INL/RPT-23-74412

Light Water Reactor Sustainability Program

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

September 2023

U.S. Department of Energy

Office of Nuclear Energy

INL/RPT-23-74412

Light Water Reactor Sustainability Program

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

Han Bao[1], Tate Shorthill[2], Edward Chen[3], Jooyoung Park[1], Jisuk Kim[1], Gulcin Sarici Turkmen[4], Heng Ban[2], Nam Dinh[3], Tunc Aldemir[4], Sai Zhang[1], Svetlana Lawrence[1]

September 2023

[1]Idaho National Laboratory
Idaho Falls, ID 83415
[2]University of Pittsburgh
Pittsburgh, PA 152601
[3]North Carolina State University
Raleigh, NC 27695
[4]The Ohio State University
Columbus, OH 43210

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.lwrs.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Light Water Reactor Sustainability Program

EXECUTIVE SUMMARY

This report documents activities performed by Idaho National Laboratory (INL) during fiscal year (FY) 2023 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 risk-informed framework was proposed for this strategy, which aims to (1) provide a best-estimate, risk-informed capability to quantitatively estimate the safety margin obtained from plant modernization, especially from safety-related 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 safety-related DI&C systems, and (5) reduce uncertainty in costs, and support the deployment of DI&C systems in the nuclear power plants (NPPs).

To achieve these technical goals, the proposed framework in this project provides a means to address relevant technical issues by: (1) defining a risk-informed analysis process for the 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 in NPPs.

Adding diversity within a system or components is the primary means to eliminate and mitigate CCFs, but diversity also increases system complexity and may not address all sources of systematic failures. Optimization of 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 safety-related DI&C systems of NPPs and supporting relevant design optimization, the proposed framework provides:

- A best-estimate, risk-informed capability to address new technical digital issues quantitatively, focusing on software CCFs in safety-related 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 in addressing the risk-informed alternatives for evaluating CCFs in safety-related 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 2023 are focused on refining current methods on software CCF modeling and estimation and exploring additional innovative approaches to risk assessment of DI&C systems to enable a more comprehensive and complete assessment of various safety-related DI&C design architectures.

The primary audience of this report is DI&C designers, engineers, and probabilistic risk assessment 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 methodology, not for the evaluation of an actual digital control system. Results are obtained based on limited design information and testing data.

CONTENTS

FIGURES

TABLES

ACRONYMS

AI	artificial intelligence
BAHAMAS	Bayesian and HRA-Aided Method for the Reliability Analysis of Software
BBN	Bayesian Belief Network
BFM	beta factor model
BP	bistable processor
CBDT	Cause-based Decision Tree
CCCG	common cause component group
CCF	common cause failure
CCS	component control system
CDF	core damage frequency
CF	coverage factor
CFG	Control Flow Path
CIM	component interface module
CPCS	Core protection calculator system
CPS	computer-based procedure system
CREAM	Cognitive reliability and error analysis method
DET	Dynamic event tree
DFG	Data Flow Graph
DI&C	digital instrumentation and control
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
DT	Digital twin
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
FNN	Feedforward neural network
FT	fault tree
FTA	fault tree analysis

FV	Fussell-Vesely
FY	Fiscal Year
GC	group controller
HAZCADS	Hazards and Consequence Analysis for Digital Systems
HEP	Human error probability
HPI	high-pressure injection
HRA	human reliability analysis
HSI	Human-system-interface
ID	In-distribution
IDHEAS-ECA	Integrated Human Event Analysis System for Event and Condition Assessment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IFPD	information flat panel display
INL	Idaho National Laboratory
IPS	information processing system
LADDR	Laplacian distributed decay for reliability
LC	loop controller
LCL	local coincidence logic
LDP	large display panel
LOCA	loss-of-coolant accident
LOSC	loss-of-seal cooling
LP	logic processor
LPI	low-pressure injection
LWR	light water reactor
LWRS	Light Water Reactor Sustainability
MCR	main control room
MCS	minimum cut set
MDAFP	motor-driven auxiliary feedwater pumps
ML	machine learning
MMI	man-machine interface
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
OOD	Out-of-distribution
ORCAS	Orthogonal-defect Classification for Assessing Software reliability
PIF	Performance influencing factor
PPS	Plant protection system
PRA	probabilistic risk assessment
PSF	Performance-shaping factor
PV	Process Variable
PWR	pressurized-water reactor
PWROG	Pressurized Water Reactor Owners Group
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 coolant system
RESHA	Redundancy-guided Systems-theoretic Hazard Analysis
RG	Regulatory Guide
RISA	Risk Informed Systems Analysis
RPS	reactor protection system
RSR	reserve shutdown room
RTB	reactor trip breaker
RTS	reactor trip system
RTSS	reactor trip switchgear system
SAPHIRE	Systems Analysis Programs for Hands-on Integrated Reliability Evaluations
SDLC	software development life cycle
SIL	Safety integrity levels
SP	selective processor
SPADES+	safety parameter display and evaluation system+
SPAR-H	Standardized Plant Analysis Risk-Human Reliability Analysis method
SSC	system, structure, and component
STPA	systems-theoretic process analysis
UCA	unsafe control action
UIF	unsafe information flow
U.S.	United States
V&V	Verification and validation

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

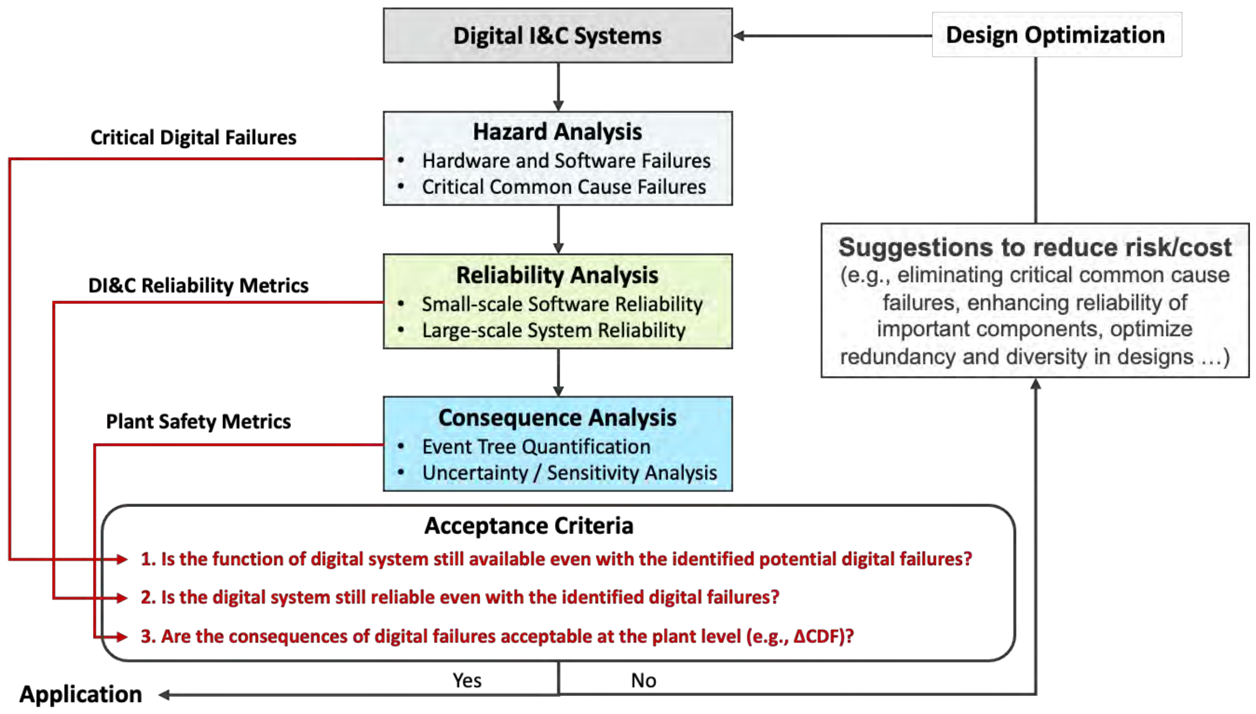
1. INTRODUCTION

This report documents the activities performed by Idaho National Laboratory (INL) during fiscal year (FY) 2023 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], [4], [5]. 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 lifetime can be extended, and the plants can continue to operate safely, efficiently, and economically.

As one of the LWRS program's R&D pathways, the 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 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, licensable, and secure DI&C technologies for digital upgrades and designs [1]. As shown in Figure 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 safety-related 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 safety-related DI&C systems
- Reduce uncertainty in costs and support integration of DI&C systems at NPPs.



Digital I&C Systems
Design Optimization
Critical Digital Failures
Hazard Analysis
• Hardware and Software Failures
• Critical Common Cause Failures
DI&C Reliability Metrics
Reliability Analysis
• Small-scale Software Reliability
• Large-scale System Reliability
Suggestions to reduce risk/cost (e.g., eliminating critical common cause failures, enhancing reliability of important components, optimize redundancy and diversity in designs …)
Plant Safety Metrics
Consequence Analysis
• Event Tree Quantification
• Uncertainty / Sensitivity Analysis
Acceptance Criteria
1. Is the function of digital system still available even with the identified potential digital failures?
2. Is the digital system still reliable even with the identified digital failures?
3. Are the consequences of digital failures acceptable at the plant level (e.g., ΔCDF)?
Yes
No
Application

Figure 1. Schematic of the proposed risk assessment framework for safety-critical DI&C systems.

The proposed risk assessment framework for DI&C systems is shown in Figure 2. In this framework, a redundancy-guided systems-theoretic method for hazard analysis (RESHA) was developed for safety-related DI&C systems to support I&C designers and engineers to address both hardware and software common cause failures (CCFs) and qualitatively analyze their effects on system availability [6] [7]. It also provides a technical basis for implementing reliability and consequence analyses of unexpected software failures and supporting the optimization of defense-in-depth applications in a cost-efficient way. RESHA integrates systems-theoretic process analysis (STPA) [8], fault tree analysis (FTA), and hazards and consequence analysis for digital systems (HAZCADS) [9] methodologies to effectively identify software CCFs in complex systems with multiple levels of redundancy. More specifically, STPA is reframed 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 [6] and a four-division digital ESFAS [7].

The second part in risk analysis is the reliability analysis which includes tasks of (1) quantifying the probability of basic events of the integrated fault tree (FT) from the hazard analysis; (2) estimating the probability of consequences resulting from digital system failures. In the proposed framework, two methods have been developed: the Bayesian and human-reliability-analysis-aided method for the reliability analysis of software (BAHAMAS) [10] and orthogonal-defect classification (ODC) for assessing software reliability (ORCAS) [11]. BAHAMAS is applicable in situations with limited data conditions (e.g., early stage of system development), and ORCAS is applicable for analyses when significant amount of data is available (e.g., a fully developed system that underwent verification and validation or a system with significant length of operating experience).

Finally, the consequence analysis is conducted to quantitatively evaluate the impact of digital failures on plant overall risks by assessing affected behaviors and responses. Some digital-based failures may initiate an event or scenario that was not analyzed before (e.g., a failure mode only applicable to a digital system), which could challenge plant safety. Uncertainty and sensitivity analyses are performed to evaluate the safety and risk significance of target components and subsystems to DI&C system reliability and plant safety [12].

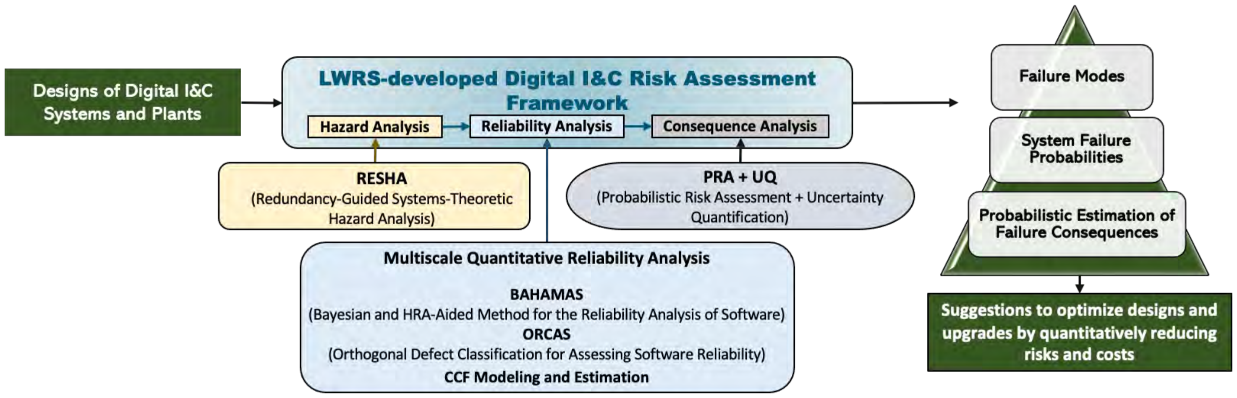

Designs of Digital I&C Systems and Plants
LWRS-developed Digital I&C Risk Assessment Framework
Hazard Analysis
Reliability Analysis
Consequence Analysis
RESHA (Redundancy-Guided Systems-Theoretic Hazard Analysis)
PRA + UQ (Probabilistic Risk Assessment + Uncertainty Quantification)
Multiscale Quantitative Reliability Analysis
BAHAMAS (Bayesian and HRA-Aided Method for the Reliability Analysis of Software)
ORCAS (Orthogonal Defect Classification for Assessing Software Reliability)
CCF Modeling and Estimation
Failure Modes
System Failure Probabilities
Probabilistic Estimation of Failure Consequences
Suggestions to optimize designs and upgrades by quantitatively reducing risks and costs

Figure 2. The flexible and modularized structure of the proposed risk assessment framework for safety-related DI&C systems.

This project's research efforts from FY 2019 through FY 2022 were focused on methodology development and demonstration of the proposed framework. In FY 2023, the framework reached a point for a technical peer review and stakeholder feedback. Peer review activities included coordinating the reviews performed by a group of industry stakeholders, documenting the peer review feedback, and providing resolutions and responses to the peer review comments. Comments from technical peer reviewers and the resolutions and responses to these comments are outlined in the peer review report [13] published in March 2023. Besides, collaborations with the nuclear industry have been initiated to support the reliability and risk assessment of their DI&C systems by using the proposed framework. Complete analysis and results have been documented in [14].

This report outlines R&D focused on refining current methods on software CCF modeling and estimation and exploring additional innovative approaches to risk assessment of DI&C systems to enable a more comprehensive and complete assessment of various safety-related DI&C design architectures. The remaining sections of the report are organized as follows: Section 2 describes the methodology refinement of the proposed framework and methods based on feedback during the collaboration with the industrial partners and the comments from the technical peer review. Section 3 evaluates the feasibility of current methods on cross-system CCF analysis between a representative four-division digital RTS and ESFAS. Section 4 proposes an innovative approach on the risk assessment of human-system-interface (HSI) by integrating RESHA and human reliability analysis (HRA) methods. Section 5 investigates the use of a dynamic probabilistic risk assessment (PRA) approach on evaluating the impacts of software CCFs to plant safety. Section 6 presents a novel method for evaluating the model agnostic reliability of potential machine learning (ML) models integrated in DI&C systems. Section 7 summarizes results of the R&D performed in collaboration with the Pressurized Water Reactor Owners Group (PWROG) in FY 2023 focused on the risk evaluation of a DI&C safety actuation system in use at a nuclear utility. Section 8 outlines conclusions and future work of this project. Appendixes A, B and C respectively document the application guides of RESHA, ORCAS and BAHAMAS.

2. METHODOLOGY REFINEMENT AND IMPROVEMENT

This section describes the methodology refinement of the proposed framework and methods based on feedback from the collaboration with the industrial partners and the comments from the technical peer review. Methodology improvement will be continued in the next fiscal year. Sections 2.1 and 2.2 describe the new methodology of the RESHA, BAHAMAS, ORCAS, and CCF modeling approach for software CCF analysis. Section 2.3 presents an approach to modeling postulated software CCFs of diverse DI&C systems using these methods.

2.1 Redundancy-guided System-theoretic Hazard Analysis

The RESHA method is an FTA-based method that incorporates STPA to identify the potential hazards and failures of DI&C systems. Reference [15] 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. Software should not be analyzed in isolation from the digital system. In addition to the inner failures of software, failures in Type II interactions should be considered in the risk analysis of a DI&C system.

RESHA incorporates the concepts from FTA, STPA, HAZCADS, and CCF modeling to support hazard analysis. 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 critical failures, including CCFs, in the DI&C design; (2) an integrated FT including both hardware and software failures, both independent failures and CCFs; and (3) hazard preventive strategies. The acceptance criterion of risk evaluation for the RESHA is: "is the function of digital system still available even with the identified potential digital failures?" In other words, are there any critical failures or failure combinations existing in the system that may lead to the DI&C system completely losing its function? A seven-step process, shown in [4], illustrates the workflow of RESHA in the proposed framework for the hazard analysis of DI&C systems, especially for software CCF analysis of highly redundant safety-related DI&C systems.

Compared with the RESHA workflow described in [4] and [5], the major revisions from this year research include:

- Added the identification of unsafe information flows (UIFs) in Step 3 with unsafe control actions (UCAs) based on redundancy-guided application of STPA. UIFs are defined as failures in the feedback pathway of the control structure, whereas an UCA is a failure in the controller pathway. They are considered one class of casual factors for UCAs under the STPA methodology.
- Detailed CCF identification in Step 5 to better serve CCF analysis efforts. The previous version of RESHA provides a discussion on CCFs and the importance of identifying software-based coupling factors, but the identification of coupling mechanisms, which is a vital part of CCF analysis, was not fully discussed. To address this, Step 5 shown in Figure 1 has been modified. The purpose of Step 5 is to identify potential CCFs and add them to the FT. Digital systems, particularly those employed for safety, contain multiple layers of redundancy. Redundant components may share common hardware and software features making them susceptible to CCFs. Based on the redundancy-guided nature of RESHA, the first task of Step 5 is to identify the redundant (or functionally redundant in the case of diversity) components, or elements of the software system. From this point on, the selection of coupling mechanisms can identify potential subgroups that may fail together.

The new workflow of RESHA is shown in Figure 3, more details and descriptions can be found in A Appendix A.

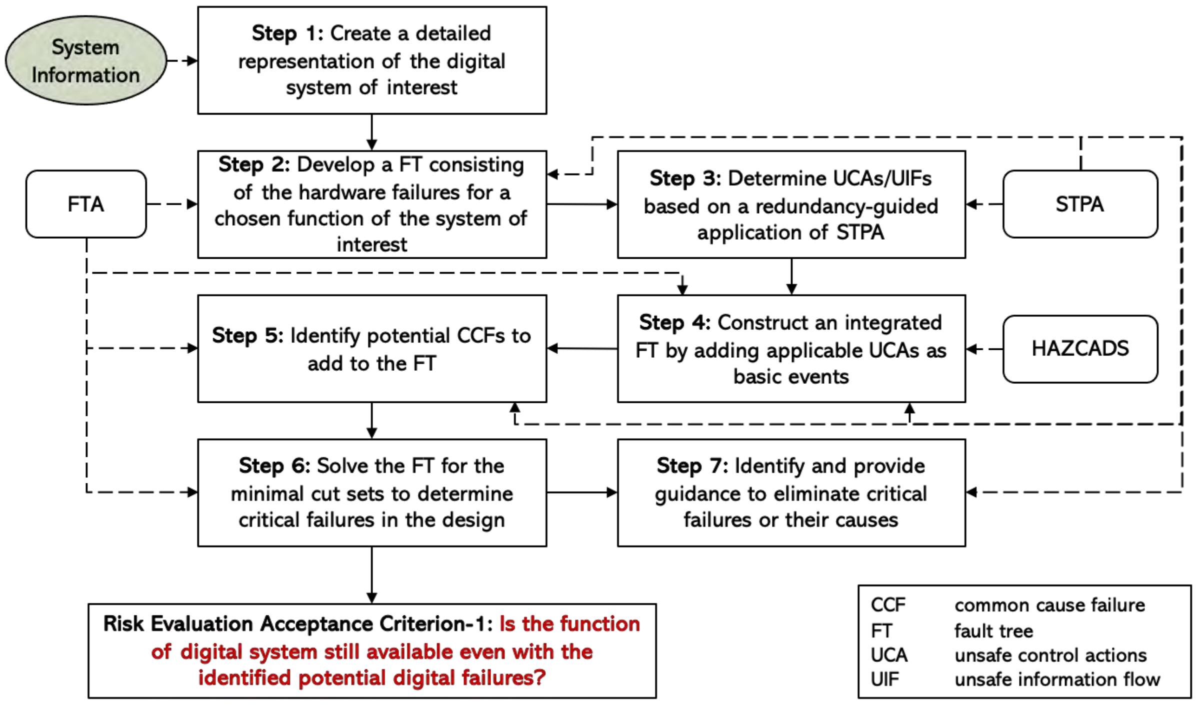

System Information
Step 1: Create a detailed representation of the digital system of interest
FTA
Step 2: Develop a FT consisting of the hardware failures for a chosen function of the system of interest
Step 3: Determine UCAs/UIFs based on a redundancy-guided application of STPA
STPA
Step 5: Identify potential CCFs to add to the FT
Step 4: Construct an integrated FT by adding applicable UCAs as basic events
HAZCADS
Step 6: Solve the FT for the minimal cut sets to determine critical failures in the design
Step 7: Identify and provide guidance to eliminate critical failures or their causes
Risk Evaluation Acceptance Criterion-1: Is the function of digital system still available even with the identified potential digital failures?
CCF common cause failure
FT fault tree
UCA unsafe control actions
UIF unsafe information flow

Figure 3. Workflow of the RESHA in the proposed framework for the hazard analysis of DI&C systems (derived from [4]).

2.2 Multiscale Quantitative Reliability Analysis

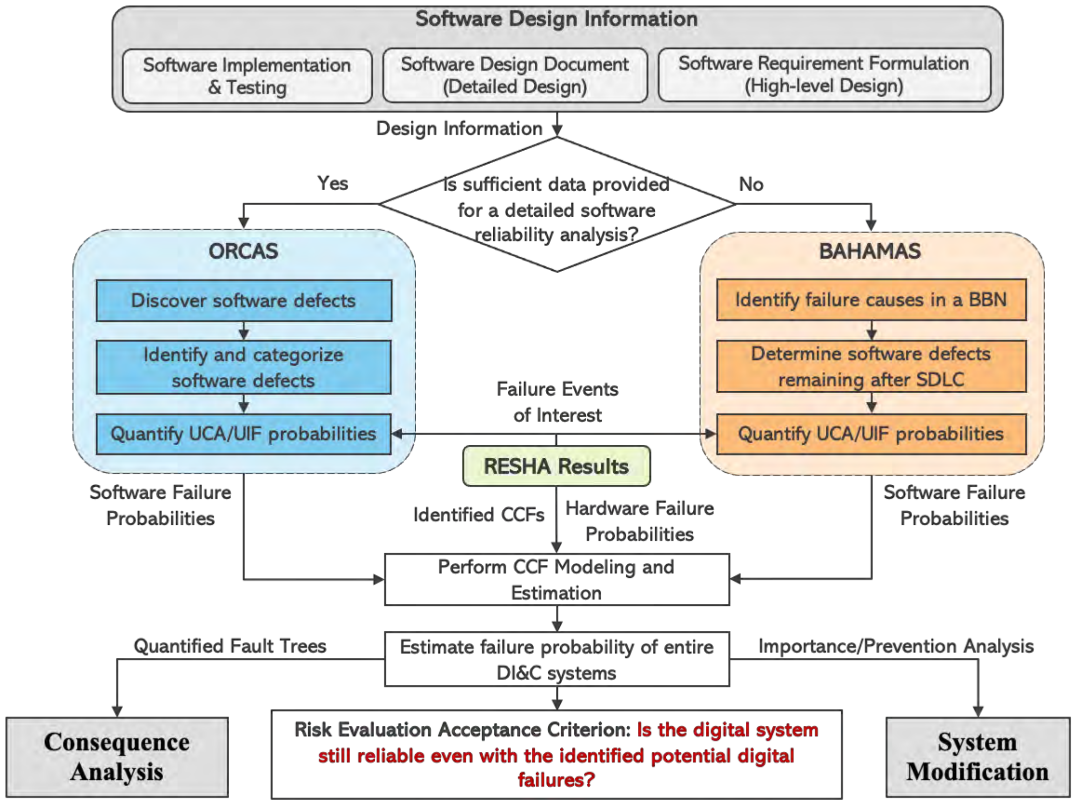
The goal of the multiscale quantitative reliability analysis is to estimate the DI&C system reliability values by evaluating the DI&C systems FT obtained in the hazard analysis and provide estimated reliability values as inputs for a subsequent consequence analysis. For the reliability analysis of large-scale DI&C systems, a small-scale analysis of software reliability in DI&C systems are also included in the reliability analysis workflow. Figure 4 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 life cycle, include but are not limited to the software requirement specifications, the software design description, and the software test documentation. These documents are necessary to determine whether design and test failure data are available to conduct detailed and highly relevant reliability analysis. 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, availability of design documentation to support reliability analyses may vary from rich to sparse. A solution is provided for each case in terms of the ORCAS or BAHAMAS methodology: BAHAMAS for data-limited conditions and ORCAS for data-rich analysis.

BAHAMAS was developed for 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 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 originating from the ODC approach. ODC is used to semantically (and systematically) classify the identified defects into disjoint software causality groups. A correlation table between causality groups and observable failures can thus be established linking root cause 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 reliability analysis of software, a modified beta-factor model (BFM) is applied for the modeling and estimation of software and hardware CCFs, the hardware failure probabilities are based on existing failure database or PRA models. Finally, when all the basic events of integrated FT are calculated, the failure probability of entire DI&C system can be estimated using FTA tools.


Software Design Information
Software Implementation & Testing
Software Design Document (Detailed Design)
Software Requirement Formulation (High-level Design)
Design Information
Is sufficient data provided for a detailed software reliability analysis?
Yes
No
ORCAS
Discover software defects
Identify and categorize software defects
Quantify UCA/UIF probabilities
BAHAMAS
Identify failure causes in a BBN
Determine software defects remaining after SDLC
Quantify UCA/UIF probabilities
Failure Events of Interest
RESHA Results
Software Failure Probabilities
Identified CCFs
Hardware Failure Probabilities
Software Failure Probabilities
Perform CCF Modeling and Estimation
Quantified Fault Trees
Estimate failure probability of entire DI&C systems
Importance/Prevention Analysis
Consequence Analysis
Risk Evaluation Acceptance Criterion: Is the digital system still reliable even with the identified potential digital failures?
System Modification

BAHAMAS	Bayesian and human-reliability-analysis-aided method for the reliability analysis of software	ORCAS	orthogonal-defect classification for assessing software reliability	SDLC	software development life cycle
		RESHA	redundancy-guided systems-theoretic method for hazard analysis	UCA	unsafe control action
CCF	common cause failure			UIF	unsafe information flow

Figure 4. The workflow of multiscale quantitative reliability analysis in the proposed framework.

Compared with the methodology described in [4] and [5], the workflow of multiscale quantitative reliability analysis keeps the same in the new version. The methodology of BAHAMAS, ORCAS, and CCF modeling approach have been refined and improved.

For BAHAMAS:

- BAHAMAS consists of two different types of root nodes—the review nodes, and the stage defect nodes. The use of these nodes has been clarified in the new version.
    - New equations were given to evaluate the SDLC stage defect nodes.

    - The evaluation of review nodes was clarified.

- Review quality plays an important role in the removal of software defects. For this work, review quality is considered a function trigger coverage and the number of reviews performed. Trigger coverage effectively considers the types of review activities that have been performed during the software development. While review number considers the impact of multiple independent reviews. Future work may incorporate other concepts. This work has introduced new equations to provide a clearer evaluation of the review number and trigger coverage metrics.

- A new workflow was provided to express guidance for the application of BAHAMAS under two pathways.
    - BAHAMAS application for the estimation software failure probability.
    - BAHAMAS application for estimation of software CCF probability including that of diverse system designs. New equations were developed to support diverse CCF modeling within the LWRS-developed framework. Given this development, BAHAMAS was considered for supporting the evaluation of diverse CCFs.

For ORCAS:

- Based on reviewer feedback, provided additional methodology description on how to conduct the analysis for meaningful results. Methodology for the requirements traceability matrix and trigger coverage assessment (part of the ORCAS process) were inadequately described.
- Retabulated the data related to the correlation table to fix the inconsistencies pointed out by industry reviewers. The inconsistencies were related to Excel algorithmic issues and have been resolved.
- Provided the description on how to conduct supplementary testing via automated combinatorial testing. The original implementation was unclear as to how the combinatorial testing should be conducted to achieve relevant dataset coverage. Guidance on boundary analysis is developed to constrain and improve test set construction for supplementary testing.

For CCF modeling approach:

- Refined the methodology of CCF modeling to allow consideration of diverse CCFs. A key concept is the idea that diverse software may have shared, common, or otherwise overlapping attributes. Some methods, such as BAHAMAS, may offer a means of directly estimating theoretical CCF vulnerability as measured by common aspects of the software requirements or other aspects of the software development life cycle. The new methodology is demonstrated in subsequent sections.
- Investigated the identification of sub-factors for improving the CCF modeling parameters employed by the CCF models of the LWRS-developed framework. One of the large challenges faced by the industry is the limited historical data for CCF especially in safety systems. Consequently, implicit modeling techniques that can incorporate qualitative and design related aspects, especially in the absence of operational data, are an important avenue for research. Industry collaboration and feedback led to the identification of several concepts that may provide pathways for more realistic software CCF modeling parameters. It was determined that defensive features such as built-in online software monitoring, alarms, watchdogs, and other such features may mitigate or eliminate some postulated CCFs.

More details and descriptions of the new version of ORCAS and BAHAMAS can be found respectively in Appendix B and C.

Methodology description and case study of the modified CCF modeling approach for diverse DI&C systems are provided in the next section.

2.3 An Approach to Modeling Postulated Software Common Cause Failures of Diverse Digital Instrumentation and Control Systems

Redundancy is a well-known strategy for maximizing reliability of safety-related systems; however, CCFs have the potential to defeat redundancy. As a means of protection, diversity may be employed to ensure the same function as a redundant system but by alternative technologies, methodologies, or techniques [16]. Despite its application for analog I&C systems, the effectiveness of diversity for DI&C systems remains a topic of concern. The use of digital technologies may increase the potential for CCF vulnerabilities due to failures in design specifications and system interactions [17]. Salako and Strigini provide a mathematical-based investigation of diversity and indicate scenarios where removing commonalities can be beneficial, damaging or have an uncertain effect [18]. Knight and Leveson have indicated that independently developed software may not necessarily fail independently [19]. Perfectly isolated development may be unlikely given the commonality of coding education and references [18]. Huang et al. indicate empirical studies, like [19], suggest programmers may be prone to common or similar errors [20]. Huang et al. use this to justify their research of the links between software diversity and human error diversity via personality traits, performance levels, cognitive styles, and, more recently, cognitive mechanisms [21]. This section proposes an approach to modeling CCF that is tailored for redundant architectures. Given the potential for common defects within multiversion software, the goal is to provide an approach for modeling potential CCFs of diverse software configurations in DI&C systems.

2.3.1 Technical Background

A CCF is the occurrence of two or more failure events due to the simultaneous occurrence of a shared failure cause and a coupling mechanism [22]. In our previous work, we introduced an approach for modeling CCFs within DI&C systems [4]. That work assumed purely redundant configurations of components. Here we provide a brief overview of the CCF methodology and the associated challenge of modeling potential CCFs of diverse components.

The LWRS-developed framework consists of an approach for CCF modeling that incorporates the modified beta factor (MBF) method [23, 24]. Our approach emphasizes the identification of software-centric coupling mechanisms necessary for simultaneous failures of redundant software components. For the purposes of analysis any group of components that share similarities via coupling mechanisms may have a vulnerability to CCF and can be considered a CCCG. One of the key aspects of the MBF method is how it accounts for asymmetry in coupling mechanisms that are used when defining CCCGs [24]. Additionally, while most models rely on operational data as the basis for estimating CCF parameters to model CCFs, ours does not. Instead, model parameters are generated by qualitative details found in the software's own development and design plans. This provides a means to overcome certain modeling challenges including limited data [24].

The general process of CCF modeling within the LWRS-developed framework requires analysts to (1) identify the CCCGs, (2) define model parameters for each CCCG, and (3) evaluate the CCFs for each CCCG. Our approach is based on the MBF method for CCF; consequently, CCF is given as a portion of total failure probability, which must be evaluated separately. Identification of the CCCGs is based on software-centric coupling mechanisms (e.g., shared software code, shared requirements, and languages). When software components have been grouped into N groups by common attributes, the next step is to define model parameters for each CCCG (i.e., $CCCG_n$). Each CCCG is assigned a beta factor (β_n) that represents the contribution of that CCCG to the total failure probability (Q_T). Total failure probability is represented as the summation of independent (Q_I) and common failure (Q_{CCF}) probability. Q_{CCF} is separated into contributions dependent upon the number of CCCGs. The equations, as employed within our previous work [24], are shown below:

$$Q_{CCF} = P(CCCG_1) + P(CCCG_2) + \cdots P(CCCG_N) \tag{1}$$

$$P(CCCG_n) = (\beta_n)Q_t \tag{2}$$

$$Q_I = (1 - \beta_t)Q_t = \left[1 - \sum_1^N (\beta_n)\right] Q_t \tag{3}$$

The MBF method is intended for use with redundant configurations. The introduction of diversity creates modeling challenges. Diversity leads to complications with traditional parametric-based modeling. Consider the redundant configuration shown in Figure 5. Two components, A and B, have been determined to share a CCCG and are susceptible to a CCF. In this scenario, component A is more reliable than component B (perhaps due to the difference in manufacturers); the failure probability for A (Q_A) is lower than the Q_B. This difference in reliability presents a challenge for CCF modeling. The failure probabilities for A and B are given by Equations (4) and (5).

$$Q_A = Q_{A_I} + Q_{CCF_{AB}} \tag{4}$$

$$Q_B = Q_{B_I} + Q_{CCF_{AB}} \tag{5}$$

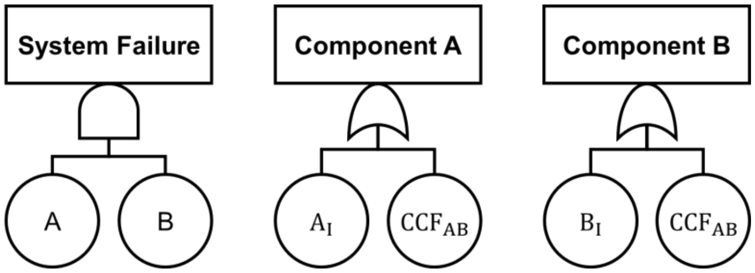


Figure 5. Simple fault tree example.

Traditional CCF modeling assumes $Q_A = Q_B = Q_T$, also known as a symmetry assumption. The resulting CCF for A&B (i.e., $Q_{CCF_{AB}}$) is given by Equation (2). A complication arises for the diverse case $Q_A \neq Q_B$; it becomes necessary to select an appropriate Q_T. Consider the scenario when $Q_A < Q_B$, the value of β is fixed for the CCCG of A&B, and performance data are unavailable to directly measure $Q_{CCF_{AB}}$. The following are options for selecting Q_T [25]:

1. The larger of the values. This is considered a conservative option by Brand [25]. It will predict a larger CCF value than for Option 2. However, it should be easily recognized that there is a potential logical inconsistency especially as β approaches 1.
2. The smaller of the values. This is supported by the Frechet-Hoeffding upper bound on joint probability distributions [26]. It considers the extreme case in which the entirety of failure events that lead to A fit within the events that lead to B (i.e., $Q_A < Q_B$) corresponding to $\beta = 1$; for such case, it should not be possible for $Q_{CCF_{AB}}$ to exceed Q_A.
3. The geometric mean. The geometric mean is presented as an option in by Brand [25]. Huage et al. indicate that this approach is preferred and adequate for rates or probabilities that are of the same magnitude [27].

Use of any option requires consideration of its impact. For example, Option 1 may lead to logical inconsistencies that are bounded by the Frechet-Hoeffding upper bound on joint distributions. Despite being considered as a preferred approach, Option 3 should be checked against Option 2. In all cases, there is still a complication for selecting the correct and meaningful representative parameter (e.g., beta) for the CCCG. Given these challenges, it is proposed to find an alternative approach that better represents the relationship between the components of the CCCG. The next section discusses our approach for CCF modeling of diverse software configurations.

