Quality Requirements for Software-dependent Safety-critical Systems
History, current status, and future needs

Sushil Birla
Office of Nuclear Regulatory Research
United States Nuclear Regulatory Commission
Phone: +1 301 2517660, Email: Sushil.Birla@nrc.gov

Mika Johansson
Electrical and Automation Systems, Nuclear Reactor Regulation
Säteilyturvakeskus (STUK) - Radiation and Nuclear Safety Authority Finland
Email: Mika.Johansson@stuk.fi

Abstract

Whereas current engineering practice focuses on functional requirements, considerations other than the function (e.g., safety; security; maintainability) are relegated into a category (unfortunately) called "non-functional requirements." Although ISO/IEC/ IEEE 24765 §3.1900 defines this term as "a software requirement that describes not what the software will do but how the software will do it," the authoritative family of ISO/IEC software engineering standards for software product quality requirements and evaluation (ISO/IEC 25000 series) uses the term, "quality requirements" instead of "non-functional requirements." ISO/IEC/ IEEE 24765 §3.1900 notes: "Nonfunctional requirements are sometimes difficult to test, so they are usually evaluated subjectively." In contrast, the ISO/IEC 25000 family of standards suggests an objective evaluation with the help of a quality model, through which an abstract quality attribute is decomposed into measurable characteristics. The authors trace the historic evolution of these ideas, present the current state of the standards, and identify needs for future research and development of a quality model, focused on the domain of digital safety systems for nuclear power plants.

1. Introduction

Many software-dependent critical systems have failed to meet their customers' needs, even after the system was verified to provide all the needed functions, all constituents of the system met their respective requirement specifications and none, by itself, "failed." These mishaps indicate gaps between the customers' needs and the "documented requirements" against which the system and its constituents are verified. Our investigation focuses on a subset of these gaps that leads to unintended behaviour, including mishaps.

Current common practice focuses on functional requirements. High-performance organizations engineer systems relatively well in providing the nominal functions that their customers expect, i.e., in identifying and satisfying functional requirements, defined as follows, in [1] §3.1229:

> Functional requirement: (1) A statement that identifies what a product or process must accomplish to produce required behaviour and/or results. (2) A requirement that specifies a function that a system or system component must be able to perform.

Other considerations (e.g.: safety; security; verifiability; comprehensibility) are relegated to the category of "non-functional requirements," defined as follows, in [1] §3.1900:

> Nonfunctional requirement: (1) A software requirement that describes not what the software will do but how the software will do it. Synonym: design constraints, non-functional requirement… Nonfunctional requirements are sometimes difficult to test, so they are usually evaluated subjectively.

Examples given in this definition include "software quality attributes" – a term that is discussed in Section 2.

We contend that the need for engineering to prevent unintended behaviour is lost in the fuzziness with which quality attributes are treated. The term, “non-functional” clouds the need to engineer for quality attributes. The Oxford Dictionary defines “nonfunctional;” as follows:

1. Not having any particular purpose or function. Example: *in some dolphins and small whales, teeth have become virtually non-functional.*
    1.1. Not operating or in working order. Example: *the cooker was non-functional except for the hotplate’*

One wonders, “If a requirement is non-functional, why bother?”

In the rest of this paper, we review the historical background of the relevant international standards that contributed to this condition, analyze the current state of relevant standards, and conclude with thoughts on a direction for the future.

2. Historical background of relevant international standards

Tracing back from the definition of the term, “non-functional requirement” cited above from [1] to find its history, we did not find any authoritative source. The term was mentioned in ISO 9126 [3], reviewed next.

2.1 ISO 9126 – Historical background

This background is intended for understanding the context of the following suggestion in [2]:

> “A product quality model and quality requirements, such as found in ISO/IEC 9126-1 … may be useful for aiding this activity (formulation of stakeholder requirements).”

Since ISO/IEC 9126-1:2001 was cancelled and replaced by ISO/IEC 25010:2011, the following contents of this section are intended for understanding the evolution history, leading up to the ISO/IEC 25000 series. Although these standards were created for software, most of the concepts may be applied to the system level also, as indicated in [2].

ISO/IEC 9126:2001 was a 4-part standard for “Software engineering – product quality’ where the four parts provided a quality model, external metrics, internal metrics, and quality-in-use metrics, respectively.

In the introduction of ISO/IEC 9126-1, it states:

> “The software product quality characteristics defined in this part of ISO/IEC 9126 can be used to specify both functional and non-functional customer and user requirements.”

