Research Summary: Software Assurance for CBSIS

May 2026

Document Number: eno-041

Revision Number: 0

ennuvo

Revision Log

Revision	Description of Changes
Rev 0	Initial issue for publication as part of UK SMR-300 Innovation project.

Executive summary

The two-legged approach to the substantiation of computer-based systems important to safety (CBSIS), preferred by the UK nuclear regulator, has led to additional effort needed to justify the use of software, particularly through the application of independent confidence building measures (ICBMs). Independent scrutiny of CBSIS forms a key foundation of software assurance that is recognised globally in the nuclear sector and other high hazard industries. However, the application of ICBMs late in the CBSIS development lifecycle, as is common with the two legged approach, adds significant additional effort to assuring software-based systems that is particular to the UK nuclear sector even when through-life evidence is available. ~~Consequently, the two-legged approach often defaults to the application of retrospective ICBMs. The economics of many modern nuclear reactor designs (e.g., small and advanced modular reactors), which aim to address global warming and domestic energy security, rely on a globally acceptable design.~~ The economic viability of SMRs is challenged by additional ICBM requirements whose cost does not scale with SMR reactor power, and the late application of UK specific additional testing and analysis is considered to be inconsistent with systems engineering good practice.

This report provides a summary of a critical analysis of literature (including international standards) and engagement with a number of leading experts in software assurance. It considers a practical framework (VARIA) for the assurance of software used in nuclear CBSIS. It builds on good practice used in the nuclear sector and in other industries for software assurance and the critical role independent scrutiny plays. It identifies where assurance activities can be conducted earlier in the development lifecycle to reduce project risk.

The VARIA framework highlights ways to reduce testing and assurance costs within the traditional two-legged approach, while still showing that the risks of using CBSIS are reduced to as low as reasonably practicable (ALARP), without compromising safety or security standards. Like other assurance frameworks VARIA has the same limitations where legacy software development is not always available. However, VARIA does provide a proactive mechanism to capture evidence as it is produced, identify evidentiary gaps early or allow for the implementation of a system architecture to manage the resulting risk. The research questions several UK regulatory expectations and, by examining comparable industries and modern software development practices, seeks to show that current approaches warrant reconsideration e.g. expectations regarding statistical testing and an emphasis to perform final code static analysis.

Table of Contents

List of Figures

Definitions and Abbreviations

ALARP	As low as reasonably practicable
C&I	Control and instrumentation
CbyC	Correct by construction
CBSIS	Computer based systems important to safety
CINIF	C&I Nuclear Industry Forum
CORDEL	Cooperation in Reactor Design Evaluation and Licensing
COTS	Commercial off the shelf
EPR	European Pressurised Reactor
FOAK	First of a kind
FSA	Functional safety assessment
HSE	Health and Safety Executive
IAEA	International Atomic Energy Agency
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronic Engineers
MBSE	Model-based systems engineering
ONR	Office for Nuclear Regulation
PFD	Probability of failure on demand
RPS	Reactor protection system
SIL	Safety integrity level
SMR	Small modular reactor
SpecTRM	Specification tools and requirements management
T&M	Techniques and measures
TAG	Technical assessment guide
USNRC	United States Nuclear Regulatory Commission
V&V	Verification and validation
VARIA	V-model, architecture, requirements, independence, assurance.

1 Introduction

1.1 Background

Software based safety systems[1] can be considered complex due to factors, including:

- The number of components (e.g. lines of code, modules etc).
- The execution of multiple tasks at any one time and the resulting importance of synchronising tasks in real time systems.
- The number and complexity of interfaces between system components.
- The discrete, discontinuous nature of software makes it extremely difficult to exhaustively test.

Due to software's inherent complexity and systematic, rather than random, failure modes, traditional tests and measurements are an insufficient basis for determining system integrity. Instead, effective software-based system assurance needs an approach that allows the system to be assessed in a number of diverse ways. This is often an iterative process where system integrity can be inferred from the assessment of the quality and robustness of the software production process and the results of verification and validation (V&V) activities.

The context for this research can be summarised as follows:

- The viability of modern nuclear power plants, such as small modular reactors (SMR), is predicated on deployment of a design in a global marketplace. This includes the need to demonstrate that systems (including CBSIS) meet regulatory expectations in the country of their deployment.
- There are many similarities in requirements within nuclear standards at an international level, for example (IAEA, 2016) (IEC, 2011) and (IEEE, 2018). However, subtle differences exist between practices used in the nuclear sector in different countries. Whilst efforts have been made to harmonise the practical interpretation of principles underpinning international standards, for example (WNN, 2021) (MPED, 2024), it remains the case that national differences exist in the practical implementation of CBSIS.
- In the United Kingdom (UK) the application of a two-legged approach (ONR, 2014) (ONR, 2023) and precedent set during previous regulatory assessments (HSE, 2008) (ONR, 2017) (WNA, 2021) has led to the need to undertake additional activities at the back-end of the development process to gain regulatory acceptance.
- V&V expectations are often derived from international standards. Throughout the V&V specification and application it is important to maintain a clear understanding of the role any technique or measure (T&M) plays in mitigating risk. International standards are developed in the absence of an application and it can be difficult to justify not implementing a normative or informative requirement from a standard although it may add little value. This can lead to unnecessary time and cost. There is a risk that

[1] Hereinafter referred to as computer-based systems important to safety (CBSIS). Note CBSIS may also include hardware description languages (HDLs). Aspects of VARIA may also be useful for the assurance of software aspects associated with these devises.

standards are viewed as prescriptive requirements. Licensees often need to justify why non-applicable standards and / or individual clauses have not been applied. Whilst challenge from independent assurance functions and regulators create important tension for safety justification, when they judge a justification to be insufficient, they should provide clear statements of deficiency to highlight the specific risks they believe are posed to the system.

- A significant barrier to the deployment of SMRs is the investment needed to develop the reactor and its first deployment (first of a kind (FOAK)). CBSIS software substantiation can be a significant percentage of the reactor development cost and therefore a barrier to deployment. Especially if expectations at a regional level vary from that of the country of origin. In addition, for many other nuclear safety components, costs scale proportionate to the size (e.g. MWth) of the plant. This is not the case for CBSIS software substantiation.

Ultimately these contextual factors present a challenge to a fleet based SMR deployment and their contribution to net zero emissions targets and sovereign energy security.

This context has resulted in a comprehensive set of research studies and the development of the VARIA framework described in this report.

1.2 Scope

This report presents a critical analysis of relevant literature, including international standards, and insights from leading experts in software assurance. It proposes a coherent approach to support the design and substantiation of CBSIS. The goal being to support the achievement and maintenance of CBSIS safety performance levels that are likely to be acceptable across multiple regulatory jurisdictions.

Many modern computer-based safety systems incorporate platforms and commercial off-the-shelf (COTS) devices already deployed in nuclear facilities or in other high-hazard sectors. Where applicable, leveraging established good practices from these domains offers clear benefits for software assurance in the UK context. This research draws on good practices from the nuclear industry in other regulatory jurisdictions and from other high hazard sectors. It explores how such practices can be applied to demonstrate that the risks associated with the use of CBSIS have been reduced to as low as reasonably practicable. Furthermore, the report recognises that similarities in international standards can enable regulatory assessment in one country to be broadly accepted in another.

Section 2 of this report presents the VARIA framework which is the result of a series of research studies commissioned by Holtec Britain Ltd to determine if a practical framework for the assurance of software used in CBSIS could be derived. These studies investigated the following topics, recognised as critical to nuclear safety, with the aim of providing actionable guidance for control and instrumentation (C&I) engineers, subject matter experts, and related disciplines.

- Nuclear safety arrangements in a number of countries.
- Arrangements in other industry sectors (e.g. aviation, defence, etc).
- Benefits of system structure and architecture.

- Software development and implementation processes.
- Good practice in independent assessment.
- Efficiencies that can be gained from vendor production excellence.
- Suitability of publicly available standards and guidance.

In the UK, the nuclear safety regulator, the Office for Nuclear Regulation (ONR), assesses the suitability of software substantiation using a two-legged approach (ONR, 2014) (ONR, 2023). **Section 3** compares the findings of this research with ONR's methodology, highlighting areas of alignment and potential enhancement. The aim is to identify opportunities to reduce some of the costs related to testing and assurance in the traditional two-legged approach, while maintaining the required safety and security standards. This includes examining modern software production methods that integrate assurance measures within the development process itself, rather than relying predominantly on back-end testing – a characteristic of the two-legged approach.

Section 4 outlines emerging trends in software-based safety systems, as identified by experts monitoring developments in critical national infrastructure. These trends may influence future approaches to software assurance, including for CBSIS.

1.3 Context setting for this research

1.3.1 Predeveloped devices and platforms

The design, development, construction and use of CBSIS, often includes predeveloped platforms and COTS smart instruments and devices (e.g. systems that contain software adapted for use). Therefore, an essential element in determining the software assurance activities needs to involve the supply chain. The use of these devices, which can include beneficial features such as self-monitoring and trending of measured variables, needs to be justified to demonstrate alignment with expectations in the UK nuclear sector (including the use of standards). In addition, the use of programmable digital devices has increased due to the reduced availability of analogue equivalents and experienced people to support them through life. This has resulted in an increased need to assure software used in CBSIS. However, the benefits of CBSIS are often lost due to the increased cost of qualification.

The UK nuclear sector has developed tools to aid the deployment of predeveloped devices for use in CBSIS including:

- The EMPHASIS toolset is an assessment framework for the substantiation of COTS smart instruments and devices, developed by the C&I Nuclear Industry Forum (CINIF). EMPHASIS is used as a means of satisfying the production excellence aspects of the two-legged approach set out in regulatory guidance (ONR, 2023) although access to this toolset is restricted to CINIF member organisations.
- The UK's preferred claims, argument and evidence approach to safety case structure led to the development of the Cogs structure for the collation of evidence on the behaviour of CBSIS and aims to complement the EMPHASIS approach (Guerra S, 2015). The main aim is to show that "the described behaviour and functionality of the product is sufficient to understand its intended behaviour and required properties, and the product behaves according to this description throughout its intended lifetime". Cogs

can also be considered without EMPHASIS in instances where there is no existing assessment framework.

For CBSIS platforms, the Regulator Task Force on Safety Critical Software (ONR, 2024) provided guidance on software assurance evaluation including predeveloped software platform qualification. A comparative review of VARIA with the Common Position has been undertaken and related aspects are discussed in detail with reference to TAG46 (ONR, 2023) rather than the Common Position. The guidance acknowledges that the platform and the application software can be treated separately in many areas with some common aspects (e.g. communication). The approach to substantiation of the platform software, application software and communications is likely to differ. For example, it can be difficult to gain access to code and data from activities early in the development lifecycle of predeveloped components making it necessary in some circumstances to work with platform vendors to demonstrate software assurance good practice over the whole lifecycle.

1.3.2 Purpose of software assurance

In many industries and regulatory jurisdictions, the basis for assurance of software used in CBSIS draws upon a wide range of evidence produced throughout its development lifecycle.