2.3.2 Methodology

2.3.2.1 Overview

Rather than rely on Options 1–3 given in the previous section, a direct estimation is proposed to identify a group's potential CCF. Options 1–3 provide ways of combining or approximating total failures without the consideration of the common aspects that are shared between the CCCG. A new approach introduces a variable to directly represent the similarity within the CCCG regardless of any asymmetry in their total failure probabilities. Our previous work relied on Equations (1) and (2) to define failure probability of a component due to common causes; we are now proposing to rely on the alternative shown below:

$$Q_{CCF_n} = f(Q_{CC_n}) \tag{6}$$

Q_{CC_n}represents the theoretical CCF probability for the CCCG based on the similarity that is shared between the components of the CCCG (e.g., identical software requirements). Defenses may exist that prevent the CCFs from dominating the failure space for the system and otherwise reduce the CCF of the CCCG. To account for defenses, ϕ_n is incorporated with Equation (6) producing Equation (7).

$$Q_{CCF_n} = \phi_n Q_{CC_n} \tag{7}$$

Note that ϕ_n plays role similar β_n in Equation (2). The difference being that β_n fundamentally represents a fractional percentage of CCFs over the total failures, while ϕ_n represents the defense that a CCCG has against CCF. When $\phi_n = 1$, the level of defense is poor, and Q_{CCF_n} is equivalent to the theoretical failure probability Q_{CC_n}. The variable Q_{CC_n} considers the designed commonality of the components of the CCCG, while the defense factor, ϕ_n, considers the qualitative and designed features that act as defensive measures to a CCF of the CCCG. Implementing Q_{CC_n} to the model necessitates slightly different equations than those given previously. The resulting equations are shown below.

$$Q_{CCF} = Q_{CCF_1} + Q_{CCF_2} + \cdots Q_{CCF_N} \tag{8}$$

$$Q_{CCF_n} = P(CCCG_n) = \phi_n Q_{CC_n} \tag{9}$$

$$Q_I = Q_T - Q_{CCF} \tag{10}$$

$$Q_T = \sum_{1}^{N} \phi_n Q_{CC_n} + Q_I \tag{11}$$

The challenge for this new approach is defining ϕ_n andQ_{CC_n}. In contrast to the traditional method of parameter estimation (i.e., reliance on historical data), ϕ_n will be solved with qualitative information. The process is defining model parameters based on defensive qualitative attributes and is discussed in [5]. Software failure consists of two parts, the existence of a defect and an activation scenario [17]. Q_{CC_n} defines the common or shared aspects of failure between the CCCG. Thus, a method is needed that can define the shared defects or activation scenarios that make up software failure. In the next section, we propose a method to define Q_{CC_n} specifically in terms of shared defects.

2.3.2.2 A Method to Define Q_{CC}

BAHAMAS was developed for the conditions when testing or operational data are limited, such as for reliability estimations of software in early development stages [10]. BAHAMAS was created by considering the principle of mechanisms of software failure. It has been said that software does not fail, at least in the same way that hardware can [28] [29]. Instead, software performs exactly how it has been designed to perform; any unwanted action or behavior (i.e., a failure) is due to a fault within the code that has been activated based on certain inputs or operational conditions. Software failure occurs by the activation of latent defects (e.g., deficiencies from coding errors, installation errors, maintenance errors, setpoint changes, and requirements errors) which are known to be caused by human errors in the SDLC [17]. BAHAMAS relies on this understanding to assess software reliability. Instead of relying on testing data, BAHAMAS employs a BBN to map the causes of software failures to specific defect types, defined by ODC [30], that can be traced to human errors in the SDLC. In turn, the human errors can be modeled with HRA [10]. By combining details of a specific SDLC, the BBN can provide an estimation of the existence of defects within that software. These in turn are used to find the software failure probability. The general format of the BBN is shown in Figure 6.

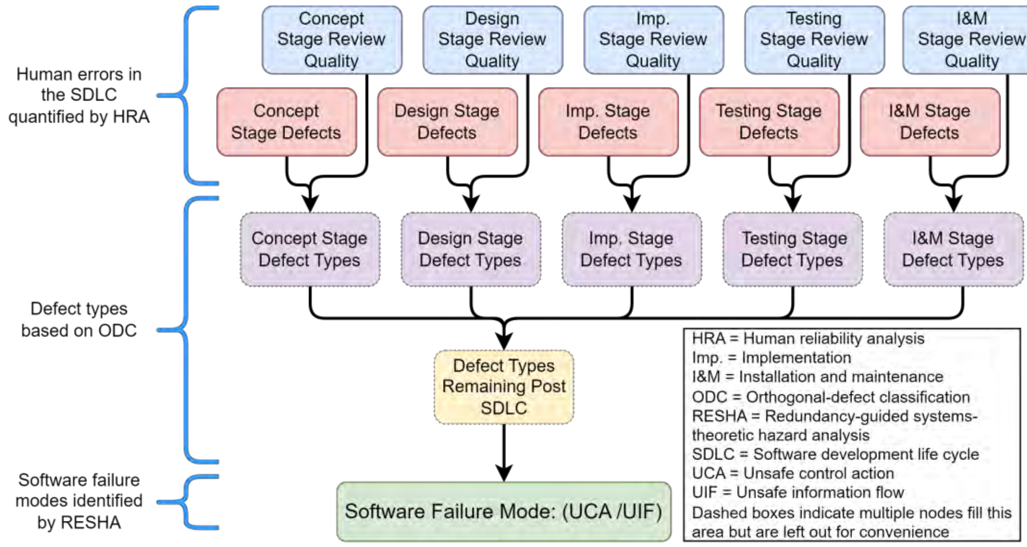

Human errors in the SDLC quantified by HRA
Concept Stage Review Quality
Design Stage Review Quality
Imp. Stage Review Quality
Testing Stage Review Quality
I&M Stage Review Quality
Concept Stage Defects
Design Stage Defects
Imp. Stage Defects
Testing Stage Defects
I&M Stage Defects
Defect types based on ODC
Concept Stage Defect Types
Design Stage Defect Types
Imp. Stage Defect Types
Testing Stage Defect Types
I&M Stage Defect Types
Defect Types Remaining Post SDLC
Software failure modes identified by RESHA
Software Failure Mode: (UCA /UIF)
HRA = Human reliability analysis
Imp. = Implementation
I&M = Installation and maintenance
ODC = Orthogonal-defect classification
RESHA = Redundancy-guided systems-theoretic hazard analysis
SDLC = Software development life cycle
UCA = Unsafe control action
UIF = Unsafe information flow
Dashed boxes indicate multiple nodes fill this area but are left out for convenience

Figure 6. General structure of the BBN used within BAHAMAS.

In our previous work, we investigated CCF of redundant components that share the same software [4]. For a CCF, there must be (1) a failure involving multiple components and (2) a common cause that is made “shareable” by the existence of some coupling mechanism. The shared cause for software must include a common “active” defect or fault. A CCF can occur when multiple components share copies of the same software and can be influenced by the same activation scenario. The only difference between the CCF and individual failure for the identical components is that a shared defect is activated in a single software vs. in the multiple redundant software components. BAHAMAS provides the indication of total failure probability for a software by estimating the existence of defects. Meanwhile, it is the system configuration that determines whether a CCF or independent failure can occur. In other words, if there is no redundant software, then there is no potential for CCF, and the BAHAMAS estimation failure represents independent failure only. Because BAHAMAS provides an indication of the remaining defects within a software, when that software is used redundantly, BAHAMAS provides a direct indication of the common defects that exist between redundant software components. As an example, consider CCCG of two redundant components that share identical software. The estimate of software failure probability provided by BAHAMAS is equivalent for both components:

$$Q_{Sw1} = Q_{A_{total}} = Q_{B_{total}} \tag{12}$$

By tracking the shared aspects of the SDLC, which happens to be 100% identical for A and B, BAHAMAS has provided an indication of Q_{CC}. Barring any exterior defenses against CCF, the total failure value predicted by BAHAMAS is equivalent to the theoretical CCF of the CCCG.

$$Q_{Sw1} = Q_{CC} \tag{13}$$

BAHAMAS provides an implied representation of the common defects that exist for CCCG, where these defects can be traced to human errors during the SDLC. It is hypothesized, that diverse software may share defects due to common human errors during their respective SDLC activities, resulting in a set of defects that are common and can lead to common failure of diverse software. Software defects found within Software A are given as a function of the SDLC of A, while the software defects found within Software B are given as a function for the SDLC of B. The defects that are shared within Software A and Software B are due to common human errors within both SDLC-A and SDLC-B. Applying BAHAMAS for the common aspects between SDLC-A and SDLC-B provides a direct indication of the Q_{CC} term.

This work emphasizes the necessary aspects for structuring BAHAMAS to find Q_{CC} term. Additional details of the BAHAMAS can be found in [10]. BAHAMAS works by identifying the details of the SDLC to construct a BBN like the one in Figure 6, the root nodes are quantified, and then the BBN is solved for the failure of interest. There are two types of root nodes—the nodes for stage review quality and the nodes for stage defects. The defect nodes are a function of human error probability (HEP) for each critical activity of the SDLC stage. The review quality nodes are a function of the average number of reviews performed and the trigger coverage for each of the critical activities evaluated. For practical applications, the output from BAHAMAS can be considered a function of HEP, review number, and trigger coverage. The subsequent paragraphs detail how these should be determined for the evaluation of Q_{CC}.

The process for defining defect nodes requires that critical activities of the SDLC stage be evaluated for human errors. Each SDLC stage has multiple activities consisting of multiple tasks. Each of these activities may introduce errors to the software. HRA is applied to determine the HEP of each critical task (T), the union of which provides an indication of the probability of defects for each stage; see Equation (14) below. To find Q_{CC} term for the diverse CCCG, only the common tasks should be assessed; thus, only a subset of T will be evaluated (i.e., T_{cc}). Additionally, the HEP from Equation (14) should be replaced by HEP_{min}, corresponding to the smaller evaluated HEP of the task shared by the diverse CCCG. The result of this change is shown in Equation (15). As an example, given a task that is shared by software-1 and software-2, the maximum overlap of the HEPs will be given by the smaller of the two. This follows the discussion in [26] concerning the upper bound on joint probability distributions.

$$P(Stage\ Defects) = \sum_{i=1}^{T} (HEP)_i \tag{14}$$

$$P_{cc}(Stage\ Defects) = \sum_{i=1}^{T_{cc}} (HEP_{min})_i \tag{15}$$

The SDLC stage review quality nodes shown in Figure 6. These nodes are evaluated based on the average number of reviews and trigger coverage that occurs during independent review of the SDLC activities. Triggers are introduced and discussed in [4]. The trigger coverage (TC) is determined from the average of each task-level trigger coverage (tc) which is the percent of relevant triggers that have been covered for a task of a particular SDLC stage; see Equation (16) below. Trigger coverage for the diverse CCCG (TC_{cc}) is given by Equation (17) and depends on the subset of tasks that are shared by the CCCG.

$$TC = \frac{1}{T}\sum_{i=1}^{T}(tc)_i \tag{16}$$

$$TC_{cc} = \frac{1}{T}\sum_{i=1}^{T_{cc}}(tc_{cc})_i \tag{17}$$

Within Equation (17), the task-level trigger coverage is defined as a function of the set of triggers investigated ($\{tr\}_{i,inv}$) by the CCCG out of the total relevant set of triggers ($\{tr\}_{i,tot}$) that are applicable to a particular task. Equation (18) provides the relationship between $\{tr_{cc}\}_i$ and $\{tr\}_{i.s}$, where $\{tr_{cc}\}_i$ is the set of relevant triggers to the CCCG, and $\{tr\}_{i,s}$ is the set covered by the SDLC of software *s*. The index, K, is the total number of diverse software implementations used by the CCCG. Equation (19) gives the task-level trigger coverage for the diverse CCCG.

$$\{tr_{cc}\}_i = \{\{tr\}_{i,1} \cup \{tr\}_{i,2} \ldots \cup \{tr\}_{i,K}\} \tag{18}$$

$$(tc_{cc})_i = \frac{size\ of\ \{tr_{cc}\}_{i,inv}}{size\ of\ \{tr_{cc}\}_{i,tot}} \tag{19}$$

The average number of reviews for a particular stage is defined by Equation (20), where *r* is the number of independent reviews performed for a specific task. For the diverse CCCGs, only the common or shared tasks should be assessed; thus, a subset of *T* is employed to determine *R*. The result is Equation (21), where r_{cc} represents a weighted average number of reviews for the CCCG. Equation (22) is given to show how r_{cc} is defined for each task where $r_{i,j,s}$ is the number of reviews performed during the SDLC of software *s,* for task *i*, that involved trigger *j* of the $\{tr\}_{i,s}$. Note J_i corresponds to the size of $\{tr\}_{i,s}$.

$$R = \frac{1}{T}\sum_{i=1}^{T} r_i \tag{20}$$

$$R_{cc} = \frac{1}{T}\sum_{i=1}^{T_{cc}}(r_{cc})_i \tag{21}$$

$$(r_{cc})_i = \frac{1}{size\ of\ \{tr\}_{i,tot}}\sum_{j=1}^{J_i} {}_{s=1}^{K}max(r_{i,j,s}) \tag{22}$$

2.3.3 Case Study

This section details a simplified case study. The case study provides an example of our approach for modeling CCFs for systems containing diverse software. The case study is applied to the system shown in Figure 7. The figure has three redundant divisions, each with two redundant processors. Each processor must perform a single critical function. The system requires at least one successful critical function from 2 out of 3 (2oo3) divisions for successful operation. The developers chose two diverse software versions to prevent a CCF of all six processors and ensure system reliability.

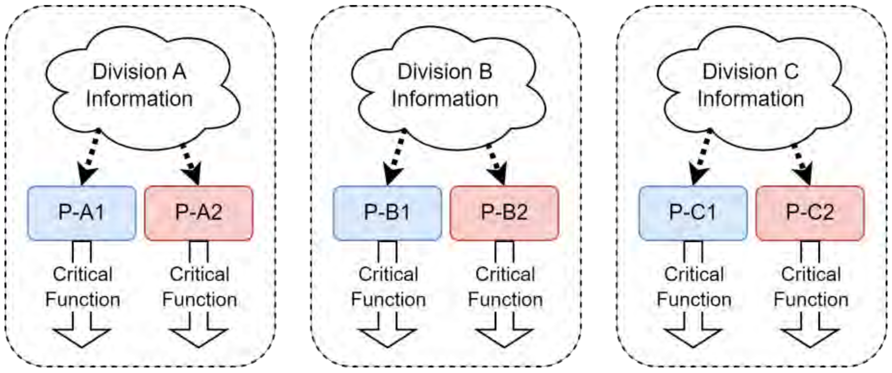


Figure 7. Case study control system.

The following are assumptions for the case study:

- The blue (software-1) and red (software-2) processors in Figure 7 are assumed to be diverse
- Software-1 is developed and maintained by a separate manufacturer and team than software-2
- There is no hardware diversity
- The SDLC for both diverse software implementations contain overlapped activities and tasks
- The HEPs necessary to define SDLC stage defects have been assumed
- All stages found within a particular SDLC have the same trigger coverage and review number
- The values for CCF model subfactors have been assumed based on expert judgment.

The process for modeling CCF consists of three steps: (1) identify the CCCGs, (2) define model parameters for each CCCG, and (3) evaluate the CCFs for each CCCG. Step 3 requires an evaluation of Q_{cc} which will be determined by BAHAMAS. For Step 1, the failure of interest is defined as P-A1 fails to provide critical function when it is needed during operation. It is assumed to be a failure-on-demand scenario. Step 2 requires that the CCCGs be identified and that model parameters for those CCCGs be defined. The process for identifying software-specific CCCGs and assigning model parameters was introduced in [4] and [24], though some modification is still needed for quantifying the parameters of diverse configurations. The CCCGs are defined by the coupling mechanisms shown in Table 1, where the score for each CCCG defense factor (i.e., model parameter) is also shown. In Step 3, the evaluation of the CCFs requires that Q_T or Q_{cc_n} be defined for each CCCG. For this case study, Q_{cc_n} is the combination of software-1 and software-2, found in CCCGs 1–3 and 6 from Table 1.

Table 1. CCCGs, coupling mechanisms, and CCF model parameters.

CCCGs		Coupling Mechanisms	ϕ_n
1	BPs in Division A	Division A Specific Inputs	0.639
2	BPs in Division B	Division B Specific Inputs	0.639
3	BPs in Division C	Division C Specific Inputs	0.639
4	SW1 BPs	Software-1 SDLC, Software-1 Maintenance Team	0.483
5	SW2 BPs	Software-2 SDLC, Software-2 Maintenance Team	0.483
6	All BPs	Software Requirements, Function, partially shared SDLC practices	0.605

The evaluation of Q_{cc_n} is found by employing BAHAMAS. Software-1 has 76.4% trigger coverage and an average of three reviews per task. Software 2 has 84% trigger coverage and an average of 2.5 reviews per task. Equations from previous sections were used to define the common aspects necessary to evaluate BAHAMAS for Q_{cc_n}. The common or shared SDLC has 93.8% trigger coverage and an average of 2.675 reviews per task. Q_T, and Q_{cc_n} are shown in Table 2 along with the probability of SDLC stage defects.

Table 2. Probabilities for SDLC stage defects, $\boldsymbol{Q_T}$, and $\boldsymbol{Q_{cc}}$ for each software.

Stages	Software 1	Software 2	Common
Concept	0.13	0.15	0.09
Design	0.25	0.28	0.18
Implementation	0.33	0.36	0.26
Testing	0.27	0.25	0.19
I&M	0.11	0.12	0.07
UCA-A	Q_T = 3.556E-5	Q_T = 8.137E-5	Q_{cc} = 1.077E-5

The defense factors and the values for Q_{cc_n} and Q_T are used to define the software CCF values for each of the CCCGs identified in Table 2. Specifically, Equations (8–12) are used to find the desired CCF and independent failure values. These can then be used as part of a PRA. The results for software component P-A1 are shown in Table 3. For this assessment, an FTA was conducted separately and indicated a top event probability of 6.517E-6. This result is dominated by the CCF of CCCG-6 which represents a CCF of all processors shown in the system. Additionally, the developers' choice to use diverse software did not eliminate the threat of CCFs to system reliability. The case study demonstrates how, due to shared aspects of the SDLC, potential CCFs of diverse software can be identified and quantified.

Table 3. Result table for software processor A1.

CCCGs		CCF	Q_{cc_n} or Q_T or Q_I	ϕ_n
1	BPs in Division A	6.882E-6	1.077E-5	0.639
4	SW1 BPs	1.718E-5	3.556E-5	0.483
6	All BPs	6.516E-6	1.077E-5	0.605
N/A	P-A1 Ind.	N/A	4.987E-6	N/A

2.3.4 Discussion

This section introduces a modification to the CCF approach employed by the LWRS-developed framework to increase its capabilities for modeling CCFs of diverse DI&C systems. The CCF model was adjusted by the introduction of Q_{cc}, and the defense factor, ϕ. Q_{cc}, represents the common parts of diverse software and is therefore used to represent the theoretical commonality and that leads to CCF, while ϕ represents the how well defended a CCCG is against CCF. This Q_{cc} term provides a direct model of the commonality between functionally redundant software elements, even diverse elements. This work proposed BAHAMAS to evaluate Q_{cc} given BAHAMAS has the capability to directly consider commonality of components. A simple case study was shown to demonstrate the approach to modeling potential CCFs of diverse components. Future work will include methodology development and improvement in addition to demonstrations on more detailed and complex analyses.

3. PRELIMINARY STUDY OF CROSS-SYSTEM COMMON CAUSE FAILURES

This section evaluates the feasibility of current methods on cross-system CCF analysis between a representative four-division digital RTS and ESFAS. Section 3.1 introduces the background and motivation of this research effort. Section 3.2 provides more details about the target systems in case studies. Section 3.3 introduces the application of BAHAMAS and CCF modeling approach to support the risk analysis of systems described in Section 3.2. Section 3 ends with a discussion in Section 3.4.

3.1 Motivation

DI&C system can integrate discrete and separate systems into a single system. The integrated system may share resources including communication channels, power supplies, as well as multifunction displays and controls. It allows for the implementation of new designs using a limited number of components, cost reduction, and upgrading of existing analog systems into DI&C system. However, highly integrated DI&C systems can increase vulnerabilities related to CCF, leading to increased costs and design complexity. Lungmen in Taiwan combines the functions of reactor shutdown and isolating the reactor system within the reactor protection system (RPS) [31]. Similarly, at plants such as Sizewell in United Kingdom, Ulchin in South Korea, and Dukovany in Czech Republic, the plant protection system (PPS) or digital RPS manages both of reactor trip functions and engineered safety feature (ESF) [31]. Although the subsystems are designed with functional diversity, and independent and diverse systems, such as the diverse protection system, are provided to implement defense-in-depth, the analysts should identify potential CCFs.

Diversity serves as a tool to prevent CCFs by providing various ways to accomplish intended function. There is design diversity, functional diversity, equipment diversity (equipment manufacturer/logic processing equipment diversity), software diversity (logic diversity), human diversity (life cycle diversity), and signal diversity [31]. Among these, software diversity evaluates diversity based on different algorithms, logic, program architecture, timing and/or order of execution, operating systems, and computer languages. ORNL/TM-2013/563 evaluates diversity based on diverse inputs for data diversity, separate developments, diverse development teams, diverse requirements or specifications for design diversity, and functional diversity [32].

In digital systems, the task of comprehensively testing software proves to be elusive, while accurately predicting the failure modes of a software is also challenging. Additionally, the assumption of independent failure for independently produced digital components warrants further discussion. Even with different manufacturers, programmers, programming languages, and algorithms, there may be CCFs due to similarities in application nature and difficulties experienced by individual programmers [33]. For instance, in 1990, despite being developed by different companies using different programming languages, mathematical programs (MACSYMA, Maple, Mathematica, and Reduce) produced the same incorrect result [33]. Moreover, two programs that may seem distinct can still perform the same function. For this reason, verifying diversity among algorithms of software is generally impossible. Consequently, DI&C systems are deemed susceptible to CCF in cases of identical system designs and identical copies of software in redundant divisions or integrated and interconnected systems. Previous studies have considered coupling factors for software, including common external operational conditions (e.g., shared inputs), shared software development life cycles, shared human interactions (e.g., shared users and installation/maintenance teams), and common requirements [5].

In this section, a comprehensive evaluation of cross-system CCF is conducted by examining various coupling factors. These coupling factors include software requirements (such as SDLC, software language, and software algorithms), human interactions (involving maintenance teams and plans), input similarity, and potential functional similarities. It is noteworthy that similar requirements, even with equivalent programmers, programming languages, and algorithms, might result in varying functions being performed. A case is the utilization of local coincidence logic (LCL) in both RTS and ESFAS. While their logical processes might be similar to certain specifications, differences arise in terms of setpoints and objectives for functions such as reactor trip actuation and ESF signal actuation. Consequently, such distinctions are classified as separate functional requirements, despite similarities in design diversity.

3.2 System Description

In this study, RTS initiates a reactor scram automatically or manually to rapidly reduce reactivity. ESF provides mitigation systems such as a core cooling system and containment system to prevent and mitigate accidents. Each system consists of four divisions, with two bistable processors (BPs) and two LCL racks per division. Within each LCL rack, there are four LCL processors. Signals from independent sensors in each division are transmitted to a BP, while trip signals produced by BPs are conveyed to LCLs. The LCL processors perform two-out-of-four voting on trip signals received from multiple redundant divisions. The RTS and ESFAS share the same BPs, and two of the four LCL processors generate reactor trip signals for selective relays, while the remaining two LCL processors produce ESF actuation signals, transmitted to the engineered safety features-component control system (ESF-CCS).

This study further analyzes cases involving LCL processors that share inputs and utilize similar coincidence logic since the prior study [5] has encompassed a case study involving BP. The baseline configuration employs one common software for each LCL processor. While there might be some differences in certain functionalities between RTS and ESFAS, the baseline configuration assumes that LCL processors perform the same function of two-out-of-four voting logic in both RTS and ESFAS. Figure 8 and Figure 9 represent the baseline configuration for RTS and ESFAS.

The diverse configuration is further analyzed to assess the impact of diversity on the extent of CCF across the system. In this configuration, two distinct software applications are utilized within the LCL processors. Software-1 is implemented in Division A&C, while Software-2 is used in Division B&D. The diverse configuration for RTS and ESFAS is illustrated in Figure 10 and Figure 11.

A sensitivity analysis is conducted using the second baseline configuration. Within this configuration, a functional requirement is isolated from design diversity and treated as one of the coupling factors. Despite the significant resemblance between LCL processors developed by the same programmer, utilizing common programming language and similar SDLC, distinctions in logic trip setpoints or the types of the trip signal, whether it triggers a reactor trip or an ESF actuation signal, are assumed to differentiate the functional requirement.

In an extension of the analysis, the second diverse configuration is explored, which presumed that the LCL processors in RTS and ESFAS were equipped with entirely separate software applications from the design phase. This specific configuration is visually illustrated in Figure 12 and Figure 13.

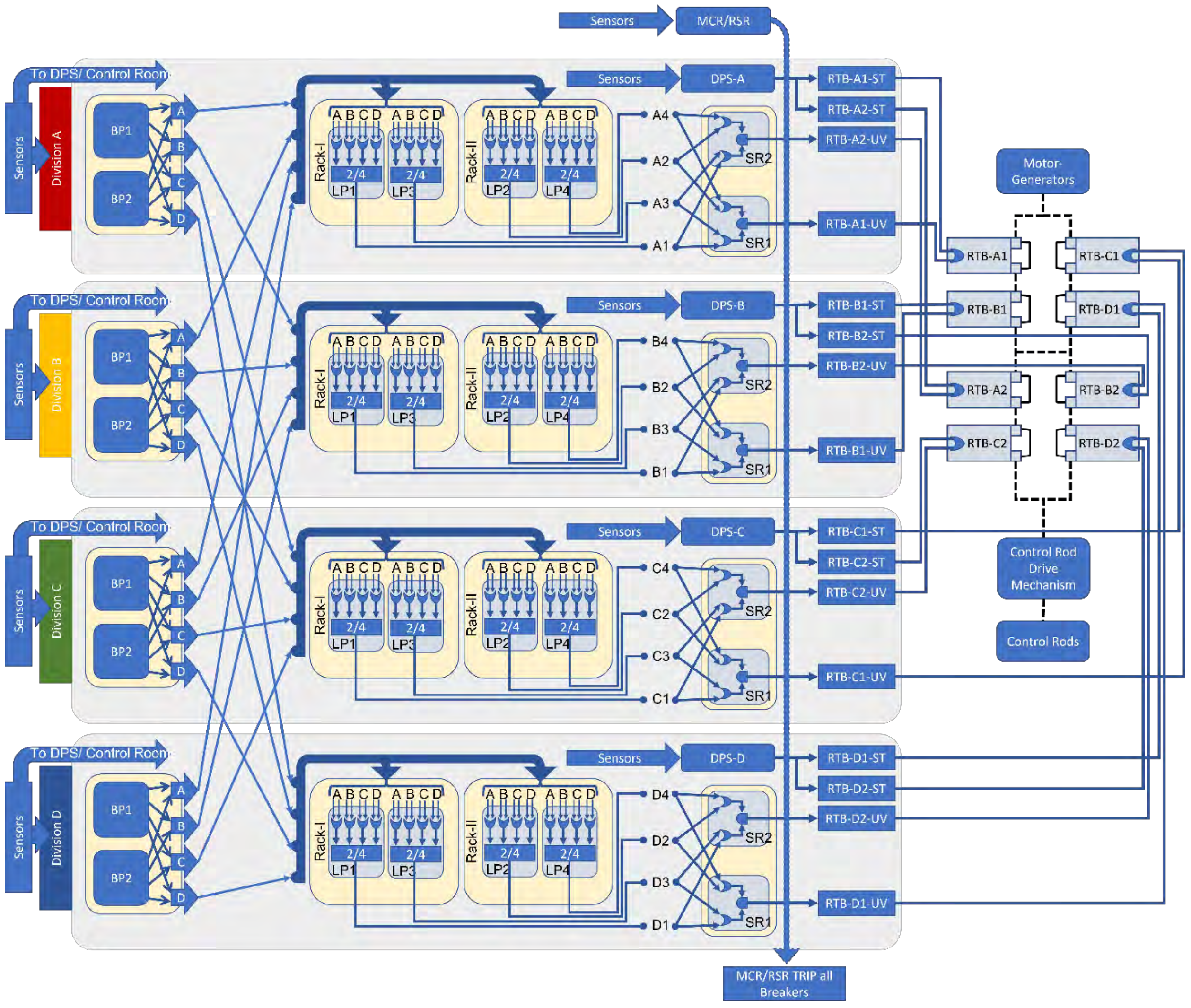


Figure 8. Baseline four-division digital RTS (LCL processors have identical software).

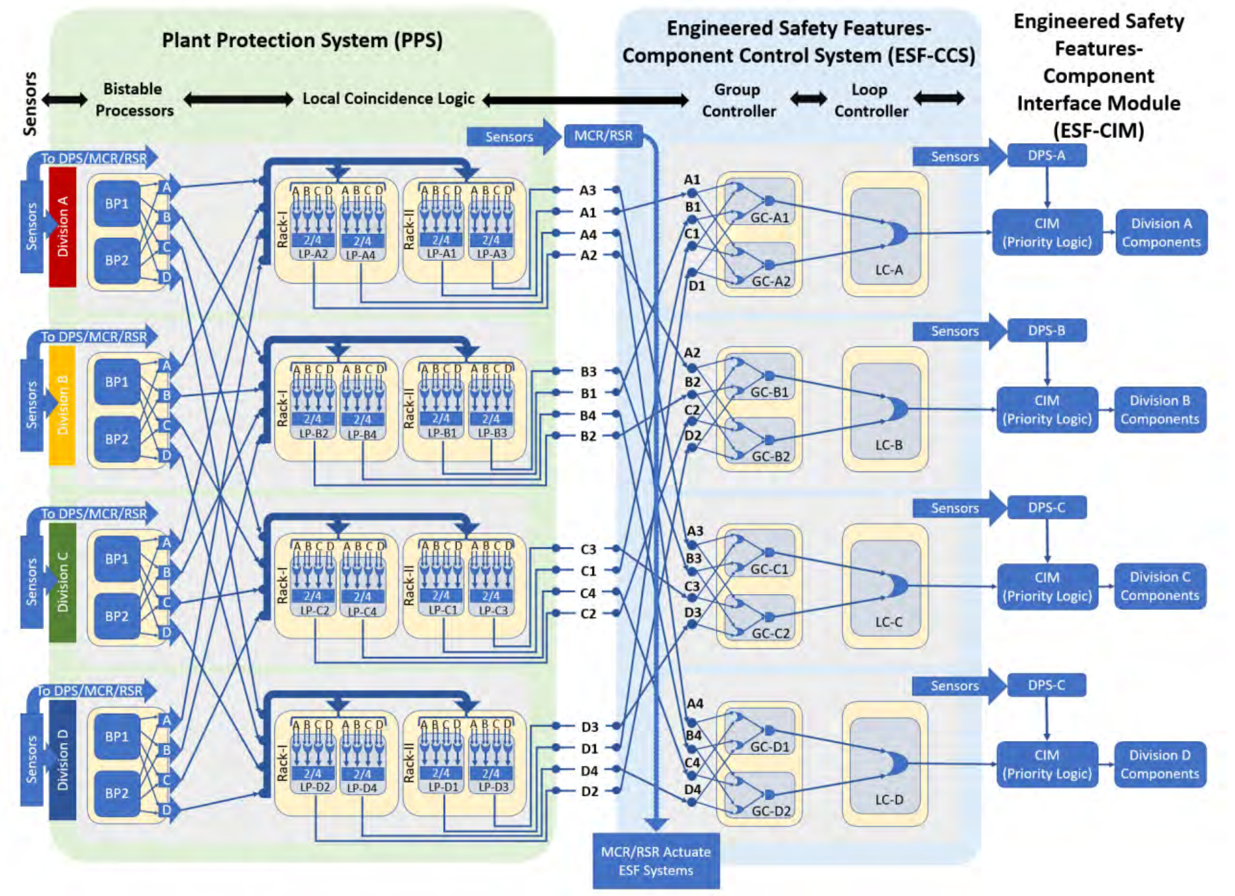


Figure 9. Baseline four-division digital ESFAS (LCL processors have identical software).

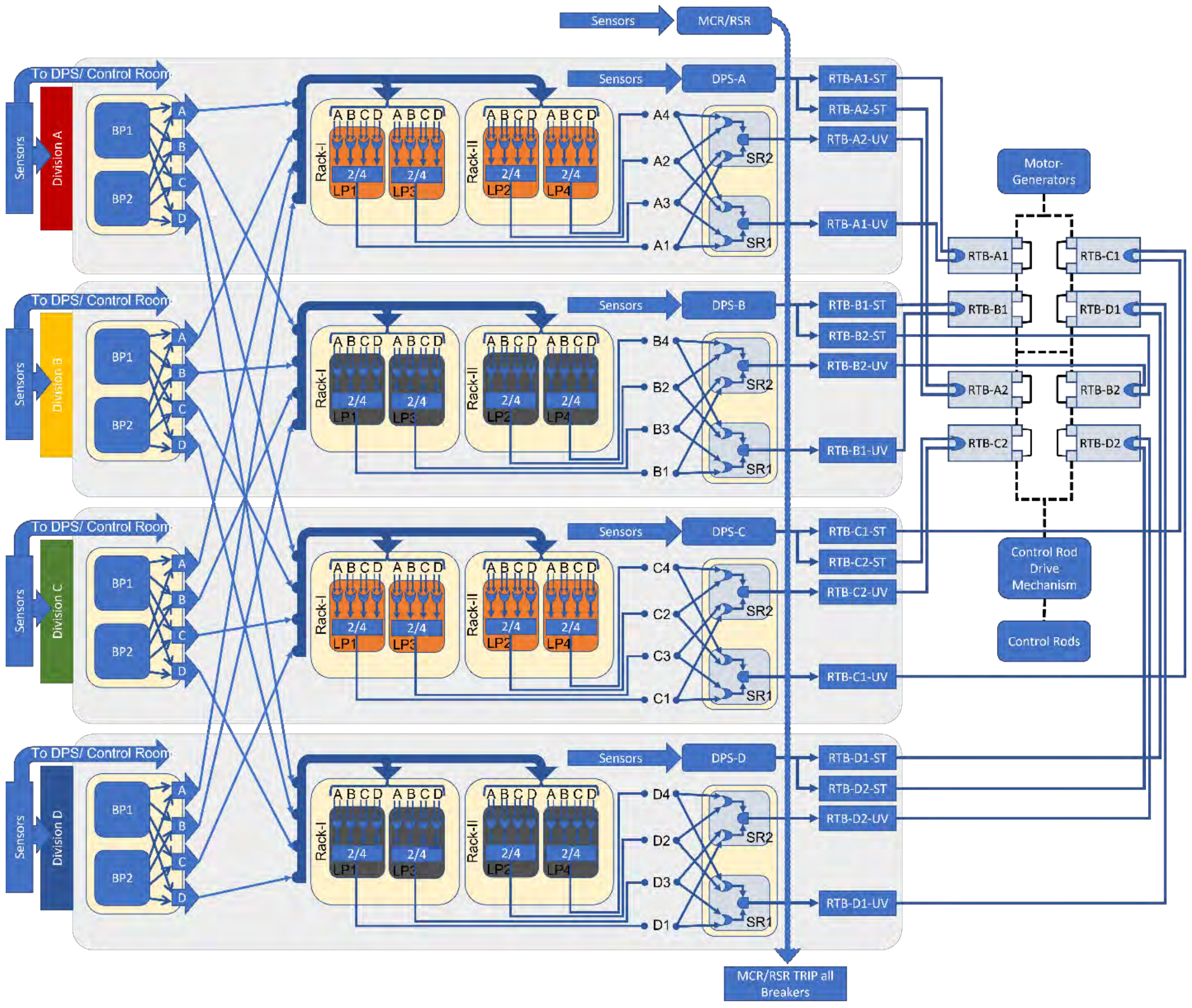


Figure 10. Diverse configuration four-division digital RTS (Orange components show Software-1 in Division A&C. Black components show Software-2 in Division B&D).