This is the only occurrence of the term “non-functional requirement” in [3]. However, note that ISO/IEC/IEEE 29148:2011(E) [2] does not use this term.

ISO 9126 defined six quality characteristics and described a software product evaluation process model.

2.1.1 History of characterizing quality

The following excerpts from Annex C of ISO 9126-1 [3] summarize the history of evolving the concept of quality up to the year 2001:

1. To evaluate the quality of a product through some quantitative means, a set of quality characteristics that describe the product and form the basis for the evaluation is required. This part of ISO/IEC 9126 defines these quality characteristics for software products.

2. The state of the art in software technology does not yet present a well-established and widely accepted description scheme for assessing the quality of software products.
3. Much work has been done since about 1976 by a number of individuals to define a software quality framework. Models by McCall, Boehm, the US Air Force, and others have been adopted and enhanced over the years.
4. … the need for one standard model came about … It is for this reason that the ISO/IEC JTC1 began to develop the required consensus and encourage standardization world-wide. … First considerations originated in 1978, and in 1985 the development of ISO/IEC 9126 was started. … it was decided that the best chance for establishing an International Standard was to stipulate a set of characteristics based on a definition of quality that was subsequently used in ISO 84021. This definition is accepted for all kinds of products and services. It starts with the user's needs.

Various scholars have traced the history of prior work in formulating software quality models; for example, see [4][5][6].

2.1.2 Characterization of attributes in ISO 9126-1

Section 7 of [3] organizes attributes of "quality in use" into the following four characteristics:

1. Effectiveness
2. Productivity
3. Safety
4. Satisfaction

Section 6 of [3] categorizes attributes of "external and internal quality" into the following six characteristics:

1. Functionality
2. Reliability
3. Usability
4. Efficiency
5. Maintainability
6. Portability

Furthermore, it lists Security, Suitability, and Interoperability as sub-characteristics of Functionality.

Section 6.4 of [3] states:

> "Clauses 6 and 7 define a hierarchical quality model (although other ways of categorizing quality may be more appropriate in particular circumstances)."

We agree that "other ways of categorizing quality may be more appropriate in particular circumstances," e.g., for NPP digital safety systems, as discussed next.

2.1.3 Some observations and conclusions about information from ISO 9126

The following observations and conclusions focus on the information from ISO 9126 that is used to evolve the ISO 25000 series of standards:

1. Different placement of Safety and Security does not support their integrated evaluation well. In an NPP safety system, a security breach could become a safety hazard. The model should reflect the contribution path.
2. In ISO 9126, "Analyzability" and "Testability" are sub-characteristics of "Maintainability." However, these characteristics are important in their own right (i.e., needed for an NPP safety system, even if the system was not required to be maintainable) and should not be buried under Maintainability.

[1] "Quality management and quality assurance - vocabulary"' withdrawn in 1994; superseded by ISO 9000.

3. The definition of analyzability in ISO 9126 (The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified) is not suitable for use in the safety analysis context, where the meaning of analysis is much broader. It covers analysis during system development. For example, the scope of the term includes preliminary hazard analysis and analysis to support verification and validation.
4. "Testabilility" is too limited in scope. "Verifiability" is the more comprehensive and more suitable term, including analysis of intermediate work products during system development, performed well before any testable work product is available.
5. From these observations, we conclude that the ISO 9126 quality model is not suitable for the domain of NPP safety systems. A domain-specific model should be devised, using similar foundational concepts.
    5.1. For NPP digital safety systems, where the safety function is the primary function, SAFETY should be a top-level attribute (also known as intrinsic property or characteristic) associated with the function, as well as, with the safety system.
    5.2. Appropriate structure of sub-characteristics should be developed.
    5.3. From those sub-characteristics, system-specific conditions and constraints should be derived, such that the leaf-node level constraint is measurable [7].

3. Current state of knowledge about quality requirements

In this section, we review two international standardization efforts, ISO 29148 [2] and the ISO 25000 family [8][9][10] and three streams of research and development activities [11][12][13]0[16].