Software assurance is an approach where credit is taken for activities conducted throughout the software development process, including the production and documentation of clear requirements, collection of evidence that tracks the demonstration that requirements have been met and the adequacy of quality assurance arrangements.

To support this approach, the starting point for this research has been to define the high-level purpose that describes the intent of software assurance and safety performance targets for CBSIS. The aim of each software assurance arrangement and activity needs to be clear (e.g. qualitative confidence, validation of requirements, satisfy regulatory expectations etc.) so the overall aim of reducing the risk of systematic error in line with the required integrity of the system can be shown. The following four software assurance purposes used in this research are derived from a number of sources that have been identified and reviewed during this work (J McDermid D. P., 2001), (McDermid, 2012), (Holloway, 2019), (J McDermid T. K., 2006).

1. Software safety functional and non-functional requirements are robust and include all failure modes (e.g. software failure creating a hazard, software misleading operators and mismatch between the software and the environment).
2. Independent software assurance arrangements and activities demonstrate safety, taking account of the role and application of the software-based safety system.
3. Software safety requirements are demonstrably met across the system lifecycle in a way that the evidence links to the performance of the system's safety in terms of:
    - The intended behaviour is correct and complete with respect to the specified behaviour.
    - The implementation is correct with respect to its defined intended behaviour, under foreseeable operating conditions.
    - Any part of the implementation that is not required by the defined intended behaviour has no unacceptable impact.
4. Uncertainty and known limitations are reduced to an acceptable level across the system lifecycle and in a manner that does not compromise the safety goal. Including:

- Assumptions and constraints (e.g. signal values, timing, and combinations; hardware constraints).
- Influence from other integrated components / systems and ageing and obsolescence.
- Lack of completeness and consistency of requirements.
- Incomplete coverage of V&V (including combinations of defects).
- Other potential shortfalls (humans, tools, information).

A critical enabler for software assurance is the collaboration between the device or system vendor and the end user to enable adoption of the embedded software assurance techniques proposed within the VARIA framework. Initial supplier engagement to determine the willingness to engage collaboratively is required to mitigate any collaboration risk and help inform the most appropriate techniques to adopt across the VARIA framework.

2 VARIA – An assurance framework

This section provides a summary of a critical analysis of literature (including international standards), investigation into various approaches used in the development of software intended for use in safety systems, and insights resulting from engagement with a number of leading experts in software assurance. The work has resulted in the development of the VARIA framework for software assurance.

The VARIA framework focuses specifically on software assurance and is intended to be applicable to COTS and bespoke systems and devices of all classes as each element can be scaled appropriately[2].

2.1 Summary of the framework

VARIA is a mnemonic that has been derived to assist C&I safety system engineers in their development of a holistic, multifaceted, approach to software substantiation. Etymologically the word *varia* derive from Latin (a derivative of variō) which in one sense means various, but more specifically for this application it means diversified.

The mnemonic emphasises the importance of:

- **V**-model (software lifecycle) – Like many software assurance frameworks VARIA recognises the importance of a flexible approach across a lifecycle model with the emphasis placed on, as necessary, iterative activities and whole lifecycle management.
- **A**rchitecture – VARIA recognises the importance of establishing a system and software architecture that reduces complexity and supports diversity and defence in depth early in the lifecycle.
- **R**equirements – Research supports the implementation of a systems engineering approach that stresses the importance of establishing a strong requirements basis for software-based safety systems that should be linked to provide clarity and traceability.

[2] Note: Emphasis and Cogs typically have a focus on lower classification devices, i.e. UK Class 2 and 3, and were designed for assessment and substantiation of COTS devices.

- **I**ndependence – Safety system development needs robust independent challenge provided by competent persons knowledgeable in software assurance and the intended application throughout the software lifecycle.
- **A**ssurance – An integrated, risk-based approach to software assurance including quality assurance arrangements and V&V activities applied across the whole lifecycle aimed at demonstrating the software requirements have been met.

The framework gives the approach to select the most appropriate techniques and measures for assurance and is considered to be applicable to both platform and application software layers. The framework does not include requirements for overall system substantiation such as hardware reliability, equipment qualification and cyber security which are outside the scope of this research.

2.2 V-model or lifecycle approach

The V-model or lifecycle approach comes in many forms. It provides a systematic and structured representation of the software development process. It is based on the idea of a "V" shape, with the two legs of the "V" representing the progression of the software development process from requirements gathering, optioneering, specification development and analysis to design and system integration. It can, in principle, be expanded to include through-life management and maintenance.

Figure 1 depicts a V-model[3] software development lifecycle which has been developed from a combination of elements from standards reviewed during this work (IEC, 2010), (IEC, 2018), (IEC, 2006), (IEEE, 2016), (IEEE, 2016) and (NASA, 2022). This lifecycle is based on the benefits of a systematic modular approach, adherence to design principles, minimising hazards through design review, assessing design optimisation and optioneering, and the verification of requirements through software prototyping, analysis and testing. Although activities on the left-hand side of the V-Diagram are often iterative to optimise software performance, activity on the right-hand side should be staged such that the next phase should only start after completion of the previous phase. To be effective, the V-model relies on configuration control to maintain traceability and demonstrate suitable verification of the final software.

A structured development lifecycle approach, as per the requirements of IEC62138 or IEC60880, such as the V-model, should be applied to achieve the following important aspects of CBSIS software development:

- Early defect identification by incorporating verification tasks into every stage of the development process. This lowers the cost and effort needed to remedy problems later in the development lifecycle. The verification activities between the development stages includes assessments of requirement coverage, system decomposition and system / component development and optimisation. The verification activities across the lifecycle relate to testing and traceability to requirements.

[3] The V-model or similar lifecycle approaches is a normative requirement of a number of international standards (IEC, 2010), (IEC, 2018) and (IEC, 2006) that address software developed for use in CBSIS.

- Provide and maintain a structured approach with distinct relationships between phases of software development and V&V, including testing, by ensuring that development processes and assurance activities are clearly mapped out.
- Create structured traceability between the requirements (including requirements that emerge during development) and how these are decomposed at each development stage as well as their related test cases (including regression testing needed for the assurance of emerging requirements).
- Apply robust independent assurance across the CBSIS lifecycle and avoid the reliance on assurance activities undertaken at a late stage in the development lifecycle.
- Promote cooperation between the assurance and development teams. Through this collaboration, project requirements, design choices, and assurance activities are better understood. This improves the effectiveness and efficiency of the development process.
- A methodical approach to manage complexity through the focus on assurance activities (including independent assurance) that can reduce risks related to system development, integration and interface problems for projects with a high level of complexity and interdependency among system components.

This approach and the considerations listed align with Purpose 3 presented in section 1.3.2: '*Software safety requirements are demonstrably met across the system lifecycle*.'

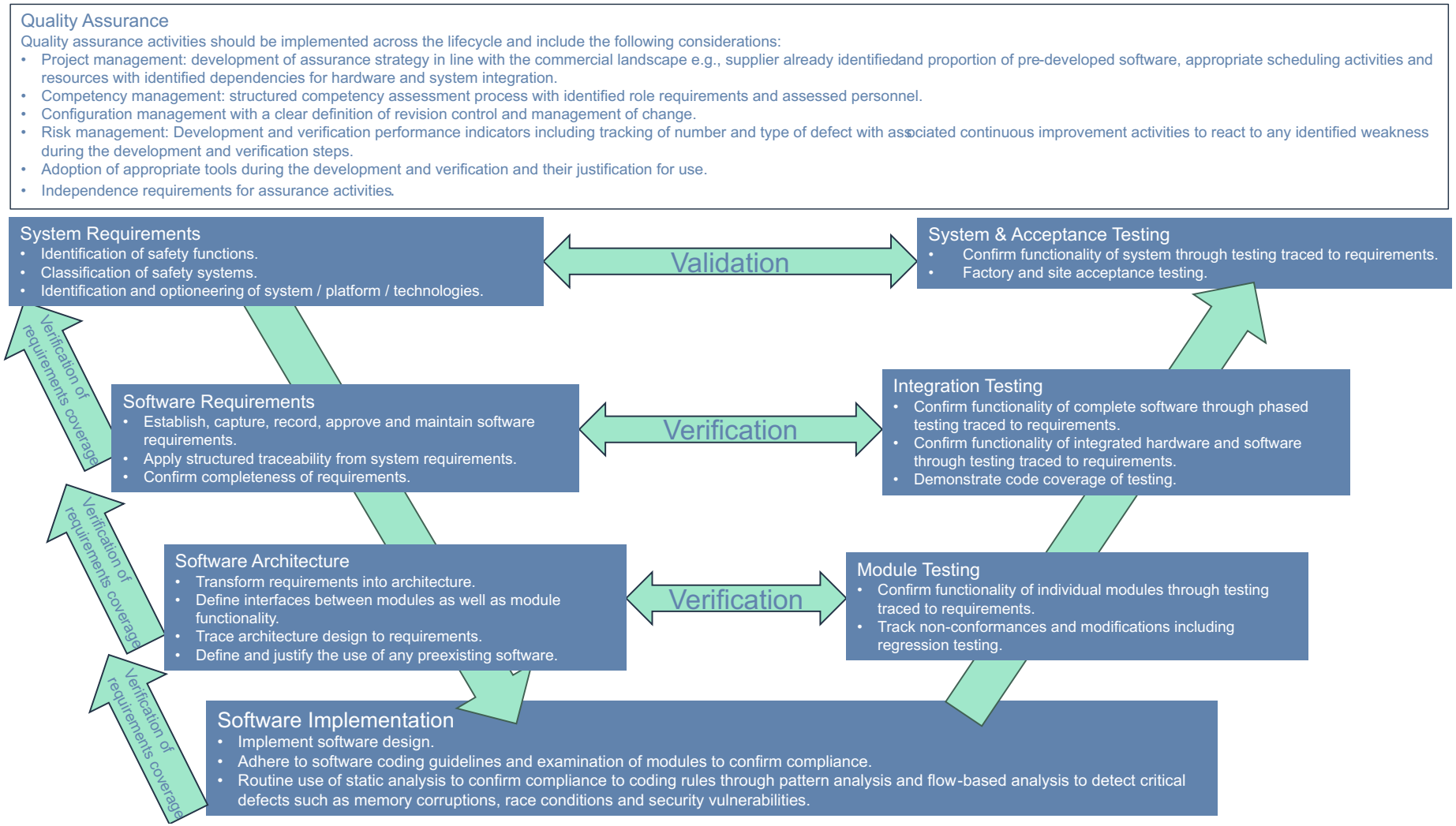

Figure 1 Software lifecycle – V-model

Despite some potential disadvantages compared to other models (e.g. waterfall), this research has found that use of the V-model in the software development lifecycle lends itself to delivering CBSIS software that can be demonstrated to have, for example:

- A high level of traceability of requirements.
- High integrity given its focus on analysis and testing throughout the development of the software.