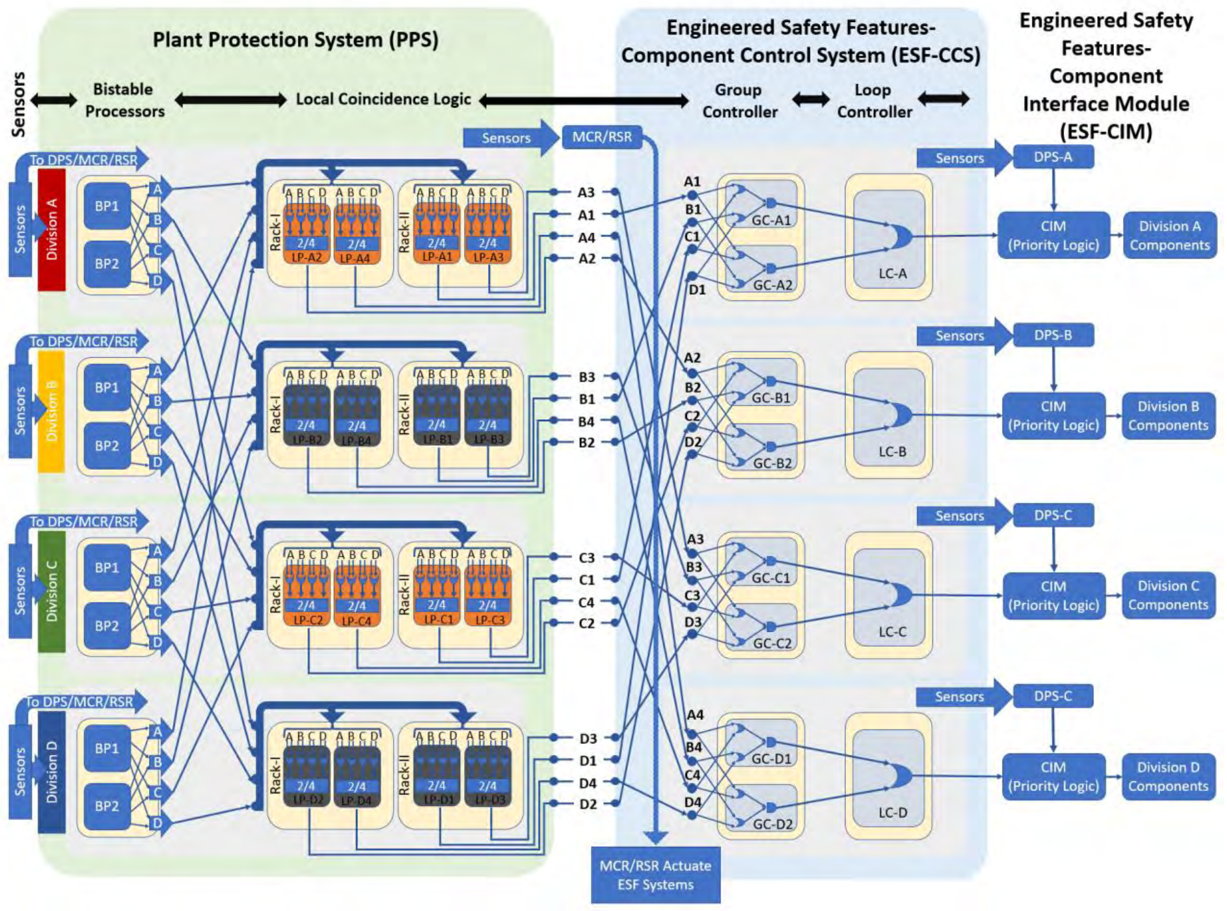


Figure 11. Diverse configuration four-division digital ESFAS (Orange components show Software-1 in Division A&C. Black components show Software-2 in Division B&D).

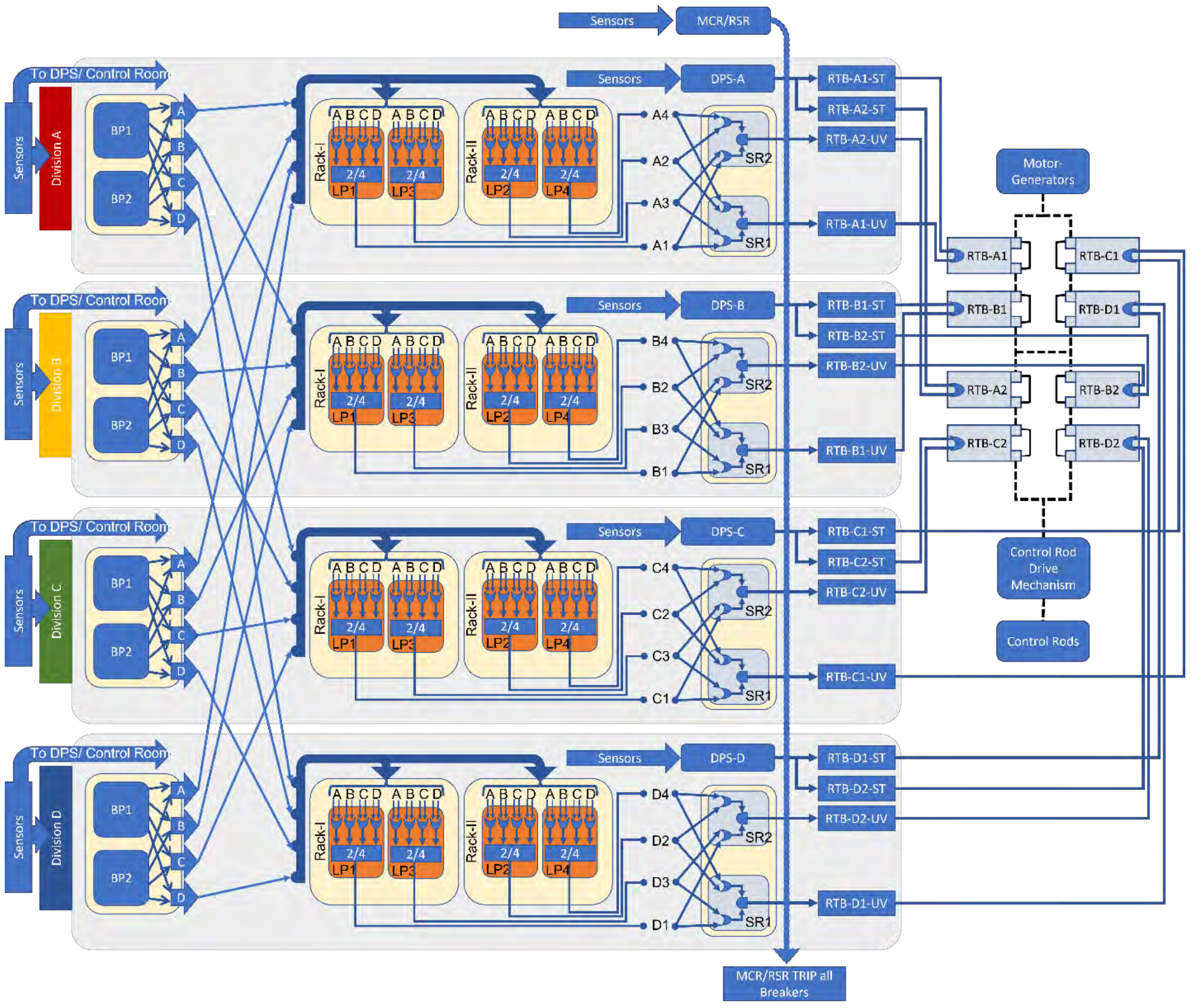


Figure 12. The second diverse configuration four-division digital RTS (Orange components show Software-1 in RTS).

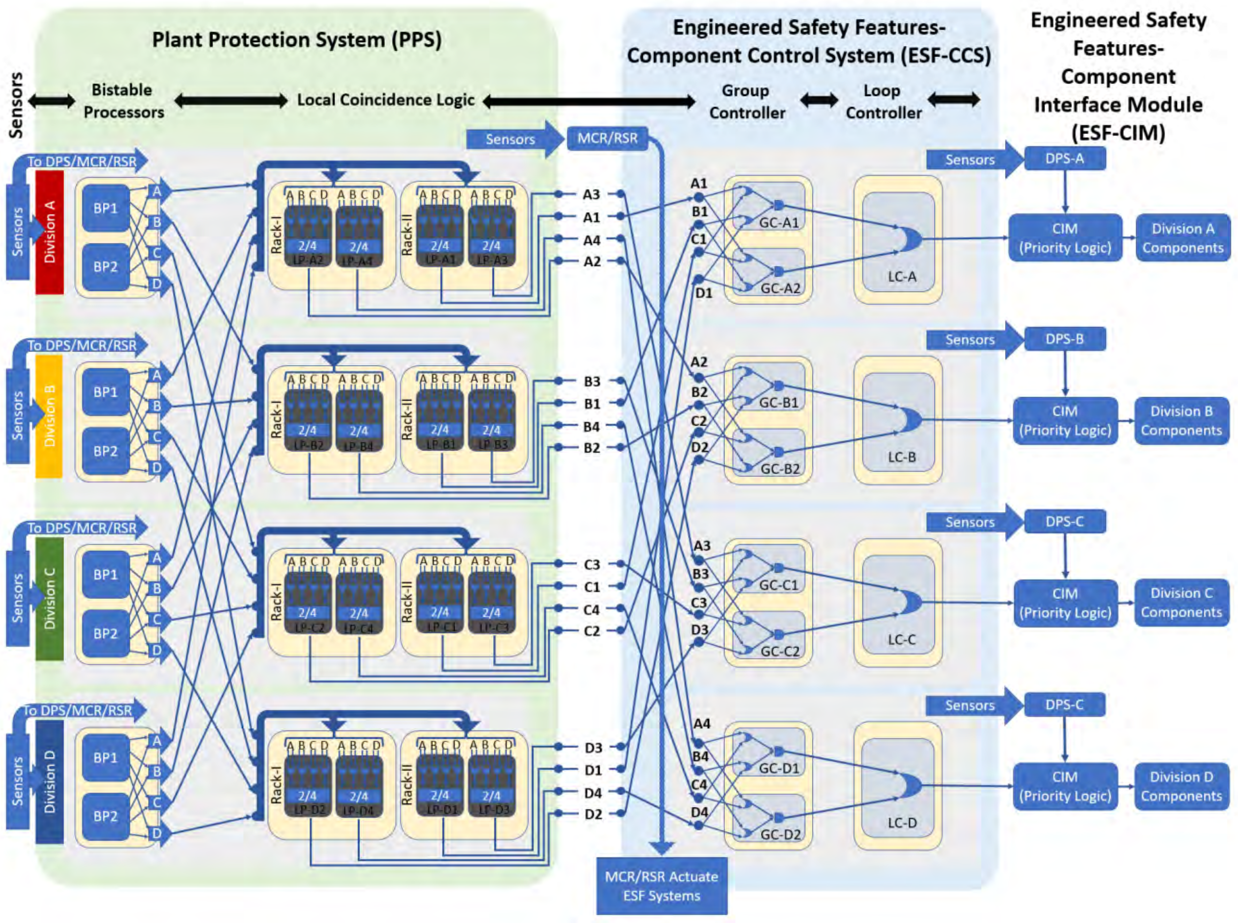


Figure 13. The second diverse configuration four-division digital ESFAS (Black components show Software-2 in ESFAS).

3.3 Case Studies for Cross-system Software Common Cause Failure

This section provides a description of quantifying the CCF of LCL processors across RTS and ESF. Beginning with a comprehensive understanding of each system, the next step involves identifying the coupling factors (or mechanisms) that can share or interconnect RTS and ESFAS. Distinct coupling factors serve as the basis for constructing CCCGs and beta factors are assigned to each CCCG. This is then followed by the calculation of CCF. This quantification approach follows the previous research [5].

3.3.1 Baseline Configuration #1

The baseline configuration #1 for this analysis is outlined as follows:

1. The hardware is identical for all LCL processors.
2. The software of LCL processors for RTS is the same as for ESF.
3. All component failure probabilities are identical to the baseline RTS and ESF.
4. Software is evaluated with BAHAMAS to provide $Q_{SW1} = 2.077E-4$.
5. Certain details of the system that would be common knowledge for a real system had to be assumed for this hypothetical design. These assumptions make it possible to assign subfactor scores.

a. It is assumed that the software employed within the LCL processors is purpose-built.

b. The LCL processors only perform two-out-of-four votes from the BP trip signals.

c. It is assumed that the software is a new design with little operational experience (less than 10 years).

d. The manufacturer recognized and performed a PRA analysis for the CCF of all RTS LCL processors.

e. The manufacturer recognized and performed a PRA analysis for the CCF of all ESF LCL processors.

f. The manufacturer did not recognize a cross system CCF as a threat.

g. There are redundant LCL processors in both RTS and ESF. No diversity is used.

h. The interaction of operators with software is guided by checklists, and interaction is minimal.

i. Maintenance team interactions with the software are guided by procedures that include testing and checking of actions.

j. The maintenance team has regular safety training, safety-oriented work culture, and specialized education. Specialized education indicates they have detailed training and education related to the specific components/software of software.

k. Access to the LCL processors software is strictly controlled. Each division is located in its own secured room where only authorized and trained personnel are allowed access. There are multiple software systems besides the RTS and ESF within each division-specific room.

l. LCL processors in RTS and ESF are tested separately.

In the baseline configuration, the CCCG is identified through coupling factors such as software requirements, human interaction, and common inputs. It is assumed that the software for the LCL processors in both RTS and ESFAS performs identical function of two-out-of-four voting, implying the use of identical software. Common inputs are received from BP, and the same maintenance team and plan are assumed. Accordingly, the CCCG for the baseline configuration can be distinguished in Table 4.

Table 4. CCCG for the LCL processors in the baseline configuration #1.

CCCG		Components			Coupling Factors (or Mechanisms)
			RTS	ESFAS	
CCCG 1	All LCL processors	LCL processor R1	A1, A3	A2, A4	Similar software requirements and function, the same maintenance team/plan, common inputs
		LCL processor R2	A2, A4	A1, A3	
		LCL processor R1	B1, B3	B2, B4	
		LCL processor R2	B2, B4	B1, B3	
		LCL processor R1	C1, C3	C2, C4	
		LCL processor R2	C2, C4	C1, C3	
		LCL processor R1	D1, D3	D2, D4	
		LCL processor R2	D2, D4	D1, D3	

In the next step, subfactors are scored to assess the extent to which each CCCG defends against CCF. These subfactors comprise input similarity, understanding, analysis, man-machine interface (MMI), safety culture and training, control, and testing (detailed descriptions of these subfactors are available in prior report [5]). Given that all LCL processors share common inputs from BP, the input similarity for the 32 LCL processors within the CCCG is scored as grade A. In terms of understanding, the design of the CCCG is not novel for purpose-built LCL processors, and their limited experience leads to grade D. Additionally, it is important to note that the developers and analysts may not have taken into consideration all LCL processors in both RTS and ESF together. However, they have implemented redundant LCL processors, which has led to grade B in the analysis. MMI is scored as grade D due to minimal operator/user interaction and control being guided by checklists, along with maintenance adhering to a test and checking process. Safety culture and training receive grade D based on assumptions of regular safety training, safety-oriented culture, and specialized education. Control is scored as grade D because the LCL processors have limited access granted only to authorized personnel in the secure physical location. RTS and ESF are assumed to be tested separately, leading to grade A for testing since testing is not conducted for both systems concurrently. Based on these considerations, the subfactors are assessed as shown in Table 5, and the result of the CCF analysis is shown in Table 6.

Table 5. Subfactor scores for the baseline configuration #1.

Baseline Configuration #1	
Subfactors	CCCG 1
Input Similarity	A
Understanding	D
Analysis	B
MMI	D
Safety Culture and Training	D
Control	D
Testing	A
Defense factor	0.494
CCF	1.027E-04

Table 6. CCF results for LCL processors of the baseline configuration #1.

Baseline Configuration #1			
Component	IND	CCCG 1	Total
LCL processor	1.050E-04	1.027E-04	2.077E-04

3.3.2 Diverse Configuration #1

The diverse configuration #1 is depicted through Figure 10 and Figure 11, encompassing the utilization of two different software applications. All other assumptions are identical with that of the baseline configuration as follows:

1. The hardware is identical for all LCL processors.
2. Software-1 and Software-2 are diverse. Software-1 is applied to LCL processors of both RTS and ESF within Division A & C, and Software-2 is applied to LCL processors of both RTS and ESF within Division B & D.
3. Software failure probabilities of LCL processors within Division A&C are identical to the baseline RTS and ESF, except for the LCL processors found within Division B&D.
4. Software-1 is evaluated with BAHAMAS to provide $Q_{Sw1} = 2.077E - 4$.
5. As this is a preliminary work, we did not have design information for Software-2; therefore, its attributes had to be assumed. In this case, the $Q_{Sw2} = 1.75Q_{Sw1} = 3.635E - 4$.
6. Certain details of the system that would be common knowledge for a real system had to be assumed for this hypothetical design. These assumptions make it possible to assign subfactor scores.
   a. It is assumed that the software employed within the LCL processors is purpose-built.
   b. The LCL processors only perform two-out-of-four votes from the BP trip signals.

c. It is assumed that the software is a new design with little operational experience (less than 10 years).

d. The manufacturer recognized and performed a PRA analysis for the CCF of all RTS LCL processors.

e. The manufacturer recognized and performed a PRA analysis for the CCF of all ESF LCL processors.

f. The manufacturer did not recognize a cross system CCF as a threat.

g. There are redundant LCL processors in both RTS and ESF.

h. The interactions of operators with Software-1 and Software-2 are guided by checklists and interaction is minimal.

i. Maintenance Team interactions with the Software-1 and the Software-2 are guided by procedures that include testing and checking of actions.

j. Maintenance Team-1 has regular safety training, safety-oriented work culture, and specialized education. Specialized education indicates they have detailed training and education related to the specific components/software of Software-1.

k. Maintenance Team-2 has infrequent safety training and general education related to the specific components/software of Software-2.

l. Access to the LCL processors software is strictly controlled. Each division is located in its own secured room where only authorized and trained personnel are allowed access. There are multiple software systems besides the RTS and ESF within each division-specific room.

m. All LCL processors in both RTS and ESFAS are tested concurrently.

Table 7 shows the CCCGs for the LCL processors in the diverse configuration #1. CCCG 1 is grouped because of the coupling factor involving shared SDLC and common input although it encompasses two distinct software applications. CCCG 2 consists of LCL processors utilizing Software-1, and a maintenance team and plan are assumed. On the other hand, CCCG 3 includes LCL processors employing Software-2, and another maintenance team and plan are assumed. These CCCGs based on distinct software, and maintenance team and plan contribute to a comprehensive analysis of cross-system software CCF.

Table 7. CCCGs for the LCL processors in the diverse configuration #1.

CCCG		Components			Coupling Factors (or Mechanisms)
			RTS	ESF	
CCCG 1	All LCL processors	LCL processor R1	A1, A3	A2, A4	Similar software requirements (shared SDLC), common inputs
		LCL processor R2	A2, A4	A1, A3	
		LCL processor R1	B1, B3	B2, B4	
		LCL processor R2	B2, B4	B1, B3	
		LCL processor R1	C1, C3	C2, C4	
		LCL processor R2	C2, C4	C1, C3	
		LCL processor R1	D1, D3	D2, D4	
		LCL processor R2	D2, D4	D1, D3	
CCCG 2	LCL processors in Division A &C	LCL processor R1	A1, A3	A2, A4	SDLC Software-1, Maintenance Team/Plan 1, common inputs
		LCL processor R2	A2, A4	A1, A3	
		LCL processor R1	C1, C3	C2, C4	
		LCL processor R2	C2, C4	C1, C3	
CCCG 3	LCL processors in Division B&D	LCL processor R1	B1, B3	B2, B4	SDLC Software-2, Maintenance Team/Plan 2, common inputs
		LCL processor R2	B2, B4	B1, B3	
		LCL processor R1	D1, D3	D2, D4	

		LCL processor R2	D2, D4	D1, D3	

CCCG 1 exhibits diversity through different software applications, leading to grade C in subfactor "Analysis". CCCG 2 and CCCG 3 employ identical software, resulting in grade B due to the incorporation of general redundancy. Evaluations of subfactor "Safety culture and training" depend on the maintenance team/plan, resulting in grade D and C, respectively. Other subfactors generally align with the baseline configuration.

Table 8. Subfactor scores for the diverse configuration #1.

Diverse Configuration #1			
Subfactors	CCCG 1	CCCG 2	CCCG 3
Input Similarity	A	A	A
Understanding	D	D	D
Analysis	C	B	B
MMI	D	D	D
Safety Culture and Training	D	D	C
Control	D	D	D
Testing	A	A	A
Defense factor	0.479	0.494	0.497
CCF	4.973E-05	1.027E-04	1.805E-04

The calculation of CCF in this study is based on the approach previously proposed and involves the implementation of the BAHAMAS method. A detailed description of this approach can be found in [5]. The common failure of Software-1 (Q_{SW1}) is assumed as $2.077E-04$ by the BAHAMAS method, and the common failure of Software-2 (Q_{SW2}) is assumed as $3.635E-04$. Additionally, the assumption of $Q_{\mathrm{CCn}} = 0.5Q_{SW1} = 1.038E-04$ is adopted, which represents theoretical common failure for the CCCG that contains both Software-1 and Software-2.

Table 9. CCF results for LCL processors of the diverse configuration #1.

CCF	Diverse Configuration #1			
Component	IND	CCCG 1	CCCG 2	Total
LCL processor in Division A&C	5.531E-05	4.973E-05	1.027E-04	2.077E-04
Component	IND	CCCG 1	CCCG 3	Total
LCL processor in Division B&D	1.332E-04	4.973E-05	1.805E-04	3.635E-04

3.3.3 Baseline Configuration #2

In the baseline configuration #2, the functional requirements of the LCL processors are differentiated between RTS and ESF, even though the software possesses the same general requirements. This configuration assumes that the first baseline configuration remains consistent, but CCCGs are additionally segregated due to functional requirements. For example, LCL software for RTS has RTS function, while LCL software for ESF has ESF function. Table 10 to Table 12 provide a comprehensive overview of the baseline configuration #2, encompassing details regarding the CCCGs, subfactor scores, and the resulting CCF analysis outcomes.

Table 10. CCCGs for the LCL processors in the baseline configuration #2.

CCCG	Components		Coupling Factors (or Mechanisms)
	RTS	ESF	

CCCG 1	All LCL processors	LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2	A1, A3 A2, A4 B1, B3 B2, B4 C1, C3 C2, C4 D1, D3 D2, D4	A2, A4 A1, A3 B2, B4 B1, B3 C2, C4 C1, C3 D2, D4 D1, D3	Similar software design requirements (shared SDLC), the same maintenance team/plan, common inputs
CCCG 2	LCL processors in RTS	LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2	A1, A3 A2, A4 C1, C3 C2, C4 B1, B3 B2, B4 D1, D3 D2, D4		RTS function requirement
CCCG 3	LCL processors in ESF	LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2		A2, A4 A1, A3 C2, C4 C1, C3 B2, B4 B1, B3 D2, D4 D1, D3	ESF function requirement

Table 11. Subfactor scores for the baseline configuration #2.

Baseline Configuration #2			
Subfactors	CCCG 1	CCCG 2	CCCG 3
Input Similarity	A	A	A
Understanding	C	D	D
Analysis	B	C	C
MMI	D	D	D
Safety Culture and Training	D	D	D
Control	D	D	D
Testing	A	C	C
Defense Factor	0.497	0.326	0.326
CCF	1.032E-04	6.774E-05	6.774E-05

Table 12. CCF results for LCL processors of the baseline configuration #2.

CCF	Baseline Configuration #2			
Component	IND	CCCG 1	CCCG 2	Total
LCL processor in RTS	3.673E-05	1.032E-04	6.774E-05	2.077E-04
Component	IND	CCCG 1	CCCG 3	Total
LCL processor in ESF	3.673E-05	1.032E-04	6.774E-05	2.077E-04

3.3.4 Diverse Configuration #2

The diverse configuration #2, which is derived from the baseline configuration #2, assumes the deployment of Software-1 in RTS and Software-2 in ESF. Table 13 to Table 15 provide a detailed representation of the CCCGs, subfactor scores, and the calculated CCF.

Table 13. CCCGs for the LCL processors in the diverse configuration #2 from the baseline configuration #2.

CCCG		Components	RTS	ESF	Coupling Factors (or Mechanisms)
CCCG 1	All LCL processors	LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2	A1, A3 A2, A4 B1, B3 B2, B4 C1, C3 C2, C4 D1, D3 D2, D4	A2, A4 A1, A3 B2, B4 B1, B3 C2, C4 C1, C3 D2, D4 D1, D3	Similar software design requirements (shared SDLC), common inputs
CCCG 2	LCL processors in RTS	LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2	A1, A3 A2, A4 C1, C3 C2, C4 B1, B3 B2, B4 D1, D3 D2, D4		SDLC Software-1, Maintenance Team/Plan 1, RTS function requirement, common inputs
CCCG 3	LCL processors in ESF	LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2 LCL processor R1 LCL processor R2		A2, A4 A1, A3 C2, C4 C1, C3 B2, B4 B1, B3 D2, D4 D1, D3	SDLC Software-2, Maintenance Team/Plan 2, ESF function requirement, common inputs

Table 14. Subfactor scores for the diverse configuration #2 from the baseline configuration #2.

Diverse Configuration #2			
Subfactors	CCCG 1	CCCG 2	CCCG 3
Input Similarity	A	A	A
Understanding	C	D	D
Analysis	C	C	C
MMI	D	D	D
Safety Culture and Training	D	D	C
Control	D	D	D
Testing	A	C	C
Defense Factor	0.482	0.326	0.329
CCF	5.002E-05	6.774E-05	1.194E-04

Table 15. CCF results for LCL processors of the diverse configuration #2 from the baseline configuration #2.

CCF	Diverse Configuration #2			
Component	IND	CCCG 1	CCCG 2	Total
LCL processors of RTS	8.994E-05	5.002E-05	6.774E-05	2.077E-04
Component	IND	CCCG 1	CCCG 3	Total
LCL processors of ESF	1.941E-05	5.002E-05	1.194E-04	3.635E-04

3.4 Discussion

This section performs a preliminary study on potential cross-system software CCFs in a four-division representative RTS and ESFAS. These systems incorporate a variety of redundant and diverse software designs.

Baseline configuration #1 ad #2 assumes no software diversity, while Diverse configuration #1 introduces division-level software diversity in LCL processors (e.g., LCL processors in Division A&C with software #1 and LCL processors in Division B&D with software #2). Diverse configuration #2, on the other hand, incorporates system-level software diversity in the LCL processors (e.g., LCL processors in RTS with software #1 and LCL processors in ESFAS with software #2).

CCCGs are identified based on commonality in LCL processor inputs and software designs. Table 16 provides a comparison of the CCF results for LCL processors in RTS Division &C from four case studies. The results demonstrate that the implementation of software diversity significantly reduces the probability of cross-system software CCFs from 1E-4 to 1E-5. Such CCFs, if they occur, could lead to failures in both RTS and ESFAS when critical functions like reactor trip and safety features need activation. In future studies, we will evaluate how cross-system CCFs and diversity implementations can impact the probabilities of failure in RTS and ESFAS, as well as overall plant safety during various accident scenarios.

Table 16. Comparisons of CCF results for LCL processors in RTS Division &C from four case studies.

Baseline Configuration #1			
CCCG 1 (All LCL processors)	CCCG 2 (LCL processor in Division A&C)	IND	Total
1.027E-04	NA	1.050E-04	2.077E-04
Diverse Configuration #1			
CCCG 1 (All LCL processors)	CCCG 2 (LCL processor in Division A&C)	IND	Total
4.973E-05	1.027E-04	5.531E-05	2.077E-04
Baseline Configuration #2			
CCCG 1 (All LCL processors)	CCCG 2 (LCL processors in RTS)	IND	Total
1.032E-04	6.774E-05	3.673E-05	2.077E-04
Baseline Configuration #2			
CCCG 1 (All LCL processors)	CCCG 2 (LCL processors in RTS)	IND	Total
5.002E-05	6.774E-05	8.994E-05	2.077E-04

Identifying coupling factors that lead to CCFs across systems can be challenging. Even when different programming languages, algorithms, and approaches are used, the potential for CCF remains. There are unpredictable failure modes and spurious operations, and their impacts on other systems can emerge, sometimes leading to cascading effects. The software of interconnected system means that failures can spread through different parts of the system, but pinpointing the exact causes and predicting their effects can be difficult. Despite efforts to ensure diversity, the intricate interactions within software and between systems can still result in unexpected CCFs.

In the context of "Awareness" term in the subfactor of "Analysis", it's important to differentiate between redundant features that are specifically designed as a defense against cross-system CCF and those that are implemented for other reasons, such as preventing different types of CCFs. If there is no evidence of a deliberate implementation of redundancy as a countermeasure against cross-system CCF, it would not be appropriate to credit or consider these redundancies in the assessment of cross-system CCF.

Future work should address this by providing guidelines and recommendations for addressing cross-system concerns in a systematic and deliberate manner.

Identifying and understanding data communication also plays a crucial role in identifying interconnected systems. It may significantly influence “Feedback” within the subfactor “Analysis”. Gaining insights into how data flows and is exchanged between different components and systems holds the potential to enhance the overall understanding of interconnected systems, including potential CCFs.

The further development and improvement of the proposed approach will be carried on in FY-24.

4. ADVANCED HUMAN-SYSTEM INTERFACE RISK ANALYSIS BASED ON REDUNDANCY-GUIDED SYSTEMS-THEORETIC HAZARD ANALYSIS AND HUMAN RELIABILITY ANALYSIS

This section proposes an innovative approach on the risk assessment of HSI by integrating RESHA and HRA methods. Section 4.1 introduces the technical background of this work. Section 4.2 reviews how the risk of HSI is evaluated in HRA. Section 4.3 proposes an approach to HSI risk evaluation within DI&C Systems, which is demonstrated in Section 4.4.

4.1 Introduction

In PRA, the HSI design has been mainly considered and evaluated in terms of HRA [34, 35]. HSIs play an important role in enabling operators to communicate with the NPP side (e.g., getting the information required to understand an NPP's status or perform necessary actions for responding to a given operational context). In HRA, it has been assumed that good/poor HSIs decrease/increase HEPs. This notion is based on the performance-shaping factor (PSF) concept, which highlights error contributors and adjusts nominal HEPs in respect to PSFs such as stress, HSI, or task complexity.

The application of digital technology is a recent trend in the design of NPP main control rooms (MCRs). Newly constructed NPPs in various countries around the globe (e.g., the APR1400 in Korea [36], AP1000 in the United States [37], and EPR-1600 in France [38]) adopt fully digitalized and computerized MCRs. Digital MCRs possess characteristics that distinguish them from conventional (i.e., analog) MCRs. These traits include increased automation and the use of computer-based HSIs and/or intelligent operator aids. Many researchers believe these new features carry the potential to improve both human performance and nuclear safety. However, the current PSF-based HRA approach may be insufficient for specifically reflecting the risk carried by digital HSIs that present (or summarize) the information flow of DI&C systems.

This study proposes a method of specifically evaluating HSIs for DI&C systems, based on RESHA and HRA. RESHA, a method for analyzing DI&C systems with redundancy features, was technically developed based on the STPA method [8]. RESHA identifies the hazards and failure modes of digital systems and provides a foundation for quantification efforts such as reliability and consequences analyses. In this study, we first investigated how HSIs are evaluated in HRA methods because HSIs have been mainly considered in HRA. To better evaluate HSIs for DI&C systems, we extended the existing HSI evaluation process in HRA by enlarging the HSI evaluation coverage and HRA event tree (ET). Then, feasibility of the approach was identified based on the APR1400 DI&C systems and a RTS of the GPWR PRA model.

4.2 Traditional HSI Evaluation using HRA

As mentioned above, HSIs have been mainly evaluated in the context of HRA. Table 17 summarizes the various HSI-related PSFs, as well as the capabilities of four different HRA methods to properly evaluate these PSFs. These four methods are the Standardized Plant Analysis Risk-HRA (SPAR-H) [35], Cause-based Decision Tree (CBDT) [39], Cognitive Reliability and Error Analysis Method (CREAM) [40], and Integrated Human Event Analysis System for Event and Condition Assessment (IDHEAS-ECA) [41]. In HRA, practitioners review the HSIs required for human actions, determine their PSF levels, then use them to estimate the final HEPs.

The current HSI evaluation process using HRA faces challenges on two key fronts. Firstly, the existing HRA methods are effective at assessing how a well-executed HSI impacts the likelihood of human errors. However, they fall short in determining whether an analog HSI surpasses a digital one or which of two distinct digital HSIs is superior. This limitation can result in inconsistent HRA outcomes when evaluating the significance of different HSI designs.

Secondly, in traditional PRA models, the event of "failure of manual control via HSIs" typically combines HSI system failures with operator errors (i.e., HEPs) estimated through HRA. This approach does not adequately consider the influence of HSI hardware or software malfunctions on the quantification of HEPs. In reality, certain critical tasks performed by humans may carry a higher risk of errors or even guaranteed failure due to HSI degradation. Nonetheless, this critical aspect is often overlooked in existing HRA methodologies. Consequently, there is a pressing need for a novel HRA approach for HSI evaluation that considers the impact of HSI degradation on the accurate quantification of HEPs.

Table 17 Summary of how HSI-related PSFs are handled in four different HRA methods.

HRA Method	**PSF**	**PSF Level**
SPAR-H [35]	Ergonomics/HSI	Missing/misleading
		Poor
		Nominal
		Good
		Insufficient information
CBDT [39]	Indicator available in control room	Yes
		No
	Accuracy of control room indicator	Yes
		No
	Front vs. back panel	Front
		Back
	Alarmed vs. not alarmed	Alarmed
		Not alarmed
	Indicator easy to locate	Easy
		Not easy
	Good/bad indicator	Good
		Bad
CREAM [40]	Adequacy of HSI and operational support	Supportive
		Adequate
		Tolerable
		Inappropriate
IDHEAS-ECA [41]	HSI	No impact
		Indicator is similar to other nearby sources of information
		No sign or indication of any technical difference from adjacent sources (meters, indicators)
		Information regarding a given task is spatially distributed, unorganized, or cannot be accessed in simultaneous fashion
		Unintuitive or unconventional indications

		Poor salience of the target (indicators, alarms, alerts) out of the crowded background
		Inconsistent formats, units, symbols, or tables
		Inconsistent interpretation of displays
		Similarity in elements
		Poor functional localization
		Ergonomic deficits
		Labeling of the controls is confusing or does not agree with the document nomenclature
		Controls lack labels or indications
		Controls provide inadequate or ambiguous feedback
		Confusion in action maneuver states
		Unclear functional allocation

4.3 Methodology

To handle the two issues introduced above and to specifically evaluate HSIs by reflecting the specific structures and characteristics of DI&C systems, this study enlarges the scope of the existing HSI evaluation process in the manner illustrated in Figure 14.

The approach suggested in the present study considers the HSI back-end hardware (I&C components supporting the back-end software functions), HSI back-end software (system, structure, logics, and data), HSI front-end hardware (visual and controllable parts) and HSI front-end software (human factors design), whereas the current HSI evaluation process concentrates solely on the relationship between the front-end software and human performance. By conducting a comprehensive evaluation that involves detailed modeling of both back-end and front-end aspects, this approach assesses the potential risks associated with digital HSIs themselves as well as the interactions between humans and these HSIs.

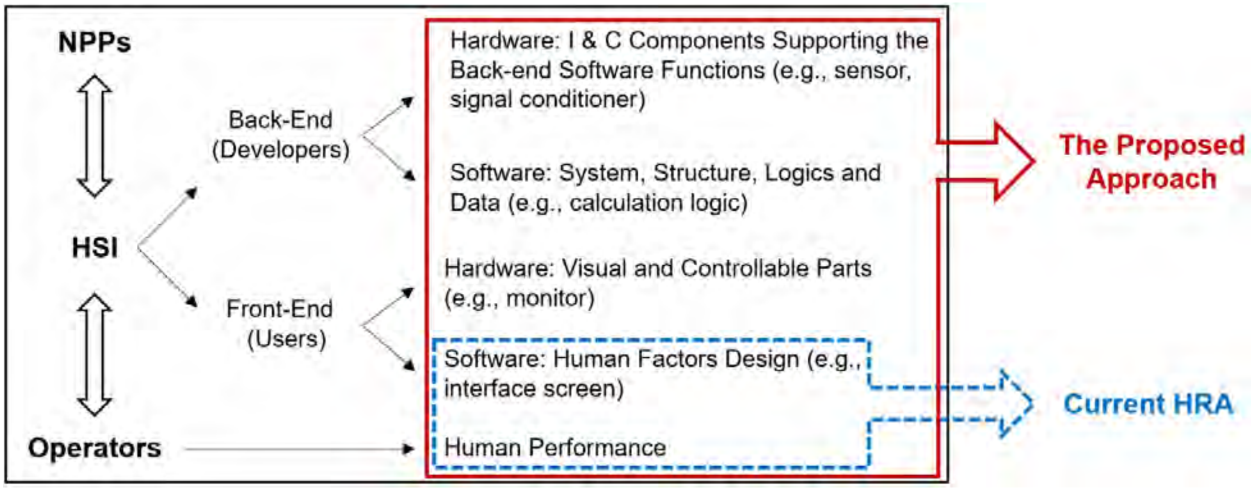

NPPs
HSI
Operators
Back-End (Developers)
Front-End (Users)
Hardware: I & C Components Supporting the Back-end Software Functions (e.g., sensor, signal conditioner)
Software: System, Structure, Logics and Data (e.g., calculation logic)
Hardware: Visual and Controllable Parts (e.g., monitor)
Software: Human Factors Design (e.g., interface screen)
Human Performance
The Proposed Approach
Current HRA

Figure 14. Extension of HSI Evaluation Process via the Suggested Approach.

