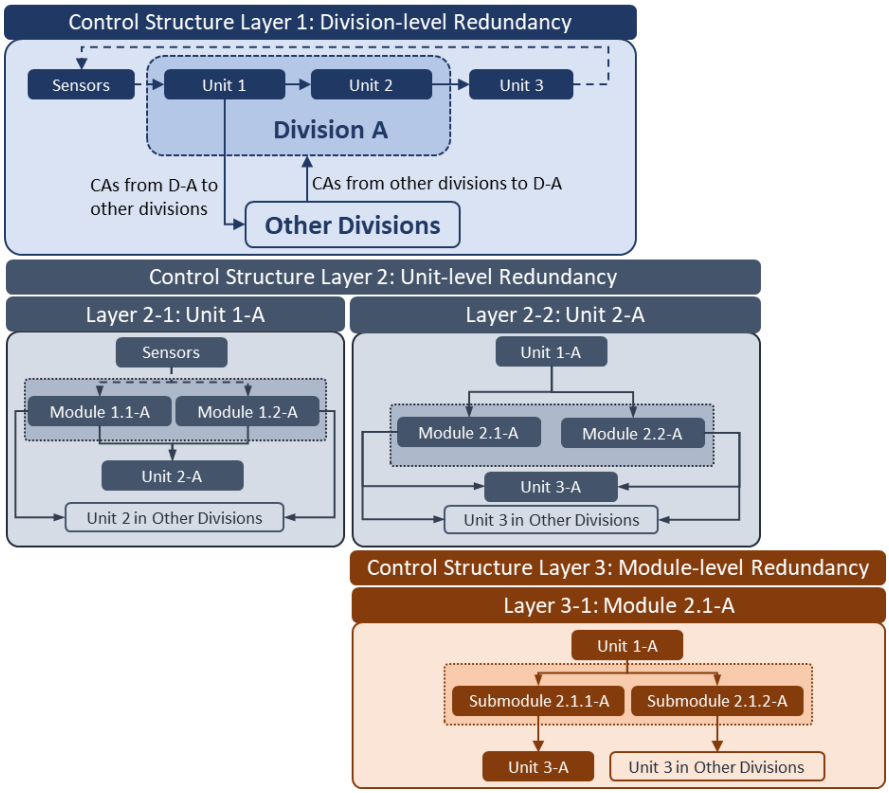

Figure 14. Illustration of a multilayer control structure.

The UCAs are identified according to the multilayer control structure and specified hazards. A UCA is defined as, "a control action that, in a particular context and worst-case environment, will lead to a hazard" [9]. There are four types of UCAs in an STPA:

- Control action is not provided when it is needed
- Control action is provided when it is not needed
- Control action is provided when it is needed, but too early, too late, or in a wrong order
- Control action lasts too long or stops too soon (only applicable to continuous control actions).

Each UCA should take the following format for consistency [9]:

$$\boldsymbol{UCA} = [\boldsymbol{source}] + [\boldsymbol{UCA\ type}] + [\boldsymbol{Control\ Action}] + [\boldsymbol{Context}] + [\boldsymbol{Link\ to\ System\ Hazards}] \quad (1)$$

The specification of the context for UCAs is important; usually words like "when," "while," or "during" are used to define the context. The UCA context should represent an actual or true condition that would make the control action unsafe and not a controller process model that may or may not be true.

As mentioned previously, UIFs are identified based on the physical separation of the hardware components for intermediate processors. A top-down systems-analysis approach (i.e., STPA) is used to deconstruct the control or monitoring system into relevant subsystems and components. Here, it is important to identify the major hardware components, inputs and outputs of each module, and dependency across components. For general purpose monitoring systems, there may be multiple outputs from the physical system. Each variable is assumed to have equal importance in the analysis and treated as a batch. Any incorrect signal in the batch is assumed to trigger the corresponding UIF (i.e., 1 out of 5 output signals is missing when needed is considered a UIF-A event).

The UIF types are described and follow similar categories as the UCAs. In Table 1, the original UCAs are provided for comparison against UIF formulation. The formulation of UIFs follow a similar format to UCAs) above. The parts of a UIF include the source, UIF type, information flow, context, and link to system hazards. After relevant modules and UIFs have been identified, they can be included in the fault tree under their corresponding software failure branch.

Table 1. Comparison between unsafe control actions and information flows.

	Type A	**Type B**	**Type C**	**Type D**
Unsafe Control Action (UCA)	Control action not provided causing hazard.	Control action provided causing hazard.	Control action is early, late, or out of order.	Control action is stopped too soon or applied too long.
Unsafe Information Flow (UIF)	Feedback is missing when needed causing hazard.	Feedback is provided when not needed causing hazard.	Feedback is early, late, out of sync, or out of order.	Feedback value is too low, too high, NaN, or Inf.

An important assumption regarding UIFs is that they always trigger UCA failure modes under the worst-case condition. This assumption is reasonable as dependencies that fail will result in a hazardous condition for the controller. Any action taken by the controller will thus initiate a UCA. The immediate benefit to this separation of UIFs and UCAs under different failure mechanism branches is that it offers superior clarity when tracing failure sources. It allows analysts to compare the number of internal and external factors that can cause UCAs, thereby focusing on which aspects of the system need improvement. Furthermore, dependency branches can be added to existing fault trees without modifying already identified UCA failure modes. This allows for UIFs to be incorporated into many existing methods like RESHA and HAZCADS. Ultimately, the goal is to understand which areas of the DI&C control system introduces more risk to operation (e.g., controller software vs. dependencies).

Step 4: Construct an integrated FT by adding applicable UCAs/UIFs as basic events.

In this step, relevant software failure modes from Step 3 are added to the hardware FT from Step 2. Importantly, not all UCAs/UIFs identified will fit into the one FT. Rather, only UCAs/UIFs that can result in the top event selected in the FT are added. For example, if the top event selected was a failure to actuate a control train, then only UCAs and UIFs with loss scenarios leading to this event are added into this FT. For controllers and information processors, UCAs/UIFs due to an internal failure mechanism are added under the “internal failure mechanism software basic events” FT branch. If a UCA/UIF is identified that is relevant but is not due to an internal failure mechanism, then it is added under the corresponding dependencies’ internal failure mechanism branch. The overall flow to add UCA/UIFs into the hardware fault tree can be seen in Figure 15.

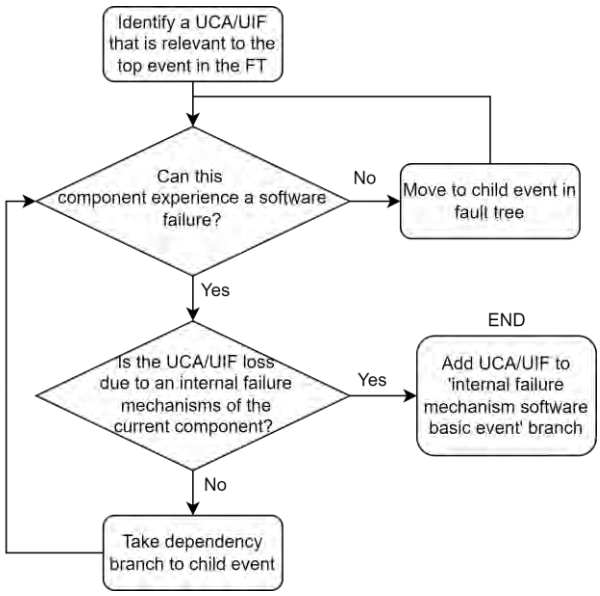

Figure 15. Decision flow when adding UCAs/UIFs to hardware fault tree.

Step 5: Identify potential software CCFs based on common or duplicate UCA/UIFs within the FT.

A software CCFs is specifically when shared commonalities among the design or implementation of the source code result in more than one UCA/UIF among different control trains. For instance, if two control trains utilize the same software architecture to control a physical process, a defect that is common among the two will trigger a UCA/UIF when the same input conditions are provided to both. Furthermore, CCFs are not restricted to redundant trains or to a single level. Shared software features among diverse trains across different levels can also result in CCFs. For instance, a component can share a CCF with common components of its own unit, of common units across divisions, or with common modules across divisions. These shared characteristics of a component could be the use of the same programmable logic controller (PLC) manufactured by the same company with the same low-level operating system but having different high-level functions. Exactly what constitutes for sufficient similarity such that a non-negligible CCF probability exists is beyond the scope of this work. However, we can identify potential software CCFs by identifying all common UCAs/UIFs that exist in the integrated fault tree. If a CCF event is speculated, it is added under the "internal failure mechanism software basic event" branch and duplicated to all susceptible components.

Step 6: Solve the FT for the minimal cut sets to determine potential SPOFs in the design.

As the main outcome of the systematic-theoretic hazard analysis, the minimal cut sets of the integrated FT should be calculated and evaluated to determine how many potential SPOFs have been added by considering the software failures. If the digital system has a low level of software diversity, the software CCF types occurring in all divisions could lead directly to the top event (e.g., the loss of function of the entire digital system), regardless of the contributions from other safety designs. As a part of risk analysis, hazard analysis directly provides evidence to evaluate the question, "does the individual digital-based failure lead to the loss of function of DI&C system?" If the individual digital failure is one of the SPOFs, a redesign request will be made for system designers and engineers based on the risk evaluation results.

Solving the FT is typically conducted using an FTA software package such as Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) [30], which assesses the minimal cut sets. A cut set is a collection of linearly causal basic events that will result in the top event. The number of basic events in a cut set depends on the selected top event of interest. The minimal cut set is therefore a cut set with the least number of basic events that can lead to the top event. Accordingly, a

SPOF is a cut set with only one basic event can lead to the top event. By evaluating the list of SPOFs, designers can determine where and how the system of interest is vulnerable and redesign accordingly.

Step 7: Identify and provide guidance to eliminate triggers of critical failures in the design including CCFs and SPOFs.

In software, latent dormant defects in the software that trigger failures are only active once a specific set of initiating conditions are achieved (i.e., triggering condition). Therefore, in this step, internal failure mechanisms and causal factors that can potentially trigger a UCA or UIF are identified and mitigated against. Unlike in previous revisions of RESHA, causal factors were separated into external and internal categories [16]. The internal category included failures related to the controller itself due to an inadequate control algorithm and process/belief model. The external category included failures related to information/feedback that is not received by the controller such as inadequate information/feedback and unsafe control input from other controllers. In this work, separation of failure categories has already been accomplished via the dependency branch. Therefore, only the internal failure mechanism group and causal factors need to be accounted for and mitigated. This update of the methodology provides increasing resolution on exactly where the software failures can occur, how they can impact upstream systems, which causal factors have yet to be addressed (and whether it is required), and can reduce the overall uncertainty regarding how the system can fail.

3.4 Case Study

A demonstration of the proposed method is conducted on various systems. The first system is a smart sensor platform [31] with a lightweight operating system (ChibiOS) [32]. The ChibiOS is an example of an all-purpose multiuse resource management system. The ChibiOS is also compatible with multiple different hardware platforms and does not have dedicated hardware to separate functionality. The ChibiOS software was used to control various functionalities of the smart sensor including filtering and processing of the analog sensor values as well as communication to peripherals. The main processor for the Virginia Commonwealth University (VCU) sensor—STM32F4 ARM Cortex-M4—can be used for a variety of functions and can compute numerous intermediate variables. A conventional assessment of the operating system hardware or software would be difficult as tracing all parameters used would be highly labor intensive. The sensor software (including ChibiOS) and hardware were integrated as a physically separate module on the ARIES_2 Advanced Autopilot Platform for an unmanned vehicle control. To determine altitude, ARIES system uses pressure and temperature from the sensor's main intended outputs, which are then used to adjust true altitude. This case study is used to demonstrate how software failures in generalized hardware architectures can be modeled with UIFs. An integrated fault tree is developed based on the overall system description, and qualitative safety improvements are provided. The case study is described in Section 3.4.1.

The second case study is on a representative HSI modeled after the QIAS-P from the APR-1400 [22]. Note that this HSI is listed as safety-critical infrastructure for reactor trip safety. The HSI presented hereafter is not a true replica of the APR-1400, and the mentioned components are assumed from public documentation. All specifications pertaining to the QIAS-P were derived from publicly available design information from reference [22]. While the exact design of the system is proprietary, sufficient detail was provided to approximate the structure of the system. This case study does not constitute a complete safety assessment but demonstrates how various interweaving and dependent modules can be modeled in a highly complex information retrieval system. The QIAS-P was implemented to provide safety-critical information and alarm notification on the reactor state through various sensors such as the heated-junction thermocouples (HJTCs) and core exit thermocouples (CETs). Within the system, there are two redundant divisions (e.g., A and B), which are assumed architecturally identical, as diversity was not specified. Within each division, there are various value calculators that determine intermediate variables, such as the reactor coolant saturation margin (RCSM) as well as alarms for each intermediate variable. Aside from the QIAS-P, the HSI design also incorporated diverse monitoring systems such as the Qualified Indication

and Alarm System for Non-Safety (QIAS-N), the Diverse Instrumentation System (DIS), and information processing system (IPS). These systems are mentioned for completeness but not assessed in the case study. In Section 3.4.1, only one control action is assessed for the entire monitoring system which is the operator manual reactor trip. It is likely the system also influences other control actions, but in this work, only actions relevant to reactor trip safety were considered. Importantly, no automated trip safety features can be influenced as the monitoring systems was intended to be independent from the existing automatic RTS system and provide an unattestable reading of the physical plant state. This case study is used to demonstrate how multiple layers of intermediate processors, within a complex multi-division DI&C architecture, can be modeled following UIF principles.

The RESHA method was also demonstrated based on a representative four-division digital RTS and ESFAS, which were modeled based on the DI&C design of APR-1400 [22]. Results have been included in previous milestones and journal articles and briefly discussed in Section 3.4.3.

3.4.1 VCU Smart Sensor Integrated Fault Tree Analysis

The exact details of the VCU smart sensor system can be found in references [31, 33]. A full analysis using RESHA is conducted for clarity on each step.

Step 1: Create a detailed hardware representation of the digital system of interest.

The smart sensor system measures various pressure and temperature variables. These values are then used by an altitude controller to ultimately increase or decrease the UAV's altitude. The control block diagram of the smart sensor system is derived from the schematic shown in Figure 16. Starting from left to right, the system is comprised of: (1) "sensor/Arduino" which are the various pressure and temperature sensors connected to the system, (2) "LCD display" which convey relevant environmental information to the UAV operator, (3) "MS5611_thread" which is a software module that translates received data into usable information, (4) "display_thread" which is another software module that manages incoming data to be displayed on the "LCD display," (5) "comm_thread" which buffers information to and from the "MS5611_thread" from external peripheral sources, (6) "serial_modem" software module which handles all incoming and outgoing data packets to the altitude controller, and (7) "host machine" which represents consumers of the information provided by the smart sensor and includes the altitude controller. The hardware information was also collected and can be seen in Table 2. Based on this information, the approximate control block diagram is created and shown in Figure 17.

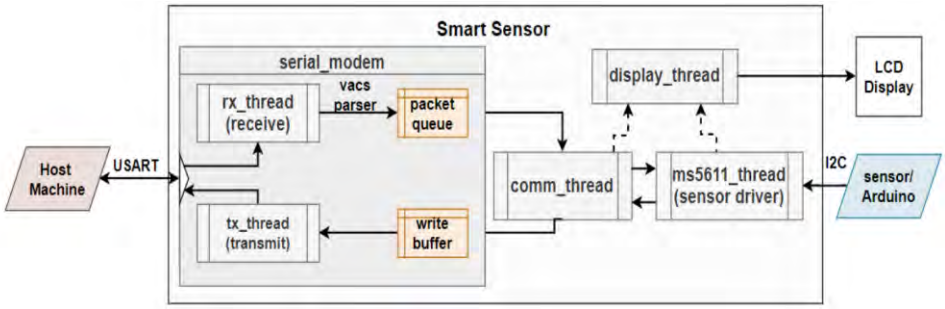

Figure 16. Schematic of the smart sensor system provided in [31].

Table 2. Relevant hardware for assessment for the VCU smart sensor.

Component ID	Function	Output Type
MS4525DO	Absolute or differential pressure and temperature sensor.	Digital I2C data
MP3H6115A	Static pressure sensor (no software).	Analog voltage level
MP3V5004DP	Dynamic pressure sensor (no software).	Analog voltage level
ADC (STM32)	Analog to digital converter for microcontroller input channels.	Digital data
STM32FM407	General purpose micro, calculates pressure, temperature, and Kalman-filtered pressure.	Digital UART data

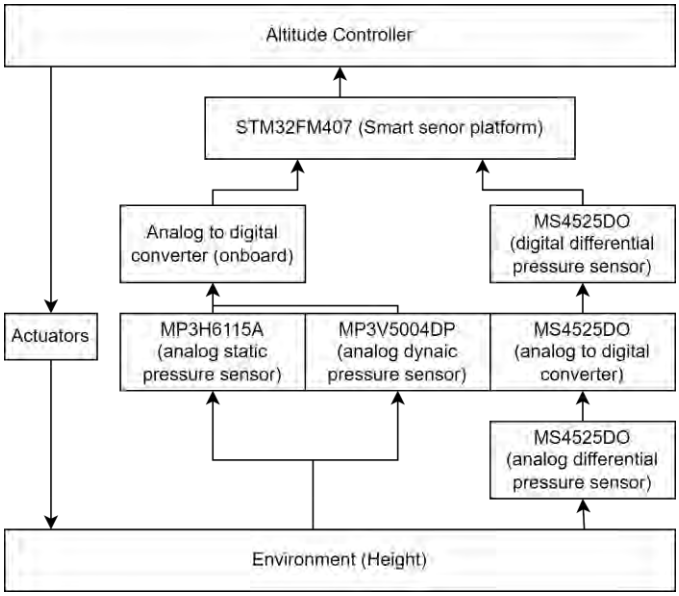

Figure 17. Approximate control block diagram for the VCU smart sensor and interaction with controller.

Note the STM32F4, while conventionally (and commercially) referred to as a "microcontroller," does not control a physical process but only augments information. The output types of digital components listed in the table follow standard I2C and UART convention. The "digital data" referenced represents a binary square wave format that is read by the STM32F4 (with custom protocol). Analog data is represented as a floating voltage level between zero and the max output voltage. The first three sensors measure the various pressure and temperature readings required by the system. The MS4525DO outputs digital data via I2C communication protocol to the microcontroller. The MS4525DO is comprised of an analog pressure sensing filament connected to a digital sensor head. Voltage values received from the filament are converted into digital logic which is transmitted via I2C protocol to the main controller board. The MP3H6115A and MP3V5004DP are analog pressure sensors and output analog voltage signals but are soldered on the same circuit board as the STM32FM407. They require the onboard ADC of the STM32F4 to convert the analog information to usable digital format. Unused components on the STM32DF4 are not listed. Importantly, the selection of relevant components included in the control block diagram is at the discretion of the safety analyst but should encompass all information devices of particular concern to the top event. Following STPA principals, the construction of the diagram should begin from high-level abstract function blocks broken down to lower-level component blocks. This iterative approach at scope refinement prevents inclusion of arbitrary components at the beginning of the assessment and can greatly assist in work reduction.

Step 2: Develop a FT consisting of hardware failures for a given top event of the system.

The VCU smart sensor's primary function is to provide readings for the adequate control of altitude [31]. Therefore, the assigned top event for this assessment is "autonomous drone crashes into ground due to failure of flight controller." The corresponding hardware devices relevant to this top event were identified in Table 2, and the hardware fault tree is shown visually in Figure 18. The fault tree is constructed starting from the top event which also corresponds to the top block in the control block diagram. From there, each block identified in the IFP is added iterative to the fault tree. For each component, a hardware failure branch and two empty branches associated with the software failures and the dependency failures are included. "Placeholders" are inserted into the tree where UCA/UIF software failure modes will be added in later steps. The process is repeated until the bottom of the control block

diagram is reached (i.e., “environment [height]”). From Table 2, the only digital components identified in the fault tree are the altitude controller, MS4525DO, ADC, and STM32FM407.

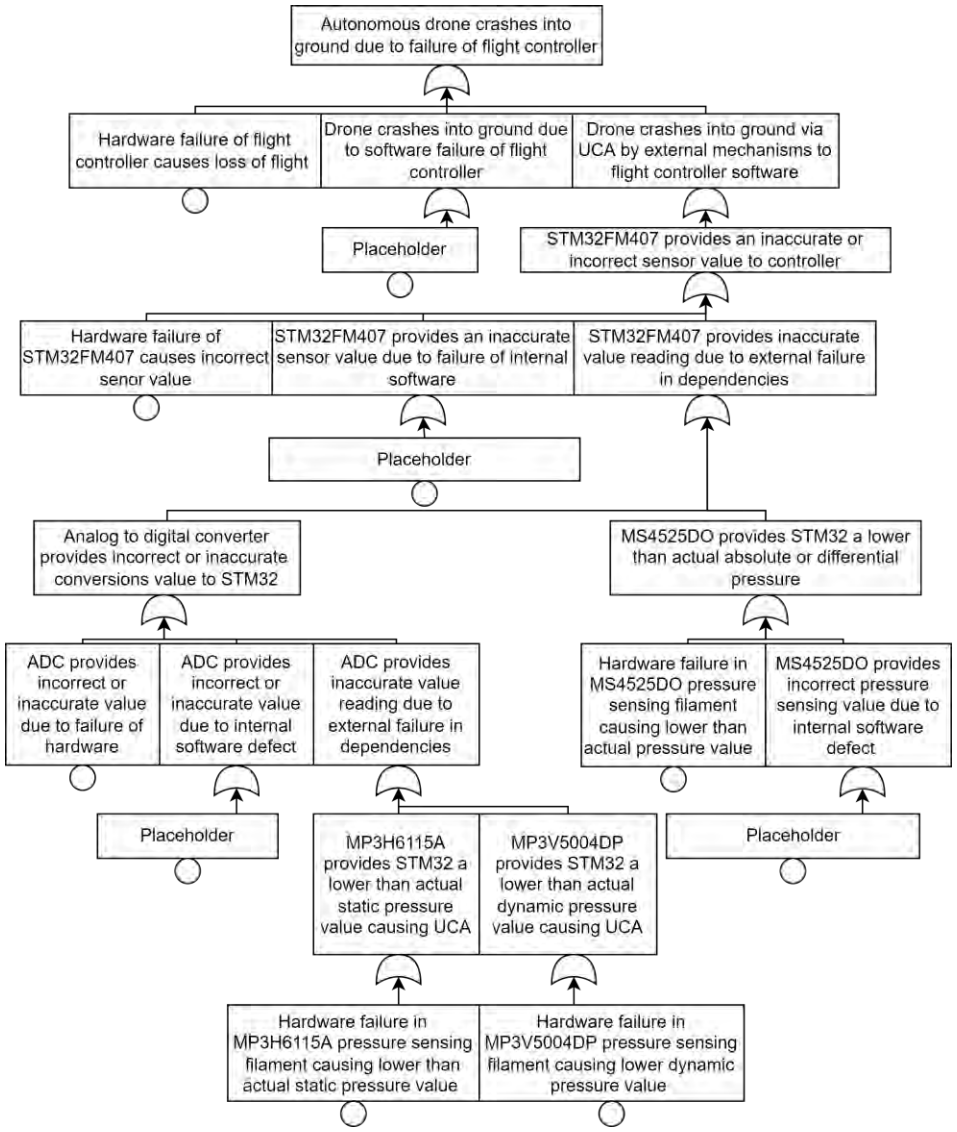

Figure 18. Hardware fault tree developed for the VCU smart sensor system.

Step 3: Determine UCAs/UIFs based on a redundancy-guided application of STPA.

In this step, all speculated UIFs and UCAs that can cause loss to the operator are listed. In this case study, only the subset of potential UCAs/UIFs that can lead to the chosen top event are identified. The potential UIFs are if any of the pressure sensor readings are reported incorrectly (UIF-D). For example, if the absolute pressure reported by the MS4525DO is lower than the true pressure, it would suggest that the UAV is at a higher altitude than it currently is (as atmospheric pressure decreases with altitude). A potential root cause for this problem is the incorrect assumption of the operating state of the MS4525DO. By default, the output parameter of the MS4525DO is differential pressure rather than absolute pressure. If the STM32FM407 assumes that the MS4525DO is reporting absolute pressure, the system may falsely report that the absolute pressure is always growing (or decreasing in altitude). This would ultimately inform the altitude controller incorrectly when conducting altitude adjustments. The direct impact is that the controller may “believe” that the UAV is at a sufficiently high altitude and fail to increase height

when near the ground. This also specifies the context of loss which is UIF failure near or approaching unacceptably low altitudes.

A failure to increase altitude when near the ground can also be a UCA of the flight controller (UCA-A). While the loss scenario is the same, the trigger of the UCA is an internal software defect of the controller algorithm or process model while the UIF is an internal software defect of the information device. The key differences are where the defect is located and which software component it ultimately affects. This is important when it comes to adding the UCAs/UIFs into the hardware fault tree. All identified UCAs/UIFs relevant to this top event are presented in Table 3. Note that there may exist additional UCAs/UIFs that do not correspond to the chosen top event. Instead, they should be added under a separate fault tree that corresponds with the speculated loss scenario.

Table 3. Identified UCA/UIFs relevant to chosen hardware top event.

UCA/UIF Type	**Impacted Component**	**Description**
UCA-A	Altitude controller	Controller does not increase altitude when UAV is close to the ground due to an internal software defect.
UIF-D	STM32F4M07	The information transmitted to the controller is value incorrect (too high or too low) due to an internal defect.
UIF-D	Onboard ADC	Conversion of analog values to consumers is value incorrect due to internal software defect.
UIF-D	MS4525DO	Information conversion to I2C and/or retrieval from sensing filament flawed due to internal software defect.

Step 4: Construct an integrated FT by adding applicable UCAs/UIFs as basic events.

The integrated FT, seen in Figure 19, can be constructed by including the UCAs/UIFs in Table 3 into the hardware fault tree constructed in Figure 18 under their corresponding hardware counterparts. Accordingly, only controllers can have UCAs, and only intermediate processors can have UIFs. If a component is identified to be capable of both UCAs and UIFs, return to Step 1 and further refine the component into its immediate dependencies. The immediate qualitative information that can be drawn from the integrated fault tree is how the various UIF feedback failures in the sensory system can lead to the top event. In addition, it becomes traceable how an information software failure of lower components (i.e., MS4525DO) can lead to consecutive informational failures of upstream components (i.e., STM32FM407).

Step 5: Identify potential software CCFs based on common or duplicate UCA/UIFs within the FT.

The VCU smart sensor system is a simplistic monitoring system and does not have redundant control or information trains. The lack of redundancy precludes the system from having inter-train software CCFs as no duplicate UCAs/UIFs exist among redundant trains. Furthermore, the system lacks complex dependencies within the train for an intra-train software CCF. Specifically, no one component can cause the failure of two or more systems. Therefore, no software CCFs were identified in this particular FT. This can be one of the qualitative conclusions deduced from the integrated fault tree.

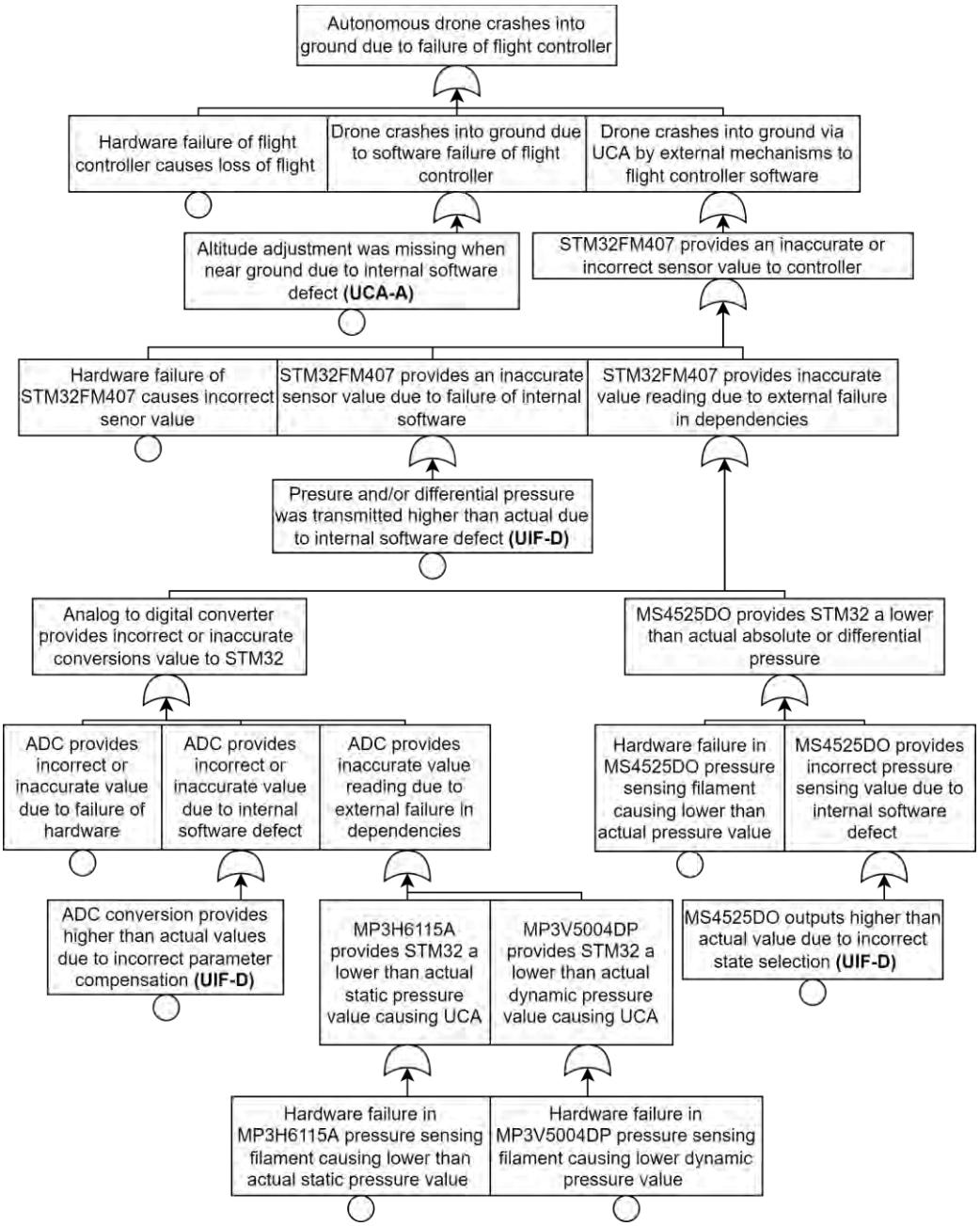

Figure 19. Integrated fault tree with UCAs/UIFs as software basic events.

Step 6: Solve the FT for the minimal cut sets to determine potential SPOFs in the design.

Due to the relative simplicity of the system, 10 minimal cut sets were identified containing only one basic event. For example, a minimal cut set could contain the single basic event "hardware failure in the MP3H6115A pressure sensing filament causing lower than actual static pressure value." The conclusion is that a failure of any component in the smart system would lead to the top event. While this may seem dire from a qualitative perspective, only through risk quantification of basic events can the true efficacy of a system be determined. For example, suppose each hardware and software component in the system has a safety integrity level of 1 (SIL 1) [23]. SIL 1 components are the lowest rated safety level and have a maximum continuous dangerous failure probability of 1E-5 per hour. Using this value for all basic events, the probability of the top event is calculated to be 1E-4 per hour or one failure every 10,000 hours.

This value matches well with the operational experience of the UAV which has over 10,000 hours of flight time without catastrophic failure.

Step 7: Identify and provide guidance to eliminate triggers of critical failures in the design including CCFs and SPOFs.

If the intention of the VCU smart sensor is to offer reliability like in nuclear reactor systems, one conclusion that can be drawn from the qualitative fault tree assessment is the need for additional device redundancy, specifically, for improved resiliency against single failures. One way to improve resiliency could be to change the software to treat the three onboard pressure sensors as redundant and diverse backups to each other rather than as independent sensors. For example, the MP3H6115A and MP3V5004DP both sense pressure in different ways and may be used as diverse protection against single software failures of the MS4525DO without introducing CCF into the system. Using the SIL 1 approximation from before, this would lower the overall anticipated top event probability to approximately 2E-5 per hour.

3.4.2 HSI Integrated Fault Tree Analysis

In this case study, only the QIAS-P HSI system and its divisional redundancies were considered. The steps of RESHA are abridged for brevity. Previous work by the author on this particular HSI system was conducted in [34], however, is further elaborated upon in this work. From documentation, the system consists of an array of PLCs. PLCs are general DI&C systems that can execute any program instructions. In this work, it is assumed that each PLC implements only one function. From [22], the following major functions were identified as observed in Table 4. Parameter calculations include those for the heated-junction thermocouple (HJTC), reactor vessel level (RVL), RCSM, CET, and margin until inadequate core cooling (ICC). These calculators read the plant state from sensory information, which is then used in the alarm system to alert the operator if an abnormal condition is detected. Note that all identified functions are considered individual components. Furthermore, each component is considered an intermediate processor as they lack control authority. The abridged control structure diagram can be seen in Figure 20.

The CAP resulting from operator manual trip and consisting of four manual trip undervoltage (UV) breakers is on the left-hand side. Tripping any of the four UV breakers will cause both redundant reactor trip switchgear systems to engage and trip the reactor. The only control action possible by the operator is thus activating (or not activating) a UV breaker. The IFP that leads to operator manual trip is on the right-hand side. It is the informational system that informs the operator of the current reactor state. Starting from the top is the operator display which is directly connected to both redundant divisions (A&B) of the monitoring system. Each redundant division consists of the major functional components identified in Table 4. The approximate pipeline is shown. Immediately before the reactor core are two sets of ADC for the HJTC and CET sensor arrays. These separate the analog and digital portions of the reactor. Everything above the HJTC and CET ADCs are digital, and everything below is analog. The HJTC sensors are located in the core, and the CET sensors are located near the upper plenum of the reactor. Neither HJTC nor CET are digital sensors due to the extreme irradiated environment.

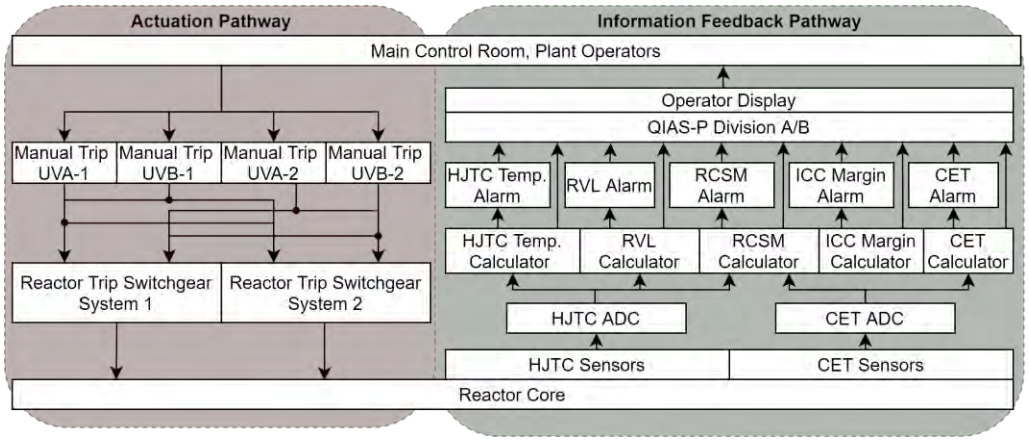

Figure 20. Control diagram of one redundant division of the HSI with manual reactor trip actuation pathway. Left side are control and actuation signals; right side are feedback signals.

Based on this control block diagram, the potential top event relevant to reactor trip safety is chosen to be "reactor fails to trip when needed causing core damage." The term "needed" here is used extremely loose, and the exact transient context should be specified whenever possible to prevent confusion. It is used here as an example to illustrate the methodology only. The hardware fault tree can thus be created from control block diagram (Figure 20), and hardware components can be identified (Table 4).

In this system, there exists only one controller and one control action possible related to reactor trip, whether the plant operator manually trips the reactor or not. The specific UCA relevant to the chosen top event is thus "operator does not engage undervoltage trip breakers (A1, A2, B1, or B2) under abnormal operational conditions (UCA-A) due to interface confusion." The cause of interface confusion exists within the QIAS-P subsystems. For instance, the "RCSM alarm does not alert the operator when the available coolant saturation margin decreases below the setpoint (UIF-A)." This could be due to an internal software defect, where the setpoint was incorrectly set, or an external issue, where one of the dependent sensors or intermediate processors has failed. The exact sequence of failures in the IFP depends solely on how the control loop is implemented.

Table 4. Major digital components identified within one division of the HSI system.

Component ID	**Function**	**Digital Output**
ICC Calculator	Uses HJTC temperature, RVL, RCSM, and CET to determine margin until ICC.	Margin until ICC
HJTC Temp. Calculator	Takes differential HJTC data and calculates to temperature.	Core temperature
RVL Calculator	Takes differential HJTC data, hot and cold leg, pressure, and vessel head to determine coolant level.	Coolant level
RCSM Calculator	Takes differential HJTC and absolute CET digital data and converts to coolant saturation margin.	Coolant saturation margin
CET Calculator	Takes raw CET digital data and calculates coolant core exit temperature.	Coolant exit temperature
ICC Alarm	Compares the available ICC margin with setpoint, triggers alarm for operator if exceeding.	Boolean/alarm to operator
HJTC Alarm	Compares the true core temperature with setpoint, triggers alarm for operator if exceeding.	Boolean/alarm to operator
RVL Alarm	Compares the available coolant level with setpoint, triggers alarm for operator if exceeding.	Boolean/alarm to operator

RCSM Alarm	Compares the available coolant saturation margin with setpoint, triggers alarm for operator if exceeding.	Boolean/alarm to operator
CET Alarm	Compares the true core exit temperate with setpoint, triggers alarm for operator if exceeding.	Boolean/alarm to operator
HJTC ADC	Converts the analog HJTC voltage readings from the core sensors to digital logic.	Raw core temperature
CET ADC	Converts the analog CET voltage readings from the upper plenum to digital logic.	Raw coolant exit temperature

The abridged integrated fault tree can be seen in Figure 21. It shows one redundant division of the monitoring system where the top event occurs when both the reactor state variables are potentially erroneous (QPD-PA-D1) and when alarms fail to alert the operator (QPD-PA-D2). The focus of each branch is the ICC calculator (QPD-PA-ICC) and alarm system (QPD-PA-ICA). Under the alarm software failure branch (QPD-PA-ICA-S), a UIF-A was identified where the alarm fails to trigger when core cooling is inadequate due to an internal software defect (QPD-PA-ICA-S-YA). Due to the similarity in division A and B of the monitoring system, this defect can also trigger an inter-division CCF (QPD-PA-ICA-S-CFA). For the ICC calculator, a similar deduction is reached where a single software failure is possible (QPD-PA-ICC-S-YD) which can also be an inter-division CCF (QPD-PA-ICC-S-CFD). However, no intra-division CCFs were identified for the ICC calculator and alarm system.

In the full fault tree, a total of 13 minimal software cut sets were discovered. All minimal cut sets contained a single software inter-division CCFs. Intra-division CCFs were also discovered but were not minimal cut sets. Three representative sets are listed in Table 5 as examples of each. In cut set #1, a latent internal software failure in the HJTC value calculator results in a nominal (but lower) than real state reading (UIF-D). This basic event can cause both an inter- and intra-division-level CCF. As both divisions A and B are assumed to be redundant and not diverse, a software defect would impact HJTC value calculators in both divisions. Furthermore, the output from the HJTC calculator is also used in the HJTC temperature alarm and margin for ICC calculator and alarm, potentially causing an intra-division CCF. Multiple dependent subsystems would thus reflect safer than true state due the single dependency on the HJTC value calculator. The top event is thus triggered as the operator is unaware the software has failed, and that the reactor is hotter than anticipated. In cut set #2, the UIF-A basic event is only an inter-division CCF due to the lack of diversity among divisions. In cut set #3, a software or hardware failure of the ADC would impact nearly all components in a division. Three separate calculators directly depend on the HJTC ADC, while six components indirectly depend on it. The system would thus experience a severe intra-division CCF should it fail in anyway. These results were not unexpected due to the assumption that divisions are redundant duplications of each other.

Table 5. Minimal software cut sets.

#	**Cut Set / Basic Event Description**
1	Division A&B HJTC calculators provide lower than real core temperature measurements under abnormal conditions (UIF-D).
2	Division A&B CET alarm does not trigger when coolant exit temperature exceeds acceptable margin (UIF-A).
3	Division A HJTC ADC provides lower than real digital core temperature conversion from analog signal (UIF-D) affecting HJTC, RVL, and RCSM calculator,

The primary causes of CCF in the system can be traced back to the lack of diversity by the two QIAS-P redundant divisions. From the documentation, there is sufficient validity to this argument as it is neither explicitly nor implicitly implied that design diversity is considered [22]. The justification for the original designers was that the IPS and DIS would act as diverse backups to the QIAS-P should a software CCF

be speculated. Unlike the QIAS-P system, which is PLC-based, the IPS is based on a distributed control system architecture and would offer diversity in both hardware and software. The IPS also monitors similar parameters to the QIAS-P system. In contrast, the DIS is a full analog counterpart to the QIAS-P system and is utilized whenever a software CCF is speculated by any system. It acts as the last information barrier in the system, however, has significantly reduced capabilities compared to its digital counterparts. For example, key safety parameters, such as the reactor coolant saturation margin, cannot be calculated with the DIS system. The preliminary conclusion is that while the proposed monitoring system has sufficient diverse and redundant backups to protect against both hardware and software single and CCFs events, the cost of implementation is significant. This type of monitoring system would most likely not be economically viable for novel small modular reactors with smaller footprints. Ultimately, without risk quantification of identified single and CCF software basic events, it is difficult to assess whether the level of protection offered is sufficient or excess. However, RESHA acts as a steppingstone towards quantification efforts by identifying targets and demonstrating causality between how and where DI&C systems can fail.