The focus on structured and systematic development and testing within the V-lifecycle ensures compliance with relevant standards (e.g. (IEC, 2010), (IEC, 2018), (IEC, 2006)) and the ability to demonstrate this in an evidential way. Confidence is also extended through independent assessment to consider the adherence to quality and development practices being conducted at multiple stages during the lifecycle. For example, software audits are discussed further in section 2.6.1.

2.3 Architecture

In the context of CBSIS, architecture can relate to two aspects:

1. I&C system architecture which includes the CBSIS architecture and its relationship (e.g. diversity, defence in depth) with other safety and safety related systems, and
2. Software architecture.

2.3.1 I&C System architecture

In terms of current good practice, this research has found that system or plant architecture design based on redundant or diverse structures represents an efficient means of satisfying safety objectives relating to reliability and defence-in-depth. However, introducing unnecessary redundancy and diversity can introduce complexity and cost. Often the need for diversity and / or redundancy is driven by deterministic expectations (e.g. addressing potential common cause failure, single failure criteria, etc.) and meeting integrity and reliability expectations. Effective system architectural design also aids the ability to demonstrate that the overall response of plant to a fault is not over reliant on a single system or component and arrangements to mitigate common cause failure are robust. This includes the need to capture the design intent of a systems in the requirements. For example, aspects of the design intended to protect against CCF (e.g. avoiding the use of CBSIS) across diverse systems.

Effective CBSIS architectural design results in the potential to reduce the need to demonstrate software reliability. For example,

- The use of smart devices can be simplified through claims and justification that the embedded software (or part of) cannot influence the output of the device and therefore does not impact the delivery of the safety function(s).
- Or additional diversity is needed to reduce the claim on software and therefore reducing qualification requirements.

2.3.2 Software architecture

Approaches such as systems engineering (including requirements-based engineering) naturally support a modular approach to software development. Increasing focus on software requirements elicitation can be shown to significantly help to manage complexity (IAEA, 2022) and costs throughout the software and broader system development lifecycle. Effective software requirements specification should consider the assurance at a modular level. Early

consideration of the ability to assure CBSIS, including testing, significantly assists in the demonstration of requirements and subsequent integration of system modules. This can also have further benefits that are applicable to the operational phases of safety systems and maintenance by clearly defining test requirements and facilitating modification impact assessments.

The architecture and structure of a CBSIS should be complemented by diagnostic and monitoring capabilities at a system architecture and software module level where appropriate so that errors and failures are detected sufficiently early to maintain the required system performance.

There may be merit in further examining the potential for formal and/or semi-formal methods to be used in software requirements specification for bespoke developments at platform and application levels (see 2.6.4).

2.4 Requirements specification

Available data suggests that more than 60-70% of defects found after unit tests occur as a result of requirements specification errors (HSE, 2003) (J McDermid D. P., 2001). Therefore, confidence in software is underpinned by the need to establish, capture, record, approve and maintain requirements throughout the design lifecycle. This is done by undertaking software requirements analysis based on a flow-down of requirements using a system engineering approach.

The CASS Scheme Limited has produced technical schedules and guidelines intended to allow any manufacturer, user, or assurance body to satisfy themselves as to the validity, appropriateness and correctness of arrangements for their particular application (CASS, 2022) through considering a graded approach of relevant international standards (e.g. (IEC, 2010), (IEC, 2006), and (IEC, 2018)). The CASS assessment framework aligns with software assurance audit (see section 2.6.1) but includes a significant focus on requirements specification which highlights its importance. The framework includes the need to assess:

- Evidence that the software safety requirements have been properly specified. These requirements should be derived from and traceable to requirements specified at a system level.
- Requirements should describe the function of the system and its associated integrity, how the functionality will be maintained and how the software will handle, for example, internal and external conditions that may compromise maintaining the safety function. The CASS scheme and tools such as the Cogs framework can help support the consideration of completeness which should include functionality, failure integrity, operability, robustness and diagnostic functions.
- Specification of non-functional and process requirements and constraints such as accuracy, time response, reliability and security.
- Articulation of the requirements to promote consistency, cohesiveness, specificity, precision, readability, and understandability.
- Traceability of requirements throughout the development process.
- Collaboration between disciplines (e.g. hardware, software).

- Quality of independent verification of requirements.

Increasingly, software engineering and systems engineering processes are being integrated to help the development and assurance tracking of requirements. These systems have been found to enable the early detection of errors, trace requirements and support the design rationale throughout the systems design and its documentation. It also assists by giving focus to the most complex safety critical aspects of the system and aiding communication across engineering disciplines (NASA, 2003). One such process is model-based systems engineering (MBSE)[4] which can enable the structured development of system requirements to facilitate automated design workflows if properly implemented as well as auto generation of test cases.

The four purposes of software assurance, described in section 1.3.2, emphasise the need to demonstrate that robust requirements (purpose 1) have been met (purpose 3) and are subject to independent assurance (purpose 2). These form the basis of the CBSIS substantiation and demonstrating the system reduces risks to ALARP (purpose 4). Specification of system requirements and the routine collection of defined evidence throughout the CBSIS lifecycle provides a traceable method to support software assurance and ultimately demonstrating that the risk associated with the use of a computer-based system have been reduced to ALARP. Requirements traceability supported by robust evidence provides a rigorous form of demonstrating the system delivers it intended functionality and operational performance.

Also, there is likely to be significant merit in further investigation of the potential savings in terms of both project schedule and cost savings in the nuclear sector that may arise from innovative methods such as correct by construction (CbyC) (Praxis-HIS, 2005), prototyping and sandboxing. Formal and semi-formal specifications are proven to help ensure system and software requirements are correct, however, their practical use is often limited by the need for specialised expertise. Albeit there is growing availability of commercial tools that support correctness claims.

2.5 Independence

This research has found that there is a clear imperative within a range of international standards (IEC, 2011), (IEC, 2010), (IEC, 2018) and (IEC, 2006) that support the role of "independent assessment" across the software and system lifecycle[5], including the design and development of a CBSIS. The effective use of independent assessment reduces potential bias and enables appropriate, robust, and competent challenges to be made to the designers and developers of a CBSIS and its software (e.g. Functional Safety Assessment (FSA) in (IEC, 2010) and (IEC, 2011)). It also provides Purpose 2 presented in section 1.3.2: '*Independent software assurance arrangements and activities demonstrate safety, taking account of the role and application of the software-based safety system*.'

The use of robust challenge by competent assessors independent of the CBSIS suppliers and/or developers is reflected in regulatory guidance (ONR, 2023) in terms of confidence building. In VARIA, the objective of the approach is to make a positive contribution to safety and

[4] Approaches, such as MBSE, may be supplemented by appropriate development environments and toolsets (e.g. JAMA, Ansys SCADE, etc) that can enhance requirements management and software traceability throughout the design. It should be noted that certification is currently limited to SIL 3 (i.e., $\geq 10^{-4}$ to $< 10^{-3}$ PFD_{avg} assuming that a CBSIS is intended to operate in a low demand mode of operation).

[5] In practice, it is likely that these independent assessment activities may go beyond the end of the software development lifecycle and cover activities such as modifications and software upgrades.

avoid reliance upon assurance activities that are applied only towards the end of a development with their associated time and cost disbenefits due to the potential to perform rework as part of CBSIS and/or software validation.

There is a practical need to determine how best to use competent independent assessment and to agree the basis by which an independent organisation can be defined either within the end-user (e.g. licensee) or as a third-party support function (e.g. within an independent part of a software development organisation, external subject matter expertise etc). This is likely to require some degree of analysis to determine how to make the most effective use of independent assurance. For example, undertaking an assessment of organisational strengths and weaknesses and / or cost-benefit of mitigations or risks.

This research has considered an approach of using different forms of independence such that the competent assessment provides assurance throughout the CBSIS lifecycle. This should include competency relating to the software-based system and contextual functional knowledge.

For example, the system focussed independent assessment team are likely to engage regularly throughout a CBSIS development. This may include:

- Independent assessment of system architecture.
- Oversight of the CBSIS development process and associated analysis.
- Oversight of system configuration and associated independently defined assurance activities.
- Application of a range of diverse and challenging assurance techniques and measures across the lifecycle.

It is expected that any identified shortfalls during software development will lead to timely recommendations and actions such as extra testing or analysis.

2.6 Assurance of the software

International standards (IEC, 2006) (IEC, 2010) (IEC, 2011) (IEC, 2018) (IEEE, 2016) (IEEE, 2018) (IEEE, 2016) describe T&M in a number of generic types with suggested level of importance often implied by descriptors such as not recommended, recommended, highly recommended, etc. Whilst these standards are often sector specific, they are implementation agnostic. There is no single technique or measure, or combination, which can be considered good / best practice. The application of suitable T&M should be selected to demonstrate specific requirements have been met (e.g. demonstrate a specified behavioural claim). This requires significant levels of professional judgement and independent challenge to demonstrate requirements have been met to the necessary levels of assurance. As shown by (NRC, 2011) and (J McDermid D. P., 2001), the best approach to software assurance requires necessary quality assurance arrangements and the selection of activities that provides a demonstration that requirements have been met and takes account of the effect of uncertainty and known limitations in a manner that does not compromise the safety goal.

This should include:

- Collection of complementary evidence and addressing any uncertainty associated with conflicting evidence (e.g. proven-in-use field data does not align with the quality

inferred from static analysis, or dynamic testing reveals performance characteristics that differ from that specified).

- Evaluation of sufficient evidence to expose weaknesses or limitations.
- Justification of how uncertainties have been managed and / or mitigated.
- A graded approach dependent on safety significance.
- Identification of system features and characteristics including interface with other systems and operational environments.
- Consideration of a modular approach and modern coding good practice.

Identification and deployment of suitable techniques and measures is required to meet purpose 4 stated in section 1.3.2: '*Uncertainty and known limitations are reduced to an acceptable level across the system lifecycle and in a manner that does not compromise the safety goal.*'

As the development process of software changes due to advances in technology there is a need for software assurance T&M to develop to take account for these changes. In addition, the increased availability of tools designed to eliminate the need for, or continually assess, manual coding has the potential to minimise coding errors and improve cost effectiveness of aspects such as unit testing as well as addressing a known development weakness. These advances are not necessarily recognised in standards, primarily because the standards represent consensus on practices and tend to lag recent developments. Importantly, the software assurance of integrated tools such as automated code generation needs to be included within the overall CBSIS justification.

The range of T&M to provide assurance of software is summarised by eight categories as illustrated in Figure 2. The concentric circles represent a level of rigour aligned to the classification with class 3 being the centre circle and class 1 being the perimeter of the diagram. The two categories in each quadrant are broadly related. A robust assessment of software includes the application of T&M from each quadrant. For higher integrity systems the rings depicted in Figure 2 would be more complete. Assessments of platform and application software should also be considered separately. For example, each would likely have different coding standards, access to code etc. So different T&M may be appropriate for each assessment with some crossover possible (e.g. Dynamic Testing). Examples are included in Annex A.

