IAEA Nuclear Energy Series

No. NP-T-3.27

Dependability Assessment of Software for Safety Instrumentation and Control Systems at Nuclear Power Plants

IAEA
International Atomic Energy Agency

IAEA NUCLEAR ENERGY SERIES PUBLICATIONS

STRUCTURE OF THE IAEA NUCLEAR ENERGY SERIES

Under the terms of Articles III.A and VIII.C of its Statute, the IAEA is authorized to foster the exchange of scientific and technical information on the peaceful uses of atomic energy. The publications in the **IAEA Nuclear Energy Series** provide information in the areas of nuclear power, nuclear fuel cycle, radioactive waste management and decommissioning, and on general issues that are relevant to all of the above mentioned areas. The structure of the IAEA Nuclear Energy Series comprises three levels: **1 — Basic Principles and Objectives**; **2 — Guides**; and **3 — Technical Reports**.

The **Nuclear Energy Basic Principles** publication describes the rationale and vision for the peaceful uses of nuclear energy.

Nuclear Energy Series Objectives publications explain the expectations to be met in various areas at different stages of implementation.

Nuclear Energy Series Guides provide high level guidance on how to achieve the objectives related to the various topics and areas involving the peaceful uses of nuclear energy.

Nuclear Energy Series Technical Reports provide additional, more detailed information on activities related to the various areas dealt with in the IAEA Nuclear Energy Series.

The IAEA Nuclear Energy Series publications are coded as follows: **NG** — general; **NP** — nuclear power; **NF** — nuclear fuel; **NW** — radioactive waste management and decommissioning. In addition, the publications are available in English on the IAEA Internet site:

http://www.iaea.org/Publications/index.html

For further information, please contact the IAEA at PO Box 100, Vienna International Centre, 1400 Vienna, Austria.

All users of the IAEA Nuclear Energy Series publications are invited to inform the IAEA of experience in their use for the purpose of ensuring that they continue to meet user needs. Information may be provided via the IAEA Internet site, by post, at the address given above, or by email to Official.Mail@iaea.org.

DEPENDABILITY ASSESSMENT OF SOFTWARE FOR SAFETY INSTRUMENTATION AND CONTROL SYSTEMS AT NUCLEAR POWER PLANTS

The following States are Members of the International Atomic Energy Agency:

AFGHANISTAN
ALBANIA
ALGERIA
ANGOLA
ANTIGUA AND BARBUDA
ARGENTINA
ARMENIA
AUSTRALIA
AUSTRIA
AZERBAIJAN
BAHAMAS
BAHRAIN
BANGLADESH
BARBADOS
BELARUS
BELGIUM
BELIZE
BENIN
BOLIVIA, PLURINATIONAL STATE OF
BOSNIA AND HERZEGOVINA
BOTSWANA
BRAZIL
BRUNEI DARUSSALAM
BULGARIA
BURKINA FASO
BURUNDI
CAMBODIA
CAMEROON
CANADA
CENTRAL AFRICAN REPUBLIC
CHAD
CHILE
CHINA
COLOMBIA
CONGO
COSTA RICA
CÔTE D'IVOIRE
CROATIA
CUBA
CYPRUS
CZECH REPUBLIC
DEMOCRATIC REPUBLIC OF THE CONGO
DENMARK
DJIBOUTI
DOMINICA
DOMINICAN REPUBLIC
ECUADOR
EGYPT
EL SALVADOR
ERITREA
ESTONIA
ETHIOPIA
FIJI
FINLAND
FRANCE
GABON
GEORGIA
GERMANY
GHANA
GREECE
GRENADA
GUATEMALA
GUYANA
HAITI
HOLY SEE
HONDURAS
HUNGARY
ICELAND
INDIA
INDONESIA
IRAN, ISLAMIC REPUBLIC OF
IRAQ
IRELAND
ISRAEL
ITALY
JAMAICA
JAPAN
JORDAN
KAZAKHSTAN
KENYA
KOREA, REPUBLIC OF
KUWAIT
KYRGYZSTAN
LAO PEOPLE'S DEMOCRATIC REPUBLIC
LATVIA
LEBANON
LESOTHO
LIBERIA
LIBYA
LIECHTENSTEIN
LITHUANIA
LUXEMBOURG
MADAGASCAR
MALAWI
MALAYSIA
MALI
MALTA
MARSHALL ISLANDS
MAURITANIA
MAURITIUS
MEXICO
MONACO
MONGOLIA
MONTENEGRO
MOROCCO
MOZAMBIQUE
MYANMAR
NAMIBIA
NEPAL
NETHERLANDS
NEW ZEALAND
NICARAGUA
NIGER
NIGERIA
NORWAY
OMAN
PAKISTAN
PALAU
PANAMA
PAPUA NEW GUINEA
PARAGUAY
PERU
PHILIPPINES
POLAND
PORTUGAL
QATAR
REPUBLIC OF MOLDOVA
ROMANIA
RUSSIAN FEDERATION
RWANDA
SAINT VINCENT AND THE GRENADINES
SAN MARINO
SAUDI ARABIA
SENEGAL
SERBIA
SEYCHELLES
SIERRA LEONE
SINGAPORE
SLOVAKIA
SLOVENIA
SOUTH AFRICA
SPAIN
SRI LANKA
SUDAN
SWAZILAND
SWEDEN
SWITZERLAND
SYRIAN ARAB REPUBLIC
TAJIKISTAN
THAILAND
THE FORMER YUGOSLAV REPUBLIC OF MACEDONIA
TOGO
TRINIDAD AND TOBAGO
TUNISIA
TURKEY
TURKMENISTAN
UGANDA
UKRAINE
UNITED ARAB EMIRATES
UNITED KINGDOM OF GREAT BRITAIN AND NORTHERN IRELAND
UNITED REPUBLIC OF TANZANIA
UNITED STATES OF AMERICA
URUGUAY
UZBEKISTAN
VANUATU
VENEZUELA, BOLIVARIAN REPUBLIC OF
VIET NAM
YEMEN
ZAMBIA
ZIMBABWE

The Agency's Statute was approved on 23 October 1956 by the Conference on the Statute of the IAEA held at United Nations Headquarters, New York; it entered into force on 29 July 1957. The Headquarters of the Agency are situated in Vienna. Its principal objective is "to accelerate and enlarge the contribution of atomic energy to peace, health and prosperity throughout the world".

IAEA NUCLEAR ENERGY SERIES No. NP-T-3.27

DEPENDABILITY ASSESSMENT OF SOFTWARE FOR SAFETY INSTRUMENTATION AND CONTROL SYSTEMS AT NUCLEAR POWER PLANTS

INTERNATIONAL ATOMIC ENERGY AGENCY
VIENNA, 2018

Marketing and Sales Unit, Publishing Section
International Atomic Energy Agency
Vienna International Centre
PO Box 100
1400 Vienna, Austria
fax: +43 1 2600 29302
tel.: +43 1 2600 22417
email: sales.publications@iaea.org
http://www.iaea.org/books

Printed by the IAEA in Austria
July 2018
STI/PUB/1808

IAEA Library Cataloguing in Publication Data

Names: International Atomic Energy Agency.
Title: Dependability assessment of software for safety instrumentation and control systems at nuclear power plants / International Atomic Energy Agency.
Description: Vienna : International Atomic Energy Agency, 2018. | Series: IAEA nuclear energy series, ISSN 1995–7807 ; no. NP-T-3.27 | Includes bibliographical references.
Identifiers: IAEAL 18-01172 | ISBN 978–92–0–101218–0 (paperback : alk. paper)
Subjects: LCSH: Nuclear power plants — Instruments. | Nuclear reactors — Control. | Digital control systems.
Classification: UDC 621.039.56 | STI/PUB/1808

FOREWORD

One of the IAEA's statutory objectives is to "seek to accelerate and enlarge the contribution of atomic energy to peace, health and prosperity throughout the world." One way this objective is achieved is through the publication of a range of technical series. Two of these are the IAEA Nuclear Energy Series and the IAEA Safety Standards Series.

According to Article III.A.6 of the IAEA Statute, the safety standards establish "standards of safety for protection of health and minimization of danger to life and property". The safety standards include the Safety Fundamentals, Safety Requirements and Safety Guides. These standards are written primarily in a regulatory style, and are binding on the IAEA for its own programmes. The principal users are the regulatory bodies in Member States and other national authorities.

The IAEA Nuclear Energy Series comprises reports designed to encourage and assist R&D on, and application of, nuclear energy for peaceful uses. This includes practical examples to be used by owners and operators of utilities in Member States, implementing organizations, academia, and government officials, among others. This information is presented in guides, reports on technology status and advances, and best practices for peaceful uses of nuclear energy based on inputs from international experts. The IAEA Nuclear Energy Series complements the IAEA Safety Standards Series.

The focus of this report is the assessment of the dependability of software for safety instrumentation and control systems in nuclear power plants. These plants are currently in the process of upgrading many analogue systems to new software based digital systems. The assessment of the software's dependability, which encompasses properties such as safety, reliability, availability, maintainability and security, is an essential and challenging aspect of the safety justification. At their 2013 meeting, the members of the Technical Working Group on Nuclear Power Plant Instrumentation and Control recommended that guidance be developed on this subject.

This report defines a framework that represents the state of the art in software assessment methodologies and describes an approach to developing and communicating assessments based on claims, arguments and evidence. Guiding principles for a dependability assessment are established to provide the basis for defining an assessment strategy and implementing the assessment process. Examples of techniques for generating evidence to support dependability claims are provided, and lessons learned from past digital instrumentation and control system implementation experience in areas such as software development, operational usage, regulatory review and platform certification are described.

The primary intent of this report is to provide a starting point for Member States to develop or improve their dependability assessment methods and guidance. The report may be particularly useful to research and development and design and technical support organizations, as well as regulatory authorities, by providing scientific background support to their activities.

This publication was produced by a committee of international experts and advisors from numerous Member States (the contributors are listed at the end of the report). The chair of the report preparation meetings was R. Wood (United States of America). The IAEA wishes to thank all participants and their Member States for their valuable contributions.

The IAEA officer responsible for this publication was J. Eiler of the Division of Nuclear Power.

CONTENTS

1. INTRODUCTION

1.1. BACKGROUND

Although the majority of reactor units built before the 1990s relied on analogue instrumentation and control (I&C) systems, digital technologies are currently playing an increasing role in the I&C systems of nuclear power plants (NPPs). Digital safety systems were introduced gradually, either as part of modernization projects at existing reactors or in the initial design of Generation III+ reactors. Currently, all new designs depend in large part on digital I&C systems and their software, and most I&C upgrades at existing units rely on digital technology.

The assessment of the dependability of software in NPP safety systems is an essential and challenging aspect of the safety justification for digital systems. Dependability is defined as the ability of the safety software to deliver a service that can be trusted and includes properties such as safety, reliability, availability, maintainability, integrity and security. Furthermore, the dependability of a system is defined as its ability to avoid service failures that are more frequent and more severe than is acceptable [1].

A difficulty that arises in applications of software important to safety is establishing what is acceptable and how trustworthy it is. In contrast to analogue hardware failures that are often a result of degradation over time, software failures are often the result of residual design errors. System developers do their best to avoid, detect and remove design errors, but in practice, it is impossible to justify a claim that software is completely error free. Residual design errors can arise at any phase of the software life cycle, from definition of requirements to maintenance/modification during its installed life. In addition, in the case of incorrect behaviour, there might be a risk of common cause failure (CCF) that could defeat redundancy or defence in depth. In some industries, the volume of software and its use make it easier to detect these errors. However, owing to the small amount of equipment used in nuclear safety I&C systems (compared with other industries), there is a need to use industrial standard devices and commercial off the shelf (COTS) products in the safety I&C systems of NPPs and in low volume specialty applications that require specific assessment procedures.

The licensing processes for various digital I&C system designs are often not the same across Member States and require different sets of documents. In addition, regulatory rules have been found to be vague or unclear and may change during an I&C project period. The use of an assessment framework, such as the one described in this report, should help to overcome these issues. To provide adequate confidence in the quality and dependability of safety critical software, academia and engineers have performed extensive work on software verification and assessment techniques. This has resulted in steady progress that can be put into practice. However, not all issues have been completely resolved. There is still no scientific or regulatory consensus on the quantification of software dependability for high quality digital systems, and even though formal verification can be used in some cases to ensure freedom from certain types of errors, it cannot yet guard against all errors (e.g. faults can arise from hardware/software interaction).

In this report, the dependability assessment of software in safety systems is based on a set of guiding principles and follows a strategy that is property and vulnerability based and standards informed. A dependability assessment involves gathering and organizing the evidence to support a particular claim or set of claims. The assessment provides the basis for an evaluation (e.g. by a regulator or other stakeholder) of whether the I&C system and its software are safe and support safe operation. Depending on the regulatory structure, claims might be developed or fixed by the regulatory requirements. Evaluation is the final judgement concerning the suitability of the I&C system and its software to fulfil its stated objectives.

This publication presents an assessment framework based on the following:

— A set of principles;
— An overall strategy to guide the assessment that considers the behaviour of the system as well as its interactions, vulnerabilities and compliance with standards;
— An approach to developing and communicating the assessment based on claims, arguments and evidence (CAE);
— Guidance on a high level process for deploying the framework with guidance on specific issues.

The assessment principles (APs) are the following:

— AP1: System and software requirements are adequately defined, valid and reviewable such that it can be confirmed that they collectively address the necessary functional and non-functional properties (Sections 3.3, 3.4.1.2, 3.5, 4.4 and 4.5).
— AP2: An effective understanding of the role of the safety I&C system is demonstrated, including its direct and possible indirect roles, and also its role in supporting safe operation (Sections 3.3, 3.5 and 4.4.4).
— AP3: The intended and unintended behaviour of the system is adequately taken into account, including adverse unintended consequences under conditions of failure (Sections 3.3, 4.3 and 4.4).
— AP4: The implementation of the requirements is adequate (Sections 3.3, 3.4.1.2, 3.5, 4.4 and 4.5).
— AP5: Active challenge to the dependability assessment is part of decision making throughout the organization (Sections 3.5, 4.2.3 and 4.6).
— AP6: The objectives of the users of the assessment are identified such that the structure and presentation of the assessment account for their needs (e.g. considering all users/uses and factors such as reviewability and maintainability of the assessment) (Section 3.4.1.1).
— AP7: The findings of the assessment are organized in a logical, coherent, traceable and accessible manner and are repeatable, with rigour commensurate with the degree of trust required of the system (Sections 3.2 and 3.4).
— AP8: Lessons learned are incorporated in the target system being assessed and in the assessment process itself (Sections 3.1.3, 3.1.4, 3.4.1.4, 4.1, 4.2.1 and 5).
— AP9: Any changes in the target system or conditions of use/maintenance that would invalidate the assessment are detected and addressed (Sections 4.2.1 and 4.2.3). Fundamental assumptions underlying the assessment are identified (Section 3.4).

This publication focuses on the assessment of dependability: the principles are neutral with regard to the design and implementation approach being taken to allow for innovation and flexibility. Conformity with the principles listed above would normally imply that among other things:

— System and software requirements are reviewable and adequately clear, concise, consistent, correct and complete.
— The overall architecture of the system design has been considered to ensure adequacy from a plant and system safety perspective (i.e. considering factors such as defence in depth, redundancy, independence and diversity).
— Prevention and protection measures against different sources of failure (e.g. inadequate design, physical degradation of hardware) are in place.
— Self-checks and defensive actions at both system level and module or function level are confirmed to be adequate (i.e. independent defences exist and are designed based on the assumption that the software will fail).
— There is sufficient overall confidence in the design as demonstrated through the consideration of safety significance and possible uncertainties, as well as other factors such as fault coverage, design complexity, built-in diagnostics and defences, reviewability, life cycle maintainability, ‘proven-ness’ and stability.
— Temporal properties such as determinism and performance under all conditions are adequately considered.

The cross-references in the list of APs above indicate where the application of these principles is addressed in this report. In applying these principles, the assessment should be embedded in a disciplined and systematic process. This is needed to ensure that the evidence used in the assessment is trustworthy and that the assessment is properly managed and does not lead to project risks. The principles apply to the dependability assessment of the software. However, there may be additional principles for licensing that may require simplicity of design, adherence to standards or organizational independence.

1.2. OBJECTIVE

The objective of this report is to provide an overview of the current knowledge, best practices, experience, benefits and challenges regarding the evaluation and assessment of software used in NPP safety I&C systems. The report is intended for use by Member States in the design, development, verification and validation of these systems to enable evaluation of the dependability of their software.

Guidance provided here, describing good practices, represents expert opinion but does not constitute recommendations made on the basis of a consensus of Member States.

1.3. SCOPE

This report covers relevant aspects of software evaluation and dependability assessment for digital safety I&C systems in NPPs. The information presented here may be useful to support decisions in new plant designs and in modernizations of existing operating NPPs. The report expands on the more general guidance found in IAEA Safety Standards Series publications, particularly SSG-39, Design of Instrumentation and Control Systems for Nuclear Power Plants [2].

The scope of this report addresses software for NPP safety systems. This refers to software used in Class 1 systems performing Category A (Cat. A) functions in the context of International Electrotechnical Commission (IEC) standards and 'safety related systems' standards from the Institute of Electrical and Electronics Engineers (IEEE). Such software has the highest level of requirements and the greatest need for rigorous assessment. Much of the information presented in this report would also be relevant to systems that are placed in lower classes based on the requirements of lower grade systems and a lower need for rigorous assessment.

In addition, this report is intended to provide a starting point for Member State regulators, independent assessors and system or software developers from which they can develop or improve their dependability assessment methods and guidance. It provides an assessment framework that can be deployed within different regulatory regimes. It enables a country specific emphasis on sources of evidence that support overall dependability claims. For some Member States, this will provide a framework for strengthening their methods for gathering and assessing evidence of software dependability. For others that use a system that stresses conformity with a particular set of design and/or development criteria, usually enforced by standards, this report will provide additional methods to show conformity with the criteria or ways to structure evidence to support requests for alternatives to particular criteria.

1.4. STRUCTURE

This report contains six main sections. Section 1 describes the background, objective and scope of work.

Section 2 establishes the system context for dependability and describes concepts relevant to the understanding of the role of software in achieving dependable functions. Specifically, the section discusses what constitutes software in this report and how its contribution to the dependability of a system function is characterized.

Section 3 defines an assessment framework suitable for assessing the contribution of software to dependability of a safety system. It provides a set of APs, including an overall strategy, that considers the behaviour of a system as well as its interactions, vulnerabilities and compliance with standards, an approach to developing and communicating the assessment based on CAE, and a high level process for deploying the framework with guidance on specific issues.

Section 4 documents examples of techniques of generating evidence, including operational experience, compliance with standards, validation, software analysis techniques, testing and review, that may be used to support dependability claims.

Section 5 describes lessons learned from past experience. It focuses on experience from software development, operational use, regulatory review and platform certification. In addition, it briefly discusses experience drawn from applications in other industries.

Section 6 provides conclusions of this report.

The publication contains five annexes. Annex I summarizes the quantification of software aspects of system reliability. Annex II provides an overview of licensing experience from recent projects. Annex III gives examples of vulnerabilities and challenges based on nuclear industry experience. Annex IV contains examples of the application of elements of the assessment framework, and Annex V shows a table of techniques for generating evidence in the evaluation process.

2. CONTEXT AND CONCEPTS FOR DEPENDABILITY

2.1. SYSTEM CONTEXT

The I&C system architecture and plant operations personnel serve as the 'central nervous system' of an NPP. Through its various constituent elements (e.g. equipment, hardware and software modules, subsystems, redundancies, systems, etc.), the plant I&C system senses basic parameters, monitors performance, integrates information and makes automatic adjustments to plant operations as necessary. It also responds to failures and abnormal events, thus ensuring the goals of efficient power production and safety are met. Essentially, the purpose of I&C systems at an NPP is to enable and support safe and reliable power generation.

A detailed discussion of NPP I&C systems appears in IAEA Nuclear Energy Series No. NP-T-3.12, Core Knowledge on Instrumentation and Control Systems in Nuclear Power Plants [3]. However, certain high level concepts are discussed in this section to establish a context for the role of software in implementing NPP I&C functions and its impact on the dependability of those functions.

Although NPP I&C systems can be characterized in different ways (e.g. safety category, physical architecture, function, design life cycle), a function based model enables the role of I&C technology in achieving plant operational and safety goals to be clearly seen. A high level view of I&C functions in an NPP focuses on plant wide system objectives and the means of achieving those objectives. Such a function based representation addresses the sensory, communications, monitoring, display, control and command systems interposed between the process (the reactor, heat transport, and energy conversion systems) and the plant personnel (operations and maintenance staff). Figure 1 illustrates the role of I&C systems within an NPP.

At a fundamental level, a system is a set of interconnected elements constituted according to a design to achieve a given objective of carrying out a specified function. As defined in IEC Standard 61513 [4], an I&C system is a "system, based on electrical and/or electronic and/or programmable electronic technology, performing I&C functions as well as service and monitoring functions related to the operation of the system itself".

The definition specifies that an I&C system "encompasses all elements of the system such as internal power supplies, sensors and other input devices, data highways and other communication paths, interfaces to actuators and other output devices".

As is clear from the definitions, the role of an I&C system depends on the functions assigned to it and their importance to plant objectives.

The functionality of an I&C system describes the extent to which one or more functions are performed, and the nature of those functions. The functionality of a system depends on the range of functions provided, the capability to execute the functions in real time (or at the required time), and the flexibility to select and implement the necessary functions if and when they are required. A function refers to a "specific purpose or objective to be accomplished, that can be specified or described without reference to the physical means of achieving it" (see IAEA-TECDOC-1140 [5]). An I&C function is defined as a "function to control, operate and/or monitor a defined part of the process" [4]. Typical I&C functions include the following:

— Process interface functions;
— Logic and control functions to make decisions;
— Data processing functions required to achieve the system mission;
— Functions related to communications between modules or to other systems;
— Human interface functions;
— Interface functions to operate plant systems or alarms or to provide display information.

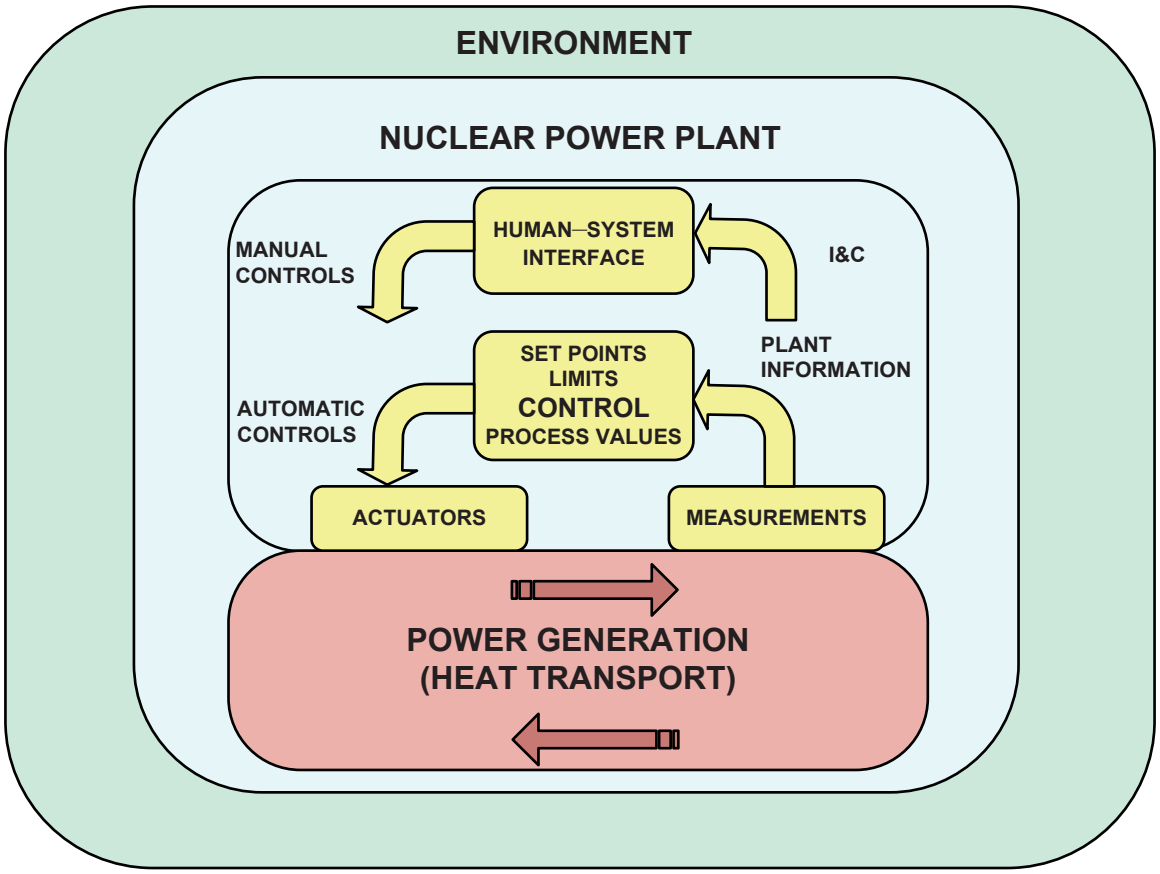


FIG. 1. High level role of I&C within an NPP. (Reproduced from IAEA NP-T-3.12 [3].)

As noted, the targeted systems for which dependability assessment of software is addressed in this report are safety systems. A safety system is defined in the IAEA Safety Glossary [6] as: "A *system* important to *safety*, provided to ensure the safe shutdown of the reactor or the *residual heat* removal from the core, or to limit the consequences of *anticipated operational occurrences* and *design basis accidents*."

In the context of NPP I&C systems, safety systems embody those high level functions identified in the listing of functions on the left side in Fig. 2. These safety systems have historically been implemented using analogue technology. However, these functions are more frequently being implemented using digital technology through modernization programmes at existing NPPs or within digital I&C architectures at new NPPs. Consequently, safety systems and their safety functions have been incorporated in programmable digital devices which rely on software instructions or programmable logic to accomplish a function. Examples of programmable digital devices include a computer, a programmable hardware device or a device with firmware. Systems using these devices are referred to as digital I&C systems.

With regard to the dependability of functions in a plant system context, software has both advantages and disadvantages. The disadvantages are related to possible design and implementation errors. Such errors are difficult if not impossible to rule out completely. Malicious attacks on software are also a concern. Advantages include the ability to provide extensive monitoring capabilities, allowing early error or failure detection (drifting sensors; failed sensors, actuators or other hardware components; incorrect system operational states; corrupted data; inconsistent human requests) and fault tolerance capabilities (management of component failures, data corruption and redundancy; support for algorithmic diversity).