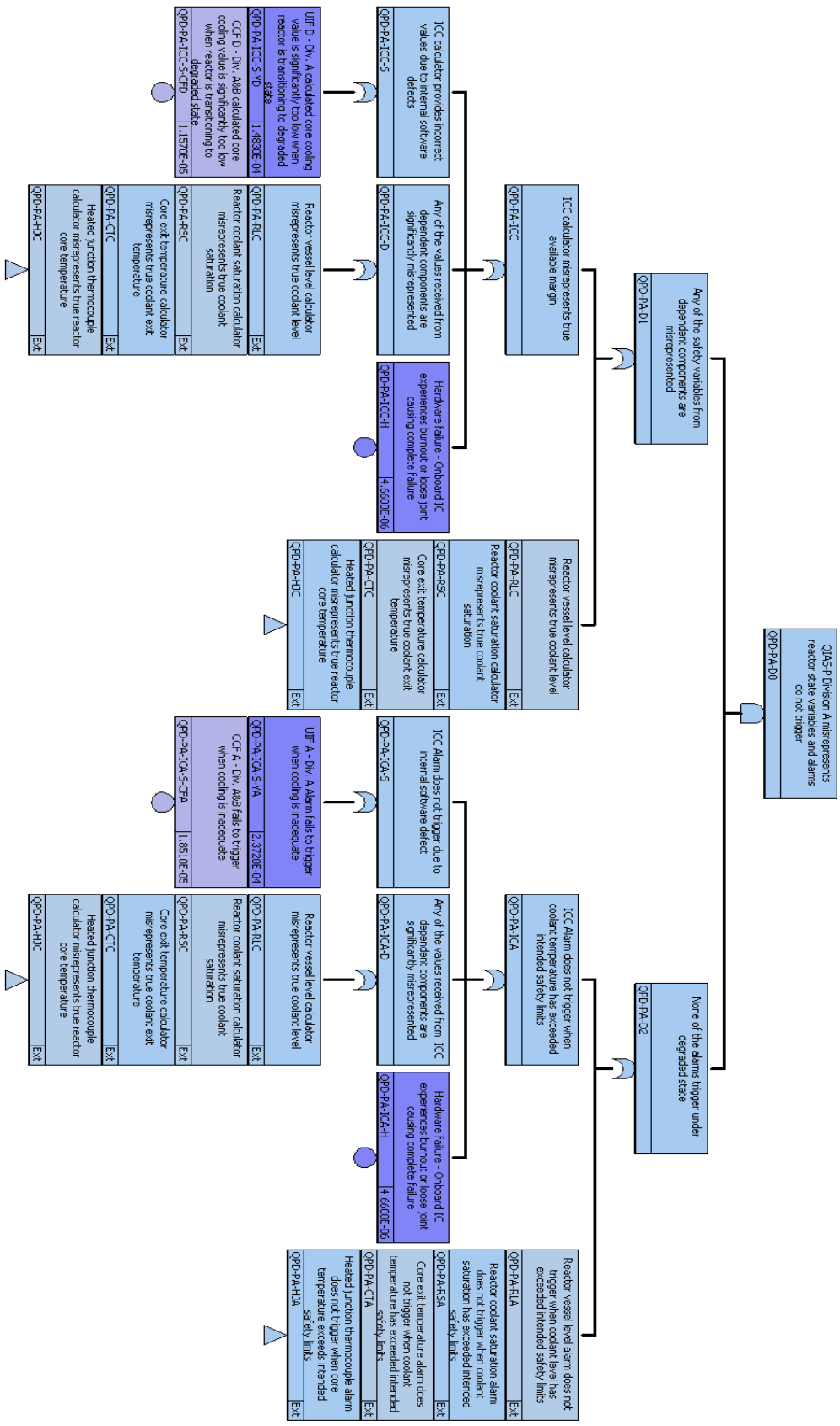

Figure 21. Abridged fault tree of the HSI system showing one of two redundant divisions.

3.4.3 RTS/ESFAS Integrated Fault Tree Analysis

The functional logic of RTS and ESFAS are shown in Figure 22 and Figure 23 (derived from [16] and [17]). Both two DI&C systems have multilevel redundancy designs, where different levels of CCFs may occur because of different coupling mechanisms. For instance, there are 16 logic processors (LPs) in the local coincidence logical (LCL) racks that receive trip signals from bistable processors (BPs) and transmit the signal to next digital modules. There are four divisions, each division has two LCL racks, and each rack has two LPs. The LPs within the LCL racks require at least one-out-of-two (1oo2) BP signals per division and at least 2oo4 divisions to transmit a reactor trip signal. Therefore, three levels of CCFs may happen in LPs: (1) malfunction of all 16 LPs, (2) malfunction of four LPs in one division, and (3) malfunction of two LPs in one LCL rack. Here it is assumed that these LPs each have identical function and share the same features except for the location of installation. CCFs are assumed to occur within a single location (e.g., division or rack) or across all divisions. Subsets of CCFs between divisions (e.g., A&B, B&C, and C&D) are not considered in RESHA because the main coupling factor in this case is the installation location. The final outputs from RESHA feed into following reliability analysis and provide component-level suggestions for system modifications (such as the elimination of CCFs that lead to potential SPOFs).

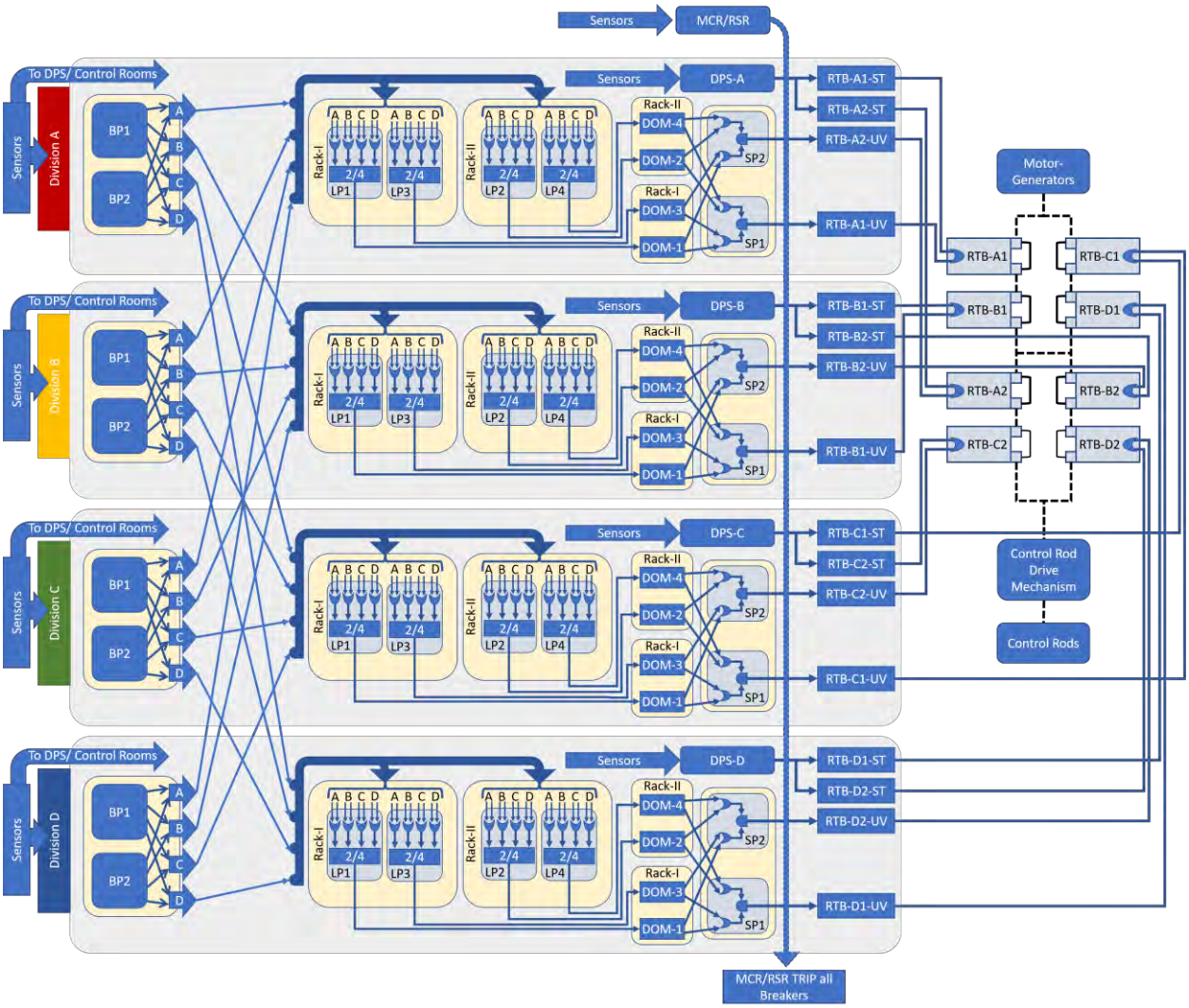

Figure 22. Functional logic of a representative four-division digital RTS (derived from [16]).

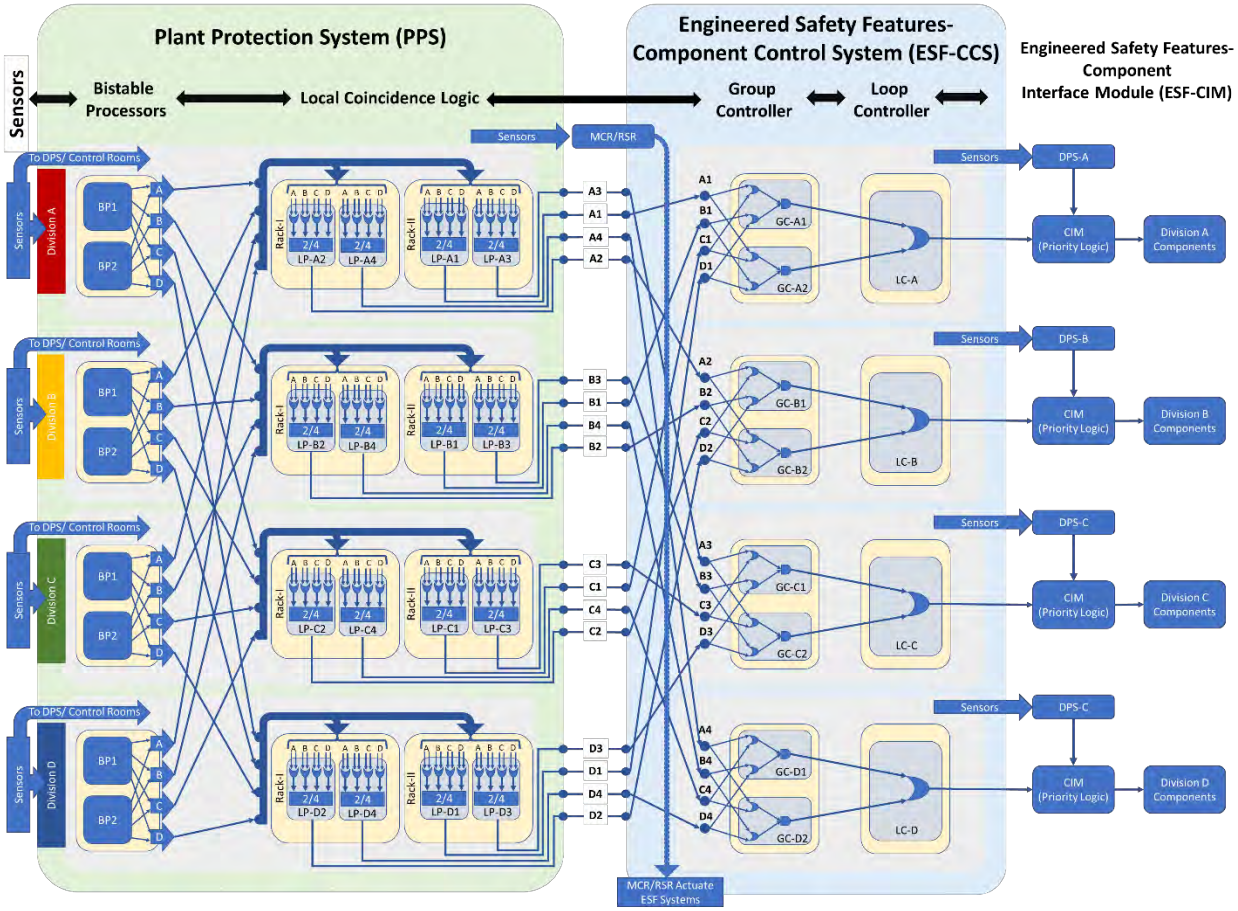

Figure 23. Functional logic of a representative four-division digital ESFAS (derived from [17]).

The Step 3 of the RESHA method determines UCAs based on a redundancy-guided application of STPA, which develops a redundancy-guided multilayer control structure for the highly redundant DI&C systems. Figure 24 illustrates a redundancy-guided multilayer control structure for a digital ESFAS (derived from [17]). The top layer of redundancy is the division-level redundancy; four independent divisions can be used to actuate ESF components (i.e., high-pressure injection [HPI] and low-pressure injection [LPI]). The functioning of each ESF component shown in Figure 23 is affected by a respective division. In each ESFAS division, two independent LCL racks receive and transmit actuation signals, which are the second layer of redundancy (i.e., unit-level redundancy). The third level of redundancy, the module level redundancy, consists of two LPs in each LCL rack. Therefore, three levels of CCFs in LP malfunction can be captured by RESHA: (1) LP CCF in all division, (2) LP CCF in one single division, and (3) LP CCF in one single LCL rack. Similarly, two levels of BP CCFs and group controller (GC) CCFs and one level of loop-controller (LC) CCF can be identified in the four-division digital ESFAS-FT.

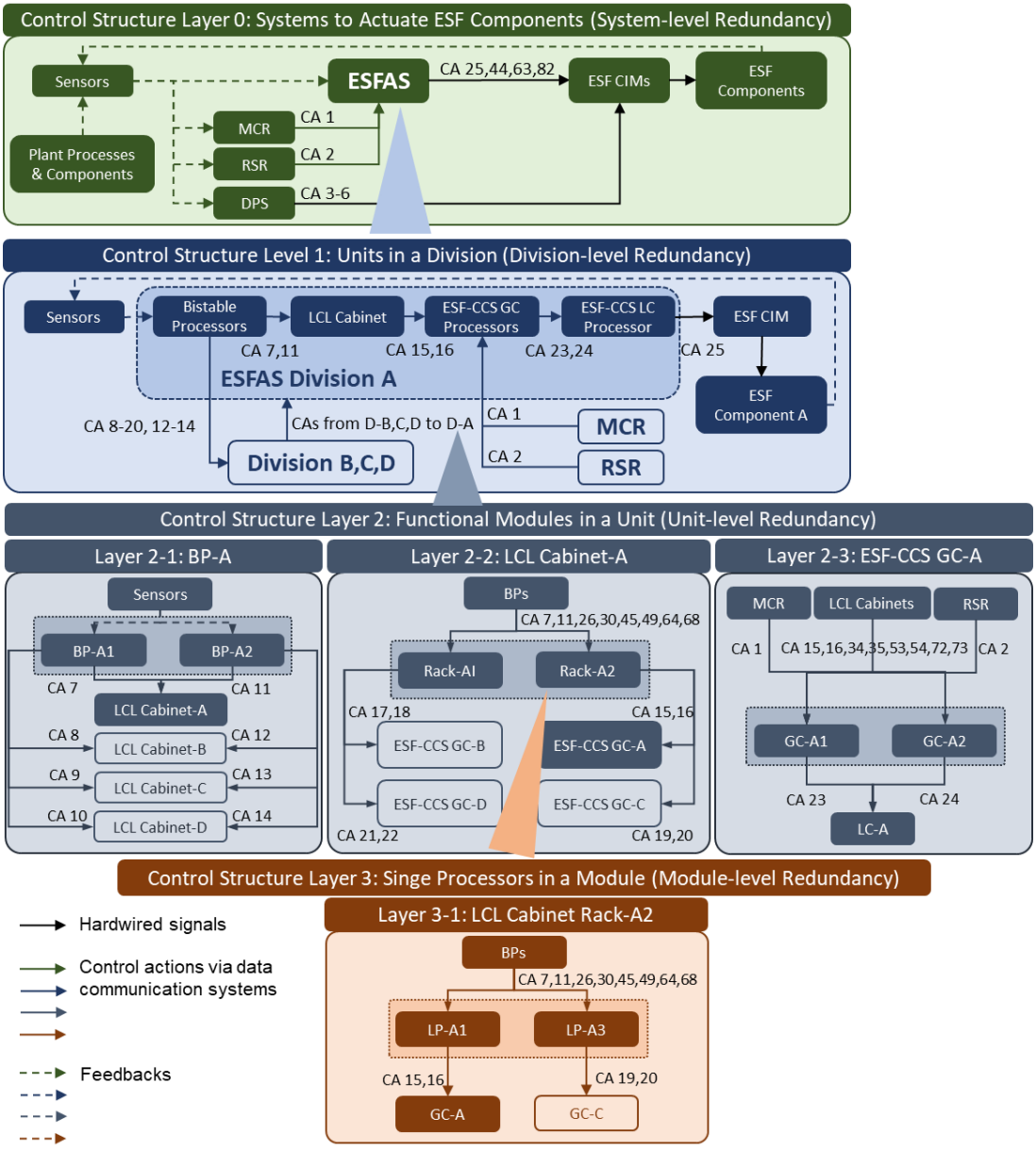

Figure 24. Redundancy-guided multilayer control structure for a digital ESFAS (derived from [17]).

The final outputs from RESHA feed into following reliability analysis and component-level risk evaluation that provide a guidance for system modifications (e.g., elimination of SPOFs, enhancement of reliability of specific components, or diversity in designs). In a word, the RESHA provided a means to identify CCFs in digital-based Type II interactions and software of highly redundant HSSSR DI&C systems by fully considering redundancy into the hazard analysis process.

4. MULTISCALE QUANTITATIVE RELIABILITY ANALYSIS

This section documents the methodology development and demonstration of a multiscale quantitative reliability analysis approach of the proposed framework for DI&C risk assessment.

The goal of the multiscale quantitative reliability analysis is to estimate the DI&C system reliability by calculating the integrated FT of DI&C systems obtained in the hazard analysis, then provide inputs for following consequence analysis. For the reliability analysis of large-scale DI&C systems, the quantitative small-scale reliability analysis of software and Type II interactions in DI&C systems are also included in the reliability analysis workflow.

Section 4.1 overviews the approach. Section 4.2 describes the technical backgrounds for the development of ORCAS and BAHAMAS. Sections 4.3 and 4.4 respectively provide the technical details of ORCAS, a method for software reliability analysis when testing data is available and sufficient, and BAHAMAS, which was developed for software reliability analysis in data-limited conditions. Section 4.5 introduces the CCF modeling method applied for both hardware and software failures in safety-critical DI&C systems.

4.1 Overview

Figure 25 illustrates the framework of the multiscale reliability analysis of DI&C system. The first step to any good reliability analysis for a DI&C system is the adequate collection and evaluation of design and requirement documentation. The required target documents, based on IEEE-guided software development lifecycle, include but are not limited to the software requirement specifications (SRS), the software design description (SDD), and the software test documentation (STD). These documents are necessary to determine if design and test failure data are available to conduct detailed and highly relevant reliability analysis.

The SRS document highlights the specific stakeholder requirements for the design and lays out the functional and non-functional requirements and use cases. The SRS document can be thought of as a high-level design. Completeness and consistency of the requirement specifications are subject to the experience and policy compliance of the development team and guide the rest of the project.

The SDD document is the detailed design of the SRS document and provides explicit guidance on software and development procedures to achieve goals outlined in the SRS. Architectural and schematical diagrams are common aspects of the SDD but may also include interface, data, and procedural designs specifying how each functional and non-functional requirement is addressed.

Lastly, the STD is a set of test documents that outline the design and test implementation plan as well as test results for the different aspects of the system. For example, STD could include the master test plan and report. The master test plan provides the overall test planning strategy and management for the different levels of tests to be implemented. The master test report summarizes the results of all tests accomplished including assessment on the quality of tests and testing efforts, potentially anomalies detected by the test strategies, and whether the final designed system meets all specified functional and non-functional requirements.

The overall adequacy of the system design is dependent on the experience of the team and may lead to data-rich and sparse scenarios. It is recognized that for many engineering software projects, from both experienced and inexperienced teams, the lack of adequate design documentation is prevalent potentially due to constraints on the project. A solution is provided for each case in terms of the ORCAS or BAHAMAS methodology.

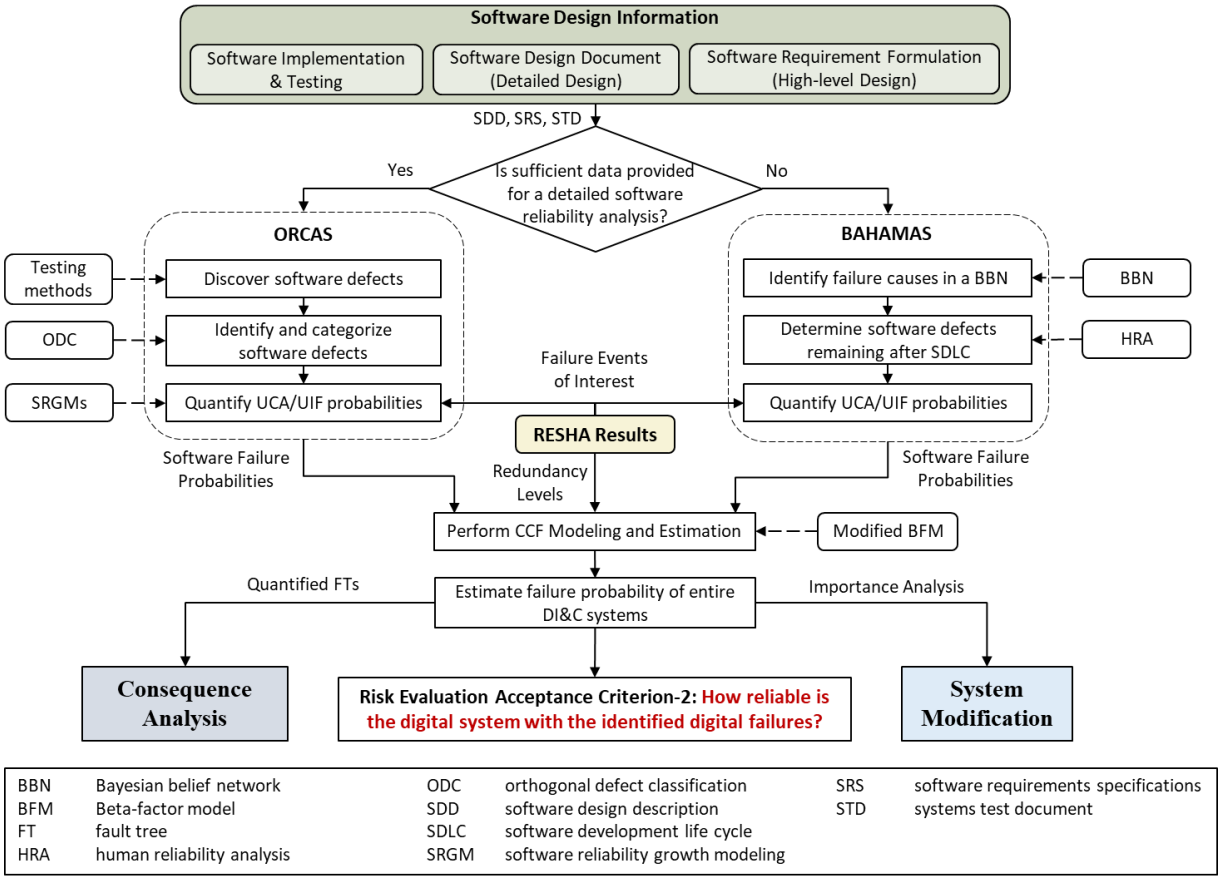

Figure 25. The workflow of multiscale quantitative reliability analysis.

The next step is to estimate the failure probability of UCAs/UIFs identified in RESHA. Two methods have been developed for different application conditions: BAHAMAS [18] for limited-data conditions and ORCAS for data-rich analysis.

BAHAMAS was developed for the conditions with limited testing/operational data or for reliability estimations of software in early development stage. It can provide a rough estimation of failure probabilities to support the design of software and target DI&C systems even when data is very limited. Instead of relying on testing data, BAHAMAS assumes software failures can be traced to human errors in the software development life cycle (SDLC) and modeled with human reliability analysis (HRA). In BAHAMAS, a Bayesian belief network (BBN) is developed to provide a means of combining disparate causal factors and fault sources in the system, and HRA is applied to quantify the potential root human errors during SDLC.

In contrast to BAHAMAS, ORCAS applies a white-box software invasive testing and modeling strategy to trace and identify the software defects that can potentially lead to software failures. The approach is a root-cause analysis methodology focused on comprehensive testing and follows the orthogonal-defect classification (ODC) approach to determine testing sufficiency. ODC is also used to systematically classify the identified defects into specific software causality groups, thereby linking defects to potential software failure modes. The failure data collected from testing strategies is also combined with software reliability growth models and linear reliability models to quantify the software failure probability of specific UCAs.

After the small-scale reliability analysis of software and Type II interactions, a modified beta-factor model (BFM) is applied for the modeling and estimation of CCFs, especially for software CCFs. Finally,

when the basic events of integrated FT are calculated, the failure probability of entire DI&C system can be estimated using FTA tools.

4.2 Theoretic Background for Software Reliability Analysis

While many new methods have been proposed to identify potential software events, these methods are focused on the qualitative identification of failure modes in a fault tree with very little guidance on direct quantification. Typically, software failure modes are identified by potential UCA made by the system [9]. Identified UCAs can lead to stakeholder losses and are integrated as basic events within fault trees [35]. Supplemental assessments to HAZCADS, such as the DRAM [36], have been used to address the risk of UCA by assigning risk reduction targets (RRTs) and control methods (CMs) to bound the risk of software basic events. The risk to the system is determined by the SIL [23] intended for the device. However, RRTs and CMs are qualitative methods at risk mitigation. Specifically, the DRAM helps identify failure mechanisms and pathways that can lead to UCAs and methods to address them. CMs listed in DRAM are scored qualitatively based on implemented type and effectiveness by expert experience and belief. Although DRAM has already provided a useful qualitative support identification framework for design activities, introducing software reliability analysis methods will be helpful to quantify the improvement in reliability by CMs and RRTs.

Aside from bounding estimate methods, other more direct risk and reliability quantification methods include software reliability growth models (SRGMs) [37]. These well-known methods attempt to predict the anticipated reliability of the software through historical failure data and has historically good generalization across multiple industries. However, conventional use of SRGMs is to measure the wholistic reliability of software rather than reliability of specific subsystems (due to the lack of failure data). In cases where failure data is limited, which is especially true for safety-critical systems, the uncertainty in the SRGMs can render predictions meaningless [38]. Our research thus aims to provide more acceptable risk and reliability information on DI&C systems without losing specificity or generalization capability. Furthermore, we attempt to collect qualitative evidence to support reliability conclusions.

4.2.1 Software Failure Probability

The subject of software failure probability is highly contentious among industry and regulatory bodies. To date, there does not exist a widely used industry standard or policy that accepts any type of software quantification methodology as a means for risk assessment. The reasoning is that software failures are viewed as systematic failures rather than random failures. In [36], software failures are defined as systematic as they can be replicated with perfect accuracy if the exact conditions of failure are known. This is different than hardware failures as material and manufacturing imperfections result in highly unpredictable failure rates even if the same conditions are supplied. However, the assumption of systematic software failure relies on the fact that the exact conditions of failure are known. In many complex DI&C systems, it is rare to know in advance the exact input conditions to trigger a failure; therefore, software will still fail regardless of the magnitude of the development effort. Rather, many failures are discovered during operation under a unique combination of conditions. These conditions can arise spontaneously and unpredictably due to a range of complex factors such as cosmic rays, weather, human action, sequence/combination, or time. These are known as initiating events and are typically treated as random occurrence. In this work, while it is acknowledged that software fails systematically in the sense it can be replicated, software failure is treated to be probabilistic due to unpredictably of initiating events that can trigger them. This distinction is critical as it guides how ORCAS and BAHAMAS quantify software failure probability.

4.2.2 Software Failure Classification

The most important theory being that “failures” in software are ill-defined. In one definition from IEC 61508-4, a failure is defined as the “termination of the ability of a function unit to provide a required

function or operation … in any way other than as required" [23]. This includes both total failures, where the software is completely inoperable (i.e., blue screen of death [BSoD]), and partial failures, where the software is still operable but behaves in a way that results in a loss to operators. While protections against total failures exist, partial failures are more difficult to address as they are unintended actions by software due to incomplete or inadequate requirement specifications. In STPA, the authors attempt to define partial software failures as "misbehaviors" of the system that are deliberate and are hazardous but still conform to specified constraints and requirements [9]. In this respect, the software never truly "fails" but rather performs actions that are undesirable. Furthermore, it is incredibly difficult to develop comprehensive constraints and requirements for software such that no "misbehaviors" can ever exist due to the complexity and size of the entire system. Misbehaviors typically manifest when the assumptions on requirements of one piece of software do not align with the assumptions of another piece of software (i.e., integration of heterogeneous DI&C systems). These misbehaviors are known in STPA work as UCAs. An example of a UCA is the automated emergency braking system of a car unexpectedly triggers the emergency brake while it is traveling along a highway causing a rear-end collision (UCA-B). Note that UCAs are only applicable to digital controllers of physical processes and cannot be applied to information systems. Instead, failures of these types of system are represented by UIFs. UIFs mimic UCAs in failure categories but identify failures in the information and feedback pathways of DI&C systems that lead to losses. An example of a UIF is a fire alarm fails to turn on when a smoke is detected causing excessive structural damage (UIF-A).

In this work, a similar ideology is utilized, where the different categories of UCAs/UIFs are referred to as relevant software failure modes. However, UCAs/UIFs are qualitative failure modes of software systems that do not have identifiable root causes. For instance, if a software is missing a conditional statement, it is unclear and difficult to trace exactly which UCA/UIF will occur. Therefore, part of this work is to extend UCAs/UIFs by assessing detected defects in the software and translating them into each UCA/UIF failure mode. In the ORCAS methodology, this relationship between defects and UCAs/UIFs is implemented with orthogonal-defect classification (ODC) [39].

In ODC, discovered defects are assumed to be the root causes of software failure modes. However, the exact cause-and-effect relationship between root causes and failure mode is not clear unless explicit cases can be identified. Here ODC acts as a semantic classification tool to sort defects into generalized classes and linking each class to a UCA/UIF via a weight factor. Specifically, it is hypothesized that defects with shared characteristics cause the same UCA/UIF failure modes. For example, if a conditional defect is detected, it suggests inadequacy in data or condition verification in the source code. This can then be linked to a higher UCA/UIF probability. Note that while a detected defect can only be assigned one defect class, a defect class can cause multiple UCAs/UIFs. The exact relationship strength is discussed in the methods section. In brief, the defect groups are (1) function, (2) assignment, (3) algorithm, (4) checking, (5) interface, (6) relationship, and (7) timing [39].A summary of each class can be found in Table 6.

Table 6. Independent defect classes identified by the ODC methodology [39].

Defect Class	**Defect Class Description**
Algorithm	Problems related to the efficiency or correctness of the algorithm affecting the task or function that can be resolved by (re)implementing an algorithm or local data structure without a request for formal design change. This can include issues with procedure, templates, or overloaded functions.
Assignment	Value(s) assigned incorrectly or not assigned at all including initialization of parameters, instances, or objects.
Checking	Errors caused by missing or incorrect validation of parameters or data in conditional statements including loops and branches.
Function	The resolution of the defect requires a formal design change as it affects significant capability, end-user interfaces, product interfaces, and interfaces with structure(s) and can include real or abstract entities.

Interface	Communication problems between modules, components, drivers, objects, functions, etc. via errors with macros, call statements, control blocks, or parameter lists.
Relationship	Problems related to the association among procedures, data structures, and objects including inheritance relationships, hierarchy, or inappropriate data structures instantiated by other data structures.
Timing	Necessary serialization of shared resources was missing, the wrong resource was serialized, or the wrong serialization technique was employed.

These groups are assumed to be independent and mutually exclusive from each other and cover all known to date potential defects in the software.

Another application of ODC is the identification of the necessary environmental and input conditions required to uncover or detect defects. These conditions are known as triggers and can also be used to assess when all conditions have been considered. From a conventional perspective, testing conditions as triggers are difficult to measure and compare with each other as tests are not equivalent. Here, ODC can also be used to semantically categorize triggers that are needed for comprehensive software testing. In brief, the groups for implementation triggers are (1a) simple and (1b) complex path; (2a) function coverage, (2b) variation, (2c) sequence, and (2d) interaction; (3) volume/stress; (4) recovery; (5) configuration; (6) startup/restart; and (7) normal mode [39]. A brief description of each trigger group can be found in Table 7.

Table 7. Defect trigger classes identified by the ODC methodology for comprehensive testing [39].

Trigger Group	**Trigger Description**
Simple Path	The test case was motivated by the knowledge of specific branches in the code and not by the external knowledge of the functionality. Specific branch features are tested to verify proper execution.
Complex Path	In white/gray box testing, the test case that found the defect was executing some contrived combinations of code paths. In other words, the tester attempted to invoke execution of several branches under several different conditions.
Function Coverage	During black box testing, the test case that found the defect was a straightforward attempt to exercise code for a single function, using no parameters or a single set of parameters.
Function Variation	During black box testing, the test case that found the defect was a straightforward attempt to exercise code for a single function but using a variety of inputs and parameters. These might include invalid parameters, extreme values, boundary conditions, and combinations of parameters.
Function Sequence	During black box testing, the test case that found the defect executed multiple functions in a very specific sequence. This trigger is only chosen when each function executes successfully when run independently but fails in this specific sequence. It may also be possible to execute a different sequence successfully.
Function Interaction	During black box testing, the test case that found the defect initiated an interaction among two or more bodies of code. This trigger is only chosen when each function executes successfully when run independently but fails in this specific combination. The interaction was more involved than a simple serial sequence of the executions.
Volume /Stress	The system is operating at or near some resource limit, either upper or lower. These resource limits can be created by means of a variety of mechanisms, including running small or large loads, running a few or many products at a time, and letting the system run for an extended period of time.
Recovery	The system is being tested with the intent of invoking an exception handler or some type of recovery code. The defect would not have surfaced if some earlier exception had not caused exception or recovery processing to be invoked. From a field perspective, this trigger would be selected if the defect is in the system's or product's ability to recover from a failure, not the failure itself.

Configuration	The system is being tested to ensure functions execute correctly under specific available software configurations.
Startup /Restart	The system or subsystem was being initialized or restarted following some earlier shutdown or complete system or subsystem failure.
Normal Mode	The product is operating well within resource limits, and the defect surfaced while attempting to execute a system test scenario. This trigger would be used when the scenarios could not be run because there are basic problems which prevent their execution. This trigger must not be used in customer reported defects.

These trigger groups are extensive and cover most relevant scenarios, but it is not a complete list. For instance, they do not explicitly consider cybersecurity vulnerabilities as a condition for defects. Nonetheless, from a development perspective, the trigger groups represent an adequate scope for required testing conditions and have been used extensively by the industry [40]. It has been experimentally shown that consideration of all defect triggers can detect more latent defects than an unstructured testing methodology [40]. Furthermore, it can also be used to verify when sufficient testing has been considered in various size software projects and at different stages of the software development lifecycle [40].

4.3 ORCAS for Software Reliability Analysis in Data-rich Conditions

In this section, a detailed method to quantify software hazards (e.g., UCAs/UIFs) identified with the RESHA method is presented and demonstrated within a case study. The method presumes sufficient operational and testing data are readily available to the risk analysts. This method, referred as ORCAS, attempts to address how specific software defects relate to identified UCAs/UIFs and provides specific software failure quantification.

The basis for ORCAS is to use pseudo-exhaustive test-based approaches [33] to generate a historical failure database. While it is not strictly required, the pseudo-exhaustive methodologies are shown to experimentally capture the vast majority of hidden defects, can partially automate the testing process, and provide qualitative evidence of comprehensive testing. Defects that are detected and removed are then classified based on ODC theory. Each defect class is then modeled with reliability growth models to quantify probabilities of different software failure modes. Qualitative evidence collected throughout this process is then used to gauge how complete and confident we are in the assessment. The overall workflow of the method (and meaningful extensions) can be seen in Figure 26. Items in the dashed box are all elements pertaining to the ORCAS methodologies. In general, the outputs of ORCAS are the software failure mode probabilities and confidence in the assessment. The qualitative evidence derived from ORCAS include the requirement traceability matrix (RTM), trigger coverage assessment (TCA), structural path coverage, and reliability modeling stability. The quantitative evidence includes the defect reports used to determine failure probability.

4.3.1 Stages of ORCAS

It is important to note that the assessment of software reliability should be continuously evolving with the implementation and design of the system. For instance, the target and scope of the analysis may not be fully known at the start of the assessment due to inherent complexity of the software system. Relevant items that are missed in the first rounds of assessment will require returning to the prior stage for further refinement. For example, when a defect is detected and classified in stage 3, it is expected to be repaired before the software is deployed. This will require returning to stage 2 for defect removal activities. Imperfect knowledge and discovery at any individual stage suggests that revisiting a prior stage is anticipated.

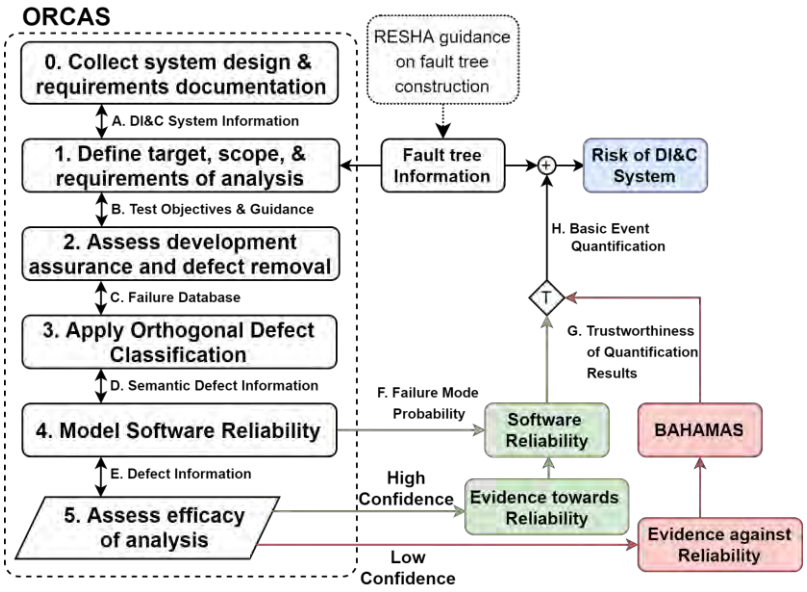

Figure 26. Overall workflow of ORCAS.

Stage 0: Collection of system design and software requirements specification documentation.

In stage 0, the relevant information and details pertaining to the system are collected. This step is assumed to occur in any assessment and thus not described in detail. The information that can be collected at this stage can include formal documentation (i.e., IEEE 29148, IEEE 829, and IEC 61508), defect and anomaly reports, design, and requirements specifications. Exact documents are not specified; however, information pertaining to the functional and non-functional requirements, implementation design, digital and hardware architecture, and test verification and validation are useful in further stages. This information is used to guide construction of objectives and relevant targets of the assessment.

Stage 1: Defining target, scope, requirements, and organization of the analysis.

In stage 1, the scope and testing adequacy requirements are defined. Here the desired modules and functions of the system are outlined, as well as the hierarchal structure and the type of tests anticipated to be completed. As software can be incredibly complex, initially assigning the scope to the entire system can be overwhelming and uninformative. Rather, it is advised that a fault tree approach is adopted to assess exactly what the stakeholders are concerned about (via top events) and how it may impact operational goals. The fault tree provides organizational structure but also linear causal relationships between identified failures to loss events. RESHA is used to first deconstruct the DI&C system into a control block diagram. From the diagram, the hardware fault tree is then constructed from identified modules of concern. Lastly, potential software failure modes, in the form of UCAs and UIFs, are then added to the hardware fault tree to form the qualitative integrated fault tree. The basic events identified in the fault tree are quantified in later stages of ORCAS.

Stage 1.1: Decomposition and construction of the hardware fault tree based on RESHA. In RESHA, modules in the control systems are initially separated based on the physical separation of hardware components and devices. Each component is then broken down into its function and major signal interactions to the level of refinement desired. The decomposition of the control system follows a top-down STPA approach rather than a bottom-up approach. For clarity, a bottom-up approach relies on first identifying all components included in the system followed by identifying how they interact with each other to construct the control block diagram. In comparison, a top-down approach first identifies the user-facing components and iteratively identifies the immediate components below the current control layer

(i.e., from the control panel to the sensors). This can also help constrain the size of the tree to basic failure events relevant to the stakeholders.

As a simple example of the top-down process, a mock PLCs is used. Most industrial applications are implemented with PLCs due to the relative ease of reprogramming, the generalizability of the hardware, and the capability at combining more than one critical function on a single platform. The PLC system is first represented in a control block diagram with its major input and outputs listed. From there, the system can be further decomposed into its immediate physically separate subcomponents or into abstract representative function modules. An analog to digital converter integrated circuit is an example of a physical subcomponent that can be modeled in the control block diagram. The abstract modules, in contrast, do not need identifiable dedicated hardware but must implement an identifiable function with clear input and output relationships. The process is repeated until the desired level of refinement is achieved. The control block diagram is then converted into a fault tree based on its hierarchal structure following the input-output relationship identified.

Once the relevant aspects for analysis are identified, the target software and corresponding modules can be assessed for testing completeness. Specifically, both white-box and black box testing approaches such as T-way combinatorial testing [41], modified condition decision coverage (MCDC) [42], boundary value analysis (BVA) [43], and equivalence partitioning (EP) [43] are used to verify functional correctness of the software module. These recommended methodologies have been proven to be experimentally effective at defect removal and can be easily automated for the testing process. For instance, the National Institute of Standards and Technology (NIST) developed a free automated test generation tool, the Automated Combinatorial Testing for Software Tools (ACTS), for T-way combinatorial and sequential testing [44]. Within the ORCAS framework, the testing adequacy is gauged in a three-level test suite hierarchy, as shown in Figure 27. Each level specifies defect triggers that need to be considered during test implementation and activities that should be conducted to address adequate testing. Importantly, software testing is not conducted in this stage, rather a collection of the anticipated necessary tests is constructed for each relevant software module identified in the control block diagram.

Level 3 Testing (User)		
Testing Activities	**Defect Triggers Covered**	**Recommended Methods**
System Test	Software Configuration	T-way Parameter Testing Bounday Value Analysis Equivalence Partitioning
System Test	Workload/Stress Recovery/Exception Startup/Restart Normal Mode	Requirements Traceability Matrix

Level 2 Testing (Subsystem)		
Testing Activities	**Defect Triggers Covered**	**Recommended Methods**
Unit Test	Simple Path Complex Path	Modified Condition Decision Coverage
Function Test	Function Coverage Function Variation Function Squence Function Interaction	T-way Parameter Testing Boundary Value Asessment Equivalence Partitioning

Level 1 Testing (Component)		
Testing Activities	**Defect Triggers Covered**	**Recommended Methods**
Unit Test	Simple Path Complex Path	Modified Condition Decision Coverage
Function Test	Function Coverage Function Variation Function Squence Function Interaction	T-way Parameter Testing Boundary Value Asessment Equivalence Partitioning

Figure 27. Three-tier software testing requirements with recommended activities and methods.

Stage 1.2: Qualitative evidence requirements. The testing requirements are organized into separate sections, specifically the (1) RTM and (2) TCA. The RTM is used to check if the current project meets the anticipated project requirements and locates where in the source code or detailed documentation, the requirement is satisfied. In Table 8, a sample RTM is provided. The rows across the matrix (e.g., REQ1) are functional and non-functional requirements that are identified in high-level design documents. The first and second row determine the sum of direct and indirect tests that were conducted to satisfy each identified requirement. The rows following are the sections in the detailed design or source code that the test is located in. An "X" for direct and "x" for indirect will be placed to demonstrate that a test exists in the source code for that requirement. At this stage, the RTM will only contain the identified requirements and the various sections of the detailed design document or source code. All other cells should be empty and will be filled in the next stage during inspection.

Table 8. Sample RTM with four specified requirements and four detailed design sections.