As this approach alters the way HSIs are evaluated, the present study suggests that the existing HRA quantification method should be updated. In the existing HRA ET representing the HEP quantification (see Figure 15), it is assumed that all HSIs are well executed as designed. A human failure event is subdivided into various tasks, with the final HEP being computed by summing the HEPs for these tasks.

On the other hand, Figure 16 illustrates an extended HRA ET designed specifically for modeling HSIs. It distinguishes between two distinct scenarios: (1) situations where HSIs function optimally, aligning with the existing HRA approach, assuming that HSIs provide operators with accurate information. (2) marked by HSI degradation due to software or hardware failures, HSIs may supply erroneous information or fail to provide adequate information to operators.

From the extended HRA ET shown in Figure 16, we defined an equation representing the final HEP consisting of the HEP given the HSI hardware and software function well and the HEP given HSI hardware and software are degraded as shown in Equation (23) below.

$$HEP_{final} = P_{HSI\_Degraded} \times HEP_{HSI\_Degraded} + (1 - P_{HSI_{Degraded}}) \times HEP_{HSI\_success} \tag{23}$$

The proposed method consists of three steps to evaluate HSIs for DI&C systems. First, the HSI degradation probability is estimated via RESHA, which offers a structured way of modeling hardware/software failure in DI&C systems. RESHA's outputted summaries of HSI conditions are generated as FTs (i.e., HSI FTs). The quantitative values in the FTs come from [4]. Second, the HEP under HSI degradation is quantified by HRA methods. The suggested approach employs the IDHEAS-ECA method since they provide relatively specific guidance for analyzing the quality of HSIs. The last step is integrating the equation and the probabilities into PRA models.

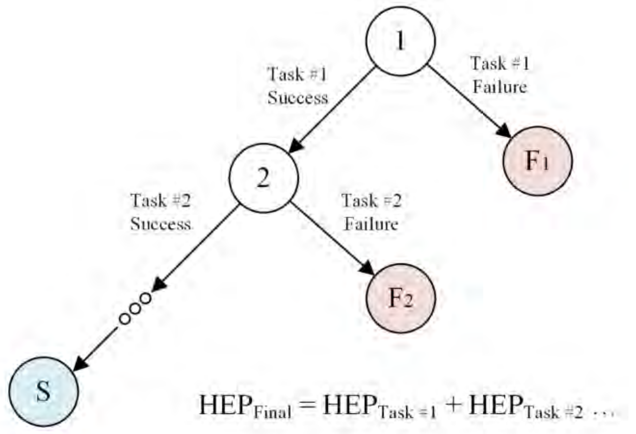


Figure 15. A HRA ET Representing Estimation of HEPs in Current HRA.

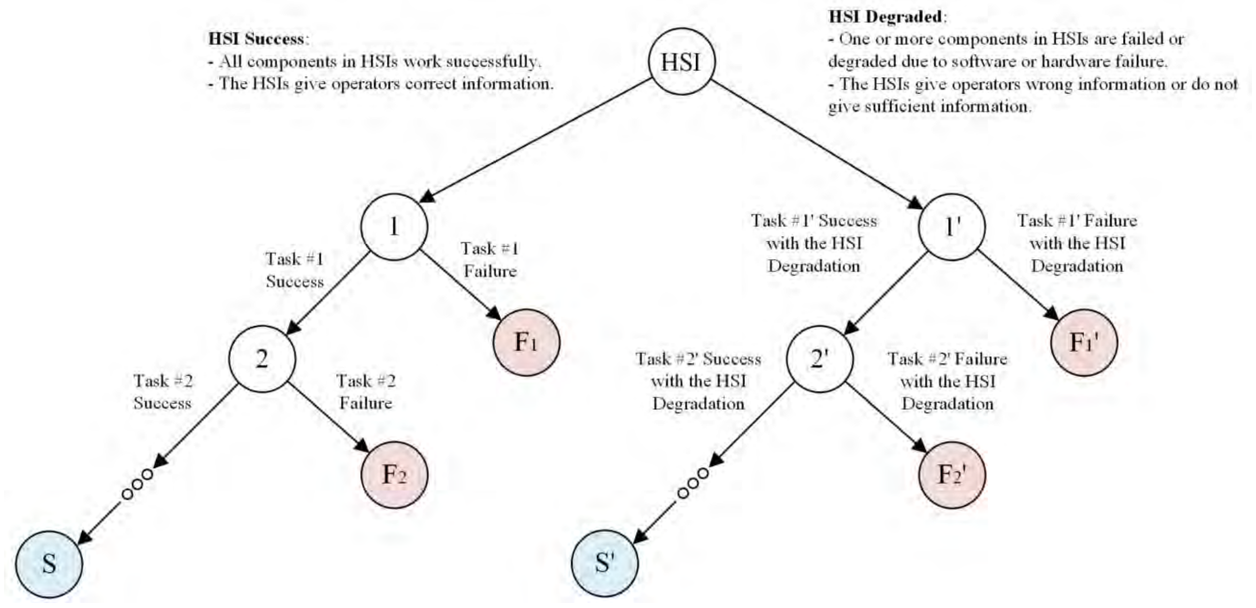

HSI Success:
- All components in HSIs work successfully.
- The HSIs give operators correct information.
HSI
HSI Degraded:
- One or more components in HSIs are failed or degraded due to software or hardware failure.
- The HSIs give operators wrong information or do not give sufficient information.
1
Task #1 Success
Task #1 Failure
F1
2
Task #2 Success
Task #2 Failure
F2
S
1'
Task #1' Success with the HSI Degradation
Task #1' Failure with the HSI Degradation
F1'
2'
Task #2' Success with the HSI Degradation
Task #2' Failure with the HSI Degradation
F2'
S'

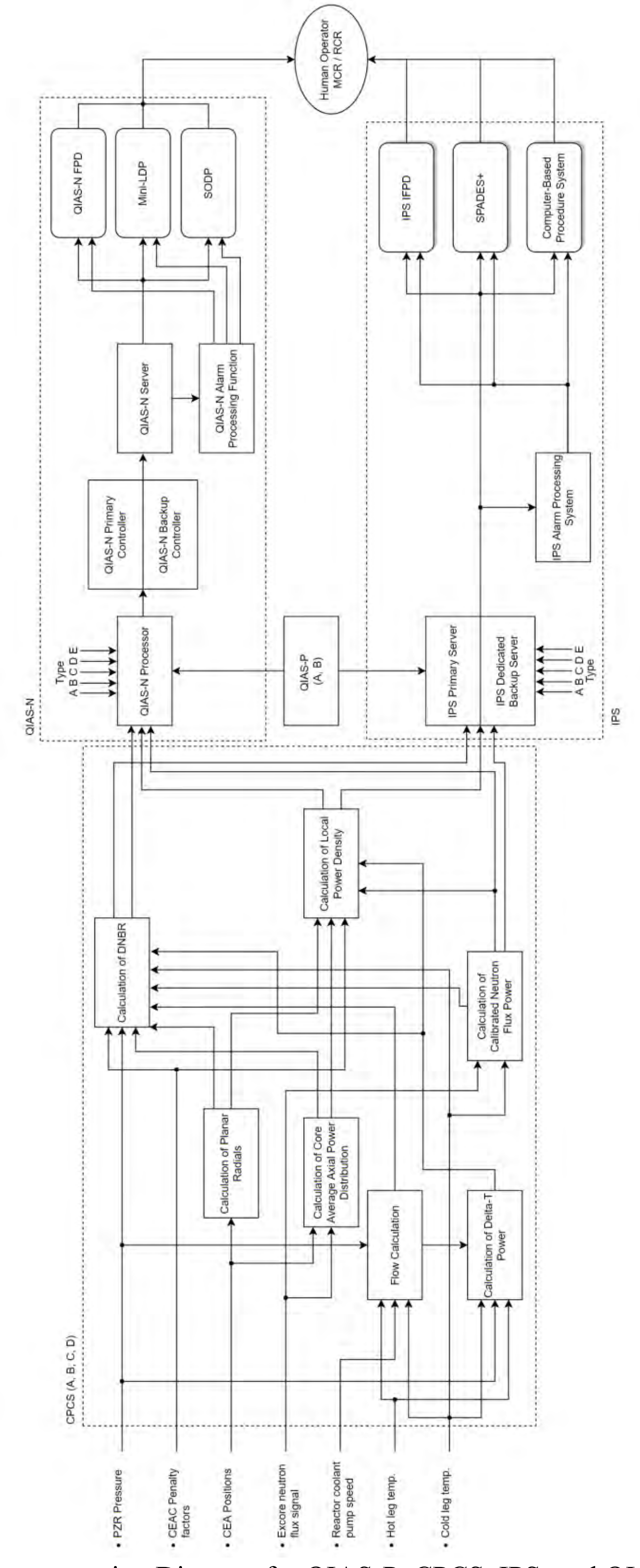
Figure 16. A HRA Event Tree Extended for Specifically Modeling HSIs.

4.4 Case Study

To test the feasibility of the method, we used the APR1400 DI&C systems prepared for the design certification application to U.S. NRC [42] and a RTS FT of the GPWR PRA model. From the RTS FT, an operator action failing to respond with RPS signal present is assumed as the action affected by four DI&C systems in APR1400 (i.e., qualified indication and alarm system [QIAS]-P, QIAS-N, information processing system [IPS], and core protection calculator system [CPCS]). These DI&C systems contribute to calculate important values and provide all the necessary information in HSIs needed for performing human actions. If wrong information is given to operators due to the HSI degradation, operators may not be able to trip the reactor at the right time.

4.4.1 Step #1 – Development of HSI FT-based on RESHA

This section mainly explores how we developed an HSI FT. The HSI FT includes degraded HSI conditions affecting the human action of initiating a manual reactor trip. Upon completing review, the APR1400 DI&C system [42] was used to develop the HSI FT, in combination with RESHA. Within the APR1400 DI&C system, those systems of greatest relevance to the human action (i.e., QIAS-P, CPCS, IPS, and QIAS-N) were specifically modeled in the HSI FT. Figure 17 shows the piping and instrumentation diagrams for QIAS-P, CPCS, IPS, and QIAS-N. The details of these diagrams were assumed based on the APR1400 DI&C system.



QIAS-N
Type A B C D E
QIAS-N Processor
QIAS-N Primary Controller
QIAS-N Backup Controller
QIAS-N Server
QIAS-N Alarm Processing Function
QIAS-N FPD
Mini-LDP
SODP
Human Operator MCR / RCR
QIAS-P (A, B)
IPS
IPS Primary Server
IPS Dedicated Backup Server
Type A B C D E
IPS Alarm Processing System
IPS IFPD
SPADES+
Computer-Based Procedure System
CPCS (A, B, C, D)
Calculation of DNBR
Calculation of Local Power Density
Calculation of Calibrated Neutron Flux Power
Calculation of Planar Radials
Calculation of Core Average Axial Power Distribution
Flow Calculation
Calculation of Delta-T Power
PZR Pressure
CEAC Penalty factors
CEA Positions
Excore neutron flux signal
Reactor coolant pump speed
Hot leg temp.
Cold leg temp.

Figure 17. Piping and Instrumentation Diagram for QIAS-P, CPCS, IPS, and QIAS-N.

First, the QIAS-P is simplified into a diagram, having already been analyzed in our previous study [4]. The information from the QIAS-P is provided to the QIAS-N processor and the IPS server. Second, the CPCS, which has four divisions (i.e., A, B, C, and D), offers several modules for calculating core-related values such as the departure from nucleate boiling and local power density, based on plant parameters (e.g., hot leg temperature). The CPCS then sends the values to the QIAS-N processor and the IPS server. Third, the QIAS-N system's processor, primary and backup controllers, server, and alarm processing function help gather plant information and analyze the data, while the QIAS-N front panel displays, mini-large display panel, and shutdown overview display panel present the information to operators in MCRs. Last, the IPS encompasses (1) the primary and dedicated backup servers for collecting information from the QIAS-P and CPCS, (2) the alarm processing system, and (3) applications such as the IPS information flat panel display, safety parameter display and evaluation system+, and CPS.

Figure 18 shows the top event of the HSI FT. The top event logic consists of FT gates that represent those cases in which QIAS-P (Figure 19), QIAS-N (Figure 20), and IPS (Figure 21) fail to notify operators via an alarm and fail to accurately reflect safety variables under a degraded reactor state. The CPCS is modeled in QIAS-N and IPS as shown in Figure 20 and Figure 21. These FT logics include the front-end hardware failure (e.g., hardware failure of display monitor), the back-end hardware failure (e.g., onboard integrated circuits experiencing burnout or loose joint causing complete failure), or the back-end software failure (e.g., unsafe information flow [UIF] representing software failure mode or CCFs in software). The failure rates of basic events modeled in these FTs are from [4] [5].

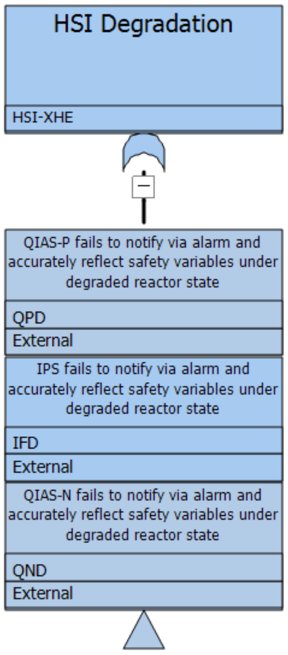
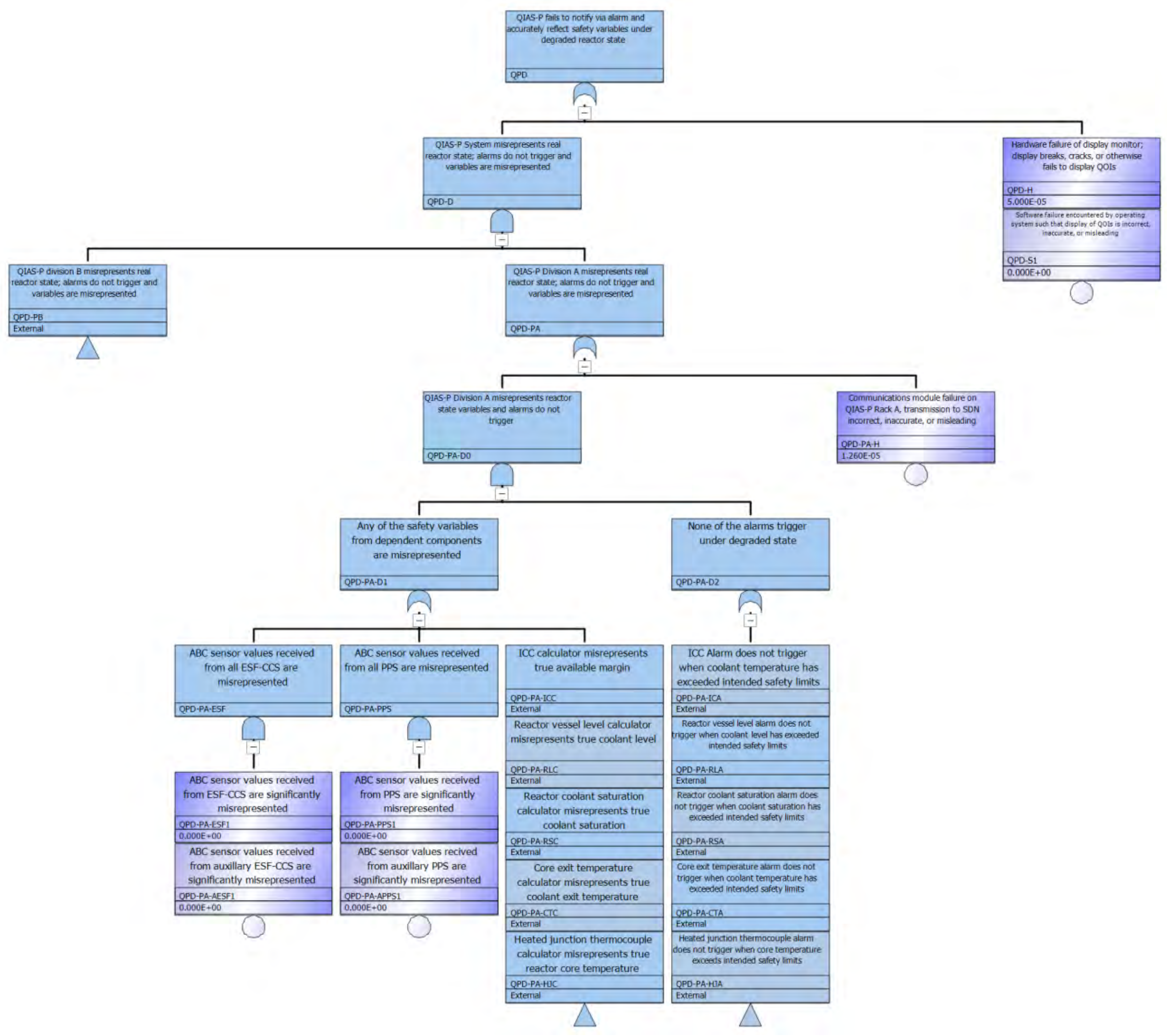

HSI Degradation
HSI-XHE
QIAS-P fails to notify via alarm and accurately reflect safety variables under degraded reactor state
QPD
External
IPS fails to notify via alarm and accurately reflect safety variables under degraded reactor state
IFD
External
QIAS-N fails to notify via alarm and accurately reflect safety variables under degraded reactor state
QND
External

Figure 18. Top Event of the HSI Fault Tree.



QIAS-P fails to notify via alarm and accurately reflect safety variables under degraded reactor state
QPD
QIAS-P System misrepresents real reactor state; alarms do not trigger and variables are misrepresented
QPD-D
Hardware failure of display monitor; display breaks, cracks, or otherwise fails to display QOIs
QPD-H
5.000E-05
Software failure encountered by operating system such that display of QOIs is incorrect, inaccurate, or misleading
QPD-S1
0.000E+00
QIAS-P division B misrepresents real reactor state; alarms do not trigger and variables are misrepresented
QPD-PB
External
QIAS-P Division A misrepresents real reactor state; alarms do not trigger and variables are misrepresented
QPD-PA
QIAS-P Division A misrepresents reactor state variables and alarms do not trigger
QPD-PA-D0
Communications module failure on QIAS-P Rack A, transmission to SDN incorrect, inaccurate, or misleading
QPD-PA-H
1.260E-05
Any of the safety variables from dependent components are misrepresented
QPD-PA-D1
None of the alarms trigger under degraded state
QPD-PA-D2
ABC sensor values received from all ESF-CCS are misrepresented
QPD-PA-ESF
ABC sensor values received from all PPS are misrepresented
QPD-PA-PPS
ICC calculator misrepresents true available margin
QPD-PA-ICC
External
Reactor vessel level calculator misrepresents true coolant level
QPD-PA-RLC
External
Reactor coolant saturation calculator misrepresents true coolant saturation
QPD-PA-RSC
External
Core exit temperature calculator misrepresents true coolant exit temperature
QPD-PA-CTC
External
Heated junction thermocouple calculator misrepresents true reactor core temperature
QPD-PA-HJC
External
ICC Alarm does not trigger when coolant temperature has exceeded intended safety limits
QPD-PA-ICA
External
Reactor vessel level alarm does not trigger when coolant level has exceeded intended safety limits
QPD-PA-RLA
External
Reactor coolant saturation alarm does not trigger when coolant saturation has exceeded intended safety limits
QPD-PA-RSA
External
Core exit temperature alarm does not trigger when coolant temperature has exceeded intended safety limits
QPD-PA-CTA
External
Heated junction thermocouple alarm does not trigger when core temperature exceeds intended safety limits
QPD-PA-HJA
External
ABC sensor values received from ESF-CCS are significantly misrepresented
QPD-PA-ESF1
0.000E+00
ABC sensor values received from auxillary ESF-CCS are significantly misrepresented
QPD-PA-AESF1
0.000E+00
ABC sensor values received from PPS are significantly misrepresented
QPD-PA-PPS1
0.000E+00
ABC sensor values recived from auxillary PPS are significantly misrepresented
QPD-PA-APPS1
0.000E+00

Figure 19. Fault Tree Logic for QIAS-P.

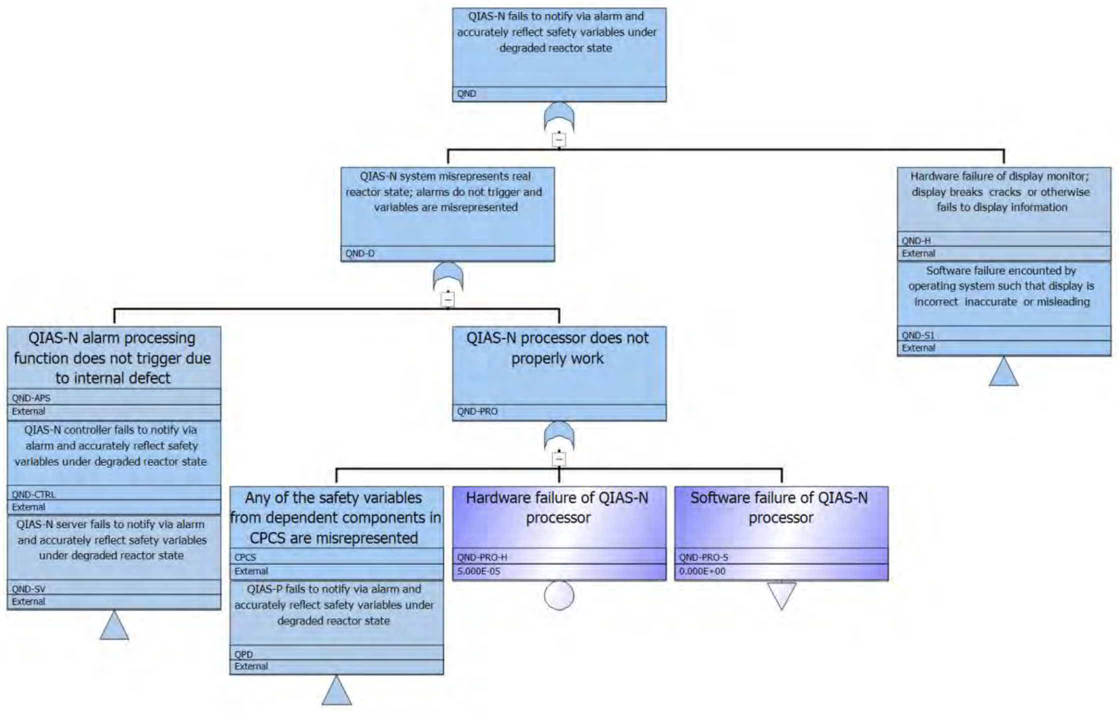

QIAS-N fails to notify via alarm and accurately reflect safety variables under degraded reactor state
QND
QIAS-N system misrepresents real reactor state; alarms do not trigger and variables are misrepresented
QND-D
Hardware failure of display monitor; display breaks cracks or otherwise fails to display information
QND-H
External
Software failure encounted by operating system such that display is incorrect inaccurate or misleading
QND-S1
External
QIAS-N alarm processing function does not trigger due to internal defect
QND-APS
External
QIAS-N controller fails to notify via alarm and accurately reflect safety variables under degraded reactor state
QND-CTRL
External
QIAS-N server fails to notify via alarm and accurately reflect safety variables under degraded reactor state
QND-SV
External
QIAS-N processor does not properly work
QND-PRO
Any of the safety variables from dependent components in CPCS are misrepresented
CPCS
External
QIAS-P fails to notify via alarm and accurately reflect safety variables under degraded reactor state
QPD
External
Hardware failure of QIAS-N processor
QND-PRO-H
5.000E-05
Software failure of QIAS-N processor
QND-PRO-S
0.000E+00

Figure 20. Fault Tree Logic for QIAS-N.

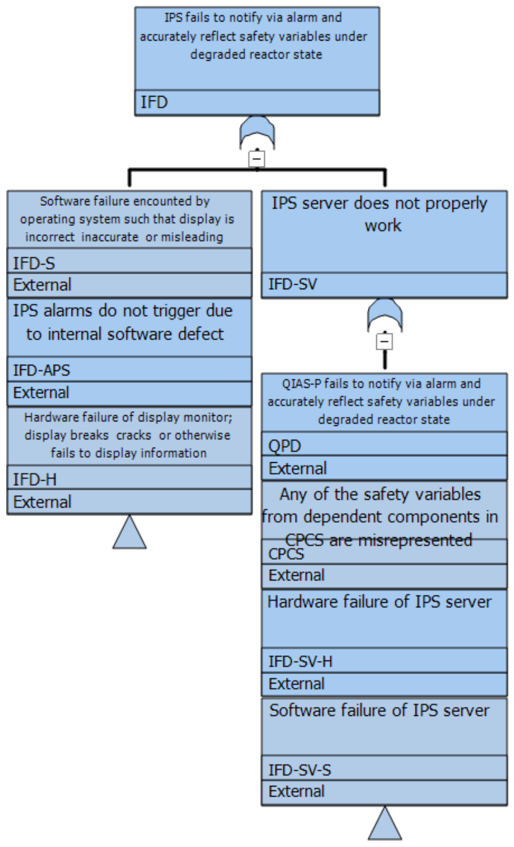

IPS fails to notify via alarm and accurately reflect safety variables under degraded reactor state
IFD
Software failure encounted by operating system such that display is incorrect inaccurate or misleading
IFD-S
External
IPS alarms do not trigger due to internal software defect
IFD-APS
External
Hardware failure of display monitor; display breaks cracks or otherwise fails to display information
IFD-H
External
IPS server does not properly work
IFD-SV
QIAS-P fails to notify via alarm and accurately reflect safety variables under degraded reactor state
QPD
External
Any of the safety variables from dependent components in CPCS are misrepresented
CPCS
External
Hardware failure of IPS server
IFD-SV-H
External
Software failure of IPS server
IFD-SV-S
External

Figure 21. Fault Tree Logic for IPS.

This study calculates failure probabilities and cutsets for three DI&C systems (i.e., QIAS-P, QIAS-N, and IPS), as well as for the top event. This FT is quantified by utilizing SAPHIRE 8, using truncation level 1E-12. Table 18 compares failure probabilities, numbers of cutsets, and cutset rankings for the top event, QIAS-P, QIAS-N, and IPS. The failure probability of the top event is 9.21E-4 with 394 cutsets. The failure probabilities for QIAS-N and IPS are 4.84e-4 and 5.34e-4, respectively, whereas that of QIAS-P is 9.66E-5. In other words, the safety-critical I&C system (i.e., QIAS-P) shows a lower failure probability than the non-safety-related I&C systems (i.e., QIAS-N and IPS). Note that the basic events from QIAS-N and IPS account for cutset rankings #1–5 in the failure probability of the top event.

Table 18. Comparison of Failure Probabilities and Numbers of Cutsets.

		Top Event	QIAS-P	QIAS-N	IPS
Failure Probability		9.21e-4	9.66e-5	4.84e-4	5.34e-4
# of Cutsets		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

4.4.2 Step #2 – HRA for Human Actions under HSI Degradation

This study used the IDHEAS-ECA method to calculate HEPs for a human action under HSI degradation and when all HSIs successfully work, respectively. The human action is that operators fail to respond with RPS signal present as introduced at the beginning of this session. Table 19 indicates the summary of performance influencing factor (PIF) evaluation for the human action under HSI degradation and when all HSIs successfully work, respectively. The PIF is intrinsically the same with PSFs, but the different term is used in the IDHEAS-ECA method.

All the PIFs are evaluated as nominal for the human action when all HSIs successfully work because the action is relatively easy and well-known to operators. On the other hand, operators may have not experienced and been trained for the situation that HSIs give them wrong information or some of HSIs have failed. With this assumption, the PIFs, such as scenario familiarity, task complexity, procedures and guidance, training and experience, and system and I&C transparency, are negatively evaluated as shown in Table 19.

Table 19. The PIF evaluation using the IDHEAS-ECA method.

HEPs for a Human Action under HSI Degradation and When All HSIs Successfully Work	The IDHEAS-ECA PIF Evaluation	HEP Value
$HEP_{HSI\_Success}$	• All the PIFs are evaluated as nominal.	1.20e-3
$HEP_{HSI\_Degraded}$	• Scenario familiarity: infrequently performed scenarios • Task complexity: no cue or mental model for detection • Procedures and guidance: procedure lacks details • Training and experience: operator is inexperienced • System and IC transparency: system or I&C does not behave as intended under special conditions	5.58e-1

4.4.3 Step #3 – Integration Outputs into PRA Models

The last step is integrating the equation and the probabilities into PRA models. The final HEP can be calculated using Equation (23). Figure 22 shows the RTS FT including the HSI failure. The outputs from the first step (i.e., the HSI FT) and the second step (i.e., the basic event—the human action under HSI degradation) are modeled in the fault tree. There was a 9% increase after adding the HSI failure. Table 20 indicates the cutsets for the RTS fault trees before and after adding the HSI failure. The cutsets including the basic events modeled in the HSI failure tree are showing up from the fifth highest cutset ranking. Table 21 including the HSI failure. By following the Fussell-Vesely (FV) values, the fourth, seventh, eighth, and tenth rankings are the basic events coming from the HSI failure tree.

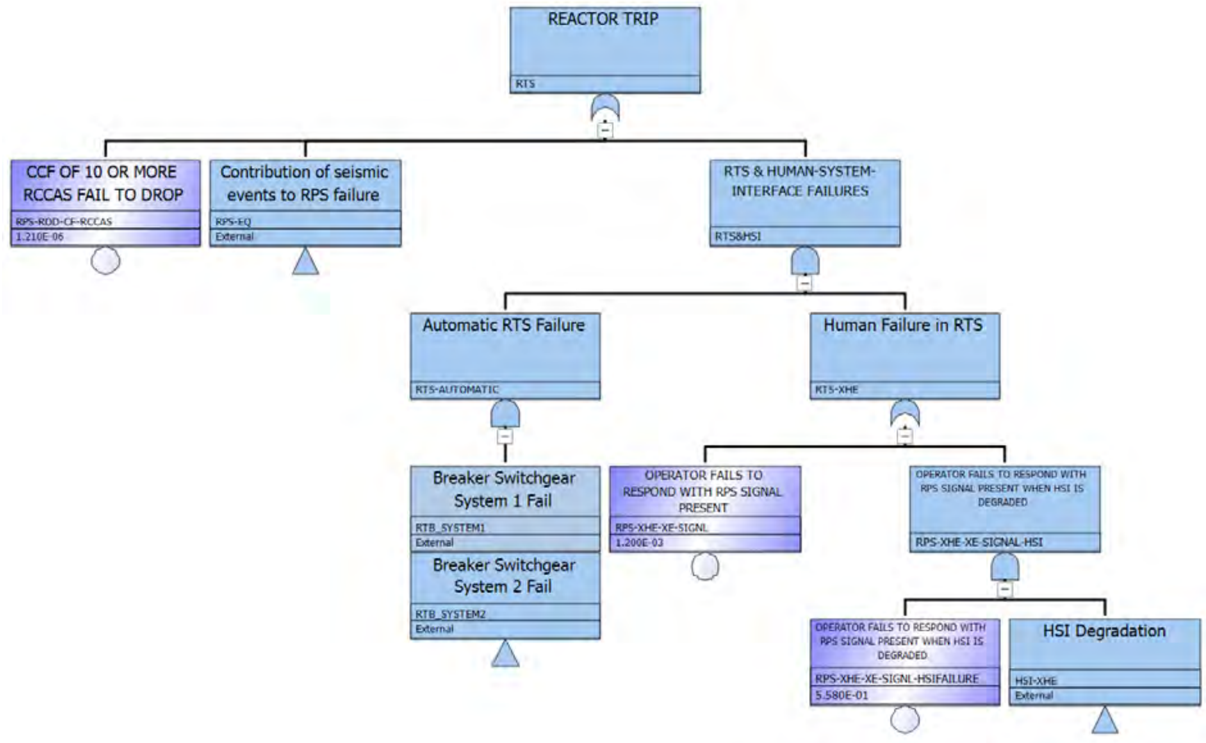

REACTOR TRIP
RTS
CCF OF 10 OR MORE RCCAS FAIL TO DROP
RPS-ROD-CF-RCCAS
1.210E-06
Contribution of seismic events to RPS failure
RPS-EQ
External
RTS & HUMAN-SYSTEM-INTERFACE FAILURES
RTS&HSI
Automatic RTS Failure
RTS-AUTOMATIC
Human Failure in RTS
RTS-XHE
Breaker Switchgear System 1 Fail
RTB_SYSTEM1
External
Breaker Switchgear System 2 Fail
RTB_SYSTEM2
External
OPERATOR FAILS TO RESPOND WITH RPS SIGNAL PRESENT
RPS-XHE-XE-SIGNL
1.200E-03
OPERATOR FAILS TO RESPOND WITH RPS SIGNAL PRESENT WHEN HSI IS DEGRADED
RPS-XHE-XE-SIGNAL-HSI
OPERATOR FAILS TO RESPOND WITH RPS SIGNAL PRESENT WHEN HSI IS DEGRADED
RPS-XHE-XE-SIGNL-HSIFAILURE
5.580E-01
HSI Degradation
HSI-XHE
External

Figure 22. The RTS FT including the HSI failure.

Table 20. The cutsets for the RTS FTs before and after adding the HSI failure.

Cutsets Ranking	RTS Before Adding the HSI Failure	RTS After Adding the HSI Failure
1	RPS-ROD-CF-RCCAS	RPS-ROD-CF-RCCAS
2	LC-LP-SF-CCF-TA, RPS-XHE-XE-SIGNL	LC-LP-SF-CCF-TA, RPS-XHE-XE-SIGNL
3	LC-BP-UCA-A-CCF, RPS-XHE-XE-SIGNL	LC-BP-UCA-A-CCF, RPS-XHE-XE-SIGNL
4	RPS-XHE-XE-SIGNL, RTB-UV-HD-CCF	RPS-XHE-XE-SIGNL, RTB-UV-HD-CCF
5	LP-HW-CCF, RPS-XHE-XE-SIGNL	IFD-APS-UIFA, LC-LP-SF-CCF-TA, RPS-XHE-XE-SIGNL-HSIFAILURE
6	LC-BP-HW-CCF, RPS-XHE-XE-SIGNL	LC-LP-SF-CCF-TA, QND-APS-UIFA, RPS-XHE-XE-SIGNL-HSIFAILURE

Table 21. Result of importance analysis on the RTS FT.

Ranking No.	Name	FV Value
1	RPS-ROD-CF-RCCAS	8.231e-1
2	LC-LP-SF-CCF-TA	1.214e-1
3	RPS-XHE-XE-SIGNAL	1.169e-1
4	RPS-XHE-XE-SIGNAL-HSIFAILURE	6.005e-2
5	LC-BP-UCA-A-CCF	3.074e-2
6	RTB-UV-HD-CCF	1.815e-2
7	IFD-APS-UIFA	1.547e-2
8	QND-APS-UIFA	1.547e-2
9	LP-HW-CCF	4.079e-3
10	IFD-APS-H	3.260e-3

4.5 Discussion

This study proposed a method of specifically evaluating HSIs for DI&C systems. A FT representing operator errors under normal and abnormal HSI software and hardware performance is developed using RESHA and calculated using ORCAS and IDHEAS-ECA method.

Degraded HSI conditions affecting the human action of initiating a manual reactor trip was analyzed in the case study. The human errors under different HSI software and hardware performance were respectively analyzed and quantified using the IDHEAS-ECA method. The outputs were integrated into the RTS FT of a generic PWR PRA model. The probability of top events, cutsets, and importance analysis results were discussed in this study. This method would be useful to provide an integrated and comprehensive evaluation of HSIs deployed in NPPs. Ongoing research will focus on further methodological improvements and practical demonstrations.

5. INVESTIGATING THE USE OF DYNAMIC PROBABILISTIC RISK ASSESSMENT METHODOLOGIES FOR ANALYZING DI&C SYSTEM COMMON CAUSE FAILURES

This section investigates using a dynamic PRA approach to evaluate the impacts of software CCFs on plant safety. The structure of ESFAS under consideration is described in Section 5.2. The implementation details for the analysis are presented in Section 5.3. The results of the analysis are discussed in Section 5.4, followed by a summary of the study's findings in Section 5.5.

5.1. Introduction

The implementation of DI&C systems in NPPs is increasingly replacing conventional analog systems due to their notable advantages. Digital systems offer enhanced plant safety and reliability by improving hardware reliability, failure detection capability, accuracy, and computational capacity. However, despite these benefits, there are currently no universally accepted methods for reliability analysis of digital systems in PRA [43]. While the traditional ET/FT approach is still employed for reliability modeling, it faces concerns regarding its ability to adequately account for dynamic interactions among system components. This limitation may result in the failure to identify potentially significant dependencies among failure events or properly quantify their likelihood [15].