3.1 Review of ISO 29148

The introduction in ISO 29148 "Systems and software engineering – Lifecycle processes - Requirements engineering" [2] states that:

> "This International Standard provides a unified treatment of the processes and products involved in engineering requirements throughout the life cycle of systems and software. This International Standard is the result of harmonization of the following sources (later we examine the claims of "unified treatment" and "harmonization"):
>
> - ISO/IEC 12207:2008 (IEEE Std 12207-2008), *Systems and software engineering — Software life cycle processes*
> - ISO/IEC 15288:2008 (IEEE Std 15288-2008), Systems and software engineering — System life cycle processes
> - ISO/IEC/IEEE 15289:2011, Systems and software engineering — Content of life-cycle information products (documentation)
> - ISO/IEC TR 19759, Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK)
> - IEEE Std 830, IEEE Recommended Practice for Software Requirements Specifications
> - IEEE Std 1233, IEEE Guide for Developing System Requirements Specifications
> - IEEE Std 1362, IEEE Guide for Information Technology — System Definition — Concept of Operations (ConOps) Document
> - ISO/IEC TR 24748-1, Systems and software engineering — Life cycle management — Part 1: Guide for life cycle management
> - ISO/IEC/IEEE 24765, Systems and software engineering — Vocabulary"

ISO 29148 provides guidance for the execution of the ISO/IEC 15288 [18] and ISO/IEC 12207 [19] processes that deal with requirements engineering (§2.1 in [2]).

The scope includes the following task relevant to the context of an NPP safety system, “Elicit stakeholder requirements from the identified stakeholders” (§6.2.3.1 in [2]). A note therein provides the following explanation:

> Stakeholder requirements describe the needs, wants, desires, expectations and perceived constraints of identified stakeholders. They are expressed in terms of a model that may be textual or formal, that concentrates on system purpose and behaviour, and that is described in the context of the operational environment and conditions. A product quality model and quality requirements, such as found in ISO/IEC 9126-1 [3] and ISO/IEC 25030 [10], may be useful for aiding this activity. Stakeholder requirements include the needs and requirements imposed by society, the constraints imposed by an acquiring organization and the capabilities and operational characteristics of users and operator staff. …

For a critical system, requirements and constraints should also be influenced through hazard analysis. For example, for an NPP safety system, both functional and quality requirements should flow from hazard analysis. However, this connection is not recognized in [2], except for general statements such as “There may also be safety or other regulatory constraints that drive system requirements.”

The scope of [2] includes the following related task, “Define technical and quality in use measures that enable the assessment of technical achievement” (§6.2.3.4 in [2]). A note therein provides the following explanation:

> This includes defining critical performance parameters associated with each effectiveness measure identified in the stakeholder requirements. The critical performance measures are analyzed and reviewed to ensure stakeholder requirements are met …. The ISO/IEC 9126 series of standards provides relevant quality measures.

This review leads to the conclusion that ISO 29148 does not provide a model for establishing quality requirements, but refers to ISO 9126. Since [2] refers to [10], recognizing the existence of the ISO 25000 family, we shift focus from ISO 9126 to the successor ISO 25000 family next.

3.2 Review of ISO 25000 family of standards

This review introduces the ISO 25000 series of standards through excerpts from [8][9][10], identifying differences from ISO 9126, and deriving observations and conclusions about a quality model suitable for NPP safety systems.

The “software quality requirements and evaluation” (SQuaRE) series of International Standards consists of the following divisions:

- Quality Management Division (ISO/IEC 2500n),
- Quality Model Division (ISO/IEC 2501n),
- Quality Measurement Division (ISO/IEC 2502n),
- Quality Requirements Division (ISO/IEC 2503n),
- Quality Evaluation Division (ISO/IEC 2504n),
- SQuaRE Extension Division (ISO/IEC 25050 – ISO/IEC 25099).

This International Standard revises ISO/IEC 9126-1:2001, and incorporates the same software quality characteristics with some amendments, e.g.:

- The scope of the quality models has been extended to include computer systems, and quality in use from a system perspective.
- When appropriate, generic definitions have been adopted, rather than using software-specific definitions.

- Context coverage has been added as a quality in use characteristic, with sub-characteristics context completeness and flexibility.
- Security has been added as a characteristic, rather than a sub-characteristic of functionality, with sub-characteristics confidentiality, integrity, non-repudiation, accountability and authenticity.
- The internal and external quality models have been combined as the product quality model.

For an NPP safety system, SAFETY is a primary property, but the model in the SQuaRE series does not provide a primary status to SAFETY.

ISO/IEC 25010 [9] cancels and replaces ISO/IEC 9126-1:2001, which has been technically revised. This International Standard is derived from ISO/IEC 9126:1991, *Software engineering — Product quality*. ISO/IEC 9126:1991 was replaced by two related multipart standards: ISO/IEC 9126, *Software engineering — Product quality* and ISO/IEC 14598, *Software engineering — Product evaluation*.