Requirements Identifiers	**Reqs. Tested**	**REQ1**	**REQ2**	**REQ3**	**REQ4**
Test Cases					
Tested Implicitly					
1.1.0					
1.1.1					
1.2					

The TCA is similar to the RTM however checks if each trigger has an associated test. In each TCA table, the first row shows either the modules, subsystems, or systems of concern (depending on level). The first column includes the number of tests both explicit and implicit and the corresponding triggers that need to be covered for the corresponding level. Rows 2 and 3 are the sum of the tests that were performed and are recorded in the following stage. A mock system in Figure 28 is provided to demonstrate how different triggers should be tested at different hierarchal levels. The "System" is comprised of two PLCs and outputs variables "C," "D," and "E." Within the system, each "PLC" performs different functions and are comprised of three separate identified components of concern. In Table 9 and Table 10, the level 1 component and level 2 subsystem mock TCA is shown. In the level 1 TCA, all identified modules form the top row. The corresponding triggers for that level are then placed in column 1. In Table 10, each PLC is treated as a black box, and the internal components are ignored (as testing consideration should have been completed in the level 1 TCA). The triggers for component and subsystem are similar since the trigger coverage requirements are the same. Finally, in Table 11, the entire system is identified that interfaces with the user, and the corresponding system-level triggers are identified. Unlike the subsystem and component triggers, it is known that the overall complexity of the system makes detailed testing (i.e., functional coverage) computational intractable due to the potential number of states that grows with each hierarchical level.

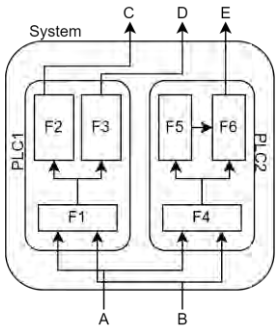

Figure 28. Mock DI&C control block diagram with two PLCs, each with three identified relevant modules.

Table 9. Level 1 component TCA.

Trigger Identifiers	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**
Test Cases						
Tested Implicitly						
Simple Path						
Complex Path						
Functional Coverage						
Functional Variation						
Functional Sequence						
Functional Interaction						

Table 10. Level 2 subsystem TCA.

Trigger Identifiers	**PLC 1**	**PLC 2**
Test Cases		
Tested Implicitly		
Simple Path		
Complex Path		
Functional Coverage		
Functional Variation		
Functional Sequence		
Functional Interaction		

Table 11. Level 3 user TCA.

Trigger Identifiers	**System**
Test Cases	
Tested Implicitly	
Software Configuration	
Workload/Stress	
Recovery/Exception	
Startup/Restart	
Normal Mode	

The output of stage 1 is the RTM and the various component, subsystem, and system-level TCAs. These tables inform the assessor which tests should exist for comprehensive testing consideration. The tables are not guaranteed to eliminate all hidden defects, however, have been shown to be experimentally effective at identifying the vast majority of developmental errors. The integrated fault tree may also be an output of this stage if a structured tree organization is preferred to trace potential losses and failure scenarios.

Stage 2: Assess software development assurances and defect removal activities.

In stage 2, the testing requirements identified in the previous stage are compared to the actual testing efforts conducted by the development team. For activities involving T-way combinatorial testing, BVA, and EP, the specific parameter range, variation, edges cases, etc. identified in the previous stage must have a traceable test(s) to verify the activity is complete. For MCDC path analysis, tests should exist that consider different path conditions to achieve near unity coverage. For the RTM, both functional and non-functional requirements of the software should be traced to the associated test cases that demonstrate adequate conformance. In Table 12, a sample of a filled RTM is presented with arbitrary requirements and associated code sections.

Table 13 and Table 14, a sample of a filled TCA is presented for both level 1 component and level 2 user testing. The completeness of the RTM and TCA are used as qualitative evidence to justify software reliability. These metrics also serve to identify areas requiring further testing. For instance, if function variation was not considered during testing, the associated tests can be implemented to satisfy this trigger. Ultimately, the objective of testing is to collect defect data that can later be used to quantify software failure modes. The defects can be collected from two sources. The first source includes existing defect reports discovered during the development process. The second source is through additional testing, which is initiated due to inadequacies in the RTM, TCA, or structural coverage.

Table 12. Sample RTM with four specified requirements and four detailed design sections.

Requirements Identifiers	**Reqs. Tested**	**REQ1**	**REQ2**	**REQ3**	**REQ4**
Test Cases	5	1	2	0	1
Tested Implicitly	4	1	0	2	2
1.1	4	X	X	X	X
1.2	1			X	
1.3	2	X			X
1.4	2		X		X

Table 13. Level 1 component TCA

Trigger Identifiers	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**
Test Cases	8	10	3	11	10	7
Tested Implicitly	5	0	10	5	9	7
Simple Path	4	1	1	4	3	2
Complex Path	1	3	3	3	1	2
Functional Coverage	3	2	2	4	5	2
Functional Variation	3	2	1	2	6	3
Functional Sequence	1	1	2	2	3	3
Functional Interaction	1	1	4	1	1	2

Table 14. Level 3 user TCA

Trigger Identifiers	**System**
Test Cases	37
Tested Implicitly	5
Software Configuration	9
Workload/Stress	1
Recovery/Exception	27
Startup/Restart	3
Normal Mode	2

Stage 3: Apply orthogonal-defect classification to collected data.

In stage 3, the defect reports are collected and categorized based on ODC theory [45]. Specifically, the defects fall under one class specified in Table 6. An analysis of defects involves understanding what went wrong and how it was resolved. Furthermore, defects are classified based on shared characteristics of the resolution or solution and not what failure occurred. For example, if a software failure is detected through testing and traced back to a missing conditional statement, then the resolution would be to add the conditional statement into the source code. This type of code repair is considered a checking defect. It should be noted that if widely different solutions exist for the same problem, it may be an indication of inadequate requirements and constraints specification for the problem.

After the classification of defects, the defect reports are assigned to specific software failure modes based on data-driven causality relationships. The relationship between defect classes and software failure modes is based on quantitative data collected from various open-source Github repositories namely, the MongoDB, Cassandra, Apache HBase, Zephyr, and OpenPilot repositories. The first three are all NoSQL database management systems; data can be found at [46]. Zephyr [47] is a scalable small-footprint real-time operating system while OpenPilot [48] is a semi-autonomous driving system.

For each defect report in the above databases, the defect class are determined first, followed by how it impacted the functional and non-functional usability of the software. The impact was generalized into the four UCA/UIF software failure modes. The contribution by each class to specific UCA/UIF was then counted and used to determine conditional probabilities, for example, $P(\text{UCA-A}|\text{Function defect})$. In this work, 605 defect reports were used to form the conditional probabilities. In Table 15, the various conditional probabilities with UCA/UIF can be seen establishing the relationship between defects and software failure mode. For example, suppose an assignment defect was detected in the software. Based on relationship data, it has a 57.6% chance to cause a UCA/UIF-B.

Table 15. Conditional probabilities of the causality between UCA/UIFs and orthogonal defect classes.

	UCA/UIF-A	**UCA/UIF-B**	**UCA/UIF-C**	**UCA/UIF-D**
Algorithm Defects	0.349	0.130	0.318	0.203
Assignment Defects	0.287	0.576	0.060	0.018
Checking Defects	0.300	0.266	0.237	0.198
Function Defects	0.392	0.246	0.237	0.125
Interface Defects	0.334	0.478	0.122	0.066
Timing Defects	0.218	0.036	0.622	0.123

The correlation derived in Table 15 is hypothesized to exist due to the emergence of the relationship in all data sets. In Figure 29, algorithm defects across all data bases are assessed separately. In all data sets, percent groupings were discovered between algorithm defects and the UCA/UIF caused in the software. For example, it was found that classified algorithm defects had a 0.38, 0.27, 0.33, 0.33, and 0.41 chance to cause UCA/UIF-A software failure modes across the MongoDB, HBase, Cassandra, OpenPilot and Zephyr datasets, respectively. The average of these values is approximately 0.349. A similar relationship was discovered for all other defects. In Figure 30, the relationship pattern is seen for assignment defects as well. The uncertainty quantification of the correlation table is conducted in the following section. In Table 16, the sample size for each defect class from each data set is shown.

Table 16. Sample size of each software defect class and data set

	MongoDB	**Cassandra**	**HBase**	**OpenPilot**	**Zephyr**	**Total**
Algorithm Defects	26	27	47	12	12	124
Assignment Defects	28	16	22	11	11	88
Checking Defects	37	32	36	14	22	141
Function Defects	17	20	28	18	13	96
Interface Defects	18	20	37	16	28	119
Timing Defects	11	3	7	5	11	37
Total	137	118	177	76	97	605

The number of defects reported for each data set is not consistent and should not be expected to be consistent as each development process will have different errors emerge. However, this impacts the confidence in the defect class to UCA/UIF relationship. For instance, only 37 samples were used to construct the timing defect conditional probabilities. This may be interpreted as too small of a sample size to adequately gauge the true relationship.

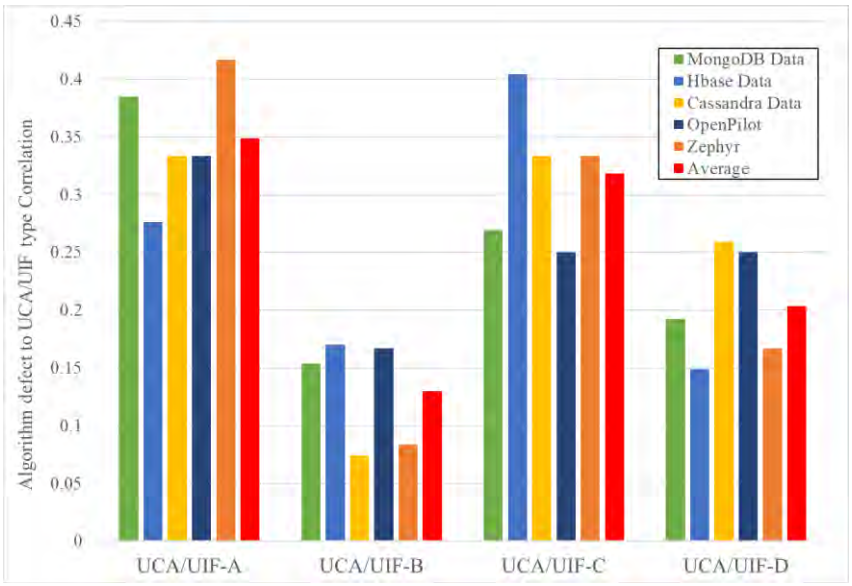

Figure 29. Correlation between reported algorithm defects in data sets and UCA/UIF failure modes.

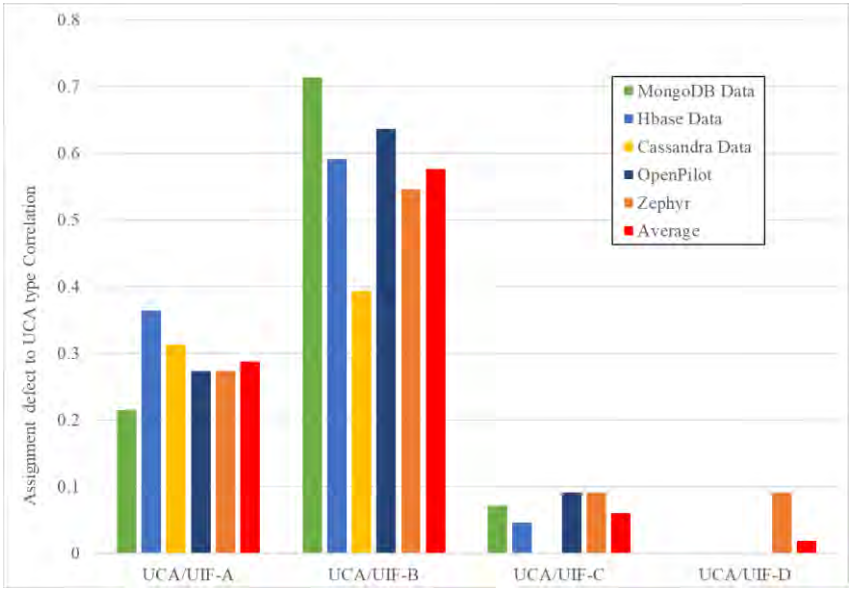

Figure 30. Correlation between reported assignment defects in data sets and UCA/UIF failure modes.

Note that the relationship between defects and UCAs is the same as that with UIFs. The justification for this is that defects cause similar failures regardless of the actual function of the software due to the generalization of shared characteristics. Function in this sense can be the control of a physical process or the control of information. A defect in the control of a physical process vs. the control of information is indistinguishable at the source code level. For example, an erroneous conditional statement cannot be used to determine the overall functionality of the software. Furthermore, all defective conditional statements share similar characteristics (i.e., incorrect comparison value and missing conditional statement) regardless of if it is used for control or for information processing.

Stage 4: Model software reliability based on Historical Defect Data.

In stage 4 of ORCAS, the defects reports collected in the previous stage are modeled with reliability growth models. When modeling with growth models, there are two conditions that must be satisfied (1) sufficiency of failure data and (2) stability of predictions. Condition 1 is to ensure that the model developed by growth models are useful and accurate. If there are insufficient data point, the growth models are highly uncertain and can lead to misleading results. This is possible if the development history is unavailable to the assessor either because it is proprietary or if the system is sufficient simple such that the number of defects recorded are limited. In [49], this latter problem is discussed in detail where testing was conducted on a mature DI&C system. The authors were not able to find enough defects in the testing process to justify usage of growth models. Specifically, the uncertainty of the model was significant and resulted in highly uncertain predictions. In such instances, it is advisable to use the BAHAMAS method to complete steps 4 and onwards. Note that this alternative pathway is represented in Figure 26 as the red pathway. Otherwise, following ORCAS to completion is the green pathway.

The exact mathematics behind reliability growth models is not discussed in detail as extensive literature already exists. In essence, growth models attempt to predict the number of defects that may exist in the software after a certain number of testing hours. The defects that are detected in the software are repaired hence increasing the overall reliability of the software over testing time. In Figure 31, an example of defects recorded over time is presented with the HBase software database. The number of combined testing days conducted over the development period of HBase is on the x-axis. Combined testing days are the number of days of testing multiplied by the number of developers. The cumulative number of defects over testing time is on the y-axis. At the end of the development cycle, pre-release, the number of defects recorded over time flattens (i.e., between 12500–17500 combined testing hours). This is an example of how reliability increases over time as testing increases.

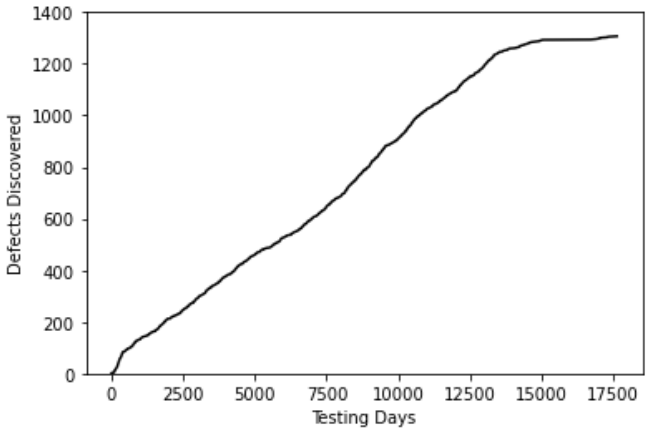

Figure 31. Recorded cumulative number of software defects over combined testing time in days.

In stage 3, the defects recorded were separated into defect classes. This can also be plotted over testing days and is seen on the left of Figure 32. The right figure is the same figure but with the y-axis limited to 200 defects to see how the other defect classes accumulate with testing time. Note that the sum of all defect classes is equal to the total number of defects. For additional details on ODC class specific SRGMs, the author recommends reference [45].

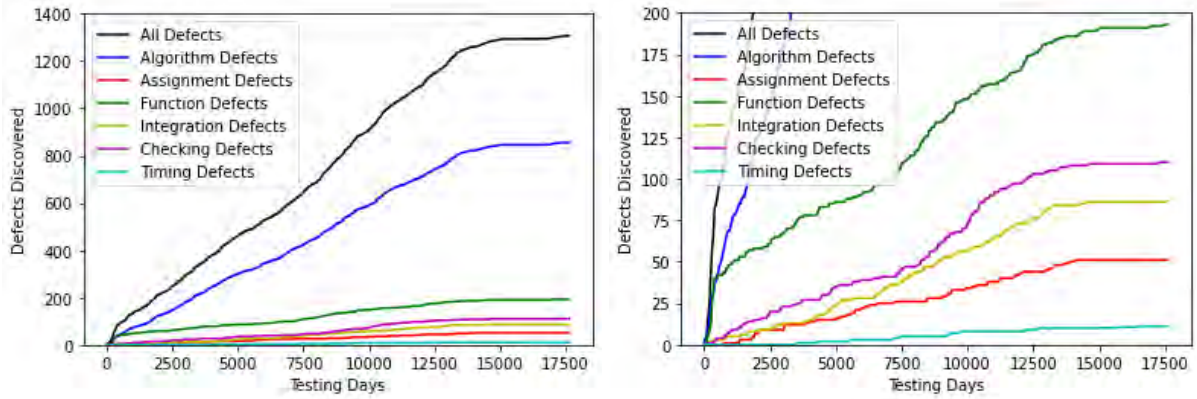

Figure 32. Left: Recorded cumulative number of defects by class over combined testing time in days. Right: Cumulative defects by class with y-axis limited to 200.

Given the historical defect data above, different types of reliability growth models can be applied. There is no one model that is applicable (and accurate) for all types of software due to the variability of the development environment. In ORCAS, five different SRGM models are evaluated each with slightly different parameter configurations. In Table 17, the different SRGM models (with references) are shown.

Table 17. Different types of software reliability growth models to model defect history.

Model Name	**Model Type**	**M(t)**	**Reference**	**Comments**
Goel-Okumoto (GO)	Concave	$a(1-e^{-bt})$	[50]	Exponential decay model.
GO S-Shaped (GOS)	S-Shaped	$a(1-(1+bt)e^{-bt}$	[51]	Modification of GO model for S-Shape.
Weibull	Concave	$a(1-e^{-bt^c})$	[52]	Accounts for increasing test time required for each defect.
Yamada Raleigh (YR)	S-Shaped	$a(1 - \exp(-r\alpha(1 - \exp(-\frac{\beta t^2}{2}))$	[53]	Accounts for testing effort.
Log Poisson (LogP)	Infinite Failure	$1/c \ln(c\alpha t + 1)$	[52]	Failure rate never approaches 0.

Each model is fitted to the cumulative defect data of all defect classes and can be seen in Figure 33. In the legend of Figure 33, the name of the model and the coefficient of determination (r^2) are shown. Note that the GO, Weibull, and LogP models overlap with similar predictions.

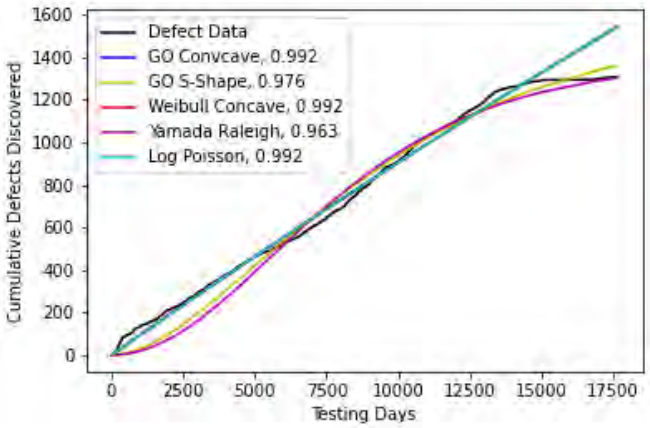

Figure 33. Various SRGM fitted to the cumulatively discovered defect data of HBase software.

The SRGMs can also be applied to each of the defect class curves seen in Figure 32. The GO S-shaped curve is used to demonstrate and seen in Figure 34 applied to all defect classes.

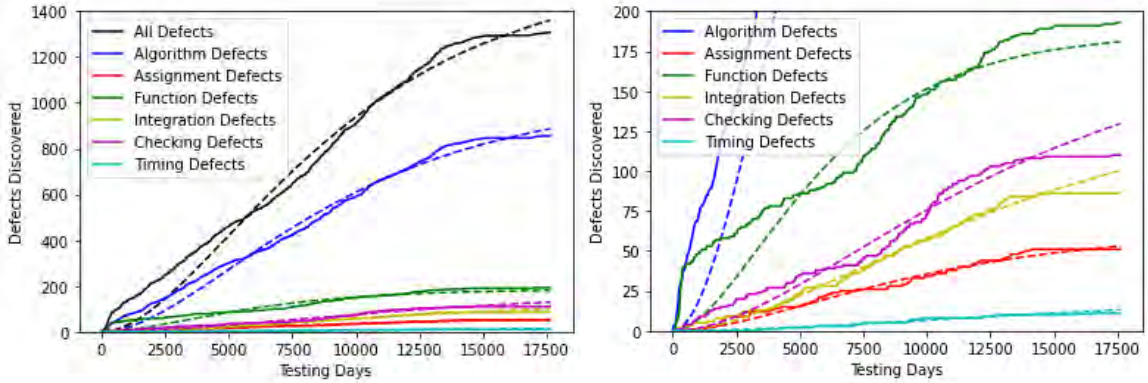

Figure 34. GO S-shaped SRGM fitted to defect data, models seen as dashed lines.

After a SRGM is fitted to the historical data, the failure probability of a specific class can be determined using the following equation

$$F_i(t|s) = 1 - exp\big(M_i(s) - M_i(s+t)\big) \tag{2}$$

where s is the last recorded defect time, $F_i(t|s)$ is the probability of failure over t duration, and i is the selected defect class. For this data set, t is assigned as 1/24 days, assuming exactly 24 hours in 1 day. This produces a failure probability in per hour units per defect class.

Finally, the probability of specific UCA/UIF classes can be determined by multiplying the conditional probabilities of each class with the failure probability per hour per class probability, as shown in Equation (3). Recall that the conditional probability was pre-determined through the historical defect data of multiple different pieces of software. Only the second term, $F_i(t|s)$ needs to be determined by the users and is determined from the testing efforts conducted.

$$P\big(\text{UCA/UIF}_j\big) = \sum_{i=\text{defect class}} P\big(\text{UCA}_j|i\big)F_i(t|s) \tag{3}$$

A sample calculation is provided below for the probability of a UIF-A occurring in the HBase software over the next hour. The defect class names are truncated.

$$P(\mathrm{UCA_A}) = \begin{bmatrix} P(\mathrm{UCA}_A|\mathrm{Alg.}) \\ P(\mathrm{UCA}_A|\mathrm{Asi.}) \\ P(\mathrm{UCA}_A|\mathrm{Fnc.}) \\ P(\mathrm{UCA}_A|\mathrm{Int.}) \\ P(\mathrm{UCA}_A|\mathrm{Chk.}) \\ P(\mathrm{UCA}_A|\mathrm{Tim.}) \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} F_{Alg}(0.0417|s) \\ F_{Asi}(0.0417|s) \\ F_{Fnc}(0.0417|s) \\ F_{Int}(0.0417|s) \\ F_{Chk}(0.0417|s) \\ F_{Tim}(0.0417|s) \end{bmatrix} = \begin{bmatrix} 0.349 \\ 0.287 \\ 0.300 \\ 0.392 \\ 0.334 \\ 0.218 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} 8.73\mathrm{e}-4 \\ 5.79\mathrm{e}-5 \\ 5.66\mathrm{e}-5 \\ 1.73\mathrm{e}-4 \\ 2.19\mathrm{e}-4 \\ 3.04\mathrm{e}-5 \end{bmatrix} = 4.86\text{e-}04 \text{ per hour} \quad (4)$$

Stability is used as a qualitative attribute to gauge usability of growth models. Generally, in stable growth models, the total number of predicted defects should not vary significantly from period to period (i.e., week to week). If variations are large, the resulting predictions also have a large variation thus rendering the model useless. In [49], this was the issue for their safety-critical DI&C system as there was insufficient failure data for convergence. A 10% maximum periodic variation allowance is recommended and was shown to be effective in [54] as a qualitative metric for model stability. This value also informs developers when sufficient failure data has been collected, and testing can be stopped.

Stage 5: Efficacy of the assessment.

The last stage of ORCAS is the qualification of the software development process. Recall that the qualitative information derived from the method includes the RTM, the TCA, the structural path coverage, and the stability of the reliability modeling. The developers and users can assess the completeness of the testing effort by reviewing how complete each qualitative factor is and which areas need further refinement. For instance, the RTM informs the developers whether each requirement was tested, while the TCA informs the developers that every scenario considered. The developers can then return to those software sections and conduct further testing.

4.3.2 Assumptions and Discussions

This method has several assumptions and limitations. The most concerning limitation is the use of reliability growth modeling in highly critical systems. In [49], insufficient software failure data was cited as one reason why this method is infeasible. When operational failure data is limited, the model is oversensitive, and the predictions have large uncertainties. While this may not be the case for all software, the lack of failure data is a highly relevant and limiting scenario. ORCAS only partially addresses this issue. When testing completeness is sufficient, but failure data is insufficient (i.e., all triggers are considered but no defects were discovered), ORCAS defers to a different methodology, which is known as BAHAMAS [18] that will be introduced in next section. The qualitative evidence derived from ORCAS can also be used to support BAHAMAS. This is seen in Figure 26, where low assessment confidence leads to BAHAMAS.

The second major assumption is that causality between defect classes and UCAs/UIFs do not differ significantly between different types of software. In this work, while this assumption held for several different types of assessed software (i.e., database management, embedded operating system, and vehicle control), further verification of this relationship is required for all software. While ODC suggests that software defect classes are process and development agnostic, more evidence is required to justify this claim.

4.3.3 Uncertainty Quantification of ORCAS

In this section, the sources of uncertainty are identified and quantified. The primary sources of uncertainty are the model uncertainty and parametric uncertainty associated with the reliability growth models and the data uncertainty associated with the UCA/UIF conditional probability table.

4.3.3.1 Uncertainty Quantification of the UCA/UIF Conditional Probability Table

In Table 15, the relationship between UCAs/UIFs and defect classes was presented. These values were averaged from graphs such as those seen in Figure 29 and Figure 30. It is assumed in this work, that the UCA/UIF relationship holds true for all types of software and is normally distributed around the

average relationship value. In the following tables (Table 18, Table 19, Table 20, Table 21, Table 22, and Table 23), the relationship strength between UCA/UIF and defect class is shown for all datasets. In columns 2–6, the name of the data set is provided along with the relationship strength discovered for each UCA/UIF failure mode. In columns 7 and 8, the row average is presented along with the two standard deviation confidence intervals. The last row presents the number of samples from each dataset.

Table 18. Algorithm defect class relationship across all software data sets with uncertainty.

Algorithm Defect Class							
UCA/UIF	MongoDB	Cassandra	HBase	OpenPilot	Zephyr	Avg.	2σ
A	0.384	0.333	0.276	0.333	0.417	0.349	0.108
B	0.153	0.074	0.170	0.167	0.083	0.130	0.094
C	0.269	0.333	0.404	0.250	0.333	0.318	0.122
D	0.192	0.259	0.148	0.250	0.167	0.203	0.099
Sample Size	26	27	47	12	12		

Table 19. Assignment defect class relationship across all software data sets with uncertainty.

assignment Defect Class							
UCA/UIF	MongoDB	Cassandra	HBase	OpenPilot	Zephyr	Avg.	2σ
A	0.214	0.313	0.364	0.273	0.273	0.287	0.111
B	0.714	0.393	0.591	0.636	0.545	0.576	0.240
C	0.071	0.000	0.045	0.091	0.091	0.060	0.077
D	0.000	0.000	0.000	0.000	0.091	0.018	0.081
Sample Size	28	16	22	11	11		

Table 20. Function defect class relationship across all software data sets with uncertainty.

Function Defect Class							
UCA/UIF	MongoDB	Cassandra	HBase	OpenPilot	Zephyr	Avg.	2σ
A	0.353	0.400	0.357	0.389	0.462	0.392	0.087
B	0.294	0.250	0.179	0.278	0.231	0.246	0.090
C	0.294	0.150	0.286	0.222	0.231	0.237	0.116
D	0.059	0.200	0.179	0.111	0.077	0.125	0.124
Sample Size	17	20	28	18	13		

Table 21. Interface defect class relationship across all software data sets with uncertainty.

Interface Defect Class							
UCA/UIF	MongoDB	Cassandra	HBase	OpenPilot	Zephyr	Avg.	2σ
A	0.389	0.350	0.324	0.250	0.357	0.334	0.105
B	0.389	0.550	0.595	0.500	0.357	0.478	0.204
C	0.111	0.050	0.081	0.188	0.179	0.122	0.120
D	0.111	0.050	0.000	0.063	0.107	0.066	0.091
Sample Size	18	20	37	16	28		

Table 22. Checking defect class relationship across all software data sets with uncertainty.

Checking Defect Class							
UCA/UIF	MongoDB	Cassandra	HBase	OpenPilot	Zephyr	Avg.	2σ
A	0.405	0.313	0.222	0.286	0.273	0.300	0.135
B	0.270	0.250	0.250	0.286	0.273	0.266	0.031
C	0.189	0.219	0.333	0.214	0.227	0.237	0.112
D	0.135	0.219	0.194	0.214	0.227	0.198	0.074

Sample Size	37	32	36	14	22		

Table 23. Timing defect class relationship across all software data sets with uncertainty.

Timing Defect Class							
UCA/UIF	MongoDB	Cassandra	HBase	OpenPilot	Zephyr	Avg.	2σ
A	0.091	0.333	0.286	0.200	0.182	0.218	0.189
B	0.091	0.000	0.000	0.000	0.091	0.036	0.100
C	0.636	0.667	0.571	0.600	0.636	0.622	0.074
D	0.182	0.000	0.143	0.200	0.091	0.123	0.161
Sample Size	11	3	7	5	11		

The UCA/UIF conditional table with uncertainty data can be produced and seen in Table 24. For some relationships, the uncertainty is relatively low (i.e., UCA/UIF-B and checking defects). However, for other defect relationships, the error is significant (i.e., 100% to 185% of the average value) and is most prominent in the timing defect relationship class. These values are expected as the number of software systems used to determine standard deviation is small. As more software systems are assessed, it is anticipated that this table will be useful at correlating defect classes and UCA/UIF failure modes.

Table 24. UCA/UIF conditional probability table with two standard deviation uncertainty.

	UCA/UIF-A	**UCA/UIF-B**	**UCA/UIF-C**	**UCA/UIF-D**
Algorithm	0.349 ± 0.108	0.130 ± 0.094	0.318 ± 0.122	0.203 ± 0.099
Assignment	0.287 ± 0.111	0.576 ± 0.240	0.060 ± 0.038	0.018 ± 0.041
Checking	0.300 ± 0.135	0.266 ± 0.031	0.237 ± 0.112	0.198 ± 0.074
Function	0.392 ± 0.087	0.246 ± 0.090	0.237 ± 0.116	0.125 ± 0.124
Interface	0.334 ± 0.105	0.478 ± 0.204	0.122 ± 0.120	0.066 ± 0.091
Timing	0.218 ± 0.189	0.036 ± 0.100	0.622 ± 0.074	0.123 ± 0.161

4.3.3.2 Model Uncertainty and Parametric Uncertainty

In this section, the model and parametric uncertainty equations are provided. Note that this source of uncertainty can only be quantified after stage 4 of ORCAS has been completed.

- **Parametric Uncertainty Quantification**

Parametric uncertainty is related to the parameters in each of the growth models. The approach adopted in ORCAS follows the least square error fitting approach to parameter estimation. The function to minimize is as follows,

$$R(q) = y(t) - M(t,q) \quad \forall \quad t \in \{t_0, t_1, \dots\ t_{N-1}\} \tag{5}$$

$$\min_{\mathrm{q}} S(q) = \min_{\mathrm{q}} \sqrt{\frac{R(q)^{\mathrm{T}} R(q)}{N}} \tag{6}$$

where y is the number of recorded defects over testing time, $M(t,q)$ is the number of predicted defects over time, N is the total number of recorded defects, q are the parameters to be fitted, and $S(q)$ is the objective function to minimize. The uncertainty of the parameters can be determined as follows

$$\sigma_R^2 = \frac{R^T R}{N-1} \tag{7}$$

$$X = \begin{bmatrix} \frac{dM(t_0)}{dq_0} & \cdots & \frac{dM(t_0)}{dq_p} \\ \cdots & \cdots & \cdots \\ \frac{dM(t_{N-1})}{dq_0} & \cdots & \frac{dM(t_{N-1})}{dq_p} \end{bmatrix} \tag{8}$$

$$V = \sigma_R^2 \times (X^{\mathrm{T}}X)^{-1} = \sigma_R^2 \times F^{-1} = \begin{bmatrix} \sigma_{q_o}^2 & \mathrm{cov}(\sigma_{q_0}, \sigma_{q_1}) & \cdots \\ \mathrm{cov}(\sigma_{q_1}, \sigma_{q_0}) & \cdots & \cdots \\ \cdots & \cdots & \sigma_{q_p}^2 \end{bmatrix} \tag{9}$$

where σ_R^2 is the residual variance, X is the sensitivity matrix, F is the Fisher information matrix, p are the number of parameters to be estimated, and V is the covariance matrix. The variance in the parameters are determined by the diagonal values in the covariance matrix.

- **Model Uncertainty Quantification**

In Table 17, the different reliability growth models are presented. After each parameter is fitted to the target data, the model uncertainty is the variation in the predication made by each model. If a normal distribution is assumed around the mean of the predictions made at each time step, then the model uncertainty can be determined at any arbitrary time step, t_m, where $0 \leq m \leq N$, using the following equations

$$E[M(t,q)] = \frac{1}{c}\sum_{i=1}^{c} M_i(t,q) \tag{10}$$

$$\sigma_M^2 = \frac{1}{c-1}\sum_{i=1}^{c} (M_i(t,q) - E[M(t,q)])^2 \tag{11}$$

where $E[M(t,q)]$ is the mean prediction, c is the number of reliability models included, and σ_M^2 is the prediction variance.

4.3.4 Case Study - VCU Smart Sensor System for Unmanned Aerial Vehicles

In this case study, the ORCAS was applied to an embedded smart sensor developed by VCU. The sensor is a barometric pressure and temperature sensing device that originates from the VCU Unmanned Aerial Vehicle (UAV) Laboratory [31]. The device consists of mature design and code, including the SRS and SDD documentation, with over 10,000 hours of tested flight time. The software is written in GNU11 C programming language for the application code and runs on top of the ChibiOS Version 17.6.4 Real-Time Operating System (RTOS) [31]. The software was tested extensively using a pseudo-exhaustive test-based approach developed by VCU [33] and incorporates methodologies such as combinatorial testing, boundary value assessment, equivalence partitioning, and MCDC structural path coverage. In Figure 35 and Figure 36, the high-level block diagram can be seen. Based on these diagrams, a high-level block diagram can be constructed. The major steps of ORCAS are abridged for brevity, and only the major outputs are provided.

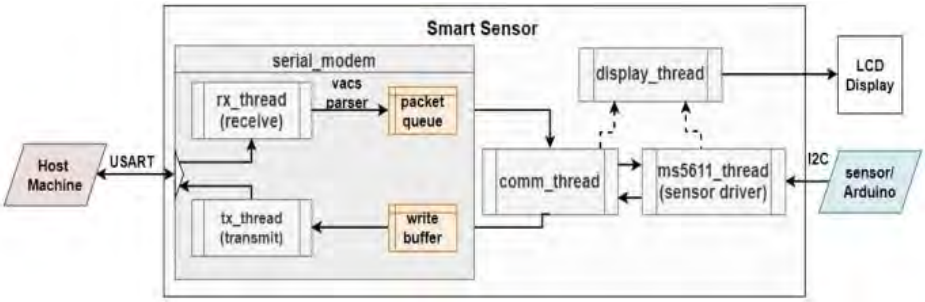

Figure 35. High-level block diagram of the VCU smart sensor device [33].

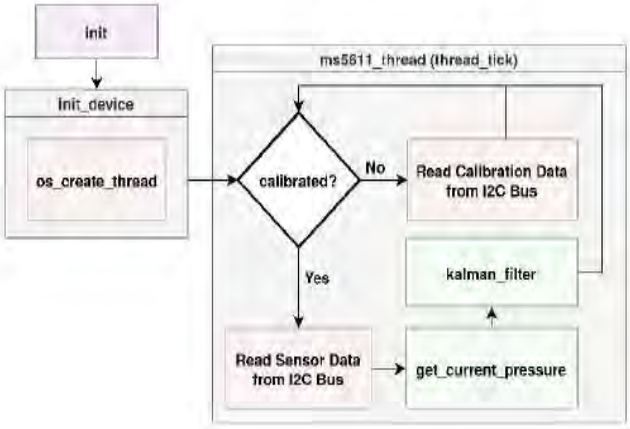

Figure 36. High-level block diagram of the ms5611_thread.

The control block diagram of the smart sensor system is derived from the schematic shown in Figure 35 and Figure 36. Starting from left to right in Figure 35, the system is comprised of: (1) “sensor/Arduino” which is the various pressure and temperature sensors connected to the system, (2) “LCD display” which conveys relevant environmental information to the UAV operator, (3) “MS5611_thread” which is a software module that translates received data into usable information, (4) “display_thread” which is another software module that manages incoming data to be displayed on the “LCD display,” (5) “comm_thread” (cbr) which buffers information to and from the “MS5611_thread” from external peripheral sources, (6) “serial_modem” (sm) software module which handles all incoming and outgoing data packets to the altitude controller, and (7) “host machine” which represents consumers of the information provided by the smart sensor and includes the altitude controller. In Figure 36, the identified software modules are the “get_current_pressure” (gcp) and the “kalman_filter” (kf). The gcp module retrieves and parses analog sensor readings from the pressure and temperature sensors. The kf module then augments and filters the data to be usable by other components of the system. The hardware information was also collected and can be seen in Table 2. Based on this information, the approximate control block diagram is created and shown in Figure 37.

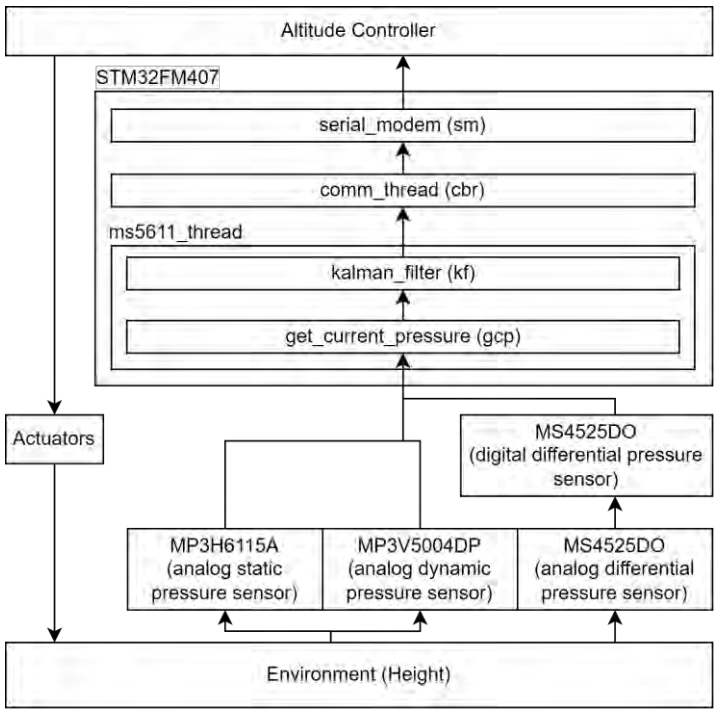

Figure 37. Detailed Software control block diagram of VCU smart sensor system

4.3.4.1 Pseudo-Exhaustive Testing Results

The tests conducted by VCU were collected and assessed for testing completeness. The SRS and SDD documents were reviewed, and test tracing was conducted. In total, 10,687 tests were conducted on three software modules: (1) "circular_buffer_read;" (2) "get_current_pressure;" and (3) "kalman_filter." Duplicate tests in T-way testing were not counted. Fault injection testing was also completed by VCU; however, details on test formulation were not available and thus not counted. However, no new defects were detected through fault injection that were not originally caught by T-way combinatorial testing. The defects that were discovered can be seen in Table 25. In column 1 and 2, the testcase failure and root cause is described and conducted by VCU. In column 3, the defect class and resolution are specified via ORCAS. In total, two checking and six algorithm defects were discovered over the testing period.

Table 25. Software failures, defect class, and resolution from VCU test data.

TestCase Failure	Root Cause	Defect Class and Resolution
Unable to fill the buffer completely. Can only fill buffersize-1 elements.	Incorrect buffer full check.	Algorithm defect, traversal method through circular buffer changed to check all elements.
TestExecution Timeout – Buffer overflow and corruption of neighboring memory addresses cause the "Memcpy" function to hang when called with a length greater than the destination buffer size.	Missing destination buffer overflow check.	Checking defect, limit on size of buffer implemented via IF statement and truncation.
Indicates successful data read operation even with invalid configurations of buffer, "size	Invalid buffer configurations not handled.	Algorithm defect, changed true statement to false when invalid configuration branch taken.

of buffer," "head," and "tail" pointers.		
Returns varying negative values of buffer read length when requested "number of bytes" is negative.	Invalid negative values of the number of bytes to be read is not handled.	Checking defect, limit on negative inputs implemented via IF statement.
Negative values of buffer size are accepted during buffer initialization, and buffer is filled with negative size value.	Invalid buffer size is not considered during buffer initialization.	Checking defect, limit on negative inputs implemented via IF statement.
Actual output value indicates "Infinity."	Missing divide by zero check.	Checking defect, try catch for divide by zero exception added.
Actual output value indicates "NaN."	Missing overflow check in float computation.	Checking defect, try catch for overflow exception added.
Function processes input values outside the valid range.	Missing invalid input value handling.	Checking defect, limit on inputs implemented via IF statement.

4.3.4.2 Qualitative Evidence

The qualitative evidence collected in this case study include the RTM, as shown in Table 26, the TCA, as shown in Table 28, Table 29, and Table 30, and the structural path coverage. In Table 27, the high-level specifications of the listed requirements are provided along with the qualitative scoring sheet for each requirement tested. The requirements were derived from the SRS document provided by VCU. In total, 10 high-level requirements were identified. In Table 26, row one, the requirement identification tag is provided, followed by the total number of explicit and implicit test cases. In column 1, row 3 to 6, the test case file identification tags are listed. The tag "gcp_t6" represents the tests for the function "get_current_pressure" module with six-way combinatorial testing. The tag "kf_t5" represents the tests for the "kalman_filter" module with five-way combinatorial testing. Lastly, the tag "cbr_c4" represents the tests for the "circular_buffer_read" module with four-way combinatorial testing. While lower order combinatorial testing was conducted (i.e., two-way), there are redundant whenever a higher-order combinatorial test exists (i.e., five-way) and therefore not included in the RTM.

Table 26. Requirements Traceability Matrix for VCU smart sensor

Req. Identifiers	**Reqs. Tested**	**Req. 1**	**Req. 2**	**Req. 3**	**Req. 4**	**Req. 5**	**Req. 6**	**Req. 7**	**Req. 8**	**Req. 9**	**Req. 10**
Test Cases	5	0	0	0	1	1	1	1	1	0	0
Tested Implicitly	10	1	3	0	1	1	0	0	0	2	2
gcp_t6	5		x		x	x				x	x
kf_t5	7	x	x		X	X	X	X	X		
cbr_c4	3		x							x	x

In Table 27, row 1, column 1, the requirement tag is provided, followed by a brief description in column 2. In column 3, tests that were implemented for each requirement were traced. If targeted tests for that requirement existed, a "complete" grade (or 1) is assigned. Similarly, if no targeted tests were created, an "incomplete" grade (or 0) is assigned. If tests that required part of the requirement existed but were targeted at other requirements, an "indirect" grade (or 0.5) is assigned. For example, the data averaging requirement (REQ-7) had direct tests associated and was completed with T-way combinatorial testing. For REQ-7 to be tested, collection of sensor data (REQ-1) must have been functional too. However, no tests were explicitly designed for REQ-1; therefore, only partial indirect testing was

conducted on REQ-1 via testing of REQ-7. Of the 10 high-level requirements, five had tests directly associated with them, four had indirect testing, and one had no tests.