It is recognised that the approach suggested in Figure 2 represents a simplistic but useful tool to assess the diversity of T&M. It should not be treated as a rule-based approach. In many cases certain T&M will always be applicable (e.g., requirements capture assessment, software assurance audit and dynamic testing). Also, compensating measures applied may not be restricted by the quadrant boundaries.

Each segment is summarised in the following sub-sections.

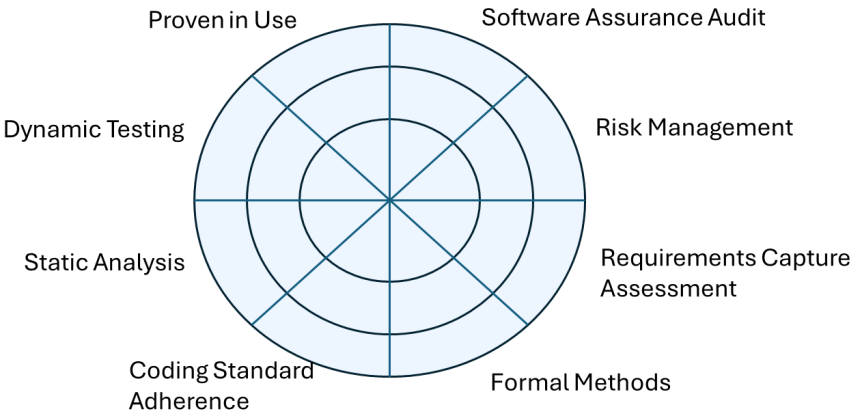

Figure 2: Illustration of the T&M categories for the substantiation of software

2.6.1 Software Assurance Audit

Software assurance auditing good practice is articulated in many standards and tools (e.g. (IEC, 2010), (IEC, 2018), (IEC, 2006), (CASS, 2022)) and includes activities such as:

- Audits of software developer's management, supply chain, quality assurance, design and development arrangements.
- Assessment of the robustness of software classification and requirements breakdown.
- Assessment of the depth and coverage of through-life T&M such as static analysis, dynamic analysis and the use of assurance tools.
- Assessment of qualification arrangements for the software-based safety system.
- Assessment of the phased verification and validation and commissioning test plan.
- Review of the application of compensating measures to account for any deficiency in other software assurance measures.
- Review of software to identify semantic issues, security gaps, and logical errors that tools may sometimes miss as well as platform specific considerations.

2.6.2 Risk Management

Like any engineering project, the management of project risk goes hand-in-hand with the management of safety risk. Project risk management is a tool for problem prevention to assure that quality goals are achieved (IEEE, 2016). Risk management should be applied throughout the lifecycle to identify potential problems including the risk associated with being able to demonstrate requirements are likely to be met, understand the potential impact, and methods of addressing the risks to assure that quality goals are achieved. This activity may normally be included in the vendor arrangements and their implementation as part of the quality management system and needs to be reflected in any assessment of CBSIS design and development. Effective risk management can also make a significant contribution to the management of uncertainty and assessing assumptions made during previous stages of the software development process.

Whilst risk management is adopted at a project level, a specific approach for software development can add confidence including consideration of:

- Project management including software development planning that takes account of project and software risk across the lifecycle.
- Project governance (e.g. peer reviews, inspection).
- Identify risks to the project in line with the project risk management approach.
- Use of tools (e.g. defect tracking including severity level, non-conformance assessment including the identification of systematic fault potential, software quality metrics) to analyse risks and determine the priority for their mitigation and define mitigation activities.
- Taking corrective actions when expected quality is not achieved.
- Ensuring effective communication between individuals and groups for the resolution of software project risks including the supply chain.
- Review of arrangements and their implementation as part of the quality management system audit and the assessment of the design and development.

2.6.3 Assessment of completeness, correctness and traceability of requirements

As stated earlier, the majority of defects found after unit tests occur as a result of requirement errors. Therefore, confidence in software is underpinned by the need to establish, capture, record, approve and maintain requirements. Increasingly software engineering and systems engineering processes are being integrated to help the development and assurance by tracking requirements.

For example, Specification Tools and Requirements Methodology (SpecTRM) is being used to aid the development of specifications for large, complex safety critical systems[6]. These systems have been found to enable the early detection of errors, trace requirements and design rationale (i.e., I&C based systems and other systems) throughout the systems design and documentation (NASA, 2003).

2.6.4 Formal Methods

The aim of formal methods is to develop software (and hardware) in a way that is based on a mathematical language that can be used to model requirements and assist in coding and subsequent verification activities for software (and hardware). Strict notation is used to describe a system such that it can be subjected to mathematical analysis to detect inconsistency or incorrectness. The mathematical description of the code can be used to perform functional analysis and provide behavioural characteristics in relation to the software's specification. This analysis can provide confidence that the system meets requirements. Formal verification can demonstrate properties of the software such as the absence of runtime errors which would typically throw an exception. They can be used to analyse and verify behaviour throughout and at any part of programme lifecycle: requirements engineering,

[6] The exchange of information by experts in different engineering disciplines during facility design can benefit from using formal languages. The largest contributor to systematic errors during design are the specification errors resulting from the incompleteness of the requirements or misunderstanding of the requirements and constraints prescribed by those in one engineering discipline to the other. The lack of ambiguity of formal languages and the possibility for formal analysis significantly reduce the potential for misunderstandings and incompleteness.

specification, architecture, design, implementation, testing, maintenance and evolution (J Wookcock, 2009).

In practice, different formal analyses of the software are necessary as each only covers specific aspects. Typically, formal verification is not performed in the target hardware, although there are now some techniques (in autonomous systems) which enable this. So, whilst formal methods cannot replace dynamic analysis and testing, it does complement verification activities. Dynamic analysis and testing can show that bugs are present but not their absence.

It is not often practical for a complete set of formal analyses to be readily adopted due to their differing strengths and weaknesses. However, the increasing availability of tools means that formal methods may be feasible for specific applications, such as reactor protection systems. Formal methods could be considered to be diverse and independent analyses to that of traditional static and dynamic analysis and testing.

Nevertheless, whilst a formal method can give a better basis for verification and validation than a semi-formal method, other project constraints (e.g. the availability of sophisticated computer support tools, or the very specialised expressiveness of a formal notation) may favour a semi-formal approach or cause significant barriers to the adoption of this assurance technique.

2.6.5 Coding standards adherence

The choice and application of coding standards make an important contribution towards software integrity by specifying good programming practice, proscribing unsafe language features, and constructing and specifying procedures for source code documentation. However, it should be noted that the generation of code through techniques such as model-based design can make assurance of coding standards more challenging (e.g. reconciling differences) albeit that it will be consistent. Robust assessment of the application of these standards ensures the benefits of coding rigour can be realised. Assurance through reviews and audits can provide evidence of:

- Fit-for-purpose selection of language, tools and compiler given the application.
- Adherence to defined coding standards. Relevant standards and guides will typically include, definitions of good practice, defined unsafe language features, promotion of comprehensibility, V&V features to be used, expected documentation (including where code is autogenerated). However, it is noted that for some new platforms, good coding standards often lag behind the technology development.
- Software structure and defensive coding expectations (e.g. software design and validation plan). Appropriate use of structure, modularisation and functional decomposition.
- Appropriate use of automated code checking.
- Code being developed to facilitate testing.

As noted by the MISRA Consortium (MISRA, 2020) when assessing compliance to coding guidelines, the compliance assessment must be an integral component of the code development phase and compliance requirements need to be satisfied before code is submitted for review or unit testing. A project that attempts to only check for compliance late in its lifecycle is likely to spend a considerable amount of time in re-coding, re-reviewing and re-testing, and it is easy for this rework to introduce defects by accident. Code is required to be

checked for compliance as it is developed and not as part of a "tick-box" exercise to be completed as part of the final delivery phase.

The use of substantiated computer-aided software engineering tools can be used to detect non-compliance with coding standards during the development process and should be considered as essential for use when developing higher integrity systems and, where practicable, formal/ semi-formal methods should be applied.

2.6.6 Static Analysis

Static analysis is a "process of evaluating a system or component based on its form, structure, content or documentation" (IEC, 2006). It provides a diverse method (from dynamic analysis and testing) to help maintain code quality primarily through the software development stages detecting issues such as syntax errors, dead code, race conditions, and whether a variable is within expected bounds before it is used. There are a number of approaches which evaluate different characteristic and potential errors, for example:

- Pattern-Based Analysis: Identifies code patterns that violate predefined coding rules, helping to prevent defects like resource leaks (which can result in system slowdown) and security issues.
- Flow-Based Analysis: Analyses control and data flow paths to detect critical defects such as memory corruptions, race conditions, and security vulnerabilities.
- Complexity Analysis: Measures code complexity to identify areas that may be prone to errors.

Static analysis tools continue to evolve to address the complexities of contemporary software development. They are often implemented as part of continuous integration/continuous deployment (CI/CD) approaches which facilitates automated code scanning throughout software development. Detection of errors and vulnerabilities at an early stage is powerful as it provides immediate feedback to the developer (CASS, 2022). In addition, the adoption of static analysis early in the lifecycle can help identify security vulnerabilities (e.g. Static Application Security Testing) which are used to detect vulnerabilities early in the development process).

Regular peer review of code throughout development by competent persons and / or independent assessment can also play an important role in identifying semantic issues, security gaps, and logical errors that tools may sometimes miss. Undertaking these peer reviews on a modular basis prior to integration of sub-systems can prevent errors and vulnerabilities going undetected and help reinforce coding standards and software architecture rules.

This research has found that, where possible, static analysis should be used as a T&M through the whole software development lifecycle to avoid reliance on assurance activities undertaken at a late stage in the development lifecycle. It is however recognised that this may not be possible for predeveloped systems, but formal methods could still be utilised for application software.

2.6.7 Dynamic analysis and testing

Dynamic analysis is a "process of evaluating a system or component based on its behaviour during execution" (IEC, 2006) whereas dynamic testing involves "executing software and/or operating hardware in a controlled and systematic way, so as to demonstrate the presence of the required behaviour and the absence of unwanted behaviour" (IEC, 2010).

Dynamic analysis and testing are particularly effective at identifying problems such as memory leaks, buffer overflows, performance bottlenecks, race conditions and improper error handling. It is used to verify that the software functions correctly and as intended.

Dynamic analysis and testing are normally performed during later stages of the software development lifecycle although it can be undertaken at several levels (e.g. unit testing, integration testing, system testing, acceptance testing). It therefore plays an important part of the right-hand side of the V-diagram in demonstrating functional and non-functional performance.

These measures (e.g., equivalence partitioning, boundary value, module testing, stress testing, performance testing, security testing, etc) provide a diverse method from static analysis to identify and rectify defects that may affect functionality, performance or reliability. Dynamic analysis and testing come in a number of forms, including.

- Model based testing: This technique is used to facilitate the effective and efficient development of a set of test cases derived from models of the systems under test (e.g. decision tables, finite state machines, Markov chain models, state charts, theorem proving, model checking, event flow etc). It uses the combination of analysis models and test coverage requirements to define the programme of functional tests.