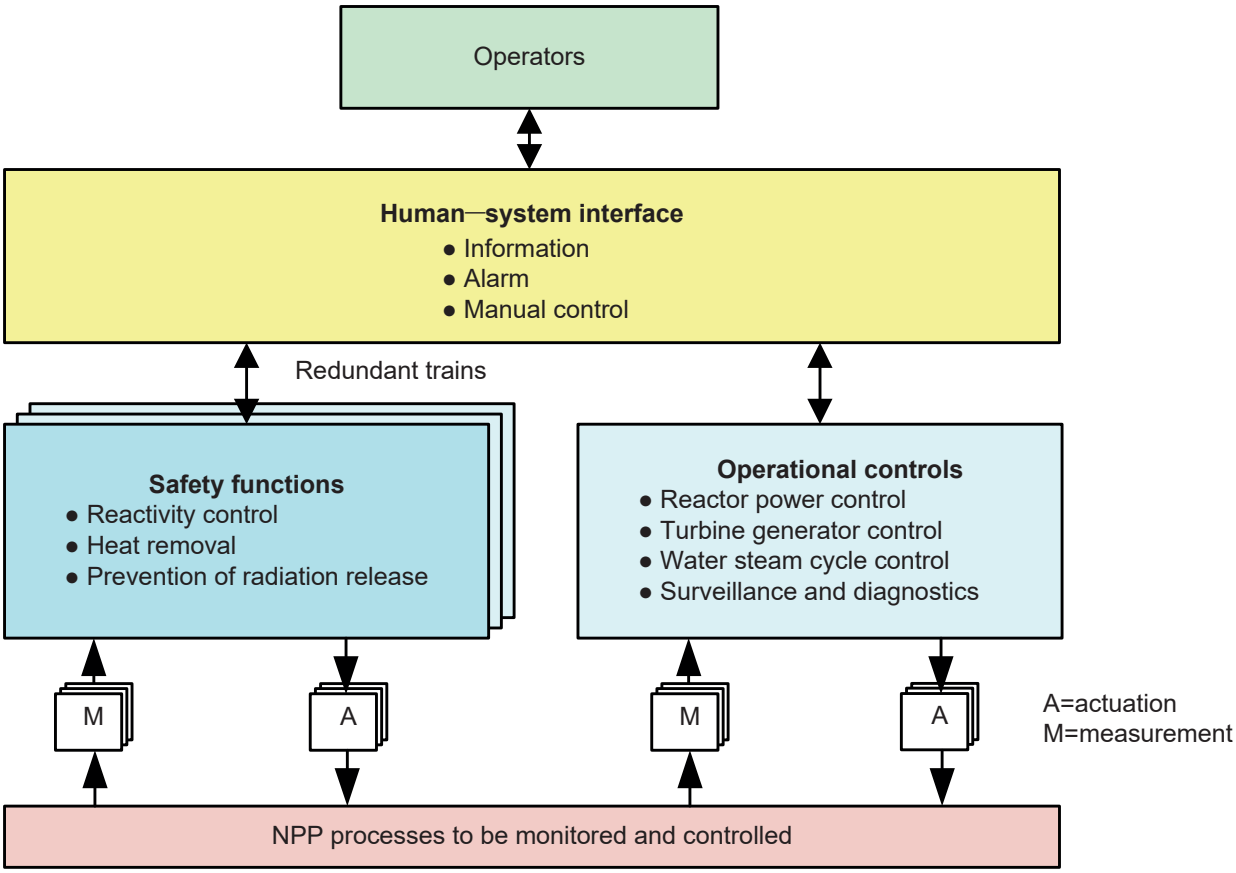


FIG. 2. Overview of NPP I&C functions. (Reproduced from IAEA NP-T-3.12 [3].)

2.2. DEFINITION OF THE TERM 'SOFTWARE'

Software is defined by the International Organization for Standardization (ISO) in ISO/IEC 2382 [7] as "all or part of the programs, procedures, rules, and associated documentation of an information processing system".

This definition includes executable software as well as related software, firmware and documentation (e.g. requirements, design, user manuals, etc.) and data.

Software in the scope of this report is composed mainly of a sequence of instructions executed on a central processing unit (CPU), the logical structure of massive parallel logic devices such as field programmable gate arrays (FPGAs) or programmable logic devices, as well as all combinations that may be implemented in an I&C system. The software also comprises all data determining the execution of calculations in the I&C system.

Documentation and the software of supporting equipment (e.g. development tools and other tools) are taken into account as far as they have an impact on the dependability of the software executed in the I&C system and as potential sources of evidence related to dependability attributes.

It should be emphasized that equipment with functionality that depends on FPGAs or similar integrated, programmable circuits are understood as 'software based' items. This approach is in line with IEC 62566 [8], which provides guidance for design, verification and validation, and application of hardware description language (HDL) programmed devices.

2.3. TYPES OF SOFTWARE

Software of interest to safety I&C systems can be classified according to the following taxonomy (see Fig. 3).

The first divide in the taxonomy is the on-line/off-line criterion. On-line software executes through I&C systems and has a direct effect on plant operation. Off-line software (e.g. software tools) executes through special

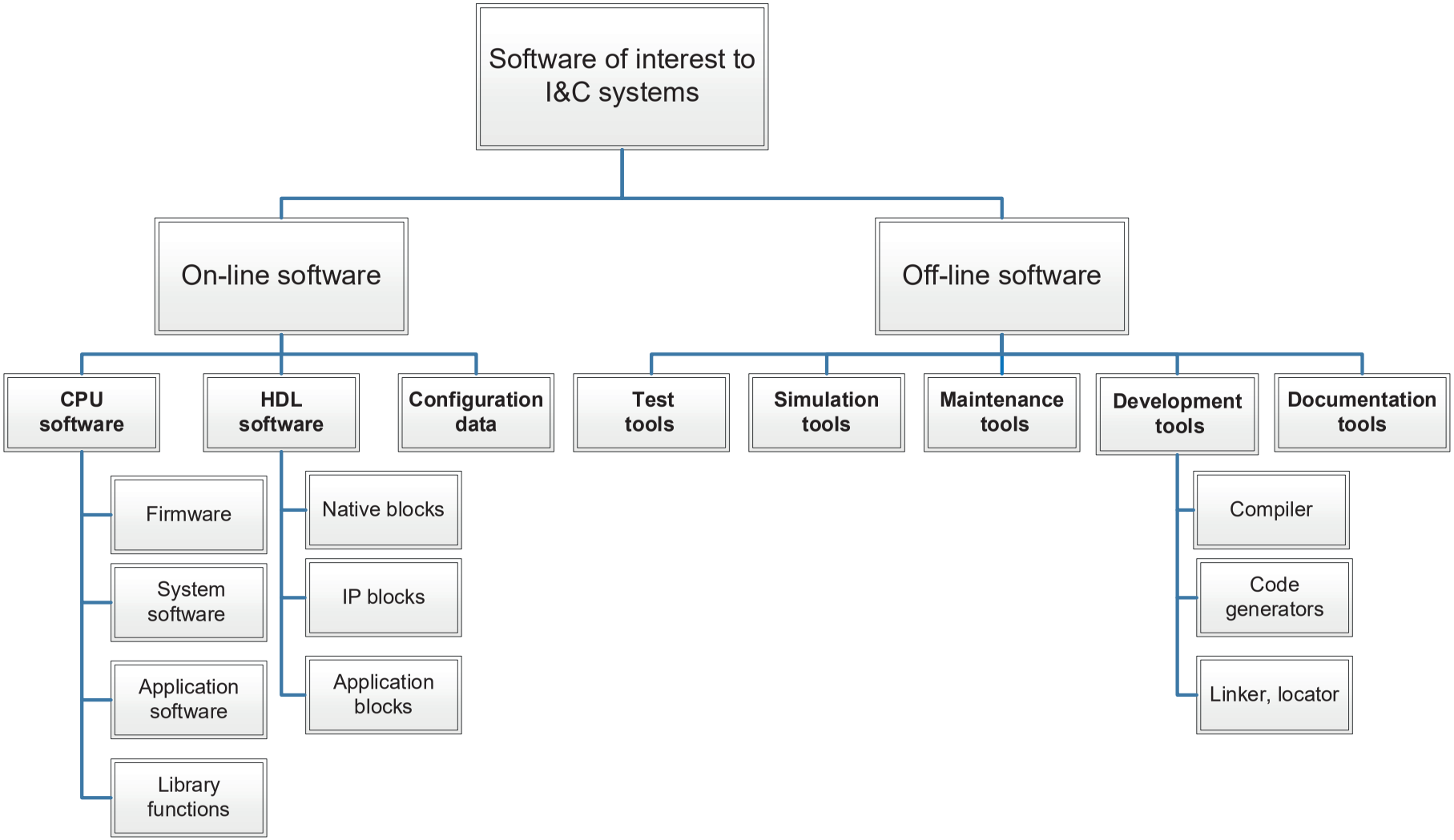


FIG. 3. Types of software for implementing NPP I&C functions.

purpose functional units of the I&C system or on systems not connected to the plant and has only an indirect effect on plant operation. Further taxonomic decomposition of off-line software is not within the scope of this publication.

On-line software can be further divided according to a type criterion, i.e. CPU code, HDL code and configuration data. Examples of configuration data include set points and gains for controls.

CPU code can be further divided into components with different roles: firmware (e.g. software of smart devices), system software (e.g. scheduler, input/output drivers, communication software, monitoring software, etc.), library functions and application software (performing functions specific to the plant).

HDL code can be subdivided into components of different types, often called blocks. Native blocks are intrinsic parts of the FPGA circuit. Intellectual property blocks are predeveloped blocks performing generic functions. Application blocks perform functions specific to the plant.

Components in CPU and HDL code can either be custom designed or predeveloped. Typically, firmware, system software, library functions, native blocks and intellectual property blocks are predeveloped. Configuration data, application software and application blocks are generally custom designed.

2.4. FAULTS AND FAILURES

It is important in dependability analyses to distinguish between faults and failures. A fault is defined as: a defect or "abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function" (IEC 61508 [9]). IEC 61513 [4] defines a fault as a "defect in a hardware, software or system component". The adoption of the general definition for fault leads to a software specific definition as an "incorrect step, process, or data definition in a computer program (called also software development/implementation error)" (ISO/IEC 25040 [10]).

A failure is defined as "termination of the ability of a product to perform a required function or its inability to perform within previously specified limits" (ISO/IEC 25000 [11]). A systematic failure is defined as failure related in a deterministic way to a certain cause, which can only be eliminated by a modification of the design or change in the manufacturing process, operational procedures, documentation or other relevant factors [9]. It is noted that 'failure' is an event, as distinguished from 'fault', which is a state.

IAEA Nuclear Energy Series No. NP-T-1.5 [12] states that a "failure is the result of the activation of a fault by a triggering event," and defines a triggering mechanism as a "Specific event or operating condition that causes structures, systems or components to fail due to a latent fault." The relationship between 'fault' and 'failure' is illustrated in Fig. 4. As illustrated, a fault is a defective state of the system or software that is caused by an error. In this publication, 'error' is used to represent a source of a fault. Errors are equated to mistakes (human errors) or deficiencies (design errors) and the coverage of fault sources is extended to include life cycle processes as well as actions or conditions. The primary concerns in the context of software dependability are latent or undetected faults that, when triggered or activated, result in a failure of the system or software. The effect of a failure can further propagate within the system or to interconnected or dependent systems.

Software failures are primarily systematic. Because they are associated with causes that were present when the software was introduced (latent faults), replication of the triggering conditions (a combination of inputs and the internal state of the software) will systematically create the same failure condition. However, sometimes the triggering conditions are random. This is the case, for example, when a software fault can be triggered by a random hardware fault.

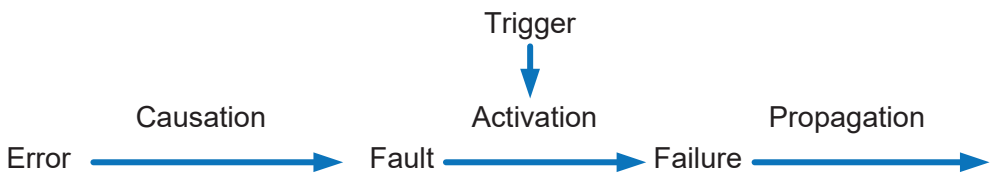


FIG. 4. Relationship between fault and failure.

2.5. NATURE OF SOFTWARE FAILURES

Hardware is subject to random failure due to manufacturing defects, ageing, wear or environmental effects. Because of this characteristic, it is possible to use hardware component failure data, together with operational profile forecasts, to determine credible estimates of the probability of failure for hardware devices composed of multiple hardware components. Software is not subject to ageing effects, and if it were possible to write perfectly correct software, it would operate correctly indefinitely. However, experience has shown that software is susceptible to failure over time for several reasons:

(1) The software works as required, but in specific conditions it does not function as expected owing to incorrect analysis during the requirements definition/capture phase.
(2) The software works as required, but the operational environment has changed, e.g. owing to modifications in plant equipment or operator procedures.
(3) The software was designed incorrectly, resulting in latent design errors which were not detected by verification and validation or confidence building measures performed prior to operational service; these errors result in failure during the operational life of the software.

In most cases, owing to its complexity, software cannot be proven to have been unaffected by any of these three mechanisms. Thus, there remains uncertainty regarding possible residual faults. This uncertainty increases concern for CCFs in systems that have common or similar software.

2.6. COMMON CAUSE FAILURE

The IAEA defines CCF as a "*Failure* of two or more *structures, systems and components* due to a single specific *event* or cause" [6]. In the comparable IEC definition, it is further noted that the:

> "coincidental failure of two or more structures, systems or components is caused by any latent deficiency from design or manufacturing, from operation or maintenance errors, and which is triggered by any event induced by natural phenomenon, plant process operation or action caused by man or by any internal event in the I&C system" [13].

CCF is a class of dependent failures, and its probability cannot be expressed as the simple product of the unconditional probabilities of the individual failures. Common mode failure, which occurs when two or more systems or components fail in the same way, is considered to be a subset of CCF.

The potential for software related CCF, and its impact, must be understood in the context of the I&C system architecture, which IEC 61513 [4] defines as the "organizational structure of the I&C systems of the plant which are important to safety". NPP safety systems are based on design principles that include high quality, integrity, reliability, independence and qualification. In addition to physical barriers and electrical isolation, separation and redundancy are generally applied as design measures to address the potential impact of a single failure of equipment and the propagation of failure effects. Consequently, shared components and non-essential interconnections are minimized within the architecture of the I&C.

Even with these design conventions, the potential for CCF vulnerability remains a concern. In response, analyses of diversity and defence in depth are conducted to assess the extent to which CCF mitigation should be incorporated as a contributing factor in satisfying safety requirements. Greater consideration of CCF mitigation strategies in digital I&C architectures results from the recognition that complex software within multiple digital systems may be subject to common systematic failures arising from concurrent activation of postulated latent faults.

The mechanisms resulting in software CCF involve potential sources of faults, triggering mechanisms and propagation of failure among systems or system components. A detailed technical discussion of these mechanisms is provided in an IAEA publication on CCFs in the digital I&C systems of NPPs [12]. In addition, the report discusses approaches to assess CCF vulnerability, I&C design measures to address CCF, and the rationale for a decision on what measures to employ for mitigation of CCF vulnerability.

Assessment of CCF vulnerability and determination of what strategies provide adequate CCF mitigation (i.e. what constitutes the necessary and sufficient combination of mitigation techniques) are generally subjective in nature. Assessment of software dependability, especially any quantitative measures, can serve to support conclusions about the resolution of CCF vulnerability.

2.7. DEPENDABILITY ATTRIBUTES

This section provides definitions for dependability and the various attributes deemed to influence dependability. These definitions are extracted from existing standards.

Dependability can be defined as a property: "the overall trustworthiness of a *system*; i.e. the extent to which reliance can justifiably be placed on this *system*. *Reliability*, *availability* and *safety* are attributes of *dependability*" [6].

Other relevant attributes include maintainability and security.

Dependability is initially a system concept, not a software concept. In this report, software dependability is defined as the extent to which reliance can justifiably be placed on software, in the framework of the system architecture or, more specifically, in the context of the system function.

The architecture defines the overall structure of the system, e.g. in terms of functional units (processing units, instrumentation, human system interfaces), communication links between functional units, characteristics of functional units and communication links, and distribution of processing (and software) among functional units. This reference to the system architecture is necessary, since software alone has no behaviour: important dependability attributes such as reliability and response times cannot be assessed on software only, independently of the architecture.

2.7.1. Safety

IAEA Safety Standards Series No. SF-1, Fundamental Safety Principles [14], defines safety as: "the protection of people and the environment against radiation risks, and the safety of facilities and activities that give rise to radiation risks."

Some systems implement functions that maintain or enable safety, and the correctness of their software provides for safe operation. This contrasts with systems whose operation does not provide a safety function, but whose failure could cause a hazardous event.

2.7.2. Reliability

Reliability is defined as: "The probability that a *system* or *component* will meet its minimum performance requirements when called upon to do so" [6]. It may be specified under stated conditions and for a specified period of time. The stated conditions are typically embedded in an operational profile that characterizes the statistical usage of the system.

Here again, reliability is initially a system concept, not a software concept. In this report, software reliability is defined as the probability that software, in the framework of the system architecture, will meet its minimum performance requirements when called upon to do so.

2.7.3. Availability

Availability is a system concept. The definition used in this report is that of operational availability, which is defined as the probability that an item will operate satisfactorily at a given point in time when used in an actual or realistic operating and support environment. For on-demand systems, the definition can be reinterpreted as the probability that an item is in a specified operable and committable state at the start of a mission, when the mission is called for at an unknown (i.e. random) time.

This applies to any component level. The unavailability of a component may reduce the availability, reliability or safety of the overall system.

Another aspect of availability of interest to the dependability of safety I&C systems is how the use of software may affect (e.g. increase or decrease) the availability of the function of the system, such as in the case of software used to monitor safety equipment status or software used in smart sensors.

2.7.4. Maintainability

The maintainability property can include consideration of the software contribution to system maintainability and to the maintainability of the software itself. In the framework of this publication, system maintainability is mainly related to the ease with which the I&C system or component can be restored to an 'as designed' state after degradation has occurred, e.g. owing to random hardware failure mechanisms or human error.

Software maintainability is a property related to the "ease with which a software system or component can be modified to change or add capabilities, correct faults or defects, improve performance or other attributes, or adapt to a changed environment" [15].

Software maintainability is important in the context of the development process where faults must be located and fixed. It can also be helpful in the adaptation of the software to modified architectures.

2.7.5. Security

Computer security is the main focus of security in this report, but it needs to be assessed in the framework of the other security measures taken to prevent, detect, delay and respond to malicious acts as well as to mitigate the consequences of such acts. Computer security measures may be technical, physical or administrative, or a combination of these. A combination of measures should be chosen using a risk informed approach based on a graded approach and defence in depth to achieve adequate computer security. The IAEA Nuclear Security Series publications (e.g. Ref. [16]) contain more detailed guidance on implementing computer security at nuclear facilities.

Computer security should be considered from two aspects in software development. The first element is establishing and maintaining a protective environment for all phases of the software life cycle. Such measures are required to protect against directed attacks against the software, including during the development phase, and to protect against a non-benign cyber environment, e.g. random malware.

The second element is implementing secure coding. The development of software is complex and program errors are not uncommon. Some errors may introduce software vulnerabilities that can be exploited by cyberattack, such as by buffer overflow and structured query language (SQL) injection. Secure coding practices and associated software testing for exploitable vulnerabilities should be a requirement in the development of all software used for safety I&C systems at NPPs. Active verification methods for secure coding may include, for example, penetration testing and vulnerability assessment methodologies such as fuzzing.

2.8. DEPENDABILITY PROPERTIES

To assess the dependability of a system, the meaning of 'system' and 'dependability' require elaboration. The five dependability attributes listed in the preceding section provide a starting point for a precise definition of dependability.

This section presents a categorization of system features that is generally applicable to safety I&C systems of NPPs. Table 1 describes how these system features for dependability could be associated with dependability attributes. Note that the same feature may contribute to several attributes.

These features result in properties that are associated with the first four of the APs presented in Section 1.1. The rationale behind this association is that these principles specifically relate to the I&C system and its software, whereas the other principles relate to the dependability assessment itself.

In the following, the system features for dependability are organized according to three groupings: functional features (2.8.1–2.8.9), non-functional features (2.8.10), and features regarding operation and maintenance (O&M) procedures (2.8.11), which specify how the system will be operated and maintained and possibly how its development environment and documentation will be kept operational and current. In some cases, a dependability assessment will be concerned mainly with the I&C system itself and its software, based on assumptions made regarding the corresponding O&M procedures. In other cases (particularly when one also wants to assess whether

TABLE 1. TYPICAL ASSOCIATION BETWEEN SYSTEM FEATURES AND DEPENDABILITY ATTRIBUTES

Systems features for dependability	Dependability attributes				
	Safety	Reliab.	Avail.	Maint.	Sec.
I&C system behavioural requirements for on-line nominal operation	✓				
I&C system behavioural requirements for on-line downgraded operation	✓	✓	✓		
I&C system behavioural requirements for on-line failure states	✓				
I&C system probabilistic requirements	✓	✓	✓		
I&C system behavioural requirements for off-line I&C system states		✓	✓	✓	✓
Software behavioural requirements	✓	✓	✓	✓	✓
I&C system behavioural requirements for computer security					✓
Software behavioural requirements supporting system reliability		✓			
Software behavioural requirements supporting system availability and corrective maintenance			✓		
Software design constraints		✓	✓		✓
Operational and maintenance procedures for the I&C system	✓	✓	✓	✓	✓

the dependability of the system and its software will be effective in the field and maintained over time), the O&M procedures are an intrinsic part of the assessment.

In this report, behavioural requirements include the functional requirements (what the system or software is required to do, and possibly, is required not to do) and the performance requirements (e.g. response times, accuracy).

2.8.1. I&C system behavioural requirements for on-line nominal operation

These are the functional and performance requirements for when the I&C system is on-line with the plant process and in fully nominal conditions. They should address the different plant states (including abnormal plant states) and operational goals (what the operators want to do with the plant at a given instant), including changing of the operational goal. They may be interdependent with some of the operational procedures.

For safety I&C systems, such requirements are usually associated with the safety attribute.

2.8.2. I&C system behavioural requirements for on-line downgraded operation

These are the functional and performance requirements for when the I&C system is on-line with the plant process and still operational, but suffers internal deviations from nominal behaviour owing to internal component failures or to intervention on some of its parts (e.g. periodic testing or maintenance). They should also address the different plant states and the possible operational and maintenance goals, and may be interdependent with some of the O&M procedures. They typically include the specification of fault tolerance and intervention tolerance requirements, and the specification of behaviour in the different downgraded situations.

For safety I&C systems, such requirements are usually associated with the safety, reliability or availability attributes.

2.8.3. I&C system behavioural requirements for on-line failure states

These are the functional and performance requirements for when failures prevent the on-line I&C system from accomplishing its missions. These requirements usually aim at ensuring that the system sets into predefined, presumably safe, failure modes and that operators are informed as appropriate.

For safety I&C systems, such requirements are usually associated with the safety attribute.

2.8.4. I&C system probabilistic requirements

These specify quantitative probabilistic limits on the failure rates of some or all of the safety I&C functions, and/or the occurrence rates of some or all of the failure states. Such limits can provide:

— An interface to probabilistic risk assessment (PRA) and PRA based sensitivity studies;
— Insights into the relative importance of different functions and components;
— Analysis to support the evolution of the I&C system design, e.g. in terms of defence in depth.

As with the use of quantified probabilities, the lessons learned and insights gained in producing the values can be as important as the values themselves.

For safety I&C systems, probabilistic requirements are usually associated with the safety attribute, but could be associated with the reliability or availability attributes.

2.8.5. I&C system behavioural requirements for off-line I&C system states

These are the functional and performance requirements for when the I&C system is not on-line with the plant process (e.g. during I&C system configuration, commissioning, initialization, testing before operation, etc.).

They are usually associated with the reliability, availability, maintainability or security attributes.

2.8.6. Software behavioural requirements

These are the functional and performance requirements for the software of each on-line functional unit of the I&C system. Some of them result from the allocation of the I&C system behavioural requirements to the functional units. Others derive from the architectural design of the I&C system, which is most often distributed and redundant, and may comprise units of different importance to safety (safety, safety related and non-safety). Also, some requirements which support particular O&M procedures may need to take into consideration the different possible internal states of the I&C system, including errors and failures.

The software behavioural requirements could be associated with any of the dependability attributes.

2.8.7. I&C system behavioural requirements for computer security

These are I&C system requirements that complement the physical security measures taken at the level of the plant, and the software constraints for computer security, to ensure that the I&C system is adequately protected against malicious attacks. When determining these requirements, one should ensure that they will not adversely affect the correct performance of safety functions.

2.8.8. Software behavioural requirements supporting system reliability

Such requirements aim at ensuring that the I&C system does not accumulate random hardware faults such that fault tolerance mechanisms would be defeated. They typically include requirements for self-monitoring and support for periodic testing. When determining these requirements, one should ensure that they will not adversely affect the correct performance of safety functions.

2.8.9. Software behavioural requirements supporting system availability and corrective maintenance

Such requirements aim at ensuring that the I&C system can be returned to normal on-line service as quickly as possible after an unplanned system outage. When determining these requirements, one should ensure that they will not adversely affect the correct performance of safety functions. However, they typically do not concern the on-line Class 1 software, to avoid unnecessary complexity.

2.8.10. Software design constraints

Such requirements specify non-functional design constraints. They may aim at software reliability (e.g. through fault avoidance and fault detection), software maintainability (e.g. through documentation) and computer security. They are often expressed as requirements to comply with specific standards, but system specific design constraints may also be specified. Standards such as IEC 60880 [17] and IEEE 7-4.3.2 [18] embody the lessons learned from past experience, and provide sets of requirements that are generally recognized as being appropriate for software reliability and maintainability. There may be emerging design constraints associated with computer security, but detailed international standards are still under development.

2.8.11. Operation and maintenance procedures for the I&C system

System dependability also depends on how the I&C system is operated and maintained. Therefore, the O&M procedures for the system need to be considered, either as assumptions or as an integral part of the assessment.

3. ASSESSMENT FRAMEWORK

This section describes an assessment framework that supports the implementation of the principles defined in Section 1.1, in the context and with the concepts presented in Section 2. It uses material from Refs [19, 20]. The section begins with a description of an overall strategy to guide the assessment that considers the behaviour of the system along with its interactions, vulnerabilities and compliance with rules and standards. Then, it introduces the concepts of CAE and how these might be used to develop and communicate the assessment. Next, guidance on a high level process for deploying the framework is provided. For each part of the framework, the underlying concepts are discussed along with how they relate to the assessment principle. Guidance is provided on generic application and some more specific illustrations are found in Annex IV. The section concludes with a summary of how the framework helps implement the APs.

3.1. THE STRATEGY TRIANGLE

3.1.1. Overall strategy

This report is concerned with the assessment of dependability, which at its core involves presenting evidence about system behaviour to support claims about dependability properties. A systematic evaluation of potential weaknesses and vulnerabilities in the system is also a key activity of the dependability assessment. Compliance with standards is another important aspect in defining design constraints (see Section 2.8) that need to be adhered to and that have a very significant role in the licensing of the system. The overall approach is therefore described as property based, vulnerability aware and standards informed, and is illustrated by the strategy triangle shown in Fig. 5.

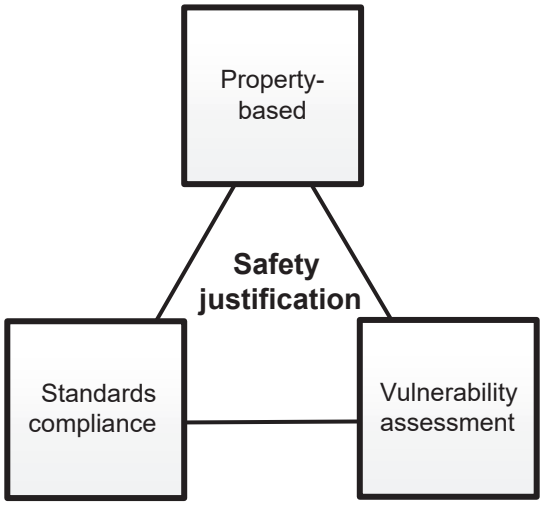


FIG. 5. The strategy triangle.

3.1.2. Property based approach

A property based approach focuses directly on the behaviour of, and constraints on, the system or software being assessed, while exploring claims about the satisfaction of the safety requirements and the mitigation of the hazards to the system. Such an approach aims at providing a direct justification for claims about specific features of the system or software. In the context of safety I&C systems, these claims relate to the first four APs:

— AP1: System and software requirements are adequately defined, valid and reviewable such that it can be confirmed that they collectively address the necessary functional and non-functional properties.
— AP2: An effective understanding of the role of the safety I&C system is demonstrated, including its direct and possible indirect roles, and also its role in supporting safe operation.
— AP3: The intended and unintended behaviour of the system is adequately taken into account, including adverse unintended consequences under conditions of failure.
— AP4: The implementation of the requirements is adequate.