Table 27. Scoring of the requirements traceability matrix for the VCU smart sensor.

Specified Requirements	**Functional Description**	**Test Complete**	**Scoring**
Collection of Sensor Data (**Req. 1**)	ASCII format starting with six calibration constants followed by float point data.	Indirect	0.5
Transmission of Data (**Req. 2**)	I2C protocol transmitting (temp., pressure, KF-pressure).	Indirect	0.5
Device Reconfiguration (**Req. 3**)	Capability at updating parameters of MS5611.	Incomplete	0
Re-ranging of Data (**Req. 4**)	Valid differential pressure (-1) to (+1) psi, valid absolute pressure range (0) to (15) psi, capability to re-scale pressures to defined ranges.	Complete	1
Temperature-effect Compensation (**Req. 5**)	Valid range (-40) to (125) °C, capability to adjust temp. to valid range.	Complete	1
Transmitter Calibration (**Req. 6**)	Recalibrate [min, max] of internal ranging and compensation parameters.	Complete	1
Data Averaging (**Req. 7**)	Analog data is converted using moving avg. Kalman filter with size of window updatable as user parameter.	Complete	1
Data Conversion (**Req. 8**)	Float to int conversion and rounding must exist with error correction.	Complete	1
Data Output (**Req. 9**)	Manage serial transmission to host via UART.	Indirect	0.5
Data Logging/Clocking (**Req. 10**)	Host update rate must be greater than 2 Hz, with three commands to shell program.	Indirect	0.5

For trigger coverage assessment, all three levels of software testing were assessed (Table 28, Table 29, and Table 30). Both component and subsystem testing were complete and had various types of tests associated. The method implemented by the VCU development team was T-way combinatorial testing with BVA and EP The test data provided did not specify how many tests were conducted. Furthermore, T-way combinatorial testing innately covers multiple triggers due to its method of test case generation. Therefore, an "X" is used to denote that a test case exists, but the exact number is unknown. For the level 2 TCA, no direct tests were discovered for the modules "ms5611_thread" and "serial_modem." However, to run the level 1 tests, both modules must be active. For instance, receiving and transmitting data to users must pass through the "serial_modem" and cannot be bypassed. Therefore, the tests from the level 1 TCA are treated as indirect tests for the level 2 TCA. Ultimately, all tests for the level 2 TCA are indirect and suggest a potential area requiring further testing. Lastly, there were inadequate tests developed for the level 3 user TCA. Specifically, no tests were found for configuration or workload/stress testing. This corresponds with the RTM as no tests were traced to REQ-3. The startup/restart trigger also only had indirect tests (as the device had to be turned ON to run). The justification from the VCU development was that the pseudo-exhaustive testing methodology conducted in [33] was to demonstrate how T-way combinatorial testing can be implement on a DI&C system and was not intended to be comprehensive.

Table 28. Level 1 component TCA.

Trigger Identifiers	**gcp**	**kf**	**cbr**
Test Cases	2410	7775	500
Tested Implicitly	0	0	0

Simple Path	X	X	X
Complex Path	X	X	X
Coverage	X	X	X
Variation	X	X	X
Sequence	X	X	X
Interaction	X	X	X

Table 29. Level 2 subsystem TCA.

Trigger Identifiers	**ms5611_thread**	**serial_modem**
Test Cases	0	0
Tested Implicitly	10185	10685
Simple Path	X	X
Complex Path	X	X
Coverage	X	X
Variation	X	X
Sequence	X	X
Interaction	X	X

Table 30. Level 3 user TCA.

Trigger Identifiers	**STM32FM407**
Test Cases	0
Tested Implicitly	10951
Software Configuration	0
Workload/Stress	0
Recovery/Exception	X
Startup/Restart	X
Normal Mode	X

In Table 31, the qualitative scoring for the testing activities conducted by VCU is presented. The scoring provides guidance on where testing was inadequate and how it may be improved.

Table 31. Qualitative TCA scoring for VCU smart sensor testing activities.

	Activity	**Defect Triggers Required**	**Implemented Method**	Score
Component	Unit Test	Simple Path	MCDC (Complete)	1/1
	Function Test	Coverage, Variation, Sequence	T-way, BVA, EP, Sequence testing (Complete)	3/3
Subsystem	Unit Test	Simple Path, Complex Path	MCDC (Complete)	2/2
	Function Test	Coverage, Variation, Sequence, Interaction	T-way, BVA, EP, Sequence testing, Interaction verification (Complete)	4/4
System	System Test	Startup/Restart	(Indirect)	0.5/1
	System Test	Recovery/Exception Normal Mode	T-way, BVA, EP, fault injection (Indirect)	1/2
	System Test	Configuration Workload/Stress	(Incomplete)	0/2

For the structural path coverage, VCU's team demonstrates that the use of T-way combinatorial testing can achieve 100% MCDC coverage [33]. Additional path assessment conducted via ORCAS agrees with VCU's results and revealed 12 unreachable but extraneous/benign code segments.

4.3.4.3 Quantitative Evidence

The quantitative evidence that was collected includes the eight defect reports from testing, as observed in Table 25, and the 10951 tests. Each test was assumed to correspond to 1 hour of effort. The software failure modes are determined from the SRS and SDD documentation. Based on specifications, one of the hazards to the system was determined to be "incorrect pressure above $\pm 1\%$ of true value was provided to dependent devices causing unstable altitude adjustments." The possible UIFs includes A, C, and D corresponding to a pressure reading that is missing when needed, a pressure reading that is asynchronous to reality, and a pressure reading that is too high or too low, NaN, or Inf invalid values. UIF-B is not applicable for continuous monitoring systems as the value will always be needed. Using the correlations shown in Table 24, the individual UIF failure probabilities are determined, observed in Table 32. Defects with zero failure probability (or not detected) are excluded from the table. The total software failure probability regardless of failure mode was determined to be 8.77E-4 per hour (bottom right sum in Table 32). The probabilities of each UIF can be seen in the last row, which is the sum contribution of the individual probabilities. In Figure 38, the UIF probabilities of each failure mode is shown with 2 standard deviation bars calculated from the uncertainty table from Table 24.

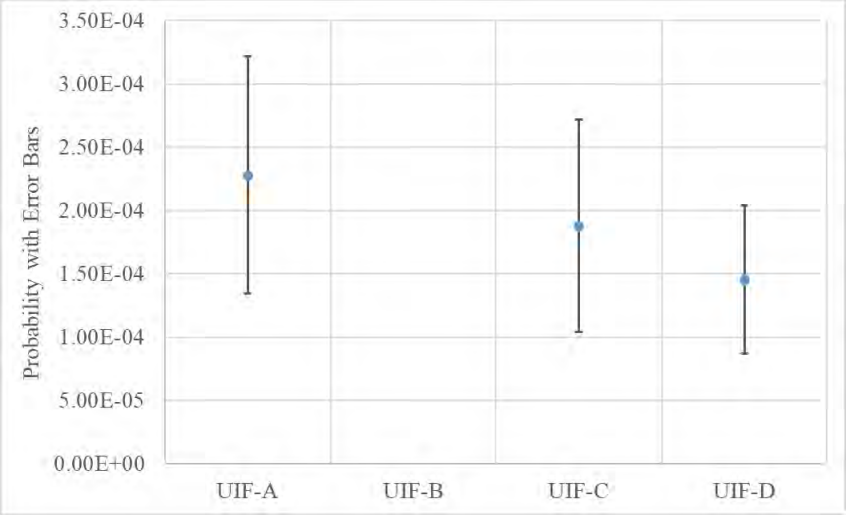

Figure 38. Total UIF probabilities by mode with two standard deviation error bars.

Table 32. Expected probabilities of each UIF by defect class.

Defect	UIF – A	UIF – B	UIF – C	UIF – D	Total
Algorithm	6.37E-05	0	5.81E-05	3.71E-05	1.83E-04
Checking	1.64E-04	0	1.30E-04	1.08E-04	5.48E-04
Total	2.28E-04	0	1.88E-04	1.46E-04	7.31E-04

4.3.4.4 ORCAS Results

The VCU smart sensor was designed to represent a safety-critical smart sensor device. As such, a thorough design, documentation, and development environment of the software was conducted. The device also had extensive operational hours justifying reliability. However, from the ORCAS assessment, we can conclude two specific things. From qualitative evidence, the inadequate areas of testing verification were identified. These primarily include the system-level configurable options, stress, and

communication with peripherals. In addition, while many conditions were tested, approximately 76%, which may be unacceptable and used as an argument against software reliability. From the quantitative evidence, a maximum failure probability of 8.77E-4 per hour was determined based on the number of tests and number of detected defects. Note this value is for any software failure regardless of severity. While this value may seem acceptable, the qualitative evidence suggests that additional hidden defects may exist due to incomplete testing.

4.4 BAHAMAS for Software Reliability Analysis in Data-limited Conditions

As stated in Section 4.3, quantification of software-based CCFs is a required and important step in the DI&C system reliability analyses. Another methodology for quantification of software CCFs has been developed as part of the framework and it is discussed below.

4.4.1 Technical Background

Frequently, data is limited for quantification efforts. Most often this will occur early during the development of a software or system. Other times, access to a software may be proprietary, but access to developmental practices may be shared more easily. For these instances, it is beneficial to have a tool to make quantitative assessment for software reliability. Thus, in addition to the ORCAS method for data-rich conditions, the proposed framework provides a secondary method for quantifying the software reliability under data-limited conditions.

The numerous approaches to reliability analysis can largely be sorted into the following categories: test-based methods, SRGMs, rule-based methods, metric-based methods, and BBNs [55]. In addition, many software reliability methods have been formulated to incorporate the timing of events via dynamic modeling [56]. Our selection of any of these methods will depend on their ability to work with limited data. The following is a discussion of each method. Immediately, test-based methods are eliminated as an option. SRGMs estimate software reliability by matching test data with empirical models [55]. Due to their need for test data, they too are not ideal for this work. The rule-based approach assumes a system reliability value can be assigned if that system was designed according to specified rules [55]. Because they do not necessarily depend on test data, the rule-based approach has potential for our work. Metric-based software reliability prediction relies on software attributes (e.g., bugs per line of code) to determine reliability [57]. The metric-based approach provides multiple ways to capture unique features of a software development process, but many metrics still require explicit software details. BBN methods rely on Bayes' theorem and acyclic graphical models to depict the influence of one event on another while also involving dependencies between events [55]. BBNs can incorporate disparate information and expert elicitation to perform assessments [55], which allows them to work well for the limited-data scenario. Expert elicitation can be a source of model uncertainty. The dynamic methods will be saved for later research efforts, allowing the focus of our current work to remain on those methods that maintain compatibility and familiarity with conventional PRAs. Due to its potential for limited-data applications, this work relies on a Bayesian approach for assessing software reliability.

Bayesian approaches rely on the Bayes equation (see Equation (12) from [58]) which is a construct of probability values from which details about a hypothesis (H) can be made, given observations on certain evidence I.

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \tag{12}$$

BBNs rely on Bayes' theorem and graphical models to depict the influence of one event on another while also involving dependencies between events. BBNs are used largely in two ways, either for prediction or for diagnosis. Prediction analysis employs prior probabilities of parent nodes and

conditional relationships to predict the state of a child node (given a cause, an effect can be determined). Diagnosis analysis proceeds in reverse; given an effect (i.e., known evidence) and conditional relationships, the probabilities of potential causes can be determined [59]. BBNs can incorporate disparate information, which is advantageous for performing analyses [55]. Reliance on a BBN to assess software reliability will depend on defining the conditional relationships between the mechanisms that influence software failure.

Software failure definitions can be giving as a combination of the IEEE definitions for software and failure. The IEEE defines software as computer programs, codes, procedures and data pertaining to the operation of a digital component or system [60]. The definition of failure, also provided by the IEEE, is the termination of the ability of a system to perform a required or intended function or its inability to perform within previously specified limits [60]. Thus, their combination provides a definition for software failure: The inability of software to perform its designed/required or intended functions or its inability to perform within specified limits.

Software does not necessarily fail in the same manner as mechanical components [61]. For example, mechanical fatigue is not a dominant failure mode for software. Software performs exactly how it has been designed to perform; any unwanted action or behavior is due to faults in the software. A software fault is a defect or vulnerability that, under certain conditions, may cause a failure [62]. (See Figure 8). Faults arise within a software due to human errors in the SDLC; these faults are the main reason for unwanted software behavior [63]. In 2018, Chu et al. demonstrated a BBN-based method that considered the introduction and removal of defects during a software's SDLC. Their work defined the conditional relationships of the BBN through heavy reliance on expert election. In our approach, we are focused on minimizing the elicitation burden by relying on existing tools. Defects, a necessary component of software failure, can ultimately be traced to human errors during the SDLC. The quantification of these defects is the subject of human reliability analysis (HRA) [64].

Through similar means, each approach to HRA evaluates human-performed activities and tasks and indicates the likelihood that the task or activity will be completed successfully. A number of methods have been developed for HRA over the years; some of the more well-known methods include the Technique for Human-Error Rate Prediction (THERP) [64], the Standardized Plant Analysis Risk-Human Reliability Analysis (SPAR-H) method [65], and the Cognitive Reliability and Error Analysis Method (CREAM) [66]. HRA falls within three categories. The first category relates to actions completed prior to a failure event (i.e., errors in maintenance, testing, or calibration). The second deals with actions that are the direct initiator of an event, and the third is for human actions made after or in response to a failure event [67]. Because most software faults are associated with faults in SDLC, the HRA applications used in this work will apply to the first category.

HRA determines the probability that a human will perform a task incorrectly, but there is not a good way to determine what human error has occurred or how many human errors exist (e.g., the HRA is not performed for every keystroke or hand calculation). We can only assume that the human-error probability (HEP) indicates whether a task was performed incorrectly due to some unknown number of errors or "faults." In other words, the HEP for a task can be considered the "probability of faults" made in the completion of a task. Consequently, it is necessary to determine how to transform a probability of faults into a probability of software failure. Researchers have demonstrated transforming the number of faults into the probability of failure by assuming software failure probability (SFP) is proportional to the number of faults within a software [68] [69]. We assume a similar relationship exists between the SFP and the probability of faults within a software.

Thus far, BBNs and HRA have been identified to support the BAHAMAS for limited-data reliability analysis. The BBN provides structure for our work, and HRA provides a tool to quantify the human errors during software development activities. In order to describe the conditional relationships between errors (i.e., software defects) and the probability of software failure, another method is needed. ORCAS, the

method developed for a data-rich reliability analysis, has already defined this relationship by employing concepts from IBM's ODC [70]. Specifically, ODC is a method for categorizing failures with respect to the defects that lead to them. ODC defines specific categories in which defects may fundamentally be classified. The ORCAS links the ODC defect classifications with the software failure modes defined for RESHA (i.e., UCAs and UIFs as discussed earlier). The relationships defined in ORCAS allow researchers to define the conditional probability that a particular failure mode will occur given the activation of specific defect types. The relationships defined in ORCAS, by its implementation of ODC, can be used to support the definition of conditional relationships necessary for an analysis based on a BBN. Table 24 shows the relationships for relating defect classifications to UCAs/UIFs.

Together, a BBN in combination with HRA, ODC, and RESHA define a set of tools to evaluate software reliability under the limited-data scenario. The BBN provides a structure for performing computation and quantification. HRA and ODC define node probabilities and conditional relationships within the BBN, and RESHA, founded in concepts from STPA, defines the software failure modes of interest for which the BBN will be solved. The next sections describe the development of the BAHAMAS BBN.

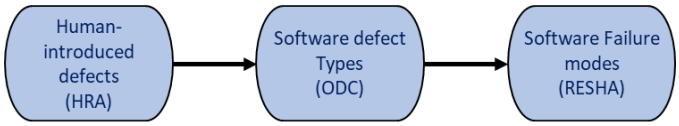

Figure 39. Methods employed in BAHAMAS. The BBN relies on HRA, ODC, and RESHA

4.4.2 Methodology

4.4.2.1. Model Development

This section will detail the construction and development of BAHAMAS model for software reliability. The model is indented to be flexible but what is shown and described is the generalized form of BAHAMAS. As mentioned, there are a few tools combined for use with BAHAMAS. This section will detail the application of each in forming the BAHAMAS generalized model structure for reliability analysis.

The BBN is used to model the effects of root errors on software performance. The BAHAMAS takes a basic event from a FT (e.g., a child node) and models the influence of causal factors (e.g., parent nodes) on the child node. BBNs are used primarily for diagnostic (inference) or predictive analyses [59]. This work will focus primarily on using predictive analysis with some minor reliance on inference through observations. The first step to construct a BBN is to identify variables of the model [71]. According to [71], variables must meet certain criteria, called a clarity test, for the construction of the BBN to work well. The clarity test is given below [71]:

- The state space of a variable *B* must consist of an exhaustive set of mutually exclusive values that the variable can take
- A variable should represent a unique set of events (i.e., there should be no other variable of the problem domain whose state is mutually exclusive of a state of *B*
- The variables should be clearly defined (i.e., easy to understand).

BAHAMAS is intended to predict the software failure modes by tracing the causes of failure to the defects that arise during the SDLC [18]. Thus, there should be variables within BAHAMAS that represent the SDLC. The SDLC has different stages that can be represented as variables for the BBN. Additionally, ODC, as employed by ORCAS, is used to represent conditional relationships between defect types and STPA-based software failure modes (i.e., UCAs/UIFs). Thus, the SDLC stages, ODC defect types, and software failure modes will define variables within the BBN.

There are a number of guidelines and styles for software/systems development processes or life cycles. The 2015 Systems Engineering Handbook contains a collection of figures from various publications describing different features of life cycle stages [72]. IEEE 24748-1:2018(E) provides international standard viewpoint of a life cycle processes [73]. Activities to be performed during these stages are also expressed directly in the context of software within ISO/IEC/IEEE 12207:2017(E) [74]. ISO/IEC/IEEE 12207:2017(E) provides details specifically from a software viewpoint. These sources provide a general list of the life cycle stages including the concept, development, production, utilization, support, and retirement stages. The purpose of the concept stage is to identify stakehold'rs' needs, explore concepts, and propose viable solutions, essentially conceptualizing the main ideas and plans for the project. The development stage works to define system requirements, create solution descriptions, build, verify, and validate the system. The production stage focuses on producing systems, then inspecting and testing them. The utilization stage is focused on the general operation and use of the designed system. The support stage emphasizes sustaining the system's capabilities. Finally, the purpose of the retirement stage is to store, archive, or dispose of the system. These examples serve as the general inspiration to our selection of variables for the SDLC. BAHAMAS relies specifically on the software development processes, thus will exclude the retirement stage. In addition, concepts from production stages and development stages will be selected to form a more segmented focus on the development part of an SDLC. The following are the SDLC stages selected for BAHAMAS:

- Concept stage
- Design stage
- Implementation (i.e., implementation of design requirements or a coding stage)
- Testing stage
- Installation and maintenance (I&M) stage (a combination of support and utilization).

These stages cover the main concepts from IEEE 24748-1:2018(E) but arranged for convenience to allow more resolution for activities where human errors may be introduced. (For example, usage and retirement are not a primary source of defects, but design and coding are.) These variables and their states are given in Table 33. The next variables for the BBN are those relating to the defect classifications from ODC.

The ORCAS section introduces seven ODC defect classes which are directly associated with the development of coding, on which ORCAS is focused. There is an additional defect class, called the documentation defect [70], that pertains directly to early design stages. Documentation defect impacts both publications and maintenance notes [70]. Documentation may lead to other mistakes in concepts and design stages. Given that BAHAMAS is focused on minimal data, any concept from early design stages may prove useful, thus, the documentation defect classification is included. These types are added to Table 33 as potential variables for the BBN.

Next are STPA-based failure modes from RESHA. These failure modes are discussed in the previous sections and so will not be readdressed here other than to state that the four general classifications of type A, B, C, and D are used. The distinction of UCA or UIF does not matter to the purpose of the software failure mode. The failure modes are also added as potential variables to Table 33.

Table 33: Potential BBN variables.

#	Variable Name	Description	States	Clarity Test?
1	Concept Stage	Defects introduced during stage	Yes, No	Y
2	Design Stage	Defects introduced during stage	Yes, No	Y
3	Coding Stage	Defects introduced during stage	Yes, No	Y
4	Testing Stage	Defects introduced during stage	Yes, No	Y
5	I&M Stage	Defects introduced during stage	Yes, No	Y

6	Assignment Defects	Assignment defects exist in the software	Yes, No	Y
7	Algorithm Defects	Algorithm defects exist in the software	Yes, No	Y
8	Checking Defects	Checking defects exist in the software	Yes, No	Y
9	Documentation Defects	Documentation defects exist in the software	Yes, No	Y
10	Function Defects	Function defects exist in the software	Yes, No	Y
11	Initialization Defects	Initialization defects exist in the software	Yes, No	Y
12	Relationship Defects	Relationship defects exist in the software	Yes, No	Y
13	Timing Defects	Timing defects exist in the software	Yes, No	Y
14	Failure	Software failure	None, UCA/UIF-A, B, C, D	Y

The potential variables are listed in Table 33. Each variable can meet the clarity test and is clearly defined. Their states are exhaustive, and no variable state is mutually exclusive with the state of any other variable states, thus they are selected to be used in the BBN. A conceptual BBN is formed in Figure 40. Where, during each stage of the SDLC, defects are introduced which can cause failures. The working form of Figure 40 is based on the general outline shown in Figure 39.

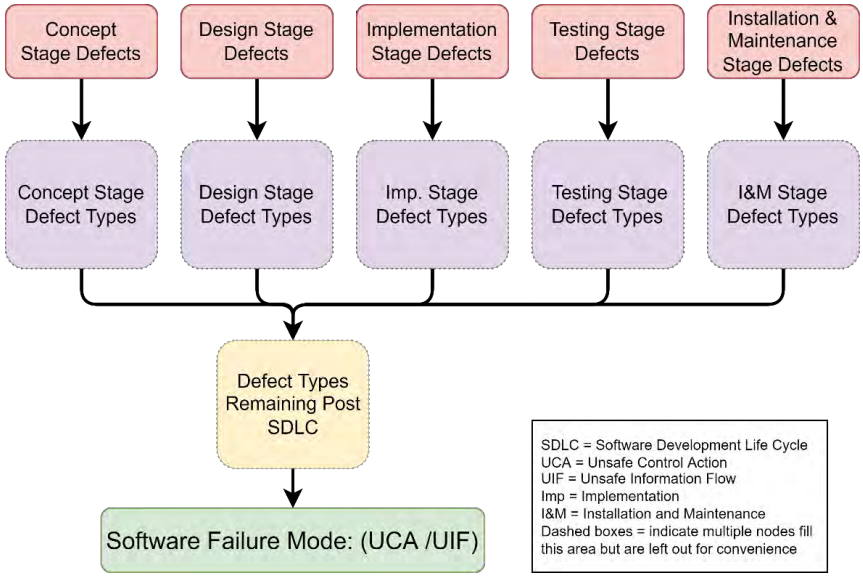

Figure 40. A conceptual BBN.

It is clear from Figure 40 that something must be done to account for defect removal. It can be reasoned that defect identification can occur through processes of testing, inspection, and review. A natural extension to the figure above is to add five new nodes corresponding to the review during each of the design stages. Review activities influence the identification and removal of defects. Better yet, the quality of review activities has a profound impact on the likelihood of defects remaining within a software. The quality of the review activities will be assigned a graded level from no review to excellent review. It is anticipated that safety-critical DI&C systems will be developed with at least a high-level review quality. Table 34 shows the BBN variables and their states associated with review quality.

Table 34: Review quality variables for the BBN.

#	**Variable name**	**Description**	**States**	**clarity test?**

1	Concept Stage Review Quality	The quality of the review activities performed during this stage	None, Low, Medium, High, Excellent	Y
2	Design Stage Review Quality	The quality of the review activities performed during this stage	None, Low, Medium, High, Excellent	Y
3	Coding Stage Review Quality	The quality of the review activities performed during this stage	None, Low, Medium, High, Excellent	Y
4	Testing Stage Review Quality	The quality of the review activities performed during this stage	None, Low, Medium, High, Excellent	Y
5	I&M Stage Review Quality	The quality of the review activities performed during this stage	None, Low, Medium, High, Excellent	Y

The variables from Table 33 and Table 34 are combined into a generalized form for BAHAMAS, as shown in Figure 41.

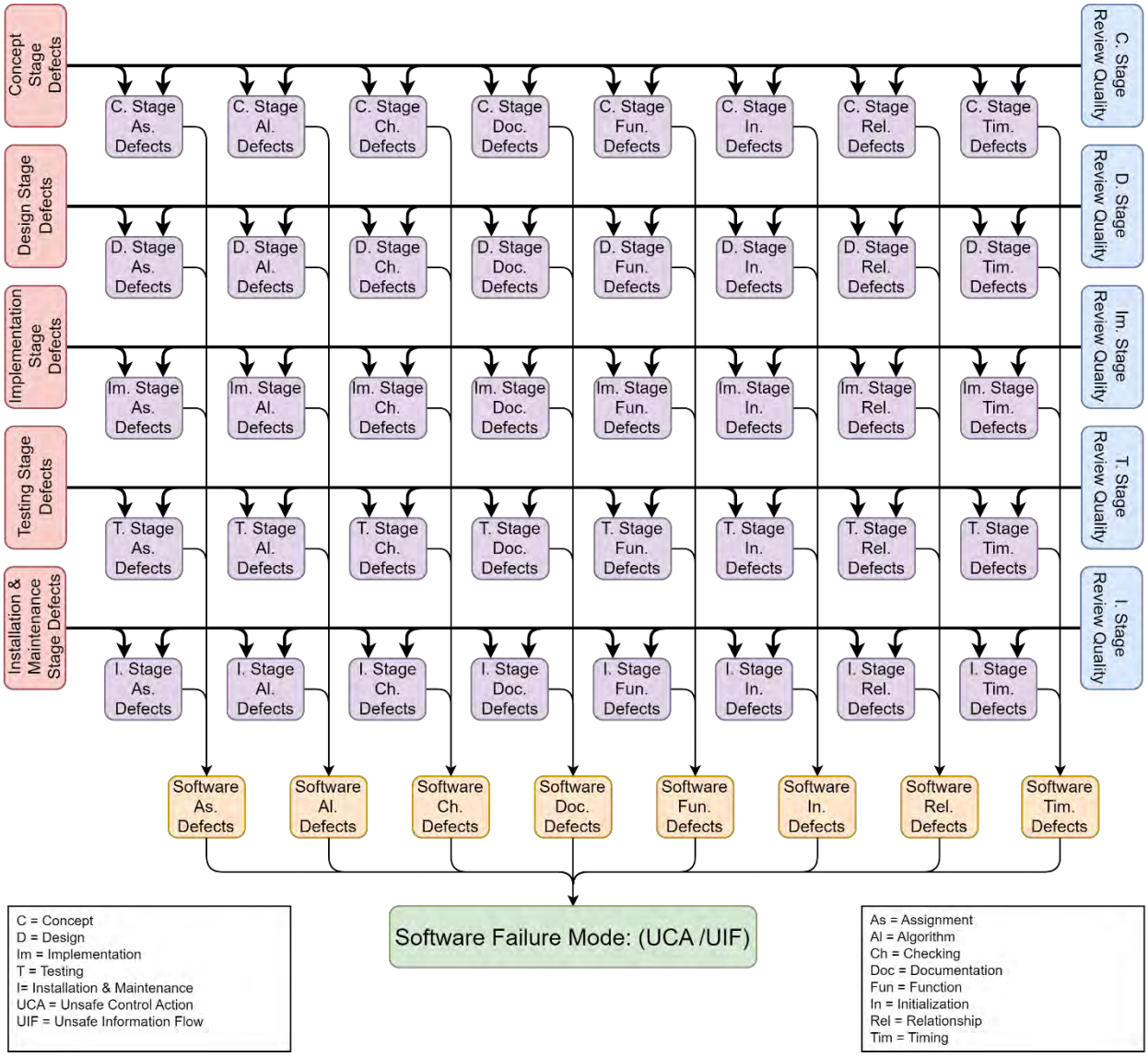

Figure 41: A generalized form for the BBN in BAHAMAS.

4.4.2.2. Model Conditional and Node Probabilities

- **Probabilities for SDLC Stage Defect Nodes:**

The BBN requires probabilities for the state of each variable of the network. In the cases where variables have parents, conditional probabilities must be defined. The root nodes that represent the SDLC are ultimately related to case-specific evaluations with BAHAMAS. For example, concept stage errors/defects are defined using HRA. THERP was selected for this analysis because it is one of the most widely known methods, having been in use for over 50 years [75]. The following is a brief discussion of the application of THERP for the assessment of SDLC activities. The full details can be found in the Appendix. THERP consists generally of the following steps for the quantification of human error [64]:

1. Define the system failures of interest
2. List and analyze the related human operations (i.e., perform a task analysis)
3. Estimate relevant human-error probabilities.

Because the results of the HRA analysis are case specific, the application and quantification of these root nodes will be demonstrated as part of the BAHAMAS case study.

The next section details the BAHAMAS to quantifying probabilities for the states of the nodes that represent the review quality at each stage of the SDLC. Here it is important to note that HRA may already provide a method to account for such activities (e.g., THERP does). However, given that we are evaluating the quality of review activities as distinct nodes of the BBN, there is no need to separate these considerations to avoid double counting. Consequently, the HRA performed for BAHAMAS will not incorporate corrective review actions. Corrective review actions will be saved for consideration in the conditional probability evaluations of the child nodes of SDLC review quality nodes and SDLC stage error nodes.

- **Probabilities for SDLC Review Quality Nodes:**

This is again a case specific exercise that will be demonstrated in the case study. However, the probabilities assigned for this root node, prior to any specific case study, will be uniformly distributed among the node states (i.e., 0.20 is assigned to all five states for the review quality nodes). This uniform distribution acts as a placeholder; ultimately, the state of this node will always be observed (e.g., the probability of high will be set as 1.0 given that the requirements for a high score have been met).

The requirements for the scoring from low to excellent review quality should consider several concepts. The first relates to the degree to which triggers are reviewed during a particular SDLC stage. The ODC methodology indicates that activities during design review, codes inspection, and testing can help identify and remove defects [39]. ODC employs investigations of triggers and triggering mechanisms during different design stages in order to find and identify defects. For example, there are 21 total defects listed in [39]. Each has been assigned to different review stages. For example, design conformance is a trigger that pertains to design review activities. As such, a quality design review should investigate the design conformance trigger. It has been experimentally shown that consideration of all defect triggers can detect more latent defects than an unstructured testing methodology [40]. Thus, providing comprehensive and complete investigations of all triggers during each stage of the SDLC will lead to excellent quality reviews, and ultimately reduce the probability of defects remaining within the software. Table 35 shows the triggers and their associated activities adapted from [39].

Table 35. Review activities and associated triggers.

For Use with Concept, Design, and Implementation Review Activities	Design Conformance
	Logic/Flow
	Backward Compatibility
	Lateral Compatibility

	Concurrency
	Internal Document
	Language Dependency
	Side Effect
	Rare Situations
For Use with Testing Review Activities	Simple Path
	Complex Path
	Test Coverage
	Test Variation
	Test Sequencing
	Test Interaction
	Workload/Stress
	Recovery/Exception
	Start up/Restart
	Hardware Configuration
	Software Configuration
	Blocked Test (Previously Normal Mode)

In addition to the consideration of the triggers, there is an added benefit of employing multiple reviews performed by different individuals. The comprehensiveness of the SDLC review activities indicates how well or how likely defects will remain within the software when it is delivered for use. Good coverage of triggers and high numbers of reviews will increase the review quality and the quality of the delivered product. Table 36 shows the scoring for the review quality nodes for each SDLC stage.

Table 36: Score table for review quality (Q) node for SDLC stages.

Review Quality ($Q = f(T, R)$)		**Average # of reviews per task of SDLC Stage (R)**			
		$0 < R \leq 1$	$1 < R \leq 2$	$2 < R < 3$	$R \geq 3$
Coverage of ODC Triggers (T)	Coverage < 50% (T = 1)	Low	Low	Medium	High
	50% ≤ Coverage < 70% (T = 2)	Low	Medium	High	Excellent
	70% ≤ Coverage < 100% (T = 3)	Medium	High	Excellent	Excellent
	Coverage =100% (T = 4)	High	Excellent	Excellent	Excellent
Note: 1. If R=0 or T=0, Q=None. 2. If average number of reviews exceeds 3 per task of SDLC, then score Review Quality accordingly. However, use R = 3 for the calculations with Equation (13).					

- **Probabilities for Defect Type Nodes:**

The next nodes that must be defined are the defect type nodes. As the software development progresses, the distribution of defect types that remain within the software vary. It is unlikely to find a defect related to the algorithm of a code prior to developing the algorithm itself (i.e., prior to the design state of the SDLC). The variation of defect types during the SDLC can be seen in [70]. It is natural to assume that the concentration of defect type changes with development, and there may be patterns consistent with defect distribution depending on the review activities that are performed. To support this assumption, we investigated a public database [76] (associated with [46]) of ODC results concerning three separate software packages (i.e., MongoDB, HBase, and Cassandra). The ODC results indicate the defects that are likely to be found during certain review activities. Figure 42 gives the distributions associated with the three databases and data reported by Chillarege et al. [70]. Despite being developed by different groups or organizations there is some consistency with the distributions. These defect distributions are used to represent the conditional probability of finding certain defects during the review

activities. In other words, the data gathered provides the Pr(Defect remains| Review activity is performed). Table 37 provides the distributions for each defect type and SDLC stage. This table provides a view of the defects at each stage, but further modification is needed to account for the quality and average number of reviews performed.

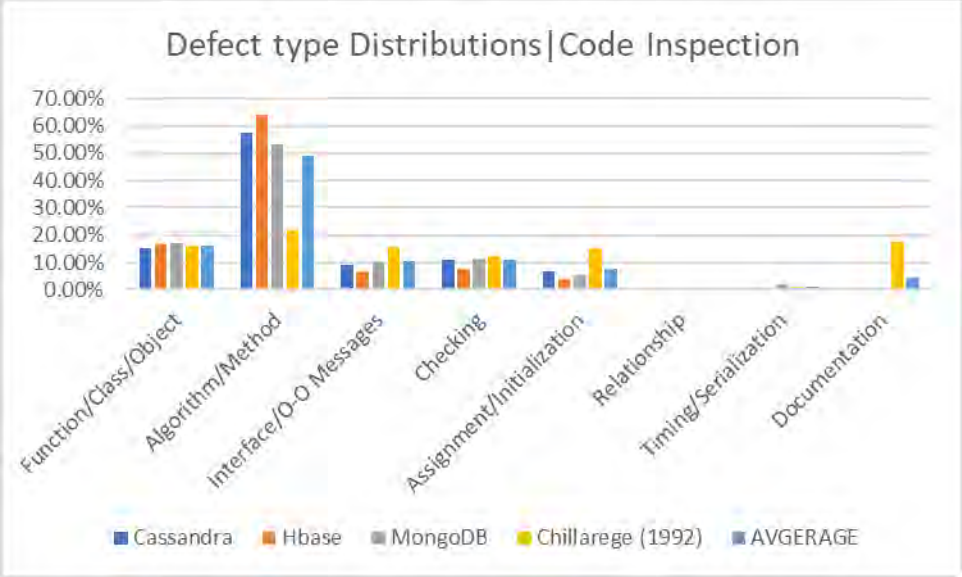

Figure 42. Defect type distributions given code inspection.

Table 37: General expected distributions for defect types for each review activity.

Defect Type	**Al.**	**As.**	**Ch.**	**Doc.**	**Fun.**	**Int.**	**Rel.**	**Tim.**
Type | Concept Rev.	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
Type | Design Rev.	0.1363	0.586	0.586	0.304	0.2464	0.1589	0.00	0.374
Type | Imp. Rev.	0.4913	0.748	0.1080	0.436	0.1609	0.1040	0.035	0.103
Type | Testing Rev.	0.4296	0.991	0.1079	0.088	0.848	0.991	0.622	0.112
Type | I&M Rev.	0.4296	0.991	0.1079	0.088	0.848	0.991	0.622	0.112
Type | no review	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
Note 1: general expectation for concept stage is that no design activities have been performed. Thus, only documentation defects will exist. Note 2: Datasets contained no Relationship defects for Design Rev. Note 3: The distribution of defects, given no review, is assumed to be uniform.								

To account for the quality of the review, we borrow ideas given in THERP as well as indicated in Humphreys [77]. Humphreys presents a model for defense against common cause failure based on the idea that "in so much as the most dramatic improvements are made initially, subsequent improvement is more and more difficult to achieve, requiring increasing amounts of effort for teach step on the road to perfection." [77] In other words, the degree of benefiting from a review to the reduction of risk is a function of time and effort. This is not purely a linear relationship but may follow an exponential curve. Figure 43 demonstrates this assumption.

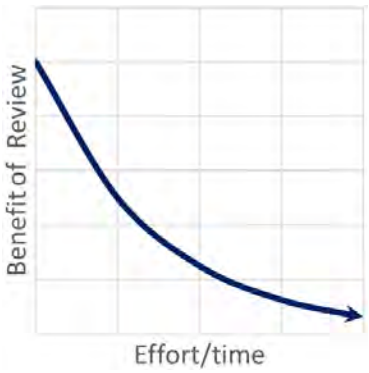

Figure 43. General expectation of the benefit of review activities for reducing defect probability.

For this work, BAHAMAS will represent the quality of review activities by employing a simple exponential curve: y = exp(-x). Further, corrective review actions (number of reviews) that are normally part of HRA are saved for consideration in the conditional probability evaluations of the child nodes of SLDC review quality nodes and SDLC stage error nodes. THERP prescribes a process that accounts for recovery actions during a specific task; for example, if a teammate checks or reviews a task, this can serve to prevent or correct a mistake. The process of accounting for a recovery considers dependency between the performer and the checker/reviewer. Essentially, the benefit of the recovery action is conditional. The more involved the checker/reviewer, the better the recovery probability. It turns out that THERP's correction for recovery also follows a nearly exponential curve. For a given task, THERP assigns an HEP. Then, when a subsequent reviewer can make a recovery, the value of the recovery is given based on THERP's dependency equations (for more detail see THERP manual [64]). Figure 44 shows an HRA event tree and the calculation of event probability given the recovery path.

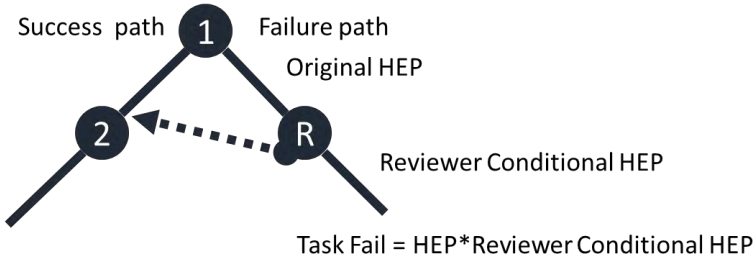

Figure 44. Example of HRA event tree with recovery (i.e., reviewer).

The total task failure probability given is based on THERP's dependency rules which follow a nearly exponential curve. To account for the number of reviews, BAHAMAS employs a modifier to the exponential equation: y = exp(-R*x) where R = review number 0- 3. Figure 45 compares THERP's model and BAHAMAS model for recovery on a HEP of 3E-03. The results show that BAHAMAS can provide similar results as HRA thus making up for the fact that we excluded the recovery actions in the HRA only to have BAHAMAS account for them in the BBN.

The BAHAMAS model for the conditional relationship between a child of an SDLC Stage node and Review quality node is given by (13). Within the defect conditional probability (DCP) equation, G is the general expected probability for a defect type, given a review has been performed (from Table 37). T represents the trigger coverage from none to excellent corresponding to integers from zero to four. And R is the average number of reviews performed for each task of the SDLC stage (see Table 36).

$$DCP = f(G,T,R) = Ge^{-TR} \tag{13}$$

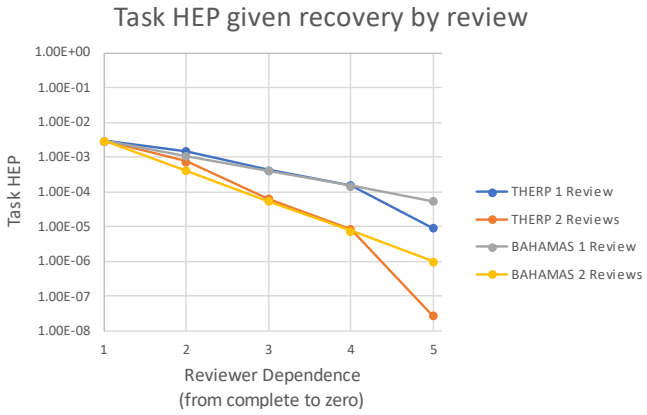

Figure 45. Comparison of BAHAMAS and THERP predictions of human error while considering the quality of review activities. X-axis = step increase in review quality quality.

- **Probabilities for Software Failure Nodes:**

The probability of software failure is accounted for in the ORCAS section. BAHAMAS employs the ORCAS with additional consideration of the documentation defect class. The conditional probability for software failure given a documentation defect was set as a uniform distribution, due to a lack of data for this condition. Additionally, a conditional relationship between nodes representing defect type remaining and the software failure mode is assumed: the occurrence of a software failure results from an activated software defect. For convenience, Pr(defect type remaining) is assumed equivalent to Pr(defect type remaining and is active). Thus, the Pr(software failure mode) becomes the union of events given by the summation of the Pr(defect type remains)*Pr(software failure mode| defect type) for each defect type. Table 24 provides the conditional probabilities used for these nodes. Finally, all the necessary information has been provided for the BBN to be solved using conventional software tools.

4.4.3 Case Study

This section details the case study for BAHAMAS starting with a description of the process for implementing the generalized BBN from the previous section. Using BAHAMAS consists of six steps the details of which will be described in the case study. This section demonstrates how BAHAMAS can be used to quantify hazards as part of an approach to the reliability analysis of digital systems. For brevity, the case study will demonstrate the quantification of a single software failure identified as part of the hazard analysis of an APR-1400-type four-division digital RTS [16]. In addition to the assumptions of the hazard analysis [16], the following are the assumptions of the present study:

- The hardware failure causal factors that could lead to a software failure have been left out of the BBN because they are grouped with the hardware failures in the FT and accounted for separately.
- CCFs have been limited to all n/n identical components failing to employ the beta-factor method for quantifying CCFs and to simplify the analysis.
- The human errors associated with the root nodes are assumed independent in the BBN. This is because any errors in one stage that could be attributed to the another should already be considered by the former. For example, errors in the design stage are independent of the testing stage errors.
- For this study, we extracted details from public documents related to the APR-1400, but there is still information redacted from the public. Therefore, the results of this work in no way reflect the reliability of the APR-1400 design.

- **Step 1: Identify a software failure of interest**

The first step of BAHAMAS is to select an event to quantify. In this case, the software failure of interest pertains to the BPs in the RTS shown in Figure 22. The details of the basic event should include the component, context, and incorrect or erroneous action (e.g., action not provided; action provided and not needed; action provided too early, too late, or in the wrong order; and action provided for too long or stopped too early). In this case, the controller/component is the BP. The action is a failure to provide a trip signal to the logic cabinets (UCA-A Failure). The context is during a plant condition for which a reactor trip is required.

- **Step 2: Gather SDLC data of the software**

BAHAMAS requires details of software development activities. This can come from software design documents, plans, etc. For the case study, we examined in detail the publicly available APR 1400 software program manual [78]. For this study we extracted details, but there is still information redacted from the public. Therefore, the results of this work in no way reflect the reliability of the APR-1400 design.