- Parametric testing: Parametric testing uses statistical methods to define “optimal” sets of tests to maximise coverage. The methods offer several benefits when testing software, particularly in scenarios where there may be underlying assumptions about the data distribution, including:

    - Statistically distributed tests (e.g. input data distribution) give a greater confidence in detecting a true effect when it exists and being representative of the operational profile of the system.

    - There can be more precision and accuracy (due to the effective use of data),

    - They are well suited to applications that contain large data sets.

    A key limitation, however, is the need to have a statistical distribution that suitably describes the performance characteristics of the software-based systems. This can be difficult due to the systematic nature of software failures. That said, there may be benefit in undertaking further research on the topic of parametric and other forms of dynamic analysis and testing and combinations of techniques.

- Statistical testing: The purpose of statistical testing is to undertake dynamic analysis and testing to derive a quantitative figure for demand based (rather than continuous operation) software reliability and it is noted by IEC60880 as a technique which may supplement confidence of class 1 systems (IEC, 2006). A key limitation of statistical testing is the time, cost and effort associated with running the large number of tests and the need to reset the system after every test so each test can be claimed to be independent from its predecessors. It differs from other dynamic analysis and testing in that an estimate of the software reliability is obtained. It is noted that attempting to claim a quantified software reliability is not without criticism:

- "The quantification of life-critical software reliability is infeasible using statistical methods whether applied to standard software or fault-tolerant software. Reliability growth models are examined and shown to be incapable of overcoming the need for excessive amounts of testing. Practical methods to prevent design errors have not been found" (NASA, 1993).
- "A problem however exists with prediction; statistical techniques are insufficient to "prove" a failure rate of better than 10^{-3} to 10^{-4} per hour prior to deployment of the system. If pre-operational claims were limited to what can be shown by testing, then this would severely limit the classes of system which could be developed and deployed, as operational achievement is around three orders of magnitude better than can be shown via testing" (A Rae, 2014).

- Enhanced functional testing: The system is assumed to be a "black box", and tests are specified which are informed by operational usage scenarios and the application environment. The tests can be specified to address known or anticipated weaknesses in the overall qualification (Kuball, 2024).

Given the significant cost, recognised limitations and pragmatic challenges of statistical testing, its application needs to be assessed to ensure any potential safety benefit compared to other T&M is understood and justified by requesting parties, licensees and regulators. This research identifies the advantage of alternative methods of dynamic analysis which aim to provide equivalent level of assurance.

2.6.8 Proven in use

Field experience from different applications is often used to build confidence either during CBSIS system integration and / or during system safety validation. In order to be valid, the system or component under consideration needs to be unchanged over a sufficient period of time (at least 1 year in accordance with Part 7 of (IEC, 2010)) in numerous different applications). In addition, all the operating experience must relate to a known demand profile of the functions of the software element to ensure that increased operating experience genuinely leads to increased knowledge of the behaviour of the software element relative to the demand profile. There should be no recorded safety-related failures.

Proven in use may go some way to assisting in the evaluation of complex components with a multitude of possible functions (e.g. operating systems). The developer needs to keep a record of the functions being "tested" through use of field experience. Care is needed to ensure the available data does not become biased due to aspects such as equipment disposal without investigation upon failure. Consequently, it is recognised that "only in rare cases will "proven-in-use" be a sufficient argument that a trusted element achieves the necessary safety integrity" in accordance with Part 7 of (IEC, 2010). These limitations are compounded by difficulties related to collection and assessment of failure data and / or software failures typically leaving no trace (J McDermid D. P., 2001).

3 Comparison of VARIA with the ONR's two-legged approach

In the UK, the nuclear safety regulatory authority, the ONR, expects the suitability of software substantiation to be demonstrated through the use of a two-legged approach (ONR, 2014) (ONR, 2023). This approach includes:

- Production excellence which is:
  - The application of design practice and quality management consistent with accepted standards.
  - The verification of the production process and validation of the installed system(s) by a person not involved in the specification and design activities.
  - The application of a testing programme prior to and following installation by a person independent of the system specification, design or development process to confirm that the system performs in accordance with its requirements specification.
- Independent confidence building measures which are:
  - Assigned to reflect the system's ability to deliver its intended functionality.
  - Diverse checking of the final validated software, including the production process.
  - Software checking / analysis of the final system.
  - Assessment of the testing programme.

This framework presented using VARIA is not intended to undermine or contradict the two-legged approach. Instead, VARIA complements and builds upon the good practice used in nuclear in other regulatory jurisdictions and other high hazard industry sectors to embed software assurance earlier in the development lifecycle. In this context VARIA can be used to demonstrate that the risks associated with the use of CBSIS have been reduced to as low as reasonably practicable.

VARIA places greater emphasis on the role of systems engineering as an integrated approach to enable the substantiation of CBSIS whereby it can be realised and used in accordance with its requirements specification. This approach applies systems engineering principles and management methods within a structured development lifecycle. It uses governance frameworks to manage risk, complexity, and uncertainty, while also tracking the success of each stage in developing the CBSIS.

This research has recognised that a range of good practice that underpins VARIA can be associated with the production excellence leg of ONR's two-legged approach and are applied by licensees throughout the CBSIS development lifecycle. This includes the need to:

- Establish and implement a strategy for the delivery of the system functionality including choice of technology, system architecture, coding standards, quality assurance, requirements, V&V, configuration and risk management.
- Establish robust arrangements for independent assessment of the software production process and the software itself, including functional safety assessment.

- Determine the categorisation of the safety function(s) and classification of the safety system that implements the function(s).
- Develop a comprehensive and robust specification of software (and hardware) requirements,
- Design and implement the software according to codes and standards, including the utilisation of unit analysis and testing throughout the production testing with the necessary independent checking consistent with the system classification.
- Incorporate modern software practices including the routine application of static analysis to identify faults early and assist in developing confidence in the software production process.
- Implement the defined software test programme through the systems integration process using analysis tools and techniques and dynamic analysis and testing consistent with expectations within standards to demonstrate requirements have been satisfied.
- Undertake factory acceptance and site acceptance tests to demonstrate that the system delivers its intended functionality.

However precedence in the UK, (HSE, 2008), (ONR, 2017) and (WNA, 2021) through ICBMs, has added significant additional tasks when compared with other sectors and other nuclear regulatory jurisdictions. Many of these measures involve repeating T&M that have been used during system development albeit that these are applied at the end of the development lifecycle resulting in their contributing towards the “back-end” loading adding significant costs with only limited improvements to overall safety. This also contradicts systems engineering good practice.

There is a fundamental difference between the two-legged approach and VARIA in the way these methodologies treat evidence availability. The two-legged approach typically operates on the assumption that early development data cannot be sufficiently influenced or collected. This drives the emphasis on independent measures at the end of the lifecycle. VARIA promotes a continuous evidentiary model. It recognises that while influencing early development is a universal challenge, the VARIA framework does recognise that it may be possible to systematically collect evidence to verify software through life and if necessary, implement a system architecture to account for the lack of through life verification data with the possibility of avoiding costly back-end V&V. If through-life evidence is available, VARIA ensures it is utilised to build the safety case; if it is missing, shortfalls are identified as a known risk early in the project and managed accordingly.

The following provide examples of where the application of ICBMs results in additional effort within the UK.

- **Independence**

  All international standards and other guidance reviewed during this research require independent assessment to be undertaken in some form as part of the safety system governance. This enables appropriate technical and internal regulatory challenges to the designers and developers of a CBSIS and its software (e.g. Functional Safety Assessment (FSA) in (IEC, 2010) and (IEC, 2011)) and independence management in IEC standards.

Discussions with experts during this research identified a lack of clarity in the competency[7] of independent assessments. This highlights the need to strike a balance between in-house assessors[8] who are more likely to have the context specific experience to make insightful contributions to the assessment and third-party assessors with deeper software knowledge, including persons with expertise from equipment suppliers.

ONR's production excellence expectations require both:

- The need for independence from "persons not involved in the specification and design activities" for V&V activity to demonstrate the performance against the system's specification prior to installation on site; and
- The need for independence from "persons not involved in the system's specification, design or manufacture" following installation.

The VARIA combination of topic focussed independent expertise with deeper knowledge of software engineering coupled with the application of FSA is considered to satisfy expectations of independence required in international nuclear standards (i.e. IEC 61513 (IEC, 2011) and IEC 61508 (IEC, 2010)).

However, in the UK, practical application of the two-legged approach has led to significant additional effort being applied as a result of ONR expectations for ICBMs towards the back-end of the system development to introduce additional independent organisations.

- **Application of additional ICBM techniques and measures**

  International standards, for example (IEC, 2006) (IEC, 2010) (IEC, 2011) (IEC, 2018), describe T&M in a number of generic types with a suggested level of importance often implied by descriptors such as 'not recommended', 'recommended', 'highly recommended', etc. Whilst these standards are often sector specific, they are implementation agnostic. Many of these standards can be considered to form good practice in the UK and are normally used during the software production process[9]. In practice, modern software development tools have increased the systematic application of these techniques and measures. Additionally, for nuclear applications, original equipment suppliers apply dynamic analysis and testing and static analysis through their V&V processes with appropriate FSA oversight. Whilst the wording of the two-legged approach does not necessarily imply that additional T&M[10] need to be applied, precedent over the years has resulted in the significant and costly application of independent

[7] The competences that need to be demonstrated by personnel involved in independent assessment of a CBSIS, including its hardware and software elements, is not well defined. Consequently, regardless of what approach is applied in independent assessment, it is considered that there is potential for practical shortfalls that might lead to a loss of assurance which may affect the veracity of substantiation reports without some form of demonstration of relevant competences.

[8] For practical purposes the personnel undertaking the independence function will be required to have a detailed knowledge of the project but should be independent from the project itself with different management reporting lines.

[9] This research has found that static analysis should be used as a T&M through the whole software development lifecycle to avoid reliance on assurance activities undertaken at a late stage in the development lifecycle.

[10] ICBM (ONR, 2014) definition states the need for a thorough independent assessment of the safety system's fitness for purpose including 1) check and assessment of the final validated software, and 2) the whole testing programme.

techniques, such as additional static analysis and statistical testing. This often leads to additional project time and cost for UK application only.

- **Taking credit for regulatory scrutiny overseas**

  Globally, many computer-based safety systems utilise a limited set of platforms that often have had regulatory scrutiny in other jurisdictions. The benchmark for these assessments is often international standards that are either recognised in multiple countries or are broadly consistent with UK relevant good practice; for example, a review of (NASA, 2022), (IEC, 2006), (IEEE, 2016) and (IEEE, 2016) has identified many consistent themes. An example of this is the application of independent software assurance auditing where good practice is articulated in many standards (e.g. (IEC, 2010), (IEC, 2018), (IEC, 2006)). The review of a number of approaches adopted in nuclear and non-nuclear standards suggests that more credit than is currently accepted in the UK can be taken for activities conducted by other regulatory authorities throughout the software development process and quality assurance arrangements. This research suggests that by reviewing and reusing prior assurance activities - while considering the similarity of the new application and evaluating updated requirements, assumptions, and changes since the last audit - the need for additional end-of-lifecycle measures in UK software development could be reduced. This is considered to be consistent with ONR's GDA guidance (ONR, 2019) which advocates that assessment of new reactor designs should consider the activities of other regulators.