3.1.3. Vulnerability based approach

An important part of these principles is to incorporate lessons learned and to address the unintended and intended behaviour of the system.

Vulnerabilities are weaknesses in a system that could be detrimental to dependability (e.g. if division by zero is not caught by error handling) but are not strictly faults. Vulnerabilities can be at different levels of abstraction (e.g. at the architecture level: lack of diversity) or within the details of coding (e.g. buffer overflow).

There are several methods and techniques that can be employed to perform a vulnerability analysis for a component or a system. At a component level, these approaches aim to identify both generic failure modes and their causes, or to provide evidence of their absence. At a system level, the device failure modes are considered in terms of the system/application vulnerabilities, and whether the mitigation steps are adequate.

Lessons learned from internal and external sources need to be incorporated into the vulnerability assessment. Lessons learned can be applied directly (e.g. from knowledge of previous problem areas in real time software) and also derived from information on the lack of compliance with design constraints (e.g. non-conformity with standards can be a vulnerability). Searching for vulnerabilities can help confirm that the implementation steps are adequate (e.g. absence of certain classes of defects) and can be a major part of implementing an active challenge to the dependability assessment.

The vulnerability part of the strategy supports the following APs:

— AP3: The intended and unintended behaviour of the system is adequately taken into account, including adverse unintended consequences under conditions of failure.
— AP4: The implementation of the requirements is adequate.
— AP5: Active challenge to the dependability assessment is part of decision making throughout the organization.
— AP8: Lessons learned are incorporated in the target system being assessed and in the assessment process itself.

3.1.4. Compliance with standards

An important part of AP8 (lessons learned) involves recognizing the experience of others and complying with the consensus defined by appropriate standards. This leads to the third part of the assessment triangle. Compliance with standards is an important part of the dependability assessment and, as this report addresses the software of safety systems, adequate compliance with standards will need to be demonstrated as part of the overall licensing or approvals process.

In the dependability assessment:

— Existing standards serve as the basis for the derivation of requirements (see Section 2) and lead to specific system and software requirements (design constraints).
— Non-compliance with standards is a potential dependability vulnerability and as such should be assessed for the impact it has on the behaviour properties (see the vulnerability part of strategy).
— Demonstrating compliance with standards can provide evidence of appropriate behaviour and design and should be used in the dependability assessment when it is available (supporting AP7).
— Standards are the basis for a systematic engineering process which provides for the disciplined production of documentation and evidence that is trustworthy (supporting AP7).

3.2. CLAIMS, ARGUMENTS, EVIDENCE

Over the past ten years there has been a trend towards an explicit claims based approach to safety assessment and assurance and considerable work has been done on the structuring of safety arguments [19–24]. The approach described here is based on a CAE approach [19–21]. CAE is now explicitly used in the generic design assessment for the new reactor construction programme in the United Kingdom, and it is also recommended, if structured justifications are being undertaken, by the "Common position of international nuclear regulators and authorised technical support organisations" [25].

There is considerable standardization work on structured cases and CAE in a number of sectors. Of particular relevance are the IEC/ISO standard that provides a definition of the CAE concept (ISO/IEC 15026-2:2011, Systems and Software Assurance: Assurance Case [26]) and the work of the Object Management Group (OMG) in defining interface standards [27].

The key elements of the CAE approach are:

— Claims, which are assertions put forward for general acceptance. These are typically statements about a property of the system or some subsystem. Claims that are asserted as true without justification become assumptions and claims supporting an argument are called subclaims.
— Arguments, which link the evidence to the claim. These are the "statements indicating the general ways of arguing being applied in a particular case and implicitly relied on and whose trustworthiness is well established" [28], together with the validation for the scientific and engineering laws used. In an engineering context, arguments should be explicit.
— Evidence, which is used as the basis of the justification of the claim. Sources of evidence may include the design, the development process, prior field experience, testing (including statistical testing), source code analysis or formal analysis.

To support the use of CAE, a graphical notation is used to describe the interrelationship of the CAE. The basic graphical elements of CAE are shown in Fig. 6.

In practice, the desired top level claims are not directly supported or refuted by evidence. Therefore, it is necessary to develop subclaims until the final nodes of the assessment can be directly supported (or refuted) with evidence.

Research supported by an empirical analysis of actual safety cases has identified a number of basic building blocks (CAE blocks [29, 30]) that are useful for expressing the assessment. These are:

— Concretion blocks;
— Substitution blocks;
— Decomposition blocks;
— Calculation blocks;
— Evidence incorporation blocks.

3.2.1. Concretion blocks

This block is used when a claim needs to be given a more precise definition or interpretation. This is often the case of top level claims, which generally need to be expressed in more measurable, less abstract, terms. For example, a claim on a system's dependability can use a concretion block to introduce subclaims for each of the dependability attributes. Although this kind of block is necessary, it needs to be clearly identified since it breaks the reasoning in the claim tree.

3.2.2. Substitution blocks

Another common type of claim expansion involves transforming a claim about an object (or property) into a claim about an equivalent object (or property), which can be viewed as a form of substitution.

For example, one might claim that the designed system has a certain property, and therefore the production system has this property too, assuming that the production system is equivalent in some clearly defined way to the designed one.

Another example would be the substitution of an executable software code with a source code, as long as the translation of the source code into executable software does not introduce effects that would invalidate the argument built on the source code.

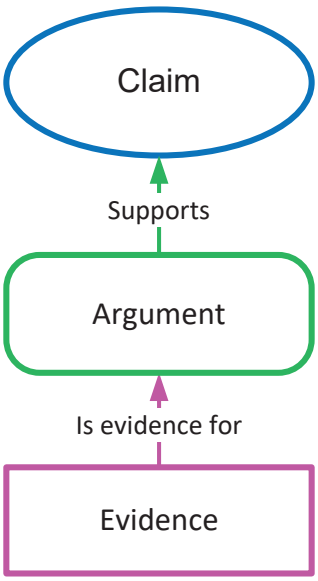


FIG. 6. Basic graphical elements of CAE.

3.2.3. Decomposition blocks

This block is concerned with structure. Many claim decompositions are about partitioning some aspect of the claim, for example, according to the functions of the system, the architecture, the properties being considered or with respect to some sequence such as life cycle phases or modes of operation.

For example, to make a claim about a property of an object, one can investigate whether the object has this property by evaluating its components. To do this, one needs to be clear regarding the decomposition of the object into components, which properties need to be claimed for these components and how these component properties combine to ensure the claimed object property.

3.2.4. Calculation blocks

This block is used to claim that the value of a property of a system can be computed from the values of related properties of other objects (e.g. its subsystems). One application of the block is to provide a quantitative argument when the value of one property can be calculated from the values of other specific properties.

For example, the availability of a system can be calculated from its reliability and its failure recovery time. As another example, the average time of data retrieval from a database can be calculated from the probability that the data are in the cache and the time of data retrieval if they are not in the cache.

3.2.5. Evidence incorporation blocks

This block is used at the edge of the CAE structure to incorporate evidence into the assessment. It is used to demonstrate that a subclaim is directly satisfied by its supporting evidence.

Figure 7 illustrates how a top level claim first needs to be made more precise (using a concretion block). The resulting claim, claim (X) cannot be directly shown by evidence, so an argument is made that it can be split into two subclaims (using a substitution, decomposition or calculation block). One of these subclaims, claim (A) can be directly supported by evidence and the results needed for this can be identified.

It is important to note that structures such as those shown in Fig. 7 illustrate the outline of the assessment being made, but in addition, there will be important narratives and analyses explaining and detailing the claims and arguments being made. A narrative is an essential part of the assessment.

In practice, some of the basic blocks might be merged together into composite blocks. In such situations, expanding the structure further into a more detailed assessment could help in understanding the underlying logic and determining which basic blocks have been applied.

The block definitions relate to an object, such as a system. It may sometimes be desirable to make the environment and configuration explicit, but otherwise keeping them implicit may make the assessment easier to read. When a claim is made, confidence is implicit that it is true. If the desire is to make it explicit, each property could have a confidence value, resulting in the need for a calculus for devising and propagating these confidence values.

3.3. DETERMINING DEPENDABILITY CLAIMS

The identification of the top level claims for software dependability is dependent on the scope of the assessment. In some cases, the object of the assessment is the software of individual functional units. In other cases, the scope is wider and the object of the assessment is the software of a complete distributed system composed of multiple, interconnected functional units, taking system requirements as a given. In yet other cases, the object of the assessment also includes confidence in the adequacy of the system requirements. The object of the assessment could also cover the software of multiple systems, addressing issues such as diversity. Each project may decide on the scope of assessment based on its specific circumstances.

Normally, claims would be closely related to requirements. However, as the number of requirements could be very high, it is usually preferable to organize the assessment on a limited number of top level claims that are the most important for dependability. In practice, top level claims would also address a number of related requirements. The identification of such top level claims could be based on the main dependability attributes listed

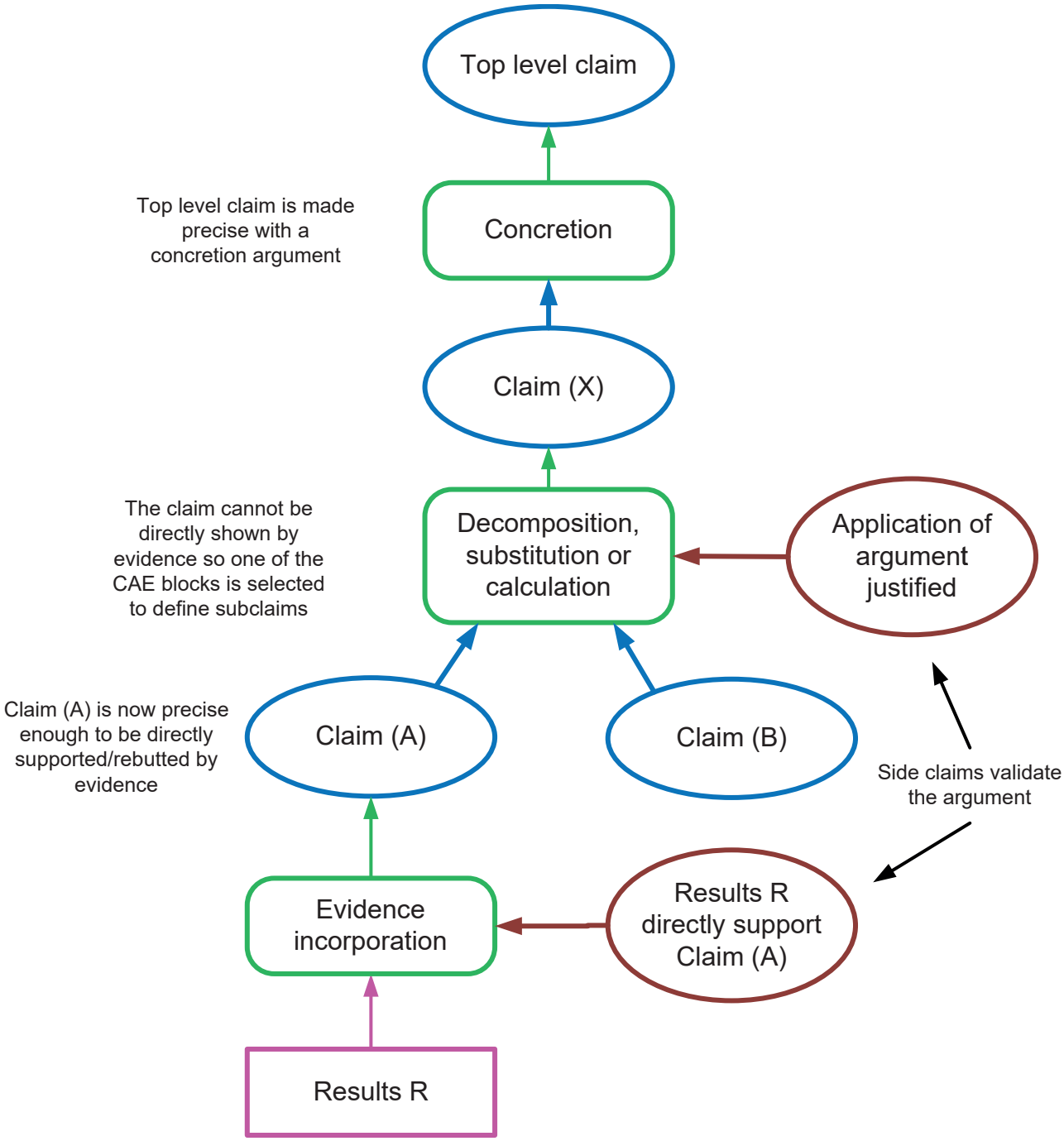


FIG. 7. Example of a typical CAE structure based on CAE blocks.

in Section 2.7 and the features and properties identified in Section 2.8. Additional top level claims could also address the pillars of the strategy triangle (see Fig. 5) that are not already covered by addressing the dependability attributes (e.g. compliance to standards or rules).

A subset of top level claims could include the following conditions:

— *Reliability functional suitability.* The requirements specification for the object of the assessment is adequate.
— *Reliability functionality.* The object of the assessment implements its functional requirements (including timing and accuracy requirements).
— *Reliability functionality*. The probability of failure on demand (PFD) of the object of the assessment is less than X (similar claims could be made regarding spurious actuations and probabilities of software CCFs).

— *Safety robustness.* The object of the assessment has adequate protection against the specified failures.
— *Safety robustness.* Failures of subset A of the object of the assessment will not spread to subset B.
— *Safety fail safe*. In the case of a detected failure, the object of the assessment will behave as specified.
— *Rule compliance.* The object of the assessment complies with the applicable standards.
— *Vulnerability assessment*. The object of the assessment does not contain intrinsic software faults (that can be recognized independently of functional requirements, such as division by zero).

3.4. DEPLOYING THE FRAMEWORK

This section provides guidance on the deployment of the assessment framework.

There are many examples of delays to I&C projects that can be attributed in part to the lack of clarity or technical inadequacy of the assessment process. The design of the dependability assessment is a significant project activity. Constructing the assessment — that is, developing the claims and arguments and identifying the nature of the evidence to be provided — is relatively inexpensive compared with design changes that have to be made later if the assessment approach is flawed. It is also relatively inexpensive compared with the production of the detailed evidence. Indeed, designing and securing agreement on the assessment strategy can be seen as a form of requirements analysis with a similar return on investment for getting it right early in the project.

This section defines four phases in the assessment: mobilization, definition of the assessment strategy, performance of the assessment and maintenance of the assessment. These can be mapped to different parts of the I&C system life cycle, for example:

— For a new item, the assessment phases can closely follow the system or software life cycle, i.e. the assessment strategy is dealt with in the requirements phase, the performance of the assessment during the implementation phase, etc.
— For a predeveloped item, the assessment might be part of a suitability analysis. In this case, the four phases of the assessment are done within this single activity.

In practice, I&C systems will involve a mixture of predeveloped, configured and custom made software, and the assessment phases should be mapped and integrated to the project life cycle. The assessment approach proposed here is life cycle independent and can be configured to address the heterogeneous nature of I&C systems. The phases are defined in more detail in Section 3.4.1.

The progressive development of the evaluation described in the assessment phases is essential for dealing with the scale and complexity of projects. Another approach for dealing with scale comes from using layers of assurance that deal with different abstraction levels. For most projects, there are at least three layers of assurance addressing requirements, architecture, and implementation (Section 3.4.2 provides some details). The assurance layers can either be developed in a top down manner closely linked to the life cycle phases or, as is more likely, developed according to the specific needs of a project. For example, in dealing with predeveloped items, implementation evidence may be available at an early stage and help shape the selection and architecture of the system and the assessment.

3.4.1. A phased approach

The production of the assessment should be phased and systematic. It should be undertaken within a safety or quality management system that ensures that the roles and responsibilities for the assessment are defined and that engineering processes are defined that lead to the orderly execution of the assessment [2]. Apart from the usual project initiation that ensures resource availability, clarifies requirements and identifies training requirements and project risks, there are four main assessment phases (introduced in the previous section): mobilization, definition of the assessment strategy, performance of the assessment and maintenance of the assessment.

These four phases can be developed into tasks and work packages and mapped to specific project and evaluation engineering life cycles. It may be useful to consider interim delivery of assessment findings within Phase 3. In addition, the evaluation of prototypes and existing components or systems that are similar to the proposed system may be used to manage the risk in both the assessment process and the findings.

3.4.1.1. Phase 1: Mobilization

In this phase, the scope and context of the assessment is established and preparatory information and resource gathering undertaken. In particular, AP6, "The objectives of the users of the assessment are identified such that the structure and presentation of the assessment account for their needs" is addressed.

Establishing the system and the assessment context and scope involves:

(1) Identifying the system to be assessed, its safety role and its relationship with other systems and the environment and stakeholders.
(2) Identifying the types of decision being supported by the assessment and the expectations and roles of the different stakeholders in making the decisions.
(3) Identifying the initial high level dependability claims that need to be addressed.
(4) Establishing the scope of and motivation for the assessment, and establishing the stakeholders' expectations and responsibilities, as well as how communication with them can be effective. An element of the scope is the expected longevity of the assessment.

Configuring and preparing for the assessment includes:

(a) Identifying any existing analyses (e.g. safety justifications, business continuity assessments) that provide details of the system, the impact of failures, and the mitigations that are in place.
(b) Identifying CAE or other assessment templates that may be appropriate.
(c) Identifying sources of operating experience and licensing experience.
(d) Identifying information on vulnerabilities of products and previous issues with the licensing approach. Examples of generic vulnerabilities are provided in Annex III.
(e) Characterizing the maturity of the systems or project and the key uncertainties.
(f) Identifying the communication needs for the different stakeholders, e.g. the technical focus or the level of detail required.
(g) Defining the set of assessment documents to be produced.

3.4.1.2. Phase 2: Definition of the assessment strategy

The next phase involves development of an assessment strategy. At this stage, a preliminary CAE structure is produced that defines the claims and the arguments to be used and identifies the evidence needed. This involves:

(a) Defining the top level claims for the assessment, and expanding and clarifying the initial claims (see Sections 2.8 and 3.3).
(b) Exploring and brainstorming to produce a preliminary claim–argument structure and an initial identification of analysis techniques and evidence. This would progress from high level attributes and properties to precise dependability claims. The assessment might be developed with CAE blocks informally or it may employ existing templates or examples.
(c) Reviewing the claim–argument–evidence at the edges of the overall CAE structure, and considering whether it is feasible for the evidence and arguments proposed to demonstrate the claims with sufficient confidence and consider alternative strategies. Identifying areas open to challenge and uncertainty. Developing options for changes to requirements if assurance strategies are shown not to be appropriate.
(d) Interacting with the engineering life cycle, particularly the requirements and architecture phases. Examples of issues are:
   — Considering trade offs between a more complex architecture and ease of assessment, e.g. extra diversity so that the assessment of subsystem reliability is less onerous;
   — Simplifying the diversity argument by using technological diversity;
   — Choosing a deterministic platform to make it easier to substantiate certain claims;
   — Using a system or platforms that support formal verification approaches to support certain types of argument.
(e) Conducting a preliminary analysis of information on vulnerabilities of the system components.

(f) Conducting a preliminary analysis of previous assessments and an appraisal of which CAE might be reused.
(g) Developing a more detailed and precise CAE structure with completed documentation that justifies the structure and identifies any additional assessment that is required, e.g. completing the side claims associated with CAE blocks.
(h) Defining the detailed evidence and supporting analyses that are required. Revisiting the relationship to the engineering life cycle and the scheduling of evidence production.

3.4.1.3. Phase 3: Performance of the assessment

At this stage, the assessment strategy should be stable and the assessment is completed by the incorporation of evidence which may either support or rebut the claims being made. This phase includes:

(a) Detailed analysis and evidence production activities, and linking these to the assessment strategy;
(b) Incorporating results and evidence into the assessment;
(c) Adjusting the assessment in the light of results.

3.4.1.4. Phase 4: Maintenance of the assessment

In this phase, the assessment is maintained. It directly addresses the first sentence of AP9: "Any changes in the target system or conditions of use/maintenance that would invalidate the assessment are detected and addressed". The changes that need to be considered include:

(a) Changes to the system (e.g. from maintenance or upgrade of components);
(b) Changes to assumptions (e.g. dependencies on other systems that might themselves change);
(c) New evidence becoming available (e.g. impact of new analyses, new field data, new vulnerabilities discovered);
(d) New analysis becoming available (e.g. a new type of analysis, a more capable tool set);
(e) Changes to the environment (e.g. computer and general security threat assumptions, assumptions on hazards, changes to use of the system);
(f) The trustworthiness of the assessment becomes undermined (e.g. changes to computer and general security threat assumptions, fraud).

Anticipating the potential for changes to the system and their impact on the assessment, the development of an assessment strategy (Section 3.4.1.2) needs to account for maintainability of the assessment itself.

Although computer security is generally excluded from this report, there are a number of areas where it is essential. The impact on dependability of designs for updating the systems, the obsolescence of controls, and the trustworthiness of the supply chain should be addressed.

3.4.2. Layers of assurance

The progressive development of the evaluation described above is essential for dealing with the scale and complexity of projects. Another important approach for dealing with scale comes from using layers of assurance that deal with different abstraction levels [31]. For most projects, there are at least three layers of assurance:

— L0: Policy and requirements. The highest level of abstraction, where the many different types of system requirements and interdependencies are analysed.
— L1: Architectural layer. The intermediate level where the abstract system components and architecture are analysed.
— L2: Implementation layer. The detailed level where the implementation of specific components and their integration within the specific system architecture are scrutinized.

Nuclear I&C projects can benefit from a more detailed set of layers, as shown in Fig. 8.

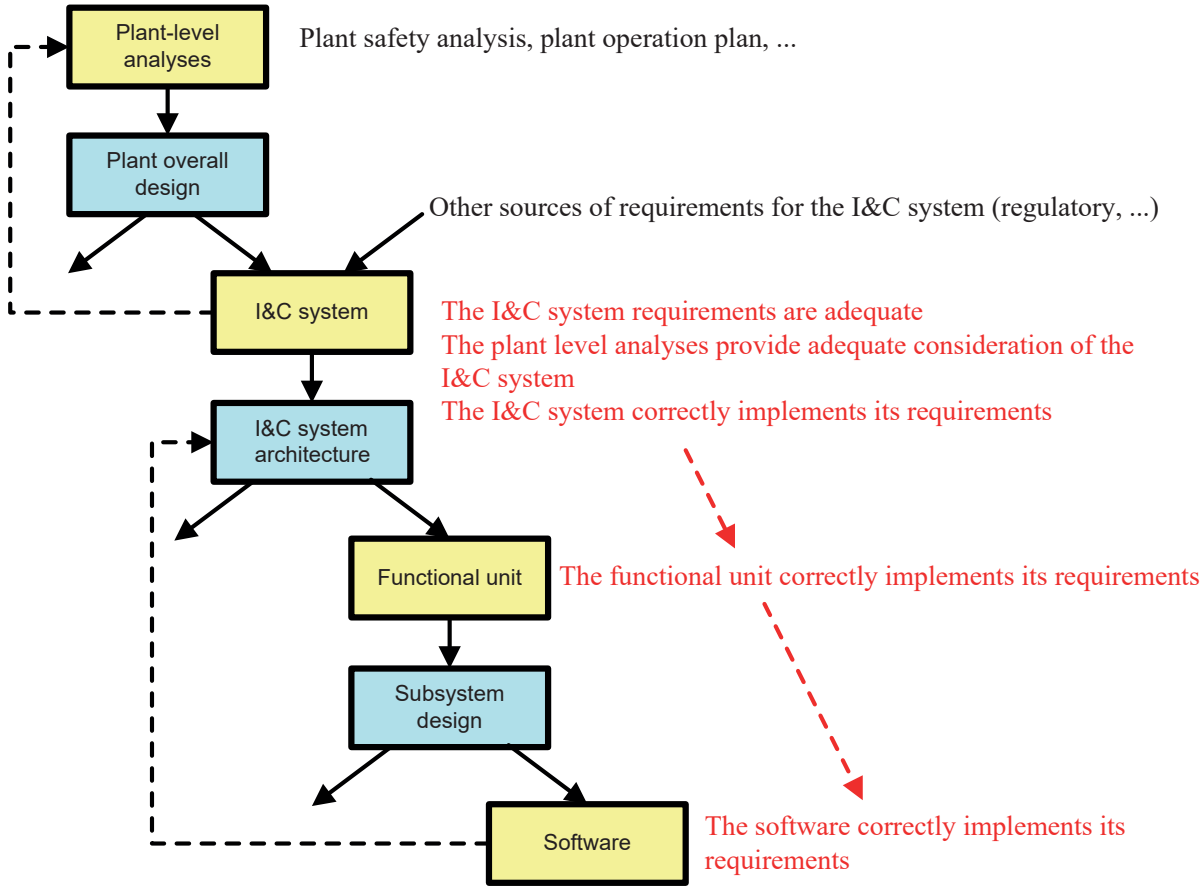


FIG. 8. System hierarchy.

The figure illustrates a typical example of hierarchical systems, and how claims (shown in red) made at one level are related to upper or lower levels:

— The top level system is the plant itself. Many plant level considerations affect what is required of the safety I&C system, such as the plant safety analysis (which identifies the hazards that could affect the plant and that need to be prevented, minimized or mitigated), the plant operation plan (which specifies the main steps and procedures by which the plant is commissioned, operated and maintained in operational conditions) and the overall plant design (which identifies the main functions that need to be performed, and the main systems that will implement them). Note that this list of plant level considerations is not exhaustive and is likely to include other considerations not mentioned here.
— One of the systems identified by the overall plant design is the safety I&C system under consideration. A key item for that system is its system requirements specification. The term 'requirements' mainly designates those requirements that are of particular importance to system dependability. They address topics such as operational modes, functions to be implemented in each mode, response times, accuracy, interfaces (with other systems or with human operators), self-monitoring, fault tolerance, acceptable/unacceptable failure modes, acceptable failure rates, assumptions made, etc.
— The system is itself composed of multiple functional units (controllers, communication links, instrumentation, etc.) as determined by the I&C system architecture. The architecture allocates the I&C system requirements to the units, specifies the interactions between the units and specifies the requirements for each unit.
— Some of the units are digital (e.g. digital controllers, smart instruments, data communication units) and are a combination of hardware and software (more generally, logic). The design for such a unit allocates the subsystem requirements to hardware and software, specifies their interactions and specifies in particular the software requirements.

In projects that follow a waterfall life cycle model, the layers are closely related to the phases of the project. More usually an I&C system will consist of both project specific developments (such as the application logic in a protection system) and already developed and deployed platforms, smart devices and interfaces that may be non-nuclear in origin.

3.4.3. Mapping properties, techniques and evidence

Section 4 describes the wide variety of techniques for generating evidence relevant to dependability, including operational experience, compliance with standards, functional validation, software analysis techniques, validation, testing and review, all of which may be used to support dependability claims.

To develop a particular assessment, the property will need elaboration and concretion and the claim that might be directly inferred from the technique will need to be carefully defined. The stages of argumentation needed to bridge the gap should be defined as part of the assessment strategy (see Phase 2). Annex V provides a table mapping generic property types to the techniques, and Section 4 provides some examples of claims that the techniques might address.

The analysis techniques will have many assumptions and caveats associated with them, and it is important that these be captured within the assessment. For example, a prover might provide the answer 'TRUE' or a fault tree analysis (FTA) may provide a numerical result. However, these results alone would be insufficient and unconvincing without further elaboration of their basis.

There are other engineering issues that need to be addressed in deploying the techniques. In designing the assessment approach, the evidence actually provided by the techniques needs to be carefully identified. Often, tools and test beds are needed to deploy the techniques, and the direct evidence is often of the form 'software in test bed passed all results'. A detailed argument therefore needs to be developed about the dependability of the test bed, the process of sentencing the results, and the extrapolation from the system and configuration that is being tested to the actual deployed system.