- **Step 3: Perform HRA for the SDLC**

This step requires that each stage of the SDLC be assessed for the specific human activities associated with them. Also called a task analysis, this is a vital part of HRA. The THERP manual indicates that the "general approach used for HRA has been to identify, analyze, and estimate HEPs for human tasks that system analysts and human reliability analysts determine could have a major impact on the system criteria of interest" [64]. The actions to identify, analyze, and estimate form the basis of steps used in this application of THERP: (1) Identify the system of interest; (2) list and analyze the related human operations (perform a task analysis); and (3) estimate probabilities.

As an abbreviated example, the software program manual defined the following tasks that fit with the concept stage:

1. Define the project purpose
2. Identify hazards and risks
3. Identify codes and standards
4. Create software management plan
5. Create software quality assurance plan
6. Create software verification and validation plan
7. Create software configuration management plan
8. Create software development plan
9. Create software safety plan.

Each task is assessed with THERP to define the total probability of human error for the concept phase. The following is the results for concept phase, task 1: Define the project purpose:

- **Performer**: Group consisting of stakeholders, engineers, and managers
- **Background**: The process of defining a project is similar to assessing or diagnosing a situation that is new. There may be similarities from past experiences, but ultimately the scenario is a new problem that must be solved. THERP was designed for assessing activities within the context of a nuclear power plant; therefore, its application must fit similar conditions. THERP prescribes HEPs for group activities associated with diagnosis of abnormal events. The assumption is made that the diagnosis of an abnormal event is akin to solving a new problem. Experience, training,

and expertise play a role in the diagnosis. But ultimately the effort is not strictly routine. Hence, because each project is unique, we shall assume this is equivalent to diagnosing an abnormal event.

- **Error sources**: faulty diagnosis made by a group.
- **Score guidance:** THERP's Table 20-3 provides guidance diagnostic actions made by a team or group. Item 5 corresponds to one hour for time to diagnose. It is assumed that one hour is sufficient time for a group of qualified individuals to define the purpose of the project. The HEP is not modified because the adjustment rules (Table 12-5) do not apply well to the SDLC activities.
- **HEP:** lognormal distribution. Median 1E-4, Error Factor (EF) 30.

The results of the HRA are shown below for which BAHAMAS will rely on the mean values for a discrete evaluation. Full details can be found in the Appendix.

Table 38. HRA results for root nodes of the BBN.

SDLC Stage	Mean	Median	Lower 5th	Upper 95th
Concept Stage	0.0316	0.0278	0.0137	0.0612
Design Stage	0.0753	0.0682	0.0362	0.1372
Coding Stage	0.0484	0.0425	0.0203	0.0957
Testing Stage	0.0533	0.0470	0.0226	0.1045
I&M Stage	0.0581	0.0515	0.0249	0.1130

- **Step 4: Evaluate the SDLC for the quality of review activities**

This step relies on Table 36 to define the score for review quality with the assumption that the SDLC for safety-critical DI&C systems must have high review quality. Therefore, this case study assumes that 80% of the ODC trigger categories are covered for each stage of the SDLC, and there are an average of two reviews for each task within each stage of the SDLC.

- **Step 5: Evaluate the BBN outputs.**

For this work, we relied on BayesFusion's GeNIe [79] to perform the evaluation of the BBN. The mean probability of UCA-A of BPs is given as 2.077 E-04. We incorporate the influence of the HRA and review quality uncertainties on BAHAMAS outputs, as shown in Table 39.

Table 39. The bounds of BAHAMAS outputs due to uncertainties in HRA and review quality.

Results given review quality	Lower HRA values	Mean HRA values	Upper HRA values
Medium	6.669E-04	1.514E-03	2.895E-03
High	9.166E-05	2.077E-04	3.984E-04
Excellent	2.276E-07	5.159E-07	9.901E-07

- **Step 6: Perform CCF evaluation if necessary**

If it is desired to evaluate the CCF probability for the software of interest, it is necessary to know that BAHAMAS outputs the total failure probability of a software of interest. Software failure only occurs due to the activation of latent defects. BAHAMAS predicts the likelihood that these defects remain in a software. For a CCF (i.e., concurrent failure due to a "shared cause") to occur, there must be (1) multiple components and (2) a cause that is made "shareable" by the existence of a coupling mechanism. The shared cause for software must include common "active" defect or faults. The CCF can occur because multiple components share copies of the same software and trigger mechanism. The only difference between the CCF and independent/individual failure is that a shared defect is activated in a single software vs. in multiple redundant software components. Thus, the configuration of system is what

determines whether a CCF or independent failure occurs. Ultimately the susceptibility of the software to failure still only depends on the activation of the hidden defects. BAHAMAS predicts defects that can be activated by both common and independent means; therefore, the BAHAMAS output is considered as a total value rather than an independent failure value. The process of performing a CCF evaluation is detailed in the following section. The results of the CCF analysis for a four-division digital RTS are shown in Table 54.

4.5 CCF Modeling and Estimation

This section introduces the CCF modeling and estimation approach we developed in this project. Previous sections describe the quantification of total failure probability of software, which can be considered as a summation of individual failure probabilities and CCF probabilities. Currently, quantification of software CCFs still remains as a technical challenge. In this section, a modified BFM is modified for the modeling and estimation of CCFs, especially for software CCFs.

4.5.1 Overview

A CCF is the occurrence of two or more failure events due to the simultaneous occurrence of a shared failure cause and a coupling factor (or mechanism) [80]. The failure cause is the condition to which failure is attributed, whereas the coupling mechanism creates the condition for the failure cause to affect multiple components, thereby producing a CCF [80]. Some examples of coupling mechanisms given in NUREG/CR-5485 include design, hardware, function, installation, maintenance, and environmental conditions [80]. Any group of components that share similarities via coupling mechanisms may have a vulnerability to CCF; a group of such components are considered a common cause component group (CCCG) [80].

The identification of coupling factors and, by extension, CCCGs is an essential part of CCF analysis. Often, CCF models attempt to simplify an analysis by assuming symmetry for the components of a CCCG. For example, a CCCG may be assigned by assuming components are identical where any differences in the coupling factors are ignored.

There are many methods for modeling CCFs, including direct assessment methods, ratio models (e.g., beta-factor and alpha-factor models), Bayesian inference methods, and shock models [81]. Nearly all of them rely on symmetry; the most notable exceptions are the direct assessment methods and those based on Bayesian inference. However, it may be important to explicitly consider the influences of multiple coupling factors that might otherwise be ignored by the symmetry assumption. A software failure is the direct result of operational conditions (i.e., a trigger scenario) activating some hidden software defect(s) causing the inability of the software to perform its require or intended functions (based on concepts from [60] and [63]). A software CCF will occur when a coupling mechanism creates a scenario for operational conditions to activate a common software defect. Given a group of redundant software components, variations in their operating conditions may lead to some, but not all, components failing together. Variation of maintenance activities, input variable sources, component locations, and installation teams influence the operational environment; ultimately, subtle differences in coupling mechanisms may influence which components fail together. Capturing asymmetry between components may be necessary for software CCF modeling, but it can be challenging with conventional methods.

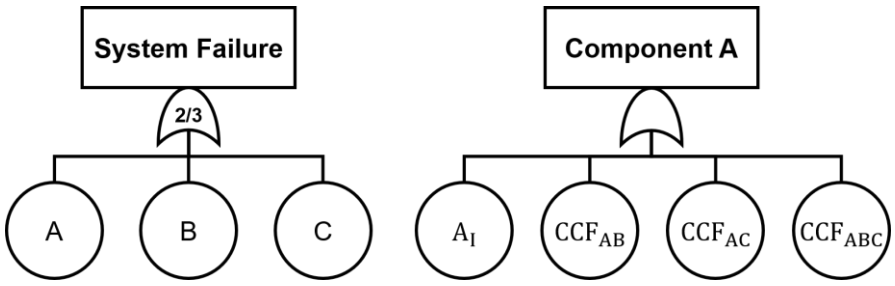

Figure 46. Example system showing the relationship of independent and dependent failures in the context of a fault tree.

Consider the scenario shown in Figure 46 where the components are arranged in the 2/3 criteria for failure. The probability of failure for the system, as given in NUREG/CR-5485, is shown below:

$$P(F) = P(A_I)P(B_I) + P(B_I)P(C_I) + P(A_I)P(C_I) \\ +P(CCF_{AB}) + P(CCF_{BC}) + P(CCF_{AC}) + P(CCF_{ABC}) \quad (14)$$

The common practice in reliability modeling is to assume the failure probabilities (or rates) of similar components are the same [80]. This symmetry assumption results in the following:

$$Q_1^3 = P(A_I) = P(B_I) = P(C_I) = Q_1 \quad (15)$$

$$Q_2^3 = P(CCF_{AB}) = P(CCF_{BC}) = P(CCF_{AC}) = Q_2 \quad (16)$$

$$Q_3^3 = P(CCF_{ABC}) = Q_3 \quad (17)$$

where Q_k^m represents the failure rate or probability of an event involving k components in a CCCG of size m. Now, consider the case when the components of Figure 46 share some, but not all, coupling factors. In this new scenario, components A, B, and C are coupled by procedures, while A and B are coupled by location. The options are to either ignore the differences or account for them directly. Ignoring the differences leads to a single CCCG and reliance on Equations (14) - (17). When the differences are considered, the traditional approach forms two CCCGs: (1) CCCG1 with components A, B, and C; and (2) CCCG2 with components A and B. This ultimately requires a single component to be part of multiple CCCGs. The symmetry assumption applied to CCCG1 relies on the same equations as above. The symmetry assumption applied to CCCG2 gives:

$$Q_1^2 = P(A_I) = P(B_I) = Q_1 \quad (18)$$

$$Q_2^2 = P(CCF_{AB}) = Q_2 \quad (19)$$

Placing A and B within multiple CCCGs creates additional challenges because conventional models (i.e., the alpha factor model [80]) may provide two different probabilities for the same CCF event [82]. For example, some CCF models may determine $P(CCF_{AB})$ from CCCG1 to be different than $P(CCF_{AB})$ from CCCG2. This is because conventional models incorporate the CCCG size as part of their evaluation process and consider combinations of failures between the components of the CCCG. If modeling is performed using a program such as the SAPHIRE [30], having a single component within multiple CCCGs may lead to double counting of failure events. Ma et al. address this issue further and suggest using the largest CCCG that is reasonable [82]. However, this solution requires the analyst to ignore the potential asymmetry of the coupling factors. They suggest a second option may be to select which value of the duplicate failure events is appropriate. Additional examples exist that allow components to be part of multiple CCCGs, such as when each CCCG represents a unique failure mode [82].

In order to directly consider subtle differences in coupling mechanisms, there are two approaches for forming the CCCGs. The first, as mentioned earlier, forms unique CCCGs for each shared set of coupling

factors and may require some components to be part of multiple CCCGs. The second approach forms CCCGs that allow for some variation in the coupling mechanisms—for example, from Figure 46, a single CCCG that contains A, B, and C, but allows for differences in $P(CCF_{AB}), P(CCF_{BC})$, and $P(CCF_{AC})$ directly, rather than assume they all equal $P(Q_2)$. The second approach requires an asymmetric model to directly account for these differences within the CCCG. Models for asymmetry and models that allow components to be part of multiple CCCGs have been addressed by several publications. Rasmussen and Kelly proposed a method to deal with asymmetric failure probabilities within the context of the basic parameter model [83]. In 2012, Kančev and Čepin proposed a modification of the beta-factor model that allows components to be assigned to multiple CCCGs based on their coupling factors [84]. O'Connor and Mosleh proposed a partial alpha factor model and a Bayesian approach (the general dependency model); an extension to the alpha factor model, the partial alpha factor works to explicitly model coupling factors between components [85]. The general dependency model relies on a Bayesian network to account for three parameters—a cause condition probability, component fragility, and coupling factor strength [81]. In 2020, Higo et al. developed a method to account for the combined influence of asymmetric and symmetric CCF probabilities by assessing the degree of shared coupling factors [86]. This work was later refined by combining with a gamma factor model to express inter-unit CCF probability [87].

The challenge with these methods is their dependence on proprietary data for model parameters. Far less data is available for software-based CCFs, than for analog CCFs which challenges the application of these recent innovations. In addition, those methods that account for qualitative differences in coupling mechanisms (e.g., [81]) rely on data that may not exist for newly designed software systems. The goal of our work is to quantify software CCFs given minimal data while also considering the influence of software attributes on coupling mechanisms. Given most asymmetric models require data that is unavailable for software, we will forgo the formation of asymmetric CCCGs and instead rely on approach that considers qualitative information for CCF modeling while also allowing components to be part of multiple CCCGs.

This work proposes a method for modeling software CCF given: (1) a lack of operational and CCF data for defining software CCF model parameters and (2) the need to model single components as part of multiple CCCGs simultaneously. The model best suited for a limited-data scenario may be the one requiring the fewest parameters. In this case, the modified beta-factor model by Kančev and Čepin (referred to as the modified BFM in our work) is demonstrated for software CCF analysis.

4.5.2 A Modified Beta-Factor Method for the Analysis of Software Common Cause Failures

This section is focused on answering two needs for modeling software CCFs. The first part of this section discusses an approach for modeling components as part of multiple CCCGs simultaneously as provided by the modified BFM. The second half details the innovative application of the modified BFM for software CCF analysis. Specifically, this section details our innovations for addressing the lack of operational and CCF data typically used to define model parameters.

The modified BFM, as its name suggests, is based on the beta-factor model [84]. The beta-factor model is one of the oldest CCF models and assumes that a total failure probability (Q_t) of a component is a contribution of independent (Q_I) and dependent (Q_D) failures; the dependent failure probability is given as a fraction (i.e., β) of the total failure probability (Q_t) of the component as observed in Equation (20). Likewise, the independent failure is also a function of β. The beta-factor model implements the symmetry assumption such that all the components within a CCCG fail together according to the dependent (i.e., CCF) probability defined by beta. The model does not account for combinations of failures within a CCCG [80]. The beta-factor model applied to a CCCG of A, B, and C will only find CCF_{ABC}. Therefore, the only way to consider a CCF of two components is to assign them their own CCCG. This is the basis of the modified BFM. Our work assumes that the potential for combinations of failures with the CCCG is

largely dependent on the existence of subtle differences in the coupling mechanisms. Hence, to account for any distinct CCFs, we rely on coupling factor-based CCCGs.

$$Q_t = Q_I + Q_D \tag{20}$$

$$Q_D = \beta Q_t \tag{21}$$

$$Q_I = (1-\beta) Q_t \tag{22}$$

The modified BFM is designed to allow components to be members of multiple CCCGs [84]. Like the beta-factor model, the modified BFM assumes the total failure probability/rate of a component is the summation of independent and dependent failures. Equation (23) shows the basis of the modified BFM, which is that the total dependent failure consists of the contribution of each CCCG failure. Each CCCG is assigned a group beta (β_w) that represents the contribution of that CCCG to the total failure probability. Equation (27) shows the independent failure probability in terms of each CCCG beta and total failure probability.

$$Q_D = P(CCCG_1) + P(CCCG_2) + \cdots P(CCCG_w) \tag{23}$$

$$P(CCCG_w) = (\beta_w) Q_t \tag{24}$$

$$\beta_t = \sum_{1}^{w} (\beta_w) \tag{25}$$

$$Q_D = Q_t \sum_{1}^{w} (\beta_w) \tag{26}$$

$$Q_I = (1-\beta_t) Q_t = \left[1 - \sum_{1}^{w} (\beta_w)\right] Q_t \tag{27}$$

Some advantages of this method include its ease of application, its consideration of CCCG-specific coupling factors, and its ability to account for multiple CCCGs directly. Double counting is avoided because the model assumes that CCFs represent the failure of each component within the CCCG and no other sub-combinations. For example, given two CCCGs (e.g., components A, B, and C for CCCG1 and A and B for CCCG2), there will be no chance of counting $P(CCF_{AB})$ twice because $P(CCF_{AB})$ is only evaluated for CCCG2. The modified BFM, like most methods, requires reference data to determine each CCCG failure probability/rate. Like other ratio models, the quantification of its parameters can be challenging for a limited-data scenario. The modified BFM is limited to identical components with identical total failure probabilities. If the Q_t for the components within a CCCG are not identical, depending on the Q_t selected for Equation (24), there will be differing values for the same CCFs. Sources [83] and [77] provide support for this scenario.

An additional limitation can occur if the total beta, shown by Equation (25), exceeds unity. If this happens, then the summation of dependent failures will exceed the total failure probability. To account for this issue, Kančev and Čepin indicate a possible solution is to normalize the CCCG beta factors such that they sum to unity while maintaining their relative magnitudes. The second and third options include normalizing by the largest CCCG beta or using weight factors for each CCCG, respectively [84]. It is best to select the option which matches model assumptions (e.g., the first option will work better for software CCF low-diversity systems, because it is expected that dependent software failure will exceed the independent software failure probability). Despite its known limitations, this work will employ the modified BFM for the quantification of CCFs because it works directly for the multiple CCCG scenario.

The next challenge is defining the model parameters. The emphasis of the current work is the limited-data scenario that naturally requires some form of expert elicitation. For elicitation, it is desirable to

consider qualitative defenses against CCFs [77, 88]. There are at least two methods presented in literature that express the elicitation of the beta parameter without the use or dependence on operational data. These two methods, both of which are called "partial beta methods," develop beta from a combination of partial attributes; one employs an additive scheme to find beta [77], while the other a multiplicative scheme [88].

The first method, called partial beta factor-1 (PBF-1) in our work, was developed on the claim that dependent failures could not be determined without an engineering assessment of that system's defenses for such failures [88]. An assessment is made according to 19 defenses (e.g., functional diversity, and maintenance), where each defense receives a partial beta value (i.e., β_i between zero and one, where a zero score indicates a high defense against CCF). The product of the 19 scores is then used as the beta factor for the system. This multiplicative scheme may tend to predict small values for beta. For example, if 18 of the defenses are given $\beta_i = .99$, the CCF likelihood for the system should be high. However, the remaining defense (β_{19}) can dominate the system, resulting in an improper score for the system beta (e.g., if $\beta_{19} = .1$ and $\beta_{1-18} = .99$, then $\beta = .083$). Further complications could arise if additional defense categories are added. Ultimately, PBF-1 may underpredict dependent failures.

The second method, called partial beta factor-2 (PBF-2), does not actually use partial betas, rather the method uses a collection of sub-factors that contribute to an overall beta score [77]. Humphreys' method was later modified by Brand [89] and served as a foundation for a hardware CCF model used in the International Electrotechnical Commission (IEC) 61508 [90]. The PBF-2 was founded by asking, "what attributes of a system reduce CCFs?" [77]. These attributes, called sub-factors, are shown in Table 40. Each sub-factor was weighted by reliability engineers for their importance. The method requires the analyst to assign a score (e.g., A, B, and C) for each sub-factor. An "E" indicates a component is well-defended against CCFs (i.e., A= poor and E= ideal). The sub-factor names alone are not sufficient for assessing each sub-factor; therefore, readers are advised to visit the original source material for scoring guidance. Beta, given by Equation (28), is a function of the assigned sub-factor scores and the denominator d. The model was arranged such that the upper and lower limits for beta correspond with dependent failure values reported in literature [77]. The limits are ensured by the sub-factors and d given in Table 40. The beta value determined by this method was intended to be used with beta-factor model; but in this work, it will be used with the modified BFM.

$$\beta = \frac{\sum(Sub - factors)}{d} \tag{28}$$

Table 40. Beta-factor estimation table for hardware.

Sub-factors	**A**	**A+**	**B**	**B+**	**C**	**D**	**E**
Redundancy (& Diversity)	1800	882	433	212	104	25	6
Separation	2400		577		139	33	8
Understanding	1800		433		104	25	6
Analysis	1800		433		104	25	6
MMI	3000		721		173	42	10
Safety Culture	1500		360		87	21	5
Control	1800		433		104	25	6
Tests	1200		288		69	17	4
Denominator for Equation (28), $d = 51000$.							
Note: The current work relies on an automatic calculation that provides slightly different table values than those given in the source material. The original derivation indicates that scoring an "A" for each sub-factor will result in 0.3 for the beta factor [77]. The current table provides 0.300 while the original provides 0.302. The difference is negligible, so this work employs the automated calculation for convenience.							

PBF-2 provides a convenient and structured determination of beta associated with the hardware failure of DI&C components, yet only minimal consideration is provided for software [77]. In fact, some methods (e.g., IEC 61508) prefer to provide qualitative approaches to avoid or control software failures

[91]. In contrast, this work emphasizes the quantification of both hardware and software failures. As mentioned, CCFs are conditional on a shared root cause and coupling factor. Within the context of highly redundant DI&C systems and low instances of software diversity, it is anticipated that CCFs should represent a significant portion of the software failure. Redundant components share application software failure by nature of their common (i.e., identical) software.

Software failure occurs by the activation of latent defects (e.g., deficiencies from coding errors, installation errors, maintenance errors, setpoint changes, and requirements errors). Activation of latent defects is a result of certain operational conditions (i.e., trigger events) [63]. Trigger events act as software inputs, without which there would be no fault activation and, ultimately, no failure. A software CCF will result from a shared root cause (i.e., a shared trigger event and a defect) leading to the failure of two or more components by means of a coupling mechanism. Coupling mechanisms influence how a trigger event and/or a defect is shared by multiple components. As an example, consider that a software developer (i.e., a coupling mechanism) introduces a shared defect in redundant controllers allowing a trigger event to cause a CCF. In contrast, a maintenance procedure (i.e., a coupling mechanism) may shut down half of a system thereby creating a condition for a trigger event to affect only the active components.

Given a group of redundant software components, variations in their operating conditions may lead to some, but not all, components failing together. Variations in the operational environment of otherwise identical components may result from differences in maintenance staff, inputs variables, etc. In other words, subtle differences in coupling mechanisms may lead to unique combinations of CCFs. Thus, it is essential to consider software-based coupling mechanisms when assessing the potential for CCFs within a DI&C system.

To account for software features, PBF-2 was modified in two ways: (1) the model was adjusted to increase the upper and lower limits of beta (i.e., 0.001–0.999), allowing for greater applicability to low-diversity software systems; and (2) the sub-factor weights were changed to emphasize software-centric features. It is understood that diversity affects CCFs [63]. Consequently, the sub-factors that influence diversity were weighted heavily. As an example, the adjusted model emphasizes the introduction of software faults and coupling mechanisms by placing greater weight on those defenses that pertain to human interaction and the diversity of software. Subtle variations in the coupling mechanisms create quasi-diverse components, ultimately influencing the potential for CCFs. Table 41 shows the adjustments made to PBF-2 to account for software. It, along with Table 40, are used to define the beta factors for software and hardware failures, respectively. Sub-factors are scored according to the guidance given by [89] with some additional considerations for software; the scoring for Redundancy (& Diversity), Separation, and Testing have been modified.

Table 41. Beta-factor estimation table for software.

Sub-factors	**A**	**A+**	**B**	**B+**	**C**	**D**	**E**
Redundancy (& Diversity)	23976	10112	4265	1799	759	135	24
Input Similarity	23976	10112	4265		759	135	24
Understanding	7992		1422		253	45	8
Analysis	7992		1422		253	45	8
MMI	11988		2132		379	67	12
Safety Culture	6993		1244		221	39	7
Control	4995		888		158	28	5
Tests	11988		2132		379	67	12
Denominator for Equation (28), $d = 100000$.							

The sub-factor guidance for Redundancy (& Diversity) is based mostly on the configuration of redundant components (e.g., 1/2, 2/3 required for success). This logic works well with the behavior or hardware components. For hardware considerations, increasing the number of redundant components can

benefit reliability. In the very least, the assumption is that concurrent hardware failure in many units is less likely than in a few. However, software does not fail in the same random way that hardware can. Software failure is driven by fault activation. Thus, variations of redundant configurations of software make little difference, given each redundant software component receives the same input and has the same internal defects. Simply, the configuration of redundant copies is less significant than controlling the degree of internal similarity between redundant units by introducing diversity. Thus, the subfactor guidance for Redundancy (& Diversity), when directed for software consideration, should focus on the sources of internal diversity between redundant components, rather than logical configurations. Table 42 was developed to provide a structure for this consideration.

Table 42. Sub-factor Guide for Redundancy (& Diversity) in Software CCCG.

Score	**Guidance**
A	No Diversity, δ=0
A+	Very low diversity, 0<δ≤0.167
B	low diversity, 0.167<δ≤ 0.333
B+	Intermediate diversity, 0.333<δ≤0.50
C	Intermediate/high diversity, 0.50<δ≤0.667
D	high diversity, 0.667<δ≤0.833
E	Very high diversity, 0.833<δ≤1
Internal diversity $\delta=(h-\theta)/(H-\theta)$ H= maximum diversity of attributes within the CCCG, $H=m\theta$ m= the number of components within the CCCG θ= number of attributes (e.g., software language, process model, control algorithm) h= number of unique attributes states in the CCCG	

The sub-factor guidance for Separation was changed to Input Similarity. Physical separation alone does not influence software failure unless there is consideration for how that physical separation changes the operational conditions of the components. Whereas the Redundancy (& Diversity) sub-factor considers the degree of internal similarity, the Input Similarity sub-factor considers the degree to which redundant software share external and input similarity. The degree of input similarity provides an indication of how similar the external inputs are for each member of the CCCG. An input ratio (R) provides an indication of approximately how many of the inputs to the CCCG are shared. Additional consideration has been made to account for any degree of diversity that may influence the input (e.g., analog vs digital signals). Guidance for scoring the Input Similarity is shown in Table 43.

Table 43. A sub-factor guide for input similarity in software CCCG.

Score	**R=0**	**0 < R ≤ .5**	**.5 < R < 1**	**R ≥ 1**	**Zero Diversity**	**Partial Diversity**	**Complete Diversity**
A	X				X	X	X
A+		X			X	X	X
B			X		X		
C			X			X	X
D				X	X	X	
E				X			X
The input ratio (R) is defined: $R=(s-1)/m$ for $s=1$ and $R=s/m$ for $s>1$ where, $m=$ the number of components within the CCCG, and $s=c/i$. $i=$ (number of inputs to the CCCG)/m and c = number of immediate sources.							

This work presents a method for performing CCF analysis on DI&C systems given limited data by integrating the modified BFM and PBF-2. The method relies on the modified BFM to allow components to be part of multiple CCCGs, and PBF-2 defines beta factors for each CCCG. The hybrid method

provides a means to overcome limitations of conventional methods. A formalized process that relies on the modified BFM and PBF-2 is shown in Figure 47, which has been demonstrated in [28] [92]. The subsequent section will demonstrate this process as with a case study.

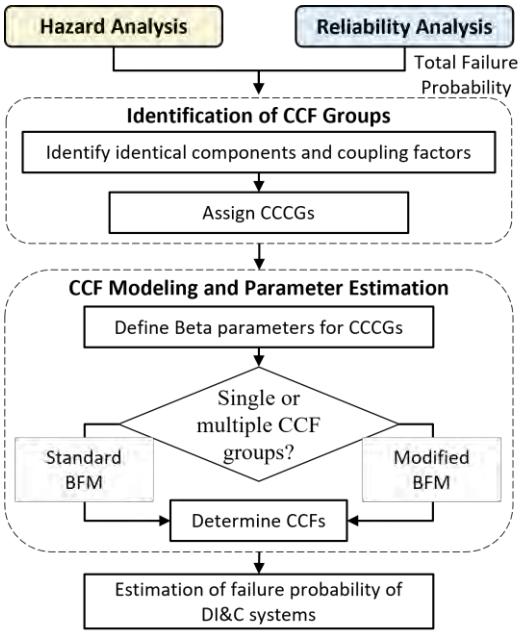

Figure 47. Flowchart for Software CCF Modeling and Estimation.

4.5.3 Case Study: Human-System Interface for Nuclear Power Plants

4.5.3.1 Qualitative Evidence

In this case study, only the QIAS-P HSI system and its divisional redundancies were considered. Due to overall complexity of the system, RESHA is used to construct the integrated fault tree to specify relevant losses. Relevant software failure probabilities will be quantified using ORCAS. Control diagram of one redundant division of the HSI with manual reactor trip actuation pathway can be found in Figure 20. In this system, there exists only one controller and one control action possible relevant to reactor trip, whether the plant operator manually trips the reactor or not. The specific UCA relevant to the chosen top event is thus, "Operator does not engage undervoltage trip breakers (A1, A2, B1, or B2) under abnormal operational conditions (UCA-A) due to interface confusion." The cause of interface confusion exists within the QIAS-P subsystems. For instance, the "RCSM alarm does not alert the operator when the available coolant saturation margin decreases below the setpoint (UIF-A)." This could be due to an internal software defect, where the setpoint was incorrectly set, or an external issue, where one of the dependent sensors or intermediate processors has failed. The exact sequence of failures in the IFP depends solely on how the control loop is implemented.

In Table 44, the identified potential software UIF failure modes are presented. The human-related UCAs are to be included in future HRA work. Based on the top event, the concern for alarms systems is if they fail to trigger when the reactor is in a degraded state (i.e., anomalously high core temperature). This is a UIF-A, failure to trigger under specific hazardous context, leading to the top event. It is assumed that if any of the alarms fail to trigger, the operator will not trip the reactor in time. For parameter calculators, the primary concern is accuracy of the output value. For example, the RCSM calculator takes multiple different parameters and calculates the remaining coolant margin until saturation. If an error causes the calculator to show significant margin until saturation, the operator may believe that the reactor is still safe when it may not be. It is assumed that if any of the parameter calculators do not accurately reflect the true

reactor state, then the operator will not trip the reactor in time. Table 44 lists all potential software UIFs failure modes for the representative HSI system.

Table 44. Potential software UIFs failure modes for the representative HSI system.

UCA/UIF	Location	Event Description
D	ICC Calculator	Div. A calculated core cooling value is significantly too low when reactor is transitioning to degraded state
D	HJTC temp. Calculator	Div. A calculated core temperature value is significantly too low when reactor is hotter than anticipated
D	RVL Calculator	Div. A calculated coolant level is significantly too low when reactor is transitioning to degraded state
D	RCSM Calculator	Div. A calculated saturation margin is significantly too high when real coolant is near liquid saturation limit
D	CET Calculator	Div. A calculated coolant temperature is significantly too low when reactor is hotter than anticipated
A	ICC Alarm	Div. A alarm fails to trigger when cooling is inadequate
A	HJTC Alarm	Div. A alarm fails to trigger when core temperature exceeds upper limit
A	RVL Alarm	Div. A alarm fails to trigger when coolant level falls below lower limit
A	RCSM Alarm	Div. A alarm fails to trigger when saturation margin falls below intended limit
A	CET Alarm	Div. A alarm fails to trigger when coolant temperature exceeds upper limit

* Full names of the acronyms can be found in the Acronyms list on Page xiii.

4.5.3.2 Discovered Defects

In [92], various software defects were detected and classified based on operational data from NUREG/CR-7042 [93]. In the report, software-operating data was collected over a period of 281 months from multiple NPPs on the two separate microprocessors (μp) and on the communications module (CP). The functionality of the μp is to process, augment, and collect sensor information and conduct internal calculation. This is similar to the sensors and PLCs implemented in the QIAS-P system. For the communication processor, its role is to handle alarm notification and parameter monitoring to the main control room. While no individual component represents the CP in the APR-1400 HSI system, it is assumed each parameter calculator and alarm system has an internal communication processor. Due to the similarity of the components explored in NUREG/CR-7042 with the components in the APR-1400 HSI system, the report data is used to explore how defect data from similar systems can be used to estimate software failure probability.

In [93], test plans for each component were examined, and the identified requirement from the SRS and SDD were traced. In addition, separate verification and fault injection tests were conducted on each module. In total, 26,799 tests were performed over the entire study period. From testing, 23 software defects were discovered and believed to be credibly significant (Table 46). From left to right, column 1 is the defect identification tag followed by the component type, description of the defect, and the defect classification based on anticipated resolution. Using Table 17, the discovered defects are grouped together, and the average probability of occurrence per class was estimated as shown in Table 45. Recall that the probability of occurrence is the probability that the specific defect class will be detected after 1 additional hour of testing effort.

Table 45. Grouped ODC defect classes and tags.

Defect Class	# of Defects	Defect Tag ID	Average Probability
Algorithm	1	D8	3.73E-5
Assignment	1	D17	3.73E-5
Function	8	D1, D2, D3, D5, D6, D7, D12, D13	2.98E-4
Interface	1	D14	3.73E-5
Checking	11	D9, D10, D11, D15, D16, D18 – D23	4.10E-4
Timing	1	D4	3.73E-5

Table 46. Defect tags, location, description, and classification.

Tag	Location	Defect Description	Defect Classification
D1	μp1	Does not increment the EEPROM test counter if the Tuning in Progress flag set	Function
D2	μp1	Algorithm does not detect coupling faults between two address lines	Function
D3	μp1	Does not copy contents of table to Dual Port RAM	Function
D4	μp1	Program does not give up semaphore	Timing
D5	μp2	Algorithm does not detect coupling faults between two address lines	Function
D6	CP	Algorithm does not detect coupling faults between two address lines	Function
D7	μp1	Check cannot detect coupling failure but only stuck at high or low failures.	Function
D8	μp2	If trip condition is calculated, the logic will force a different incorrect calculation for the second trip calculation.	Algorithm
D9	μp2	Processor has 16-bit address, but only 13 bits are examined. Remaining three bits are not tested and may cause error.	Check
D10	CP	Check cannot detect coupling failure but only stuck at high or low failures.	Check
D11	μp1	Check does not cover all address lines	Check
D12	μp1	During Power-On, processors cannot detect missing inputs and does not indicate fatal error	Function
D13	μp1	Inputs changed from continuous to discrete caused spurious trip	Function
D14	μp1	Dual port ram detects the wrong module ID	Interface
D15	μp2	Online RAM test is incomplete	Check
D16	μp2	Online EEPROM failure not identified as fatal failure	Check
D17	CP	Cannot initialize variable successfully	Assignment
D18	μp1	Cannot detect incorrect value of the variable SA_TRIP_1_DEENRGZE	Check
D19	μp1	Cannot detect fAnalog_Input_6	Check

D20	μp2	Cannot detect incorrect value of the variable Trip_condition	Check
D21	μp2	Cannot detect incorrect value of the variable AIN	Check
D22	μp1	Cannot detect incorrect value of the variable chLEDs_Outputs	Check
D23	μp2	Cannot detect incorrect value of the variable have_dpm	Check

4.5.3.3 Quantitative Evidence

- **Hardware Failure Probability**

Values for PLC and sensors failure are derived from [94], [95], and [96]. In Table 47, the values for hardware failure are presented.

Table 47. Hardware total failure probability for QIAS-P digital components.

Hardware Name	**Failure Probability**
Heated-junction thermocouple sensor	1.05E-07
Heated-junction thermocouple sensor controller	2.21E-06
Core exit thermocouple	1.05E-07
Signal conditioner	1.00E-06
Analog to digital converter	7.13E-06
Parameter calculator	2.21E-06
Parameter alarm	2.21E-06

- **Software Failure Probability**

Using the probabilities estimated in Table 45 and the UCA/UIF conditional probabilities in Table 24, the probabilities of each UCA/UIF software failure mode can be estimated. The last row of Table 48 is the total probability that specific UIF software failure mode can occur. These estimated values can now be placed in the integrated fault tree to estimate the top event probability.

Table 48. Estimated average failure probability per UIF class based on defects data.

Defect	**UIF – A**	**UIF – B**	**UIF – C**	**UIF – D**	**Total**
Algorithm	1.23E-05	5.11E-06	1.26E-05	7.24E-06	3.73E-05
Assignment	1.06E-05	2.42E-05	2.13E-06	4.10E-07	3.73E-05
Function	1.25E-04	1.07E-04	9.88E-05	7.83E-05	4.10E-04
Interface	1.15E-04	7.15E-05	7.15E-05	4.02E-05	2.98E-04
Checking	1.25E-05	1.82E-05	4.40E-06	2.20E-06	3.73E-05
Timing	7.05E-06	2.01E-06	2.32E-05	5.04E-06	3.73E-05
Total	2.82E-04	2.28E-04	2.13E-04	1.33E-04	8.57E-04

- **Software Common Cause Failure Probability**

To determine the beta factor, the score value column is summed up and divided by a scaling value of 100,000. The beta factor based on the modified UPM for the HSI system was determined to be 0.563. To determine the probability of CCF, Equations (20), (21) and (22) used along with the single UCA/UIF failure probability determined in Table 48, where P_{total} is the total event probability, P_{single} is the

singular event probability, P_{CCF} is the probability that multiple components fail simultaneously, and β is the coupling strength between components (also known as beta factor). In Table 50, the total event probability (P_{total}), single event probability (P_{single}), and common cause probability (P_{CCF}) are shown for each UCA/UIF failure mode.

Table 49. Beta-factor scoring based on environmental and development conditions.

Metric	Score	Score Value	Score Reasoning
Diversity	A	23976	QIAS-P Division A&B are identical
Input Similarity	A	23976	Division A&B racks are physically isolated
Understanding	A	7992	Less than 10 operating years of software experience
Analysis	C	253	Reliability studies have been conducted on QIAS-P development
Man-Machine Interface	D	67	Tests and checks exist for QIAS-P software
Safety Culture	E	7	QIAS-P simulations in normal and emergency conditions exist
Control	D	5	Limited access to hardware modules and interfaces
Tests	D	67	Detailed checks have been performed for a reasonable period of time

Table 50. Calculated event probabilities by UCA/UIF class.

UCA/UIF class	Single Event Prob. P_{single}	CCF Event Prob. P_{CCF}	Total Event Prob. P_{total}
A	1.23E-04	1.59E-04	2.82E-04
B	9.98E-05	1.29E-04	2.28E-04
C	9.30E-05	1.20E-04	2.13E-04
D	5.83E-05	7.51E-05	1.33E-04

4.5.4 Case Study - RTS CCF Estimation

This case study describes the quantification of the CCFs found in the automatic trip function of a four-division digital RTS. The software failure probabilities of the BPs were quantified by BAHAMAS. Division-based sensor signals are sent to the BPs, which determine whether a trip is needed. When required, trip signals from the BPs are sent to each of the divisions' local coincidence LPs. The LPs vote on the incoming trip signals and send the output via digital output modules (DOMs) to selective relays, which again vote on the trip signals. The outputs of the selective relays pass through undervoltage trip devices (e.g., RTB-D1-UV) and activate the undervoltage reactor trip breakers (e.g., RTB-A1). The correct combination of breakers results in a reactor trip. Diverse trip mechanisms (e.g., shunt trip devices like RTB-DA-ST) via the diverse protection system (DPS) and manual trip mechanisms via the main control room (MCR) or the remote shutdown room (RSR) are not part of the case study. Table 51 provides the list of components for which failure rates need to be quantified. In this work, the only components shown in Figure 22 to contain application software are the BPs and LPs, both of which are programmable logic controllers. Evaluation of the software CCF values follows the method described in the previous section.

Table 51. Total hardware and software failure probabilities for CCF case study.

Components	Hardware failure	Total Hardware failure probability	Software failure	Total Software failure probability

BPs	YES	4.00E-5	YES	2.077E-4
LPs	YES	6.48E-5	YES	2.077E-4
Digital Output Modules	YES	1.64E-5	N/A	N/A
Selective Relay	YES	6.20E-6	N/A	N/A
RTB-UV device	YES	1.70E-3	N/A	N/A
RTB-Shunt device	YES	1.20E-4	N/A	N/A
RTBs	YES	4.50E-5	N/A	N/A
All hardware values came from [97].				

* Full names of the acronyms can be found in the Acronyms list on Page xiii.

The details of the RTS were based on limited publicly available information [22]; consequently, some assumptions were made to complete the case study: (1) there is no diversity in the software; (2) all hardware components are not diverse (unless otherwise specified); (3) installation teams and maintenance teams are assumed identical for each CCCG; (4) each set of identical components that are part of the same CCCGs have the same total failure probabilities. For convenience, the failure probability of the BPs and LPs are assumed to be identical.

The first step shown in Figure 47 is to assign the CCCGs after identifying the identical components and their coupling factors. There are eight identical BPs in the RTS, two per division. They each have an identical function and are assumed to share the same features, except for their installation location. All BPs share identical coupling factors, except for location, resulting in two CCCGs. One CCCG is based on shared function, hardware, software, and manufacturer. The second CCCG considers location. Table 52 shows the CCCGs identified for the BPs RESHA. Location creates an operational environment that is unique for the BP software. Despite having identical software, input from division-specific sensors creates the potential for the BPs to have division-specific CCFs associated with their operational conditions.

Table 52. CCCGs for the BPs.

CCCGs		**Coupling Factors**
1	All BPs	Function, Hardware, Software, & Manufacturer
2	Division A: BP1, BP2	Location (Division A)
3	Division B: BP1, BP2	Location (Division B)
4	Division C: BP1, BP2	Location (Division C)
5	Division D: BP1, BP2	Location (Division D)

The next step from Figure 47 is to define the beta-factor parameters. Each CCCG receives a score for each sub-factor category. Sub-factors are scored according to the guidance provided in [89], with additional provisions for software as indicated in the preceding section. For example, CCCG 1 for the BPs receives an A+ for input similarity. Specifically, CCCG 1 consists of eight BPs (i.e., m= 8). Each division receives its own sensor input that is shared by its BPs (i.e., s= 4). The result is $R=s/m=0.5$ (i.e., A+ from Table 43). Table 53 shows the sub-factor scores for the BPs of CCCG1 and the calculation for beta based on Equation (28). The BPs for CCCGs 2–5 share the same qualitative features and receive beta-factor scores of 0.123 and 0.568 for their hardware and software, respectively.

Table 53. Sub-Factor Scores for BPs CCCG 1 (All BPs CCF).

Sub-factors	**Hardware**		**Software**	
Redundancy (& Diversity)	B+	212	A	23976
Separation/Input Similarity	E	8	A+	10112
Understanding	A	1800	A	7992
Analysis	D	25	D	45
MMI	C	173	C	379
Safety Culture	E	5	E	7

Control	D	25	D	28
Tests	C	69	C	379
Beta for the CCCG	$\beta_{HD1} = 0.045$		$\beta_{SW1} = 0.429$	

The next step from the CCF modeling flowchart is to determine the CCFs. The BPs have multiple CCCGs; therefore, the modified BFM is used. For example, Division A, BP1 is found in two groups, CCCG1 and CCCG2, as shown in Table 52. Equations (20) and (23) - (27) are used to find the independent and dependent failures of the BPs. The results of the CCF analysis are shown in Table 54. Note that RACK, DIVISION, and ALL correspond to the CCCG categories, while INDIVIDUAL corresponds to individual component failure. The CCCG ALL contains all the identical components within the system of interest. The given CCCG categories are not shared by all components; hence, there are no RACK CCCGs for the RTBs. Regarding the results, there is a difference between the software and hardware CCCGs of the LPs. The hardware CCCGs for the LPs are separated by location, just like the BPs. However, the potential for DIVISION- and RACK-level CCFs are precluded from consideration because there is nothing to distinguish them from the CCCGs representing all LPs; according to the case study, each LP has the same software and receives the same inputs. By contrast, the BPs have the potential for input variation amongst divisions. Thus, the BPs have DIVISION-level software CCCGs, but the LPs do not. The results show that our method allows predicted software CCF to represent a larger failure probability than independent failure which matches our assumptions for a high-redundancy low-diversity software system.

Table 54. Hardware and software failure probability for RTS components.