The following excerpt shows the relevance of a quality model in the context of safety:

> "Realization of goals and objectives for human safety relies on high-quality software and systems."

ISO/IEC 25010 states:

> "Prior to software development or acquisition, quality requirements should be defined from the perspective of stakeholders. Analysis of the in use requirements will result in derived functional and quality requirements needed for a product to achieve the in use requirements."

Note that the standard uses the term, "quality requirements" – not non-functional requirements! The terms "nonfunctional" and "non-functional" do not appear anywhere in the ISO 25000 series of standards. Users of standards should follow suit, stop using the term "non-functional requirement" and use terms such as "quality requirement" or "requirements for quality properties."

3.2.1 Intended uses of ISO/IEC 25010

This International Standard is intended to be used in conjunction with the other parts of the SQuaRE series of International Standards (ISO/IEC 25000 to ISO/IEC 25099).

The quality models in this International Standard can be used in conjunction with ISO/IEC 12207 [19] and ISO/IEC 15288 [18], particularly the processes associated with requirements definition, verification and validation with a specific focus on the specification and evaluation of quality requirements. ISO/IEC 25030 [10] describes how the quality models can be used for software quality requirements, and ISO/IEC 25040 [20] describes how the quality models can be used for the software quality evaluation process. ISO/IEC 25041:2012 [21] is an evaluation guide.

ISO/IEC 25010 states:

This International Standard can also be used in conjunction with ISO/IEC 15504 (which is concerned with software process assessment) to provide:

- a framework for software product quality definition in the customer-supplier process;
- support for review, verification & validation,
- support for a framework for quantitative quality evaluation, in the support process;
- support for setting organizational quality goals in the management process.

Further, ISO/IEC 25010 states:

Activities during product development that can benefit from the use of the quality models include:

- identifying software and system requirements;
- validating the comprehensiveness of a requirements definition;
- identifying software and system design objectives;
- identifying software and system testing objectives;
- identifying quality control criteria as part of quality assurance;
- identifying acceptance criteria for a software product and/or software-intensive computer system;
- establishing measures of quality characteristics in support of these activities.

3.2.2 Some foundational definitions in ISO/IEC 25010

Basic concepts of the quality model in [9] are introduced through definitions of foundational terms, given below. Where a definition includes the word “software” as a qualifier, the definition may be generalized to a system level by dropping the qualifier “software.”

Then, for example, “quality” is defined as “degree to which a software product satisfies stated and implied needs when used under specified conditions.” Note that this definition differs from the meaning in common usage ““degree to which a software product satisfies its requirements.”

A “quality model” is a “defined set of characteristics, and of relationships between them, which provides a framework for specifying quality requirements and evaluating quality” where “quality characteristic” is defined as “category of quality attributes that bears on quality”; quality characteristics can be refined into multiple levels of sub-characteristics and finally into software quality attributes. An “attribute” is defined as an “inherent property or characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means” where a “property” is a “measurable component of quality.” Note that an “inherent property” is a permanent characteristic, in contrast with an assigned characteristic. An inherent property may be:

1. Functional property: It determines what the software is able to do.
2. Quality property: It determines how well the software performs.

Then, a “quality requirement” is a “requirement that a software quality attribute be present in software.”

The gap between needs and requirements (especially for the SAFETY property) is significant in the evaluation of a safety critical system. The ISO 25000 family of standards addresses this gap through the concept “quality in use” defined as “the degree to which a product or system can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency, freedom from risk and satisfaction in specific contexts of use.”

The standard enwraps the concept “SAFETY” in “freedom from risk” rather than “the expectation that a system does not, under defined conditions, lead to a state in which human life, health, property, or the environment is endangered” (3.2622 in [1]). The standard does not treat SECURITY in a manner comparable to SAFETY. It characterizes security as information security. The standard also inherits the issues with ISO 9126-1 identified in Section 2.1.3. Thus, from these observations, again we conclude that the ISO 25010 quality model is not suitable for the domain of NPP safety systems. A domain-specific model should be devised, using similar foundational concepts.

4. Ongoing research and development activities

In this concluding section, we report known ongoing R&D efforts to characterize quality of a software-dependent system or its elements.

4.1 Related R&D at the University of Southern California

Barry Boehm through the Systems Engineering Research Center [11] is extending his prior work (which various scholars have reported in their surveys, e.g., in [4][5][6]). The current direction of research links a large range of quality attributes into a trade-off space to assist in design decisions.