  For example, the Mitsubishi Electric Total Advanced Controller (MELTAC) is a programmable logic controller (PLC) platform with field programable gate array (FPGAs) used within communication modules. It consists of a pre-defined set of hardware and software components developed for nuclear applications. It was initially designed, qualified and manufactured in accordance with Japanese nuclear safety and quality standards. The US Nuclear Regulatory Commission (NRC) evaluated the MELTAC system for general applications within safety systems for nuclear power generating stations in accordance with US NRC regulations (NRC, 2018). This included an audit at the Mitsubishi Electric Company facilities to evaluate the effectiveness of the company's software development activities and to confirm that processes described are being effectively implemented to achieve a high-quality system that can be used to perform safety-related functions in a nuclear facility (NRC, 2018). NRC's safety evaluation included a review of the processes used for the development platform and application software, the development and test plans, specifications and procedures for design, and V&V activities. The NRC's assessment excluded evaluation of the integration and testing of plant specific system applications, factory acceptance test of plant systems and maintenance activities to support installed plant systems on the basis that these aspects would be assessed by NRC at a later stage.

- **UK emphasis on the use of statistical testing**
  The ONR SAPs (ONR, 2014) describe statistical testing as a "highly recommended" approach for demonstrating the numerical reliability of computer-based safety systems. Previous precedent has emphasised its regulatory importance in the UK (e.g. role in the substantiation of CBSIS used at Sizewell B and currently at Hinkley Point C as well as previous GDA exercises). It has also been used as a pragmatic technique for Smart instruments and devices where access to code may be a challenge and for lower integrity claims.

ONR's emphasis on the need to substantiate a numerical reliability target for software has been a key driver for the use of statistical testing in the UK. Given the recognised limitations of the proposed reliability derived from statistical testing of software, see section 2.6.7, this research proposes that it should be possible to use alternative forms of dynamic analysis (i.e. not statistical testing) to provide the required confidence in the ability of the system (including software) to deliver requirements.

A key limitation of statistical testing is the time, cost and effort associated with running the significantly large number of tests and the need to reset the system after every test so each test can be claimed to be independent from its predecessors. Dynamic testing will always play an important component of CBSIS assurance. This research has identified that other forms of dynamic test methods (see 2.6.7) may result in a more pragmatic and effective approach to provide the necessary systems assurance.

- **Proven in use.**

  The limitations of proven in use as a robust measure for software assurance are well documented. However, as a consequence of the general acceptance that requirements are the biggest source of error, where one system is a close derivative of another, then proven in use data is considered beneficial for inclusion in a broader safety case justification of software (J McDermid D. P., 2001). This would be appropriate in nuclear applications where safety functions implemented by CBSIS typically remain consistent throughout the operational lifetime of a plant and, as there are fewer suitable commercial platforms for nuclear plants to select from, there should be knowledge to be gained by co-operation between plant developers and the supply chain that can be mutually beneficial to all users. It is recognised, however, that the usefulness of this T&M is predicated on an assessment of the quality of the available data and the level to which it is representative of the specific application.

In summary, the application of ICBMs and its precedent in the UK as part of ONR's two-legged approach results in significant additional effort being applied at the back-end of the lifecycle. This differs from regulatory arrangements in other countries and results in the application of additional assurance activities and the use of third-party independence towards the back-end of software development. This research has identified an alternative approach that aims to be more effective and efficient.

Many modern CBSIS utilise computer-based platforms and individual commercial off the shelf (COTS) devices that are used already in nuclear facilities in other countries or other high hazards sectors. There is, therefore, clear benefit in having software assurance methods that build on good practice applied outside of the UK nuclear sector where applicable. The intention being that some of the costs associated with testing and assurance in the two-legged approach can be reduced whilst maintaining safety and security standards. In addition, it is recognised that there are many similarities in requirements expressed across these international standards, which may result in assessments undertaken by a regulator in one country being broadly acceptable in another jurisdiction.

4 The Future

In 2024 a group of UK experts considered software risks for critical national infrastructure (CNI) towards 2040 (Lancaster, 2024). As expected, these considerations highlighted the predicted:

- Continual expansion of system operation through software rather than through hardwired systems or human intervention,
- Increased use of decentralised control,
- Greater availability of COTS hardware and software; and
- Additional environmental considerations from climate change.

It can be expected that wider industry development of C&I systems and a reduction of available non-computerised systems will result in the nuclear industry needing improved risk management processes, availability of competent persons, development of new T&M and education of end-users. It is therefore suggested that the optimum method for substantiation of software-based safety systems is likely to evolve over time as new methods and associated technology develops.

5 Conclusions

In the UK, the two-legged approach to software substantiation (ONR, 2014) (ONR, 2023) has historically resulted in the application of significant additional assurance activities at the back-end of the development process to satisfy regulatory expectations (for example, the application of statistical testing and final code static analysis for high integrity systems). This contradicts established systems engineering good practice. This research has found that other high hazard sectors (e.g. aviation, space) and nuclear facilities in other countries (e.g. USA) do not have such expectations for software substantiation.

The starting point for this research defined the high-level purpose of software assurance as:

1. Software safety functional and non-functional requirements are robust and include all failure modes.
2. Independent software assurance arrangements and activities demonstrate safety, taking account of the role and application of the software-based safety system.
3. Software safety requirements are demonstrably met across the system lifecycle in a way that the evidence links to the performance of the system's safety.
4. Uncertainty and known limitations are reduced to an acceptable level across the system lifecycle and in a manner that does not compromise the safety goal.

VARIA provides a framework to assist C&I safety engineers, subject matter experts and associated disciplines to define and implement software assurance that has the potential to avoid costly "back-end" loaded assurance activity whilst maintaining nuclear safety. It is possible to map aspects of the VARIA framework to the purposes of software assurance. As illustrated in Figure 3. It is considered that this mapping illustrates the completeness of the model and provides a clear, evidenced based route to support the demonstration of the acceptability of the CBSIS. By shifting the safety case from a retrospective exercise to a proactive framework, VARIA encourages the licensee to actively determine the strength of the

evidentiary base throughout the system's entire lifecycle and if this evidence is not available, to implement CBSIS with, for example, the necessary architecture to mitigate the associated risks.

There is no single technique or measure, or combination, which can be considered good/best practice. The application of suitable T&M is use case specific and requires significant levels of professional judgement and challenge to demonstrate requirements have been met to the necessary levels of assurance. The best approach to software assurance requires the selection of activities to provide a demonstration that requirements have been met as well as taking account of the effect of uncertainty and known limitations in a manner that does not compromise the safety goal.

Whilst it is recognised that a requesting party or licensee will have to make a robust case, the VARIA framework is intended to provide confidence in the reliability and robustness of a CBSIS, including its software, through a combination of:

- **V**-model (lifecycle) – Processes and approaches that enable system development and substantiation to be effectively managed across a lifecycle with the emphasis placed on, as necessary, iterative activities and whole lifecycle management whilst adopting practices aligned with relevant international standards and other guidance that describe good practice in software engineering.
- **A**rchitecture – VARIA recognises the importance of establishing a system and software architecture that supports diversity and defence in depth early in the lifecycle.
- **R**equirements – Research supports the implementation of a systems engineering approach that stresses the importance of establishing a strong requirements basis for software-based safety systems that should be linked to provide clarity and traceability. This would likely lead to greater use of software design and development tools with a clear rationale for their use as the most appropriate for the CBSIS software under assessment.
- **I**ndependence – Safety system development needs robust independent challenge provided by competent persons knowledgeable in the application throughout the software lifecycle.
- **A**ssurance – An integrated, risk-based approach to software assurance including quality assurance arrangements and V&V activities, such as through life static analysis robust dynamic analysis and testing. The routine application of statistical testing for high integrity systems is questioned due to the level of uncertainty associated with the reliability figure obtained, and the availability of equally effective and more efficient dynamic analysis and testing methods.[11]

[11] This may involve the application of risk analysis techniques applied to evidence that becomes available during CBSIS development to address uncertainty at earlier points in the software lifecycle.

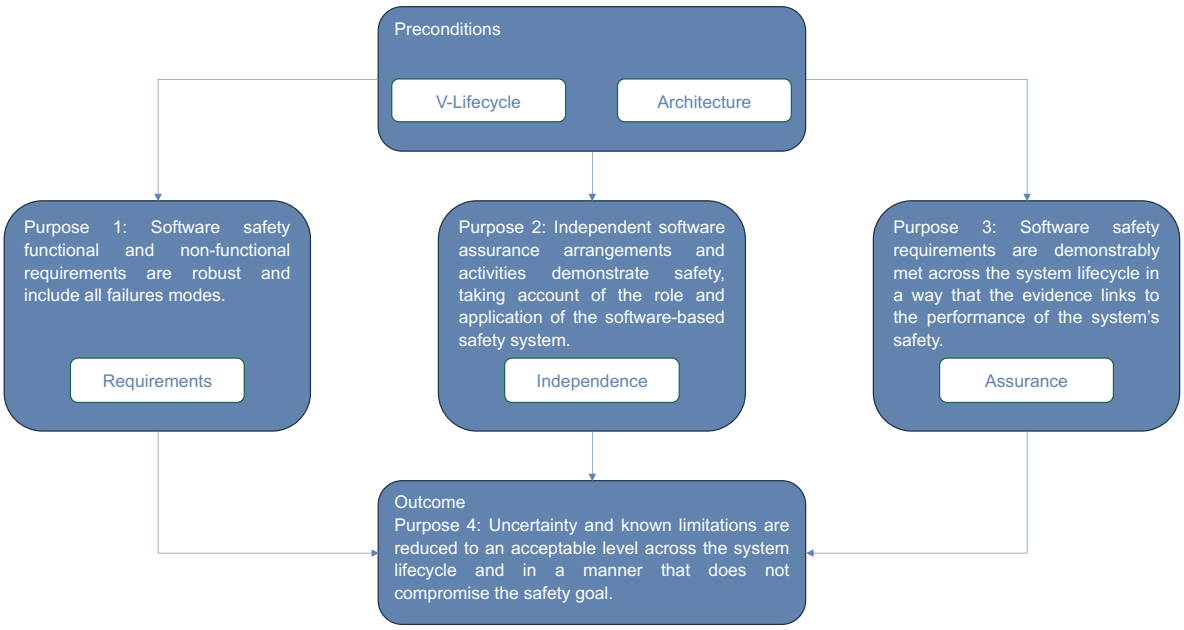

Figure 3: Mapping of VARIA components to the purpose of software assurance

The VARIA framework also aims to maximise the use of COTS substantiation from nuclear facilities in other countries or other high hazards sectors to ensure the UK nuclear sector benefits from pragmatic substantiation and use of CBSIS to aid safe nuclear operations. Additionally, with the strong emphasis in VARIA on selection of suitable techniques and measures, the experience and competencies of personnel involved in the development and assessment of CBSIS software must be adequately demonstrated. This competence is also critical when assessing the value of new techniques and methods and associated technology.