Component	**INDIVIDUAL**	**RACK**	**DIVISION**	**ALL**	**Total**
BPs-Hardware	4.000E-05	N/A	5.943E-06	2.187E-06	4.813E-05
LPs-Hardware	6.480E-05	1.076E-05	7.647E-06	3.961E-06	8.717E-05
DOMs	1.640E-05	1.706E-06	1.015E-06	1.983E-07	1.932E-05
Selective Relay	6.200E-06	N/A	6.073E-07	7.059E-08	6.878E-06
RTB-UV device	1.700E-03	N/A	N/A	1.763E-05	1.718E-03
RTB-Shunt device	1.200E-04	N/A	N/A	1.244E-06	1.212E-04
RTB RTSS2	4.500E-05	N/A	N/A	1.944E-06	4.694E-05
BPs-Software	6.207E-07	N/A	1.179E-04	8.914E-05	2.077E-04
LPs-Software	8.976E-05	N/A	N/A	1.179E-04	2.077E-04

* Full names of the acronyms can be found in the Acronyms list on Page xiii.

4.5.5 Discussions

This work introduces a method for modeling software CCFs. A software CCF will be the result of a shared root cause (i.e., a trigger event and a latent fault) leading to the failure of two or more components by means of a coupling mechanism. Given a group of redundant software components, variations in their operating environments may lead to some, but not all, components failing together. Variations in the operational environment may result from differences in maintenance staff, input variables sources, and installation teams. These subtle differences may lead to unique combinations of CCFs. Thus, it is essential to consider software-based coupling mechanisms when assessing the potential for CCFs within a DI&C system. When a group of components share coupling mechanisms, they form a CCCG. For most analyses, the components that belong to a CCCG do not belong to any other groups. This is because the components have no other coupling factors to share with components outside their existing group. When components can be grouped into multiple CCCGs (e.g., based on software-operating environments), it becomes difficult to model their failure probabilities using conventional methods.

The chosen method employs the modified BFM and PBF-2 for modeling software CCFs by introducing modifications to PBF-2 for defining software-specific model parameters. The modified BFM

was selected because it conveniently models components with multiple CCCGs. Normally, CCF methods rely on historical data or experience to define model parameters. However, limited data associated with novel designs requires a solution for quantifying model parameters. Innovations to PBF-2, together with the modified BFM, allow for a successful quantification process for the multiple CCCGs under a limited-data scenario. Several aspects of CCF modeling remain for future work. First, PBF-2 defines model parameters by considering the quality of a component's defenses against CCF. The method only considers eight sub-factors for assessing beta. There may yet be additional software-specific qualitative attributes to refine PBF-2. In addition, future research may provide an enumeration of software-specific coupling factors to aid the selection of software CCCGs. The modified BFM can also be improved. In its current form, the method, as with other ratio-based methods, is limited to similar components; future work may provide guidance for CCFs between non-identical components.

In conclusion, the method developed for this work provides a convenient means to quantify software CCF given a lack of operational data and allow components to be part of multiple CCCGs simultaneously. Most of existing CCF methods rely on historical data, or experience, to define model parameters; however, the absence of software failure data, such as for novel designs, demands an alternative. There are two options for assessing CCFs under limited data. The first option employs conservative or bounding assessments (e.g., assuming all components fail). The second option relies on expert judgment. The LWRS-developed framework informs the judgment process by employing subfactors (e.g., analysis, training, testing and safety culture) to determine strategically and qualitatively how well a CCCG is defended against potential CCFs. The works by Humphreys [77] and Brand [89], which have been used by the IEC [90] for hardware-based CCF modeling, serve as the technical foundation for implementing the chosen subfactors as part of this work for software-based CCF modeling. Due to lack of software-based CCF data, the thorough verification and validation of this CCF modeling and estimation method has not been performed. A detailed guideline of the method is provided in this report to reduce the user effect. And a group of experts, rather than a single expert, is suggested to work on the scoring process. Even with uncertainty and subjectiveness from user effect, quantifying important software-based CCFs can be very useful for DI&C designers to make informed decisions on diversity design and optimization, and help reduce the cost of system development. Future collaborations with industry partners may afford our team the opportunity to validate and improve this method.

5. CONSEQUENCE ANALYSIS OF A GENERIC PWR WITH ADVANCED HSSSR DI&C SYSTEMS

This section documents the consequence analysis of a generic PWR PRA model with improved digital RTS and ESFAS FTs, which quantitatively evaluates how the previously identified and quantified CCFs affect the overall plant safety Section 5.1 describes the generic PWR PRA model (developed using SAPHIRE tool [30]). In Section 5.2, the scenario to be analyzed is introduced as well as the original ET models for these scenarios including a FT for the failure of an analog RTS and one CCF basic event for the ESFAS failure. Section 5.3 compares the original FTs for analog RTS and ESFAS and the new FTs for digital RTS and ESFAS. Results for consequence analysis of these selected ET models are discussed in Section 5.4.

5.1. Generic PWR SAPHIRE Model

A generic internal events PRA model was developed at INL using SAPHIRE 8 for a typical PWR plant for the accident scenario analysis with various initiating events, which has been applied for different purposes including plant-level scenario-based risk analysis for enhanced resilient plant (ERP) during station black-out (SBO) and loss-of-coolant accident (LOCA) [98], risk-informed analysis for an ERP with accident tolerant fuel (ATF), optimal use of diverse and flexible coping strategy (FLEX), and new passive cooling systems [99] [100]. There are 24 ETs included in this generic PWR SAPHIRE model, as listed below:

- EQK-BIN-1: Seismic Initiator (0.05 - 0.3g)
- EQK-BIN-2: Seismic Initiator (0.3 - 0.5g)
- EQK-BIN-3: Seismic Initiator (> 0.5)
- INT-ISL-HPI: ISLOCA IE 2-CKV HPI interface
- INT-ISL-LPI: ISLOCA IE 2-CKV LPI interface
- INT-ISL-RHR: RHR pipe ruptures
- INT-LLOCA: LARGE LOCA
- INT-LOACA: LOSS OF VITAL 4160V AC BUS A
- INT-LOCCW: LOSS OF CCW INITIATING EVENT
- INT-LOCHS: LOSS OF CONDENSER HEAT SINK
- INT-LODCA: LOSS OF VITAL 125 VDC BUS A
- INT-LODCB: LOSS OF VITAL 125 VDC BUS B
- INT-LOMFW: LOSS OF MAIN FEEDWATER
- INT-LONSW: LOSS OF NSW COOLING INITIATING EVENT
- INT-LOOPGR: LOSS OF OFFSITE POWER INITIATOR (GRID-RELATED)
- INT-LOOPPC: LOSS OF OFFSITE POWER INITIATOR (PLANT-CENTERED)
- INT-LOOPSC: LOSS OF OFFSITE POWER INITIATOR (SWITCHYARD-RELATED)
- INT-LOOPWR: LOSS OF OFFSITE POWER INITIATOR (WEATHER-RELATED)
- INT-MLOCA: MEDIUM LOCA
- INT-SGTR: SG TUBE RUPTURE
- INT-SLBOC: STEAM LINE BREAK OUTSIDE CONTAINMENT
- INT-SLOCA: SMALL LOCA
- INT-TRANS: GENERAL PLANT TRANSIENT
- INT-XLOCA: EXCESSIVE LOCA INITIATING EVENT.

SAPHIRE [30] has been widely used to model plant response to both internal hazards (e.g., general transients, loss of offsite power, and loss of feedwater) and external hazards (e.g., seismic, fire, external flooding, and high wind). SAPHIRE 8 is a powerful PRA software that has both the basic PRA modeling capabilities such as creating event trees and fault trees, defining and assigning basic event failure data, linking and solving event trees and fault trees, documenting and reporting the results, and the advanced capabilities such as integrated Level 1 and Level 2 PRA analysis, performing sensitivity and uncertainty

analyses, and conducting specialized analyses for the NRC's Accident Sequence Precursor program and Significance Determination Process. [98]

5.2. Scenario Selections

In this project, an accident scenario is defined as the combination of an initiating event and the success or failure states of relevant plant systems performing mitigating functions. An initiating event is an unplanned event that occurs while a plant is in critical operation, requiring that the plant to shut down and achieve a stable, controllable state. The accident scenarios led by each initiating event are modeled using ETs. An ET starts from an initiating event and ends with multiple accident scenarios in different end states. The first box in the top row of an ET represents the initiating event, the rest of boxes in the top row represents different plant systems, and the tree branches represents the success (upper branch) or failure (lower branch) of a system. If an ET has a very large structure, it may be broken down into several sub trees.

This project applies SAPHIRE 8 for the FT development of DI&C system and combines these improved FTs with the existing generic PWR ET models. The consequence analysis of DI&C failures documented in this report covers the following accident scenarios: INT-TRANS (initiating event - general plant transient) with ATWS (anticipated transient without scram), LOSC (loss-of-seal cooling), SBLOCA (small-break loss-of-coolant accident), and MBLOCA (medium-break LOCA). These five accident ETs are respectively shown in Appendix B from Figure 57 to Figure 61. INT-TRANS is selected here for the DI&C consequence analysis because it shows relatively significant impacts of digital failures to key plant responses. RTS and ESFAS failures are treated as initiating events or important basic events included in cut sets that have significant contributions to change of CDF (ΔCDF).

5.3. Original and Improved Fault Trees for HSSSR DI&C Systems

5.3.1. Original RTS Fault Tree

The original RTS-FT in the generic PWR SAPHIRE model has identified analog/hardware failures in detail. The main logic of original RTS-FT is shown in Figure 48. A two-train analog RTS was modeled; the main failure modes include electric failures, CCF of rod cluster control assembly (RCCA) fail to drop, contribution of seismic events, operator errors, and RTS failures during test and maintenance.

This FT was quantified with SAPHIRE 8 using a truncation level of 1E-12; RTS failure probability is 4.288E-6 with five cut sets. Table 55 lists these cut sets for the original RTS-FT; it shows the main contributed basic events are the CCF of reactor trip breaker A and B, and CCF of RCCA, which contributes about 65.77% of the total RTS failure. Results indicate hardware CCFs are the main concerns for the failure analog safety-related redundant I&C systems.

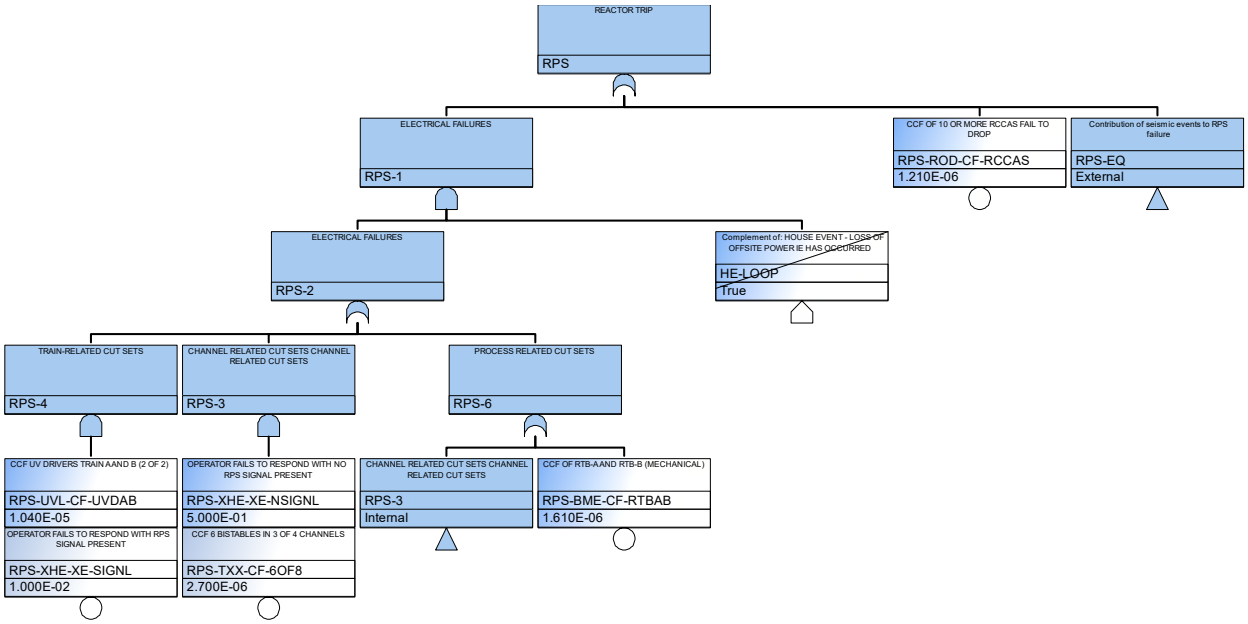

Figure 48. Main fault tree of original RTS-FT in the generic PWR SAPHIRE model.

Table 55. Cut sets for the original RTS-FT.

#	Probability	Total %	Cut Sets
1	1.610E-6	37.55	RPS-BME-CF-RTBAB
2	1.343E-6	31.33	RPS-CCP-TM-CHA, RPS-TXX-CF-6OF8, RPS-XHE-XE-NSIGNL
3	1.210E-6	28.22	RPS-ROD-CF-RCCAS
4	1.040E-7	2.43	RPS-UVL-CF-UVDAB, RPS-XHE-XE-SIGNL
5	2.052E-8	0.48	RPS-CCP-TM-CHA, RPS-TXX-CF-4OF6, RPS-XHE-XE-NSIGNL
Total	4.288E-6	100	-

5.3.2. Original ESFAS Fault Tree

In the original generic PWR SAPHIRE model, ESFAS failure is presented using a CCF of ESF actuation signal in both Train A and B, named ESF-VCF-CF-TRNAB with a probability as 6.420E-4. These CCF basic events are used in the FTs of several top events in IE-TRANS scenarios including AFW (representing failure of auxiliary feedwater), AFW-ATWS (representing failure of auxiliary feedwater for ATWS scenarios), HPI (representing failure of high-pressure injection), and LPI (representing failure of low-pressure injection). It should be noted another basic event is used to represent "operator fails to manually initiate safety features," which will be replaced by an integrated FT representing failure of digital HSI and operator errors in future work.

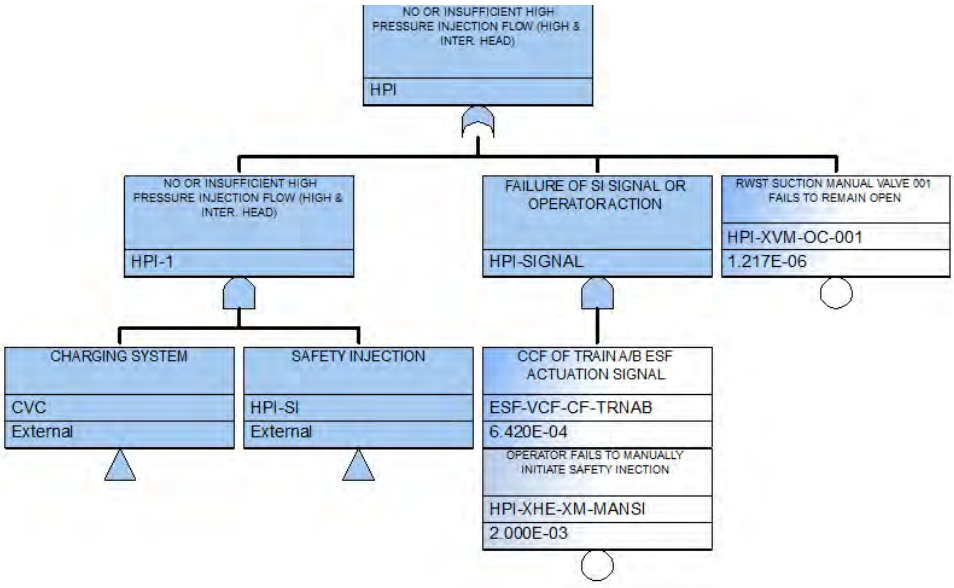

Figure 49. Main FT of HPI failure in the generic PWR SAPHIRE model where CCF of analog ESFAS is considered.

5.3.3. Improved RTS Fault Tree

The integrated FT for the failure of four-division RTS is added into the generic PWR SAPHIRE model and quantified in this section. This RTS system consists of numerous hardware and software components and have a large number of failure combinations. One failure path of many paths is shown as below ("->" means "caused by"): RTS system failure -> system actuation failure -> breaker failure -> undervoltage breaker trip failure -> logic cabinet rack failure -> digital output module failure -> local coincidence logic processor failure -> bistable processor failure.

This RTS-FT was developed using RESHA in FY-20 [16] and quantified using BAHAMAS in FY-21. Compared with the original FT for a 2-train analog RTS, this improved RTS-FT keeps part of failure modes: CCF of RCCA fail to drop, contribution of seismic events, operator errors, and RTS failures during test and maintenance and extends the electric failures to an integrated automatic RTS failure including both hardware and software failures. The main logic of this integrated RTS-FT is displayed in Figure 50.

This FT was also quantified with SAPHIRE 8 using a truncation level of 1E-12, RTS failure probability is 1.253E-6 with 41 cut sets. Table 56 lists part of these cut sets with significant contributions. Compared with the original RTS-FT, the total failure probability of RTS is reduced about 70%. Mechanical CCF of RCCA becomes the main contributor (>95% of total); the software CCFs do not significantly affect the reliability of digital RTS because of the highly redundant design and high reliability of PLC-based digital components.

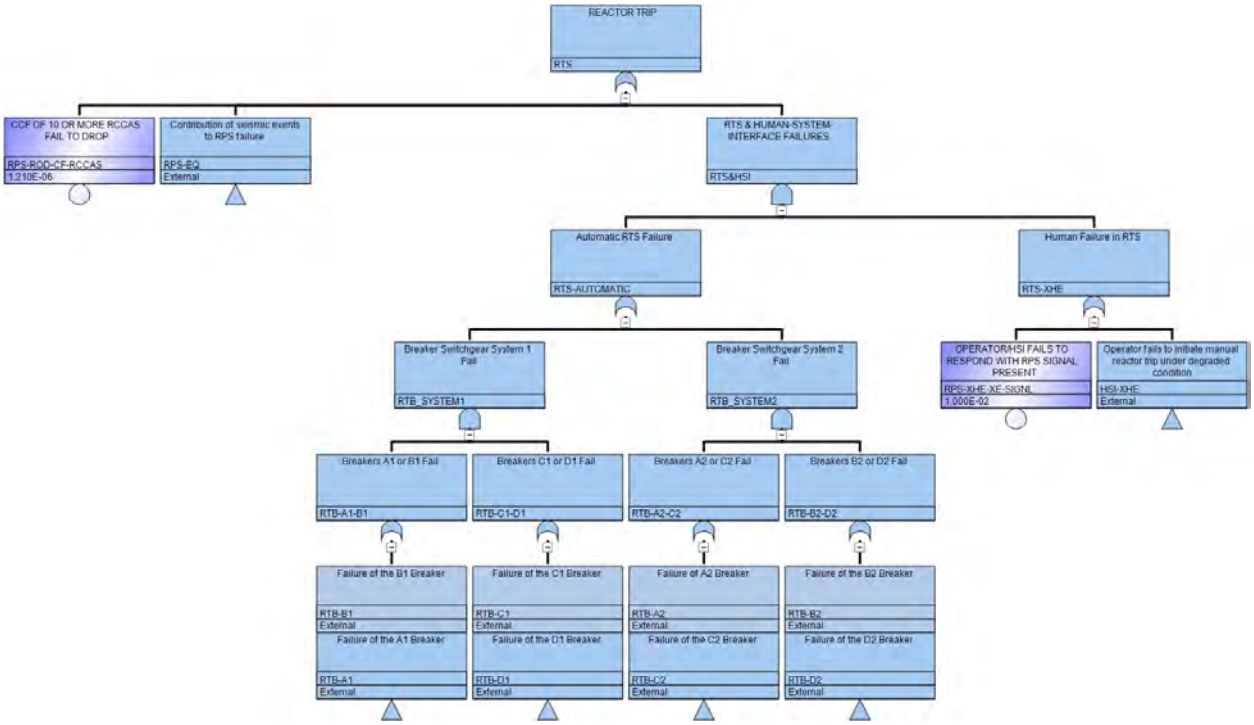

Figure 50. Main fault tree of improved RTS-FT.

Table 56. Cut sets for the improved RTS-FT.

FT Name	#	Probability	Total %	Cut Sets
Improved RTS-FT	1	1.210E-6	96.60	RPS-ROD-CF-RCCAS
	2	1.944E-8	1.55	RPS-CCP-TM-CHA, RPS-TXX-CF-4OF6, RPS-XHE-XE-NSIGNL
	3	1.944E-8	1.55	RPS-XHE-XE-SIGNL, RTB-SYS-2-HD-CCF
	4	4.611E-10	0.04	RPS-XHE-XE-SIGNL, RTB-SYS-1-HD-CCF
	Total	1.253E-6	100	-
Original RTS-FT	Total	4.288E-6	100	-

* Full names of the acronyms can be found in the Acronyms list on Page xiii.

5.3.4. Improved ESFAS Fault Tree

The integrated FT for the failure of four-division ESFAS for the actuations of AFW, HPI, and LPI is added into the generic PWR SAPHIRE model and quantified in this section. This ESFAS-FT was developed using RESHA in FY-20 and quantified using BAHAMAS in FY-21. Compared with the original ESFAS-CCF, this integrated ESFAS-FT has a complicated logic to match the four-division digital ESFAS structure deployed in APR-1400. This FT was also quantified with SAPHIRE 8 using a truncation level of 1E-12; ESFAS failure probability is 2.600E-5 with 13 cut set.

Table 57 lists part of these cut sets with significant contributions. Compared with the original ESFAS failure, the total failure probability of ESFAS is significantly reduced. Hardware CCF of ESF-component interface modules (ESF-CIMs) becomes the main contributor, and the software CCFs do not significantly

affect the reliability of digital ESFAS because of the high-redundant design and high reliability of PLC-based digital systems.

Table 57. Cut sets of the improved ESFAS-FT.

FT Name	Probability	Total %	# of Cut Sets
Improved ESFAS-FT	2.600E-6	100	13
Original ESFAS failure	6.420E-4	100	1

5.3.5. Human-System Interface Fault Tree

This section develops a HSI-FT representing a human-action initiating manual reactor trip under the degraded HSI condition of digital main control rooms. The DI&C system of APR1400 [22] has been reviewed and used to develop the HSI-FT. The most relevant systems of the human action (i.e., QIAS-P, core protection calculator system (CPCS), IPS, and QIAS-N) have been specifically modeled in the HSI-FT. RESHA process has been employed for the FT development. Then, the FT is integrated into the general PWR transient ET, then further analyzed, and discussed.

The DI&C systems mentioned above provide all information required for achieving plant safe shutdown and performing emergency operating procedures. First, as a safety-graded system, QIAS-P provides unambiguous indication of inadequate core cooling as well as advanced warning of the approach toward it. It also acts as a continuous source of accident monitoring information under both nominal and emergency scenarios. Second, CPCS calculates important values to core cooling and protection such as departure from nucleate boiling or local power density and provides them to other digital I & C systems. Third, as a non-safety I & C system, QIAS-N receives analog and digital signals from both safety and non-safety systems, analyzes the data, and presents the information to the operator via the QIAS-N front panel displays (FPDs), mini-large display panel (mini-LDP), and shutdown overview display panel (SODP). Lastly, IPS collects relevant sensor information and plant states, as a non-safety I & C system. The IPS and QIAS-N are independent and diverse from each other. The IPS information is processed and used in IPS information flat panel display (IFPD), the safety parameter display and evaluation system+ (SPADES+), or computer-based procedure system (CPS).

Figure 51 shows the piping and instrumentation diagrams for QIAS-P, CPCS, IPS, and QIAS-N. These have been assumed from the DI&C system of APR1400. First, for the QIAS-P in the figure, it is simplified into a diagram because it already developed and analyzed in the previous study. The information from the QIAS-P is provided to the QIAS-N processor and the IPS server. Second, CPCS having four divisions (A, B, C, D) has several calculation modules, which calculate core-related values such as departure from nucleate boiling or local power density using plant parameters (e.g., hot leg temperature). Then, CPCS sends the values into the QIAS-N processor and the IPS server. Third, in the QIAS-N system, there are the processor, primary and backup controllers, server, and alarm processing function, which play a role in gathering plant information and analyzing the data, while the QIAS-N FPDs, mini-LDP, and SODP present the information to operators in main control rooms. Lastly, the IPS has the primary and dedicated backup servers that collect information from the QIAS-P and CPCS, the alarm processing system, and the applications such as the IPS IFPD, SPADES+, and CPS.

Figure 52 shows the top event of the HSI-FT. The operator-action initiating manual reactor trip under degraded condition is located on the top of the FT. The top event logic consists of the FT gates representing that QIAS-P, IPS and QIAS-N fail to notify via alarm and accurately reflect safety variables under degraded reactor state. Figure 53 and Figure 54 show the FT logics for QIAS-N and IPS, respectively. The CPCS is modeled in the sub-logics. In the FTs, all the values for basic events such as probabilities for UCA/UIF, hardware failure, or CCF have been assumed from [3].

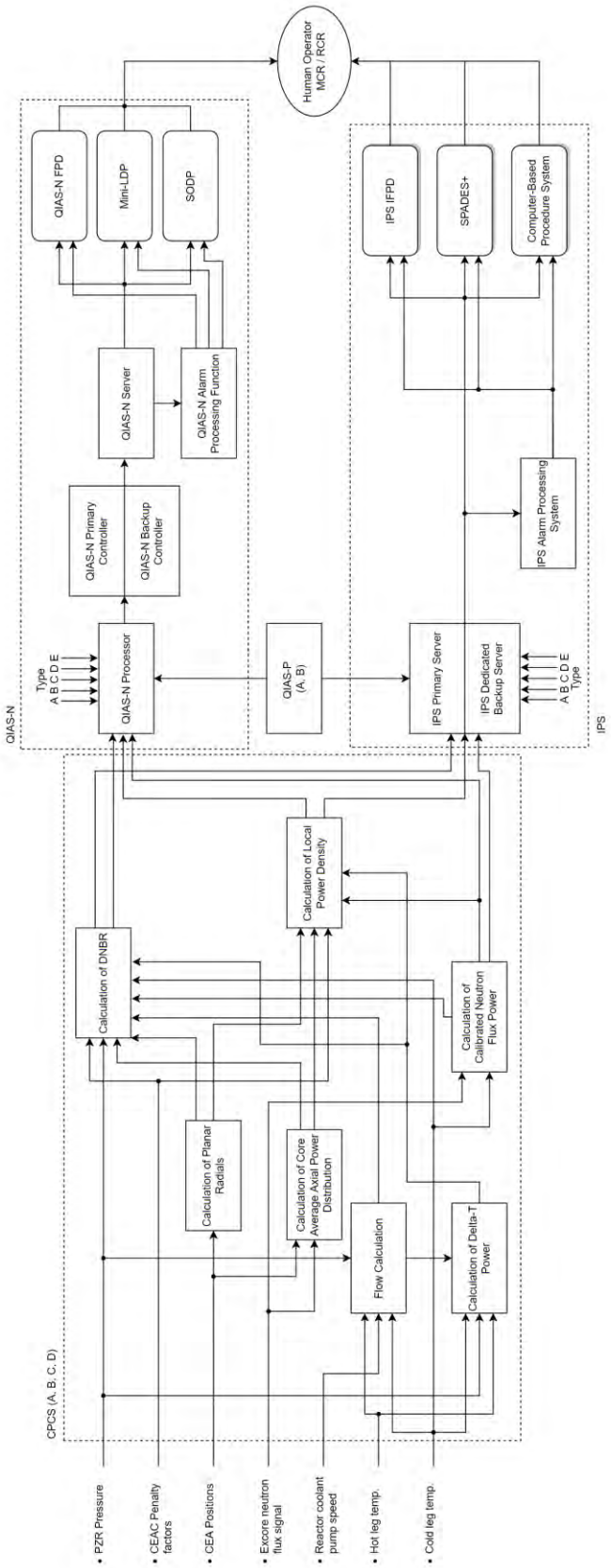

Figure 51. The piping and instrumentation diagrams for QIAS-P, CPCS, IPS, and QIAS-N.

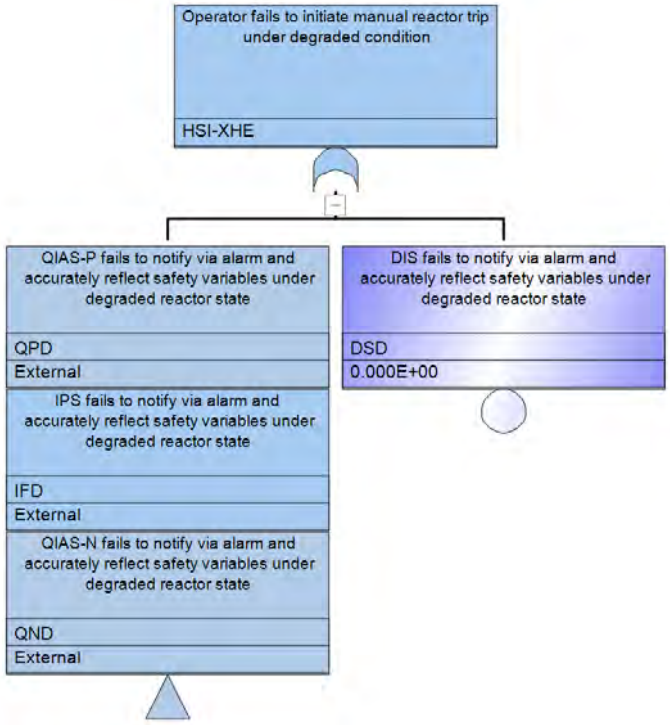

Figure 52. The top event of the HSI-FT.

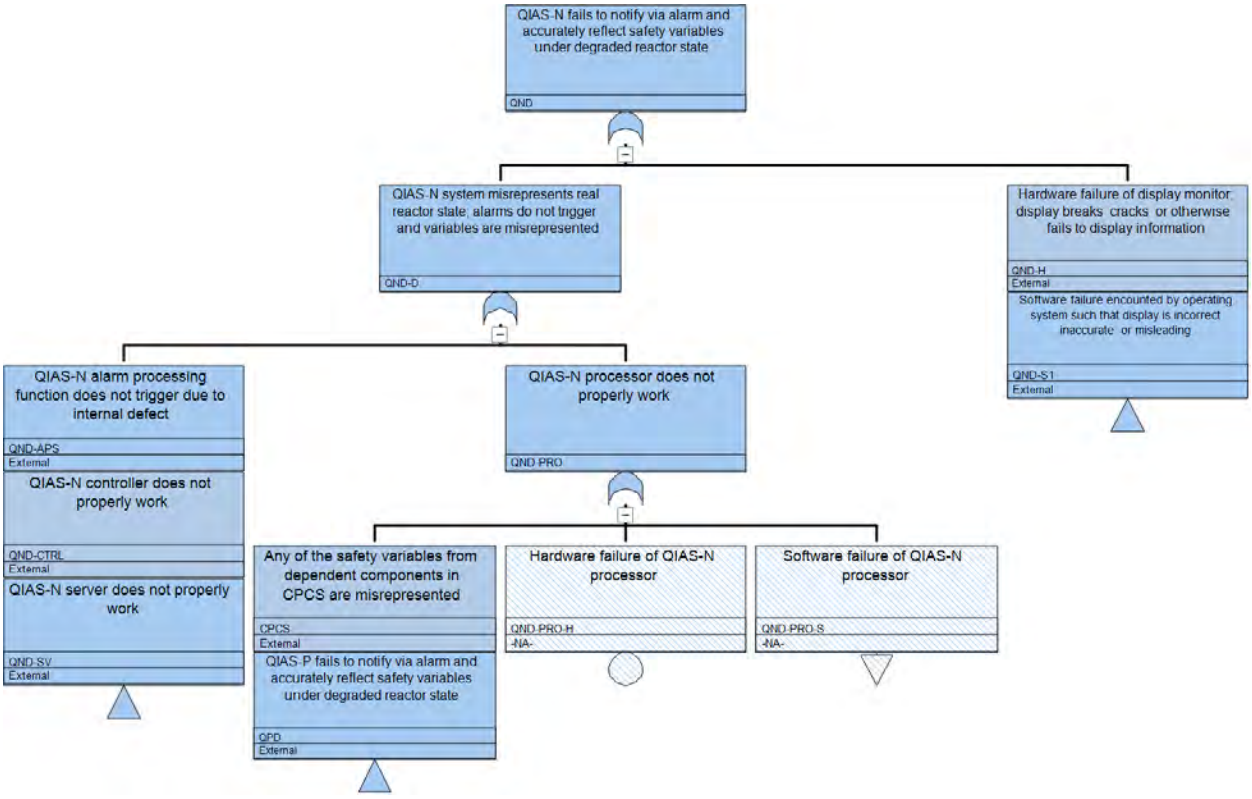

Figure 53. The FT logic for QIAS-N.

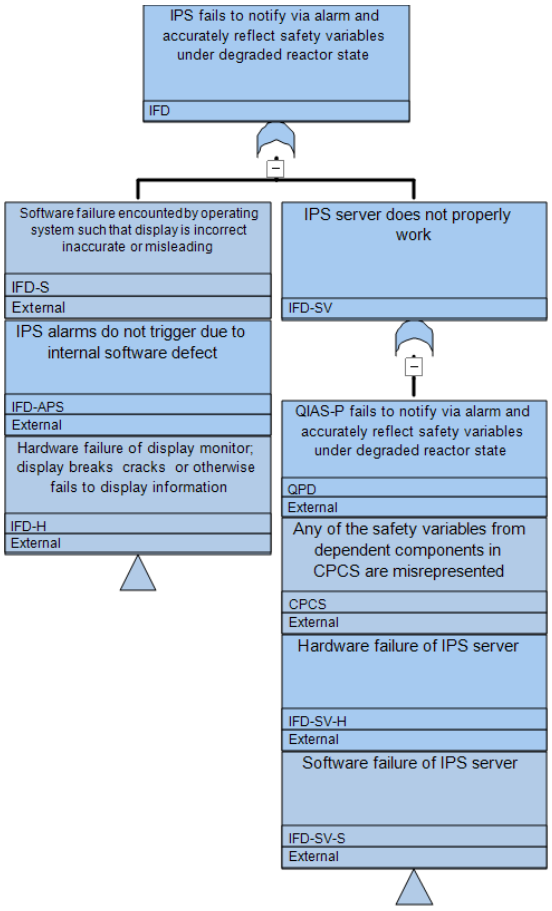

Figure 54. The FT logic for IPS.

This study calculates failure probabilities and cutsets for each digital I & C system (e.g., QIAS-P, QIAS-N, and IPS) and the top event. This FT is quantified with SAPHIRE 8 using a truncation level of 1E-12. Table 58 shows the comparison of failure probabilities, the number of cutsets, and the cutset ranking on the top event, QIAS-P, QIAS-N, and IPS. The failure probability of the top event is 9.207E-4 with 394 cut sets. The failure probabilities for QIAS-N and IPS are 4.837e-4 and 5.337e-4, respectively, while the probability for QIAS-P indicates 9.663E-5. In other words, the safety-critical I&C system (i.e., QIAS-P) shows the lower failure probability than the non-safety I&C systems (i.e., QIAS-N and IPS). If we take a look at the cutset ranking, the basic events from QIAS-N and IPS account for the cutset ranking #1 to #5 in the failure probability of the top event.

Table 58. Comparison of failure probabilities, the number of cutsets, and the cutset ranking.

		Top Event	QIAS-P	QIAS-N	IPS
Failure Probabilities		9.207e-4	9.663e-5	4.837e-4	5.337e-4
# of Cut Sets		394	383	388	389
Cutset Ranking	**#1**	QND-APS-UIFA	QPD-H	QND-APS-UIFA	IFD-APS-UIFA
	#2	IFD-APS-UIFA	QPD-PA-CTC-S-CFD	QND-APS-H	IFD-SPA-H
	#3	QND-APS-H	QPD-PA-RCS-S-CFD	QND-SV-H	QPD-H
	#4	IFD-SPA-H	QPD-PA-HJC-S-CFD	QPD-H	IFD-APS-H
	#5	QND-SV-H	QPD-PA-RLC-S-CFD	QND-PRO-H	IFD-CPS-H

* Full names of the acronyms can be found in the Acronyms list on Page xiii.

This HSI-FT is added into the RTS-FT and also considered for the ESFs that can be manually controlled. Accordingly, the impact of ESFAS failure with HSI failure to the actuations of safety features were estimated by solving the FTs of AFW, AFW-ATWS, HPI, and LPI; results are listed and compared in Table 59. All the failure probabilities of these safety features have been reduced due to the decrease of ESFAS failure probability.

Table 59. Comparison of the top events with the improved ESFAS-FT and HSI-FT.

Top Event	Probability		# of Cut Sets	
	Original	**New**	**Original**	**New**
Failure of AFW	1.487E-5	1.231E-5	1539	1539
Failure of AFT-ATWS	2.367E-4	2.342E-4	906	906
Failure of HPI	1.104E-5	9.756E-6	1163	1163
Failure of LPI	8.416E-4	2.018E-4	1567	1567

5.4. Accident Scenario Analysis for General Plant Transient

In this section, IE-TRANS, IE-SBLOCA, and IE-SBLOCA ET tree quantification results are discussed. As IE-ATWS and IE-LOSC are sub-ETs of IE_TRANS developed in the generic PWR SAPHRIE model, they will be discussed in the section on INT-TRANS.

5.4.1. INT-TRANS

The generic PRA model representing general plant transient as INT-TRANS was shown in Figure 57. The IE-TRANS ET was quantified with SAPHIRE 8 using a truncation level of 1E-12. Table 60 compares the quantified CDF with original and new ETs. The original total IE-TRANS CDF is 1.073E-6/year and half-reduced to 5.746E-7/year with the new FTs. There are 16 non-zero CDF sequences out of a total of 145 INT-TRANS accident sequences (i.e., the sequence end state is core damage).

INT-TRANS:21-16 from ATWS scenarios is one of the most risk-significant sequences with a CDF reduced from 5.388E-7/year to 1.573E-7/year and contributes 27.38% of the CDF of improved INT-

TRANS. In this sequence, RTS fails to trip the reactor; primary and secondary side depressurization are not successful. Core damage occurs as long-term cooling cannot be established.

INT-TRANS:20 from TRANS scenarios is another risk-significant sequence with a CDF reduced from 3.895E-7/year to 3.263E-7/year, contributing 56.79% of the CDF of improved INT-TRANS. In this sequence, RTS successfully trips the reactor; core damage occurs as neither the auxiliary feedwater is available nor could the HPI provide makeup water to the reactor coolant system.

Table 60. Comparison of INT-TRANS ET quantification results.

Sequence	CDF			# of Cut Sets	
	Original ET	Improved ET	Δ CDF/ Original CDF	Original ET	Improved ET
INT-TRANS:21-16	5.388E-07	1.573E-07	-70.81%	51	93
INT-TRANS:20	3.895E-07	3.263E-07	-16.23%	1060	1081
INT-TRANS:21-14	7.262E-08	2.120E-08	-70.81%	49	37
INT-TRANS:02-02-09	5.830E-08	5.832E-08	0.03%	1248	1272
INT-TRANS:19	8.132E-09	6.692E-09	-17.71%	282	236
INT-TRANS:02-03-09	2.731E-09	2.763E-09	1.17%	387	427
INT-TRANS:02-02-10	9.546E-10	1.006E-09	5.38%	168	238
INT-TRANS:21-15	7.568E-10	2.124E-10	-71.93%	102	29
INT-TRANS:02-04-10	5.865E-10	5.896E-10	0.53%	142	146
INT-TRANS:02-14-10	1.994E-10	2.091E-10	4.86%	81	93
INT-TRANS:02-03-10	7.653E-12	1.131E-11	47.79%	4	8
INT-TRANS:02-09-09	7.558E-12	1.117E-11	47.79%	4	8
INT-TRANS:02-06-09	7.558E-12	1.117E-11	47.79%	4	8
INT-TRANS:02-08-09	7.558E-12	1.117E-11	47.79%	4	8
INT-TRANS:02-07-09	2.287E-12	2.287E-12	0.00%	2	2
INT-TRANS:02-10-09	2.287E-12	2.287E-12	0.00%	2	2
Total	1.073E-6	5.746E-07	-46.45%	3590	3711

5.4.2. INT-SLOCA

The generic PRA model representing small-break LOCA as INT-SLOCA was shown in Figure 60. The INT-SLOCA ET was quantified with SAPHIRE 8 using a truncation level of 1E-12. Table 61 compares the quantified CDF with original and new ETs. The original total INT-SLOCA CDF is 7.784E-8/year and was reduced to 7.508E-8/year with the new FTs. There are seven non-zero CDF sequences out of a total of 10 INT-SLOCA accident sequences (i.e., the sequence end state is core damage).

INT-SLOCA:03 is the most risk-significant sequences with a CDF of 6.433E-8/year and contributes 85.68% of the CDF of improved INT-SLOCA. In this sequence, RTS successfully trips the reactor, and the auxiliary feedwater and HPI are available. Core damage still occurs, as long-term low-pressure cooling cannot be established.

Table 61. Comparison of INT-SLOCA ET quantification results.

Sequence (with CDF > 1E-9)	CDF			# of Cut Sets	
	Original ET	Improved ET	Δ CDF/ Original CDF	Original ET	Improved ET

INT-SLOCA:03	6.433E-08	6.433E-08	0.00%	564	566
INT-SLOCA:05	2.867E-09	2.867E-09	0.00%	97	97
INT-SLOCA:09	7.386E-09	6.619E-09	-10.38%	142	142
Total	7.784E-8	7.508E-08	-3.55%	838	842

5.4.3. INT-MLOCA

The generic PRA model representing medium-break LOCA as INT-MLOCA was shown in Figure 61. The INT-MLOCA ET was quantified with SAPHIRE 8 using a truncation level of 1E-12. Table 62 compares the quantified CDF with original and new ETs. The original total INT-MLOCA CDF is 6.279E-7/year and reduced to 4.984E-7/year with the new FTs. There are eight non-zero CDF sequences out of a total of nine INT-MLOCA accident sequences (i.e., the sequence end state is core damage).

INT-MLOCA:03 is the most risk-significant sequences with a CDF of 4.917E-7/year and contributes 98.66% of the CDF of improved INT-MLOCA. In this sequence, RTS successfully trips the reactor, and the auxiliary feedwater and HPI are available. Core damage still occurs, as long-term low-pressure cooling cannot be established.

The most significant CDF reduction appears in INT-MLOCA:10 from 1.305E-07 to 2.567E-9, which contributes to 1.98% of Δ CDF. In this sequence, RTS successfully trips the reactor, and the auxiliary feedwater is available. Core damage still occurs as neither HPI nor LPI can be established for long-term cooling. According to the change of failure probability of LPI shown in Table 59, LPI availability was highly increased by adding a FT of a more reliable 4-division digital ESFAS instead of using a conservative CCF. Improved ET models can provide a more accurate prediction for ESFAS failure and relevant sequences.

Table 62. Comparison of INT-MLOCA ET quantification results.