Studies suggest that the traditional ET/FT approach is inadequate for ideal utilization in NPP reliability/safety assessments, particularly due to its inability to model time-dependent hardware/software/firmware/process interactions [15]. Furthermore, the traditional ET/FT approach encounters challenges in modeling changes in accident progression based on variations in the plant state, representing repair actions, and addressing software aging issues [44].

Software aging poses a significant risk to safety-critical systems in NPPs, such as DI&C systems, leading to performance degradation rather than immediate failure [45]. Although no physical failure occurs in the software, it has been observed that software systems "age" due to error accumulation or depletion of operating system resources during prolonged execution time. Software failure rates also increase significantly with increased usage [46]. However, the estimation of software failure rates due to aging remains uncertain, as there is no consensus on a specific model, and various factors contribute to the issue. It is suggested that failure rates gradually increase, and artificially increased failure rates are assumed for parametric studies [47]. To mitigate performance degradation caused by software aging, software rejuvenation emerges as an important method [48]. Determining the optimal schedule for software rejuvenation is crucial for software reliability analysis, but the traditional ET/FT approach lacks explicit representation of time, hindering its consideration [45].

To overcome the limitations of traditional PRA, dynamic PRA (DPRA) methods have been developed to provide a more accurate representation of the probabilistic evolution of I&C systems over time by accounting for complex interactions. Evidence in the literature suggests that the traditional method tends to overestimate top event frequencies when compared to DPRA methods [49]. DPRA methods also allow the evaluation of safety impacts with higher resolution, considering the timing and sequencing of events without relying on overly conservative modeling assumptions or success criteria [9]. The inclusion of event timing as a new degree of freedom in the issue space with DPRA enables the evaluation of scenarios that were previously overlooked, leading to insights into hidden risks.

The dynamic event tree (DET) approach is a DPRA method that aligns well with the existing PRA structure and is similar to the traditional ET approach. However, unlike ETs, where the sequence of system responses following initiating events is predetermined by the analyst, DETs determine both the timing and sequence of system responses using a time-dependent model of system evolution and branching conditions selected by the analyst. DETs provide more comprehensive and systematic coverage of possible event scenarios compared to the traditional ET approach, facilitating the consideration of hardware/process/software/human interactions in a phenomenologically and stochastically consistent manner [50].

In line with previous research [15], [43], [44], this study explores the feasibility of employing DPRA methodologies in DI&C system reliability analysis. The DET approach is used as a case study to investigate the impact of software CCFs and software aging in a representative ESFAS based on the APR 1400 design.

5.2. System Description

Figure 9 shows the functional logic of the example ESFAS. This four-division digital ESFAS includes the portion of PPS that activates the ESFs and their component control system (CCS) [7]. The safety I&Cs of the ESF systems consist of the electrical and mechanical devices and sensor circuitry to the actuation-device input terminals that are involved in generating signals that actuate the required ESF systems. The ESFAS portion of the PPS includes the following functions: (1) bistable logic (BL), (2) LCL, (3) ESFAS initiation, and (4) the testing function. After receiving ESFAS initiation signals from the PPS, MCR operator console, or remote shutdown room (RSR) shutdown console, the ESF-CCS generates ESF actuation signals to ESF component interface modules (CIMs), which transmit signals to the final actuated device [7].

The ESFAS portion of PPS consists of four divisions, as indicated by Divisions A through D in Figure 9. Each PPS division is located in an I&C equipment room and contains both an input and output module, two BPs, two racks for the LCL functions, and other hardware for the interface with other PPS divisions. The redundant BPs generate ESF actuation signals to the LCL racks in the four redundant divisions if the process values exceed their respective setpoints. Each LCL rack contains two logic processors (LPs); the initiation signals are provided to the ESF-CCS, which consists of four divisions of group controller (GC) and loop controller (LC) cabinets. Each GC supports component control and provides ESF actuation signals to the LC. Each LC has component control logic and multiplexing function. Each ESF-CCS GC performs selective two-out-of-four coincidence logic. The output of the ESF-CCS GC is transmitted to the component control logic in the LC. The logic produces digital output (DO) signals to control the component through the CIM, which performs signal prioritization [7].

5.3. Case Study

The DET was assessed by simulating failures in all divisions of the ESFAS and the timing of activation for each division using simplified case studies and failure injection. Figure 23 illustrates an example DET. Failures were intentionally introduced into the system when ESFAS was required during the progression of accidents. The study encompassed two distinct cases to explore software CCFs and software aging effects.

In Case Study I, the DET was employed to determine the probabilities of ESFAS failures with different sequences of failures, with the aim of identifying worst-case scenarios. On the other hand, Case Study II focused on evaluating the consequences of increasing ESFAS failure probabilities due to software aging. It is essential to emphasize that all the analyses performed in the demonstration of the DET methodology and the numerical results obtained do not necessarily represent the actual implementation of ESFAS.

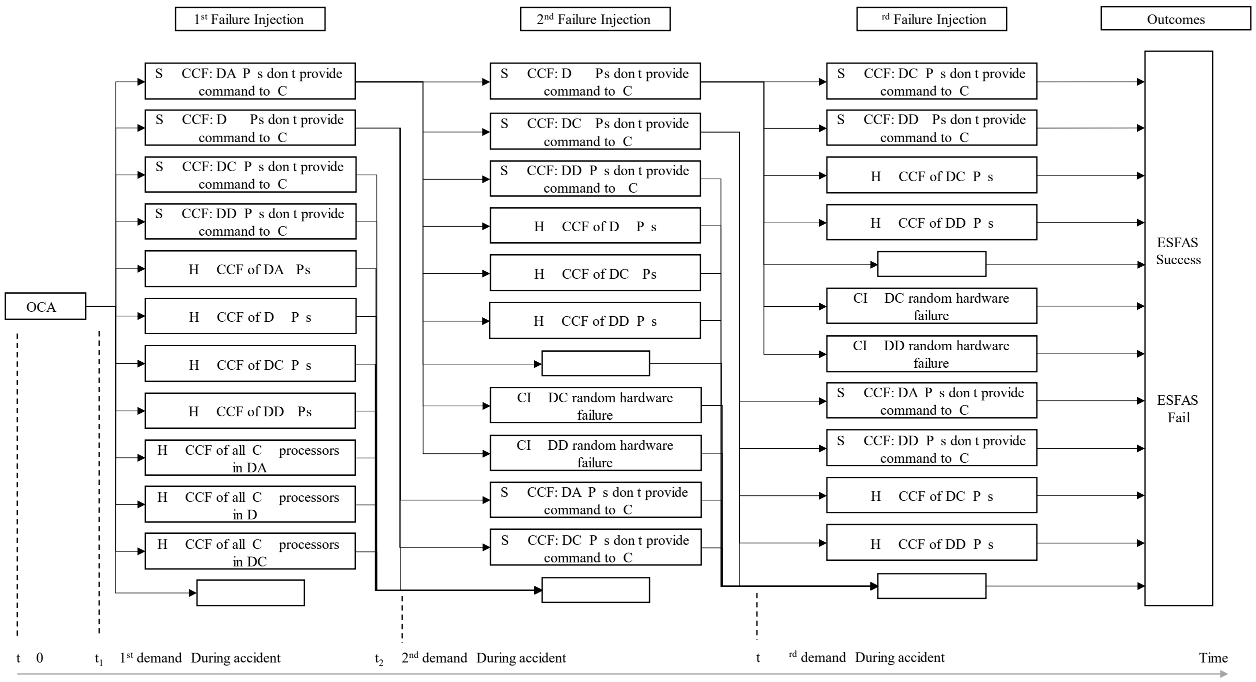

1st Failure Injection
2nd Failure Injection
rd Failure Injection
Outcomes
OCA
S CCF: DA P s don t provide command to C
S CCF: D Ps don t provide command to C
S CCF: DC P s don t provide command to C
S CCF: DD P s don t provide command to C
H CCF of DA Ps
H CCF of D P s
H CCF of DC P s
H CCF of DD Ps
H CCF of all C processors in DA
H CCF of all C processors in D
H CCF of all C processors in DC
CI DC random hardware failure
CI DD random hardware failure
ESFAS Success
ESFAS Fail
t 0
t1 1st demand During accident
t2 2nd demand During accident
t rd demand During accident
Time

Figure 23. Example DET.

In both Case Study I and Case Study II, several assumptions have been made to simplify the analysis. These common assumptions are as follows:

- Medium loss-of-coolant accident (MLOCA) is considered as the initiating event
- Failures can only occur when the ESFAS is needed during the progression of the accident
- All four ESFAS divisions are initially active and operational before the accident
- Failures of the MCR, reactor safety system (RSR) and division permissive system are not taken into account in the DET
- There is no maintenance or replacement of failed components during the analysis
- Following a software failure in a division, a hardware failure cannot occur in the same division in the next time step and vice versa
- The branch probability cutoff value is set to 1, allowing all possible accident scenarios to be considered in the DET
- The ESFAS failure probability cutoff value is set to 10–18 per demand, avoiding unnecessary expansion of the DETs.

In Case Study I, the application of the DET method resulted in obtaining ESFAS failure probabilities. The analysis assumed that all ESFAS divisions are in the same condition, and failure rates are assumed to be constant.

In Case Study II, the analysis assumed that all ESFAS divisions are in different conditions, with increased failure rates due to software aging. Four divisions of the ESFAS were replaced or maintained at different times. The parametric study in Case II was used to analyze the effects of increasing failure rates, which were represented in Cases IIa, IIb, and IIc. The percentage increase for each case is presented in Table 22.

Table 22. Increasing percentages of failure rates of ESFAS.

# Division	Case IIa	Case IIb	Case IIc
Division A failure rates	10% ↑	20% ↑	40% ↑
Division failure rates	40% ↑	55% ↑	80% ↑
Division C failure rates	20% ↑	5% ↑	60% ↑
Division D failure rates	0% ↑	40% ↑	50% ↑

The branching conditions and failure rates presented in In the DET analysis, only scenarios with three faults were considered. Scenarios involving four or more faults were not considered due to their exceptionally low likelihood of occurrence, resulting in outcome likelihood values on the order of 10–18 per demand or even lower.

Table 23 were intentionally created for the purpose of methodology development and demonstration; they are not actual values used for real-world analysis. These values were used in the DET analysis to test and showcase the methodology.

In the DET analysis, only scenarios with three faults were considered. Scenarios involving four or more faults were not considered due to their exceptionally low likelihood of occurrence, resulting in outcome likelihood values on the order of 10–18 per demand or even lower.

Table 23. Branching conditions and failure rates (per demand) of for Cases I and II.

ESFAS Modules	Branching Conditions	Failure Rate (Per Demand)			
		Case I	Case IIa	Case IIb	Case IIc
BP	SW CCF: DA BPs do not provide command to LCL	1.062E-04	1.168E-04	1.274E-04	1.487E-04
	SW CCF: DB BPs do not provide command to LCL	1.062E-04	1.487E-04	1.646E-04	1.912E-04
	SW CCF: DC BPs do not provide command to LCL	1.062E-04	1.274E-04	1.434E-04	1.699E-04
	SW CCF: DD BPs do not provide command to LCL	1.062E-04	1.381E-04	1.487E-04	1.593E-04
	HW CCF of DA BPs	5.943E-06	6.537E-06	7.132E-06	8.320E-06
	HW CCF of DB BPs	5.943E-06	8.320E-06	9.212E-06	1.070E-05
	HW CCF of DC BPs	5.943E-06	7.132E-06	8.023E-06	9.509E-06
	HW CCF of DD BPs	5.943E-06	7.726E-06	8.320E-06	8.915E-06
LCL	HW CCF of all LCL processors in DA	7.647E-06	8.412E-06	9.176E-06	1.071E-05
	HW CCF of all LCL processors in DB	7.647E-06	1.071E-05	1.185E-05	1.376E-05
	HW CCF of all LCL processors in DC	7.647E-06	9.176E-06	1.032E-05	1.224E-05
	HW CCF of all LCL processors in DD	7.647E-06	9.941E-06	1.071E-05	1.147E-05
ESF-CCS	SW CCF: DA GC processors fail to provide signal	1.062E-04	1.168E-04	1.274E-04	1.487E-04
	SW CCF: DB GC processors fail to provide signal	1.062E-04	1.487E-04	1.646E-04	1.912E-04
	SW CCF: DC GC processors fail to provide signal	1.062E-04	1.274E-04	1.434E-04	1.699E-04

	SW CCF: DD GC processors fail to provide signal	1.062E-04	1.381E-04	1.487E-04	1.593E-04
	HW CCF on GC 1-2 in DA	5.97E-06	6.57E-06	7.17E-06	8.36E-06
	HW CCF on GC 1-2 in DB	5.97E-06	8.36E-06	9.26E-06	1.08E-05
	HW CCF on GC 1-2 in DC	5.97E-06	7.17E-06	8.06E-06	9.56E-06
	HW CCF on GC 1-2 in DD	5.97E-06	7.76E-06	8.36E-06	8.96E-06
CIM	CIM DA random hardware failure	4.00E-05	4.40E-05	4.80E-05	5.60E-05
	CIM DB random hardware failure	4.00E-05	5.60E-05	6.20E-05	7.20E-05
	CIM DC random hardware failure	4.00E-05	4.80E-05	5.40E-05	6.40E-05
	CIM DD random hardware failure	4.00E-05	5.20E-05	5.60E-05	6.00E-05
BP: Bistable Processors, **LCL:** Local Coincidence Logic, **Dx**: Division X (A, B, C, D), **ESF-CCS**: Engineered Safety Features-Component Control System, **ESF-CIM**: Engineered Safety Features-Component Interface Module, **SW CCF**: Software Common Cause Failure, **HW CCF**: Hardware Common Cause Failure, **GC**: Group Controller					

5.4. Results

Table 24 and Table 25 present the best and worst results obtained from Case Study I accident progression scenarios using the DET methodology. These tables show the failure probability per demand for each scenario, along with the sequences of failures and their respective probabilities. In total, 11,088 different accident scenarios were evaluated, and 432 of them resulted in the ESFAS failing to transmit a signal to activate the emergency safety systems.

From the results in Table 24, it can be observed that the system performs better in handling hardware CCFs without signal transmission failures. All of the best scenarios, labeled as SN 1-5 in Table 24, consist of hardware CCFs. However, software CCFs are of significant importance for the system, as most of the worst-case scenarios in Table 25 include at least one software CCF. The top three worst cases, SN 1–3 in Table 25, comprise almost all software CCFs.

Moving on to Case Study II, a total of 33,264 different accident scenarios were evaluated, and 1,296 of them resulted in the ESFAS failing to transmit a signal to activate the emergency safety systems. Table 26 presents the results obtained from Case Study II, focusing on the top five worst-case scenarios. These scenarios explore the impact of increasing the ESFAS failure probabilities due to software aging.

It is important to note that the data in these tables were obtained for the purpose of the study's demonstration and may not necessarily reflect actual ESFAS implementation or real-world conditions. The analysis provides valuable insights into the system's performance under various failure scenarios, especially concerning hardware and software CCFs and their respective effects on the ESFAS.

Table 24. The best five accident progression scenarios of the ESFAS failure in Case Study I.

Scenario Number (SN)	1st Failure Injection	2nd Failure Injection	3rd Failure Injection	Failure Probability (Per Demand)
1	HW CCF of DA BPs	HW CCF of DB BPs	HW CCF on GC 1-2 in DA	2.109E-16
2	HW CCF on GC 1-2 in DA	HW CCF on GC 1-2 in DB	HW CCF of DA BPs	2.118E-16
3	HW CCF of DC BPs	HW CCF of DD BPs	HW CCF of all LCL processors in DA	2.701E-16

4	HW CCF of DA BPs	HW CCF of all LCL processors in DB	HW CCF on GC 1-2 in DC	2.713E-16
5	HW CCF on GC 1-2 in DB	HW CCF on GC 1-2 in DC	HW CCF of all LCL processors in DD	2.725E-16
BP: Bistable Processors, **LCL:** Local Coincidence Logic, **Dx**: Division X (A, B, C, D), **ESF-CCS**: Engineered Safety Features-Component Control System, **ESF-CIM**: Engineered Safety Features-Component Interface Module, **SW CCF**: Software Common Cause Failure, **HW CCF**: Hardware Common Cause Failure, **GC**: Group Controller				

Table 25. The worst five accident progression scenarios of the ESFAS failure in Case Study I.

Scenario Number (SN)	1st Failure Injection	2nd Failure Injection	3rd Failure Injection	Failure Probability (Per Demand)
1	SW CCF: DB BPs do not provide command to LCL	SW CCF: DC BPs do not provide command to LCL	SW CCF: DD BPs do not provide command to LCL	1.198E-12
2	SW CCF: DB GC processors fail to provide signal	SW CCF: DC GC processors fail to provide signal	SW CCF: DD GC processors fail to provide signal	1.191E-12
3	SW CCF: DB GC processors fail to provide signal	SW CCF: DD GC processors fail to provide signal	HW CCF on GC 1-2 in DC	6.708E-14
4	HW CCF of DC BPs	HW CCF of DD BPs	SW CCF: DB BPs do not provide command to LCL	6.703E-14
5	CIM DB random hardware failure	CIM DC random hardware failure	CIM DD random hardware failure	6.400E-14
BP: Bistable Processors, **LCL:** Local Coincidence Logic, **Dx**: Division X (A, B, C, D), **ESF-CCS**: Engineered Safety Features-Component Control System, **ESF-CIM**: Engineered Safety Features-Component Interface Module, **SW CCF**: Software Common Cause Failure, **HW CCF**: Hardware Common Cause Failure, **GC**: Group Controller				

These findings highlight the importance of considering software aging as a critical factor in the reliability analysis of DI&C systems like ESFAS. As software ages, its failure rate increases, leading to potential performance degradation and higher probabilities of system failure, which could impact the safety and reliability of NPPs. Therefore, determining the optimal schedule for software rejuvenation and addressing software aging in DI&C reliability modeling are crucial aspects that must be thoroughly addressed for ensuring the safe and reliable operation of NPPs.

Table 26 provides insight into the impact of software aging on the ESFAS failure probabilities by presenting the results of the worst five scenarios. The data clearly demonstrate that software aging leads to a substantial increase in the ESFAS failure probabilities.

When comparing the results of Case IIa, IIb, and IIc (as defined in Table 22) to the results obtained from Case I, it becomes evident that the probability of error significantly rises with increasing software aging. Specifically, the ESFAS failure probabilities increase by 118.9% for Case IIa, 193.7% for Case IIb, and 333.2% for Case IIc, respectively, when compared to the probabilities observed in Case I.

These findings highlight the importance of considering software aging as a critical factor in the reliability analysis of DI&C systems like ESFAS. As software ages, its failure rate increases, leading to potential performance degradation and higher probabilities of system failure, which could impact the safety and reliability of NPPs. Therefore, determining the optimal schedule for software rejuvenation and addressing software aging in DI&C reliability modeling are crucial aspects that must be thoroughly addressed for ensuring the safe and reliable operation of NPPs.

Table 26. The worst five accident progression scenarios of the ESFAS failure in Case Study II.

Failure Injection Order	Branching Conditions	Failure Probability (per demand)			
		Case I	Case IIa	Case IIb	Case IIc
1st	SW CCF: DB BPs do not provide command to LCL	1.198E-12	2.616E-12	3.510E-12	5.175E-12
2nd	SW CCF: DC BPs do not provide command to LCL				
3rd	SW CCF: DD BPs do not provide command to LCL				
1st	SW CCF: DB GC processors fail to provide signal	1.191E-12	2.616E-12	3.510E-12	5.175E-12
2nd	SW CCF: DC GC processors fail to provide signal				
3rd	SW CCF: DD GC processors fail to provide signal				
1st	SW CCF: DB GC processors fail to provide signal	6.708E-14	1.472E-13	1.973E-13	2.912E-13
2nd	SW CCF: DD GC processors fail to provide signal				
3rd	HW CCF on GC 1-2 in DC				
1st	HW CCF of DC BPs	6.703E-14	8.194E-15	1.099E-14	1.621E-14
2nd	HW CCF of DD BPs				
3rd	SW CCF: DB BPs do not provide command to LCL				
1st	CIM DB random hardware failure	6.400E-14	1.398E-13	1.875E-13	2.765E-13
2nd	CIM DC random hardware failure				
3rd	CIM DD random hardware failure				

BP: Bistable Processors, **LCL:** Local Coincidence Logic, **Dx**: Division X (A, B, C, D), **ESF-CCS**: Engineered Safety Features-Component Control System, **ESF CIM**: Engineered Safety Features-Component Interface Module, **SW CCF**: Software Common Cause Failure, **HW CCF**: Hardware Common Cause Failure, **GC**: Group Controller

5.5. Discussion

This study demonstrates the application of DETs to estimate failure likelihoods for DI&C system reliability assessment, specifically focusing on the effect of CCFs and software aging through two simplified case studies.

In Case Study I, the analysis assumed that failure probabilities used in the DET methodology did not change over time, and all subsystem components were new without any maintenance or replacement. The analysis considered an MLOCA as the initiating event, and failures were injected at times when the

ESFAS was required during accident progression. Case Study I revealed that the system was better equipped to handle hardware CCFs without transmitting signal failures. However, software CCFs had a significant impact, leading to several worst-case scenarios.

In Case Study II, the analysis incorporated increasing failure probabilities over time due to software aging, assuming that all four ESFAS divisions were in different conditions. The results demonstrated that software aging, when not addressed with proper software rejuvenation, significantly impacted the reliability of DI&C systems.

The DET approach offers a closer representation of reality compared to traditional PRA methods and can be integrated into the traditional ET/FT analysis of NPPs. This study highlights that the DET approach is a valuable tool to uncover hidden risks in systems with existing PRA structures and provides valuable insights for understanding and mitigating the impact of software aging on DI&C system reliability.

Overall, the research emphasizes the importance of considering software aging and implementing software rejuvenation strategies to maintain the safety and reliability of DI&C systems in NPPs. The DET methodology is instrumental in this process and enables a more comprehensive assessment of the system's behavior under dynamic conditions.

6. DYNAMIC MODEL AGNOSTIC RELIABILITY EVALUATION OF MACHINE-LEARNING MODELS INTEGRATED IN INSTRUMENTATION & CONTROL SYSTEMS

This section presents a novel method for evaluating the model agnostic reliability of potential ML models integrated in DI&C systems. The structure of this section is as follows: in Section 6.2, the theoretical background behind trustworthiness in artificial intelligence (AI) and ML systems is discussed, and the concept of relative reliability in model predictions is presented. In Section 6.3, the mathematical details within Laplacian distributed decay for reliability (LADDR) are discussed, and a method for parameter optimization is presented. In Section 6.4, a case study using LADDR within an AI/ML integrated instrumentation system is presented based on the Nearly Autonomous Management and Control (NAMAC) [51, 52] system. Finally, in Section 6.5, a brief discussion on the results and conclusion of this work is presented.

6.1. Introduction

Interests in integrating AI/ML into instrumentation and control systems has significantly grown in recent years. AI/ML systems have been used for enhanced plant diagnostics [51], automated scheduling of maintenance tasks [53], autonomous control [52], and numerous other applications. Typically for these AI/ML systems, a training dataset is used to define the function of the model by learning a specific correlation between input and target parameters. Unlike conventional software programs, where the function can be described succinctly by the implemented algorithm, the function achieved through training is governed solely by the incoherent multiplication of neuron weights and biases. As the weights and biases have no tangible meaning, this leads to a problem where the functional correctness of the model cannot be comprehensively interpreted and verified by users or stakeholders of the system. This issue is not trivial as there are numerous examples of AI/ML systems failing due to various hidden root causes and failure mechanisms. Notably they include such as regressional inconsistencies [54], inherent distributional rigidness [55], metric optimization failures [56], or unintended adversarial examples [57]. While verification and validation methodologies for AI/ML model development has grown significantly, these intelligent systems still experience significant drops in performance when applied to real-world operational conditions [58]. Fundamentally, it is well known that AI/ML models excel at interpolation (or near-interpolation) tasks and experience performance degradation at extrapolation. One possible way to develop trustworthiness in AI/ML predictions is by assessing how well the real-world operational data matches the training data of the model. If the operational data and training data are similar (i.e., interpolation , we can assume there is some basis for trust in the model's predictions. This is analogous to inductive reasoning, where the samples in the training database act as specific evidence to support generalized conclusions made by the AI/ML model. However, determining when (or how far) a new input sample is considered an extrapolation task is a challenge and is the basis for out-of-distribution (OOD) [59] detection research.

Thus, the purpose of this work is to determine "how far" a new sample must be to be considered OOD and how this value can be used to support trust in ML integrated systems. We established a method to evaluate the reliability of model predictions relative to the training data. The fundamental assumption being that training data can be scrutinized for correctness and representative of the mission objective. In this work, training data is analogous to student education and serves as the knowledge base. An OOD detection method is applied on the operational data to determine reliability. The method, named LADDR, can be used to determine the relative reliability of model predictions in real-time and can be applied to any data-driven time-invariant (or memoryless) model.

6.2. Technical Background

6.2.1. Trustworthiness

The concept of trustworthiness is a human conceived aspect and is relative based on the contextual scenario of the application. Simply applying quantitative numbers and verifiable facts does not necessarily establish trustworthiness of a system by humans. Nuclear energy related deaths versus any other fossil fuel source can attest to this statement [60]. Therefore, building a foundation of trustworthiness in AI/ML integrated systems is a difficult and convoluted task. Currently, one of the most concise and up-to-date definitions of trustworthiness as a human aspect can be found in the National Institute of Standards and Technology's report titled "*Artificial Intelligence Risk Management Framework*" [61]. In their work, they describe seven broad characteristics of trustworthiness, each describing a socio-technical attribute related to the processes and activities of the design, development, and deployment of an AI/ML system.

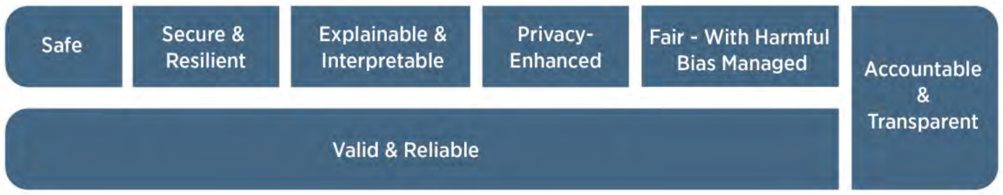

Safe
Secure & Resilient
Explainable & Interpretable
Privacy-Enhanced
Fair - With Harmful Bias Managed
Accountable & Transparent
Valid & Reliable

Figure 24. Characteristics of trustworthy AI systems [61].

In this work, we focus on the "Valid & Reliable" category, specifically, the reliability subcategory. In this subcategory, reliability is defined as "the ability of an item to perform as required, without failure, for a given time interval, under given conditions" [61]. Unfortunately, this definition is difficult to apply to AI/ML integrated control systems for a variety of reasons.

6.2.2. ML Reliability

The genesis of reliability was based on hardware systems and components that could fail due to mechanical wear and degradation. The fundamental assumption behind reliability is that the system has a limited life span due to environmental, chemical, physical, or other sources of degradation (i.e., corrosion) such that the time to functional failure (or inverse reliability) is probabilistic in nature. That is not the case for software systems, which include ML models. While hardware components (i.e., integrated circuits) that support the software can fail and be modeled with conventional hardware methods, software reliability is not the same case. Conventional methods for functional verification and validation in the development (and even pre-deployment) phase are not sufficient to guarantee reliability in the deployment phase [62]. While functional testing can improve the quality of the software, it cannot guarantee the failure-free operation of the software. Inconsistencies in design, errant assumptions, and incorrect implementations can lead to hidden latent defects (like landmines) in the source code. Thus, in [63, 64, 62], software failures are viewed as systematic failures of the design process, where existing statistical methods may be inadequate to determine reliability. If the same set of input conditions is consistently provided to software, the output will remain correct regardless of the number of tests performed. Unless a specific sequence and combination of unknowable input conditions is supplied, the system will never fail (see Ref. [65] for an example of unknowable conditions). Apparently "reliable" software can unexpectedly fail when a unique combination of unanticipated inputs is provided. Otherwise, the defect will remain hidden and have no visible impact on the overall system function. The concept of degradation over time on a software system also does not apply as software is virtual and without physical form. The generic hardware required to run software can be easily swapped making the theoretical life of software infinite until irrelevant. Therefore, the conventional definition of reliability in software is problematic.

Unfortunately, the current definition of reliability is used also for AI/ML systems and is difficult to apply in a similar manner. Take for instance, a non-reinforcement memoryless deep feedforward neural network (FNN), which in essence, approximates an unknown correlation through array multiplication of neuron layers. Typically, training samples ($x \in X$) are used to refine and develop the correlation through backpropagation of weights and biases of the model. During training, the model is also validated and verified on the withheld samples within X which is ideally sampled from the operational set (W) that describes the set of all inputs and outputs encountered in the realistic application of the system. Suppose during operation, the model is supplied with inputs, $x' \in W$, that mimic training samples from X such that $x_1' = x \in X$. As the model was validated on the training set, it is guaranteed it will perform as expected for any time interval and thus achieve perfect reliability. However, realistically, the training set will always be a subset of the operational set (i.e., $X \subset W$) due to various limitations in dataset construction. This suggests that the domain of x' is larger than X such that it is possible to draw samples that are in W but not in X as observed in Figure 25. Furthermore, NNs are known to have degraded performance at extrapolation tasks, or generally whenever $x_2{}' \notin X$.

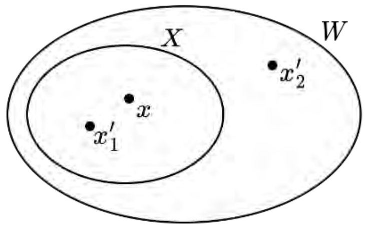
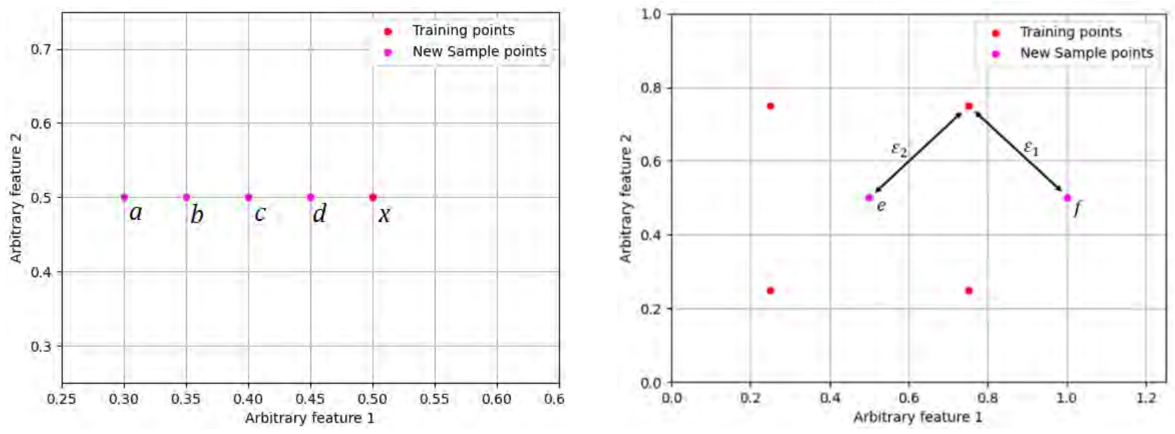


Figure 25. Input samples drawn from the training set, as well as the operational set.

A rudimentary definition of reliability in AI/ML can be described as the probability that new input samples are drawn exclusively from X and not within $W \backslash X$. This is the basis for OOD detection [59], which addresses the question: at what point is a new input sample within X and not within $W \backslash X$? To refine the definition of reliability, we use the following arbitrary two feature dataset as an example.



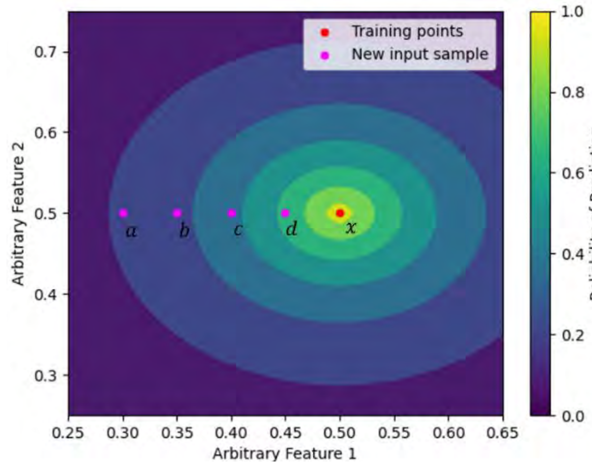
(a) (b)

Figure 26. (a) Single training point. (b) Multiple training points.

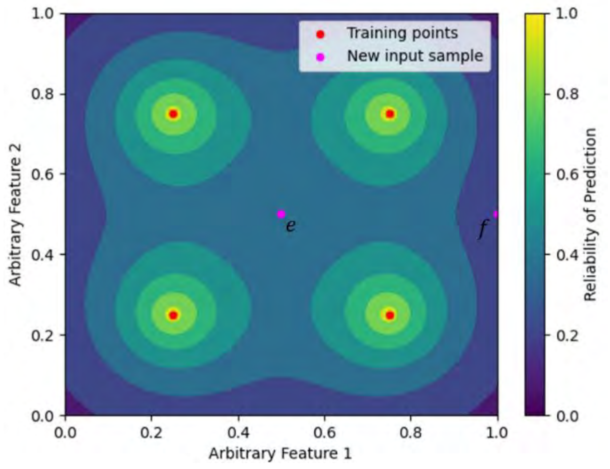
In Figure 26(a), the training set contains a single point, x, and four input sample points (i.e., a through d). At what distance, is point a considered to be too far from the x and thus OOD? Conversely, point d is the closest to x; therefore, is it the most in-distribution (ID)? Similarly, in Figure 26(b), point e is internally far from training points while point f is externally far. The Euclidean and Mahalanobis distance [66] are two distance functions to calculated separation (i.e., ε_1 and ε_2), but it is difficult (and subjective) to determine a consistent decision boundary for when ε is large enough to be considered

OOD. It also overlooks the fact that datapoints may be internally far versus externally far or in densely versus sparsely populated clusters.

In this work, we propose the concept of relative reliability of ML predictions as to whether sample points can be represented by a limited set of local training points. Mathematically, if $x_1' = x \in X$, then the predictions are considered reliable (i.e., $R = 1$) as they emulate training data. If $x_1' \neq x$ but $x_1' \in X$, then the predictions are somewhat reliable (i.e., $0 < R < 1$) based on the function, $L(\vec{x}_1', \vec{x}, q)$, between x_1' and the $x \in X$. The function $L(\vec{x}_1', \vec{x}, q)$ is defined by a localized Laplacian distribution decay distribution applied to all points $x \in X$. For the example shown in Figure 26(a), the reliability of a prediction at any point can be mapped as the contour plot shown in Figure 27(a). As input samples a through d move further away from the training point x, the reliability of the prediction decays exponentially. In Figure 27(b), the superposition of Laplacian distributions over all training points illustrates how internal samples (i.e., point e) may have higher reliability than the external sample (i.e., point f) but may still be OOD relative to the training data (i.e., low reliability). The maps generated in Figure 27 are known as reliability maps. Note that this method of reliability determination can also extend to include the target of each training sample as well. This enables the capability of determining whether a prediction's outcome is reliable relative to the training data. The method proposed is called LADDR.



(a)



(b)

Figure 27. Relative reliability of model prediction over (a) a single point and (b) multiple points.

6.3. Methodology

6.3.1. Formulation of LADDR

LADDR is implemented with the normalized multivariate Laplacian distribution from [67]; see Equation (24).

$$L(\vec{x}', \vec{x}, V) = \frac{1}{2\alpha} \exp\left(-\frac{|x - \mu|}{\alpha}\right) = \exp\left(-2D(\vec{x}', \vec{x})\right) \tag{24}$$

$$D(\vec{x}', \vec{x}) = \sqrt{(\vec{x}' - \vec{x})^T V^{-1} (\vec{x}' - \vec{x})} \tag{25}$$

Here, $\vec{x}'$ is the new sample vector, $\vec{x}$ is the nearest training sample, $D(\vec{x}', \vec{x}, V)$ is the Mahalanobis distance, $L(\vec{x}', \vec{x}, V)$ is the relative reliability from $\vec{x}'$ to $\vec{x}$, and $V$is the covariance structure matrix for distribution spread in n-dimensions, as in Equation (26). For the desired behavior where $L(\vec{x}', \vec{x}, V) = 1$ when $x_1' = x \in X$, the scale factor is set to $\alpha = 0.5$. V is a diagonal matrix where β_n represents the decay rate in the axis of each feature. It is not the covariance of the dataset. For a single training sample, $\tilde{L}(\vec{x}', \vec{x}, V)$ will produce a number between 0 and 1, where 1 represents perfect alignment of the training and the input sample.

$$V = \begin{pmatrix} \beta_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \beta_n \end{pmatrix} \tag{26}$$

A superposition of Equation (24) was used to generate the relative reliability maps seen in Figure 27 and is the relative reliability of model predictions to the training data knowledge base.