In establishing VARIA, it is apparent that in an organisational context there may be a need to initiate management of change initiatives, e.g. to introduce systems engineering concepts and, as necessary, adapt the management system to align with the framework. Given that this framework differs from the approach adopted by some regulatory authorities it should be discussed, and as appropriate agreed, as the basis for an alternative form of substantiation for CBSIS software.

To further progress this topic, the research concludes that there is a benefit in further consideration of:

- Regulatory assurance frameworks that facilitate the efficient and effective assessment of previous regulatory oversight.
- Testing methodologies to support a shift away from the UK's preference for statistical testing through the inclusion of other forms of systematic dynamic testing with the aim of reducing cost and improve realism in software testing.
- Independent assurance good practice including assessor competency framework.
- Continued monitoring and assessment of the value from modern assurance techniques, including assessment of the value and practical application of:
    - Formal Methods.

  - Correct-by-Construction (CbyC).
  - Sandboxing.
  - Complexity reduction methodologies.
  - Automatic code generation.
  - Software risk management strategies and metrics.

- Socialising the findings of this work to gain further insight, continue to share good practices and implement change to reduce UK specific software assurance burdens.

6 References

A Rae, R. A. (2014). *Fixing the cracks in the crystal ball: A maturity model for quantitative risk assessment*. Department of Computer Science, University of York.

CASS. (2022). CASS Templates for software requirements in relation to IEC 61508 Part 3 Ed 2 Safety Function Assessment. Version 2.

Guerra S, C. N. (2015). *Justifying digital COTS components when compliance cannot be demonstrated - The Cogs approach*. London: Adelard LLP.

Holloway, C. (2019). *Understanding the overarching properties NASA/TM-2019-220292*. Hampton: Langley Research Center .

HSE. (2003). *HSG238 Out of Control: Why control systems go wrong and how to prevent failure*. Liverpool: Health and Safety Executive. Retrieved from https://www.hse.gov.uk/pubns/books/hsg238.htm

HSE. (2008). *Public report on the generic design assessment of new nuclear reactors - Areva NP SAS and Electricité de France SA UK EPR Nuclear Reactor*. Health and Safety Executive.

IAEA. (2016). *SSG-39 IAEA Safety Standards, Design of instrumentation and control systems for nuclear power plants*. Vienna: International Atomic Energy Agency.

IAEA. (2022). *NR-T-2.14 "Introduction to Systems Engineering for the Instrumentation and Control of Nuclear Facilities"*. International Atomic Energy Agency.

IEC. (2006). *IEC 60880: Nuclear power plants — Instrumentation and control systems important to safety — Software aspects for computer based systems performing category A functions*. International Electrotechnical Commission.

IEC. (2010). *IEC 61508 Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, various parts*. International Electrotechnical Commission.

IEC. (2011). *IEC 61513: Nuclear power plants — Instrumentation and control important to safety — General requirements for systems*. International Electrotechnical Commission.

IEC. (2018). *IEC 62138: Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category B or C functions*. International Electrotechnical Commission.

IEEE. (2016). *IEEE 1012 - 2016 Standard for system, software and hardware verification and validation*. Institute of Electrical and Electronics Engineers.

IEEE. (2016). *IEEE 7-4.3.2 Standard Criteria for Programmable Digital Devices in Safety Systems of Nuclear Power Generating Stations*. Institute of Electrical and Electronic Engineering.

IEEE. (2018). *IEEE 603 Standard Criteria for Safety Systems for Nuclear Power Generating Stations*. Institute of Electrical and Electronic Engineering.

IET. (2016). *Code of Practice: Competence for Safety Related Systems Practitioners*. London: Institute of Engineering and Technology. Retrieved from https://shop.theiet.org/code-of-practice-competence-for-safety-related-systems-practitioners

J McDermid, D. P. (2001). *Software Safety: Why is there no consensus*. University of York.

J McDermid, T. K. (2006). *Software in safety critical systems: Achievement and prediction*. York, UK: Institute of Nuclear Engineering.

J Wookcock, P. G. (2009). Formal Methods: Practice and experience. *ACM Computing Surveys, Vol 41 No 4 October*, pp 1 - 40.

Kuball, S. (2024). Black box testing for functional safety. *Cyber and Software Assurance Group, The 61508 Association*.

Lancaster, U. o. (2024). *Software risks for critical infrastructure towards 2040: expert forecasts*. Lancaster: University of Lancaster.

McDermid, J. (2012). Software safety and reliability: Achievements and challenges. IMechE conference preceedings.

MISRA. (2020). *MISRA Compliance: 2020, Achieving compliance with MISRA Coding Guidelines*. Motor Industry Software Reliability Association.

MPED. (2024). *Multinational Design Evaluation Programme (MDEP)*. Retrieved 2024, from https://www.oecd-nea.org/mdep/

NASA. (1993). *The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software. R Butler G Finelli.* Hampton,: NASA Langley Research Center.

NASA. (2003). *Building Safer systems with SpecTRM*. Retrieved from https://spinoff.nasa.gov/spinoff2003/ct_10.html

NASA. (2022). *NASA-STD-8739.8B Software sssurance and software safety*. National Aeronautics and Space Administration.

NRC. (2011). *Research Information Letter 1001: Software-related uncertainties in the assurance of digital safety systems - expert clinic findings, Part 1*. Nuclear Regulatory Commission.

NRC. (2018). *U.S. NRC staff safety evaluation for Mitsubishi Electric Total Advanced Controller (MELTAC) platform topical report and supporting documents, ML18257A014*. Nuclear Regulatory Commission.

ONR. (2014). *Safety assessment principles for nuclear facilities*. Office for Nuclear Regulation.

ONR. (2017). *ONR-NR-AR-17-017 Step 4 Assessment of Control and Instrumentation for the UK Advanced Boiling Water Reactor*. Office for Nuclear Regulation.

ONR. (2019). *New Nuclear Power Plants: Generic Design Assessment Guidance to Requesting Parties. ONR-GDA-GD-006 Revision 0* . Office for Nuclear Regulation.

ONR. (2023). *NS-TAST-GD-046 Computer based safety systems.* Liverpool: Office for Nuclear Regulation.

ONR. (2024). *Licensing of safety critical software for nuclear reactors. Common position of international nuclear regulators and authorised technical support organisations.*

Praxis-HIS. (2005). Correctness by Construction: A manifesto for high integrity software. In A. C. Society (Ed.), *10th Australian workshop on safety related programmable systems (SCS'05)*. Sydney. Retrieved from https://crpit.scem.westernsydney.edu.au/confpapers/CRPITV55Chapman.pdf

WNA. (2021). *Different Interpretations of Regulatory Requirements. Cooperation in Reactor Design Evaluation and Licensing – Licensing & Permitting Task Force.* London: World Nuclear Association. Retrieved from https://world-nuclear.org/images/articles/DIRR-Report-FINAL.pdf

WNN. (2021). *CORDEL: Regulatory harmonisation needed to speed SMR deployment*. Retrieved 2024, from https://www.world-nuclear-news.org/Articles/CORDEL-Regulatory-harmonisation-needed-to-speed-SM

7 Annex A – Example application of VARIA

The following provide illustrative examples of how the VARIA framework could be used to consider and define an appropriate level of diverse T&M to substantiate safety system software in a graded and risk informed manner. Examples 1 and 2 consider how the framework could be applied at both a predeveloped platform layer (example 1) and at the application layer (example 2) for a reactor protections system (RPS). Example 3 illustrates how the framework could be applied to the substantiation of a commercial off the shelf smart instrument for lower class systems (i.e. not a reactor protection system).

7.1 Example 1: The application of techniques and measures at the platform layer

This example illustrates how the framework could be applied to inform the range of T&M to be adopted for the substantiation of a predeveloped platform that has previously been assessed / approved by another regulatory authority. It is recognised that often through-life evidence is unavailable. However, from experience, especially for high value systems, strategic commercial relationships are often established with suppliers for lifecycle support. Whilst, it is not possible to say that these relationships will have the longevity of the reactor life, the contractual arrangements do have system lifetime aspects including, for example, ageing and obsolescence. For some aspects of the assurance, credit could be taken for activities conducted at the application layer and some assurance could be applied concurrently at both the platform and application level. So, in practice the totality of the assurance activities would be the sum of the T&M described in 7.1 and 7.2.

Attribute	Description
V-Diagram	• The reactor vendor fault analysis is used to attribute key functional and non-functional requirements for the CBSIS RPS. This results in clear categorisation of functional requirements and appropriate designation of the RPS systems classification. • Internal procedures to systematically track requirements and undertake activities (i.e. software design codes, roles and responsibilities for testing, through life maintenance and ageing and obsolescence). • Audit of RPS developer's lifecycle management including mapping the original V-model stages (from requirements down to module and integration testing), staged development reviews and tracking the development across the system lifecycle. • Demonstration of the application of rigorous configuration management and application of regression control. • Gap analysis between the RPS vendor development standards and those expected in the UK.
Architecture	• Any interfaces between systems intended to be independent are justified, defined and included in subsequent V&V activity. • Interfaces within the platform include fault diagnostics to minimise the impact of a component failure. • Appropriate levels of diagnostics and self-test are included without introducing any unnecessary complexity. • Activities undertaken to identify "unused features" or legacy capabilities of the platform and define of how these are isolated or structurally deactivated to ensure they cannot interfere with core safety functions.
Requirements	• For a predeveloped platform, requirements capture defines aspects such as the safety functional and integrity requirements. It also includes addressing any gaps or uncertainties associated with the original design documentation to establish a functional baseline and capture any specific platform behaviours that the application layer relies upon to ensure safety e.g. diagnostic functions. • A Traceability matrix is used to map platform-level requirement to a specific dynamic test case to demonstrate completeness and correctness of functionality. • The requirements capture also identifies "unused features" of the predeveloped platform and defines how these are isolated or deactivated to ensure they do not interfere with safety functions.
Independent Assurance	• Utilising a combination of competent safety assessors with different degrees of independence to perform a Functional Safety Assessment (FSA) on the developer's shared records. • Demonstration that suitable and sufficient independent assurance has been applied throughout the development process in line with requirements identified in standards (i.e. functional safety assurance in IEC 61508 and IEC 61513).

	• Assessment of the application of previous independent reviews conducted by other bodies / regulators including a judgement on any gaps between previous applications against the reactor vendors application. This includes sampling of the assessment material to provide confidence of the review artifacts.
Assurance of software	• Any existing regulatory assessment should be reviewed to confirm it covers the full scope of the platform application and previous gaps identified have been considered and addressed. • Software assurance audit undertaken by independent assessors to provide confidence of the systematic application of the necessary quality activities and oversight through the development process (including evidence of the outcome of unit testing, review of requirements traceability and assessment of any gaps). • Any risk associated with the RPS or the application of through life assurance activities are raised and managed with mitigating activities (i.e. compensating measures) identified to provide the necessary assurance. • Review of the systematic application of appropriate coding standards including evidence of peer checking, use of defensive coding techniques, application of coding tools, systematic development of documentation. • Review of the systematic application of static analysis and any associated evidence / metrics. Review records of automated code checking. Audit of corrective action taken across the development lifecycle. • Demonstration of the application of effective QA activities across the lifecycle including access to coding defect metrics.