3.5. BUILDING CONFIDENCE IN THE ASSESSMENT

The confidence in the assessment, and its constituent parts, comes from a process of elaboration, challenge, update and consensus. This cycle of challenge and review can result in the assessment being modified and typically involves a range of different stakeholders. The allocation of responsibilities is specific to a Member State's regulatory approach or the structure of a specific project. However, challenge and review is facilitated by layers of independent oversight embedded in the engineering processes.

AP5 stipulates that "Active challenge to the dependability assessment is part of decision making throughout the organization". Challenge and review should be an element of the engineering processes that support product development and assessment. As the assessment is developed, the ways in which it has been reviewed and adapted should be documented (see Section 4).

The approach recommended in this report addresses AP5 by:

— Adopting the 'strategy triangle' approach, which explicitly includes vulnerabilities as part of the assessment framework that can be the basis for challenges. Annex III provides examples of possible vulnerabilities based on industry experience of I&C applications and platforms.
— The use of CAE blocks, which provides generic questions to address, and hence a source of challenge and confidence building for the overall CAE structure.
— Drawing on nuclear industry licensing and project experience (see Section 5, Annex II and Annex III).

In building confidence in the assessment process, it may be appropriate to assess the assessment process itself. Safety analysis techniques can be applied to the process of dependability assessment itself. There are a number of different hazards, but there are two basic failure modes to be avoided:

(1) Accepting an unsafe system;
(2) Rejecting a safe system.

The second class of failures can escalate to the first class of failures by leading to a loss of focus and diversion of resources. In a risk adverse industry, most problems with the assessment process manifest themselves as delays in licensing or even rejection of systems or components (type 2 failures) rather than as type 1 failures.

Assessments currently express confidence implicitly in that when a claim is made that 'this system is reliable', what is really meant is that 'I am confident that this system is reliable'. This follows from the definition of a claim: a claim is not a goal or a requirement but a statement about something that has been achieved. The uncertainties in the dependability assessment concern both aleatory and epistemic aspects [32] (see Table 2).

When an assessment is reviewed and challenged, it is normally the epistemic aspects that are being addressed. (There is a large body of literature for non-computer safety cases. See, for example, Ref. [33].) When questioning how confident a claim needs to be and whether a certain level of confidence is justified, it may be helpful to express that confidence in the claim explicitly or even quantitatively. However, this is still very much a research topic [32, 34–36], although for software reliability there are some approaches discussed in Annex I.

TABLE 2. ALEATORY AND EPISTEMIC UNCERTAINTIES

Aleatory	The use of probabilities to characterize software dependability is sometimes still challenged. People say: software failures are systematic — if it fails in certain circumstances, it will always fail when those circumstances are repeated — so what is the role of probability? The answer, of course, is that there is uncertainty about *which* inputs (of those not yet executed) will cause failure, and about *when* these will be executed, thereby causing failure. The failures that occur when software executes form a stochastic process. The uncertainty associated with this process is *aleatory*, i.e. it is 'uncertainty in the world' and is irreducible.
Epistemic	There is also *epistemic* uncertainty concerning the models and reasoning that are used to estimate and predict reliability. For example, in operational testing, there may be uncertainty about the correctness of the test oracle, and about the representativeness (with respect to real life operation) of the test case selections. Epistemic uncertainty is in principle reducible, e.g. by collecting more and better evidence concerning the subject of the uncertainty.

The evaluation of the assessment needs to be rigorous and independent enough to avoid expert and institutional overconfidence. Overconfidence — underestimating the uncertainty surrounding one's judgements — has been long reported as common in engineering experts and even in scientists such as physicists. Henrion and Fischhoff [37] investigated the accuracy of the judgements of physicists about the values of physical constants such as the speed of light, charge on the electron and others. They found that individual teams involved in measuring these constants (over a period of about a century) were too optimistic about the errors in their measurements. More interestingly, the physics community, in periodically arriving at a scientific consensus about the values of the constants (using results from many experiments), consistently underestimated the errors. Lin and Bier summarized results from experts in many engineering problems, reporting high (and highly variable) degrees of overconfidence [38]. These various results from areas where rigorous, measurement based practice is the norm suggest that similar overconfidence should be expected in software engineering, where such a practice is far from established. Heuristics and biases that have been popularized in Refs [39, 40] point specifically to the need to address confirmation bias in safety cases. Overconfidence can work in both directions: it can lead to rejection of valid evaluations if the assessor is overconfident in rejecting systems and the acceptance of invalid ones if the assessor is overconfident in their dependability. Both have safety and cost implications.

In safety analysis, the focus on hazards and generally undesirable events happening can reduce overconfidence. More specifically, the principles of the assessment framework address possible cognitive and organizational biases and support review by:

— Including active challenge to the dependability assessment as a basic assessment principle (AP5);
— Including rebuttal and monitoring for assumption changes as a basic assessment principle (AP9);
— Recommending that the assessment be embedded in a systematic engineering process (supporting AP5 and AP7);

— Including a strategy triangle that explicitly includes vulnerabilities as part of the assessment framework (principle AP8);
— Using CAE and CAE blocks to provide justifications that can be more readily challenged and can avoid certain reasoning fallacies by the nature of their construction (principle AP7, and supporting AP5 and AP8).

3.6. KEY POINTS

This section describes an assessment framework that implements the principles defined in Section 1.1 and the overall requirements defined in Section 2. It achieves this by establishing a strategy that considers dependability properties, vulnerabilities and standards. The strategy is implemented in a phased and progressive manner and guidance is provided on the activities within each phase. The need to address a range of stakeholders and their role in decision making and assessment is included. Active challenge is discussed and is embedded in the approach. Specific support is provided for challenge in the description of vulnerabilities and lessons learned from operation and licensing I&C systems.

The following sections describe how the framework addresses the APs.

AP1: System and software requirements are adequately defined, valid and reviewable such that it can be confirmed that they collectively address the necessary functional and non-functional properties.

Section 2 describes how the overall dependability attributes can be mapped into functional and non-functional properties. This will support the assessment of dependability in terms of the requirements for an I&C system.

The consideration of explicit properties is part of the framework strategy triangle. The dependability properties associated with this principle should be the basis for top level claims. Executing the phases of the assessment, described in Section 3.4.1, will demonstrate whether or not these claims are satisfied. In particular, in developing the assessment strategy (Section 3.4.1.2), 'adequately' should be explained and defined.

Specific techniques in Section 4 address functional validation analysis and demonstrating compliance with appropriate standards.

AP2: An effective understanding of the role of the safety I&C system is demonstrated, including its direct and possible indirect roles, and also its role in supporting safe operation.

Derivation of specific dependability properties (Section 2) and developing these into top level claims (Section 3.3) should demonstrate understanding.

Explaining in the assessment how or whether the dependability properties are satisfied should demonstrate understanding.

Also, the challenge and the response to challenge (Section 3.5) will test understanding.

AP3: The intended and unintended behaviour of the system is adequately taken into account, including adverse unintended consequences under conditions of failure.

Specific modelling and analysis techniques can explore possible unintended behaviour. Section 4 provides details on the sources of evidence. Some of these are more suited than others to unintended behaviour and complex interactions. Formal modelling such as that described in Section 4.3.1 provides methods for exhaustive searching for certain properties. Brainstorming and system analysis techniques are more open ended and support checks that models are appropriate as well as searches for unexpected behaviours and interdependencies. Examples are techniques such as those mentioned in Sections 4.3.4 and 4.3.5. Section 4.5.4 also explores these issues.

The analysis of generic vulnerabilities and the search for specific weaknesses support claims that unintended behaviour has been investigated.

AP4: The implementation of the requirements is adequate.

The dependability properties associated with this principle should be the basis for top level claims. Executing the phases of the assessment, as described in Section 3.4.1, demonstrates whether or not these claims are satisfied. In particular, in developing the assessment strategy (Section 3.4.1.2), 'adequate' should be explained and defined.

Many of the techniques in Section 4 may be used for generating evidence to support these claims.

AP5: Active challenge to the dependability assessment is part of decision making throughout the organization.

Challenging and building confidence in the assessment is addressed explicitly in Section 3.5. In particular, embedding the assessment in a systematic engineering process means that review is a natural part of the life cycle.

The guidance supporting AP3 on searching for unintended behaviour (e.g. via hazard oriented methods or vulnerability assessments) also supports active challenge.

AP6: The objectives of the users of the assessment are identified such that the structure and presentation of the assessment account for their needs.

As recommended in Section 3.4.1.1, the identification of stakeholders and their needs is part of the deployment process and initiated in the very first mobilization phase.

Specific approaches might be to map claims and evidence to the interests and roles of stakeholders, e.g. for operational issues. Some stakeholders may require different documentation of specific findings of the assessment written for their perspective.

AP7: The findings of the assessment are organized in a logical, coherent, traceable and accessible manner and are repeatable, with rigour commensurate with the degree of trust required of the system.

The CAE concepts presented in Section 3.2 provide a basis for organizing the assessment. The use of a specific CAE structure can aid communication. Repeatability is assisted by clarity in what is evidence and what claims it supports, and by the use of defined phases of assessment. Coherence and repeatability can be supported using templates.

AP8: Lessons learned are incorporated in the target system being assessed and in the assessment process itself.

This principle is addressed throughout the framework. Overall lessons learned are addressed explicitly in Sections 4.1 and 5. Standards are an important repository of experience and lessons learned and are discussed in Sections 3.1.4, 4.2.1 and 5.

Known vulnerabilities are addressed explicitly in the framework (Section 3.1.3 and Annex III).

In addition, monitoring for lessons learned is part of Phase 4 of the assessment process (Section 3.4.1.4). The use of operational experience as a source of evidence is discussed in Section 4.1.

AP9: Any changes in the target system or conditions of use/maintenance that would invalidate the assessment are detected and addressed. Fundamental assumptions underlying the assessment are identified.

Section 3.4.1.4 describes Phase 4 of the assessment to address this principle directly and provides examples of the changes that need to be considered. This principle suggests that maintainability of both the system and the assessment should be considered in the assessment strategy.

4. SOURCES OF EVIDENCE FOR THE ASSESSMENT

This section describes techniques of generating evidence for the various dependability claims, including operational experience, compliance with standards, functional validation, software analysis techniques, validation, testing and review. It should be noted that this section does not cover every available technique but does address many of those commonly employed. It should also be noted that the mention of a technique in this section does not imply endorsement. Lastly, although it is expected that a project will use some of the techniques mentioned (or equivalent ones), there is no implication that it needs to use all of them. For each technique, the section provides a brief description and then a non-exhaustive list of claims, sublaims and dependability attributes for which it could provide evidence.

Annex V provides a table summarizing techniques of generating evidence for the dependability properties discussed in this section. Individual subsections provide details for each of the techniques presented and describe their role in the assessment.

An assessment may reuse evidence that has been generated, for example, on previous projects or different versions of the product. The CAE framework should be used to justify the confidence in the evidence (e.g. assessing its provenance), and also to provide a clear chain of reasoning from the existing evidence to the claims being made in an assessment. When predeveloped safety grade digital I&C products/equipment have been used (i.e. 'certified for safety' platforms/products such as an IEC 61508 [9] safety integrity level (SIL) 3 certified safety programmable logic controller), a dependability assessment must still be performed and sufficient evidence must be made available (i.e. typically both from the design organization/original equipment manufacturer and the project). There may be pre-existing evidence in the form of generic certification reports or application specific qualification assessments. These may support the overall dependability assessment, but consideration should be given to defining the specific claims they support and how the evidence they provide substantiates these claims.

4.1. OPERATIONAL EXPERIENCE

Operational experience data should be analysed to identify events, if any exist, that could provide evidence of unsuccessful experience. The actions taken to correct the software and improve its development processes should also be analysed.

Evidence of successful operational experience may be used to support dependability claims for predeveloped software, but the credit that can be taken for this depends on the context. In particular, the experience cited as evidence must be shown to be relevant to the nuclear application under consideration. Such relevance depends on several factors:

— The objectives, criteria and procedures for collecting operating experience data should be identified and shown to correspond to the level of trustworthiness required by the assessment.
— The operational experience taken into consideration should correspond to similar applications and conditions of use.
— The facts in the operational experience taken into consideration should correspond to precisely identified versions of the predeveloped software. When this software is specific to, and embedded in, equipment, the facts should also correspond to precisely identified versions of the equipment in which it operates.
— When all or part of the experience corresponds to versions other than the one under consideration, the differences should be assessed, and the relevance should be justified.
— The volume of the experience (e.g. the number of units, the length of time in service, and for software acting on demand, the number of demands) must also be considered sufficient.

The trustworthiness of the experience could be established by an analysis of the procedures used for collecting the operational experience. In particular, one needs to make sure that claimed procedures were applied, that the failures (if any) were correctly detected, analysed and reported and that the corresponding software faults were successfully corrected. Consideration also needs to be given to the demand space that has been exercised in the operating experience and its relationship to the demand space for the application of interest. Specifically, fault handling routines for rare events/conditions may never have been exercised during historical usage of the software.

The relevant standards and guidelines that describe acceptance of operational experience include:

— IEC 60880, Nuclear Power Plants: I&C Systems Important to Safety (2006) [17]. Software aspects for computer based systems performing Cat. A functions.
— Licensing of Safety Critical Software for Nuclear Reactors: Common Position of International Nuclear Regulators and Authorised Technical Support Organisations [25].

Examples of high level claims that could be supported by this technique include the following:

— Subsystem availability meets the specified goal under identified conditions with a known configuration. Quantitative claims of availability may be substantiated based on statistical data (safety, availability).
— Component reliability meets the specified goal under identified conditions. A successful operating history can provide assurance of reliability (safety, reliability).

4.2. COMPLIANCE AND QUALITY ASSURANCE

4.2.1. Standards and guidelines

Several standards and guideline publications give guidance on the specification, design, development, verification and validation of software for digital I&C systems for NPPs. These publications have been issued by a number of organizations, including the IAEA, IEC and IEEE, and reflect the knowledge, experience and consensus of subject matter experts. Providing evidence of compliance with relevant standards and guidelines can help establish assurance that the practices used to develop, verify and validate software promote aspects of dependability.

A system is typically regarded as compliant with a standard when it meets all of the requirements of the standard on a clause by clause basis. Any non-compliance will need to be justified as providing an acceptable alternative to the related clause in the standard [25].

With regard to evidence of compliance with standards and guidelines for safety software, Ref. [25] prescribes the following:

— Regulations, standards and guidelines used for safety demonstration should be identified.
— The applicability of the standards to be used should be justified, and any deviations from the standards should be evaluated and justified.
— Indication should be given when standards are to support specific claims or evidence.
— All supporting evidence should be subject to configuration management, change control and impact analysis.

Typical claims that can be supported by compliance with standards and guidelines are:

— *Maintainability.* Established life cycle standards and guidelines promote the maintainability of software as a key aspect.
— *Reliability.* Several standards and guidelines provide recommendations for the production of reliable software.
— *Security.* Standards and guidelines for software important to safety provide recommendations to establish computer security.

Examples of high level claims that could be supported by compliance with standards and guidelines are:

— A rigorous development process based on a well defined, established life cycle for the production of software has been applied (safety, reliability).
— Software design and documentation have technical features that promote software maintainability (maintainability).
— Adequate configuration management has been applied (maintainability).
— Application of measures to establish computer security (security).

4.2.2. Coding guides

A coding guide or a coding standard is a collection of guidelines for developers to follow when writing code. It helps ensure that the quality attributes of a software development activity are appropriate to the required level of dependability. The aim of individual coding rules may be to force developers to avoid problematic coding constructs, such as the use of pointers, unless their use is clearly justified. Another aim may be to encourage developers to write clearly comprehensible code that increases the effectiveness of code verification and code maintenance processes. One possible means to this (among others) is compliance with particular software metrics (for more information on software metrics, see Refs [41–50]). In some cases, such as with HDL, coding guidelines may be specific to a particular manufacturer's hardware or platform.

Coding guides and standards contribute to software dependability. They are a supportive technique to promote proven and widely accepted coding practices to improve the quality and clarity of the code and these code characteristics are associated with software dependability.

Typical claims that can be supported by coding guides are:

— *Maintainability.* One of the main reasons for using coding guides and standards is generally to improve maintainability of the code.
— *Reliability.* A subset of the coding rules may represent best engineering practices to avoid common coding errors. However, even strict adherence to such rules cannot guarantee that errors can be eliminated.
— *Security*. Certain coding standards may aim at addressing aspects of this dependability attribute.

Examples of coding standards and guides include:

— Guidelines for the Use of the C Language in Critical Systems [51];
— Guidelines for the Use of the C++ Language in Critical Systems [52];
— CERT C Coding Standard, 2nd edition [53];
— HDL Coding Practices to Accelerate Design Performance [54];
— Actel HDL Coding Style Guide [55];
— Individual coding guidelines of software developers.

Examples of high level claims that could be supported by compliance to coding guides include:

— The code has technical features that facilitate software maintenance (maintainability).
— The code has technical features that facilitate software verification (safety, reliability).
— The code has technical features that eliminate or minimize certain vulnerabilities (security).

4.2.3. Audits

Audits are a means of providing evidence to support claims regarding the compliance of software life cycle activities with defined process requirements. Effective audits have the following characteristics:

— Persons performing an audit have adequate independence from those individuals within the scope of the audit.
— The development or assessment processes/standards which the audit is performed against are clearly defined.
— A representative 'deep slice' sample of work is audited.
— Deficiencies against expected standards are recorded and corrective action is taken in a timely manner.

Where deficiencies reveal systematic shortfalls, it may be necessary to re-audit following the implementation of required corrective actions.

Typical aspects of a software development life cycle that may be audited include:

— Configuration control of design documentation;
— Configuration control of software;

— Compliance with coding standards;
— Compliance with test coverage recommendations;
— Software error recording and correction;
— Process for determining extent of repeat testing required following software modification.

Audit processes should be well defined and should be documented. Use of an accredited audit process can allow extra assurance to be taken from audit results. An example of a standard that defines recommended audit processes is: IEEE Std. 1028-2008, Standard for Software Reviews and Audits [56].

Examples of high level claims that could be supported by auditing include:

— Software development processes comply with specified standards. An audit provides evidence of compliance (safety, maintainability, reliability and security).
— Software complies with specified coding guides. An audit provides evidence of compliance (safety, maintainability, reliability, and security).

4.3. FUNCTIONAL VALIDATION

4.3.1. Modelling and simulation

Modelling and simulation techniques can play a significant role in the functional validation of the I&C requirements. This subsection addresses modelling and simulation at system level. Modelling and simulation at programming level is addressed in the section on testing.

Normally, behavioural requirements (functional and performance) are assigned to the I&C system, but in digital I&C systems, these requirements are mostly reassigned to software. Formal specification of the requirements using an executable specification language (possibly including animation) can also be considered a (precise) form of requirements modelling and simulation. According to Ref. [25], “Animations may be used for checking certain aspects of the requirements and may provide a reference to test the final software.”

A typical approach would include the following:

— Modelling of the I&C system environment, including process modelling and human actions modelling, modelling the requirements that need to be satisfied at the level of the environment, and modelling of the assumptions made regarding the environment.
— Modelling of the operating states of the I&C system and of its environment. Typically, requirements (for both the system environment and the I&C system) depend on these states. The states cover normal and abnormal operating conditions.
— Modelling of the I&C system functional requirements.
— Cosimulation of both the system environment and I&C functional requirements in various scenarios (including scenarios with component failure) to verify that the requirements at the level of the system environment are satisfied.

Various approaches may be used to determine the simulation scenarios to be used, such as:

— Monte Carlo approaches, where the scenarios are chosen at random, preferably using an automatic scenario generator.
— Statistical approaches, where the scenarios follow a given statistical distribution (supposedly representative of real operating conditions).
— Coverage approaches, where the scenarios collectively meet chosen coverage criteria. Examples of such criteria include coverage of failure modes of all components of the environment, coverage of the operating states, etc.
— Manual construction of scenarios.
— Accident scenarios based on safety analysis, postulated initiating events.

For the first three approaches mentioned, a random generator of compliant scenarios could be used. Also, verification of the satisfaction of the system environment requirements could be performed automatically (oracle).

These techniques rely on the correctness and the accuracy of the models and the simulation toolset used, and on the correctness of the scenario generator and of the oracle. Significant effort may be needed to develop the models, but the general trend in systems engineering is in favour of such an effort, for many purposes other than I&C systems. Also, the models and scenarios may be used for system testing.

Examples of high level claims that could be supported by modelling and simulation include:

— The safety I&C functions and performance levels required of the I&C system: determine that they are appropriate and cover the different plant situations and operator goals (safety).
— The I&C system architectural design: determine that the functional and performance requirements for the different subsystems ensure compliance of the I&C system with its own requirements (safety).

4.3.2. Functional failure mode and effects analysis

Failure mode and effects analysis (FMEA) is a systematic method for evaluating the effects of the identified failure modes of the system, of its constituent parts, or of its functions. It can help identify hazards and safety properties that can be the object of claims.

Design FMEA (or DFMEA) is a bottom up procedure. The system under analysis is hierarchically divided into components. The granularity of the division is determined so that the failure modes of the components at the bottom level can be identified. Then the effects of the failure modes of each component are systematically propagated up to the boundary of the system. DFMEA aims mainly at improving system design or justifying the effectiveness of the fault avoidance and fault tolerance measures built into the system.

Functional FMEA may be used to address software aspects and is a top down analysis. It starts by identifying the software intensive components in the system. For some such components, it may further identify their main software components. The next step is to identify the possible failure modes of each identified component. Then, as for DFMEA, the effects of the failure modes of each component are systematically propagated up to the boundary of the system.

DFMEA also supports CCF analysis, when the analyst postulates that software modules using identical software components fail simultaneously. This type of FMEA can be applied even in the very early stages of the life cycle with the aim of refining the specification and providing evidence of its completeness.

The definition of the failure modes of hardware components is simple because the dominant form of faults in hardware components are random faults, representing the effect of the environment, ageing, wear or other deterioration. Thus, the hardware failure modes are typically based on operational experience with the same and similar components. Component manufacturers often maintain a list of failure modes and occurrence frequencies for their products. For software, such information does not exist and failure modes are determined on a functional basis.

The following standards and guidelines are related to classical DFMEA:

— IEC 60812, Analysis Techniques for System Reliability: Procedure for Failure Mode and Effects Analysis (FMEA) (2nd edition, 2006) [57] describes FMEA and the failure mode, effects and criticality analysis and gives guidance as to how these techniques may be applied to achieve various reliability programme objectives.
— BS EN 60812:2006, Analysis Techniques for System Reliability: Procedure for Failure Mode and Effects Analysis (FMEA) [58] is the European adoption of IEC 60812, 2006 edition.
— SAE J1739, Potential Failure Mode and Effects Analysis in Design (Design FMEA), Potential Failure Mode and Effects Analysis in Manufacturing and Assembly Processes (Process FMEA) [59]. These are geared for the ground vehicle community.
— MIL-STD-1629A, Procedures for Performing a Failure Mode, Effects and Criticality Analysis [60], was published by the United States Department of Defense in 1980. The standard establishes requirements and procedures for performing a failure mode, effects and criticality analysis.

More information and examples for DFMEA can be found in the following:

— Nuclear Energy Agency of the Organisation for Economic Co-operation and Development (OECD/NEA) NEA/CSNI/R(2014)16, Failure Modes Taxonomy for Reliability Assessment of Digital I&C Systems for PRA (2015) [61].
— Electric Power Research Institute (EPRI) EPRI-TR-3002000509: Hazard Analysis Methods for Digital Instrumentation and Control Systems (2013) [62].

Examples of high level claims that could be supported by FMEA include the following:

— The I&C system design provides adequate measures against postulated single hardware faults (safety, availability).
— The I&C system design provides adequate measures against postulated single software faults (safety, availability).

4.3.3. Fault tree analysis

FTA analyses the events that may lead to hazards. It can be applied both in a qualitative and quantitative manner. The aim of the qualitative application of FTA is to discover the fault propagation paths in the system to determine the root causes of a failure, and to identify the best ways to reduce risk. Quantitative application of FTA computes the probability (or occurrence rate) of the analysed top level failure event, and provides information on the relative importance of the contributing faults. With reference to software dependability assessment, the representation of the system in an FTA can support derivation of dependability requirements for software implemented functions, as well as in some areas for evaluating software.

FTA is a top down procedure. It starts with the selection of the failure event for which causes are to be resolved. The event is resolved into its immediate and necessary sufficient causal events, using backward inference rules expressed in Boolean logic. This backward reasoning from events to their immediate causal events proceeds until basic causes (primary causes, i.e. faults) are identified. FMEA, hazard and operability analysis (HAZOP) and CCF analysis are inputs to this process. After the construction of the fault tree, the so-called minimal cut sets can be calculated, with the minimum set of basic events whose (simultaneous) occurrence ensures that the top event occurs.

Standards and guidelines related to FTA include the following:

— IEC 61025, Fault Tree Analysis (FTA) (2nd edition) [63]. Describes FTA and offers guidance on its application to perform an analysis, identifies appropriate assumptions, events and failure modes and provides identification rules and symbols.
— BS EN 61025:2007, Fault Tree Analysis (FTA) [64]. The European version of IEC 61025 (2nd edition).

Examples of high level claims that could be supported by FTA include the following:

— The causes of a given system failure mechanism have been removed (safety, reliability).
— The system failure probability for given failure modes is lower than or equal to the required value (safety, reliability).
— Minimal cut sets that could cause unavailability are identified (safety, availability).

4.3.4. Hazard and operability analysis

As mentioned in the previous section, HAZOP analysis is one of the inputs for the backward reasoning process when attempting to identify basic causes, or faults. The following discussion of HAZOP analysis is based to a very great extent on IEC Standard 61882 [65]. Much of the language in this section is in fact adapted from that standard, which can be considered an authoritative source on this topic.

A HAZOP analysis is a systematic examination of a process or operation — planned or existing — with the aim of identifying issues that could pose a risk to personnel or equipment, or may prevent efficient operation. This

method was originally developed to analyse chemical process systems, but was later broadened to cover other, more complex systems such as NPP operation, using software to record deviations and the resulting consequences. In particular, software HAZOP focuses on the identification of possible errors in the development of software used in the system.

This type of analysis is most effective when it is carried out early in the design phase so that its results can influence the design. It can also be used at an existing facility to recommend modifications that reduce risk and eliminate problems during operation.

A HAZOP analysis is a qualitative technique that uses guide words and is carried out by an interdisciplinary team. The guide words are key in executing a HAZOP analysis. According to IEC Standard 61882 [65]:

> "The identification of deviations from the design intent is achieved by a questioning process using predetermined 'guide words'. The role of the guide word is to stimulate imaginative thinking, to focus the study and elicit ideas and discussion."

The team assembled for a HAZOP analysis should be kept as small as possible, consistent with the members having the relevant skills and experience. The ideal size is four members, with a maximum of seven for any single analysis. The team should have a leader, someone to record the proceedings, and individuals who have experience in designing, using and maintaining systems.

The HAZOP analysis has four stages. Members specializing in risk assessment are appointed in the first, or definition, stage, which is also when the scope of the assessment is determined. In the preparation stage, supporting data and information are identified and collected. In this stage, consensus is also reached on the guide words to be used during the analysis. The third, or examination, stage includes the identification of all elements of the system or process to be examined and the application of the guide words to these elements. Deviations are also identified in this stage, though it is clear that not all combinations of guide words and elements will necessarily indicate possible deviations. However, as a rule, all conditions involving use or misuse should be identified and it should be determined if they are credible and require further assessment. In the fourth stage, documentation and follow-up, the outputs and conclusions of the analysis are listed and evaluated in terms of the risks assessed in the study. The conclusion of the HAZOP analysis should also verify that there is a process to ensure that recommended actions are completed in a satisfactory manner.

The standards and guidelines related to FTA include the following:

— IEC 61882, Hazard and Operability Studies (HAZOP Studies): Application Guide [65].
— BS EN 61882:2016, Hazard and Operability Studies (HAZOP Studies): Application Guide [66]. This standard reproduces verbatim IEC 61882:2016 and implements it as the UK national standard.