6.3.2. Optimization of LADDR Parameters

Without a doubt, the hyper parameters within LADDR will influence the outcome of the model. Therefore, in this section, we provide guidance and analytical metrics to optimize LADDR. First, to facilitate optimization, we introduce the concept of extrapolation diameter, γ, which represents how far a single training point can be "stretched" to cover interpolation tasks. The extrapolation diameter is defined per input feature as the width at $L(\cdot) = 20\%$. In Figure 28(a), for instance, a training point for feature 1 is located at $x = 0.5$ with a specified $\gamma = 0.36$. The extrapolation radius (or half the diameter) $\gamma_{1/2} = 0.18$ suggests that samples less than $x = 0.68$ or greater than $x = 0.32$ are a fifth as reliable as closer samples to $x = 0.5$. For users, this allows them to specify the literal distance when data points become OOD. This also allows hyperparameters to be optimized relative to the extrapolation diameter which is more interpretable than the covariance structure matrix. By specifying the extrapolation diameter for each feature, the covariance structure matrix can be solved for using Equation (27). In Figure 28(b), the variance in arbitrary feature 2 is set to five times greater than arbitrary feature 1.

$$V = \frac{\vec{\gamma}_{\frac{1}{2}}^{T} \vec{\gamma}_{\frac{1}{2}}}{(-\ln(L(\cdot))^2} \tag{27}$$

Instances where $\beta_1 \neq \beta_2$ in V are evident when the importance of training features is not equivalent. Selection of β parameters is relative to the optimization goal and the usage context of the system. In Figure 29, a utilization framework is presented where LADDR is integrated with an arbitrary predictive ML model. Within this framework, the training data used to train the model serve as the knowledge base for LADDR. For a new sample, if LADDR determines the sample and subsequent prediction are likely to be ID, the prediction is accepted, and the system can proceed normally. Alternatively, if it is likely to be OOD, the prediction is rejected regardless of the actual correctness or capability of the model, and auxiliary functions are engaged.

If stakeholders are concerned about safety, LADDR may be configured where only samples that emulate training data are accepted, whereas all others are rejected. Conversely, LADDR can be configured to maximize model performance while only rejecting significantly different contextual scenarios. In this respect, LADDR acts as a supervisor, filtering out scenarios that are irrelevant to the ML model. As LADDR acts as a filter, it is expected the ML model's performance will degrade whenever LADDR rejects a correct prediction. However, if LADDR were to accept all predictions, it is likely an incorrect prediction is not filtered and is a peril to the operational safety of the system. These two perspectives are used to form two analytical metrics known as degradation (f_D) and peril (f_P) seen in Equations (28) and (29), respectively. These metrics are used to minimize what is most important to stakeholders of the ML system in the context of their application.

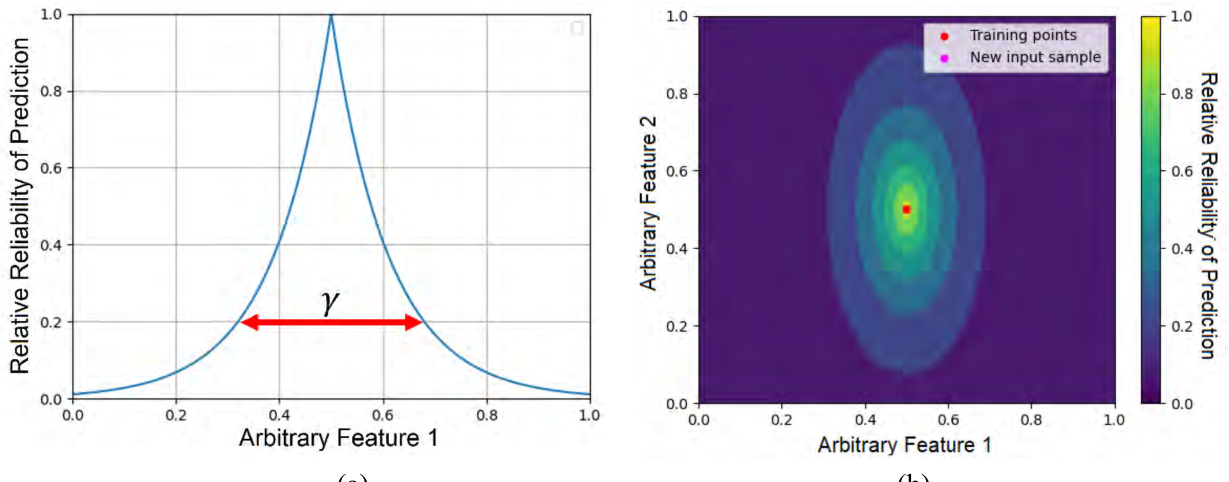


Figure 28. (a) Extrapolation diameter. (b) Elliptical distribution generated by $\beta_2 = 5\beta_1$ in V.

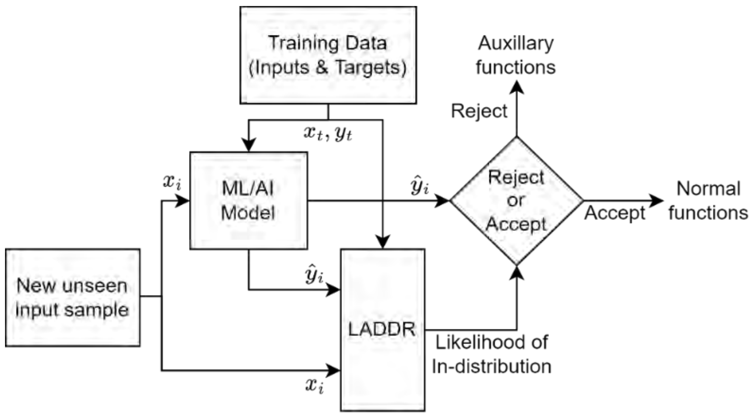


Figure 29. Basic framework of LADDR coupled with an ML-aded I&C system.

$$f_D = \frac{R_o}{R_o + A_o} \tag{28}$$

$$f_P = \frac{A_x}{A_x + A_o} \tag{29}$$

Here, R_o and A_o are the total number of rejected and accepted predictions that were correct while R_X and A_X are the total number of rejected and accepted predictions that were incorrect. The degradation metric, Equation (28), describes the ratio of correct predictions made by the model but were subsequently rejected over the total number of correct predictions that were rejected or accepted. A value of 0 suggests that all correct predictions made by the model were also accepted by LADDR while a value of 1 suggests that all correct predictions were rejected. This can occur if the extrapolation diameter specified for each feature in LADDR is too small such that only the immediate vicinity of the training data points is considered sufficient evidence to justify predictions. For stakeholders concerned about peak performance and less so about functional safety of the system, the extrapolation diameters can be chosen to minimize this parameter.

Alternatively, if the system is considered safety critical, the extrapolation diameters are chosen to minimize the peril metric, Equation (29). This metric describes the ratio of incorrect predictions accepted over the total number of predictions accepted by LADDR. A value of 0 is where all accepted predictions are correct while a value of 1 is where that all accepted predictions are incorrect. Correctness is determined relative to the specified acceptance criteria by stakeholders (e.g., $\pm 1\%$). Peril (f_P) grows as extrapolation diameters are increased and represent when limited data is used to make gross predictions.

However, importantly, if degradation is minimized, peril will grow, and vice versa, as the extrapolation diameters that minimize either are different. Therefore, we also introduce a third metric, ineptitude (f_f), which combines of the degradation and peril metrics; see Equation (30). The ineptitude metric describes the total number of rejected correct predictions and accepted incorrect predictions over the total number of predictions that are filtered. A value of 1 suggests that all predictions filtered are erroneous while a value of 0 suggests LADDR is perfectly proficient at accepting correct—but rejecting incorrect—predictions. This metric can be minimized if both performance and safety are desired by stakeholders.

$$f_f = \frac{R_\mathrm{o} + A_\mathrm{x}}{A_\mathrm{o} + R_\mathrm{x} + R_\mathrm{o} + A_\mathrm{x}} \tag{30}$$

6.4. Case Study

The LADDR framework presented in Figure 29 is applied to a diagnostic digital twin (DT) model [51] developed for the NAMAC [52] system. The DTs are deep FNN developed for partial loss-of-flow accident scenarios to predict the peak fuel centerline temperature (T_{FCL}). The datasets were gathered and simulated using a GOTHIC$^{\mathrm{TM}}$ model of the Experimental Breeder Reactor-II (EBR-II) [68]. In EBR-II, two separate primary sodium pumps, designated as P1 and P2, provide coolant flow through the core. In the postulated scenario, P1 partially loses pump rotational speed, thus decreasing the overall coolant flow through the core block. The scenario is monitored by the diagnostic DT.   ased on NA AC   's automated recommendations, the rotational speed of P2 is increased to compensate. The scope of the numerical demonstration can be represented by the time-dependent curve of the rotational speed of P1 Equation (31).

$$\omega_1(t) = \omega_0 \left(1 - \frac{1 - (\omega_1)_{end}}{T_1} t\right), \quad t_0 \le t \le t_0 + T_1 \tag{31}$$

Here, ω_0 is the nominal pump speed, T_1 is the ramp-down duration, $(\omega_1)_{end}$ is the normalized P1 end speed, and t_0 is the transient start time. By varying pump end speed and ramp-down duration, different profiles can be simulated. Core flow rate, plenum temperatures, and T_{FCL} are sampled for 2,000 time-steps per transient (Table 27). In column 1, the reference label is provided; column 2 shows the number of transients per scenario; columns 3–4 show the degree of degradation; and columns 5–6 show percentage of data used for training and forming LADDR knowledge base vs. used in testing. In Figure 30, the transient data is plotted. Each color represents a different transient from start to end. The set D2 was not used to train the DT and instead is used to gauge the performance of LADDR when an entirely different scenario is provided. The objective of the diagnostic DT is to predict T_{FCL} within $\pm 10^{\mathrm{o}}$C of the simulated temperature and serves as the acceptance criteria for predictions. As T_{FCL} is generally an unobservable variable due to obstructions within the fuel bundle, establishing the reliability of model predictions is critical to operational safety. For this case study, the upper plenum temperature and total core flow rate are used as inputs to train the DTs while T_{FCL} is used as the predictive target. The same three parameters are used as the knowledge base within LADDR. Finally, note that the post-training performance of the diagnostic DT in this case study is irrelevant. Model performance is intentionally poor to demonstrate the capability of the LADDR framework at rejecting and accepting predictions based on training data knowledge.

Table 27. Size of training and testing sets used within LADDR and for the FNN DTs.

Label	# of Episodes	T_1	$(w_1)_{end}$ (% of w_0)	% Used in Training	% Used in Testing
D1	1,024	467.81	51.6 – 100	10%	90%
D2	250	467.81	0 – 38.7	Not Used	100%

In Figure 31a, the reliability map is generated for Figure 30a where only the inputs features are considered for reliability. As the inputs are two-dimensional, the reliability map illustrates where there is a high degree of training evidence to support model predictions. In Figure 31b, the simulated T_{FCL} is also considered as evidence, and a two-dimensional slice of the three-dimensional reliability map for when $T_{FCL} = 0.877$ is shown. Figure 31b illustrates that even if input data is similar to training data, if the prediction is not correct, there is very little reliability in the outcome.

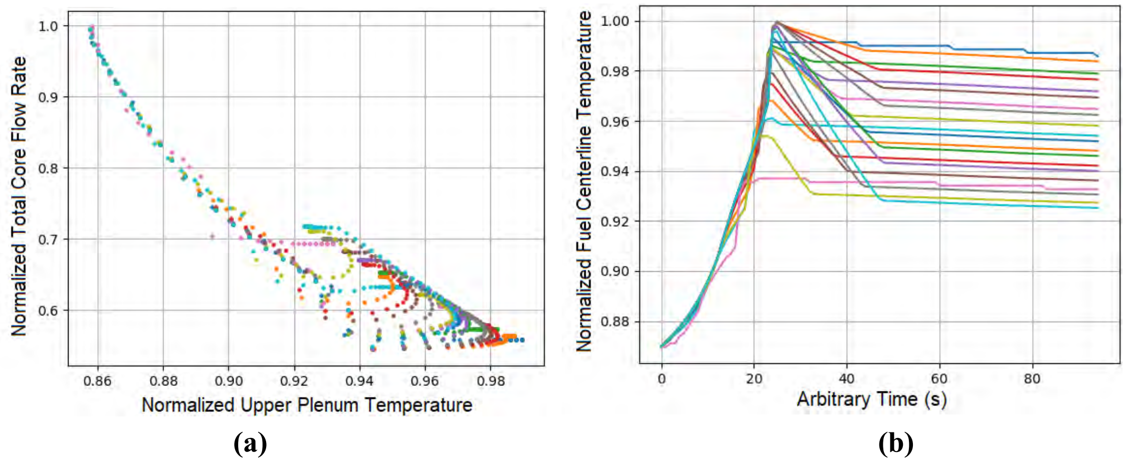


Figure 30. (a) Normalized input training data. (b) Normalized fuel centerline temperature transients.

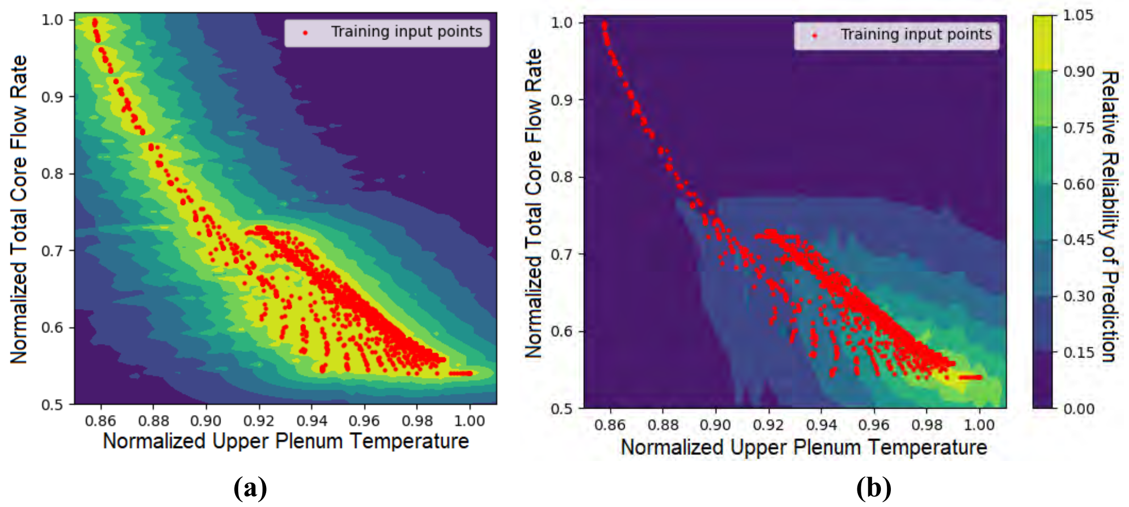


Figure 31. a) Reliability map where only the input features are considered for reliability. b) Reliability map where the simulated T_{FCL} is also considered as evidence given $T_{FCL} = 0.877$.

Table 28. Analytical metrics for extrapolation diameter optimization.

Minimized Metric	Optimized Normalized Extrapolation Diameters			Analytical Metrics		
	UP Temp.	Total Core Flow Rate	T_{FCL}	Peril	Degradation	Ineptitude
Peril	0.0254	0.0254	0.064	0	0.105	0.085

Degradation	0.0254	0.0254	0.113	0.116	0	0.105
Ineptitude	0.0254	0.0254	0.072	0.027	0.079	0.084

In Table 29, the total number of incorrect and correct rejections and acceptances are shown for the testing sets D1 and D2. Although the extrapolation diameters are minimized for the training set D1, LADDR can still perform well on set D2, filtering out a majority of unsafe predictions. The degradation is expected to be higher for set D2 as the scenarios are significantly different. However, as D2 is not part of A DDR’s knowledge base, the higher rejection rate is associated with lack of training data evidence to support model predictions even if they are within the $\pm 10^{\circ}$C acceptable margin.

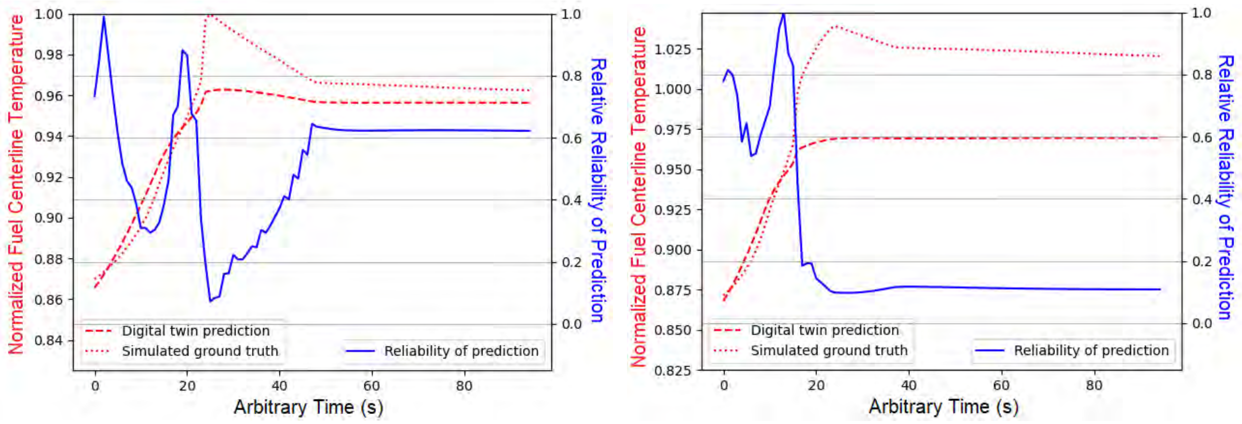


Figure 32. Prediction reliability (blue line) against model prediction and simulated truth (red dotted line).

Table 29. Summary of LADDR performance of the developed loss-of-flow transient datasets.

Dataset	Correct Accept	Correct Reject	Incorrect Accept	Incorrect Reject	Peril	Degradation	Ineptitude
D1	13,825	4,084	635	931	4.4%	6.3%	8.04%
D2	6,865	3,487	400	3,783	5.5%	35.5%	28.8%

6.5. Discussion

The preliminary results demonstrate the utility of LADDR at filtering out predictions that are unreliable relative to the training database. Three additional optimization metrics were also presented to allow optimization of LADDR parameters based on stakeholder concerns: (1) peril, which dictates the rate of incorrect predictions not filtered; (2) degradation, which dictates the percentage of performance drop as a result of LADDR acting as a filter function to the ML model; and (3) ineptitude, which is a culmination of the other two metrics. Currently, LADDR has only been tested on three-dimensional training sets. As most training sets can contain upward of 10 features, we plan to verify the capability of LADDR on higher dimensions and improve upon the optimization techniques for the extrapolation diameters. It is anticipated that LADDR can improve the trustworthiness of ML models by acting as a “supervisor” for predictions made by intelligent systems.

7. COLLABORATIVE EFFORTS WITH NUCLEAR INDUSTRY

This section briefly summarized the collaboration initiated by the PWROG and the LWRS program in FY 2023 to support a pilot risk evaluation of a safety-related DI&C system in use at a nuclear utility. Complete analysis and results have been documented in [14].

7.1. Purpose and Motivation

The purpose of this work is to demonstrate the capabilities of the methodologies developed by LWRS to assess software CCF on a pilot risk evaluation of a DI&C safety related system, and to address research topics identified by the PWROG. The PWROG identified several research topics relevant to software CCF, some of which are used to define the scope of this collaboration:

1. **Identification of Software Failures**: Existing software development and testing processes for DI&C systems are already rigorous. Few, if any, relevant software CCFs have been identified in operating nuclear power plant DI&C systems. Identifying relevant software failures is challenging, in part, due to the complex interaction of heterogeneous digital components developed under different operational assumptions by the manufacturer, which may lead to emergent hazards or failures. A more realistic method is desired to qualitatively identify relevant software failures including CCFs and their consequences.
2. **Quantification of Software Failures**: In addition to the challenge of identification of software failures that impact safety related functions of DI&C systems, the quantification of software failures is another matter of interest. Due to the relative novelty of software in safety-related DI&C systems and the lack of significant software related failures in the historical record, there is insufficient failure data and operational history to adequately support realistic failure estimates. Until sufficient experimental data is accumulated, a new evaluation technique is sought to more accurately quantify software failure.
3. **Addressing Conservative Assumptions in CCF modeling**: Limited software CCF data requires conservative assumptions for CCF model and estimation, an example would be assuming software CCF always fails the entire system. The traditional beta factor treatment of software CCF assumes that all redundant channels and trains of a safety-related DI&C system subject to a potential CCF will fail with a conditional probability of beta. Modeling software CCF with bounding assumptions (e.g., beta=1) negates the defenses anticipated from redundancy and diversity applications (e.g., redundancy of signals, channel separation within a DI&C system, deployment of watch dogs) and may overestimate the impact of software CCFs on the overall DI&C system risk. A detailed treatment of software CCF within DI&C systems should be employed by considering or reducing the limitations of bounding assumptions.

7.2. Technical Approach

The PWROG supplied the LWRS research team with data including design information and a PRA model from a pilot safety-related DI&C system. The models and capabilities of the LWRS-developed framework offer a pathway to addressing the research motives and thus represent a useful pathway for the PWROG to investigate. Each of the research motives identified above are addressed as follows:

1. The LWRS-developed framework employs a systems-based hazard analysis capable of considering systems interactions such as those that may be encountered and lead to emergent CCF. This method is called the RESHA. Guided by RESHA, the complex interactions can be directly incorporated as failure modes for FT analysis providing a basis for both qualitative and quantitative assessments.
2. The LWRS-developed framework employs two separate tools capable of addressing limited data. One method, called the A HA AS , considers the specifics of a software's development plans

and activities in a structured manner to addresses software failure quantification. The other method, called the ORCAS, provides a means of quantification by direct consideration of the software testing and development. Both methods have been developed based on experience with diverse industrial software development strategies, and there are known conservatisms in their applications to safety-related DI&C systems.

3. As an alternative to the traditional beta-factor method for CCF modeling, the LWRS-developed framework employs a hybrid model. The hybrid model allows for more complex investigations of CCF events such that the benefits of redundant configurations can be seen. In other words, the hybrid model does not necessarily limit its consideration of CCF events to those that completely wipe out all redundant defenses.

7.3. Summary

This work demonstrates risk assessment and quantification specifically directed toward software elements of that system. Hardware-based failures are outside of the scope of this report. The results of this analysis demonstrate the ability to calculate a software common cause failure value via these methods although improvements to the methodology have been identified. For example, the software CCF model can provide additional credit regarding a system's defenses against software CCFs and increased consideration of system complexity. These software defenses or features, also called subfactors, provide an indication of the strength of coupling mechanisms between redundant elements. The stronger the coupling, the more significant a software CCF. The pilot DI&C system was designed to defend against software CCF. However, a reduction of the source similarity subfactor as discussed in [14] would positively benefit the CCF defense and examination of improvements to the calculation of this subfactor and reduction of conservatisms within this subfactor is an ongoing research topic. The initial benefit from refinement of this subfactor is a reduction of the total system failure through train-specific information sources. Introducing sub-train independence (i.e., reducing sub-train source similarity), can also provide additional improvement for CCF defense.

Ultimately, this collaborative project provided the LWRS team with insights to improve and further develop the LWRS-developed framework to respond to the needs of the nuclear industry. Several paths have been defined to provide more realistic CCF estimates including research in activation pathway of software CCFs, applying Bayesian updating based on accumulated operational or testing data, improving CCF modeling methodology, and accounting for software defensive design features in the refinement of attributes that are used to define the software CCF modeling parameter.

Ultimately, this collaborative project provided the LWRS team with insights to improve and further develop the LWRS-developed framework to respond to the needs of the nuclear industry. This work represents an example of the successful partnership and research benefits that can be achieved through industry collaboration. It is a goal of the LWRS team to continue such collaborations and provide science- and technology-based solutions and insights that support the long-term operations of the existing nuclear fleet.

8. CONCLUSIONS AND FUTURE WORKS

8.1. Conclusions

This report outlines R&D focused on refining current methods on software CCF modeling and estimation and exploring additional innovative approaches to risk assessment of DI&C systems to enable a more comprehensive and complete assessment of various safety-related DI&C design architectures. This research is intended to enhance the robustness of the methodology for the risk assessment and design optimization of safety-critical DI&C systems.

Methodology of the proposed framework and methods (e.g., RESHA, BAHAMAS, ORCAS, and CCF modeling approach) has been refined and improved based on feedback from the collaboration with the industrial partners and the comments from the technical peer review. A modification to the CCF approach employed by the LWRS-developed framework to increase its capabilities for modeling CCFs of diverse DI&C systems. The CCF model was adjusted by the introduction of Q_{cc} and the defense factor, ϕ. Q_{cc} represents the common parts between software, including diverse software, and is therefore used to represent the theoretical commonality that can leads to CCF; ϕ represents the how well defended a CCCG is against CCF. This Q_{cc} term provides a direct model of the commonality between functionally redundant software elements, even diverse elements. This work proposed BAHAMAS to evaluate Q_{cc} given BAHAMAS has the capability to directly consider the commonality of components. A simple case study was shown to demonstrate the approach to modeling potential CCFs of diverse components. Future work will include methodology development and improvement in addition to demonstrations on more detailed and complex analyses.

Identifying and classifying coupling factors that contribute to software CCFs is a critical and challenging task, especially in multi-function control platforms applied across different systems. Even when distinct programming languages, algorithms, and methodologies are used, the potential for CCFs remains. Unpredictable failure modes and spurious operations can impact other systems, potentially leading to cascading effects. Despite efforts to ensure diversity, intricate interactions within software and between systems can result in unexpected CCFs. Specifying dependencies between coupling factors and sub-factors in the CCF scoring table can enhance the CCF modeling and estimation approach.

In addition to the work on CCFs, this research includes the development of new methodologies for HSI risk assessment, dynamic PRA, and model-agnostic reliability assessments of potential ML models integrated into DI&C systems. Detailed findings for each research area can be found in the respective discussion sections.

8.2. Future Works

In FY-24, this project will further develop and improve the proposed framework to better meet the needs of the industry in DI&C reliability and risk analysis, particularly, for (1) function-based risk assessment of multi-function DI&C systems, (2) methodology refinement and maturation to keep supporting the pilot project with PWROG on DI&C reliability analysis, (3) development of capabilities on evidence generation and evaluation to support DI&C safety assurance and design optimization, and (4) risk assessment of (semi-) autonomous DI&C systems.

Key activities in FY-24 include:

- Improve and further develop the current framework and methods for the function-based risk assessment of multi-function DI&C systems in collaboration with the industry.
- Refine the current methods to (1) keep supporting the need of DI&C reliability analysis from the industry; (2) align better with international standards and existing risk-informed approaches and guides.

- Develop capabilities on risk-informed evidence generation and evaluation to support DI&C safety assurance and design optimization with the industry and other research institutions.
- Develop novel approaches to inform risk management and design optimization of advanced (semi-) autonomous DI&C systems designed for existing LWR fleets.

9. 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] H. Bao, T. Shorthill, E. Chen, J. Park, S. Zhang, A. V. Jayakumar, C. Elks, N. Dinh, H. Ban, H. Zhang, E. Quinn and S. Lawrence, "An Integrated Framework for Risk Assessment of High Safety-significant Safety-related Digital Instrumentation and Control Systems in Nuclear Power Plants: Methodology and Demonstration," Idaho National Laboratory, Idaho Falls, ID, August 2022.

[5] H. Bao, S. Zhang, R. Youngblood, T. Shorthill, P. Pandit, E. Chen, J. Park, H. Ban, M. Diaconeasa, N. Dinh and S. Lawrence, "Risk Analysis of Various Design Architectures for High Safety-Significant Safety-Related Digital Instrumentation and Control Systems of Nuclear Power Plants During Accident Scenarios," Idaho National Laboratory, Idaho Falls, ID, November 2022.

[6] 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.

[7] 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.

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

[9] "HAZCADS: Hazards and Consequences Analysis for Digital Systems - Revision 1," EPRI, Palo Alto, CA, 2021.

[10] 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.

[11] E. Chen, H. Bao, T. Shorthill, C. Elks, A. Jayakumar and N. Dinh, "Application of Orthogonal Defect Classification for Software Reliability Analysis," in *16th Probabilistic Safety Assessment & Management conference (PSAM 16)*, Honolulu, HI, June 26 – July 1, 2022.

[12] S. Zhang, H. Bao, T. Shorthill, J. Park and E. Chen, "Sensitivity and Importance Analyses for Various Design Architectures for High Safety-Significant Safety-Related Digital Instrumentation and Control Systems of Nuclear Power Plants," in *NPIC&HMIT 2023 and PSA 2023 Co-Located Meetings*, Knoxville, TN, July 15 -21, 2023.

[13] H. Bao, T. Shorthill, E. Chen, S. Zhang and Lawrence Svetlana, "Summary of Technical Peer Review on the Risk Assessment Framework proposed in Report INL/RPT-22-68656 for Digital Instrumentation and Control Systems," Idaho National Laboratory, Idaho Falls, ID, March 2023.

[14] E. Chen and T. Shorthill, "Preliminary Demonstration of Risk Assessment Methodologies for Safety Significant Digital Instrumentation and Control System," Idaho National Laboratory, Idaho Falls, ID, March 2023.

[15] 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.

[16] U.S. NRC, "A Defense-In-Depth and Diversity Assessment of the RESAR-414 Integrated Protection System," U.S. Nuclear Regulatory Commission, Washington, DC, 1979.

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

[18] K. Salako and . S trigini, " he n Does "Diversity" in Development Reduce Common Failures? Insights from Probabilistic Modeling," *IEEE Transactions on Dependable and Secure Computing,* vol. 11 No. 2, pp. 193-206, 2014.

[19] J. C. Knight and N. G. Leveson, "An Experimental Evaluation of the Assumption of Independence in Multiversion Programming," *IEEE Transactions on Software Engineering,* vol. 12, no. 1, pp. 96-109, 1986.

[20] F. Huang, B. Liu, Y. Song and S. Keyal, "The links between human error diversity and software diversity: Implications for fault diversity seeking," *Science of computer Programming,* vol. 89, pp. 350-373, 2014.

[21] F. Huang and L. Strigini, "HEDF: A Method for Early Forecasting Software Defects Based on Human Error Mechanisms," *IEEE Access,* vol. 11, pp. 3626-3652, January 2023.

[22] 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.

[23] D. Kančev and . Č epin, "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.

[24] T. Shorthill, H. Bao, E. Chen and H. Ban, "An Application of a Modified Beta Factor Method for the Analysis of Software Common Cause Failures," in *Probabilistic Safety Assessement and Managment Conference (PSAM 16)*, Honolulu, Hawaii, 2022.

[25] 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.

[26] 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.

[27] S. Hauge, S. Håbrekke and M. A. Lundteigen, "Reliability Prediction Method for Safety Instrumented Systems – PDS Example collection, 2010 Edition," SINTEF Technology and Society , Trondheim, Norway, 2010.

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

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

[30] 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.

[31] R. T. Wood, R. Belles, M. S. Cetiner, D. E. Holcomb, K. Korsah, A. S. Loebl, A. S. Mays, M. D. Muhlheim, J. A. Mullens, W. P. Poore III, A. L. Qualls, T. L. Wilson, Jr. and M. E. Waterman, "Diversity Strategies for Nuclear Power Plant Instrumentation and Control Systems," U.S. NRC, Washington, D.C., 2008.

[32] R. Wood, M. Muhlhelm, L. Pullum, C. Smith, D. Holcomb and K. Korsah, "Update on Common-Cause Failure Experience and Mitigation Practices," Oak Ridge National Labrotary, Oak Ridge, TN, 2013.

[33] D. M. Chapin, J. B. Dugan, D. A. Brand, J. R. Curtiss, D. L. Damon, M. DeWalt, J. D. Gannon, R. L. Goble, D. J. Hill, P. E. Katz, N. G. Leveson, C. M. Mitchell, C. Rodriguez and J. D. White, Digital Instrumentation and Control Systems in Nuclear Power Plants: Safety and Reliability Issues, Washington, D.C. : National Academy Press, 1997.

[34] J. Park, A. 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.

[35] D. Gertman, H. Blackman, J. Marble, J. Byers and C. Smith, "The SPAR-H human reliability analysis method, NUREG/CR-6883," US Nuclear Regulatory Commission, 2005.

[36] "Development of human performance measures for human factors validation in the advanced MCR of APR-1400," *IEEE transactions on nuclear science,* vol. 54(6), pp. 2687-2700, 2007.

[37] W. E. Cummins, M. M. Corletti and T. L. Schulz, "Westinghouse AP1000 advanced passive plant," in *Proceedings of ICAPP*, 2003.

[38] R. C. Twilley, "EPR development-an evolutionary design process," *Nuclear News,* vol. 47(4), pp. 26-30, 2004.

[39] G. Parry, A. Beare, A. Spurgin and P. Moieni, "An approach to the analysis of operator actions in probabilistic risk assessment, EPRI Report TR-100259," 1992.

[40] E. Hollnagel, Cognitive reliability and error analysis method (CREAM), Elsevier, 1998.

[41] J. Xing, J. Chang and J. DeJesus, "Integrated Human Event Analysis System for Event and Condition Assessment (IDHEAS-ECA), RIL-2020-02," US Nuclear Regulatory Commission, 2020.

[42] KHNP, "Chapter 7: Instrumentation and Controls. Rev 3," US Nuclear Regulatory Comission, 2018.

[43] T. Aldemir, S. Guarro, D. Mandelli, J. Kirschenbaum, L. Mangan, P. Bucci, M. Yau, E. Ekici, D. Miller, X. Sun and a. S. Arndt, "Probabilistic risk assessment modeling of digital instrumentation and control systems using two dynamic methodologies," *Reliability Engineering and System Safety,* vol. 95, pp. 1011-1039, 2010.

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

[45] J. Zhao, K. Trivedi, Y. Wang and X. Chen, "Evaluation of software performance affected by aging," in *IEEE*, 2011.

[46] Y. Bao, X. Sun and K. S. Trivedi, "A Workload-Based Analysis of Software Aging, and Rejuvenation," in *IEEE*, 2005.

[47] S. Ballerini, L. Carnevalli, M. Paolieri, K. Tadano and F. Machida, "Software Rejuvenation Impacts on a Phased-Mission System for Mars Exploration," *IEEE,* 2013.

[48] T. Dohi, A. Avritzer and K. S. Trivedi, Handbook Of Software Aging And Rejuvenation: Fundamentals, Methods, Applications, And Future Directions, Singapore: World Scientific, 2020.

[49] D. Mandelli, A. Alfonsi, C. Wang, Z. Ma, C. Parisi, T. Aldemir, C. Smith and R. Youngblood, "Mutual Integration of Classical and Dynamic PRA," *Nuclear Technology,* vol. 207, pp. 363-375, 2020.

[50] T. Aldemir, "A Survey of Dynamic Methodologies for Probabilistic Safety Assessment of Nuclear Power Plants," *Annals of Nuclear Energy,* vol. 52, pp. 113-124, 2013.

[51] E. Chen, L. Lin and N. Dinh, "Advanced Transient Diagnostic wwith Ensemble Digital Twin Modeling," Cornell University (arXiv), 22 May 2022. [Online]. Available: https://arxiv.org/abs/2205.11469.

[52] 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, no. 1, p. 107861, 2021.

[53] V. Agarwal, C. Walker, K. Manjunatha, T. Mortenson, N. Lybeck and A. Gribok, "Technical Basis for Advanced Artificial Intelligence and Machine Learning Adoption in Nuclear Power Plants," Idaho National Laboratory, Idaho Falls, 2022.

[54] R. Roelofs, S. Fridovich-Keil, J. Miller, V. Shankar, M. Hardt, B. Recht and L. Schmidt, "A Meta-Analysis of Overfitting in Machine Learning," in *Proceedings of the Conference on Neural Information Processing Systems*, Vancouver, 2019.

[55] P. Ortega and V. Maini, "Building safe artificial intelligence: specification, robustness, and assurance," DeepMind Safety Research Blog, 2018.

[56] D. Manheim and S. Garrabrant, "Categorizing Variants of Goodhart's Law," Cornell University, arXiv, 2019. [Online]. Available: https://arxiv.org/abs/1803.04585.