7.2 Example 2: The application of techniques and measures at the application layer

This example illustrates how the framework could be applied to inform the range of T&M to be adopted for the substantiation of application software. In practice the totality of the assurance activities would be the sum of the T&M applied to the overall system (i.e. combined application and platform layers described in 7.1 and 7.2). Every effort should be taken to develop the software assurance plan at the outset with regular, searching reviews conducted as part of the governance process. However, it is likely that vulnerabilities will be identified as part of the application of these T&M. The additional of, preferably diverse, compensating measures is an important aspect of developing confidence in the final software.

Attribute	Description
V-Diagram	• The reactor vendor's fault analysis derives the specific, site-dependent safety logic functions, interlocks, and setpoints required for the RPS application software. • Arrangements for and evidence of a rigorous, phased V-model lifecycle which details the progression from functional requirements allocation to application software specification, module design, block-logic programming, and progressive integration testing. • Configuration control over the application source code, graphical logic diagrams, database constants, and calibration setpoints. Implementing a regression control framework to ensure any change to a plant setpoint or a logic block triggers targeted regression testing to guarantee that unrelated safety functions remain unaffected. • The use of internal procedures to systematically track requirements and undertake lifecycle activities (i.e. software design codes, roles and responsibilities for testing, through life maintenance and ageing and obsolescence).

	• Demonstration of the systematic application and recording of verification activities (such as peer reviews, requirements checklists, and traceability audits) to prove all application requirements have been correctly interpreted.
Architecture	• Documentation demonstrating how the application software logic uses a modular structure with defined interfaces which minimise fault propagation and assists with demonstrating code coverage of verification. • Defining and verifying the logical boundaries and data flow restrictions between independent safety functions and ensuring robust isolation exists where the application layer communicates with lower-safety-class systems (e.g., sending diagnostic data to the main control room or data logging systems). This should include consideration of the suitability of the architecture for verification (i.e., defined boundaries between modules rather than complex 'cross module' functionality). • Designing and auditing the cyclic execution model of the application programs. This involves defining maximum execution cycle times (task scheduling) to guarantee that input processing, safety logic evaluation, and trip output signals occur within the deterministic response times assumed in the safety analysis. • Ensuring that the application logic is designed with a high degree of simplicity to minimise complexity. This includes a strict review to guarantee that no unintended or undocumented functionality is introduced. • Formalising the review and architectural sign-off of the application layer's software design and signal routing by both the reactor vendor (responsible for the safety case) and the RPS vendor (responsible for platform constraints).
Requirements	• Systematically tracing the high-level plant safety functions (Category A) down into specific application-layer software requirement specifications. Capturing the precise functional safety specifications, including exact analytical setpoints, trip algorithms, defining and justifying permissive interlocks or overrides, cyber security controls and any response time constraints. • Establishing and maintaining a rigorous traceability matrix that maps high-level plant safety requirements down to individual application software requirements, and subsequently down to specific logic blocks, code modules, and dynamic test cases. This demonstrates complete functional coverage and ensures no unintended functionality has been introduced. • Conducting structured reviews of the requirements specification to identify and resolve any ambiguities, omissions, or conflicts before application-layer coding begins, thereby ensuring a verifiable baseline. • RPS platform requirements are captured by the reactor vendor (i.e. non-functional and process requirements and constraints such as accuracy, time response, reliability, qualification and security). The RPS vendor can demonstrate the systematic application and recording of assurance activities used to demonstrate platform requirements are clear and have been met (including the capture of relevant evidence). These records are made available to the Reactor Vendor and Independent Assurance. Evidence provided of the application of high standards of software engineering and systems engineering across the development stages. • The systematic use and promulgation (e.g., available to the reactor vendor and Independent Assurance bodies) of software defect tracking metrics.
Independent Assurance	• Utilise a combination of competent in-house safety assessors and independent third-party experts (completely separate from the application development team) to perform a staged FSA throughout the application lifecycle. This includes independent reviews of software specifications, logic diagrams, and test results to ensure compliance with standards. • Conduct independent assessment of any bi-directional requirements traceability matrix to confirm that all plant-specific safety functions are fully and accurately implemented in the application logic and ensuring that no unauthorised or undocumented behaviour has been introduced.

	• Audit the rigor, completeness, and independence of the application V&V activities. This involves verifying that the engineers specifying and executing the application-level tests are organisationally independent from the engineers who developed the application logic. • Performing a dedicated evaluation to ensure that the site-specific application software fully addresses any specific boundary conditions, assumptions, or open items identified in previous platform-level assessments. • Conducting independent sampling and safety audits of the compiled graphical block logic, database constants, and application configuration files to verify adherence to strict programming standards and defensive coding rules.

Assurance of software	• Software assurance. The development of application layer software includes the benefits of robust through life software assurance for the application code. This could include,
	  ○ Audits of the software development and assurance activities conducted across the lifecycle (as discussed above).
	  ○ Assessment of the depth and coverage of through life of T&M such as static analysis, dynamic analysis and use of assurance tools.
	  ○ Assessment of the phased verification and validation and commissioning test plan.
	  ○ A review of any necessary compensating measures to account for any deficiency in other software assurance measures.
	• Risk management. Given the safety significance and complexity of a computer-based reactor protection systems the application of a high level of risk management including risk monitoring and metrics to assess risks during the developing process.
	• Formal methods. The use of formal methods, for example to support requirements capture assessment has the potential to provide supporting evidence of the:
	  ○ The correctness of the application code.
	  ○ The analysis of network communication including timing characteristic.
	  ○ Any underpinning “model” used in the CBSIS checking.
	• Static analysis. The routine, through life application of static analysis with appropriate independent oversight and review to provide high levels of assurance.
	• Evaluation of opportunities to apply Formal Methods to the predeveloped platform software code or logic to provide mathematically rigor in its correctness
	• Review (including independent review) of arrangement for the integration of the platform software and application software and associated dynamic analysis / test schedule. This should demonstrate a phase approach which challenges the platform and application software and the routine collection of evidence and any resolutions necessary.
	• Dynamic testing. It is likely that this will be carried out in conjunction with the platform layer assessment. The aim being that dynamic testing provides an integrated, phased and progressive package of challenging assurance conducted from unit testing through to final active commissioning using suitable tools and equipment. This suite of tests provides the basis for tests necessary to support through life operations (e.g. maintenance and modification). Consideration of the through life role of dynamic testing, including during commissioning, to undertake complete code coverage and robust interface testing though phased and defined test and integration processes. In conjunction with effective modular design and test, effective demonstration of defence-in-depth within the C&I architecture, and alignment with other good practice to demonstrate that sufficient assurance exists. Techniques described in Section 2.6.7 allow the licensee to define and undertake these challenging tests. Independent scrutiny of the integrated test plan provides additional and important assurance.

7.3 Example 3: The application of techniques and measures for a smart device

This example illustrates how the framework could be applied to inform the range of T&M to be adopted for the substantiation of a smart device where a device of relative low integrity is needed (i.e. a SIL1 device for a Class 3 application). It is recognised that often through-life evidence is unavailable. This may make it difficult to apply the optimum V&V. VARIA provides a proactive mechanism to capture evidence as it is produced, identify evidentiary gaps early or implement a system architecture to manage the resulting risk. Potential alignment with IEC 62671 should be considered.

Attribute	Description
V-Diagram	• High-level plant requirements are used to derive the functional requirements for the Smart Instrument, particularly for the safety functions of the device. • Because through-life lifecycle evidence from the commercial vendor is often limited or unavailable, a retrospective gap analysis is performed against the expected UK software lifecycle standards. A proactive mechanism is established to log and map the available vendor documentation. • Rigorous configuration management is applied to the specific firmware version and hardware revision of the device. A process for application-level regression control is defined to manage future field modifications, device replacements, or parameter configuration changes, ensuring the device's baseline integrity is maintained throughout the plant operating life.
Architecture	• A system architecture is utilised to manage risks arising from any Smart device assessment evidence gaps with consideration of introducing diversity within the overall plant system to reduce the impact of common cause failure. • The smart instrument's software architecture is evaluated to ensure it remains simple and dedicated to its single, low-complexity function. Strict physical and logical boundaries are defined between the smart device and higher-integrity safety systems. • Assessment to identify and evaluate the device's unused features or auxiliary software capabilities (e.g., advanced digital protocols, secondary diagnostics); where these cannot be deactivated, architectural compensating measures (such as external signal validation or hardwired bypasses) are implemented to guarantee they cannot interfere with the Class 3 safety function.
Requirements	• Requirements capture process including the flow down of functional requirements through robust systems engineering (often from the broader systems perspective rather than just the Smart device) benefits from oversight from an independence perspective to assess the validity, appropriateness and correctness of requirements. • The functional, performance, and environmental constraints of the Smart Instrument are explicitly captured by the plant designer. • Requirements capture focuses on establishing a clear functional description, outlining measurement ranges, response times, and failure-state behaviours. Bi-directional traceability is established from the high-level system requirements down to the device's user-configurable parameters and calibration constants. • Where gaps or uncertainties exist in the vendor's original requirement specifications, these are formally logged as project risks and bounded through clear application-layer design constraints.

Independent Assurance	• Independent assurance is applied in a manner proportionate to a Class 3 / SIL 1 application. It involves a structured review by competent, independent assessors to evaluate the validity, appropriateness, and correctness of the requirement flow-down. Rather than auditing the vendor's secure facilities, independent scrutiny is focused on reviewing the completeness of the compiled data package, evaluating previous Type Approval or independent certification records (such as SIL 1 certificates from assurance bodies), and assessing the rigor of the proposed confidence-building testing schedule.
Assurance of software	• A proportionate software assurance review is undertaken by independent assessors to evaluate the available quality records and oversight applied during the device's production. • Given that through-life evidence is often unavailable, an active risk management framework is applied. Evidentiary gaps regarding the device's internal software are systematically raised as project risks, with clear, external compensating measures identified to fulfil the necessary safety justification. • A progressive, integrated package of challenging dynamic tests is conducted on the physical device in conjunction with its surrounding systems using suitable tools and equipment. This should include functional testing and particularly boundary conditions as well as performance testing, so the devices failure behaviour is understood. Diversity is introduced into the test specification (e.g., using independent test specifiers) to ensure the schedule thoroughly challenges the device and reveals its true performance and error-handling characteristics. • Where the smart instrument has an established commercial history, operational experience and field data are collated to support the safety case. This data is reviewed to analyse historical failure rates and confirm performance stability, while acknowledging that this data may be incomplete due to untracked field replacements.