Sequence (with CDF > 1E-9)	**CDF**			**# of Cut Sets**	
	Original ET	**Improved ET**	**Δ CDF/ Original CDF**	**Original ET**	**Improved ET**
INT-MLOCA:03	4.917E-07	4.917E-07	0	722	724
INT-MLOCA:05	1.870E-09	1.870E-09	0	47	47
INT-MLOCA:07	8.999E-11	8.999E-11	0	26	26
INT-MLOCA:09	1.866E-09	1.866E-09	0	192	192
INT-MLOCA:10	1.305E-07	2.567E-09	-98.03%	47	47
INT-MLOCA:11	5.206E-10	8.654E-12	-98.34%	3	3
INT-MLOCA:12	5.320E-10	3.510E-12	-99.34%	8	2
INT-MLOCA:14	8.576E-10	2.498E-10	-70.87%	5	3
Total	6.279E-7	4.984E-07	-20.62%	1050	1044

5.5. Summary of Consequence Analysis

In Section 5, consequence analysis has been performed based on INT-TRANS and relevant accident scenarios by comparing compare the changes of CDF after adding integrated FTs of digital RTS, ESFAS and HSI to the generic PWR ET models. As the last part of the proposed framework, consequence analysis successfully. evaluates the impact of identified CCFs, especially software CCFs, to plant safety. Results show the CDF of INT-TRANS accident scenarios are reduced significantly, plant modernization

including the improvement of HSSSR DI&C systems such as digital RTS, ESFAS and HSI will make great benefits to plant safety by enhancing DI&C reliability and plant safety. Software CCFs identified in the case study doesn't have significant contributions to CDF.

6. FUTURE APPLICATIONS ON AI-AIDED CONTROL SYSTEMS

The interest in AI and machine learning (ML) is growing rapidly in both the public and private sectors and anticipates the increased use of AI in the reactor operation and control. AI refers to a machine-based system that can go beyond defined results and scenarios and can emulate humanlike perception, cognition, planning, learning, communication, or physical action. For a given set of human-defined objectives, AI can make predictions, recommendations, or decisions influencing real or virtual environments which can bring in great benefits to the nuclear industry by supporting more reliable, safe and cost-effective operation and control of nuclear power plants.

Anticipating the industry's potential application of AI to NRC-regulated activities, the NRC also developed a strategic plan [101] to ensure the agency's readiness to review such uses. The strategic plan includes five goals: (1) ensure NRC readiness for regulatory decision-making, (2) establish an organizational framework to review AI applications, (3) strengthen and expand AI partnerships, (4) cultivate an AI-proficient workforce, and (5) pursue use cases to build an AI foundation across the NRC. The overall goal of this strategic plan is to ensure continued staff readiness to review and evaluate AI applications effectively and efficiently.

These goals are very relevant to the scope of this project, particularly for goals (2) and (5). The framework developed in this project has the potential to be extended and modified for risk assessment and design optimization of advanced intelligent AI-aided control systems. Therefore, this section discusses the motivation, applicability, limitations, and potential benefits of applying the proposed framework on potential AI-aided control systems that are designed to existing LWR fleets.

6.1. Motivation

There are many different algorithms that have been used and called as "AI/ML systems." Statistical and mathematical that approximate patterns (i.e., k-nearest clustering and support vector machines) are one class of popular methods. There also exists the class of neural network models (i.e., feedforward neural networks, and recurrent neural networks) that are inspired from brain neuron architecture. In the most recent definition of "AI algorithms" in NUREG-2261, "Artificial Intelligence Strategic Plan, Fiscal Years 2023-2027," it is defined as a "computer program that has been trained on a set of data to recognize certain types of patterns" [101]. Furthermore, it can "reason and learn…with the overarching goal of providing solutions that mimic human-based decision and predictions for problems" [101].

The motivation for the usage of AI/ML algorithms in DI&C systems is based on the anticipated economic cost-savings and improved safety monitoring that AI/ML systems can provide over existing systems. Unlike conventional software systems that are hard coded with specific functionality and limited variability, AI/ML systems can theoretically adapt during operation to any functional change with little to no modification in the base programming given sufficient input data. Furthermore, they can be used to derive hidden correlations within the data to extract useful information that otherwise would be incredibly difficult to model with conventional physics. For example, AI/ML algorithms can be used for fault detection based on the subtle changes in the vibrational pattens of rotating equipment [102]. Such systems are incredibly difficult to model with physics-based equations or spectrum analysis due to the innate computational complexity and intractability of components involved. From a regulatory perspective, industry licensees have also sought other methods for meeting licensing requirements in 10 CFR 50.65 in complex AI systems and have opted to discontinue the method endorsed by the NRC [103]. Therefore, to meet the upcoming challenges, it is highly desirable to have a qualitative and quantitative framework to assess potential risks that can be introduced by these AI/ML systems. The proposed framework in this project already has existing methodologies (i.e., ORCAS and BAHAMAS) to assess potential software failure modes and risks in conventional DI&C software systems. It is hypothesized that extending these methods to include AI/ML systems can also provide useful risk and design information where there currently does not exist an established method to do so.

Specifically, the "Near Autonomous Management and Control System (NAMAC)" [104] is used as an example of how AI/ML models are used for the active control of NPP reactors relevant to manual trip safety. The NAMAC system is a comprehensive control system to assist plant operations by furnishing recommendations to operators. Such recommendations are derived by integrating knowledge from scenario-based model of plant, plant operating procedures, real-time measurements, etc. The NAMAC system aims to achieve an alignment of NPP safety design, evaluation, operator training, and emergency management by extracting useful information from the knowledge base. The knowledge extraction and storage for different intended uses are achieved by various ML algorithms in digital twins (DTs). The development of a ML-based DT for an intended use relies on the construction of a knowledge base and the use of advanced data-science techniques, while its deployment and operation are realized based on DI&C systems that consist of different software, data networks, digital platforms, and hardware carriers. Although the potential and feasibility for DTs by ML algorithms to improve safety and efficiency in reactor control have been recognized, one of the concerns for the NRC and the nuclear industry is whether the information from a DT is credible to support decisions of high consequence and how this information could affect further DT development in an autonomous control system. To provide evidence toward the credibility assessment for autonomous control systems with DTs, software risk assessment should be implemented.

6.2. Challenges Associated with AI/ML Deployment and Risk Analysis

Under the current regulatory framework, there does not exist enough guidance on how AI/ML algorithms will be deployed to assist NPPs. There are potentially two types of ways AL/ML algorithm will be used, passive (non-safety critical) and active (safety critical). In passive AI/ML systems, the algorithms are used to predict non-operational plant characteristics that do not relate directly to plant trip safety. These may include maintenance schedule prediction, non-safety equipment fault prediction, etc. The algorithms are not used for the control of the operator decision-making process in any safety critically defined systems. In active AI/ML systems, the AI/ML system is used for either partial or complete automatic control or has direct influencing capability in the operator decision-making process. For example, the AI/ML algorithm is to control pump speed (and therefore coolant flow rate through the core) or predicts safety-critical parameters that are used to gauge core stability (i.e., fuel centerline temperature).

While direct risk assessment is not anticipated to be needed in passive AI/ML systems as it is anticipated to be neither economical nor risk significant, the framework developed in this project could be used to conduct performance/availability analysis. For example, equipment fault prediction can be used to anticipate the maintenance schedules and remaining useful life which directly impacts performance metrics [102]. More concerning are the AI/ML systems used in active safety-critical systems. Currently, there are two significant issues that current and proposed regulatory strategies fail to address:

- The maintenance requirements for models incorporated over the lifetime of the plant also known as technical debt [105]
- The overarching premise that baseline data collected in the development of said predictive models are representative of the problem scope for the entirety of operation.

These two problems are even more significant than any technical development issues of predictive models. In NUREG/CR-7294 [106], the NRC has explored existing state of technology of AI/ML models. Repeated throughout the report is the issue of data quality, quantity, applicability, and uncertainty. These are significant non-trivial problems that currently plague all industries that utilize AI/ML. However, the difference is that in nuclear energy, the consequence of model failure is significant compared to other industries. The design of the model is less relevant as nearly all models suffer from the two problems above. Take the following statement from NUREG/CR-7294 as an example:

> However, due to AI/ML uncertainty, the insufficiency of data quality and quantity, and lack of cognition about how to efficiently incorporate knowledge and data, challenges of adapting AI/ML techniques still exist. New perspectives and advanced frameworks should be proposed for different purposes in nuclear engineering. Particularly, the "black box" nature of ML/AI brings challenges with respect to the trustworthiness and transparency of the results in nuclear industry. This challenge makes the deployment of ML/AI-guided applications difficult to satisfy the regulatory requirements of NRC [106].

This leads to the first proposed problem that existing regulation has not yet addressed. Suppose for the sake of argument that sufficient quality data exists during the software development process of the ML/AI model to develop highly accurate predictive models for an arbitrary safety-critical variable (e.g., fuel centerline temperature). The model is dispatched to a plant to assist operator decisions. At first glance, this seems to be the optimal "goal" of the ML/AI project, a dispatchable model that can improve safety; however, significant issues will arise through the lifetime of the plant. The primary problem is the assumption that sufficient quality data is available is fundamentally flawed. The operating physics of NPPs are constantly changing from beginning of life (BOL) to end of life (EOL). The data collected can only represent a subset snapshot of the reactor physics and will never represent all possible states. Therefore, it is highly probable that the model will slowly (or suddenly) become irrelevant at some changing point in the reactor lifecycle. Models developed based on BOL or snapshot reactor physics data will lose predictive accuracy overtime and be detrimental to operators rather than useful. On the other hand, sudden changes in the reactor state (known as context shifts [107]) can disrupt temporal predictive models (e.g., RNN). Anticipated sudden changes, such as replacing fuel or a single sensor degrading, are examples of context shifts. More serious context shifts could be power transients and could cause serious predictive issues. This data problem cannot be easily resolved by including all possible states of concern in the data set. Aside from the computational intractability and size required for all possible state, even limiting the data set to the subset including anticipated operational occurrences (AOO) or design-basis accidents (DBA), would rely on strong assumptions about how systems fail that may be physically infeasible. This dataset for anticipated possible states would also be expensive and would most likely rely on multiple models developed for different context scenarios. The multiple model requirement is due to the overgeneralization problem of neural network-based AI/ML models [108]. In essence, as the problem scope increases to include more scenarios, the accuracy of the model steadily decreases as the model weights become averaged over the entire input set. In general, there is no acceptable solution to the latter problem as there are too many highly speculated (and unsupported) scenarios to consider developing a single comprehensive model. Certain solutions do exist where single models solve the overgeneralization problem such as ResNET 50 [109]; however, in these instances, multiple highly complex pre-processing layers are required to separate the input data into smaller identifiable subsets before a neural network is applied. However, these types of large models are difficult to update and are less versatile whenever unfamiliar conditions arise. Furthermore, due to the numerous layers in such models, they are also less interpretable and less transparent to human operators. Both methods have serious challenges and benefits; however, if the former approach is adopted, it may be possible to update the AI/ML model routinely to reflect changing states of the NPP without requiring an overly complex model while maintaining interpretability.

Generally, there are three approaches for implementing AI/ML models in continuously evolving environments: (1) locked models, (2) updating locked models, and (3) continuously updating models. In the most basic approach, a locked model is a model with weights and parameters that cannot change regardless of the state of the plant. The benefits to this approach are that the model can be reviewed and controlled for quality without concern of any deviation from function. The outputs from the model can be anticipated based on the training data and determined if they are valid or not. However, the drawback is that the model is guaranteed to become irrelevant over time in any continuously evolving environment. It also removes the primary benefit of AI/ML models which is the ability to adapt and continuously "learn"

the operational environment. The second approach is the updating locked model approach. In this approach, locked models (with updated weights) are periodically dispatched as software updates whenever a sufficient change in the operational environment is detected. Each of the locked models are verified against a benchmark to ensure that they meet performance and safety requirements. In this approach, the ML/AI models can still update and "learn" however under the strict oversight of developers. The drawback to this approach, although, is given multiple versions of the same model will be developed, then which model should be trusted and used. If an earlier version of the model contradicts later versions of the same model, should operators not trust the later version of the model? Version contradiction and management is the chief problem with the updating locked model approach. The last approach is the conventional approach to ML/AI models, that is to develop a continuous learning model with build-in restrictions and limitations and to hope that the ML/AI model will always perform accordingly. The model will adapt to any minute changes in the operational environment from BOL to EOL and theoretically to always provide accurate results. However, this approach is incredibly dangerous. Routinely, it has been seen where optimal models that have been developed in the ideal laboratory conditions become corrupted in production environments, cause unanticipated side effects, or abuse constraints (in the reward function) and result in loss to stakeholders. It is also incredibly difficult to design comprehensive constraints such that the model will always perform as expected. It is difficult in conventional systems (which is why software failure routinely occurs), and it is nearly impossible for a continuously learning system.

Depending on how the regulation and industry intend to deploy AI/ML models will influence how risk assessment will be conducted. For example, the risk assessment for locked models may require a periodic reassessment of the relevancy of the model's predictions against changing operational conditions of the plant. This is because any risk assessment conducted on the AI/ML model during its development or BOL phase can lead to misleading confidence on the failure probability as the technology and plant ages. This poses the first technical challenge to the software risk assessment in the proposed framework. As operational profile changes over time, the risk of using AI/ML algorithms in active safety-critical systems will change. Regulatory choices on how AI/ML systems will be implemented in operational environments will have significant influence on how these systems will be assessed for safety.

The second major challenge for AI/ML risk assessment is data control and management for risk assessment. This challenge is related to a new emerging topic in AI/ML risk assessment, specifically out-of-distribution (OOD) detection [110]. The premise of OOD being that AI/ML models are only highly accurate whenever the input data is within the training data subset (i.e., interpolation). However, AI/ML models routinely fail at the extrapolation task. OOD detection therefore is "sensing" or calculating how "far" away the input data is from known training data. The Mahalanobis distance [110] is one example metric used to gauge distance of an input to training data distributions. Training data will always be a subset of the operational environment (for the reasons discussed above). Furthermore, training data is typically developed in a highly augmented environment (i.e., without data noise, normalized, trimmed, and synthesized). This means that while training performance can be incredible high (>90%), when applied to real-world applications, the models typically experience a 30–40% decrease in predictive accuracy. Therefore, a method to measure OOD is required to ensure trust, transparency, and interpretability of the predictions made by an AI/ML algorithm. The significance of OOD detection is not mentioned in any current or proposed plan.

6.3. The Role of LWRS-developed Framework in Risk Assessment for AI/ML Supported Control and Information System

This project has developed a few methods for qualitative and quantitative risk assessment. In data-rich scenarios, the ORCAS was presented to predict the remaining number of latent defects hidden in software and its relationship to UCA/UIF failure modes. In BAHAMAS, the quality of the software development lifecycle is assessed via Bayesian Belief Networks to estimate the probability of software

failure modes. Both methods were developed for conventional software systems, where the functionality of the system is defined explicitly by the source code. Furthermore, the functionality of the software does not change regardless of state or age of the operational environment.

However, AI/ML algorithms have a mutable function and, depending on the method of maintenance, will change as the operational environment changes. This is due to how function is ascribed to AI/ML systems, mainly through adjusting the various weights and bias of the model. Unlike conventional software systems, the function of the AI/ML system can be changed without any change to the actual source code. This can result in higher than initially anticipated false positives or false negative predictions made. This is the primary concern of AI/ML algorithms in control and information systems as false predictions may confuse dependent devices or operators on the true state of the plant. This poses a significant technical challenge for the current framework. While the framework can assess the quality of the software and predict failures due to inadequacies in the source code, it currently has no capability at predicting how, when, or to what frequency AI/ML systems will fail. ORCAS depends on the quality of the source code and the described functionality. The function of AI/ML systems is defined by training data, and therefore, ORCAS would not be able to assess how data impacts the reliability of the system. BAHAMAS depends on the quality of the software development lifecycle. The reliability of the AI/ML system can change in the operational environment beyond the development environment and therefore, assessing only the development lifecycle would provide skewed confidence in the prediction results. Although neither method is geared towards assessing risk of AI/ML systems and the potential problems that may arise during the operational environment, the LWRS-developed framework can provide software design support through its native function, which can be used as the basis to support the development of an advanced integrated tool for the risk assessment of AI-aided control systems.

During the operation of ML-based DTs and respective autonomous control systems that are designed for safety purposes, crucial software CCFs may occur in different DTs for different intended uses in a NAMAC system or in redundant DTs for the same intended use. The capability of an integrated risk-informed analysis approach, similar to the LWRS-developed framework, is significantly needed to meet the industrial requirements for the deployment of intelligent DI&C systems (e.g., NAMAC system). A systematic hazard analysis should be applied to construct a knowledge base of plant anomaly causes, including all potential failure modes (e.g., digital and mechanical malfunctions, human errors, physical and cyberattacks, and environmental disturbance). Respective probabilities of these root causes should be provided by reliability analysis to evaluate the reliability of DT and the NAMAC system. Considering that some failures, especially software CCFs in DTs, may introduce some unanalyzed initiating events to the autonomous control systems, results from consequence analysis should be used to evaluate the impacts of these unanalyzed events and sequences on plant responses. All the information is useful for the real-time operation of ML-based DTs. [19]

7. CONCLUSIONS AND FUTURE WORKS

7.1. Conclusions

This report documents the research activities in FY-22 for the methodology improvement and demonstration of the LWRS-developed framework for the risk assessment and design optimization of safety-critical DI&C systems. All these capabilities offer a common and modularized platform to DI&C designers, software developers, cybersecurity analysts, and plant engineers for the evaluation of various design architectures of DI&C systems to support system design decisions in diversity and redundancy applications.

Section 2 describes the technical background of identification and quantification of risks associated with HSSSR DI&C systems. Section 3 documents the methodology and demonstration of a hazard analysis method developed in the framework. RESHA has been further developed with a capability to trace software failures in digital feedback pathways in highly redundant safety-critical DI&C systems; potential failures to a DI&C system are organized in a fault tree for clear visual and linear traceability. Case studies demonstrated the identification of digital failure mechanisms in key instrumentation, construction of the software fault tree in highly complex DI&C systems, and the identification of software single points of failure and key CCFs. Based on the software failure traceability, an innovative method was also developed to quantify probabilities of various software failure modes including CCFs in a DI&C system.

A limit of this method is related to scale. Specifically, at NPPs, there can be hundreds of parameters that are tracked by the IFP. Consistently modeling and tracing all parameters and how they can impact the system can be difficult and potentially intractable. However, the premise of adding UIFs in fault trees is intended to provide traceability support in a logical and linear causality fault tree model. Furthermore, UIFs are intended for safety case demonstration and should not be used for all parameters used in the IFP, rather only parameters that have been identified as being safety critical. In case study two, the advanced human-system interface, some key safety parameters identified were the ex-core coolant exit temperature and in-core temperature. These parameters were propagated through the information system and intermediate systems that directly modify these values and were assessed for UIFs. This limits the number of arbitrary parameters that need to be assessed for UIFs.

Section 4 documents the methodology development and demonstration of a multiscale quantitative reliability analysis approach of the proposed framework for DI&C risk assessment. The goal of the multiscale quantitative reliability analysis is to estimate the DI&C system reliability by calculating the integrated FT of DI&C systems obtained in the hazard analysis, then provide inputs for following consequence analysis. For the reliability analysis of large-scale DI&C systems, the quantitative small-scale reliability analysis of software and Type II interactions in DI&C systems are also included in the reliability analysis workflow.

The first step to any good reliability analysis for a DI&C system is the adequate collection and evaluation of design and requirement documentation. The next step is to estimate the failure probability of UCAs/UIFs identified in RESHA. Two methods have been developed for different application conditions: BAHAMAS for limited-data conditions and ORCAS for data-rich analysis. BAHAMAS was developed for the conditions with limited testing/operational data or for reliability estimations of software in early development stage. It can provide a rough estimation of failure probabilities to support the design of software and target DI&C systems even when data is very limited. Instead of relying on testing data, BAHAMAS assumes software failures can be traced to human errors in the SDLC and modeled with HRA. In BAHAMAS, a BBN is developed to provide a means of combining disparate causal factors and fault sources in the system, and HRA is applied to quantify the potential root human errors during SDLC. In contrast to BAHAMAS, ORCAS applies a white-box software invasive testing and modeling strategy to trace and identify the software defects that can potentially lead to software failures. The approach is a

root-cause analysis methodology focused on comprehensive testing and follows the ODC approach to determine testing sufficiency. ODC is also used to systematically classify the identified defects into specific software causality groups, thereby linking defects to potential software failure modes. The failure data collected from testing strategies is also combined with software reliability growth models and linear reliability models to quantify the software failure probability of specific UCAs.

After the small-scale reliability analysis of software and Type II interactions, a modified BFM is applied for the modeling and estimation of CCFs, especially for software CCFs. Finally, when the basic events of integrated FT are calculated, the failure probability of entire DI&C system can be estimated using FTA tools. Due to lack of software-based CCF data, the thorough verification and validation of this CCF modeling and estimation method has not been performed. A detailed guideline of the method is provided in this report to reduce the user effect. And a group of experts, rather than a single expert, is suggested to work on the scoring process. Even with uncertainty and subjectiveness from user effect, quantifying important software-based CCFs can be very useful for DI&C designers to make informed decisions on diversity design and optimization, and help reduce the cost of system development. Future collaborations with industry partners may afford our team the opportunity to validate and improve this method.

Section 5 describes consequence analysis based on INT-TRANS and relevant accident scenarios. The changes of CDF after adding integrated FTs of digital RTS, ESFAS, and safety-relevant HSI to the generic PWR ET models are compared and discussed. These integrated FTs include both software and hardware failures, particularly CCF, that may occur in a highly redundant safety-critical DI&C systems. Results show the CDF of INT-TRANS accident scenarios are reduced significantly. It should be noted that all the analyses are performed for the demonstration of the proposed framework, not for the evaluation of relevant systems. Results are obtained based on very limited design information and testing data.

Section 6 discusses the motivation, applicability, limitations, and potential benefits of applying the proposed framework on AI-aided control systems. A feasibility study was performed and showed that the framework developed in this project has the potential to be extended and modified for risk assessment and design optimization of advanced AI-aided control systems. Although further improvement and development is needed, the proposed framework can provide software design support through its native function, which can be used as the basis to support the development of an advanced integrated tool for the risk assessment of AI-aided control systems. The capability of a risk-informed analysis approach, like the proposed framework, is significantly needed to meet the industrial requirements for the deployment of intelligent DI&C systems (e.g., NAMAC system). A systematic hazard analysis should be applied to construct a knowledge base of plant anomaly causes, including all potential failure modes (e.g., digital and mechanical malfunctions, human errors, physical and cyberattacks, and environmental disturbance). Respective probabilities of these root causes should be provided by reliability analysis to evaluate the reliability of DT and the NAMAC system. Considering that some failures, especially software CCFs in DTs, may introduce some unanalyzed initiating events to the autonomous control systems, results from consequence analysis should be used to evaluate the impacts of these unanalyzed events and sequences on plant responses. All the information is useful for the real-time operation of ML-based DTs.

7.2. Future Works

The project expects to develop the capability of quantitative assessment to fill the technical gaps and follows the trend and the need for the digital modernization of existing NPPs. The work scope of this project in FY-23 is to further improve and demonstrate the framework based on the achievements from FY-19 to FY-22, particularly in demonstrating the proposed framework in the evaluation of various DI&C design architectures in terms of safety assurance and economic efficiencies, explore the applicability of performance-based approaches for the safety/reliability/resilience assurance of safety-critical DI&C systems, and the further development of a software CCF modeling method.

Key activities in FY-23 include:

- Improve current framework and complete a demonstration case of evaluation of various safety-critical DI&C design architectures in terms of safety assurance and economic efficiencies
- Develop capability to perform risk-informed and performance-based analysis of various DI&C design architectures
- Explore the applicability of performance-based approaches for the safety/reliability/resilience assurance of safety-critical DI&C systems
- Collaborate with university partners to evaluate, adjust, and integrate state-of-the-art methods in current framework for estimating probabilities of potential software CCFs in highly redundant and diverse safety-critical DI&C systems
- Collaborate with the industry to couple the LWRS-developed framework with existing risk-informed approaches to better support the optimization of safety-critical DI&C designs and upgrades.

8. REFERENCES

[1] H. Bao, H. Zhang and K. Thomas, "An Integrated Risk Assessment Process for Digital Instrumentation and Control Upgrades of Nuclear Power Plants," Idaho National Laboratory, Idaho Falls, ID, August 2019.

[2] H. Bao, H. Zhang and T. Shorthhill, "Redundancy-guided System-theoretic Hazard and Reliability Analysis of Safety-related Digital Instrumentation and Control Systems in Nuclear Power Plants," Idaho National Laboratory, Idaho Falls, ID, August 2020.

[3] H. Bao, T. Shorthill, E. Chen and H. Zhang, "Quantitative Risk Analysis of High Safety-significant Safety-related Digital Instrumentation and Control Systems in Nuclear Power Plants using IRADIC Technology," Idaho National Laboratory, Idaho Falls, ID, August 2021.

[4] U.S.NRC, SECY-18-0090 - Plan for Addressing Potential Common Cause Failure in Digital Instrumentation and Controls, Washington, D.C.: U.S.NRC, 2018.

[5] U.S.NRC, "Integrated Action Plan to Modernize Digital Instrumentation and Controls Regulatory Infrastructure," U.S.NRC, Washington, D.C., 2019.

[6] M. Qi, y. Kan, X. Li, X. Wang and M. Il, "Spurious activation and operational integrity evaluation of redundant safety instrumented systems," *Reliability Engineering & System Safety,* vol. 197, p. 106785, 2020.

[7] S. A. Arndt and A. Kuritzky, "Lessons Learned from the U.S. Nuclear Regulatory Commission's Digital System Risk Research," *Nuclear Technology,* vol. 173, no. 1, pp. 2-7, 2010.

[8] A. J. Clark, A. D. Williams, A. Muna and M. Gibson, "Hazard and Consequence Analysis for Digital Systems – A New Approach to Risk Analysis in the Digital Era for Nuclear Power Plants," in *Transactions of the American Nuclear Society*, Orlando, Florida, USA, 2018.

[9] N. G. Leveson and J. P. Thomas, STPA Handbook, March 2018.

[10] *Guidance for Addressing CCF in High Safety Significant Safety-related DI&C Systems,* NEI, July 1, 2021.

[11] U.S.NRC, "Regulatory Guide 1.174, An Approach for Using Probabilistic Risk Assessment in Risk-Informed Decisions on Plant-Specific Changes to the Licensing Basis," U.S.NRC, Washington, D.C., 2018.

[12] U.S.NRC, "SECY-93-087 - Policy, Technical, and Licensing Issues Pertaining to Evolutionary and Advanced Light-Water Reactor Designs," U.S. NRC, Washington, D.C., 1993.

[13] U.S.NRC, "Guidance for Evaluation of Defense in Depth and Diversity to Address Common-Cause Failure due to Latent Design Defects in Digital Safety Systems, BTP 7-19," U.S.NRC, Washington, 2021.

[14] U.S.NRC, "Staff Requirements - SECY-98-144 - White Paper on Risk-Informed Performance-Based Regulation," U.S.NRC, Washington, D.C., 1999.

[15] U.S.NRC, "Regulatory Guide 1.200, Acceptability of Probabilistic Risk Assessment Results for Risk-informed Activities," U.S.NRC, Washington, D.C., 2020.

[16] T. Shorthill, H. Bao, H. Zhang and H. Ban, "A Redundancy-Guided Approach for the Hazard Analysis of Digital Instrumentation and Control Systems in Advanced Nuclear Power Plants," *Nuclear Technology,* vol. 208, no. 5, pp. 892-911, 2022.

[17] H. Bao, T. Shorthill and H. Zhang, "Hazard Analysis for Identifying Common Cause Failures of Digital Safety Systems using a Redundancy-Guided Systems-Theoretic Approach," *Annals of Nuclear Energy,* vol. 148, p. 107686, 2020.

[18] T. Shorthill, H. Bao, Z. Hongbin and H. Ban, "A novel approach for software reliability analysis of digital instrumentation and control systems in nuclear power plants," *Annals of Nuclear Energy,* vol. 158, 2021.

[19] L. Lin, H. Bao and N. Dinh, "Uncertainty quantification and software risk analysis for digital twins in the nearly autonomous management and control systems: A review," *Annals of Nuclear Energy,* vol. 160, p. 108362, 2021.

[20] T. Aldemir, D. Miller, M. Stovsky, J. Kirschenbaum, P. Bucci, A. Fentiman and L. Mangan, "Current State of Reliability Modeling Methodologies for Digital Systems and Their Acceptance Criteria for Nuclear Power Plant Assessments," NUREG/CR-6901, U.S. Nuclear Regulatory Commission, Washington, DC, 2006.

[21] National Research Council, "Digital Instrumentation and Controls Systems in Nuclear Power Plants: Safety and Reliability Issues," The National Academies Press, Washington, 1997.

[22] K. H. &. N. P. C. L. Korea Electric Power Corporation, "Chapter 7: Instrumentation and Controls. Rev 3," Nuclear Regulatory Commission, Washington, 2018.

[23] International Electrotechnical Commission, "IEC 61508-1:2010," International Electrotechnical Commission, Geneva, 2010.

[24] International Electrotechnical Commission, "Root Cause Analysis, IEC 62740:2015," International Electrotechnical Commission, Geneva, 2015.

[25] International Electrotechnical Commission, "Failure modes and effects analysis (FMEA and FMECA), IEC 60812:2018," International Electrotechnical Commission, Geneva, 2018.

[26] M. Stamatelatos, W. Vesely, J. Dugan, J. Fragola, J. Minarick III and J. Railsback, "Fault Tree Handbook with Aerospace Applications Version 1.1," NASA, 2002.

[27] H. Bao, T. Shorthill and H. Zhang, "Redundancy-guided System-theoretic Hazard and Reliability Analysis of Safety-related Digital Instrumentation and Control Systems in Nuclear Power Plant," Idaho National Laboratory, Idaho, 2020.

[28] H. Zhang, H. Bao, S. Tate and E. Quinn, "An Integrated Risk Assessment Process of Safety-Related Digital I&C Systems in Nuclear Power Plants," *Nulcear Technology,* 2022.

[29] M. Jockenhovel-Barttfeld, S. Karg, C. Hessler and H. Bruneliere, "Reliability Analysis of Digital I&C Systems within the Verification and Validation Process," in *Probabilistic Safety Assessment and Management*, Los Angeles, CA, Septemmber 2018.

[30] U.S.NRC, "Systems Analysis Programs for Hands-on Integrated Reliability Evaluations (SAPHIRE) Version 8.0," U.S.NRC, Washington, D.C., USA, 2011.

[31] C. Elks, C. Deloglos, A. Jayakumar, A. Tantawy, R. Hite and S. Guatham, "Specification of a Bounded Exhaustive Testing Study for a Software-based Embedded Digital Device," Idaho National Laboratory, Idaho, 2018.

[32] G. D. Sirio, "ChibiOS/RT The Ultimate Guide," ChibiOS EmbeddedWare, 2020. [Online]. Available: https://www.chibios.org/dokuwiki/doku.php?id=chibios:documentation:books:rt:start. [Accessed 2022].

[33] A. Jayakumar, D. R. Kuhn, B. Simons, A. Collins, S. Gautham, R. Hite, R. N. Kacker, A. Rajagopala and C. Elks, "A Pseudo Exhaustive Software Testing Framework for Embedded Digital Devices in Nuclear Power," National Institute of Standards and Technology, Gaithersburg, 2021.

[34] E. Chen, H. Bao, H. Zhang, T. Shorthill and N. Dinh, "Systems-theoretic hazard analysis of digital human-system interface relevant to reactor trip," in *12th Nuclear Plant Instrumentation, Control and Human-Machine Interface Technologies*, 2021.

[35] A. D. Williams and A. J. Clark, "Using Systems Theoretic Perspectives for Risk-Informed Cyber Hazard Analysis in Nuclear Power Facilities," in *29th Annual INCOSE International Symposium*, Orlando, 2019.

[36] Electric Power Research Institute, "Digital Reliability Analysis Methodology," Electric Power Research Institute, Washington, 2021.

[37] J. D. Musa and K. Okumoto, "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement," Institute of Electrical and Electronics Engineers, Whippany, 1984.

[38] H. S. Son, H. G. Kang and S. C. Chang, "Procedure for Application of Software Reliability Growth Models to NPP PSA," *Nuclear Engineering and Technology,* vol. 41, no. 8, pp. 1065-1072, 2009.

[39] International Business Machines, "Orthogonal Defect Classification v5.2 for Software Design and Code," International Business Machines, Armonk, 2013.

[40] M. Butcher, H. Munro and T. Kratschmer, "Improving software testing via ODC: Three Case Studies," *IBM Systems Journal,* vol. 41, no. 1, pp. 31-44, 2002.

[41] R. N. K. Y. L. Richard D. Kuhn, "Practical Combinatorial Testing," National Institute of Standard Technology, 2010.

[42] K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski and L. K. Rierson, "A Practical Tutorial on Modified Condition / Decision Coverage," National Aeronautics and Space Administration, Washington, 2001.

[43] S. C. Reid and S. Shrivenham, "An Empirical Analysis of Equivalence Partitioning, Boundary Value Analysis and Random Testing," in *Proceedings Fourth International Software Metrics Symposium*, Como, 1997.

[44] National Institute of Standards and Technology, "Combinatorial Testing," National Institute of Standards and Technology, 14 July 2022. [Online]. Available: https://csrc.nist.gov/Projects/automated-combinatorial-testing-for-software. [Accessed 2 July 2022].

[45] R. Chillarege, "Orthoginal Defect Classification," in *Handbook of Software Reliability Engineering*, McGraw-Hill Book Company, 1996, pp. 359-399.

[46] J. Agnelo, N. Laranjeiro and J. Bernardino, "Using Orthoginal Defect Classification to Characterize NoSQL Database Defects," *The Journal of Systems and Software,* vol. 159, 2020.

[47] Zephr, "Zephr," Github, 2022. [Online]. Available: https://github.com/zephyrproject-rtos/zephyr. [Accessed 2022].

[48] CommaAi, "OpenPilot," Github, 2022. [Online]. Available: https://github.com/commaai/openpilot. [Accessed 2022].

[49] M. C. Kim, S. C. Jang and J. Ha, "Possibilities and Limitations of Applying Software Reliability Growth Models to Safety-Critical Software," *Nuclear Engineering and Technology,* vol. 39, no. 2, pp. 129-132, 2007.

[50] A. Goel and K. Okumoto, "A Time Dependent Error Detection Model for Software Reliability and Other Performance Measures," *IEEE Transactions on Reliability,* vol. 28, pp. 206-211, 1979.

[51] S. Yamada, M. Ohba and S. Osaki, "S-Shaped Reliability Growth Modeling for Software Error Detection," *IEEE Transactions on Reliability,* vol. 32, pp. 475-484, 1983.

[52] J. Musa, A. Iannino and K. Okumoto, Software Reliability, McGraw-Hill, 1987.

[53] S. Yamada, H. Ohtera and H. Narihisa, "Software Reliability Growth Models with Testing Effort," *IEEE Transactions on Reliability,* vol. 35, no. 1, pp. 19-23, 1986.

[54] A. Wood, "Software Reliability Growth Models," Hewlett-Packard, Palo Alto, 1996.

[55] T.-L. Chu, M. Yue, G. Martinez-Guridi and J. Lehner, "Review of Quantitative Software Reliability Methods," Brookhaven National Laboratory, 2010.

[56] A. Mosleh, "PRA: A Perspective on Strengths, Current Limitations, and Possible Improvements," *Nuclear Engineering and Technology,* vol. 46, no. 1, February 2014.

[57] Y. Shi, M. Li, S. Arndt and C. Smidts, "Metric-based software reliability prediction approach and its application," *Empirical Software Engineering,* no. 22, pp. 1579-1633, 2017.

[58] N. Fenton and M. Neil, Risk Assessment and Decision Analysis with Bayesian Networks, 2nd ed., CRC Press LLC, 2018.

[59] A. Bobbio, L. Portinale, M. Minichino and E. Ciancamerla, "Comparing Fault Trees and Bayesian Networks for Dependability Analysis," in *Proceedings of the 18th International Conference on Computer Safety, Reliability and Security, SAFECOMP99*, 1999.

[60] Institute of Electrical and Electronics Engineers, "ISO/IED/IEEE International Standard - Systems and software engineering--Vocabulary," in *ISO/IEC/IEEE 24765:2017(E)*, Institute of Electrical and Electronics Engineers, 2017, pp. 1-541.

[61] EPRI, "Methods for Assuring Safety and Dependability when Applying Digital Instrumentation and Control Systems," EPRI, Palo Alto, CA, 2016.

[62] Nuclear Energy Agency, "Failure Modes Taxonomy for Reliability Assessment of Digital Instrumentation and Control Systems for Probabilistic Risk Analysis," OECD NEA, 2014.

[63] M. Muhlheim and R. Wood, "Technical Basis for Evaluating Software-Related Common-Cause Failures," Oak Ridge National Laboratory, Oak Ridge, 2016.

[64] A. D. Swain and H. E. Guttmann, "Handbook of Human Reliability Analysis with Emphasis on Nuclear Power Plant Applications Final Report," NUREG/CR-1278. U.S. Nuclear Regulatory Commission, 1983.

[65] D. Gertman, H. Blackman, J. Marble, J. Byers and C. Smith, "The SPAR-H Human Reliability Analysis Method," NUREG/CR-6883, U.S. Nuclear Regulator Commission, Washington, DC, 2005.

[66] E. Hollnagel, Cognitive Reliability and Error Analysis Method (CREAM), Elsevier, 1998.

[67] J. Park, A. M. Arigi and J. Kim, "A Comparison of the quantification aspects of human reliability analysis methods in nuclear power plants," *Annals of Nuclear Energy,* vol. 133, pp. 297-312, 2019.

[68] T.-L. Chu, A. Varuttamaseni, M. Yue, S. J. Lee, H. G. Kang, J. Cho and S. Yang, "Developing a Bayesian Belief Network Model for Quantifying the Probability of Software Failure of a Protection System," NUREG/CR-7233, U.S. Nuclear Regulatory Commisison, 2018.

[69] H. G. Kang, S. H. Lee, S. J. Lee, T.-L. Chu, A. Varuttamaseni, M. Yue, S. Yang, H. S. Eom, J. Cho and M. Li, "Development of a Bayesian belief network model for software reliability quantification of digital protection systems in nuclear power plants," *Annals of Nuclear Energy,* vol. 120, pp. 62-73, 2018.

[70] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray and M.-Y. Wong, "Orthogonal Defect Classification - A Concept for In-Process Measurements," *International Institute of Electrical Engineers Transactions on Software Engineering,* vol. 18, no. 11, pp. 943-956, 1992.

[71] U. B. Kjaerulff and A. L. Madsen, Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis, 2nd ed., New York: Springer, 2013.

[72] International Council on Systems Engineering (INCOSE), Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, D. D. Walden, G. J. Roedler, K. J. Forsberg, R. D. Hamelin and T. M. Shortell, Eds., Hoboken, New Jersey, USA: John Wiley & Sons, 2015.

[73] ISO/IEC/IEEE, "ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle management -- Part 1: Guidelines for life cycel management," in *ISO/IEC/IEEE 24748-1:2018(E)*, 2018.

[74] "ISO/IEC/IEEE International Standard - Systems and software engineering -- Software life cycle processes," in *ISO/IEC/IEEE 12207:2017(E)*, 1 ed., 2017, pp. 1-157.

[75] R. Boring, "Fifty years of THERP and Human Reliability Analysis," in *Joint Probabilistic Safety Assessment and Management and European Safety and Reliability Conference*, 2012.

[76] J. Agnelo, N. Naranjeiro and J. Bernardino, "NoSQL odc dataset, results, and support code," March 2019. [Online]. Available: https://eden.dei.uc.pt/~cnl/papers/2019-jss.zip. [Accessed August 2022].

[77] R. A. Humphreys, "Assigning a Numerical Value to the Beta Factor Common Cause Evaluation," in *Reliability '87*, 1987.

[78] K. E. P. Corporation and L. Korea Hydro & Nuclear Power Co., "Software Program manual," Korea Electric Power Corporation; Korea Hydro & Nuclear Power Co., Ltd;, South Korea, 2018.

[79] BayesFusion, LLC., "GeNIe Modeler: Complete Modeling Freedom," 2022. [Online]. Available: https://www.bayesfusion.com/genie/. [Accessed August 2022].

[80] A. Mosleh, D. Rasmuson and F. Marshall, "Guidelines on Modeling Common-Cause Failures in Probabilistic Risk Assessment," NUREG/CR-5485, U.S. Nuclear Regulatory Commission, Washington, D.C., USA, 1998.

[81] A. O'Connor and A. Mosleh, "A General Cause Based Methodology for Analysis of Dependent Failures in System Risk and Reliability Assessments," University of Maryland, 2013.

[82] Z. Ma, R. F. Buell, J. K. Knudsen and S. Zhang, "Common-Cause Component Group Modeling Issues in Probabilistic Risk Assess," Idaho National Laboratory, Idaho Falls, ID, USA, 2020.

[83] D. M. Rasmuson and D. L. Kelly, "Common-cause failure analysis in event assessment," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability,* vol. 222, pp. 521-532, 2008.

[84] D. Kančev and M. Čepin, "A new method for explicity modelling of single failure evetn within different common cause failure groups," *Reliability Engineering and System Safety,* vol. 103, pp. 84-93, 2012.

[85] A. O'Connor and A. Mosleh, "Extending the Alpha Factor Model for Cause Based Treatment of Common Cause Failure Events in PRA and Event Assessment," in *PSAM 2014 - Probabilistic Safety Assessment and Management*, Honolulu, Hawaii, 2014.

[86] E. Higo, S. Soga and H. Miura, "Inter-unit common cause failure analysis based on data from intra-unit cases," in *ICONE2020 - Proceedings of the 2020 Conference on Nuclear Engineering*, Virtual, 2020.

[87] S. Soga, E. Higo and H. Miura, "A systematic approach to estimate an inter-unit common-cause failure probability," *Reliability Engineering and System Safety,* vol. 2015, 2021.

[88] B. D. Johnston, "A Structured Procedure for dependent Failure Analysis (DFA)," *Reliability Engineering,* vol. 19, pp. 125-136, 1987.

[89] V. P. Brand, Ed., UPM 3.1: A pragmatic approach to dependent failures assessment for standard systems, SRDA-R13, Warrington, UK: AEA Technology, Safety and Reliability Directorate, 1996.

[90] International Electrotechnical Commission, "Part 6: Guidelines on the application of parts 2 and 3," in *IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems*, 2 ed., International Electrotechnical Commission, 2010.

[91] Norwegian Oil and Gas Association, "070 Application of IEC 61508 and IEC 61511 in the Norwegian Petroleum Industry (Recommended SIL requirements)," Rev. 4, Norwegian Oil and Gas Association, 2020.

[92] H. Bao, H. Zhang, T. Shorthill and S. Lawrence, "Quantitative Evaluation of Common Cause Failures in High Safety-significant Safety-related Digital Instrumentation and Control Systems in Nuclear Power Plants," 7 April 2022. [Online]. Available: https://arxiv.org/abs/2204.03717. [Accessed 2022].

[93] C. Smidts, Y. Shi, M. Li, M. Kong and J. Dai, "A Large Scale Validation of a Methodology for Assessing Software Reliability, NUREG-7042," Nuclear Regulatory Commission, Washington, 2010.

[94] J.-j. Park, D.-i. Kim and D.-j. Kim, "Development of Qualified Indication and Alarm System-PAMI based on POSAFE-Q," 2008.

[95] J.-K. Lee, K.-I. Jeong, G.-O. Park and K.-Y. Sohn, "A Quantitative Reliability Analysis of FPGA-based Controller for applying to Nuclear Instrumentation and Control System," *The Journal of the Korea institute of electronic communication sciences,* vol. 9, pp. 1117-1123, 2014.

[96] D. of Defense, Military Handbook, Reliability Prediction of Electronic Equipment, Department of Defense, 1990.

[97] P. V. Varde, J. G. Choi, D. Y. Lee and J. B. Han, "Reliability Analysis of Protection System of Advanced Pressurized Water Reactor-APR 1400," Korea Atomic Energy Research Institute, South Korea, 2003.