Examples of claims that could be supported by HAZOP analysis include the following:

— Identification of internal behaviour that could lead to unacceptable system failure (safety).
— Identification of internal behaviour that could lead to system unavailability (availability).

4.3.5. System-theoretic process analysis

A new hazard analysis technique, known as system-theoretic process analysis (STPA), identifies the scenarios that could lead to hazards. It is based on a new accident model — the system-theoretic accident model and process — which identifies causal factors. It is capable of analysing the entire accident process, including social and human factors. Much of the following discussion is taken from Ref. [67].

While traditional hazard analysis techniques have reliability theory as their foundation, STPA is based on systems theory, focusing on safety as a dynamic control problem rather than an issue involving the failure of a component, and considers the behaviour of the components and system as a whole. Thus, in STPA, accidents occur as a result of inadequate control, specifically the inadequate enforcement of safety constraints.

STPA identifies a larger set of causes, many not involving failures or unreliability. The technique was also developed to address accidents originating from common component interactions that can result from design flaws

or unsafe interactions between operational components that have not failed. (Traditional techniques have sought to prevent accidents caused by one or more components that fail.)

Since STPA is a top down engineering approach focusing on system safety, it can be used early in the design and development stage to 'steer' the system design process to generate high level safety requirements and constraints, and later to develop safety requirements for individual components. Similarly to other hazard analysis techniques, STPA can also be used for completed designs or existing systems, and in the investigation of accidents, generating causal scenarios that might be relevant to the events that have occurred [67].

The use of STPA for hazard analysis involves four steps:

(1) Establishment of the foundation for the analysis based on systems engineering and development. This involves defining the accidents to be considered, identifying the hazards associated with these accidents, and specifying the relevant safety requirements. Based on this, a functional control structure is developed, which is used in the STPA.
(2) Identification of potentially unsafe control actions. Once unsafe control actions are identified (the process does not have to be completely serial), STPA can highlight the potential causes of these unsafe actions.
(3) Creation of safety requirements and constraints based on the identified high level system hazards and unsafe control actions.
(4) Determination of how each potentially hazardous control action could occur. The aim here is to streamline the design to avoid hazardous actions during operation [67].

To summarize:

— STPA, when used early, can influence design decisions to take better account of safety issues.
— While early use of STPA is preferable, it can also be used in later hazard assessments.
— While sharing the goals of traditional FTA, STPA offers a wider range of potential scenarios, including those where there are no failures, but problems still occur because of unsafe or unintended interactions between system components.

Examples of significant claims that can be made using this technique include:

— Identification of behavioural requirements that are suitable for a specified safety goal (safety).
— Comprehensive identification of hazards (safety).

4.4. SOFTWARE ANALYSIS TECHNIQUES

4.4.1. General benefits of software analysis techniques

A general form of a claim supported by software analysis is: 'Item I has property P'. Property P may have defined conditions under which the claim is valid. Typical properties analysed by the techniques related to software dependability assessment are:

— Functional or correctness properties;
— Temporal properties (e.g. response times or how sequences of inputs are mapped to sequences of outputs);
— Accuracy;
— Integrity properties (e.g. freedom from software faults identifiable without knowledge of functional requirements, such as division by zero);
— Structural properties;
— Complexity properties.

Software analysis techniques aim at verifying the claim on a rigorous and exhaustive basis. This is opposed to testing, which verifies the claim based on a finite, non-exhaustive set of cases. Software analysis techniques also rely on defined, systematic and objective principles of analysis.

4.4.2. General limits of software analysis techniques

Though software analysis could in theory be performed manually, this would be very error prone, and the large size of software items under consideration and the complexity of most analysis principles mandate in practice the use of analysis tools. Several questions may need to be addressed regarding the use of such tools. In particular:

— Does the tool support analysis principles that are adequate for the verification of the property that is claimed for the item? In other words, is the tool functionally suitable to provide evidence for the claim?
— Is the tool a correct implementation of these principles? In other words, will the tool fail in a non-evident manner and provide incorrect and misleading results?
— Has the tool been correctly used? In other words, did the analysts provide the right tool parameters, the right inputs and the right sequence of commands? Did they correctly analyse the tool outputs?

Analysis tools do not always directly take item I and assess for property P: instead, models M(I) and M(P) must be provided as substitutions. Therefore, one needs to make sure that the models are adequate and will not lead to incorrect conclusions. The equivalence of models M(I) and M(P) on the one hand, and the software item I and property P on the other, with respect to the analysis technique applied and parameters needs to be assessed. Also, even if the analysis technique operates directly on item I (i.e. the source code), the equivalence of the executable binary code with the source code needs to be validated. This is achieved in practice either by direct confirmation of the equivalence or, more often, using a compiler with adequate 'pedigree'.

For analysis techniques that require substantial semantic analysis (i.e. understanding of the logic and behaviour of the software), tools often have 'accuracy' limits. A given analysis may end up with three sorts of answers: at these points in M(I), M(P) is satisfied; at these other points in M(I), M(P) is violated; and at these remaining points, the tool cannot decide. This is also the case when the complexity of the analysis exceeds the capabilities of the tool, and thus a more abstract (therefore simpler to analyse) model of I and/or P is used (abstraction). Indeed, some software properties have been shown to be mathematically undecidable in the general case. Thus, an analysis of tool results is always necessary:

— To decide whether M(P) violations are acceptable or not;
— To decide whether 'cannot decide' answers need further investigation, and to perform the investigation and make a decision when necessary.

4.4.3. Formal verification

Formal verification methods are systematic, mathematically based techniques. When necessary, the artefact to be verified is first modelled in a formal language (i.e. a language with well defined syntax and semantics), as are the properties to be verified. Techniques such as formal proof, constraint solving or model checking can then be used to determine if the specification satisfies the properties. If a property fails to hold, then some of the techniques, such as model checking, can provide a counterexample that can be analysed to better understand what caused the violation.

Formal verification techniques can produce evidence about all of the software dependability attributes at many different levels. Some can also support other techniques, such as testing, e.g. by automatically generating 'interesting' test cases.

Examples of high level claims that could be supported by formal verification include the following:

— The software code for a function produces the required output for all its specified inputs (safety, reliability).
— The specification of required behaviour is consistent with specific overall constraints (safety).
— The specification of required behaviour is complete in the sense that for all input scenarios, the required behaviour of the outputs is described (domain coverage or completeness) and free from conflicts (safety).

4.4.4. Static analysis

Static analysis is a technique performed without actually executing the software. In most cases, the analysis is performed on the source code, but in other cases it uses some form of the object code. The term refers to the analysis being performed by an automated tool, with manual (human) code analysis being called code review. Static analysis techniques provide evidence on properties of program behaviour that is independent of the specification. They address classes of vulnerabilities that can be detected without the cost and complication of formalizing the intended properties of the software. Examples of properties are:

— At the code or, possibly, model level:
- Freedom from division by zero, square roots and logarithms of negative numbers, etc.;
- Freedom from range overflow;
- Initialization of variables before use;
- Freedom from buffer overflows;
- Freedom from errors in typing, units and measures.

— At requirements, design or code level:
- Freedom from deadlock in normal operation;
- Termination or non-termination.

Typical claims that can be supported by static analysis techniques are:

— *Reliability.* Static analysis can detect poor coding habits and dangerous code constructs.
— *Safety*. Static analysis can find potential vulnerabilities in the code and check if secure programming practices are being followed.
— *Maintainability*. Complexity analysis of the code can be used to reveal hard to maintain parts of the code, and structural analysis can find redundant or unused sections of code.
— *Development metrics*. Structural and complexity analysis techniques compute many different metrics for the code.
— *Compliance with standards and guidelines.* Static analysis can be implemented (and such tools exists on the market) to check many of the practices recommended by coding standards and guides.
— *Quality.* Static analysis facilitates the use of coding standards and guides, and metrics can be generated. These techniques can be used as an indicator of code quality.

4.4.4.1. Structural analysis

Structural analysis is a subclass of static analysis techniques that analyse the data flow, measure the structural complexity and study execution pathways (control flow). The analysis of the code structure can show if the hierarchical organization of the software is inadequate, if hierarchical levels are not being respected, or if certain parts of the code are isolated and unreachable. Structural analysis can generate evidence of robustness by checking if internal redundancies in an FPGA are separate and can be an element in a CCF analysis.

4.4.4.2. Complexity analysis

Complexity analysis is a subclass of static analysis techniques that produce complexity metrics for the software being analysed. The complexity of the code is an indicator linked to quality, to the required test efforts and to the maintainability and reliability of the code. Acceptance criteria for complexity metrics may be specified in coding guides.

Examples of high level claims that could be supported by static analysis include the following:

— The code does not contain faults that could cause divisions by zero, out of bound array indexes, numerical overflows or underflows, etc. (safety, reliability).
— This part of the code does not influence the execution of this other part of the code.

— The (distributed) system architecture is tolerant of ‘Byzantine failures’ [1] (safety, reliability).
— The code complies with the complexity limits specified (reliability, maintainability).

4.5. TESTING

4.5.1. Functional testing

Functional testing verifies, for a finite number of test cases, that an item behaves as specified and provides outputs as specified. It can be performed at different stages of the software life cycle at different levels of software integration (unit, module, integration or system validation testing).

Testing can be performed on software separately (on a test bench that executes the software or simulates its execution), or on software within its final hardware. The advantage of testing software separately is that it can be done early in the life cycle, and that the test bench used can usually offer more control and observability on the tested item. The faithfulness of the test bench with respect to the real system hardware needs to be ascertained.

As exhaustive testing (100% coverage of all possible input sequences) is impossible in practice, sufficiency criteria often need to be relied upon. Numerous criteria have been proposed, some based on functional aspects, others on structural aspects. Coverage criteria may also depend on the programming style or language used (e.g. functional diagrams, general purpose programming languages such as C or ADA, or HDL languages for FPGA programming).

Examples of test coverage criteria that may be used include:

— *Statement coverage.* The percentage of code statements exercised during testing.
— *Branch coverage.* The percentage of code decision branches exercised during testing.

As coverage testing techniques typically require knowledge of the internal structure of the software, they are often classified as structure based techniques.

With regard to coverage testing, Ref. [25] recommends the following:

— The range of values and boundaries of all input variables should be tested.
— All system operation modes should be considered when developing test cases.
— Each system or module level test should monitor all outputs of the system or module.
— All interfaces and communication protocols should be tested.
— Fault tolerant and exception mechanisms should be tested.
— All lines of code and module calls should be exercised.
— Data structures and constant values should be verified.
— Operations on all data items should be conducted and access to the data items allowed.
— All time critical sections of code should be tested under realistic conditions.
— Calculations performed by the software should be verified against pre-calculated values.
— Coding comments and documentation should be verified.
— Compliance with appropriate coding guides and standards should be verified.

The specification of the software item under consideration should in principle be complete and cover all possible input conditions. This is not always the case, and well-chosen functional and structural criteria for coverage testing should ensure that the item is also tested in input conditions that are not explicitly addressed in the specification (negative testing).

4.5.1.1. Specification based testing

Also known as ‘black box’ testing, IEC 29119-1 [68] specification based testing involves testing the external inputs and outputs of a system based on a specification, rather than relying on knowledge of the internal states of the software. Specification based testing methods are advantageous for the following reasons [69]:

— Test results can be generated from the actual target.
— Testing can be performed by independent testers without detailed knowledge of the internal program structure.

The disadvantages of specification based testing include:

— Inability to detect errors that are not observable in external inputs or outputs;
— Lack of knowledge of the internal software structure, making it difficult to ensure that all execution paths are covered.

Although specification based testing alone cannot provide assurance of reliability, it can be used in all stages of the development cycle to ensure that externally observable specifications are met.

Examples of high level claims that could be supported by this technique include the following:

— Software provides the specified safety function (safety).
— System fails in a safe direction (safety, reliability).

4.5.1.2. Structure based testing

Also known as 'white box' testing, IEC 29119-1 [68] structure based testing requires a knowledge of the internal structure of the software being tested and can be used to test the source code or individual software functions at a lower level than specification based testing. Structure based testing methods are advantageous for the following reasons:

— Test results can provide insight into the details of software design and internal performance characteristics.
— Tests can provide the ability to detect internal errors that are not observable in external inputs or outputs.
— Knowledge of the internal software structure provides capabilities to exercise all execution paths and provide quantifiable metrics such as code coverage.

Disadvantages of structure based testing include the requirement of an in depth knowledge of the internal structure of the software, which may not always be available, and the requirement that testers possess significant domain knowledge in the programming language used to develop the software.

Examples of structure based testing techniques can be found in IEC 29119-4 [70]. Examples of high level claims that could be supported by this technique include the following:

— The software provides the specified safety function (safety).
— The system fails in a safe direction (safety, reliability).
— The software is deterministic under all combination of inputs (safety, reliability).

4.5.2. Negative testing

Negative testing can be defined as 'tests that violate the expected operating envelope'. Ideally, the software should include 'guards' against violations of the operating envelope assumed by the software designer — negative testing will reveal the response to such violations.

To undertake negative testing, the external interfaces to the software should be identified. These can include communication interfaces, user interfaces (such as graphical user interfaces), interprogram interfaces and real time plant interfaces. Negative tests should be applied at each interface to determine the response, which should indicate that the violation has been detected and controlled, e.g. by some form of error indication or rejection of the input.

Negative testing can be applied at a number of different levels:

— *Type violation*. Typical examples are a random sequence of letters in a numeric input field, a floating point number format used where an integer is expected or an invalid communication message format (e.g. wrong length wrong check code).

— *Value violation*. The type is correct but the value is not in the expected range. Typical examples are: an excessively long input string or unusual non-printing characters in a character field, negative numbers where positive values are expected, or values outside the expected range for a configuration parameter or a real time input.
— *Consistency violation*. A set of inputs has individually valid types and values, but the inputs are inconsistent with each other. For example, a parameter specifying an upper bound is less than another parameter specifying a lower bound.

The advantages of negative testing are that:

— It can demonstrate that the software is robust outside its expected operating envelope;
— It is an effective means of identifying software flaws related to 'edge cases' that might be outside the scope of conventional functional tests;
— It can be applied with no knowledge of the internal software structure.

The main disadvantages are:

— The normal envelope needs to be identified. This will require analysis of the software requirements and the interface specification.
— The response to violations may not be defined in the documentation so an assessment is needed to determine whether the response is adequate.

4.5.3. Statistical testing

Statistical testing is often used as a confidence building measure to complement rules compliance and quality assurance measures. It can also provide evidence supporting quantified claims on PFD. The general principle is as follows: if one successfully executes N independent test cases, the statistical distribution of which is representative of the distribution of real operating conditions, then one can infer a PFD with a certain level of confidence.

Though statistical testing is based on solid theoretical grounds, several practical issues need to be addressed, such as the correctness and independence of the oracle, the adequacy of the statistical distribution chosen, and the degree to which test conditions are representative with respect to operating conditions.

Statistical testing needs well defined test cases with well defined initial conditions. I&C software typically runs over long times with changes in the internal states of the I&C system. Thus, the definition of test cases which may be executed in a reasonable time may be difficult. Some technical details are provided in Annex I.

Examples of high level claims that could be supported by this technique include the following:

— Software provides the specified safety function (safety).
— System meets the specified reliability goals (safety, reliability).

4.5.4. Fault injection

Fault injection techniques are mainly used to provide evidence of robustness and fail safe properties. The overall principle is simple: during an execution (or more often, during a software simulation), one injects one or more incorrect values or hardware faults (which in turn will produce incorrect values) to determine whether the execution terminates with acceptable results or not.

Various fault injection strategies may be applied, depending on the evidence that is needed. An example of a high level claim that could be supported by this technique is the following:

— System fails safely (safety, reliability).

4.6. INSPECTIONS AND REVIEWS

Software development needs to be systematic and well documented for the software to be amenable to inspection and review. For the assessment of the dependability of safety software for I&C, the scope of review is to ensure safety and effectiveness. A systematic review approach is needed. Reviews are part of static testing, where the code is not executed. Within a conventional software development life cycle, there are a number of steps which use inspection/review techniques to provide assurance.

Reviews can be divided into four classes depending upon their level of formality (see Fig. 9):

(1) Informal;
(2) Walkthroughs;
(3) Technical review;
(4) Inspections.

For informal reviews, there is no systematic process, no documentation is required and the costs are relatively low. The objective is to find errors/defects. This type of review may be implemented by peers; one developer may review documents or the code of colleagues.

Depending upon the nature of the application, walkthroughs may be formal or informal. In many cases, clear understanding of specification documentation may be the goal of this exercise.

In technical reviews, technical experts follow a well defined process with more formal documentation. A moderator is responsible for the conduct of this review. The documentation may include memos, checklists, specification documents and, obviously, a findings list. This process involves more work, including brainstorming of problems and consideration of 'what if' scenarios.

The inspection process is a rigorous way of reviewing documentation and processes. The inspection report that is the outcome of this review consists of findings. Follow-up actions after inspections are also planned in order to ensure compliance and correction of deficiencies.

Enabling factors for success are:

— Clear objectives;
— Defined roles and responsibilities;
— Well prepared documentation;
— Working environment characterized by trust;
— Openness to discussions;
— Relevant experts.

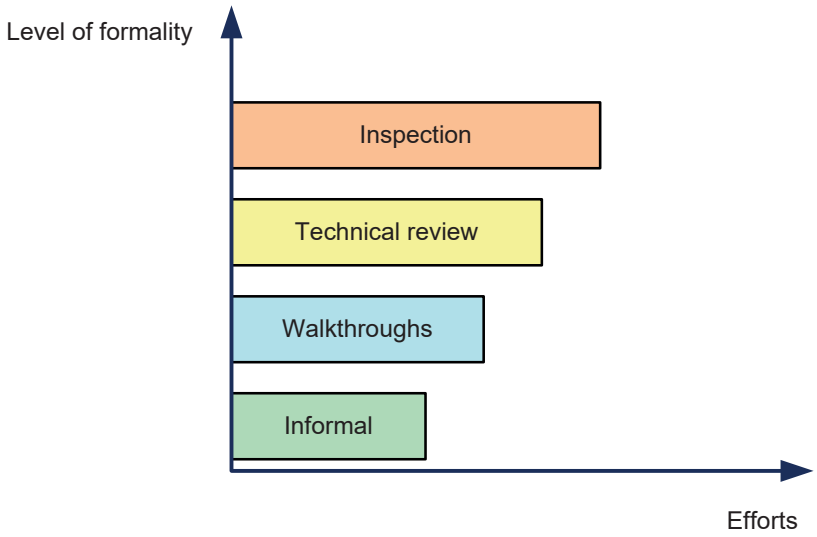


FIG. 9. Relative effort associated with different classes of review.

Where an informal review/inspection is performed by persons closely involved with the life cycle phase being examined, such as module peer reviews performed by software developers, then the evidence arising from such an activity would not carry much weight with respect to justifying the adequacy of a dependability property. Conversely, if there was a formally defined review/inspection process and there was clear independence between those performing the review/inspection and those who have performed the design work, then the successful outcome of such activities may be used as a persuasive element of the evidence supporting software dependability characteristic claims.

Where a review/inspection process is performed using a well defined, rigorous and effective process, e.g. if the clarity and extent of commenting on the software code is reviewed against clear coding guidelines, then such activities may be used as a persuasive element of the evidence supporting software dependability claims (in the example cited, the dependability characteristic claim would be that of software maintainability). Also, where the effectiveness of a review/inspection is supported by a qualified tool, such as checking for uninitialized variables by use of a static analysis software tool, then the effectiveness of such activities is generally considered to be enhanced (as the potential for human error is reduced).

The extent to which records from such activities can provide evidence to support claims that software has the necessary dependability properties depends upon a number of factors. Accurate and easily reviewable documentation should be produced for all stages of the development process. The documentation used to demonstrate adequacy to the regulator should be the same as the documentation used in the design. For computer based system applications, top down decomposition, levels of abstraction and modular structure are important concepts for coping with the problems of unavoidable complexity. They not only allow the system developer to tackle several smaller, more manageable problems, but also allow a more effective review by the verifier. The logic behind the system modularization and the definition of interfaces should be made as simple as possible.

The personnel working on a software development project for a system with a high level of safety importance should include application specialists as well as computer software and hardware specialists. This combination of expertise helps to ensure that safety requirements, which are generally well known, owing to the maturity of the industry, are effectively communicated to the computer specialists. It should be ensured that all personnel understand how their jobs are related to the fulfilment of the safety requirements, and they should be encouraged to question activities or decisions that may compromise the safety of the system. This means that the software specialists should also have a good understanding of the application. An appropriate specification language (e.g. a graphical language) can be used for the description of safety functions.

Examples of high level claims that could be supported by this technique include the following:

— Software development process complies with specified standards (safety, maintainability, reliability, security).
— Software complies with specified coding guides (safety, maintainability, reliability, security).

5. LESSONS LEARNED

Of the assessment principles, AP8 involves applying lessons learned. This section discusses the sources of the lessons learned. They can be derived from experience with software development, plant operation, regulatory review and platform certification. Digital systems are widely used in other industries, and relevant experience from those industries is available. Consensus practice embodied in standards accounts for insights from this experience. Relevant experience can be used to identify vulnerabilities and challenges to inform an assessment (see Section 3 and Annex III).

5.1. SOFTWARE DEVELOPMENT

Software is developed using a variety of different methods and technologies which are intended to reach different goals, such as time to market, broad applicability and, more seldom, high safety or high reliability. Thus, not all software development processes are useful for nuclear applications. A systematic and disciplined

development process is necessary to produce high dependability software. To accomplish this, many of the technical attributes discussed in this report need to be a critical part of the development of safety software.

Programmable digital safety systems have been installed in NPPs since the 1980s. While the first applications were based on microprocessors, some recent applications have been based on FPGAs. In this publication, the term *software* also applies to the programming of FPGAs.

In many instances, the software of NPP digital safety I&C systems has been developed specifically to the standards of the nuclear industry. The development processes are based on a life cycle that is structured in different phases. The life cycle phases and traceability between requirements, design items, and implementation items up to test cases support stringent verification and validation of the developed software.

Products developed for, and used by, non-nuclear applications sometimes need to be integrated into nuclear safety systems because no specific nuclear products are available. These products are often called COTS products. Before such products can be used, specific assessment procedures are applied.

There is a wide range of national practices and requirements, and the IAEA, IEC, ISO and IEEE provide detailed and internationally accepted guidelines and standards for software development and assessment. The standardized software development process has benefitted from the lessons learned from prior experience.

5.2. OPERATIONS

The software in digital safety I&C systems in NPPs is usually specific to the application, and the same software is not used in different applications. This presents a challenge when trying to apply lessons learned for a specific product. Operating experience has been collected, and some potential trends were found that apply to the performance of all digital safety I&C systems.

The OECD/NEA Computer-based Systems Important to Safety (COMPSIS) project was initiated to collect and analyse data regarding failure events relating to computer based systems in nuclear facilities. The study identified four main root causes of the events, including hardware failures, human error, software failures and system issues, and concluded that weaknesses in requirements are one of the most significant contributors to software and systems failures. The report of the COMPSIS project [41] recommends that more failure events be collected and analysed to provide more insights into computer based system failures in nuclear facilities.

There are also publications on operational experiences in individual Member States. For example, a comprehensive analysis of CCFs in the United States of America is given in Ref. [71]. This publication analyses reports of plant events involving digital I&C systems spanning a 20 year period. A similar analysis for the Republic of Korea is given in Ref. [72]. Like the COMPSIS project, these two studies concluded that hardware problems and human errors were the dominant sources of CCFs affecting digital safety I&C systems. They also concluded that errors in I&C functional requirements specification played a significantly larger role than errors in the software itself. Lastly, they highlighted the fact that for many events, software changes were part of the solution, even when software had not been part of the problem. These conclusions are confirmed by the fact that after a few functional changes at the beginning of operation, many digital safety I&C systems are operated without any software modification over years and decades.

5.3. REGULATORY REVIEW

Safety systems in NPPs need licences from the relevant national regulatory bodies. Owing to the high level of complexity of digital safety systems and their associated software, regulatory reviews often represent a major effort both from the licensees (who must provide all relevant information and justification) and the regulator (who must review, understand and analyse a very large volume of documentation). One reason for this is that the regulators are often provided with an extremely large amount of design information and development artefacts without clear linkage and argumentation between these and their own requirements. Another reason is that in some instances, regulators are confronted directly with a detailed design, or even an implementation.

In addition, some regulatory or standard requirements are expressed in terms of principles or objectives, without being associated with objective and unambiguous acceptance criteria, to the point that for some regulatory or standard requirements, there is a lack of agreed interpretation [73]. That may result in a need for significant

design changes in the course of a construction project, and consequently additional costs and delays. In many cases, the need for such changes is revealed late in a project, when there is no time for extensive redesign. The solution then is to add extra systems or equipment. This could lead to designs that are more complex, costlier and more difficult to operate than necessary.

Some Member States have specific national regulatory requirements for software and software based equipment in addition to the internationally accepted standards. That results in different licensing processes, requiring different information and sets of documents, and sometimes different designs.

Significant licensing difficulties are also caused by the many trade offs that are necessary to satisfy as best as possible unavoidable conflicting objectives. Justifying that these trade offs are necessary and acceptable with respect to safety is sometimes a difficult and labour intensive task.

Therefore, although many digital safety systems and their software have been successfully licensed, significant licensing difficulties and risks remain. The use of the assessment framework described in Section 3 may help to overcome these issues. It supports structuring the review process so that evidence is determined that is reasonable for all stakeholders involved in the licensing process. It also allows early discussions with regulators on architectures and preliminary designs, when designs may still be changed and when any necessary changes do not incur significantly high penalties.

Licensing issues are also addressed in Annex II to this report.

5.4. I&C PLATFORM CERTIFICATION EFFORTS

Several vendors of I&C systems provide platforms, i.e. a set of hardware and software modules, tools and guidelines on how to design I&C functions and systems with the platform. This holds for the safety I&C systems of nuclear plants as well as safety and safety related applications outside the nuclear domain.

To design a safety I&C system for NPPs from the ground up is expensive and only makes sense if it is intended for a series of identical plants. Otherwise, a platform is a good basis that can be tailored effectively to different functions and requirements.

In some Member States, the licensing of a safety I&C system can be based on a generic certification of the platform. However, this is not a universal practice, and other Member States do not offer the possibility of platform certification. Even when it is recognized, the span of time between a certification and a license application is often such that the platform has been subjected to design changes that require a new certification.

In all cases, the platform itself should be associated with the documentation, arguments and evidence (as defined in the assessment framework described in Section 3) that are necessary for the licensing of the I&C system.

5.5. OTHER INDUSTRY APPLICATIONS

In the nuclear field, I&C contributes to the three safety functions described in IAEA Safety Standards Series No. SSR-2/1 (Rev. 1) [74]:

(1) Control of reactivity;
(2) Removal of heat from the reactor and from the fuel store;
(3) Confinement of radioactive material, shielding against radiation and control of planned radioactive releases, as well as limitation of accidental radioactive releases.

These safety functions are unique for the nuclear field. Dependable software based systems are essential to perform these functions. Additionally, means such as defence in depth, redundancy and diversity contribute to the dependability of these functions.

There are other industries with strong requirements for software dependability such as the aviation, railway and automobile industries. The number of aircraft, railway engines and automobiles provides data for statistical analysis. Owing to the differences regarding safety functions and safety objectives it is difficult to extrapolate the experiences to the nuclear field.

The safety goals and environmental conditions are often very different. For an airplane, the safety goal is to keep it flying, for a train it may be to stop the train but not in a tunnel in case of fire and for automobiles the safety goal may be to stop as fast as possible. The environmental constraints on the system are also very different. For example, in the aviation industry, the system size and weight are very important, but they are typically not constraints in NPPs. This constraint changes the implementation of the system. Functions of different safety classifications may be mixed in an aviation system, but they are kept separate in NPPs. Therefore, other solutions are implemented in NPPs where sufficient space with controlled environmental conditions is available.