[57] D. Hendrycks, K. Zhao, S. Basart, J. Steinhardt and D. Song, "Natural Adversarial Examples," 2021. [Online]. Available: https://arxiv.org/abs/1907.07174.

[58] A. Barbu, D. Mayo, J. Alverio, W. Luo, C. Wang, D. Gutfreund, J. Tenenbaum and B. Katz, "ObjectNet: A large-scale bias-controlled dataset for pushing the limits of object recognition models," in *Proceedings of the Conference on Neural Information Processing Systems*, Vancouver, 2019.

[59] 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].

[60] International Energy Agency, "Energy and Air Pollution, World Energy Outlook Special Report," International Energy Agency, 2016.

[61] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," Department of Commerce, Washington, 2023.

[62] R. I. Rolland and R. Schneider, "Lessons Learned in PRA Modeling of Digital Instrumentation and Control Systems," in *Proceedings of the 16th Probabilistic Safety Assessment & Management Conference*, Honolulu, 2022.

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

[64] R. W. Rolland III and R. E. Schneider, "Digital I&C Model Pilot Development, PA-RMSC-0607, Rev. 2," Westinghouse Electric Company, 2021.

[65] U.S. General Accounting Office, "Patriot Missile Defense, Software Problem led to System Failure at Dhahran, Saudi Arabia," U.S. General Accounting Office, Washington, 1992.

[66] P. M. Chandra, "On the generalized distance in statistics," *Proceedings of the National Institute of Sciences of India,* vol. 2, no. 1, pp. 44-55, 1936.

[67] T. Eltoft, T. Kim and T.-W. Lee, "On the Multivariate Laplace Distribution," *IEEE Signal Processing Letters,* vol. 13, no. 5, pp. 300-303, 2006.

[68] J. W. Lane, J. M. Link, J. M. King, T. L. George and S. W. Claybrook, "Benchmark of GOTHIC to EBR-II SHRT-17 and SHRT-45R Tests," *Nuclear Technology,* vol. 206, no. 7, pp. 1019-1035, 2020.

[69] 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.

[70] 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.

[71] 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.

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

[73] U.S. Nuclear Regulatory Commission, "Digital Instrumentation and Controls, Task Working Group #2: Diversity and Defense-in-Depth Issues," DI&C-ISG-02, U.S. Nuclear Regulatory Commission, 2009.

[74] Institute of Electrical and Electronics Engineers, "Systems and software engineering -- Life cycle processes -- Requirements engineering IEEE 29148," Institute of Electrical and Electronics Engineers, 2018.

[75] Institute of Electrical and Electronics Engineers, "Standard for Software and System Test Documentation IEEE 829," Institute of Electrical and Electronics Engineers, 2008.

[76] International Electrotechnical Commission, "Functional safety of electrical/electronic/programmable electronic safety-related systems," International Electrotechnical Commission, 2010.

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

[78] 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.

[79] 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.

[80] 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].

[81] 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.

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

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

[84] P. K. Kapur, S. Anand and K. Yadav, "A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation," *IEEE Transactions on Reliability,* vol. 60, no. 1, pp. 331-340, 2011.

[85] Q. Li and H. Pham, "A generalized software reliability growth model with consideration of the uncertainty of operating environments," *IEEE Access,* vol. 7, no. 1, pp. 84253-84267, 2019.

[86] 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.

[87] 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.

[88] G. S. Mudholkar and D. K. Srivastava, "Exponentiated Weibull family for analysing bathtub failure rate data," *IEEE Transaction on Reliability,* vol. 42, no. 2, pp. 299-302, 1993.

[89] 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.

[90] E. Chen, H. Bao, T. Shorthill, C. Elks, A. V. Jayakumar and N. Dinh, "Application of Orthogonal Defect Classification for Software Reliability Analysis," Cornell University, Arxiv, 24 May 2022. [Online]. Available: https://arxiv.org/abs/2205.12080. [Accessed 11 August 2022].

[91] R. W. Butler and G. B. Finelli, "The infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," *IEEE Transactions on Software Engineering,* vol. 19, no. 1, pp. 3-12, 1993.

[92] International Business Machines Corporation (IBM), "Orthogonal Defect Classification (Archival): ODC v5.2 for Design & Code Document," 2013. [Online]. Available: https://researcher.draco.res.ibm.com/researcher/view_group_subpage.php?id=5020.

[93] 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].

[94] 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.

[95] 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.

APPENDIX A – REDUNDANCY-GUIDED SYSTEM-THEORETIC HAZARD ANALYSIS (RESHA)

This appendix provides the steps in detail for performing RESHA. Each step is elaborated subsequently as shown in Figure 3.

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

The purpose of STEP-1 is to establish a system sketch that serves as a blueprint for the analysis. This involves collecting system design information, such as wiring diagrams, piping and instrumentation diagrams, logic diagrams, etc. This information is then used to create a system sketch that outlines the processors, sensors, controllers, components, interactions, and connections of the system. It is important to note that the goal of this step is not to cram everything into a single diagram, rather it is to gain a sufficient understanding of the system to complete the hazard analysis; the level of detail obtained in this step lays out the foundation for the work.

Digital I&C structures can be divided into three hierarchical levels [69]: (1) divisions; (2) units; and (3) modules. Divisions typically fulfill the role of monitoring processes, systems, or functions redundantly with other divisions by relying on their subsystems of units and modules. Units perform specific tasks that are supported by the function processing capabilities of modules [69]. The system information collected should clearly account for the flow of information between divisions, units, and modules. Emphasis should be placed to clearly illustrate the redundancy and diversity of the system to provide a basis for constructing the FT in STEP-2 and control structure in STEP-3 and identifying potential CCFs in STEP-5. Key points 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 that provide input to process modules. The level of detail to be included in the hazard analysis depends on the scope of the investigation.

At this stage of RESHA, the purpose of the analysis should be defined. The system of interest (SOI) should be defined along with its losses, hazards, boundaries, and environments. The system details gathered in this step should provide clear indication of the boundary, and environment. Losses can be identified as something of value to stakeholders, including loss of human life or injury, property damage, environmental pollution, loss of mission, loss of reputation, loss or leakage of sensitive information, or any other loss that stakeholders deem unacceptable [8]. Losses should be listed to provide direction for identifying hazards, i.e., those states or sets of conditions that, in a particular worst-case scenario, will lead to one or more losses [8]. These hazards pertain to the system as a whole and do not represent a complete hazard analysis. Instead, the losses and hazardous states that can be identified provide a foundation for contextual assessment of the unsafe interactions that may occur within the systems.

Step 2: Develop a FT consisting of the hardware failures for a chosen function of the system of interest.

A FT is constructed based on the information gathered in STEP-1. For analysis of a digital system with redundancy, the structure of the FT should follow the levels of redundancy (e.g., from the division to the unit and module levels). The structure of the FT should capture the details of redundancy that will aid in the subsequent steps of the hazard analysis. In most cases, the highest level of redundancy will be associated with protecting the main function of the SOI. For instance, a system may have two or more divisions, independent and redundant in function, to ensure the reliability of that system. The use of a redundancy-guided structure makes it convenient to add and track postulated CCF events. The FT is created using the following 2-part process[a] adapted from the NASA Handbook [70].

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[b] are based on the purpose of the SOI. The failure of the SOI's most significant function is a priority event to be analyzed by the FT—a top event. STEP-3A may also be visited briefly to ensure the proper selection of top events. For example, some events are not failures, but are events of interest. The spurious activation of a shutdown process may cause no immediate harm, but rather a financial burden to the stakeholders. Such events may be selected as the top event within the FT.

STEP-2B: Construct the FT. Starting from the top event, the construction proceeds by determining the "necessary and sufficient" logical combinations e.g., "and," "or," and "n/m" functions of events that contribute to the occurrence of the top event [70]. This analysis proceeds by gradually identifying the essential events that are logically connected to the top event, progressing step-by-step until reaching an event that cannot be further resolved, either by way of fact, or at the discretion of the analyst [70]. At this stage of the assessment, the focus is on tracking hardware failures, while software failure identification is a subject of RESHA Step 3. However, at this stage, it can be useful to add undeveloped software events as temporary placeholders to assist in the organization of the FT. Thus, the FT comprises a pattern of three branches that repeat for various DI&C components of the system: the hardware failure branch, the software failure branch, and the dependency or external failure branch, as depicted in the Figure 33 below.

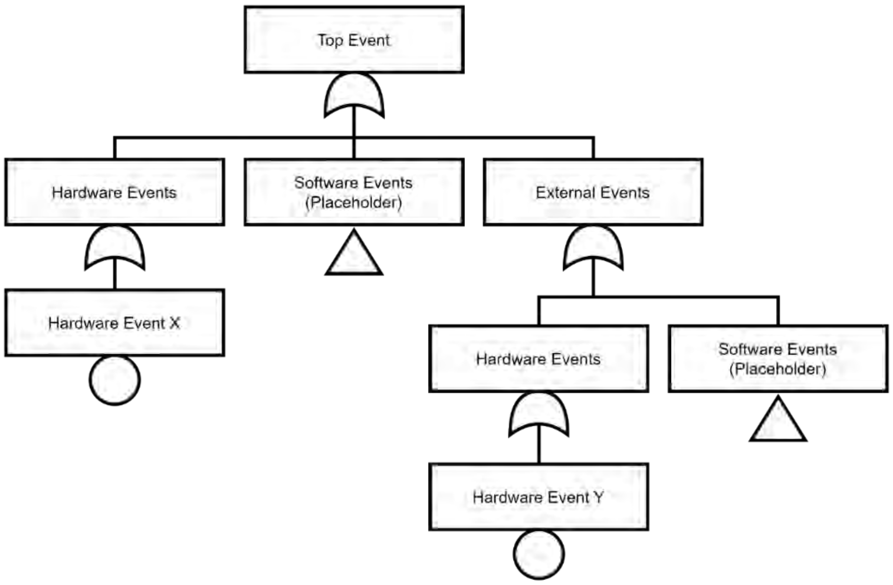


Figure 33. Example of FT organization.

[a] This process has been simplified to two parts and directed to the analysis of highly redundant DI&C from the original eight-step FTA given in the NASA FTA Handbook [71].

[b] There can be multiple top events for a single system with the FT for each new top event varying significantly. The number of FT to be analyzed will depend on the project scope.

Under the hardware branch of a component, there are hardware-based events relevant to the specific component and its applicable hardware failure modes. Under the software branch, a place holder is added to account for the internal failure mechanisms, such as those associated with a processor's control algorithm or process model that will be added in STEP-3. Under the external failure branch, the upstream dependencies are added and can include both hardware and software failures. This typically will include other controllers, information processors, or sensors. The process follows a pattern until the dependency branch can be resolved no further.

A note on the construction of the FT: The FT should be populated with hardware events and if readily known and within the scope of the assessment, hardware CCF events can and should also be included. Though the focus of CCF comes in STEP 5, there may be instances where an existing hardware FT is available. In such an instance it is logical to work with the existing FT including its CCF event data.

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

The main purpose of STEP-3 is to identify, by means of a redundancy-guided application of STPA, the software failure events to add to the FT from STEP-2. These failures can be categorized as unsafe control actions (UCAs) or unsafe information flow (UIFs) as result of an unrealistic process model, an inappropriate control algorithm, incorrect feedback, or outside information. A UCA is control action that, in a particular context and environmental conditions, will lead to a hazard [8]. UIF is analogous to UCA, but for information flow rather than control actions UCAs and UIFs offer insights to the control action and feedback pathways of a system. 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 feedback from controlled process—have a potential to lead to unsafe behavior of the digital system. Potential software failures can be understood and analyzed by identifying UCAs/UIFs. RESHA works by identifying the UCAs through a thorough examination of a redundancy-guided control structure diagram (i.e., from a redundancy-guided application of STPA). Once identified, the relevant UIFs and UCAs are defined and inserted under the software branches of the FT.

According to the STPA Handbook there are four main parts to the STPA. The first two focus on the construction of a control structure, and the second two focus on the analysis of unsafe component interactions. A complete description of STPA can be found in the STPA Handbook 2018 [8]. The process given here highlights the key parts directly applicable to RESHA.

STEP-3A: Create a model of the control structure. This step involves identifying, grouping, and organizing the system components, controllers, and controlled processes and information flow to create a control structure diagram. For RESHA, a redundancy-guided multilayer-control structure is created which focuses on systematic information exchanges within each redundancy level. By reframing the standard STPA approach, complex and highly redundant digital systems can be broken down into more manageable portions. In addition, the redundancy-guided approach allows a straightforward creation of a multi-layer control structure to both capture the details of the SOI and support the identification of potential CCFs.

The first layer of the control structure should be based on the highest level of redundancy (e.g., divisional), followed by any sub-layers (e.g., unit and module-based[c] redundancies). Each control structure layer is created with numbered control actions and feedback signals until a final, redundancy-guided, multi-layer control structure is created for the complete SOI, as shown in Figure 34.

STEP-3C: Identify UCAs/UIFs. A UCA is control action that, in a particular context and environmental conditions, will lead to a hazard [8]. UIF is similar to UCA, but for information flow rather

[c] Creating a control structure for the internal structure of a module may not be required based on the assumed/desired model resolution and scope.

than control actions. Thus, a UIF is information that, in a particular context and conditions will lead to a hazard. UCAs/UIFs can be determined by the assessment of the numbered control actions and feedback signals identified from the control structure diagram. The UCAs/UIFs should be numbered and tracked within a table. The UCAs/UIFs will be used to add software failures to the FT from STEP-2. For each control action identified, there are four categories of potential UCAs/UIFs that may affect the behavior of the SOI. These categories are shown in Table 30. Note that each UCA/UIF shall be formatted based on the standard format given in [8] for consistency and clarity. (Note that destination/user of control action or information flow can also be indicated as part of the control action and context [4].)

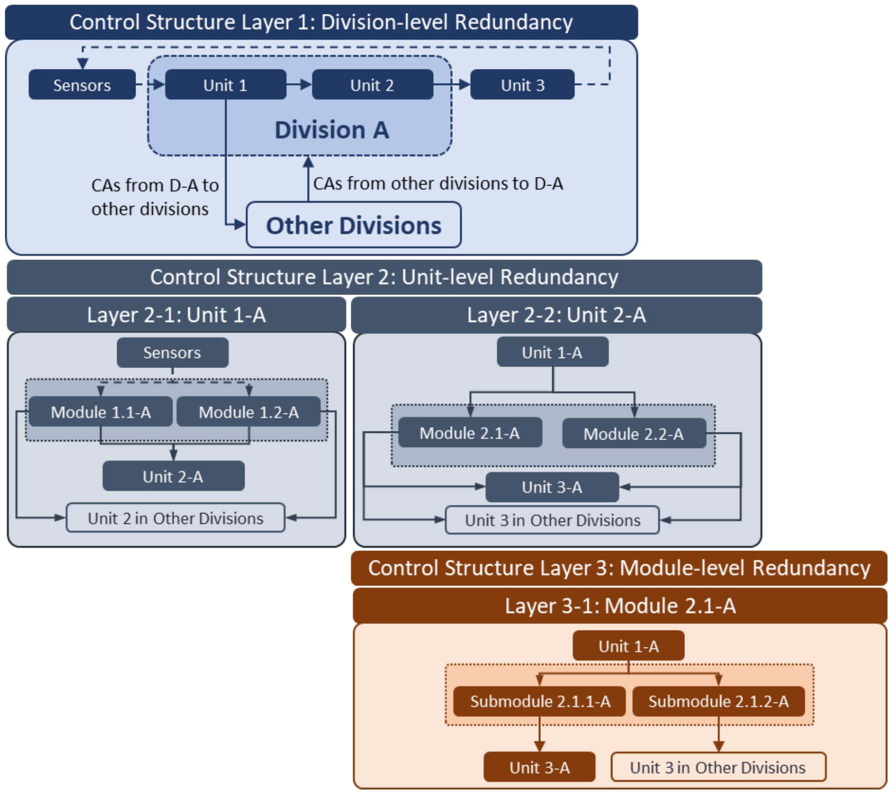

Control Structure Layer 1: Division-level Redundancy
Sensors
Unit 1
Unit 2
Unit 3
Division A
CAs from D-A to other divisions
CAs from other divisions to D-A
Other Divisions
Control Structure Layer 2: Unit-level Redundancy
Layer 2-1: Unit 1-A
Sensors
Module 1.1-A
Module 1.2-A
Unit 2-A
Unit 2 in Other Divisions
Layer 2-2: Unit 2-A
Unit 1-A
Module 2.1-A
Module 2.2-A
Unit 3-A
Unit 3 in Other Divisions
Control Structure Layer 3: Module-level Redundancy
Layer 3-1: Module 2.1-A
Unit 1-A
Submodule 2.1.1-A
Submodule 2.1.2-A
Unit 3-A
Unit 3 in Other Divisions

Figure 34. Illustration of a multilayer control structure.

$$\text{UCA} = [\text{Source}] + [\text{UCA Type}] + [\text{Control Action}] + [\text{Context}] + [\text{Link to System Hazards}] \tag{32}$$

$$\text{UIF} = [\text{Source}] + [\text{UIF Type}] + [\text{Information Flow}] + [\text{Context}] + [\text{Link to System Hazards}] \tag{33}$$

Table 30. Examples of basic types of UCAs and UIFs.

	Type A (Not providing causes hazard)	Type B (Providing causes hazard)	Type C (Timing/order of discrete events causes hazard)	Type D (Timing of continuous events causes hazard)
Unsafe Control Action (UCA)	Control action is **not provided** when needed	Control action is provided when it is **not needed**	Control action is provided **to early**, **too late**, or in the **wrong order**	Control action is **stopped too soon** or **applied too long**

Unsafe Information Flow (UIF)	Information is **not provided** when needed	Information is provided when it is **not needed**	Information is **early**, **late**, **out of sync**, or **out of order**.	Information is **stopped too soon**, **applied too long**,
Note: UIF is still under development, and value such as **too low**, **too high**, **not-a-number**, **infinite** or otherwise **corrupt** information may be classified under Type B or D and is up to the discretion of the analyst.				

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 FT from STEP-2. Not every identified UCAs/UIFs belong in the same FT; rather, only those UCAs/UIFs whose occurrences are necessary and sufficient to lead to the specific top event. For example, control actions that are continuous, like those relating to category four above, may, in some cases only be applicable to top events requiring continuous control. Also consider, if an event represents a was a failure to actuate (i.e., a type A or UCA-A mode of failure), then it is the analysis must carefully select reasonable causal events that would result in a particular UCA-A event. Each appropriately selected UCA/UIF is added to the FT. These events will fill the software event placeholders that were added to the FT previously. Figure 35 provides an example of this FT.

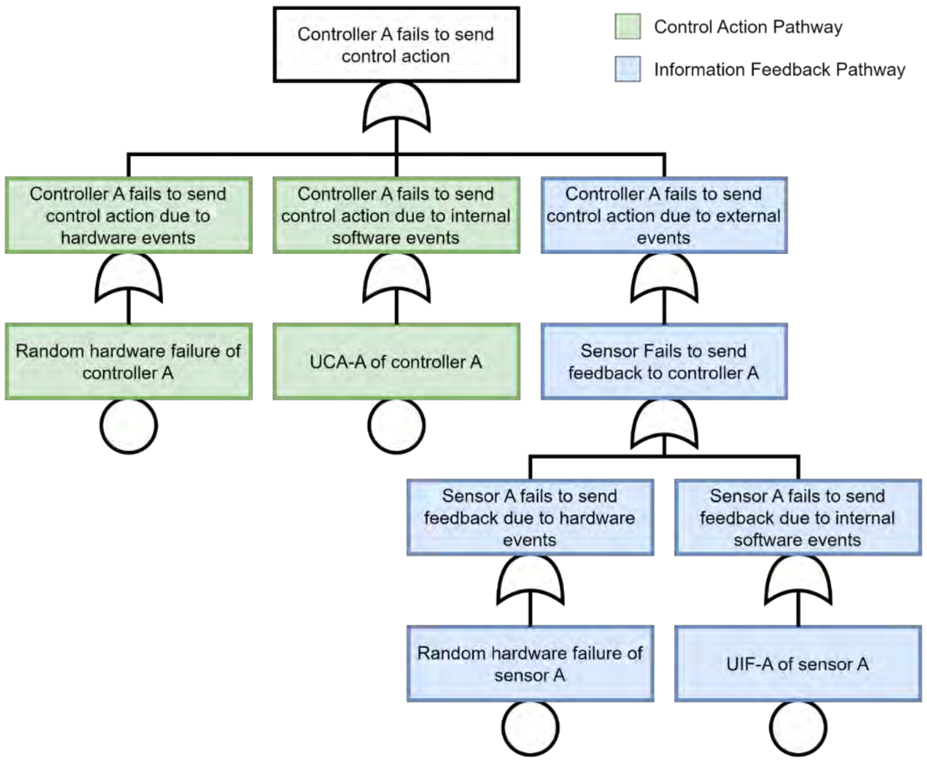

Controller A fails to send control action
🟩 Control Action Pathway
🟦 Information Feedback Pathway
Controller A fails to send control action due to hardware events
Controller A fails to send control action due to internal software events
Controller A fails to send control action due to external events
Random hardware failure of controller A
UCA-A of controller A
Sensor Fails to send feedback to controller A
Sensor A fails to send feedback due to hardware events
Sensor A fails to send feedback due to internal software events
Random hardware failure of sensor A
UIF-A of sensor A

Figure 35. Example FT structure for RESHA.

Step 5: Identify potential CCFs to add to the FT.

The purpose of this step is to identify potential hardware and software CCFs and add them to the FT. CCFs are conditional on a shared root cause and coupling factor. Again, redundancy is employed as a guide to identify potential CCFs. Within the DI&C system, redundant components may share common hardware and software features making them susceptible to CCFs. Also, I&C systems will frequently employ diversity to ensure functionally redundancy. It has been noted that diversity does not guarantee the prevention of software CCF [17], [19]. Thus, the functionally redundant components should be identified and grouped. These groupings represent a high-level assessment of potential CCFs. At this stage, a conservative approach may be to create CCF events for each of these groups and add them to the FT. However, further refinement should be considered.

The identification of coupling factors beyond simple functional redundancy helps to determine unique CCCGs. Various examples of coupling mechanisms for hardware or analog systems have been indicated in the literature, for example, some provided by NUREG/CR-5485 include design, function, installation, maintenance, and environmental conditions [22]. Thus, the simple group of functionally redundant hardware components may be refined into subgroups. Consider a system with three redundant controllers. In this system, controllers 1 and 2 are in a separate room from controller 3. At a high level, all three controllers are redundant and may be susceptible to CCF. Further refinement may be accomplished by considering that location may allow an additional CCF to occur between controllers 1 and 2 while controller 3 remains unaffected. Figure 36 provides an example FT with CCFs added.

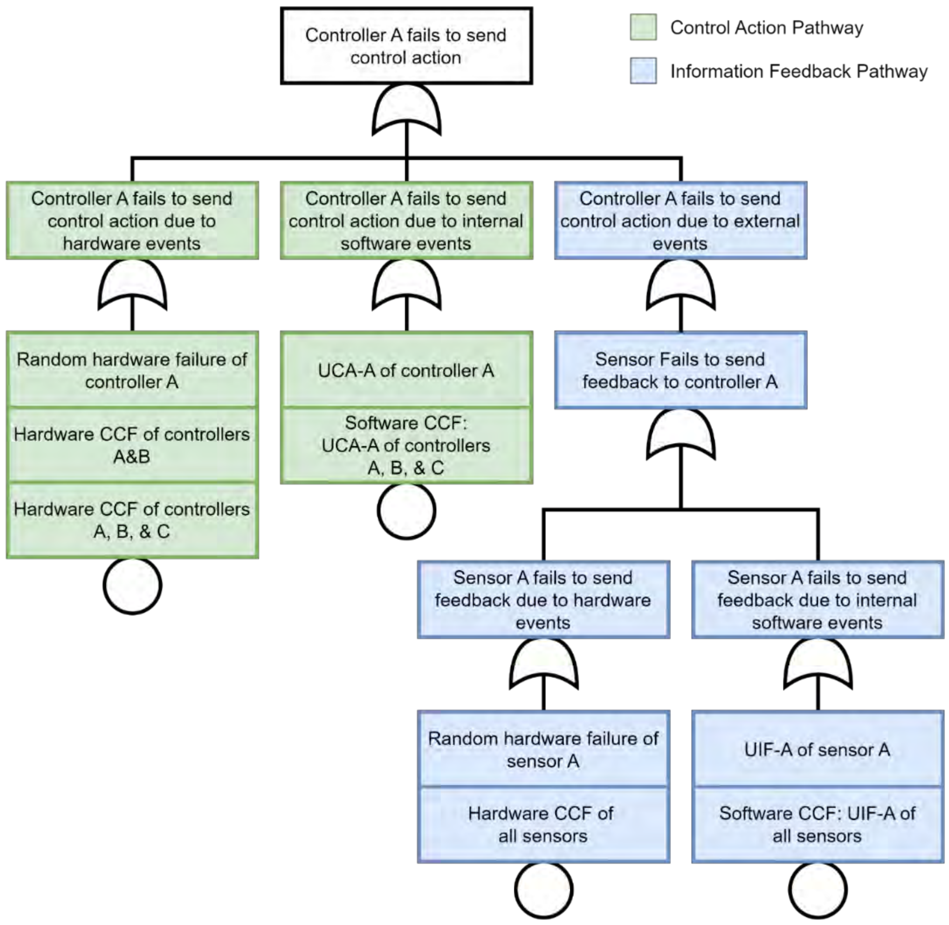

Controller A fails to send control action
🟩 Control Action Pathway
🟦 Information Feedback Pathway
Controller A fails to send control action due to hardware events
Controller A fails to send control action due to internal software events
Controller A fails to send control action due to external events
Random hardware failure of controller A
Hardware CCF of controllers A&B
Hardware CCF of controllers A, B, & C
UCA-A of controller A
Software CCF: UCA-A of controllers A, B, & C
Sensor Fails to send feedback to controller A
Sensor A fails to send feedback due to hardware events
Sensor A fails to send feedback due to internal software events
Random hardware failure of sensor A
Hardware CCF of all sensors
UIF-A of sensor A
Software CCF: UIF-A of all sensors

Figure 36. Example FT with CCFs added.

Identification of software CCFs requires consideration of additional software-centric features. 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 [71] and [17]). A software CCF will occur when a coupling mechanism creates a scenario for operational conditions to activate a common software defect. 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, input variables, etc. Ultimately, 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.

There are several action items that should occur for CCF analysis. To start, the analyst should identify the functionally redundant components or elements of the system. Next, the analyst identifies and lists all the known coupling mechanisms for the system. Emphasis on software-centric features (e.g., shared software code, shared requirements, similar languages employed, common inputs, shared operators/users). The third action item is to group the components by their shared coupling mechanisms thereby identifying the CCCGs of the system. Finally, CCFs are added to the FT corresponding to each identified CCCG. A CCF event should be added to the FT for each CCCG that can be distinguished by a unique set of coupling factors. The CCF events should be added under each component of the FT where applicable.

- Action Item 1: Identify functionally redundant components or elements of the system.
- Action Item 2: List coupling mechanisms that are known to exist for the system.
- Action Item 3: Form CCCGs based on the coupling mechanisms.
- Action Item 4: Add CCF events to the FT where applicable.

In many instances the hardware FT constructed for Step 2 may already include hardware CCFs. However, the analysis may benefit from revisiting these 4 action items given as they may help capture any hardware CCFs that were not included previously. The focus of Step 5 is the identification of potential CCF, including hardware and software CCFs. The basic action items listed are generally applicable and for supporting CCF analysis.

Step 6: Solve the FT for the minimal cut sets to determine critical failures in the design.

The integrated FT and minimal cut sets are chief outcomes of RESHA. The purpose of this step is to solve the FT for the minimal cut sets and evaluate the design. Solving the FT is typically conducted using a software package such as SAPHIRE [72]. The number of basic events within each cut set varies; those cut sets that only consist of a single event are called first-order cut sets. For a qualitative analysis, the significance of each cut set is assumed to be dependent only on order. Therefore, the fewer basic events in the cut set, the more likely it will occur[d]. Single points of failure can be classified as first-order cut sets, yet the NRC does not consider CCFs as SPOFs for the sake of design basis evaluations [73]. To avoid any confusion, in this work, cut sets containing a CCF as their only basic event are simply referred to as first-order cut sets. By evaluating the results for first-order cut sets, designers can determine where and how the SOI is vulnerable and redesign accordingly.

There will be instances where, due to the system design, there are no first-order cut sets to analyze. In such cases, the analysis should proceed with second-order, or third-order, etc. Essentially, the smallest or shortest cut-sets are most likely to occur. And, in the absence of data, the shortest cut-sets are a logical area to focus on for re-designs. A useful exercise is to evaluate portions of the FT such as those that were

[d] The addition of probability values can lead to higher order cut sets occurring with greater probability than those of lower order. These effects will be investigated as part of the reliability analysis in a future work.

identified based on functional redundancy (i.e., system or division levels of the FT). Evaluating such portions of the FT, including those that are most concerning to designers, may reveal localized first-order cut sets that may serve as focal points for redesign efforts, or additional analysis.

Step 7: Identify and provide guidance to eliminate critical failures or their causes.

The previous six steps serve to identify the hazards of the system. The intention of STEP-7 is to identify and provide guidance to eliminate critical failures or their causes. This step comes from the fourth part of STPA and focuses on the context for which UCAs or hazards may occur. The study of loss scenarios encompasses identifying and evaluating the combination of causal factors that can lead to a specific loss. It is in these loss scenarios that Type 1 or Type 2 interactions [15] may be identified. The STPA Handbook indicates that causes of UCAs can be grouped into two categories: (1) unsafe controller behaviors; and (2) inadequate feedback or other inputs [8].

- Category 1: failures related to the controller itself (physical failures); inadequate control algorithms (the decision-making process may be inappropriate); inadequate process model (the controller's view of the controlled process may be incorrect [8].
- Category 2: Information/Feedback is not received by the controller; inadequate feedback/information is received by the controller; unsafe control input (a controller receives an UCA input from another controller) [8].

According to the STPA handbook [8], "emergent properties are properties that are not in the summation of the individual components but 'emerge' when the components interact." And that "emergent properties arise from relationships among the parts of the system." Therefore, it is important to consider how the interactions of components may influence the system behavior. As a simple example, a UCA of an altitude controller may occur due to a UIF of one of the sensors that provide information to the control system. RESHA models the UIF and the UCA within a fault tree to capture the relationship and the emergent behavior of the altitude controller due to interactions it has with other components of the system. So, even though the altitude controller itself does not fail, the interaction with other components has led to unwanted and unsafe behavior. In other words, A UCA may occur when correct information (i.e., Category 2) is not provided to the controller.

The study of loss scenarios is intended to capture the emergent behaviors and interactions that can lead to the UCAs, which may lead to a loss. The importance of this activity is not missed by RESHA, rather loss scenarios are represented by failure paths within a fault tree. To aid in the representation of loss scenarios, RESHA models directly unsafe information flow associated with the system of interest and its interactions with itself, controlled processes, or other systems. Together, failures associated with the control action pathway and the information feedback pathways are detailed by the integrated fault tree, particularly by the inclusion of UCAs and UIFs events.

Of course, there may be numerous causes (i.e., causal factors) that may result in a particular UCA or UIF (e.g., hardware, software, or human causes) and a loss scenario. The last step of STPA is to identify these causal factors and present options for eliminating them. A key insight provided by the integrated fault tree is the relative relations and links between each failure pathway of the FT (i.e., cut sets). In other words, some cut sets may appear more critical (i.e., first-order cut sets) and efforts can be made to reduce or eliminate the causal factors of certain basic events within these cut set. For example, a given UCA event may be eliminated or mitigated by addressing or reducing the identified causes of that event (i.e., adding a diverse system, a watchdog time, specific processing criteria). The initial efforts in identifying causal factors may result in a resource bank of typical causal factors that can be used to expedite future analyses, thus reducing the cost of the RESHA over time.

APPENDIX B – ORTHOGONAL-DEFECT CLASSIFICATION FOR ASSESSING SOFTWARE RELIABILITY (ORCAS)

The essence of ORCAS is to methodically collect failure data throughout the SDLC. This is then used to assess the anticipated operational reliability qualitatively and quantitatively. Different recommended approaches to collect and assess the completeness of the SDLC are included in ORCAS.

The overall workflow of the proposed methodology can be seen in Figure 37. Items in the dashed box are elements pertaining to ORCAS. In general, the outputs are the software failure probabilities and confidence in the assessment process. Note that qualitative evidence is used to support or reject the conclusions of the assessment. The objective of stages 0 to 3 is focused on comprehensive data collection within the scope of the assessment. In stage 4 and 5, the data are then used to determine failure probability and the confidence in the assessment. It is important to note that the assessment of software reliability should be continuously evolving with the implementation and design of the 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.

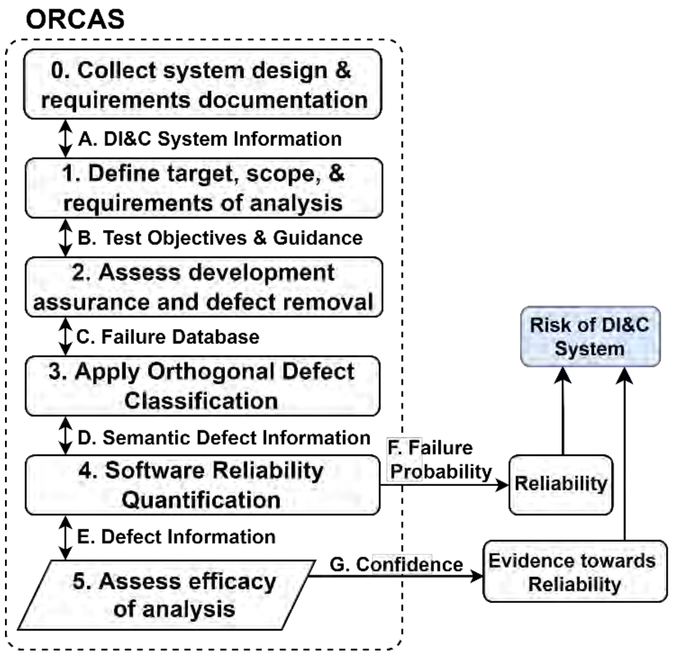

ORCAS
0. Collect system design & requirements documentation
A. DI&C System Information
1. Define target, scope, & requirements of analysis
B. Test Objectives & Guidance
2. Assess development assurance and defect removal
C. Failure Database
3. Apply Orthogonal Defect Classification
D. Semantic Defect Information
4. Software Reliability Quantification
E. Defect Information
5. Assess efficacy of analysis
F. Failure Probability
G. Confidence
Reliability
Evidence towards Reliability
Risk of DI&C System

Figure 37. Overall process of ORCAS for software reliability analysis.

Stage 0: Collect system design and software requirements specification documentation.

In stage 0, the relevant information and details pertaining to the system are collected. This stage 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 [74], IEEE 829 [75], IEC 61508 [76]), 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 the construction of objectives and relevant targets of the assessment.

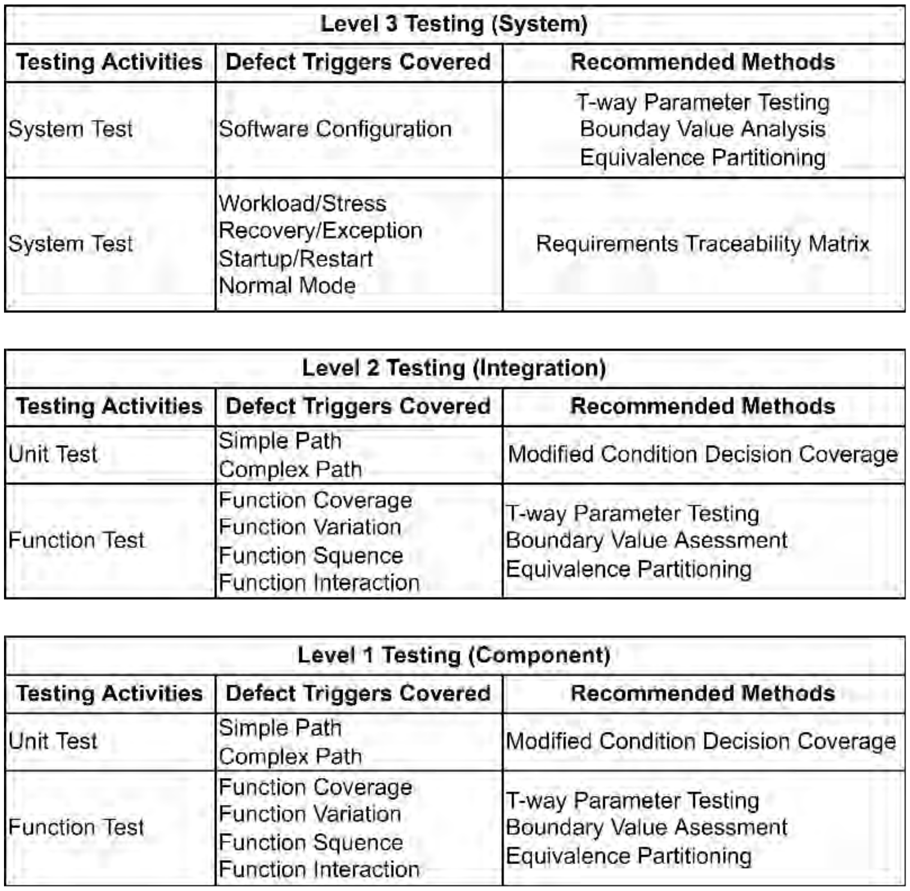
Stage 1: Define target, scope, and requirements of analysis.