[98] Z. Ma, C. Parisi, H. Zhang, D. Mandelli, C. Blakely, J. Yu, R. Youngblood and N. Anderson, "Plant-Level Scenario-Based Risk Analysis for Enhanced Resilient PWR – SBO and LBLOCA," Idaho National Laboratory, Idaho Falls, ID, 2018.

[99] Z. Ma, C. Parisi, C. Davis, J. Park, R. Boring and H. Zhang, "Risk-Informed Analysis for an Enhanced Resilient PWR with ATF, FLEX, and Passive Cooling," Idaho National Laboratory, Idaho Falls, ID, 2019.

[100] Z. Ma, C. Davis, C. Parisi, R. Dailey, J. Wang, S. Zhang, H. Zhang and M. Corradini, "Evaluation of the Benefits of ATF, FLEX, and Passive Cooling System for an Enhanced Resilient PWR Model," Idaho National Laboratory, Idaho Falls, ID, 2019.

[101] U.S.NRC, "Artificial Intelligence Strategic Plan," U.S.NRC, Washington, 2022.

[102] J. Ma and J. Jiang, "Applications of fault detection and diagnosis methods in nuclear power plants: A review," *Progress in Nuclear Energy,* vol. 53, no. 3, pp. 255-266, 2011.

[103] Nuclear Energy Institute, "Industry guideline for monitoring the effectiveness of maintenance at nuclear power plants, NUMARC 93-01," Nuclear Energy Institute, Washington, 2011.

[104] L. Lin, P. Athe, P. Rouxelin, M. Avramova, A. Gupta, R. Youngblood, J. Lane and N. Dinh, "Development and assessment of a nearly autonomous management and control system for advanced reactors," *Annals of Nuclear Energy,* vol. 150, p. 107861, 2021.

[105] D. Sculley, G. Holt, D. Golovin, E. Davydov and T. Phillips, "Hidden technical debt in machine learning systems," in *NeurIPS Proceedings*, Montreal, 2015.

[106] Z. Ma, H. Bao, S. Zhang, M. Xian and A. Mack, "Exploring Advanced Computational Tools and Techniques with Artificial Intelligence and Machine Learning in Operating Nuclear Plants," Nuclear Regulatory Commission, Washington, 2022.

[107] R. S. S. Kumar, D. O. Brien, K. Albert, S. Viljoen and J. Snover, "Failure Modes in Machine Learning Systems," 25 November 2019. [Online]. Available: https://arxiv.org/abs/1911.11034#. [Accessed 28 April 2022].

[108] H. N. A. Pham, "The Impact of Overfitting and Overgeneralization on the Classification Accuracy in Data Mining," LSU Digital Commons, Louisiana, 2010.

[109] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 15 December 2015. [Online]. Available: https://arxiv.org/abs/1512.03385. [Accessed August 2022].

[110] K. Z. Y. L. Z. L. Jingkang Yang, "Generalized Out-of-Distribution Detection: A Survey," 21 October 2021. [Online]. Available: https://arxiv.org/abs/2110.11334. [Accessed 29 April 2022].

APPENDIX A – HUMAN RELIABILITY ANALYSIS IN BAHAMAS

This appendix provides a demonstration of the THERP for the evaluation root nodes within the BAHAMAS BBN. The root nodes of BAHAMAS represent general stages of the SDLC. Quantification of the root nodes for the BBN requires that details of the SDLC for a given software of interest must be provided. This appendix details the application of HRA as part of the case study for the reliability analysis of BPs found within the four-division digital RTS shown in Figure 22. For details of the BPs and the RTS, the reader is advised to see earlier sections of this report.

The THERP manual indicates that the "general approach used for HRA has been to identify, analyze, and estimate HEPs for human tasks that system analysts and human reliability analysts determine could have a major impact on the system criteria of interest" [64]. The actions to identify, analyze, and estimate form the basis of steps used in this application of THERP: (1) identify the system of interest; (2) list and analyze the related human operations (perform a task analysis); and (3) determine probabilities. Concerning the analysis of the BP software, a description of the general assumptions is given below, followed by the details of the analysis.

A.1. General Assumptions

- At the time of this report, there is no HRA methodology directed specifically for software development practices. This work assumes that traditional HRA methods can be extended to address errors associated with human actions made during the SDLC (i.e., diagnostic activities and errors of commission and omission when following creating or developing written checklists and documents).
- For the case study, details of the SDLC were extracted from the publicly available APR 1400 software program manual [78]. The public version of the software program manual provides only limited detail; therefore, the results of this work in no way reflect the actual reliability of the APR-1400 design.
- Organizational details such as team and task assignments were redacted from the software program manual. Therefore, this work assumes the details and performers for the tasks within each stage of the SDLC. Persons involved in the SDLC include stakeholders, engineers, and project managers.
- The persons involved in the activities of the SDLC have the required training and expertise to perform their tasks.
- In addition to the tasks derived from the software program manual, gaps were supplemented by concepts given in IEEE 24748-1:2018(E).
- The work environment is optimal (i.e., there are no negative impacts to human performance associated with workload stress).
- THERP allows for the influence of dependency between tasks. Despite this capability, no dependency is considered for the SDLC tasks because the dependency model does not adequately reflect the dependency of tasks that are not immediate predecessors. Whereas, the dependency model appropriately fits typical sequential activities performed within a nuclear power plant, not all activities are purely sequential for the SDLC. Thus, SDLC task dependencies are not considered with this application of HRA.
- HEPs provided by THERP correspond to the median of a lognormal distribution. For additional details regarding assumptions of the THERP methodology, the reader is advised to visit [64].
- To better facilitate clarity for BAHAMAS, the review and recovery actions for SDLC tasks are accounted by the BBN directly, rather than by HRA analysis. Note: Had recovery actions been left for

THERP analysis (rather than the BBN), then THERP's dependency model would have been used for recovery within the failure path of individual tasks.

- THERP indicates when an appropriate HEP is unavailable, the nominal value of 3E-3 is assigned for errors of omission or for errors of commission (see page 20-13 in [64]). It is an assumption of the case study that for events not clearly omission- or commission-dominant, a union of the two will be used. Additionally, it is assumed that the omission and commission are mutually exclusive; therefore, the resulting value comes from P(A or B) = P(A)+P(B)=6E-3. Table 20-20 in [64] provides the EF (e.g., All EF values for HEP>.001 are 5 for activities that are not strictly step-by-step.)
- The unrecovered failure of any task results in an undetected error for the SDLC activity but does not prevent the completion of the remaining tasks.

A.2. Step 1: Identify the System of Interest

Generally, this process involves familiarization with a system, or event, of interest and its potential failure modes. More particularly, the failures of interest are those identified by RESHA (i.e., UCAs type A, B, C, and D). In this case, the event of interest is the failure of the BP to send a trip signal to the logic cabinets of the RTS system. Details of the BPs and RTS can be found in the main body of this report. The BPs are part of the larger RTS system for which there should be details regarding the SDLC. Details of the software development are assumed to follow the software program manual [78].

A.3. Step 2: Perform a Task Analysis

The THERP manual indicates that the goal of the task analysis is to identify the human-related actions associated with a task and the potential for human error. Task analysis involves determining the performance required of people and the influence that certain factors, such as environmental conditions, may have on them [64]. The task analysis should identify problem areas likely to cause human error (e.g., written procedures policies, practices, and people skills). For this work, the task analysis is directed at the stages of the SDLC. Each stage is treated as an activity of human action for which there are certain tasks. Each of the tasks relate to the activity that is necessary for developing the software. For example, human concept stage of the SDLC consists of a number of tasks pertaining to the development of key concepts for the SDLC.

BAHAMAS defines five general stages of the SDLC (i.e., concept, design, implementation, testing, and installation and maintenance). For each stage, there are tasks that must be completed by some human or group of humans. Thus, the task analysis for software development requires the identification of human tasks associated with the five general stages of the SDLC. The software program manual was assessed for tasks associated with the five stages of the SDLC. These tasks are given below.

1. Concept Stage Tasks
    a. Define the project purpose
    b. Identify hazards and risks
    c. Identify codes and standards
    d. Create a software management plan
    e. Create a software verification and validation plan
    f. Create a software configuration management plan
    g. Create a software development plan
    h. Create a software safety plan
2. Design Stage Tasks
    a. Define objectives of the software
    b. Define system architecture and interfaces
    c. Define operational modes and conditions
    d. Define component functionality

e. Define functions, parameters, and setpoints
f. Test parameters
g. Specify languages
h. Create a software requirements specification document
i. Develop prototypes
j. Select design solution
k. Create software design document
l. Create software integration plan

3. Implementation Stage Tasks
    a. Develop software function codes
    b. Develop software unit/module codes
    c. Develop system software code
    d. Integrate hardware and software
    e. Create a software installation plan
    f. Create software operational and maintenance plans
4. Testing Stage Tasks
    a. Create a software test plan
    b. Perform software function tests
    c. Perform software unit/module tests
    d. Perform system software tests
    e. Repair defects
    f. Document test/fixes
5. Installation and Maintenance Stage Tasks
    a. Install system
    b. Repair defects
    c. Create install/maintenance logs

The task analysis provides indication of the task, who performs the task, potential error sources for completion of the task, factors that may shape the likelihood of human error, and the HEP for the task. In this work, it is assumed that external factors, such as training and stress, are nominal, so they have no negative impact on the HEP for each task. Details of the task analysis for the concept stage of the SDLC are provided subsequently. The remaining task analyses are shown in subsequent condensed tables, but each follow similar scoring and justifications as those for the concept stage.

- Task 1: Define the project purpose
    - **Performer**: Group consisting of stakeholders, engineers, and managers
    - **Background**: The process of defining a project is similar to assessing or diagnosing a situation that is new. There may be similarities from past experiences, but ultimately, the scenario is a new problem that must be solved. THERP was designed for assessing activities within the context of a nuclear power plant; therefore, its application must fit similar conditions. THERP prescribes HEPs for group activities associated with diagnosis of abnormal events. The assumption is made that the diagnosis of an abnormal event is akin to solving a new problem. Experience, training, and expertise play a role in the diagnosis. But ultimately, the effort is not strictly routine. Hence, because each project is unique, we shall assume this is equivalent to diagnosing an abnormal event.
    - **Error sources**: Faulty diagnosis made by a group
    - **Score guidance:** Table 20-3 in [64] provides guidance diagnostic actions made by a team or group. Item 5 corresponds to 1 hour for time to diagnose. It is assumed that 1 hour is sufficient time for a group of qualified individuals to define the purpose of the project. The HEP is not modified because the adjustment rules (Table 12-5 in [64]) do not apply well to the SDLC activities.

- **HEP:** Lognormal distribution. Median 1E-4, EF 30.

- Task 2: Identify hazards and risks
  - **Performer**: Group consisting of engineers and managers
  - **Background**: The process of defining hazards is similar to identification and diagnostic activities. This task follows the same justification as Task 1.
  - **Error sources**: Faulty diagnosis made by the group.
  - **Score guidance**: Table 20-3 in [64]. The timing aspect for this task is different than Task 1. Efforts to perform risk and hazard analysis will take longer than 1 hr. The table maximum is approximately 24 hours. Early concept phase has limited detail (for example, it is unlikely that there will be in-depth instrumentation diagrams). It is assumed that the limited detail for this stage will lead to relatively quick risk and hazard assessments. These reasons correspond with item 5 from Table 20-3 in [64]. The HEP is not modified because the adjustment rules (Table 12-5 in [64]) do not apply well to the SDLC activities.
  - **HEP**: Lognormal distribution. Median 1E-4, EF 30.

- Task 3: Identify codes and standards
  - **Performer**: Team
  - **Background**: Assumed to be a diagnosis effort made by the team. The selection of standards relies on the correct diagnosis of the problem needs and attributes.
  - **Error sources**: Faulty diagnosis of problem needs to lead to the incorrect selection of codes and standards.
  - **Score guidance**: It is assumed that the limited detail for this stage will lead to relatively quick risk and hazard assessments. These reasons correspond with item 5 from Table 20-3 in [64]. The HEP is not modified because the adjustment rules (Table 12-5 in [64]) do not apply well to the SDLC activities.
  - **HEP**: Lognormal distribution. Median 1E-4, EF 30.

- Task 4: Create a software management plan
  - **Performer**: Design team member
  - **Background**: Creation of a management plan or set of instructions is straightforward. It will follow guidance from IEEE standards (e.g., 12207). The process generally will follow a checklist approach where the team decides how and which parts of the standards to follow.
  - **Error sources**: The primary mode of failure is omission of tasks or concepts based on those provided by written guidance/standards.
  - **Score guidance**: This is a normal event, which follows rule-based actions. The basic idea of the task is to use written materials to prepare written materials. This is a preparation task, with omission as the dominant form of failure, thus Table 20-5 in [64] is chosen. General assumptions indicate that stress and experience are optimal. Therefore, Table 20-16 in [64] (which accounts for stress and experience modification) indicates that no modification is needed for the HEP provided by Table 20-5 in [64].
  - **HEP**: Lognormal distribution. Median 3E-3, EF 5.

- Task 5: Create a software quality assurance plan
  - **Performer**: Design team member
  - **Background**: Creation of a management plan or set of instructions is straightforward. It will follow guidance from IEEE standards (e.g., 12207). The process generally will follow a checklist approach where the team decides how and which parts of the standards to follow.
  - **Error sources**: The primary mode of failure is omission of tasks or concepts based on those provided by written guidance/standards.

- **Score guidance**: This is a normal event, which follows rule-based actions. The basic idea of the task is to use written materials to prepare written materials. This is a preparation task, with omission as the dominant form of failure, thus Table 20-5 in [64] is chosen. General assumptions indicate that stress and experience are optimal. Therefore, Table 20-16 in [64] (which accounts for stress and experience modification) indicates that no modification is needed for the HEP provided by Table 20-5 in [64].
- **HEP**: Lognormal distribution. Median 3E-3, EF 5.

➢ Task 6: Software verification and validation plans
- **Performer**: Design team member
- **Background**: Creation of a management plan or set of instructions is straightforward. It will follow guidance from IEEE standards (e.g., 12207). The process generally will follow a checklist approach where the team decides how and which parts of the standards to follow.
- **Error sources**: The primary mode of failure is the omission of tasks or concepts based on those provided by written guidance/standards.
- **Score guidance**: This is a normal event, which follows rule-based actions. The basic idea of the task is to use written materials to prepare written materials. This is a preparation task, with omission as the dominant form of failure, thus Table 20-5 in [64] is chosen. General assumptions indicate that stress and experience are optimal. Therefore, Table 20-16 in [64] (which accounts for stress and experience modification) indicates that no modification is needed for the HEP provided by Table 20-5 in [64].
- **HEP**: Lognormal distribution. Median 3E-3, EF 5.

➢ Task 7: Software configuration management plans
- **Performer**: Design team member
- **Background**: Creation of a management plan or set of instructions is straightforward. It will follow guidance from IEEE standards (e.g., 12207). The process generally will follow a checklist approach where the team decides how and which parts of the standards to follow.
- **Error sources**: The primary mode of failure is omission of tasks or concepts based on those provided by written guidance/standards.
- **Score guidance**: This is a normal event, which follows rule-based actions. The basic idea of the task is to use written materials to prepare written materials. This is a preparation task, with omission as the dominant form of failure, thus Table 20-5 in [64] is chosen. General assumptions indicate that stress and experience are optimal. Therefore, Table 20-16 in [64] (which accounts for stress and experience modification) indicates that no modification is needed for the HEP provided by Table 20-5 in [64].
- **HEP**: Lognormal distribution. Median 3E-3, EF 5.

➢ Task 8: Software development plans
- **Performer**: Design team member
- **Background**: Creation of a management plan or set of instructions is straightforward. It will follow guidance from IEEE standards (e.g., 12207). The process generally will follow a checklist approach where the team decides how and which parts of the standards to follow.
- **Error sources**: The primary mode of failure is the omission of tasks or concepts based on those provided by written guidance/standards.
- **Score guidance**: This is a normal event, which follows rule-based actions. The basic idea of the task is to use written materials to prepare written materials. This is a preparation task, with omission as the dominant form of failure, thus Table 20-5 in [64] is chosen. General assumptions indicate that stress and experience are optimal. Therefore, Table 20-16 in [64] (which accounts

for stress and experience modification) indicates that no modification is needed for the HEP provided by Table 20-5 in [64].
- **HEP**: Lognormal distribution. Median 3E-3, EF 5.

➢ Task 9: Software safety plans
- **Performer**: Design team member
- **Background**: Creation of a management plan or set of instructions is straightforward. It will follow guidance from IEEE standards (e.g., 12207). The process generally will follow a checklist approach where the team decides how and which parts of the standards to follow.
- **Error sources**: The primary mode of failure is omission of tasks or concepts based on those provided by written guidance/standards.
- **Score guidance**: This is a normal event, which follows rule-based actions. The basic idea of the task is to use written materials to prepare written materials. This is a preparation task, with omission as the dominant form of failure, thus Table 20-5 in [64] is chosen. General assumptions indicate that stress and experience are optimal. Therefore, Table 20-16 in [64] (which accounts for stress and experience modification) indicates that no modification is needed for the HEP provided by Table 20-5 in [64].
- **HEP**: Lognormal distribution. Median 3E-3, EF 5.

Table 63. HEPs for design stage tasks.

Task	**Score**	**Justification**
Define objectives of the software	Median 1E-5, EF 30	Table 20-3 item 6. More than 24 hrs. to perform diagnosis is reasonable, but HEP exceeding 24 hrs. provides an unrealistic decrease in HEP.
Define system architecture and interfaces	Median 1E-4, EF 30	Table 20-3 item 5. More than 24 hrs. to perform diagnosis is reasonable, but HEP exceeding 1 hr. provides an unrealistic decrease in HEP for this complex task.
Define operational modes and conditions	Median 1E-4, EF 30	Table 20-3 item 5. More than 24 hrs. to perform diagnosis is reasonable, but HEP exceeding 1 hr. provides an unrealistic decrease in HEP for this complex task.
Define component functionality	Median 6E-3, EF 5	Errors of commission and omission are relevant. See general assumptions.
Define functions, parameters, and setpoints	Median 6E-3, EF 5	Errors of commission and omission are relevant. See general assumptions.
Test parameters	Median 6E-3, EF 5	Errors of commission and omission are relevant. See general assumptions.
Specify languages,	Median 6E-3, EF 5	Errors of commission and omission are relevant. See general assumptions.
Create software requirements specification document	Median 6E-3, EF 5	Errors of commission and omission are relevant. See general assumptions.
Develop prototypes	Median 6E-3, EF 5	Errors of commission and omission are relevant. See general assumptions.

Select design solution	Median 1E-4, EF 30	Table 20-3, item 5. More than 24 hrs. to perform diagnosis is reasonable, but HEP exceeding 1 hr. provides an unrealistic decrease in HEP for this task.
Create Software design document	Median 6E-3, EF 5	Errors of commission and omission are relevant. See general assumptions.
Create software integration plan	Median 3E-3, EF 5	This task relies on guidance from standards. Following Table 20-5. Omission of guidance from standards is the dominant error source.

Table 64. HEPs for implementation stage tasks.

Task	**Score**	**Justification**
Develop software function codes	Median 6E-3, EF 5	Software design and development activities will introduce variability, so omission and commission are relevant. See general assumptions.
Develop software unit/module codes	Median 6E-3, EF 5	Software design and development activities will introduce variability, so omission and commission are relevant. See general assumptions.
Develop system software code	Median 6E-3, EF 5	Software design and development activities will introduce variability, so omission and commission are relevant. See general assumptions.
Integrate hardware and software	Median 6E-3, EF 5	Software design and development activities will introduce variability, so omission and commission are relevant. See general assumptions.
Create software installation plan	Median 6E-3, EF 5	Software design and development activities will introduce variability, so omission and commission are relevant. See general assumptions.
Create software operational and maintenance plans	Median 3E-3, EF 5	This task relies on guidance from standards. Following Table 20-5. Omission of guidance from standards is the dominant error source.

Table 65. HEPs for testing stage tasks.

Task	**Score**	**Justification**
Create software test plan	Median 3E-3, EF 5	This task relies on guidance from standards. Following Table 20-5. Omission of guidance of tests or specific plans are the dominant error source.
Perform software function tests	Median 6E-3, EF 5	Performance of software testing activities will vary, so omission and commission are relevant. See general assumptions.
Perform software unit/module tests	Median 6E-3, EF 5	Performance of software testing activities will vary, so omission and commission are relevant. See general assumptions.

Perform system software tests	Median 6E-3, EF 5	Performance of software testing activities will vary, so omission and commission are relevant. See general assumptions.
Repair defects	Median 6E-3, EF 5	Performance of software testing activities will vary, so omission and commission are relevant. See general assumptions.
Document test/fixes	Median 6E-3, EF 5	Performance of software testing activities will vary, so omission and commission are relevant. See general assumptions.

Table 66. HEPs for installation and maintenance tasks.

Task	**Score**	**Justification**
Install system	Median 6E-3, EF 5	Performance of software installation activities will vary, so omission and commission are relevant. See general assumptions.
Repair defects	Median 6E-3, EF 5	Performance of software repair or fixes will vary, so omission and commission are relevant. See general assumptions.
Create Install/maintenance logs	Median 6E-3, EF 5	Creation of log files will contain both omission and commission errors. See general assumptions.

A.4. Step 3: Determine Probabilities

The goal of this step of the HRA is to evaluate the SDLC stage HEP given the influence of each of the tasks for that stage. THERP relies on an HRA event tree (ET) where each of the tasks are assembled in a manner like that shown in Figure 55 [64]. The ET limbs represent binary actions for "correct" or "incorrect," and the probability for each is defined in the task. For this work we relied on a Monte Carlo sampling approach to evaluate the total HEP for each SDLC stage. Figure 56 shows the convergence of the HEP for SDLC concept stage. The remaining stages converge in a similar fashion. The results of the analysis are shown in Table 38.

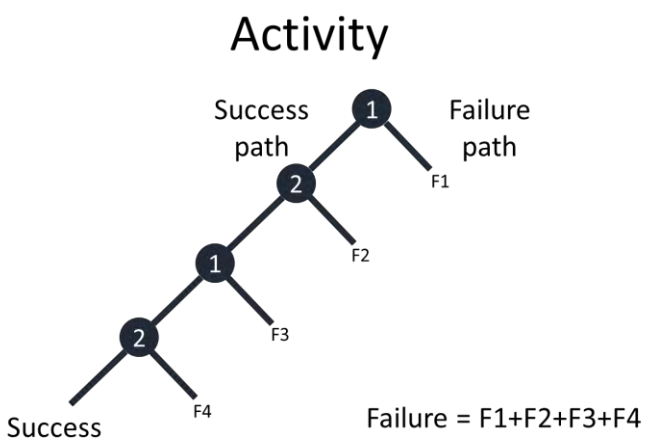

Figure 55. Simple HRA event tree

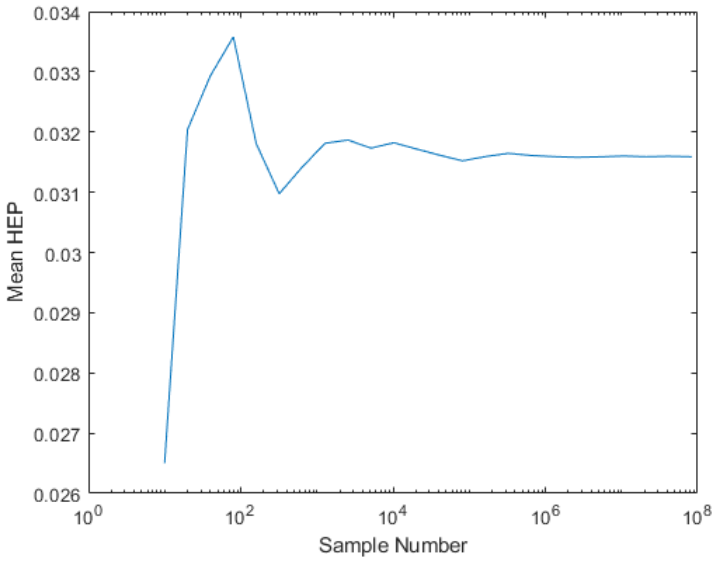

Figure 56. Convergence of the concept stage mean HEP.

APPENDIX B – SELECTED ACCIDENT SCENARIOS FOR CONSEQUENCE ANALYSIS

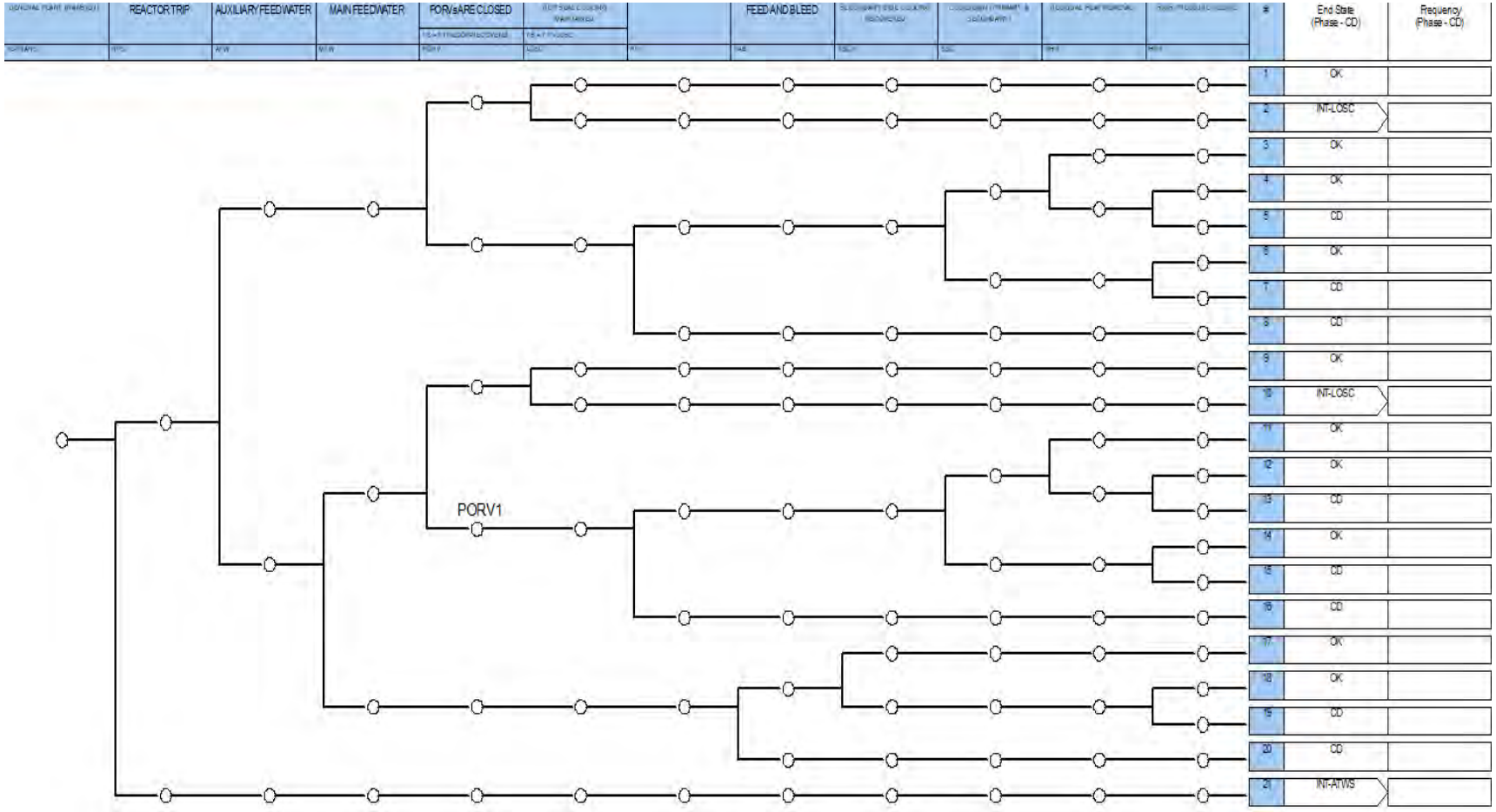

Figure 57. Generic PWR ET for general plant transient (INT-TRANS).

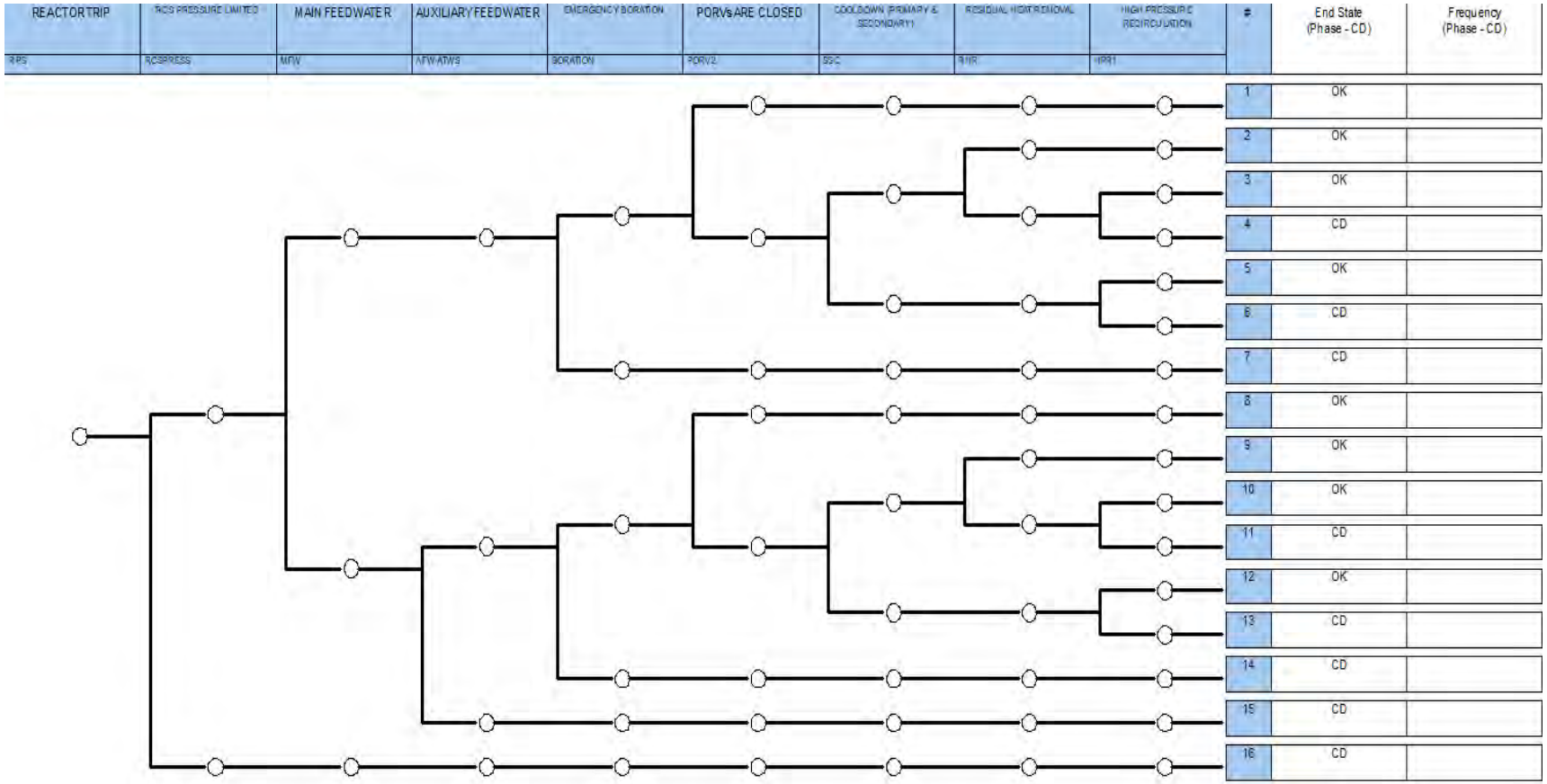

Figure 58. Generic PWR ET for INT-ATWS.

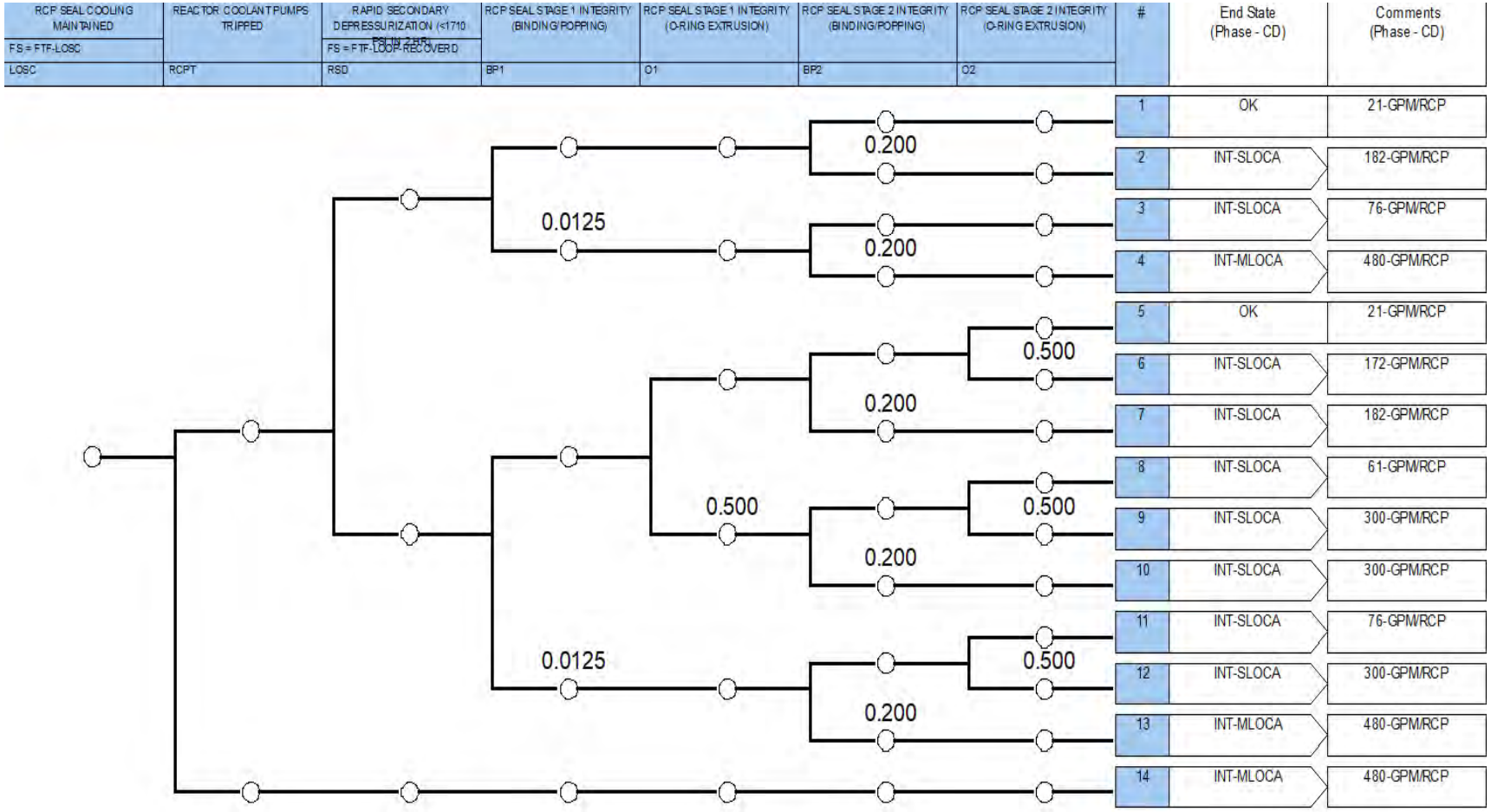

Figure 59. Generic PWR ET for loss-of-seal cooling (INT-LOSC).

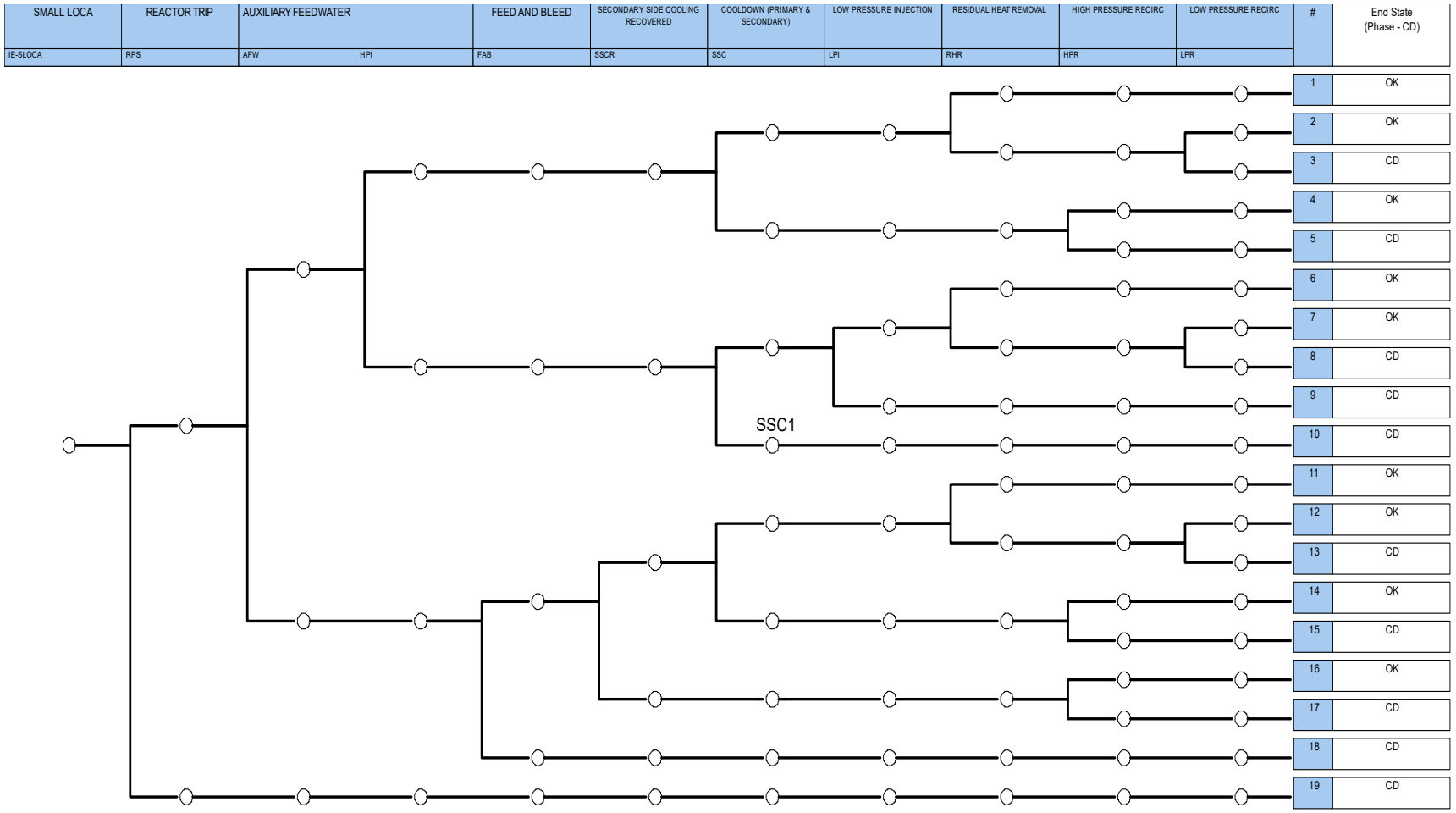

Figure 60. Generic PWR ET for small-break LOCA (INT-SLOCA).

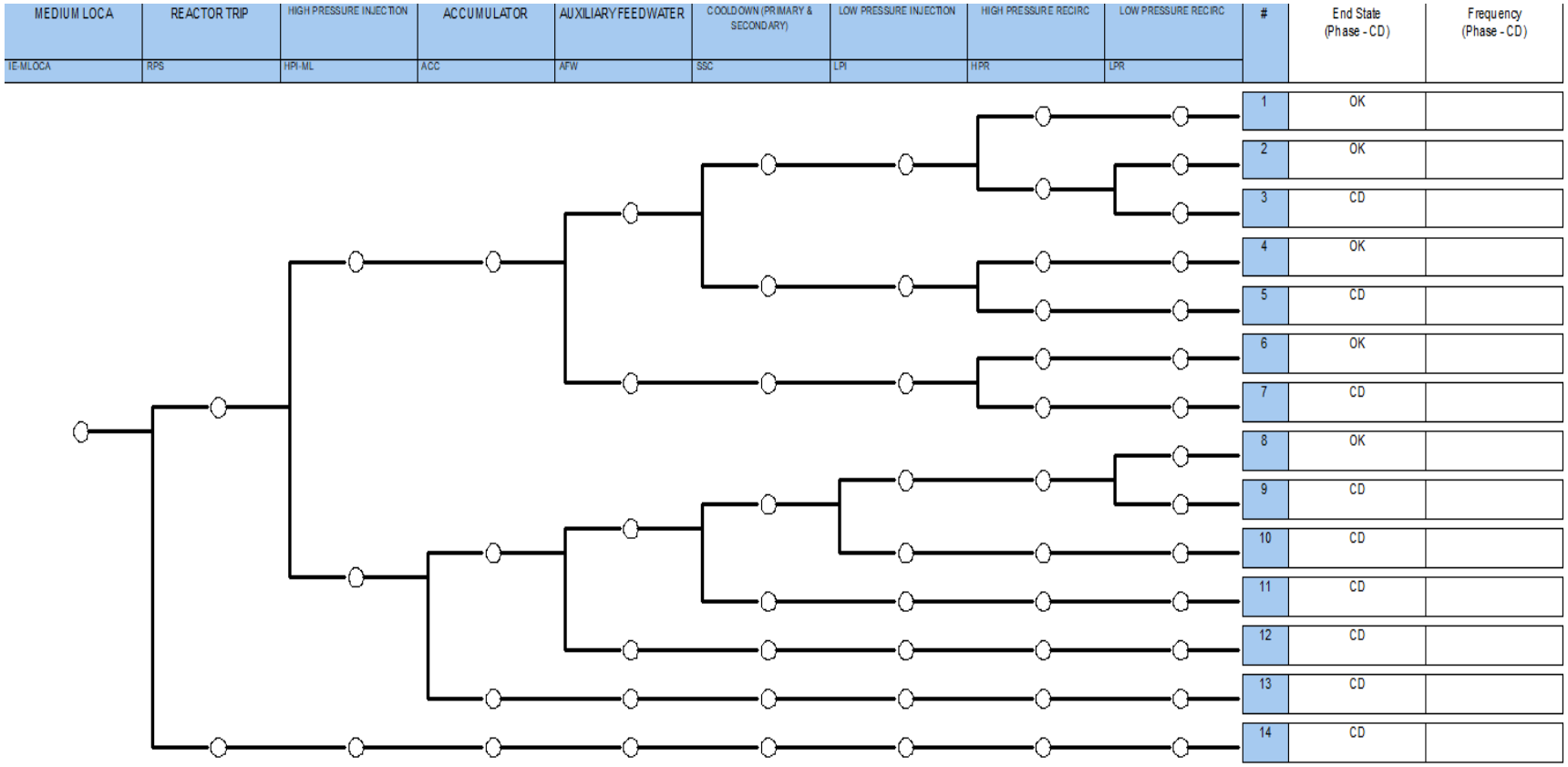

Figure 61. Generic PWR ET for medium-break LOCA (INT-MLOCA).