All these industries require compliance with the software development process. The development process for software is regulated by guidelines and standards (e.g. IEC 61508 [9], MISRA-C [51, 52], etc.). Exhaustive verification and validation including testing is an essential part of the software development process. In addition to the standards on software quality processes, other industries have developed many tools to aid in the development process.

The success of programmable digital systems in other industries shows the effectiveness of a rigorous development process for software. This lesson learned is reflected in the standards that are used for software developed for use in NPPs. Additionally, the nuclear power industry benefits from the software development tools of other industries.

6. CONCLUSIONS

Dependability represents the overall trustworthiness of a system or function and is an essential characteristic of safety systems at NPPs that needs to be properly justified. The attributes of dependability include safety, reliability, availability, maintainability and security. The focus of this report is on the contribution of software to the dependability of a function implemented in a safety system.

In this report, software includes executable software as well as related software, firmware, documentation (requirements, design, user manuals, etc.) and data. As such, software in the scope of this report is composed mainly of code executed on a CPU (sequence of instructions), the logical structure of an FPGA/programmable logic device (massive parallel logic) as well as all combinations (e.g. an FPGA with an integrated intellectual property core) that may be implemented in an I&C system. The software also comprises all data determining the execution of calculations in the I&C system.

This report establishes a framework for the dependability assessment of software in safety systems based on the following guiding principles:

— AP1: System and software requirements are adequately defined, valid and reviewable such that it can be confirmed that they collectively address the necessary functional and non-functional properties.
— AP2: An effective understanding of the role of the safety I&C system is demonstrated, including its direct and possible indirect roles, and its role in supporting safe operation.
— AP3: The intended and unintended behaviour of the system is adequately taken into account, including adverse unintended consequences under conditions of failure.
— AP4: The implementation of the requirements is adequate.
— AP5: Active challenge to the dependability assessment is part of decision making throughout the organization.
— AP6: The objectives of the users of the assessment are identified such that the structure and presentation of the assessment account for their needs (e.g. considering all users/uses and factors such as reviewability and maintainability of the assessment).
— AP7: The findings of the assessment are organized in a logical, coherent, traceable and accessible manner and are repeatable, with rigour commensurate with the degree of trust required of the system.
— AP8: Lessons learned are incorporated in the target system being assessed and in the assessment process itself.
— AP9: Any changes in the target system or conditions of use/maintenance that would invalidate the assessment are detected and addressed. Fundamental assumptions underlying the assessment are identified.

In applying these principles, the assessment should be embedded in a disciplined and systematic process. This report defines an overall strategy to guide the assessment, describes an approach to developing and communicating the assessment based on CAE, and provides guidance on a high level process for deploying the framework. The strategy is property based, vulnerability aware and standards informed. The properties to be considered in the assessment are determined by the APs and relevant system features. The system features consider the behaviour of the system as well as its interactions, its vulnerabilities and insights from standards.

There are four main phases to the assessment: mobilization, definition of the assessment strategy, performance of the assessment and maintenance of the assessment. This report describes the CAE approach, discusses determination of dependability claims, addresses deployment of the assessment framework and considers the means to build confidence in the assessment. The report identifies a wide range of techniques to develop evidence to support an assessment.

A key product of the approach described in this report is to organize the assessment into a clear, logical, coherent, traceable and accessible framework based on explicitly stated CAE. That framework could be used as an important communication tool between the various stakeholders (e.g. owners, operators, designers and assessors).

A major conclusion of this report is that the dependability assessment should be considered an integral part of the life cycle of the safety I&C system. One benefit to this is that the early insights gained from the development of an assessment strategy can influence the design and implementation of the I&C system. Awareness of the necessary evidence can inform life cycle activities and the selection of tools and techniques. One result is improved cost effectiveness. Another benefit is better management of project risk. It needs to be noted that dependability results from a combination of many components and verification and validation is one of them. From a licensing point of view, and to obtain reasonable assurance that the final product is suitable for use in safety applications, verification and validation is a key factor that facilitates regulatory decisions.

Other key messages of this publication are:

— Software dependability should be understood in the context of system dependability.
— The importance of having the right set of system requirements must not be underestimated.
— Generating and maintaining sources of evidence for dependability assessment should be addressed in all stages of the I&C project life cycle.

To summarize, this report presents a framework with which to perform a dependability assessment of software in the safety system of an NPP. The assessment framework provides a systematic and coherent structure within which to organize technical and process evidence to satisfy arguments in support of claims about software dependability properties. Additionally, the formalized approach to assessment can be extended beyond the treatment of software dependability to many system and architectural characteristics.

REFERENCES

[1] AVIZIENIS, A., LAPRIE, J.-C., RANDELL, B., LANDWEHR, C., Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. Dependable and Secure Comput. **1** 1 (2004) 11.

[2] INTERNATIONAL ATOMIC ENERGY AGENCY, Design of Instrumentation and Control Systems for Nuclear Power Plants, IAEA Safety Standards Series No. SSG-39, IAEA, Vienna (2016).

[3] INTERNATIONAL ATOMIC ENERGY AGENCY, Core Knowledge on Instrumentation and Control Systems in Nuclear Power Plants, IAEA Nuclear Energy Series No. NP-T-3.12, IAEA, Vienna (2011).

[4] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Nuclear Power Plants: Instrumentation and Control Important to Safety — General Requirements for Systems, IEC Standard 61513, IEC, Geneva (2011).

[5] INTERNATIONAL ATOMIC ENERGY AGENCY, Effective Handling of Software Anomalies in Computer Based Systems at Nuclear Power Plants, IAEA-TECDOC-1140, IAEA, Vienna (2000).

[6] INTERNATIONAL ATOMIC ENERGY AGENCY, IAEA Safety Glossary: Terminology Used in Nuclear Safety and Radiation Protection: 2007 Edition, IAEA, Vienna (2007).

[7] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, INTERNATIONAL ELECTROTECHNICAL COMMISSION, Information Technology: Vocabulary, Standard 2382:2015, ISO/IEC, Geneva (2015).

[8] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Nuclear Power Plants: Instrumentation and Control Important to Safety — Development of HDL-programmed Integrated Circuits for Systems Performing Category A Functions, IEC Standard 62566, IEC, Geneva (2012).

[9] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, IEC Standard 61508, IEC, Geneva (2010).

[10] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, INTERNATIONAL ELECTROTECHNICAL COMMISSION, Systems and Software Engineering: Systems and Software Quality Requirements and Evaluation (SQuaRE) — Evaluation Process, Standard 25040, ISO/IEC, Geneva (2011).

[11] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, INTERNATIONAL ELECTROTECHNICAL COMMISSION, Systems and Software Engineering: Systems and Software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE, Standard 25000, ISO/IEC, Geneva (2014).

[12] INTERNATIONAL ATOMIC ENERGY AGENCY, Protecting Against Common Cause Failures in Digital I&C Systems of Nuclear Power Plants, IAEA Nuclear Energy Series No. NP-T-1.5, IAEA, Vienna (2009).

[13] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Nuclear Power Plants: Instrumentation and Control Systems Important to Safety — Requirements for Coping with Common Cause Failure (CCF), IEC Standard 62340, IEC, Geneva (2007).

[14] EUROPEAN ATOMIC ENERGY COMMUNITY, FOOD AND AGRICULTURE ORGANIZATION OF THE UNITED NATIONS, INTERNATIONAL ATOMIC ENERGY AGENCY, INTERNATIONAL LABOUR ORGANIZATION, INTERNATIONAL MARITIME ORGANIZATION, OECD NUCLEAR ENERGY AGENCY, PAN AMERICAN HEALTH ORGANIZATION, UNITED NATIONS ENVIRONMENT PROGRAMME, WORLD HEALTH ORGANIZATION, Fundamental Safety Principles, IAEA Safety Standards Series No. SF-1, IAEA, Vienna (2006).

[15] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, INTERNATIONAL ELECTROTECHNICAL COMMISSION, INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Systems and Software Engineering: Vocabulary, Standard 24765, ISO/IEC/IEEE, Geneva (2017).

[16] INTERNATIONAL ATOMIC ENERGY AGENCY, Computer Security of Instrumentation and Control Systems at Nuclear Facilities, IAEA Nuclear Security Series No. 33-T, IAEA, Vienna (2018).

[17] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Nuclear Power Plants: Instrumentation and Control Systems Important to Safety — Software Aspects for Computer-based Systems Performing Category A Functions, IEC Standard 60880, IEC, Geneva (2006).

[18] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, IEEE Standard Criteria for Programmable Digital Devices in Safety Systems of Nuclear Power Generating Stations, IEEE Standard 7-4.3.2-2016, IEEE, New York (2016).

[19] BLOOMFIELD, R.E., BISHOP, P.G., JONES, C.C.M., FROOME, P.K.D., Adelard Safety Case Development (ASCAD) Manual, Adelard, London (1998).

[20] BLOOMFIELD, R.E., et al., Guidance on Claims, Arguments and Evidence, in preparation, http://www.ClaimsArgumentsEvidence.org

[21] BISHOP, P.G., BLOOMFIELD, R.E., "The SHIP safety case: A combination of system and software methods", Safety and Reliability of Software Based Systems (Proc. 14th IFAC Conf. Bruges, 1995), Springer, New York (1995).

[22] BISHOP, P.G., BLOOMFIELD, R.E., VAN DER MEULEN, M.J.P., An Example Application of the CEMSIS Guidance (2004), http://www.cemsis.org/public/deliverables/D5.6P%20wp5_ade039_v10_application_of_guidance.pdf

[23] BISHOP, P.G., BLOOMFIELD, R.E., "A methodology for safety case development", Safety-Critical Systems (Proc. 6th Symp. Birmingham, 1998) (REDMILL, F., ANDERSON, T., Eds), Springer, London (1998).

[24] KELLY, T., “The goal structuring notation: A safety argument notation”, Proc. Workshop on Assurance Cases: Best Practices, Possible Obstacles and Future Opportunities, Florence, 2004, IEEE, New York (2004).
[25] SWEDISH RADIATION SAFETY AUTHORITY, Licensing of Safety Critical Software for Nuclear Reactors: Common Position of International Nuclear Regulators and Authorised Technical Support Organisations — Rev. 2015 (2015), https://www.stralsakerhetsmyndigheten.se/contentassets/ddf0489e3437468dbf77fc88c4b390cb/201625-licensing-of-safety-critical-software-for-nuclear-reactors
[26] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, INTERNATIONAL ELECTROTECHNICAL COMMISSION, Systems and Software Engineering: Systems and Software Assurance — Part 2: Assurance Case, Standard 15026-2:2011, ISO/IEC, Geneva (2011).
[27] OBJECT MANAGEMENT GROUP, Structured Assurance Case Metamodel (SACM), OMG, Needham, MA (2015).
[28] TOULMIN, S.E., The Uses of Argument, Cambridge University Press, Cambridge (1958).
[29] BLOOMFIELD, R.E., NETKACHOVA, K., “Building blocks for assurance cases”, Proc. IEEE 25th Int. Symp., Naples, 2014, IEEE, New York (2014).
[30] NETKACHOVA, K., NETKACHOV, O., BLOOMFIELD, R.E., “Tool support for assurance case building blocks: Providing a helping hand with CAE”, Computer Safety, Reliability and Security (SAFECOMP 2015) (Proc. Workshops, Delft, 2015) (KOORNNEEF, F., VAN GULIJK, C., Eds), Springer, Berlin (2015).
[31] NETKACHOVA, K., MÜLLER, K., PAULITSCH, M., BLOOMFIELD, R.E., “Investigation into a layered approach to architecting security-informed safety cases”, Proc. IEEE/AIAA 34th Conf., Prague, 2015, IEEE, New York (2015).
[32] BLOOMFIELD, R.E., LITTLEWOOD, B., “Confidence: Its role in dependability cases for risk assessment”, Proc. IEEE Computer Society Int. Conf., Edinburgh, 2007, IEEE, New York (2007).
[33] HELTON, J.C., OBERKAMPF, W.L., Alternative representations of epistemic uncertainty, Reliability Eng. Sys. Safety **85** (2004) 1.
[34] LITTLEWOOD, B., WRIGHT, D., The use of multi-legged arguments to increase confidence in safety claims for software based systems: A study based on a BBN of an idealised example, IEEE Trans. Software Eng. **33** 5 (2007) 347–365.
[35] BISHOP, P.G., BLOOMFIELD, R.E., LITTLEWOOD, B., POVYAKALO, A., WRIGHT, D., Towards a formalism for conservative claims about dependability, IEEE Trans. Software Eng. **37** 5 (2011) 708–717.
[36] GOODENOUGH, J.B., WEINSTOCK, C.B., KLEIN, A.Z., Eliminative Argumentation: A Basis for Arguing Confidence in System Properties, Rep. CMU/SEI-2014-TR-013, Carnegie Mellon Software Engineering Institute, Pittsburgh, PA (2014).
[37] HENRION, M., FISCHHOFF, B., Assessing uncertainty in physical constants, Am. J. Phys. **54** 9 (1986) 791–798.
[38] LIN, S.-W., BIER, V.M., A study of expert overconfidence, Reliability Eng. Sys. Safety **93** (2008) 711–721.
[39] KAHNEMAN, D., Thinking, Fast and Slow, Penguin, London (2012).
[40] LEVESON, N., White paper on the use of safety cases in certification and regulation, J. Sys. Safety **47** 6 (2011).
[41] OECD NUCLEAR ENERGY AGENCY, Computer-based Systems Important to Safety (COMPSIS) Project: Second Period Operation (2008–2011), Rep. NEA/CSNI/R(2012)12, OECD, Paris (2012).
[42] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, IEEE Standard Dictionary of Measures of the Software Aspects of Dependability, IEEE Standard 982.1-2005, IEEE, New York (2005).
[43] NUCLEAR REGULATORY COMMISSION, Software Engineering Measures for Predicting Software Reliability in Safety Critical Digital Systems, Rep. NUREG/GR-0019, NRC, Washington, DC (2000).
[44] NUCLEAR REGULATORY COMMISSION, A Large Scale Validation of a Methodology for Assessing Software Reliability, Rep. NUREG/CR-7042, NRC, Washington, DC (2011).
[45] MURDOCH, J., Safety Measurement: Practical Software and Systems Measurement — White Paper, PSM Support Center, New York (2006).
[46] DODD, I., HABLI, I., Safety certification of airborne software: An empirical study, Reliability Eng. Sys. Safety **98** (2012) 7–23.
[47] BASILI, V., DANGLE, K., ESKER, L., MAROTTA, F., RUS, I., Measures and Risk Indicators for Early Insight into Software Safety, University of Maryland, College Park, MD (2008).
[48] OMAN, P., HAGEMEISTER, J., “Metrics for assessing a software system’s maintainability”, Proc. IEEE Conf., Orlando, 1992, IEEE, New York (1992) 337–344.
[49] WANG, C., WULF, W. A., “Towards a framework for security measurement”, Proc. 20th Natl Information Systems Security Conf., Baltimore, 1997, National Institute of Standards and Technology, Gaithersburg, MD (1997) 522–533.
[50] ZHUO, F., LOWTHER, B., OMAN, P., HAGEMEISTER, J., “Constructing and testing software maintainability assessment models”, Proc. IEEE Symp., Baltimore, 1993, IEEE, New York (1993) 61–70.
[51] MOTOR INDUSTRY SOFTWARE RELIABILITY ASSOCIATION, Guidelines for the Use of the C Language in Critical Systems, Rep. MISRA-C:2012, Motor Industry Research Association, Nuneaton (2013).
[52] MOTOR INDUSTRY SOFTWARE RELIABILITY ASSOCIATION, Guidelines for the Use of the C++ Language in Critical Systems, MISRA-C++, Motor Industry Software Reliability Association, Nuneaton (2008).
[53] SEACORD, R.C., CERT C Coding Standard, 98 Rules for Developing Safe, Reliable and Secure Systems, 2nd edn, Addison-Wesley Professional, Boston (2014).
[54] XILINX, HDL Coding Practices to Accelerate Design Performance, Rep. XILINX WP231 (1.1), XILINX, San Jose (2006).

[55] ACTEL, Actel HDL Coding, Style Guide, Actel Corporation, Mountain View, CA (2009).
[56] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Standard for Software Reviews and Audits, IEEE Standard 1028-2008, IEEE, New York (2008).
[57] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Analysis Techniques for System Reliability: Procedure for Failure Mode and Effects Analysis (FMEA), IEC Standard 60812, 2nd edn, IEC, Geneva (2006).
[58] BRITISH STANDARDS INSTITUTION, Analysis Techniques for System Reliability: Procedure for Failure Mode and Effects Analysis (FMEA), BSI Standard BS EN 60812:2006, BSI, London (2006).
[59] SOCIETY OF AUTOMOTIVE ENGINEERS, Potential Failure Mode and Effects Analysis in Design (Design FMEA), Potential Failure Mode and Effects Analysis in Manufacturing and Assembly Processes (Process FMEA), SAE Standard J1739, SAE International, Warrendale, PA (2009).
[60] DEPARTMENT OF DEFENSE, Procedures for Performing a Failure Mode, Effects and Criticality Analysis, Military Standard MIL-STD-1629A, DOD, Washington, DC (1980).
[61] OECD NUCLEAR ENERGY AGENCY, Failure Modes Taxonomy for Reliability Assessment of Digital I&C Systems for PRA, Rep. NEA/CSNI/R(2014)16, OECD, Paris (2015).
[62] ELECTRIC POWER RESEARCH INSTITUTE, Hazard Analysis Methods for Digital Instrumentation and Control Systems, Rep. 3002000509, EPRI, Palo Alto, CA (2013).
[63] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Fault Tree Analysis (FTA), IEC Standard 61025, 2nd edn, IEC, Geneva (2006).
[64] BRITISH STANDARDS INSTITUTION, Fault Tree Analysis (FTA), BSI Standard BS EN 61025:2007, BSI, London (2007).
[65] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Hazard and Operability Studies (HAZOP Studies): Application Guide, IEC Standard 61882, IEC, Geneva (2016).
[66] BRITISH STANDARDS INSTITUTION, Hazard and Operability Studies (HAZOP Studies): Application Guide, BSI Standard BS EN 61882:2016, BSI, London (2016).
[67] LEVESON, N., An STPA Primer, Version 1 (2013), http://sunnyday.mit.edu/STPA-Primer-v0.pdf
[68] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, INTERNATIONAL ELECTROTECHNICAL COMMISSION, INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Software and Systems Engineering: Software Testing, Part 1: Concepts and Definitions, Standard 29119-1, ISO/IEC/IEEE, Geneva (2013).
[69] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, IEEE Standard for System, Software, and Hardware Verification and Validation, IEEE Standard 1012, IEEE, New York (2016).
[70] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, INTERNATIONAL ELECTROTECHNICAL COMMISSION, INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, Software and Systems Engineering: Software Testing, Part 4: Test Techniques, Standard 29119-4, ISO/IEC/IEEE, Geneva (2015).
[71] ELECTRIC POWER RESEARCH INSTITUTE, Operating Experience Insights on Common-cause Failures in Digital Instrumentation and Control Systems, Rep. 1016731, EPRI, Palo Alto, CA (2008).
[72] ELECTRIC POWER RESEARCH INSTITUTE, Digital Operating Experience in the Republic of Korea, Rep. 1022986, EPRI, Palo Alto, CA (2011).
[73] GUERRA, S., MENON, C., “Why are I&C modernisations so difficult? Experiences with requirements engineering and safety demonstrations in Swedish NPPs”, Nuclear Plant Instrumentation, Control and Human–Machine Interface Technologies (Proc. 9th Int. Conf. Charlotte, 2015), American Nuclear Society, La Grange Park, IL (2015).
[74] INTERNATIONAL ATOMIC ENERGY AGENCY, Safety of Nuclear Power Plants: Design, IAEA Safety Standards Series No. SSR-2/1 (Rev. 1), IAEA, Vienna (2016).

Annex I

QUANTIFICATION OF SOFTWARE ASPECTS OF SYSTEM RELIABILITY

I–1. INTRODUCTION

This annex describes approaches to providing justifiable reliability numbers for software of computer based safety systems in nuclear power plants. The main body of this guidance describes the various roles for these numbers in, for example, the software related aspects of the computer based systems in probabilistic safety assessments or claims about common cause failure (CCF). Based on the analysis of the system and software structure, the sources of CCF and defences, the system elements that require quantification should be identified. The reliability claim should be justified using the dependability assessment framework, and this should include the derivation of the claim, the impact of the test bed quality and the sentencing of the results. It should also address:

— Partitioning of the possible software failure mechanisms;
— Identification of design and operational measures taken to prevent failures;
— Identification of dominant failure mechanisms;
— Quantification of the contribution of each dominant system failure mechanism;
— The assumption uncertainties are identified and taken into account.

The reliability of systems and components identified in this analysis can be estimated using one or more of the strategies outlined below. The reliability values obtained will have many assumptions and caveats associated with them, and it is important that these be captured within the assessment (e.g. using the claims, arguments and evidence (CAE) framework of Section 3) and form part of the overall judgement of reliability.

The scope of this guidance is software within instrumentation and control (I&C) systems that implement Category A (Cat. A) functions. The reliability assessment may have a wide range of target claim figures because:

— Requirements for I&C Cat. A functions reliability will vary between reactor types and countries.
— The overall I&C system architecture may have different approaches to defence in depth and levels of diversity, leading to differences of reliability targets for Cat. A functions.
— The assessment may only be assessing a subsystem or component that has lower reliability requirements than the system as a whole owing to defence in depth and diversity.
— As described in Section 2.8, the software will have features that contribute to dependability that may have lower reliability targets (e.g. detection of failures, or support for degraded modes of operation).

There are some general requirements to note:

— An approach to software reliability in which system reliability targets are apportioned to lower level components needs to have a corresponding argument within the assessment that the combination of component reliability achieves the system target.
— In general, the components will not fail independently; therefore, a straightforward approach of combining reliability figures (e.g. by multiplication) will not be valid. The difficulty of this part of the assessment needs to be considered in designing the overall system architecture and in designing the assessment strategy.
— As discussed in Section 3.5, the confidence in the reliability has both an aleatory and epistemic component. In high reliability systems, it is often the epistemic part that dominates, as there comes a point where increases in failure free operational data or statistical testing do not increase the reliability that can be claimed owing to assumption doubt.

The approaches are grouped into three classes:

(1) Methods of estimation that are mature (statistical testing, analysis of prior operational experience, worst case bound);
(2) More speculative approaches (probability of perfection, the chain rule);
(3) Indirect methods (infinite time bound, composite indirect models).

These are discussed in the sections that follow.

I–1.1. Statistical testing

In statistical testing, the system or component is subjected to tests that are assumed to accurately represent its usage (operational profile) when connected to the plant. This test evidence can be used to set a confidence bound on the achieved reliability. This approach can be applied at the overall system level, where the number of possible tests is often time limited, or at the software level using platform simulation, which requires an additional assumption about the fidelity of the simulation. Generally speaking, statistical testing is more easily applied to demand based systems as, with assumptions about long term issues, sufficient demands can be made in a realistic time frame. However, with simulation using parallel processing, it may be feasible to apply the approach to continuous time systems where a bound on the (dangerous) mean time to failure is required.

A bound on the probability of failure on demand (PFD) for a system can be established by testing a protection system under a statistically realistic demand profile.

If no test failures are observed over t test demands, then to a given confidence C in the probability of failure on demand (pfd) is bounded by:

$$\mathrm{PFD} < (-\ln(1 - C))/t \qquad \text{(I–1)}$$

where

PFD is the probability of failure per demand (a value between 0 and 1);
C is the confidence, a probability;
t is the number of failure free tests.

Some typical confidence values and associated PFD bounds are shown in Table I–1.

TABLE I–1. PFD CONFIDENCE BOUNDS GIVEN t SUCCESSFUL TESTS

Confidence C	PFD bound
99%	$4.6/t$
95%	$3.0/t$
90%	$2.3/t$
69%	$1.0/t$

This PFD bound can be sensitive to uncertainties in the presumed operational profile. The sensitivity can be reduced by performing extra tests for demand types that are thought to be infrequent [I–1, I–2].

If the results of statistical testing are used to support probabilistic risk assessment (PRA), research suggests that assumption doubt could be considered using the chain of confidence approach.

As with operational experience, the predictions from statistical testing can be made more credible by analysing the software and hardware to assess the significance of any history within the software and the presence or absence of any long term issues (e.g. memory leaks) that might invalidate the predictions.

As pointed out in Section 4.5.3, although statistical testing is based on solid theoretical grounds, several practical issues need to be addressed, such as the correctness and independence of the oracle, the sentencing process and the extrapolation from the system and configuration that is being tested to the actual deployed system.

I–1.2. Analysis of prior operational experience

Where standard components are used to build a system or the system is part of an established product line, past operational experience can be used to estimate software reliability. This evidence will include:

— Field failure reports;
— The component fault/modification records;
— Component usage data (or sales data).

Given data on component usage, the time in operation between successive fault reports can be estimated. This can be used to infer the interval between software failures.

This type of evidence is only likely to be available for components that form part of the 'platform' rather than components that are application specific. The main uncertainties associated with this method are:

— The level of end user reporting;
— The similarity of platform usage to that in the new application.

If it can be shown (e.g. by a combination of static analysis and confirmatory tests) that the platform software is indifferent to the plant state, platform failures could be random with respect to the plant state. Hence, the PFD of the platform software is the product of the estimated failure rate and the demand interval. For example, if the estimated platform software failure rate is 10^{-5}/hour and the demand duration is one hour, the PFD for the platform component would be 10^{-5}.

Both operating experience and statistical testing are significant potential sources of rebuttal to reliability claims and should be assessed even when the positive conclusions from their assessment may be much lower than the target being claimed.

I–1.3. Worst case bound theory

A worst case bound reliability model [I–3, I–4] could be used to calculate the expected reliability based on usage time T and the estimated number of faults N. The usage time T could, for example, be estimated from platform sales figures, while N could be estimated from code complexity or code size and 'typical' fault density figures for similar products with the same software development process quality [I–5]. For example, if the claimed fault density for the software development is 0.1 serious faults per thousand lines of code, the resulting expectation is for ten serious faults in the software for a system with 100 000 lines of code (when first deployed). In practice, this approach is more relevant to platform software or software components as the usage time T for the application software prior to deployment is likely to be too small to justify the required level of reliability.

Given a consistent operational profile, the upper bound on the expected failure rate is bounded by:

$$E(\lambda_T) \leq N/(p_{fix} \times e \times T) \qquad \text{(I–2)}$$

where

N	is the estimated number of faults;
p_{fix}	is the probability of fixing a fault;
$E(\lambda_T)$	is the expected failure rate (failures per unit time) after usage time T;
e	is the exponential constant;
T	is the usage time (or test time).

Note that this estimate is predictive (made before any testing is undertaken or experience is available) [I–3]. The bound relates to the worst case expected value at a future time T (the actual failure rate achieved could differ from this value). The number of failures observed during T is largely irrelevant, although N will need to be revised if more than N faults are found in subsequent operation.

A similar formula [I–1] can be used to predict a worst case PFD bound:

$$E(\mathrm{PFD_T}) \leq N/(p_{\mathrm{fix}} \times \mathrm{e} \times D) \quad D >> 1 \qquad \text{(I–3)}$$

where

N	is the estimated number of faults;
p_{fix}	is the probability of fixing a fault;
$E(\mathrm{PFD_T})$	is the worst case expected PFD after D demands;
e	is the exponential constant;
D	is the number of demands.

The model is also useful in providing a comparison in terms of the rigour needed in software development for different reliability targets — if Cat. A software has a reliability target of two orders of magnitude more than industrial software it would need a fault density two orders of magnitude smaller, all things being equal.