In stage 1, the target and scope are defined. A control structure diagram (CSD) based on top-down deconstruction is created to identify the target components and their interactions with each other. Methods to construct the CSD can be found in [8]. The scope of the analysis is also inherently defined by the CSD. Here, the desired modules and functions of the system are outlined, the hierarchal structure, and the type of tests anticipated to be completed based on specified constraints.

From the formal documentation collected in stage 0, the requirement traceability matrix (RTM) is formed. The RTM specifies the functional and non-functional requirements on the software system as well as the associated tests required to achieve each target. The RTM is used as qualitative evidence toward the confidence in the assessment.

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 [77], modified condition decision coverage (MCDC) [78], boundary value analysis (BVA) [79], and equivalence partitioning (EP) [79] are recommended to verify the functional correctness of the software modules. These methodologies have been experimentally proven to be effective at defect removal and can be easily automated for the testing process. For instance, the Automated Combinatorial Testing for Software Tools (ACTS) is a free automated test generation tool developed by the National Institute of Standards and Technology (NIST) for T-way combinatorial and sequential testing [80]. They showed in Reference [80], that ACTS can capture nearly all hidden input related defects. Furthermore, the tool can partially automate the testing process thereby reducing effort.

Within the ORCAS methodology, testing adequacy is gauged by a three-level test suite hierarchy known as the Trigger Covering Array (TCA), as shown in Figure 38. Each level specifies contextual conditions (also known as triggers) that need to be considered during test implementation for more comprehensive debugging. In Reference [81], they experimentally demonstrated that debugging can be improved by considering the triggers mentioned. Each trigger also has associated recommended activities to complete each test requirement. The purpose of this stage is to ensure that the tests conducted through the SDLC are effective and complete at detecting software defects. By considering the various triggers of defects and levels of software in the test process, the potential for latent hidden defects in the code are reduced qualitatively, improving reliability. Note that the test execution is not conducted at this stage. Rather, a collection of the anticipated and necessary test cases is constructed so it can be compared to the actual tests conducted to identify inadequate areas.



Level 3 Testing (System)		
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 (Integration)		
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 38. Three-tier software testing requirements with recommended activities and methods.

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 associated test cases that demonstrate adequate conformance. 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 historical defect data that can later be used to quantify software failure modes. The defect reports can be collected from two sources. The first source includes existing defect reports discovered during the SDLC. The second source is through additional testing, which is initiated due to inadequacies in the RTM, TCA, or structural coverage.

Stage 3: Apply ODC to the collected data.

In stage 3, the defect reports are collected and categorized based on ODC theory [82]. Specifically, defects fall under one and only one class specified in Table 31. An analysis of defect reports involves understanding what went wrong and how it was resolved. Recall that defects are classified based on the resolution or solution and not what failure occurred. It should also be noted that if widely different solutions exist for the same problem, it may indicate inadequate requirements and constraints specification for the problem.

Table 31. Defect class table with abridged descriptions [83].

Defect Class	Description
Assignment	Value assigned incorrectly or not at all.
Checking	Missing or incorrect validation of parameters or data in conditional statements.
Algorithm	Efficiency or correctness problem that affects functionality without a formal design change.
Function	Missing or incorrect functionality affecting system capability requiring a formal change.
Timing	Inappropriate serialization/timing of limited resources such as memory, locks, updates, etc.
Interface	Errors in communication between any level of software (i.e., macros, drivers, objects).
Documentation	Consistency of documented functions and implemented code functions.
Relationship	Conditional association problems related to linkage between data structures and objects.

Stage 4: Quantify 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. For low-reliability systems (i.e., greater than 1E-4 occurrence probability), it is assumed that conventional SRGM models are applicable as data collection is tractable. The growth models attempt to predict the number of hidden latent defects that may exist in the software after testing is complete. 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 models developed by growth models are useful and accurate. If there are insufficient data points, the growth models can be 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 sufficiently simple such that the number of defects recorded is limited. The second condition is to ensure model fidelity. Inconsistent but plentiful failure data can also lead to uncertain predictive results and should also be avoided.

The exact mathematics behind reliability growth models is not discussed in detail as extensive literature already exists and is referenced in this work. However, importantly, no one SRGM can be used to generalize across all datasets as development lifecycles vary significantly between organizations. Within ORCAS, five SRGMs are provided in Table 32 to cover different potential shape profiles, underlying SDLC assumptions, and development lifespans. Alternatively, a unified SRGM can also be used, such as those referenced in [84, 85]. Below, *M(t)* is the predicted number of remaining defects in the code at a specific time.

Table 32. Different SRGMS to model defect history.

Model Name	*M(t)*
Goel-Okumoto (GO) [86]	$a(1-e^{-bt})$
GO S-Shaped [87]	$a(1-(1+bt)e^{-bt}$
Weibull [88]	$a(1-e^{-bt^{c}})$
Yamada Raleigh [89]	$a(1-\exp(-r\alpha(1-\exp(-\frac{\beta t^2}{2}))$
Log Poisson [86]	$1/c\ln(c\alpha t+1)$

In Figure 39, an example of defects recorded over time is presented with the HBase software dataset and modeled with the GO S-Shaped SRGM. On the x-axis is the number of combined testing days conducted over the development period of HBase. Combined testing days is the number of days of testing multiplied by the number of developers. On the y-axis is the cumulative number of defects over testing time as well by defect class. Both the cumulative number of defects detected and by defect class are plotted. The GO S-Shaped SRGM (dashed lines) is modeled against experimental data to show how SRGMs can be used to predict individual class occurrence probabilities. Note that the sum of all defect classes is equal to the cumulative number of defects. In the figure, near the end of the development cycle (i.e., between 12500 – 17500), pre-release, the cumulative number of defects recorded over time flattens, demonstrating the idea of growing reliability over increasing development effort. For additional details on ODC class specific SRGMs, the author recommends Ref. [82]. After an SRGM is fitted to the historical data, the occurrence probability of a specific defect class remaining undetected in the code can be determined using Equation (34).

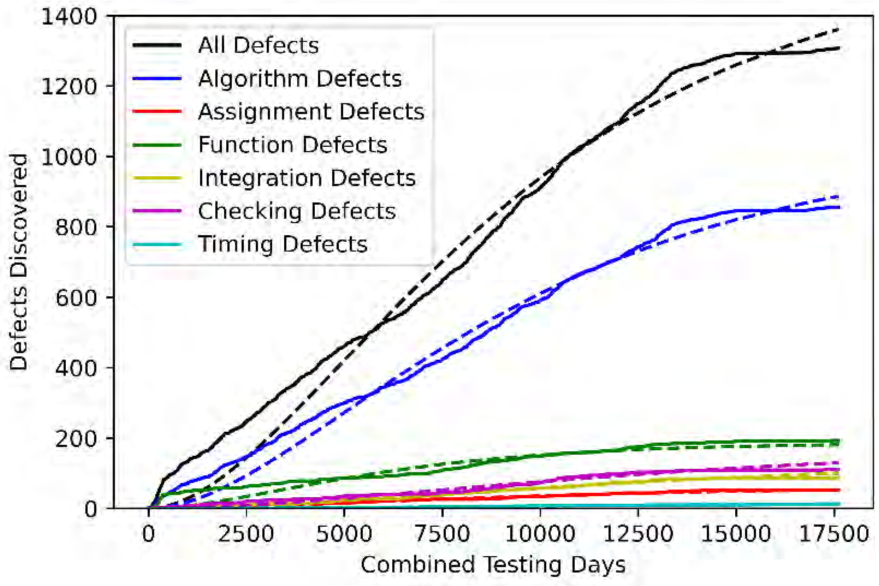


Figure 39. Defect modeling with GO S-Shaped curve for cumulative and specific defect classes [90].

$$F_i(t|s) = 1 - \exp\big(M_i(s) - M_i(s+t)\big) \tag{34}$$

$$P(\mathrm{UX_j}) = \sum_{i=\text{defect class}} P(\mathrm{UX_j}|i)F_i(t|s) \tag{35}$$

Here, $F_i(t|s)$is the probability of the i^{th} defect class occurring after t additional hours of testing, and s is the last recorded defect time. The probability of specific UCA/UIF modes can now be determined by multiplying the correlation of each class with the occurrence probability per hour per class probability, as shown in Equation (35). Recall that the correlations were 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 from the testing efforts conducted.

The first term $P(\mathrm{UX_j}|i)$ is the relational strength between defect classes and UCA/UIF mode. In Table 33, the overall mean and two standard deviation confidence intervals of all class correlations to UCAs/UIFs are shown. Table 33 also provides a global perspective of the relationship between defect classes and UCA/UIF failure modes. For instance, if the target software is speculated to have an assignment defect, from the table, it has a 28.4% chance to cause a UCA/UIF-A, a 64.8% chance of a

UCA/UIF-B, and so on. The exact failure modes (i.e., UCA-A or UIF-A) are determined by the functionality of the target software. The correlation strength is described by $P(\mathrm{UX_j}|i)$ between the j^{th} UCA/UIF failure mode and the i^{th} defect class. In row one, column D, no data points were observed so no correlation is assigned. In the last two rows, no data points from the analyzed software development lifecycles included documentation and relationship defects. An uninformed flat distribution is assumed for each UCA and UIF correlation. Note that the global correlation table (Table 31) provides a wholistic perspective on software failure and defect classes. To be used for quantification, data pertaining to the target software are still required, specifically, the occurrence probability of each defect class within the software (i.e., $F_i(t|s)$).

Table 33. Global correlations between software failure modes (UCA/UIFs) and orthogonal defect classes.

Defect Class	**UCA/UIF – A**	**UCA/UIF – B**	**UCA/UIF – C**	**UCA/UIF – D**
Assignment	0.288 ± 0.152	0.667 ± 0.130	0.045 ± 0.072	N/A
Checking	0.219 ± 0.074	0.539 ± 0.115	0.102 ± 0.072	0.141 ± 0.129
Algorithm	0.217 ± 0.051	0.525 ± 0.068	0.124 ± 0.044	0.134 ± 0.019
Function	0.250 ± 0.154	0.518 ± 0.064	0.157 ± 0.092	0.074 ± 0.216
Timing	0.095 ± 0.334	0.190 ± 0.289	0.524 ± 0.423	0.190 ± 0.289
Interface	0.262 ± 0.065	0.579 ± 0.086	0.093 ± 0.095	0.065 ± 0.039
Documentation	0.250 ± 0.250	0.250 ± 0.250	0.250 ± 0.250	0.250 ± 0.250
Relationship	0.250 ± 0.250	0.250 ± 0.250	0.250 ± 0.250	0.250 ± 0.250

$$P(\mathrm{UIF_A}) = \begin{bmatrix} P(\mathrm{UIF_A}|\mathrm{Alg.}) \\ P(\mathrm{UIF_A}|\mathrm{Asi.}) \\ P(\mathrm{UIF_A}|\mathrm{Fnc.}) \\ P(\mathrm{UIF_A}|\mathrm{Int.}) \\ P(\mathrm{UIF_A}|\mathrm{Chk.}) \\ P(\mathrm{UIF_A}|\mathrm{Tim.}) \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} F_{Alg}(t|s) \\ F_{Asi}(t|s) \\ F_{Fnc}(t|s) \\ F_{Int}(t|s) \\ F_{Chk}(t|s) \\ F_{Tim}(t|s) \end{bmatrix} \tag{36}$$

For high-reliability systems, it is already known from Reference [91], that the amount of testing effort required is intractable and therefore SRGMs are inapplicable. In addition, existing estimation methods depend on the observation of unreliability. However, highly reliable systems will inherently have low instances of unreliability which also makes the estimation highly uncertain. To address this issue, a rudimentary estimation method is implemented where the occurrence probability of a defect is assumed to be approximately constant and can be estimated using Equation (37) from Reference [91].

$$F_i = 1 - \exp\left(-\frac{n_i}{m}\right) \tag{37}$$

Here, F_i is the time-independent occurrence probability of the i^{th} defect class, n_i is the number of discovered defects of the i^{th} class, and m is the total number of tests conducted. This equation assumes that relevant and extensive validation and verification testing has already been completed and the product is ready for deployment. The implication of Equation (37) is that the source code cannot be defect free regardless of any amount of testing.

Stage 5: Assess efficacy of analysis.

The last stage of ORCAS is the qualification of the SDLC by evaluating the qualitative evidence derived from 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 is considered. The developers can then return to those software sections and conduct further testing.

APPENDIX C – BAYESIAN AND HRA-AIDED METHOD FOR THE RELIABILITY ANALYSIS OF SOFTWARE (BAHAMAS)

This appendix provides the steps in detail for performing BAHAMAS. Each step is elaborated subsequently as shown in Figure 40.

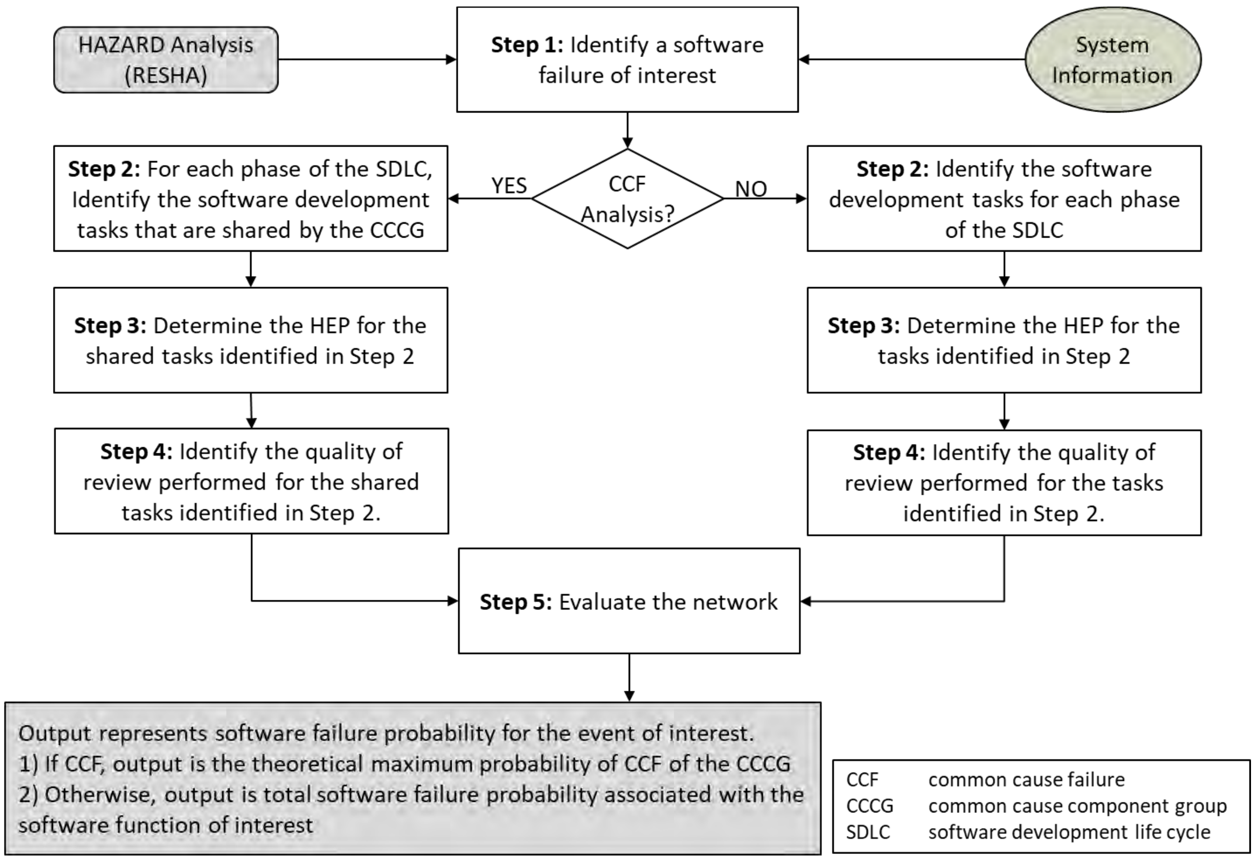

HAZARD Analysis (RESHA)
Step 1: Identify a software failure of interest
System Information
Step 2: For each phase of the SDLC, Identify the software development tasks that are shared by the CCCG
YES
CCF Analysis?
NO
Step 2: Identify the software development tasks for each phase of the SDLC
Step 3: Determine the HEP for the shared tasks identified in Step 2
Step 3: Determine the HEP for the tasks identified in Step 2
Step 4: Identify the quality of review performed for the shared tasks identified in Step 2.
Step 4: Identify the quality of review performed for the tasks identified in Step 2.
Step 5: Evaluate the network
Output represents software failure probability for the event of interest.
1) If CCF, output is the theoretical maximum probability of CCF of the CCCG
2) Otherwise, output is total software failure probability associated with the software function of interest
CCF common cause failure
CCCG common cause component group
SDLC software development life cycle

Figure 40. Workflow of the BAHAMAS in the proposed framework for the software reliability analysis of DI&C systems.

Step 1: Identify a software failure of interest.

This step is coupled with hazard analysis activities. Within the LWRS-developed framework, this connection is with RESHA for which postulated software failure events can be identified with different modes of failure UCA/UIF-A, B, C &D. The failure of interest may be found within an existing FT. In any case, the details of the failure of interest should be used to establish the overall scope of the quantitative assessment.

The failure of interest should be identified as an associated function of a piece of software. BAHAMAS is not intended to evaluate the failure probability of a system of software elements. BAHAMAS should be applied to a single software and its functions. The FT provided by RESHA allows for these larger interactions to be considered.

Step 2: Identify the software development tasks for each phase of the SDLC.

The entire focus of this step is to establish what tasks are planned or have been performed for the SOI and map them to the BBN. SDLC activities and their tasks are clarified by the standards, manuals, programs, plans, and policies followed by developers. If possible, the actual human-performed tasks for the software development should be noted. However, at early stages, the intended tasks will suffice. The human-performed tasks should be documented and grouped according to which stage of the SDLC they correspond. The stages selected for BAHAMAS are concept, design, implementation, testing, and installation & maintenance stages. Even though many standards and guidelines establish when a task is performed, the specifics vary by case. Each task, specific to the SOI, should be mapped to the stage they most directly correspond within the BAHAMAS network. For example, the human task of defining the project scope, or software interface requirements map to the concept and design stages, respectively. Once all tasks have been identified and mapped accordingly, the next step is to prepare the BBN for evaluation by assigning node probabilities. In summary for this step there are two action items:

- **Action Item for normal assessment:** Identify and list the critical or essential tasks of the SDLC and group them according to which phase of the SDLC they are performed.
- **Action Item for CCF assessment:** Identify and list the critical or essential tasks of the SDLC and group them according to which phase of the SDLC they are performed. For CCF analysis, only the tasks that are found to be common among the SDLC of each member of the CCCG should be identified.

Step 3: Determine the HEP for the tasks identified in Step 2.

Two primary concepts are employed by BAHAMAS, these are defect introduction and defect removal. Human errors account for defect introduction while review activities account for removal. This step addresses the defect introduction.

Each of the critical tasks that have been identified in Step 2 should be assessed for the HEP. To do so, the analyst should rely on HRA. Ultimately the HEP for each task identified will be used to inform the probability of defects in the software without considering review. This probability of defects is used to populate the root nodes: Stage Defects.

The process for defining defect nodes requires that critical activities of the SDLC stage be identified and then evaluated for human errors. Each SDLC stage has multiple activities consisting of multiple tasks. Each of these may introduce errors to the software. HRA is applied to determine the HEP of each critical task (*T*), the union of which provides an indication of the probability of defects for each stage (see Equation (38) below).

$$P(Stage\ Defects) = \sum_{i=1}^{T} (HEP)_i \tag{38}$$

Here it is important to note that HRA may provide coverage for review activities in the definition of the HEP (e.g., THERP does). However, BAHAMAS considers review separately in the network. Therefore, the HRA performed for this step accounts only for those actions and correction factors that pertain to the performance of the task at hand. No consideration of checking from teammates or independent reviews of the specific tasks should be considered here. The review activities will be considered in Step 4.

Regarding the CCF analysis, the need is to consider the overlapping or common tasks and their impact on the probability of defects for each stage of the SDLC. HRA is applied to determine the HEP of each critical task (*T*), the union of which provides an indication of the probability of defects for each stage for CCF analysis, only the common tasks should be assessed. And in the case of diverse software, only a subset of T may be common (i.e., T_{cc}). Additionally, the HEP from equation (38) should be replaced by HEP_{min}, corresponding to the smaller evaluated HEP of the task shared by the diverse CCCG. The result of this change is shown in equation (39). As an example, given a task that is shared by software-1 and software-2, the maximum overlap of the HEPs will be given by the smaller of the two. This follows the discussion in [26] concerning the upper bound on joint probability distributions.

$$P_{cc}(Stage\ Defects) = \sum_{i=1}^{T_{cc}} (HEP_{min})_i \tag{39}$$

- **Action Item for normal assessment:** Identify the HEP for each task identified in Step 2 and evaluate the probability of defects for each stage using equation (38).
- **Action Item for CCF assessment**: Identify the HEP for each common task identified in Step 2 and evaluate the probability of defects for each stage using equation (39) below.

Step 4: Identify the quality of review performed for the tasks identified in Step 2.

This step considers the efforts employed during the SDLC that effectively remove defects from the software. Essentially, the quality of review efforts indicates whether defects will remain in the software. For this work, quality of review is implied by the number of independent reviews and the specific review activities (i.e., trigger coverage) that are employed for the SDLC. Future work may incorporate other concepts.

First, the number of reviews employed for each task shall be assessed and recorded. These are then used to find the average number of reviews for a particular stage is defined by Equation (40), where *r* is the number of independent reviews performed for each critical task and *T* is the total number of critical tasks for a given stage. Equation (40) provides a wholistic view of the number of reviews for each SDLC stage.

$$R = \frac{1}{T}\sum_{i=1}^{T} r_i \tag{40}$$

Second, the specific review activities performed (i.e., trigger coverage) of each task shall be assessed. The ODC methodology indicates that activities during design review, codes inspection, and testing can help identify and remove defects [83]. 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 [83]. 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 [81]. 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(below)shows the triggers and their associated activities adapted from [83].

The trigger coverage (TC) is determined from the average of each task-level trigger coverage (tc) which is the percent of relevant triggers that have been covered for a task of a particular SDLC stage (see equation (41) below). Note that *T* is the total number of critical tasks for a given stage.

$$TC = \frac{1}{T}\sum_{i=1}^{T} (tc)_i \tag{41}$$

Table 34: Triggers and their associated stages of the SDLC

Review Activities	Triggers
	Design Conformance
	Logic/Flow
	Backward Compatibility

For Use (typically) with Concept, Design, and Implementation Review Activities	Lateral Compatibility
	Concurrency
	Internal Document
	Language Dependency
	Side Effect
	Rare Situations
For Use (Typically) with Testing Review Activities	Simple Path
	Complex Path
	Test Coverage
	Test Variation
	Test Sequencing
	Test Interaction
	Workload/Stress
	Recovery/Exception
	Startup/Restart
	Hardware Configuration
	Software Configuration
	Blocked Test (Previously Normal Mode)
Descriptions for each trigger can be found in 201 I “Orthogonal Defect Classification v 5.2 for Software Design & Code” [92]	

Consideration of this step for CCF analysis requires that the quality of the reviews be assessed with respect to the common or shared tasks that were identified in Step 2. This requires some modification of the general assessment equations.

The trigger coverage (TC) is determined from the average of each task-level trigger coverage (tc) which is the percent of relevant triggers that have been covered for a task of a particular SDLC stage (see equation(41)). Trigger coverage (TC_{cc}) for the CCF analysis, including diverse CCFs, is given by equation (42) and depends on the subset of tasks that are shared by the CCCG.

$$TC_{cc} = \frac{1}{T}\sum_{i=1}^{T_{cc}} (tc_{cc})_i \tag{42}$$

Within equation (42), the task-level trigger coverage is defined as a function of the set of triggers investigated ($\{tr\}_{i,inv}$) for the CCCG out of the total relevant set of triggers ($\{tr\}_{i,tot}$) that are applicable to a particular task. Equation (43) provides the relationship between $\{tr_{cc}\}_i$ and $\{tr\}_{i.s}$, where $\{tr_{cc}\}_i$ is the set of relevant triggers to the CCCG and $\{tr\}_{i,s}$ is the set covered by the SDLC of software s. The index K, is the total number of diverse software implementations used by the CCCG (for CCCG with only redundant software index $K = 1$). Equation (44) gives the task-level trigger coverage for the CCCG.

$$\{tr_{cc}\}_i = \left\{\{tr\}_{i,1} \cup \{tr\}_{i,2} \ldots \cup \{tr\}_{i,K}\right\} \tag{43}$$

$$(tc_{cc})_i = \frac{size\ of\ \{tr_{cc}\}_{i,inv}}{size\ of\ \{tr_{cc}\}_{i,tot}} \tag{44}$$

The average number of reviews for a particular stage is defined by equation (45), where *r* is the number of independent reviews performed for a specific task. For the diverse CCCGs, only the common or shared tasks should be assessed; thus, a subset of *T* is employed to determine *R*. The result is equation (46), where

r_{cc} represents a weighted average number of reviews for the CCCG. Equation (47) is given to show how r_{cc} is defined for each task where $r_{i,j,s}$ is the number of reviews performed during the SDLC of software *s,* for task i, that involved trigger j of the $\{tr\}_{i,s}$. Note, J_i corresponds to the size of $\{tr\}_{i,s}$.

$$R = \frac{1}{T}\sum_{i=1}^{T} r_i \tag{45}$$

$$R_{cc} = \frac{1}{T}\sum_{i=1}^{T_{cc}} (r_{cc})_i \tag{46}$$

$$(r_{cc})_i = \frac{1}{size\ of\ \{tr\}_{i,tot}} \sum_{j=1}^{J_i} {}_{s=1}^{K}max(r_{i,j,s}) \tag{47}$$

- **Action Item for normal assessment:** Identify the review number and trigger coverage for each task identified in Step 2 and evaluate the average number of reviews and trigger coverage for each stage of the SDLC.
- **Action Item for CCF assessment**: Identify the review number and trigger coverage for each common or shared task identified in Step 2 and evaluate the associated average number of reviews and trigger coverage for each stage of the SDLC.

A note on this Step, the root nodes of BAHAMAS that represent stage review are quite simply a binary assessment. The probability for these states is given as 1 or 0 dependent upon whether a review has occurred. This is necessary for the evaluation of the network. All other details determined for review quality are required and used for the conditional probability evaluations within BAHAMAS.

Step 5: Evaluate the Network

The final step is to incorporate all the details, if not already completed, into a conventional solver to evaluate the software failure probability. Depending on the details used the output of BAHAMAS will provide results for software failure probability or the software common cause failure probability.

Evaluation of the network requires that the root node probabilities and the conditional probabilities be known. The generic network for BAHAMAS is shown in Figure 41.

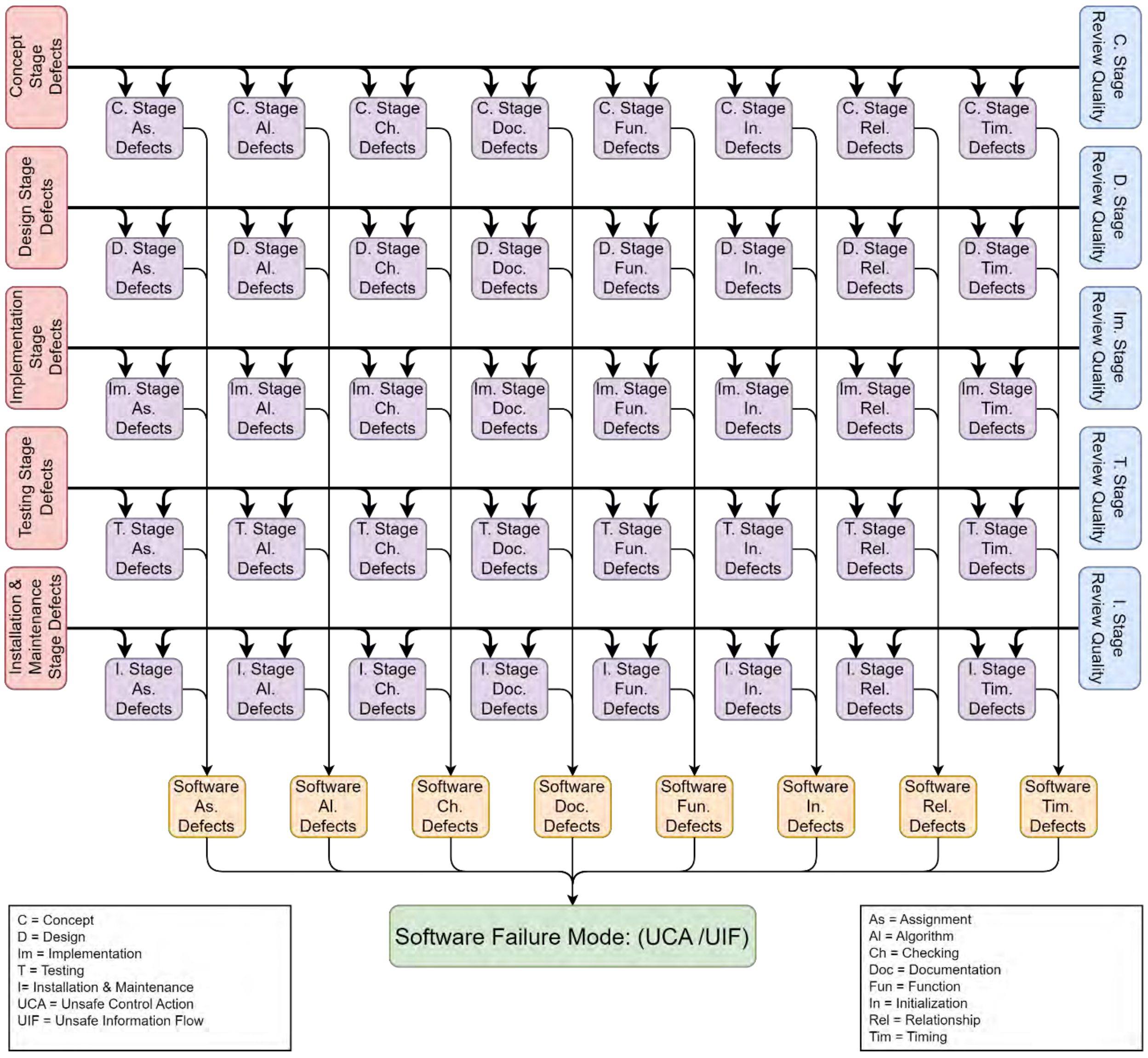


Figure 41. Generic BAHAMAS network.

Root node probabilities needed to evaluate BAHAMAS:

The root nodes include the stage defect nodes and the review nodes. Stage defect nodes are quantified using the equations given in Step 3. Review nodes are given as 1 or 0 depending on whether any reviews have occurred for the given stage (see final paragraph of Step 4).

Conditional Probabilities needed to evaluate BAHAMAS:

◦ *The conditional probability for specific defect types at each stage of the SDLC (shown as the purple nodes in Figure 41):*

The types of defects that are introduced vary throughout the SDLC. As the software development progresses, the distribution of defect types that remain within the software varies. 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 [30]. It assumed that the concentration of defect types changes during development. To support this assumption, we investigated a public database [93] (associated with [94]) 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. [30] given the code inspection. Table 35 provides the distributions for each defect type and SDLC stage. Future research may lead to adjustments in these distributions.

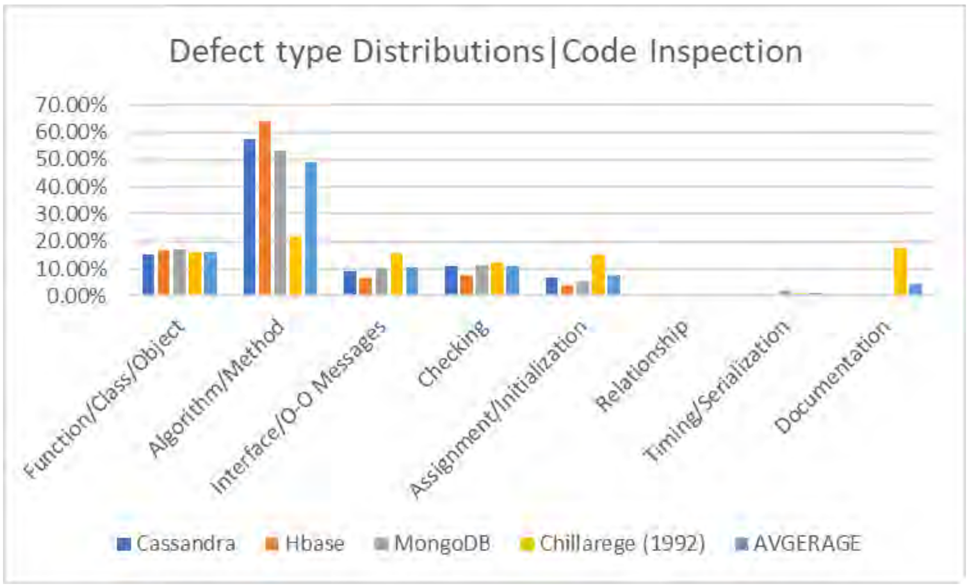


Figure 42. Defect type distributions given code inspection.

Table 35: 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.								

Table 35 provides a conditional probability of the existence of a particular defect type for a given phase of the SDLC. However, as mentioned in the review node section, the quality of review activities can have an influence on the existence of latent defects. 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 multiple reviews will increase the review quality and the quality of the delivered product.

This work considers the influence of recovery and HEP separately, whereas normally HRA may account for such in one go. For example, THERP can account for the benefit of review wherein the recovery is conditional upon the dependency between the original performer and reviewer. The more involved the reviewer, the greater potential for effective recovery from human errors. It turns out that

THERP's model for recovery follows a nearly exponential curve that is a function of the dependency relationships (i.e., review quality) between the reviewer and the number of reviews performed. Details are given in the THERP manual [95]. In this work, to achieve a similar recovery benefit, BAHAMAS incorporates the exponential function shown below. For convenience it is called the defect conditional probability (DCP) equation. Within the (DCP) equation, G is the general expected probability for a defect type for a given SDLC stage (from Table 35). TC represents the stage trigger coverage and is determined from equation (41). And R is the average number of reviews performed for each task of the SDLC stage as defined by equation (48).

$$DCP = \begin{cases} StageReview(all\ states)\ and\ StageDefect\ (yes), & DCP = Ge^{-(TC*R)} \\ StageReview(all\ states)\ and\ StageDefect(no), & DCP = 0 \end{cases} \tag{48}$$

- *Conditional probability for specific defect types at the end of SDLC (shown as the yellow nodes in Figure 41).*

The yellow nodes that represent defect types at the end of the SDLC have two states, either yes or no. The conditional probability for these states is dependent on whether specific defects have been found in any of the earlier phases. If yes, then the yellow nodes conditional probability is yes. Otherwise, it is no.

- *Conditional probability for software failure (shown as the green node in Figure 41).*

BAHAMAS follows the ORCAS methodology to evaluate software failure probability. A conditional relationship between the nodes representing defect type remaining and the probability of software failure mode is given by equation (49). Where I is the total number of defect types considered by BAHAMAS and UC_x is the UCA/UIF failure mode being evaluated.

$$P(software\ failure\ of\ mode\ x) = \sum_{i=1}^{I} P(UC_x|Type_i)P(Type_i) \tag{49}$$

Table 33 provides the conditional probabilities used for $P(UC_x|Type_i)$. These distributions have been conservatively assumed for the current work. In Table 33, is a refinement of the work from the previous FY [4]. Here the mean and two standard deviation confidence intervals of all class correlations to UCAs/UIFs have been added. The values and uncertainties for documentation and relationship defects have been assumed. As more classification is performed these distributions can be refined. The table provides a global perspective of the relationship between defect classes and UCA/UIF failure modes. Finally, all the necessary information has been provided for the BBN to be solved using conventional software tools. All the nodes have been defined along with their conditional probability values.