I–1.4. Estimating the 'probability of perfection'

Many of the activities and techniques used to develop trustworthy software focus on elimination of defects rather than directly on achieving reliability. Recent research has begun to formalize the types of judgement that might be made about perfection and relate them to operational probabilistic measures such as PFD [I–6]. In a safety context, 'perfection' is the absence of 'dangerous' faults that result in the unsafe behaviour of the component. The probability of perfection can be used to scale other estimates of reliability, which implicitly assume the software is imperfect.

This estimate is based on:

— Empirical evidence, such as the proportion of similar systems that contain known dangerous faults.
— Process evidence, where an argument is made that the development process used is sufficient to ensure that the software correctly implements its specification (e.g. program proof or process modelling). Such estimates are of course subject to the proviso that the functional specification is valid (i.e. meets plant requirements).

Given a PFD estimation procedure (e.g. based on testing without failure) and an estimate that the software is perfect (p_{p}),

$$E(\mathrm{PFD}) = (1 - p_{\mathrm{p}}) \times E(\mathrm{PFD_{test}}) \qquad \text{(I–4)}$$

where

PFD	is the probability of failure per demand (a value between 0 and 1);
p_{p}	is the probability that the software is 'perfect';
$\mathrm{PFD_{test}}$	is an estimate of the PFD of the software based on testing without failure (e.g. from operational experience or statistical testing).

Using Bayesian analysis [I–7], given a test interval t without failure:

$$E(\text{PFD}) \approx (1-p_\text{p})/\text{e}t \quad \text{as } p_\text{p} \to 1 \tag{I–5}$$

where

e	is the exponential constant;
t	is the test interval, without failure;
p_p	is the probability that the software is 'perfect';
E(PFD)	is the expected PFD.

This is very similar to the worst case bound model (Section I–1.3) except that N is fractional. In addition, the probability of perfection bounds the long term reliability of the entire fleet [I–7].

I–1.5. Assumption doubt: The chain rule

The chain rule can be used to deal with uncertainty in assumptions (e.g. whether or not two channels are independent). So, if $P_\text{assump\_OK}$ is the confidence that the assumptions underlying some PFD estimation rule are applicable, then:

$$E(\text{PFD}) = P_\text{assump\_OK} \times \text{PFD1}|\text{assump\_OK} + (1-P_\text{assump\_OK}) \times \text{PFD2} \tag{I–6}$$

where

E(PFD)	is the expected value of the PFD;
PFD1|assump_OK	is the probability of failure on demand given a set of assumptions;
$P_\text{assump\_OK}$	is the probability that the assumptions underlying the estimate of PFD1 are valid;
PFD2	is some alternative PFD estimation method using a different rule.

For the case where the rule assumptions do not apply, PFD2 is some alternative PFD estimation method using a different rule, which could itself be subject to uncertainty and be addressed by a further rule. This can lead to a sequence of estimates (a chain), and if there is no further rule the final PFD|not assump_OK value would be unity.

For example, if we were estimating the probability of failure of a system with two diverse channels, A and B, the expected value for the PFD is given by:

$$E(\text{PFD}_{AB}) = (1 - P_\text{indep}) \times \text{PFD}_A \times \text{PFD}_B + (1 - P_\text{indep}) \times P_\text{tests\_OK} \times \min(\text{PFD}_{A\text{test}}\ \text{PFD}_{B\text{test}})$$

$$+ (1 - P_\text{tests\_OK}) \times 1 \tag{I–7}$$

where

$E(\text{PFD}_{AB})$	is the expected probability that both A and B fail per demand;
P_indep	is the probability that channels A and B fail independently;
PFD_A	is the PFD of channel A;
PFD_B	is the PFD of channel B;
$\min(x,y)$	is a function that gives the minimum value of x and y;
$\text{PFD}_{A\text{test}}$	is the PFD of channel A estimated from statistical testing;
$\text{PFD}_{B\text{test}}$	is the PFD of channel B estimated from statistical testing;
$P_\text{tests\_OK}$	is the probability that the assumptions that make statistical testing valid are true.

I–1.6. Indirect estimation methods

Indirect reliability estimation methods can be useful in cases where no direct reliability evidence is available. They should be part of a general justification of the reliability and include a careful analysis of the assumptions and limitations of the arguments.

I–1.6.1. Infinite time bound

The infinite time bound model [I–8] could be used to bound the total number of failures over the lifetime of the software fleet:

$$n_{max} \leq N/p_{fix} \quad \text{(I–8)}$$

where

n_{max} is the maximum number of failures over infinite time (assuming faults are detected and fixed);
N is the number of faults present at the start of fleet deployment;
p_{fix} is the probability that a fault is fixed when a failure occurs in the fleet.

This model can be applied to both platform and application software and is not dependent on a specific operational profile. It assumes that faults will be fixed as soon as they are reported.

The model requires an estimate of the number of faults N and the failure fixing probability p_{fix}. As in the previous section, N could be based on fault density estimates of similar systems [I–5], or more sophisticated software development process models [I–9–I–11] for application software.

I–1.6.2. Composite indirect models

Bayesian belief networks [I–12] can be used to combine a number of factors that can influence software reliability, and these factors are combined based on expert opinion. The approach can be used to combine the uncertainties from different sources, e.g. from the assumption doubt of the representativeness of the operational profile in statistical testing. Attempts have been made to develop models that combine sources of expert judgement into a reliability figure, but these suffer from technical issues related to justifying the network topology and independence assumptions, as well as from the problem that the input judgements are not calibrated or empirically justified. This is most evident where adherence to a collection of development and verification and validation techniques is used to argue a reliability figure, e.g. compliance with safety integrity level (SIL) 3 techniques achieves SIL 3 reliability. A more appropriate approach for safety systems, if the efficacy of development methods is known, is to have a technical argument to support a worst case bound or, more speculatively, a probability of not perfect.

I–2. COMBINING RELIABILITY ESTIMATES: FAILURE DEPENDENCY

Combining the software reliability of components into an overall system reliability is problematic because of the need to make a judgement of the level of dependence between the components. The approach used in hardware of assuming failure independence is not justified in software. In fact, independence is a very special case of a range between total dependence (unlikely) and total independence (also implausible) [I–13]. Possible approaches to consider are:

— Estimating probability of independence [I–14];
— Combining these using the claims of PFD and probability of not perfect, in which independence, at least an aleatory level, is easier to justify [I–6, I–15].

REFERENCES TO ANNEX I

[I–1] BISHOP, P.G., "Rescaling reliability bounds for a new operational profile", Software Testing and Analysis (Proc. ACM SIGSOFT Int. Symp. Rome, 2002), Association for Computing Machinery, New York (2002) 180–190.

[I–2] BISHOP, P.G., POVYAKALO, A., Deriving a frequentist conservative confidence bound for probability of failure per demand for systems with different operational and test profiles, Reliability Eng. Sys. Safety **158** (2017) 246–253.

[I–3] BISHOP, P.G., BLOOMFIELD, R.E., A conservative theory for long term reliability growth prediction, IEEE Trans. Reliability **45** 4 (1996) 550–560.

[I–4] BRADY, R.M., ANDERSON, R., BALL, R., Murphy's Law: The Fitness of Evolving Species and the Limits of Software Reliability, University of Cambridge, Cambridge (1999).

[I–5] ELECTRIC POWER RESEARCH INSTITUTE, Estimating Failure Rates in Highly Reliable Digital Systems, Rep. 1021077, Palo Alto, CA (2010).

[I–6] LITTLEWOOD, B., POVYAKALO, A., Conservative reasoning about the probability of failure on demand of a 1-out-of-2 software-based system in which one channel is "possibly perfect", IEEE Trans. Software Eng. **39** (2013) 1521–1530.

[I–7] STRIGINI, L., POVYAKALO, A., "Software fault-freeness and reliability predictions", SAFECOMP 2013, Computer Safety, Reliability, and Security (Proc. 32nd Int. Conf. Toulouse), Springer, Berlin (2013) 106–117.

[I–8] BISHOP, P.G., "Does software have to be ultra-reliable in safety critical systems?" ibid., pp. 118–129.

[I–9] CHU, T.L., YUE, M., MARTINEZ-GURIDI, G., LEHNER, J., Development of Quantitative Software Reliability Models for Digital Protection Systems of Nuclear Power Plants, Rep. NUREG/CR-7044, NRC, Washington, DC (2011).

[I–10] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Nuclear Power Plants: Instrumentation and Control Important to Safety — Classification of Instrumentation and Control Functions, IEC Standard 61226, IEC, Geneva (2009).

[I–11] BLOOMFIELD, R., GUERRA, S., "Process modelling to support dependability arguments", Dependable Systems and Networks (Proc. Int. Conf. Washington, DC 2002), IEEE, New York (2002) 113–122.

[I–12] PEARL, J., Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Representation and Reasoning Series, Morgan Kaufmann, San Francisco, CA (2007).

[I–13] ECKHARDT, D.E., LEE, L.D., A theoretical basis for the analysis of multiversion software subject to coincident errors, IEEE Trans. Software Eng. **12** (1985) 1511–1517.

[I–14] LITTLEWOOD, B., POVYAKALO, A.A., Conservative bounds for the PFD of a 1-out-of-2 software based system based on an assessor's subjective probability of "not worse than independence", IEEE Trans. Software Eng. **39** (2013) 1641–1653.

[I–15] LITTLEWOOD, B., RUSHBY, J., Reasoning about the reliability of diverse two-channel systems in which one channel is "possibly perfect", IEEE Trans. Software Eng. **38** (2011) 1178–1194.

Annex II

LICENSING EXPERIENCE

Licensing of software based safety systems has been a significant challenge in a number of Member States for several years. The regulatory structure to support the unique aspects of software based safety systems first evolved in the early 1990s and was at first based primarily on standards compliance. Although a number of software based systems were introduced into NPPs before this time, most were reviewed and assessed using ad hoc methods to determine their dependability.

As more information became available on the significant advantages of these systems as well as the potential new hazards introduced by them, national and international standards were developed and endorsed for use in national regulatory review structures. However, licensing of these systems continued to pose significant challenges to both licensees and regulatory bodies.

II–1. PROJECT DELAYS DUE TO UNRESOLVED LICENSING APPROACHES

There are a number of examples where lack of clarity in the licensing basis or its implementation has led to project delays and costs. This problem affects many technical issues but can be mitigated by a clear definition of licensing principles and the claims and arguments that are being used. Experience has shown that lack of clarity in argumentation has often been a major problem. Some of the challenges and issues that can lead to this are elaborated below. Most national regulators will meet with licensees and vendors early in the development project to discuss assessment claims and argumentation to ensure that all parties are in general agreement as to the level of rigour required and the documentation needed for the evidence provided [II–1]. These meetings are almost always worth the time and effort involved to have them. They are generally of the greatest value early in the development project.

II–2. LACK OF UNDERSTANDING OF THE RIGOUR REQUIRED IN THE NUCLEAR INDUSTRY OR IN THE LICENSEE MEMBER STATE

In many cases, component manufacturers underestimate the evidence required by the nuclear industry, both in terms of the extent of evidence necessary and the need to supply design and implementation information. There are risks that the supply chain may not be able or willing to respond with information to support the safety case. This is particularly true when the supply chain supplies primarily non-nuclear or non-safety critical users.

There are a number of possible remedies for this vulnerability:

— Develop a strategy of engagement and partnership with the supply chain;
— Use the elaboration in guidelines and standards to explain what is needed;
— Use examples of best practices to demonstrate what is required;
— Work with the regulatory body to agree on what other evidence can be substituted for missing component information in developing the safety case or regulatory compliance.

II–3. UNDEFINED OR INADEQUATE REQUIREMENTS, SAFETY CLAIMS NOT CLEAR

Software project failure is often caused by problems in the requirements (in over 70% of cases according to some sources). Sometimes these are linked directly to functional requirements, but often they relate to non-functional aspects at the system level that have a major impact on the subsequent software architecture. In many cases, software architecture decisions are based on inadequate understanding of interface requirements or of regulatory requirements. It is imperative that these requirements be validated early in the design process and discussed with all members of the development team (and ideally the regulator).

Another source of incompleteness in the requirements is a failure to draw the boundary of the system wide enough in capturing requirements. When projects ignore operational and maintenance procedures in requirements for computer based refurbishments (for example, methods of maintenance may not translate easily from an analogue electrical system to a digital one), they frequently encounter problems. There is experience from the nuclear industry where this caused a new system to be unusable. These requirements may not be explicit in existing requirements documents; more likely they are contained in the design of the systems and procedures.

II–4. USE OF COTS/SOUP COMPONENTS

The inclusion of pre-existing software, software of uncertain pedigree (SOUP) or COTS software in safety systems can all lead to risks in the licensing. Dealing with COTS adequately is a challenging topic, with basic issues arising from:

— Lack of understanding of what the software component contains and does (both intended functionality, including functionality intended for more general use but not for the nuclear safety application, and unintended functionality);
— Lack of understanding of software failure modes and the consequences of failure;
— Lack of methods for establishing the degree of trust required in the component (e.g. via software criticality analysis) and the degree that can be justified;
— Lack of supporting information when trying to use operational data in support of licensing;
— Inability to track software changes and limited information on configuration management activities by the developer.

II–5. PREVIOUS CERTIFICATION OR ACCEPTANCE

A particular challenge to licensing software based safety systems is the reuse of components or systems that have been previously certified or approved for use in a safety related application. Although this is the preferred path in a number of Member States, there are potential problems due to:

— Lack of rigour in the original review or certification (for example safety integrity level (SIL) 3 approved devices where the software has not been inspected);
— The inevitable changes to the software configuration between the certified device and the one to be deployed;
— Lack of consensus on how differences between a certified version of a software based device and the one deployed is re-reviewed and documented.

II–6. USE OF ‘PROVEN IN USE’ AND OPERATING EXPERIENCE ARGUMENTS FOR SOFTWARE

When a software based component or a previously used component has been in operation for a long time, it is a potentially important source of evidence. However, there are a number of issues when using this information, which include the following:

— Disagreement on how to consider modifications to the component and changes to its use and environment. Since software failure modes are usually not fully known, proven in use arguments are difficult to extend beyond the particular application. At a minimum, there is a need for an impact analysis.
— Because of the uncompromising (unrealistic) requirements of some standards (such as IEC 61508 [II–2]), ‘proven in use’ is not generally accepted as part of the safety certification.
— Because of the lack of detailed theory on extending a proven in use argument beyond a specific software application, it is frequently very challenging to make the arguments technically rigorous and convincing.

Evidence of operation is often what might be called ‘grey’ evidence — superficially appealing but difficult to make convincing. Often it is not the duration of operation that is so much in doubt but the completeness of the reporting of failures and the applicability of the operating environment to other situations that is called into question.

Some regulations (e.g. for civil aviation in the United Kingdom) require experience to be examined for lessons learned and possible contradictory evidence.

II–7. ARGUMENTS THAT ARE NOT WELL DEFINED

One aspect of a lack of clarity in the basis for licensing is lack of definition of the arguments. The possible strategies for demonstrating that the safety claims are satisfied can be established early in a project, before the evidence is available. However, if this is not done, significant problems can result as the project progresses, for example:

— A ‘proven in use’ argument is vulnerable to changes in the system and challenges to the quality of the data.
— Compliance with standards may not be achievable.

This last point is important. Often, standards require that a wide range of techniques be applied. Users may argue that complying with these techniques produces systems of the required integrity, though this is not necessarily the case (in other sectors there have been IEC 61508 [II–2] SIL 2 compliant systems with failure rates 100–10 000 times the IEC SIL). At best, they can lead to the belief that the system has the required integrity, but additional arguments and evidence will be required.

II–8. ARCHITECTURE AS DESIGNED VERSUS AS IMPLEMENTED

Hazard analysis and other supporting investigations into a system will depend on representations that are abstract models of the implementation. In particular, there have been cases of systems where the architecture provided a good overall indication of intent but in practice the dependencies in the implementation were not well captured. For example, there have been examples where the safety classification of 25% of the code changed following detailed investigations.

The best remedies to deal with this issue are an awareness of the issue and a requirement to validate representations throughout the project life cycle.

II–9. SECURITY THREATS AND VULNERABILITIES

Historically, the security of safety systems in nuclear power plants has been assured by the use of physical measures, separation from the outside world and a general reliance on the security measures of the plant as a whole. While in the past security has often been the responsibility or concern of other agencies, it is now perfectly credible that a security incident can compromise safety and thus should be part of the design basis. The appropriateness of a given computer security measure will depend on safety, security and operational considerations. The implementation of these measures should not adversely affect the essential safety functions and performance of the software and associated instrumentation and control system. If a conflict between safety and security exists, then design considerations taken to ensure safety should be maintained, but a compatible solution to meet computer security requirements should also be identified.

In recent years, formal requirements for computer security, apart from general security or those integrated into general security, have become more common. The most important aspect of this is that most safety review criteria have been updated to ensure that if software and computer security is included, it does not challenge the safety of the software or computer systems.

A challenge in this area is the use of COTS components that have been developed over many years whose provenance is not entirely known. The amount of unused code is surprising. Indeed, there are well known issues of

‘Easter eggs’ (hidden functionalities that do not relate to the main purpose of the product) with large commercial software that might be used in lower integrity applications.

Also, COTS maintainability features can cause vulnerabilities, providing possibilities for inadvertent effects during maintenance and for unauthorized access. There are examples of critical systems that have had safety reducing modifications propagated in a population of devices.

II–10. NON-COMPLIANCE WITH STANDARDS

Lack of compliance with standards can be a serious issue and ensuring compliance is often a significant part of the licensing justification effort. The common standards IEC 61508 [II–2] and IEC 60880 [II–3] (or in other Member States IEEE 603 [II–4] and IEEE 7-4.3.2 [II–5]) need to be interpreted in the context of nuclear COTS equipment. Both technical and regulatory inputs are needed, as is a detailed knowledge of the safety system and its justification. Areas where the standards have frequently needed interpretation — often by reversing their recommendations — include:

— Unused code;
— Interrupts;
— Languages;
— Verification and validation;
— ‘Proven in use’ claims.

II–11. DISCOVERING SOFTWARE (OR FPGAS) IN UNEXPECTED PLACES

Sometimes, there are surprises, as when software appears where it is not expected (e.g. in so-called smart devices which, from a black box viewpoint, may look and feel like plug and play replaceable items). This has been a very significant issue with field programmable gate arrays (FPGAs) and complex programmable logic devices in recent years. There might also be unexpected software in a product that has been developed for many years (COTS/SOUP), and often networks and connectivity are larger than thought. This might not be as common in nuclear systems, but it is a very common problem in other areas. In a control room, knowing exactly what system is connected, and the scope of this connectivity together with the exact configuration, can be an issue.

II–12. LIFE CYCLE OF EVIDENCE

Sometimes, how the evidence supports the arguments throughout the lifetime of the safety system is not explicitly taken into consideration. This can lead to a justification that ‘decays’.

In some Member States, this is a known generic issue and is tackled systematically. However, the lifetime costs of evidence should be a factor in designing the argumentation strategy. For example, static analysis may be expensive the first time through, but much cheaper to maintain. The testing evidence cost may vary with the power of the SIL and can also be expensive to maintain in the light of minor changes (e.g. having to repeat statistical testing).

REFERENCES TO ANNEX II

[II–1] NUCLEAR REGULATORY COMMISSION, Licensing Process Interim Staff Guidance, Interim Staff Guidance DI&C-ISG-06, NRC, Washington, DC (2011).

[II–2] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, IEC Standard 61508, IEC, Geneva (2010).

[II–3] INTERNATIONAL ELECTROTECHNICAL COMMISSION, Nuclear Power Plants: Instrumentation and Control Systems Important to Safety — Software Aspects for Computer-based Systems Performing Category A Functions, IEC Standard 60880, IEC, Geneva (2006).

[II–4] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations, IEEE Standard 603, IEEE, New York (1998).

[II–5] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, IEEE Standard Criteria for Programmable Digital Devices in Safety Systems of Nuclear Power Generating Stations, IEEE Standard 7-4.3.2-2016, IEEE, New York (2016).

Annex III

VULNERABILITIES AND CHALLENGES

This annex provides examples of vulnerabilities and challenges based on nuclear industry experience. The examples provided are indicative rather than definitive and can be extended by considering vulnerabilities found in other software based systems, for example, challenges related to the use of tools. Additionally, vulnerabilities found in non-nuclear safety critical systems have not been considered here. By their very nature, the following lists of application specific challenges and platform specific challenges may need to be further developed for specific claims about specific systems.

III–1. APPLICATION SPECIFIC CHALLENGES

Category A (Cat. A) applications are usually constructed by 'wiring' together generic logic elements to construct a specific application. Thus, some claims about the application can be devolved into claims about these logic elements. Hence, it is important to challenge the assumptions that underpin such arguments.

III–1.1. Richness of the logic element language

The core elements needed to implement Cat. A functions are often considered to be simple compositions of logic elements. However, the actual elements being composed might be more complex than the requirements of the application logic suggest. This can be for a variety of reasons:

— The logic elements provide additional functionality that is not required in Cat. A applications.
— The logic elements need to access operating services.
— The programming platform is not as straightforward as the abstraction suggests, meaning that considerable 'knitting' is required to implement the logic elements.

Thus, in order to justify specific claims about properties such as predictability, one needs to consider the full range of logic elements, identify a subset of elements that satisfy the relevant property, impose limitations on the way in which these elements can be combined in order to ensure that the property is still valid for any composition of elements, and then impose design restrictions on the use of the logic element language to ensure that Cat. A applications can only be constructed using this limited subset of the full programming model.

For example, a typical programming model includes the following logic elements:

— Basic arithmetic and logic;
— Data conversion;
— Complex mathematical functions such as square root and trigonometric functions.

Although the third category is more complex and would therefore require more justification for use in a Cat. A application, all three categories are pure functions with no hidden state, which makes it easier to reason about their composition. However, there is also a need for logic elements such as the following:

— Memory;
— Counters;
— Timers.

Thus, the programming model is already quite complex, before considering logic elements that are specific to a particular application domain. An example would be logic elements for processing analogue signals, such as:

— Filters;

— Function generators;
— Integration and differentiation.

Reasoning about the behaviour of such logic elements requires the use of control theory and an awareness that a continuous function is being approximated by a recursive function that operates in discrete time and stores the result of the previous function invocation to feed it back into the calculation of the next function invocation.

Finally, there are logic elements for interacting with devices and performing various kinds of input/output operations:

— Polling;
— Event handling;
— Message passing.

III–1.2. Vulnerabilities: Logic elements and their composition

Table III–1 summarizes some generic vulnerabilities that can be used to challenge claims about application level programming models.

Table III–2 lists some specific vulnerabilities based on our knowledge and experience of assessing logic elements intended for use in Cat. A systems. All the vulnerabilities identified have been found in actual platforms being assessed for use in nuclear applications, so it is certain that these vulnerabilities have occurred in real systems.

TABLE III–1. GENERIC VULNERABILITIES

Issue	Rationale
Unpredictable execution time	It should be possible to predict the execution time of each logic element; otherwise, it can be difficult to calculate the worst case execution time for the application logic
Dependencies on other logic elements	There should be no hidden interfaces or dependencies between logic elements; otherwise, it is difficult to reason about compositions of elements
Uncertain error handling	All error conditions should be identified and handled transparently — there should be no hidden error conditions

TABLE III–2. SPECIFIC VULNERABILITIES IN LOGIC ELEMENTS

Category	Explanation	Examples
Incomplete or inadequate user documentation	The documentation must be complete and correct; otherwise, logic elements may not behave as expected	Undocumented error reactions Undocumented data types Undocumented or incorrect details
Range checks	Operands and results should always be within the appropriate range for the data type concerned	Missing range checks on input values Incorrect range checks on output values Failure to check that the number of inputs is appropriate for the type of output
Under specified behaviour	The behaviour of each logic element should be determined entirely by its inputs	Arbitrary non-deterministic behaviour if some of the inputs are not connected

III–2. PLATFORM SPECIFIC CHALLENGES

Claims about the application have to be guaranteed by the underlying platform that is responsible for executing or interpreting the application. In effect, an argument has to be constructed to show that the realization of the application on the platform takes into account whatever claims have been made about the abstract model of the application. For example, if it is claimed that the behaviour of the application is deterministic, then it has to be shown that the interpretation or execution of the compiled representation of the application on the platform is also deterministic.

III–2.1. Generic issues

The prompt tree in Fig. III–1 illustrates some generic issues that need to be considered at both the platform and application levels. The key challenge is to show that claims about the properties of individual components are still valid when those components are combined within a particular context. This is both an abstract modelling

FIG. III–1. Prompt tree for composition challenges.

problem and an implementation issue. First, it is necessary to show that the claims about composition are valid for the abstract model. Then it is necessary to show that the implementation conforms to the abstract model.

Problems with composition can arise because of interference between components. The prompt tree suggests potential areas of difficulty, such as competition for resources, concurrency issues such as race conditions, and lack of robustness or completeness caused by failure to consider all possible modes of operation or ranges of input values.

III–2.2. Platform vulnerabilities

To illustrate some of these threats to compositionality, a list of potential vulnerabilities is presented in Table III–3 based on current knowledge and experience from assessing platforms intended for use in Cat. A systems. These vulnerabilities have been discovered in actual platforms being assessed for use in nuclear applications, so it is known that these vulnerabilities have occurred in real systems.

TABLE III–3. EXAMPLES OF PLATFORM VULNERABILITIES

Category	Consequence	Examples
Corruption of the memory	If one process can overwrite the memory space of another, then process isolation (virtual machine abstraction) is no longer guaranteed	Uninitialized pointers. Array subscript out of bounds.
Uninitialized values	If the behaviour of the process depends on an uninitialized value, the results will be unpredictable	Conditional initialization, depending on code path. Local variables that are not initialized explicitly.
Register integrity	Register values must be preserved when calling between C and assembler; otherwise, the values of variables will change unexpectedly	Assembly code called from C that does not preserve registers that the C compiler expects to be unchanged. Assembly code that calls C code without saving registers that the C compiler is allowed to change. Register values that are saved on the stack should be restored to the same register.
Stack discipline	Every function call and interrupt should leave the stack unchanged; otherwise, it could grow or shrink over time, leading to corruption of the memory	Registers that are pushed onto the stack are not popped off the stack. The stack pointer is adjusted by the wrong amount after a function call.
Locking protocol	Every resource that is locked must also be unlocked; otherwise, resources could become blocked	Lock and unlock statements do not refer to the same resource. A timeout or error occurs before the resource is unlocked, leaving it locked.
Resource leak	Every resource that is used must be released; otherwise, the system might run out of resources	A file is opened but not always closed, depending on the flow of control, resulting in the loss of a file description.

III–3. UNINTENDED CONSEQUENCES OF FEEDBACK

III–3.1. Issues

Although many instrumentation and control (I&C) systems, particularly interlock and protection systems, are designed to behave predictably, sources of feedback can lead to unexpected behaviours. There is a need to identify and review:

— Sources of feedback in the logic:
  - Those intended by the control engineer and explicit in the requirements.
  - Those arising from implementation issues and differences between design abstraction and actual implementation. For example, the requirements might assume 'instantaneous' semantics for the graphical language, whereas the implementation might be sequential, even if it is very rapid.
— Indirect feedback from the plant and environment. Again, some of this might be intended, but interdependencies and feedback under all modes of operation should be probed.

The investigation of sources of feedback should consider normal and failure conditions of the plant, equipment and operator.

III–3.2. Vulnerabilities

Table III–4 provides some examples of vulnerabilities.

TABLE III–4. EXAMPLES OF VULNERABILITIES CAUSED BY UNEXPECTED FEEDBACK

Example	Consequence	Mitigation
Upgraded system had faster time response than original	Oscillating plant equipment	Use of claims, arguments and evidence (CAE) to justify upgrade. The justification of the upgrade should have had a precise claim about the time response and how it was changed. The assumption that faster has no impact should have been justified explicitly.
Designer underestimated time delay in digital equipment	Control system instabilities (a small time delay can cause a large phase difference in feedback loops)	Validation should address timing assumptions. For example, a CAE case would have timeliness as an explicit claim and the validity and apportionment of response times to comments should be justified, e.g. supported by a property decomposition CAE block.

III–4. GENERIC CHALLENGES: ASSESSMENT VULNERABILITIES

III–4.1. Issues

In this section the following generic vulnerabilities in dependability assessment are summarized:

— Formulation of claims;
— Precision or completeness of claims;
— Specific types of argument (e.g. composition of properties);
— Abstraction or mental model of the system (e.g. architecture).

III–4.2. Vulnerabilities

Tables III−5 through III−8 provide examples of specific vulnerabilities that can be used to challenge an assessment, but that can also serve as exemplars for a project to generate their own set. The vulnerabilities have been grouped, but the categories often overlap.

TABLE III–5. CLAIM DEFINITION: PRECISION

Category	Explanation	Examples	Mitigation
Claim not a proposition	Claims should be formulated as statements that can be true or false. The claim should be a succinct proposition but will often need additional narrative to explain and provide context.	"Test results" is not a claim, but "System passed all defined tests" is. "Hazard analysis" might refer to an activity or report, whereas the claim might be "Hazard analysis undertaken" or "All identified hazards mitigated".	Review of definitions. Ask if the claim can be true or false. If using graphical notation to support the case, the narrative might include more detailed claims. Nevertheless, the graphical node should still be a proposition.
Imprecise claims; concretion needed	Dependability properties can be very abstract and need concretion to become precise enough to argue convincingly about them.	'Deterministic' is a property that needs to be clarified in an assessment since it also has a variety of definitions in the guidelines and standards. The distinction between aspects of resilience and fault tolerance might need clarification to support the discussion of determinism.	Review definition, ask whether the claim is a proposition or not. Review argument and evidence to see what they actually demonstrate. Use concretion or substitution CAE blocks to develop more precise claims.
Overly precise claims; abstraction needed	The requirements may be too precise and contain elements of design or constraints from previous implementations.	The application logic of previous systems may be used as a requirement and contain time delays that are not explained or justified, but are an artefact of the implementation.	Review claim for level of detail and consider abstraction: is it in terms of plant measurements or signals? Top level claims could be validated against a plant model to establish completeness and the level of detail needed.

TABLE III–6. CLAIM DEFINITION: COMPLETENESS

Category	Explanation	Examples	Mitigation
Incomplete or implicit claims; environment not specified	Balance is needed in formulating a claim in terms of making issues explicit. If too much is included as assumptions, then the case can be overly complex; if too little, important details might be missed.	The claim should address a specific system, in a defined application and context. Excluding the environment might lead to changes in the environment being ignored, such as the threat levels.	Use CAE block application tables and prompts to assess whether environment, application and system are detailed sufficiently. For example, see if implicit aspects are changed or differ between claims, in which case these differences should be made explicit. If using a graphical notation to support the case, the narrative might include more detailed claims.
Incomplete or implicit claims; assumptions about usage; weak argument	Claims need to define the system configuration and usage environment precisely, or the arguments associated with them can be invalid or weak.	A particular version of a device has been evaluated, but the overall claim is about a different version. This needs to be made explicit in the case. Use is made of operating experience, but the details of operational profile and failure reporting are not included.	Use of CAE substitution blocks to carefully record the precise system and configuration changes in the case. Use of CAE block side conditions to explore assumptions and check validity with respect to the real world.
Incomplete or implicit claims; abstraction	Different states are used to categorize the behaviour of a system (operational, maintenance, failed, OK). The transition between states can be treated as an event that is not analysed further when in fact the transition should perhaps be viewed as an activity or process in its own right.	The transition time between operating modes of a plant can be significant with respect to the timescales of the I&C system. Refurbishment may involve states with more and different dependability claims than originally envisaged.	The first step is to identify when this style of argument is being used — typically when a CAE decomposition block of some sort is applicable. The CAE block application tables ask explicit questions about the validity of the decomposition in the real world. This should be recorded and reviewed by experts with domain knowledge. The explicit use of time bands can help identify these assumptions.

TABLE III–7. WEAK OR INCOMPLETE ARGUMENTS

Category	Explanation	Examples	Mitigation
Invalid argument; property does not distribute	An argument might be made that a property applies to each component and so applies to the composition of the components, i.e. the overall system.	An example would be an argument that conservative claims about reliability of the subsystems can be multiplied to provide conservative claims of system reliability.	Use of explicit CAE decomposition blocks requires the composition of the properties to be justified explicitly. Architecture decomposition of properties needs particular attention when reviewing cases. Some aspects are covered in application tables.
Logical fallacy; weak argument	Some aspects of logic might be clear from the authors' point of view, but logically incorrect.	In narrative English, logical terms are sometimes used in a way that is formally incorrect, and this can lead to ambiguous specification e.g. confusion of 'and' and 'or'. Arguing from the specific to the general is classed as a logical fallacy unless supported, e.g. arguing from one device to all devices of that type.	Review of claim formulation to identify logic. Translation of narrative into formulas. Use of tools to animate logic to see if it matches the intention (part of validation). Arguments in CAE block decompositions are always conjunctions. Use of CAE substitution block to capture this type of argument, so the move from 'the assessed device has this property' to 'all delivered devices have this property' is made clear.
Not feasible argument/claim too strong	Some claims can be seen as not being feasible.	Making a claim with very high confidence of a numerical reliability target based on statistical testing. Ignoring assumption doubt that comes from epistemic doubt in operational profile and the oracle.	Reduce the strength of the claim (e.g. changing the numerical target) or the required confidence. Make an explicit case that includes the sources of doubt and consider the trade offs (e.g. whether the test bed can have less fidelity but a higher number of test cases, whether the oracle can be developed to higher standards so it can be trusted more, whether research in 'fair testing' can be used to reduce the sensitivity of the claim).

TABLE III–8. EVIDENCE INTEGRATION

Category	Explanation	Examples
Incorrect evidence integration; incomplete argument; claim too abstract	Abstraction is essential, but by its nature it excludes things. Sometimes this exclusion leads to weaknesses in the case. This could be lead to defects in the evidence integration or in a claim.	A claim might be that a property has been proved, but in fact a more precise claim is that the tool used indicates 'proved'. A claim might be that a system 'does not fail in 100 demands', whereas in fact a more precise claim is that the test bed results shows it has passed 100 tests. Both these examples would need expanding to cover the fallibility of the experimental set-up (tools, oracles, specifications, etc.) and the sentencing process.
Invalid evidence; invalid abstraction; incomplete argument	Abstraction is essential but sometimes the abstract representation is not accurate enough for the argument being made.	A claim might be based on an architectural diagram that is not an accurate reflection of the actual connectivity. It might contain a suggested structure, with the actual code structure being more connected. This can impact claims of CCF and failure behaviour.
Irrelevant or unclear evidence; weak argument	Projects may provide detailed information, but it may not be clear whether it is really evidence or just extraneous information.	Information on detailed project plans and processes may be provided without rationale.

Annex IV

EXAMPLES OF THE APPLICATION OF ELEMENTS OF THE ASSESSMENT FRAMEWORK

This annex provides examples of the application of elements of the assessment framework. The first two examples illustrate how the claims, arguments and evidence (CAE) approach can be used to express the argument for claims and subclaims raised by the assessment strategy. These are only simplified examples, not generic solutions applicable in all situations.

IV–1. CLAIM 1: THE I&C SYSTEM REQUIREMENTS ARE ADEQUATE

This example illustrates how a strategy for assessing a very generic claim (the instrumentation and control (I&C) system requirements are adequate) can be expressed in a CAE format, so that it can be used as a tool for early communication between stakeholders of the dependability assessment. It provides only the top levels of the strategy, which in practice would need to specify further levels of refinement. Even so, one can see that the effort and cost of developing and formulating such a strategy are limited compared with the complete performance of the assessment.

The plant specific aspects of I&C system requirements are developed based on the results of upstream plant level activities that involve many different disciplines and stakeholders. Often, the I&C requirements specification team will need to engage them when additional information is needed, when these upstream results place unwarranted constraints on the I&C system (e.g. when they state unnecessarily detailed solutions rather than an original need) or when feasibility is questionable.

Other sources of requirements, such as codes of good practice or regulations, are more generic but may be expressed in general terms that need more precise and practical interpretation (concretion), and sometimes constructive discussions and agreement with assessors.

In addition to the need to provide an adequate answer to these input documents, the I&C system requirements specification should possess a number of properties intrinsic to the fact that it is a requirements specification.

Thus, Claim 1 can be made more firm using a concretion block tying together the following subclaims:

— 1.A: The set of input documents used as the basis for the I&C system requirements specification is adequate.
— 1.B: Considering the role assigned to it by the overall plant design, all the statements in these input documents that place requirements on the I&C system are identified.
— 1.C: The I&C system requirements specification adequately addresses each of these statements.
— 1.D: The I&C system requirements specification adequately covers all necessary topics.
— 1.E: The I&C system requirements specification document has key ‘intrinsic properties’.

IV–1.1. Subclaim 1.A: The set of input documents used as the basis for the I&C system requirements specification is adequate

The justification for this subclaim is based on further subclaims tied together by a concretion block:

— 1.A.1: The set of input documents includes a predefined list of documents or document types (e.g. plant level analyses, plant overall design, standards, regulations, codes of good practice).
— 1.A.2: A review of the set membership was performed by persons who collectively have the necessary competence and information.
— 1.A.3: Adequate configuration management ensures that the relevant input documents are identified and the right versions used.
— 1.A.4: These documents adequately consider the I&C system itself, in particular its interfaces with the rest of the plant, its role in the plant, how it will be operated and maintained, and how its postulated failures and vulnerabilities are taken into account.

Assumption: the information provided in these input documents is correct, appropriate and sufficient for what may concern the I&C system. Of course, this claim should at some point be justified, but it is beyond the scope of I&C and of this publication.

Evidence for subclaim 1.A.4 is based on a variety of approaches, such as modelling and simulation, failure analyses, etc. This is usually an iterative process, as particular I&C system requirements might have consequences at plant level, which in turn might need to be taken into consideration by the I&C system.

IV–1.2. Subclaim 1.B: Considering the role assigned to it by the overall plant design, all the statements in these input documents that place requirements on the I&C system are identified

The justification for this subclaim is based on further subclaims tied together by a concretion block:

— 1.B.1: This identification was performed by a team with the right set of competences and information.
— 1.B.2: The identification was performed applying appropriate and agreed methods and tools.
— 1.B.3: A review was performed by persons who collectively had the necessary competence and information and who did not participate in the identification.

IV–1.3. Subclaim 1.C: The I&C system requirements specification adequately addresses each of these statements

The justification for this subclaim is based on further subclaims tied together by a concretion block:

— 1.C.1: The I&C system requirements specification addresses each of these statements (traceability could be one possible source of evidence).
— 1.C.2: The requirements specification was developed by a team with the right set of competences and information.
— 1.C.3: The requirements specification has been developed applying appropriate and agreed methods and tools (e.g. use of modelling and simulation when applicable).
— 1.C.4: A review was performed by persons who collectively had the necessary competence and information and who did not participate in the requirements specification.

IV–1.4. Subclaim 1.D: The I&C system requirements specification adequately covers all necessary topics

The justification for this subclaim is based on further subclaims tied together by a concretion block:

— 1.D.1: The I&C system requirements specification addresses each of the topics of an approved predefined list. It includes topics such as interfaces with other systems and with personnel, operational modes, functions in each mode, response times, accuracy, self-monitoring, fault tolerance, failure modes to be avoided, maximum failure rates, provisions for future changes, etc. The list of topics should address, but should not be restricted to, the system features listed in Section 2.8.
— 1.D.2: A review was performed by persons who collectively had the necessary competence and information and who did not participate in the requirements specification.

IV–1.5. Subclaim 1.E: The I&C system requirements specification document has key ‘intrinsic properties’

An ‘intrinsic property’ is a property that can be judged based on the analysis of the specification alone. The justification for this subclaim is based on further subclaims tied together by a concretion block:

— 1.E.1: The I&C system requirements specification is unambiguous.
— 1.E.2: The I&C system requirements specification is clear to the persons concerned (limitation of complexity, terms are defined).

— 1.E.3: The I&C system requirements specification does not have ‘intrinsic holes’ (i.e. missing elements that could be identified by the analysis of the specification alone; a typical example would be an incomplete truth table).
— 1.E.4: The I&C system requirements specification is consistent (no contradiction).
— 1.E.5: The I&C system requirements specification has been approved by the authorized decision makers.

IV–1.6. Comments

The first levels of the assessment strategy for Claim 1 are mostly based on concretion blocks. The objective is to identify subclaims that have a clear meaning and for which appropriate sources of evidence can be identified. In this example, many would be provided by quality assurance plans or records (competences, reviews, methods and tools, etc.). Others would be provided by application of specific techniques (traceability, modelling and simulation, predefined templates, etc.).

IV–2. CLAIM 2: THE I&C SYSTEM CORRECTLY IMPLEMENTS ITS SPECIFIED REQUIREMENTS

This example illustrates a possible approach to justify Claim 2 (that the I&C system correctly implements its specified requirements), in a CAE format. Here again, it should be viewed only as a simplified example.

In a very abstract manner, the safety I&C system of this example can be viewed as a distributed system composed of multiple interconnected digital controllers receiving inputs from analogue sensors. The argument can be organized into the following steps:

— Step 1 substitutes the operational system with its design and possibly with system ‘samples’ (for test purposes). This substitution can be made under the justification that the design and the samples are adequate representations of the operational system.
— Step 2 decomposes the system into functional units, as specified by the system architecture. The decomposition block claims that each unit complies with its requirements. It also claims that the combination of compliant units (as specified by the system architecture) complies with the system requirements.
— Step 3 decomposes each digital unit into a combination of hardware and software (in a general sense) as specified by the unit design. As in the previous step, each decomposition block claims first that the components (hardware and software) separately meet their requirements, and then that the combination meets the unit’s requirements. In this limited example, hardware subsystems and the hardware part of digital units are assumed to be fully compliant with their own requirements specifications.
— Step 4 substitutes the software of each digital unit with its design and/or source code. Here again, the substitution can be made under the justification that the design and the source code are adequate representations of the executable software installed in the operational system.
— Step 5 further decomposes the software of each digital unit into an application specific part and a platform specific part.
— The next steps regarding the platform specific part include decomposition into smaller software components, claims on control flow and data flow, functional claims on particular software components (e.g. on elementary library functions) and innocuousness claims on others (e.g. a component performing auto tests).

Annex V

EXAMPLES OF TECHNIQUES FOR GENERATING EVIDENCE

Table V–1 summarizes techniques of generating evidence for the dependability properties discussed in Section 4 of the main text.

TABLE V–1. DEPENDABILITY CHARACTERISTICS AND EXAMPLES OF TECHNIQUES FOR GENERATING EVIDENCE

Characteristic	Definition	Examples of evidence generation techniques
Adequacy of instrumentation and control system requirements	n.a.[a]	Audits Inspections and reviews Static analysis Formal modelling and verification Failure mode and effects analysis Fault tree analysis Probabilistic reliability analysis Hazard and operability analysis System-theoretic process analysis Analysis of operating experience Modelling and simulation
Functionality	Correct implementation of the required functions	Random testing Statistical (operational profile) testing Functional testing Model based testing Negative testing Random model based testing Unit testing Interface testing Development metrics Static analysis Formal specification, modelling and verification Modelling and simulation
Time response	The time allowed for the software to respond to the inputs or to periodic events	Performance testing Design inspection evidence for invariant execution time Worst case execution time analysis Modelling and simulation
Accuracy	The degree of closeness of the measured or calculated value to its true value	Accuracy tests, as part of functional testing Numerical precision analysis Measurements
Reliability	The probability that the software will perform to a specified requirement for a given period of time under specified conditions	Statistical (operation profile) testing Development measures Coding standards compliance checking Static analysis Formal modelling and analysis Fault tree analysis Probabilistic reliability analysis Analysis of operating experience Modelling and simulation Fault injection

TABLE V–1. DEPENDABILITY CHARACTERISTICS AND EXAMPLES OF TECHNIQUES FOR GENERATING EVIDENCE (cont.)

Characteristic	Definition	Examples of evidence generation techniques
Maintainability	Ability to undergo repairs and modifications	Design inspection Code inspection Code complexity metrics Coding standards compliance checking Hazard and operability analysis System-theoretic process analysis Modelling and simulation
Robustness	Response to external events: spurious (unexpected) inputs, hardware faults and power supply interruptions	Stress testing Negative testing Overload analysis Design inspection and code inspection of features to handle overloads, out of range values, invalid input and output Hazard and operability analysis System-theoretic process analysis
Failure integrity	Probability that an internal failure can be detected externally	Runtime integrity checking Fault injection Failure integrity analysis Modelling and simulation
Operability	Avoidance of human error in maintenance and calibration	Usability testing Hazard and operability analysis System-theoretic process analysis Modelling and simulation
Security	Prevention of loss of component confidentiality, availability and integrity due to malicious acts	Audits Inspections and reviews Development metrics Coding standards compliance checking Static analysis Formal modelling and analysis System-theoretic process analysis Analysis of operating experience Modelling and simulation Penetration testing Vulnerability testing

[a] n.a.: not applicable.

ABBREVIATIONS

AP	assessment principle
CAE	claims, arguments and evidence
Cat. A	Category A
CCF	common cause failure
COMPSIS	Computer-based Systems Important to Safety (project)
COTS	commercial off the shelf
CPU	central processing unit
DFMEA	design FMEA
EN	European norm
EPRI	Electric Power Research Institute
FMEA	failure mode and effects analysis
FPGA	field programmable gate array
FTA	fault tree analysis
HAZOP	hazard and operability analysis
HDL	hardware description language
I&C	instrumentation and control
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
MISRA	Motor Industry Software Reliability Association
NPP	nuclear power plant
O&M	operation and maintenance
OECD/NEA	Nuclear Energy Agency of the Organisation for Economic Co-operation and Development
PFD	probability of failure on demand
PRA	probabilistic risk assessment
SAE	Society of Automotive Engineers
SIL	safety integrity level
SOUP	software of uncertain pedigree
STPA	system-theoretic process analysis

CONTRIBUTORS TO DRAFTING AND REVIEW

Arndt, S.	Nuclear Regulatory Commission, United States of America
Bartha, T.	Institute for Computer Science and Control, Hungary
Bloomfield, R.	Adelard LLP and City University London, United Kingdom
Eiler, J.	International Atomic Energy Agency
Kwon, K.C.	Korea Atomic Energy Research Institute, Republic of Korea
Lawford, M.	McMaster University, Canada
Lindner, A.	TÜV Rheinland ISTec, Germany
Nguyen, T.	Électricité de France — R&D STEP, France
Seaman, S.	Westinghouse Electric Company, United States of America
Shumaker, B.	Analysis and Measurement Services, United States of America
Smidts, C.	Ohio State University, United States of America
Tate, R.	Health and Safety Executive, Office for Nuclear Regulation, United Kingdom
Wood, R.T.	Oak Ridge National Laboratory, United States of America
Zahid, M.	International Atomic Energy Agency

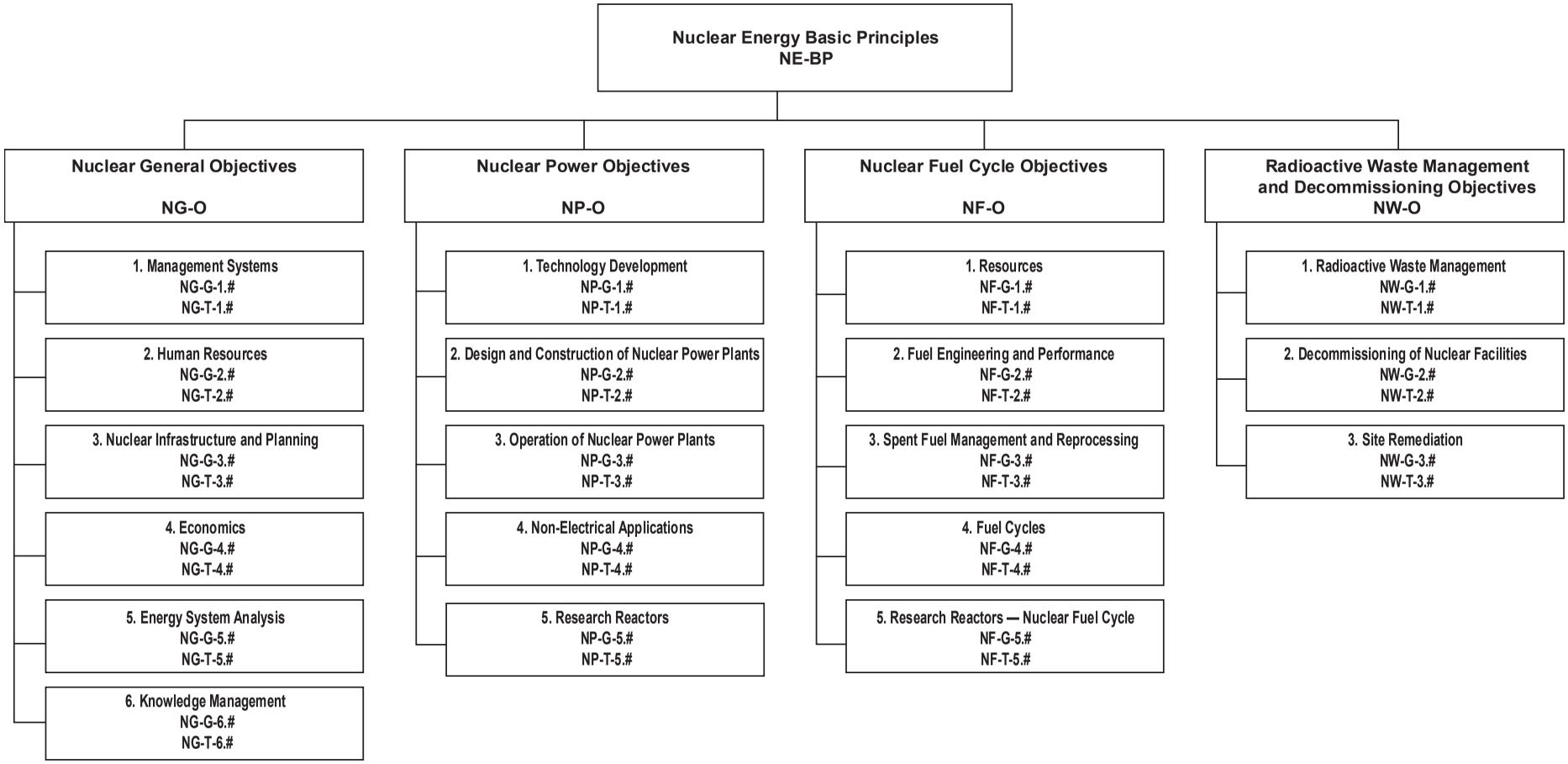
Technical Meeting

Daejeon, Republic of Korea: 23–26 September 2014

Consultants Meetings

Vienna, Austria: 12–15 May 2014; 23–27 March 2015; 5–9 October 2015

Structure of the IAEA Nuclear Energy Series



Key

BP: Basic Principles
O: Objectives
G: Guides
T: Technical Reports
Nos 1-6: Topic designations
#: Guide or Report number (1, 2, 3, 4, etc.)

Examples

NG-G-3.1: Nuclear General (**NG**), Guide, Nuclear Infrastructure and Planning (topic **3**), **#1**
NP-T-5.4: Nuclear Power (**NP**), Report (**T**), Research Reactors (topic **5**), **#4**
NF-T-3.6: Nuclear Fuel (**NF**), Report (**T**), Spent Fuel Management and Reprocessing (topic **3**), **#6**
NW-G-1.1: Radioactive Waste Management and Decommissioning (**NW**), Guide, Radioactive Waste (topic **1**), **#1**

IAEA
International Atomic Energy Agency

No. 25

ORDERING LOCALLY

In the following countries, IAEA priced publications may be purchased from the sources listed below or from major local booksellers.

Orders for unpriced publications should be made directly to the IAEA. The contact details are given at the end of this list.

CANADA

Renouf Publishing Co. Ltd

22-1010 Polytek Street, Ottawa, ON K1J 9J1, CANADA
Telephone: +1 613 745 2665 • Fax: +1 643 745 7660
Email: order@renoufbooks.com • Web site: www.renoufbooks.com

Bernan / Rowman & Littlefield

15200 NBN Way, Blue Ridge Summit, PA 17214, USA
Tel: +1 800 462 6420 • Fax: +1 800 338 4550
Email: orders@rowman.com Web site: www.rowman.com/bernan

CZECH REPUBLIC

Suweco CZ, s.r.o.

Sestupná 153/11, 162 00 Prague 6, CZECH REPUBLIC
Telephone: +420 242 459 205 • Fax: +420 284 821 646
Email: nakup@suweco.cz • Web site: www.suweco.cz

FRANCE

Form-Edit

5 rue Janssen, PO Box 25, 75921 Paris CEDEX, FRANCE
Telephone: +33 1 42 01 49 49 • Fax: +33 1 42 01 90 90
Email: formedit@formedit.fr • Web site: www.form-edit.com

GERMANY

Goethe Buchhandlung Teubig GmbH

Schweitzer Fachinformationen
Willstätterstrasse 15, 40549 Düsseldorf, GERMANY
Telephone: +49 (0) 211 49 874 015 • Fax: +49 (0) 211 49 874 28
Email: kundenbetreuung.goethe@schweitzer-online.de • Web site: www.goethebuch.de

INDIA

Allied Publishers

1st Floor, Dubash House, 15, J.N. Heredi Marg, Ballard Estate, Mumbai 400001, INDIA
Telephone: +91 22 4212 6930/31/69 • Fax: +91 22 2261 7928
Email: alliedpl@vsnl.com • Web site: www.alliedpublishers.com

Bookwell

3/79 Nirankari, Delhi 110009, INDIA
Telephone: +91 11 2760 1283/4536
Email: bkwell@nde.vsnl.net.in • Web site: www.bookwellindia.com

ITALY

Libreria Scientifica "AEIOU"

Via Vincenzo Maria Coronelli 6, 20146 Milan, ITALY
Telephone: +39 02 48 95 45 52 • Fax: +39 02 48 95 45 48
Email: info@libreriaaeiou.eu • Web site: www.libreriaaeiou.eu

JAPAN

Maruzen-Yushodo Co., Ltd

10-10 Yotsuyasakamachi, Shinjuku-ku, Tokyo 160-0002, JAPAN
Telephone: +81 3 4335 9312 • Fax: +81 3 4335 9364
Email: bookimport@maruzen.co.jp • Web site: www.maruzen.co.jp

RUSSIAN FEDERATION

Scientific and Engineering Centre for Nuclear and Radiation Safety

107140, Moscow, Malaya Krasnoselskaya st. 2/8, bld. 5, RUSSIAN FEDERATION
Telephone: +7 499 264 00 03 • Fax: +7 499 264 28 59
Email: secnrs@secnrs.ru • Web site: www.secnrs.ru

UNITED STATES OF AMERICA

Bernan / Rowman & Littlefield

15200 NBN Way, Blue Ridge Summit, PA 17214, USA
Tel: +1 800 462 6420 • Fax: +1 800 338 4550
Email: orders@rowman.com • Web site: www.rowman.com/bernan

Renouf Publishing Co. Ltd

812 Proctor Avenue, Ogdensburg, NY 13669-2205, USA
Telephone: +1 888 551 7470 • Fax: +1 888 551 7471
Email: orders@renoufbooks.com • Web site: www.renoufbooks.com

Orders for both priced and unpriced publications may be addressed directly to:

Marketing and Sales Unit
International Atomic Energy Agency
Vienna International Centre, PO Box 100, 1400 Vienna, Austria
Telephone: +43 1 2600 22529 or 22530 • Fax: +43 1 2600 29302 or +43 1 26007 22529
Email: sales.publications@iaea.org • Web site: www.iaea.org/books

18-03411

INTERNATIONAL ATOMIC ENERGY AGENCY
VIENNA
ISBN 978–92–0–101218–0
ISSN 1995–7807