4.2 Related R&D at the Software Research Institute (SEI)

The Software Engineering Institute (SEI) has a long-standing ongoing effort in quality attributes [12] and using these to derive architectural constraints [13]. Recent work [14][15] shows a framework (Figure 1) aligned with the ISO 25000 series of standards [8][9][10], explained as follows:

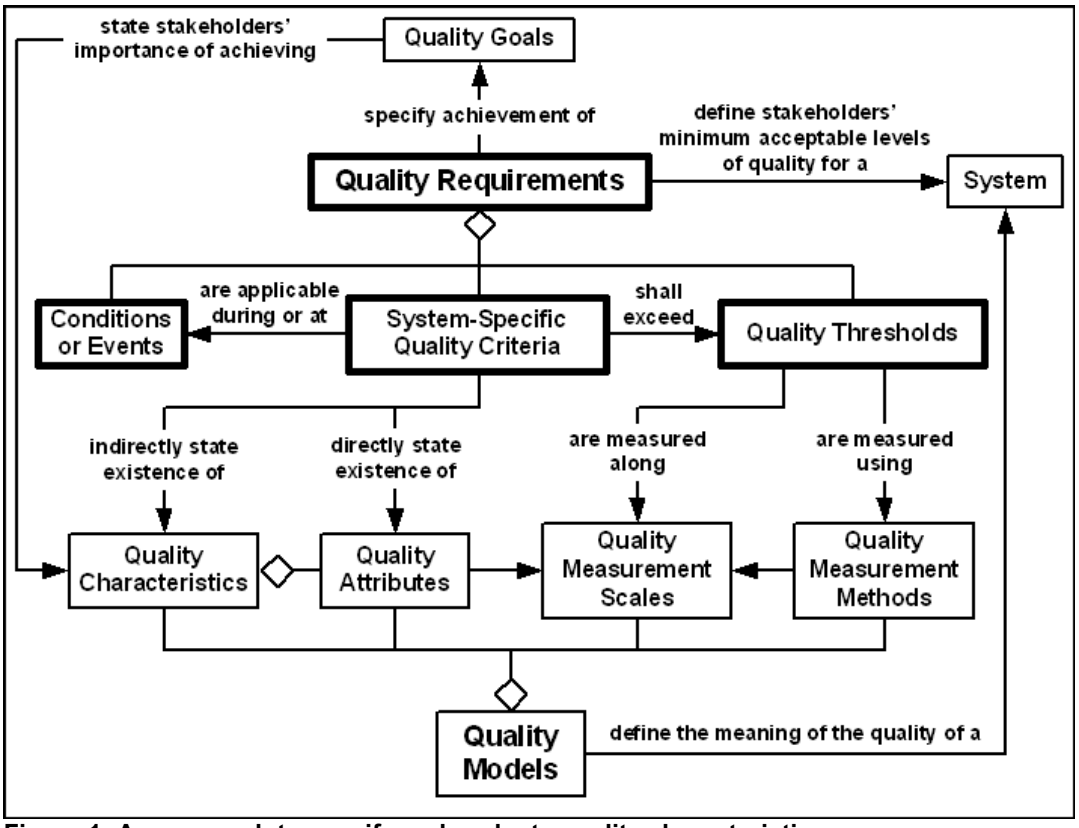

Figure 1: An approach to specify and evaluate quality characteristics

1. A quality model defines the meaning of the quality of a system. The same model is applied to the constituent elements of the system at every level of integration, as identified in the architecture.
2. A quality model is defined in terms of a set of quality characteristics (see Figure 1).**Error! Reference source not found.**
3. A quality characteristic is defined in terms of a set of quality attributes.
4. An attribute may be defined in terms of other attributes (i.e., it may be a composite or elemental).

5. A finest-grained (or elemental) attribute is measured along a measurement scale, using a quality measurement method.
6. The method is defined in the context of the scale.
7. For each quality requirement, a set of system-specific quality criteria are defined in terms of the respective quality attributes and corresponding thresholds.
8. Different thresholds may be established for the same attribute under different conditions or upon the occurrence of different events. (This may also be viewed as state-dependent thresholds, if the system behavior is modeled in a finite state machine paradigm).

4.3 A direction of research at the NRC

One direction of R&D being explored at the NRC [11] addresses the question, "What would be the most streamlined quality model to support the safety evaluation of a digital safety system (the automation portion in the context of its environment) for an NPP?" In other words, it explores the decomposition of the SAFETY property and to derive constraints on the system.

Figure 2 shows quality requirements associated with functional requirements. Examples of top-level quality requirements include Safety and Security.

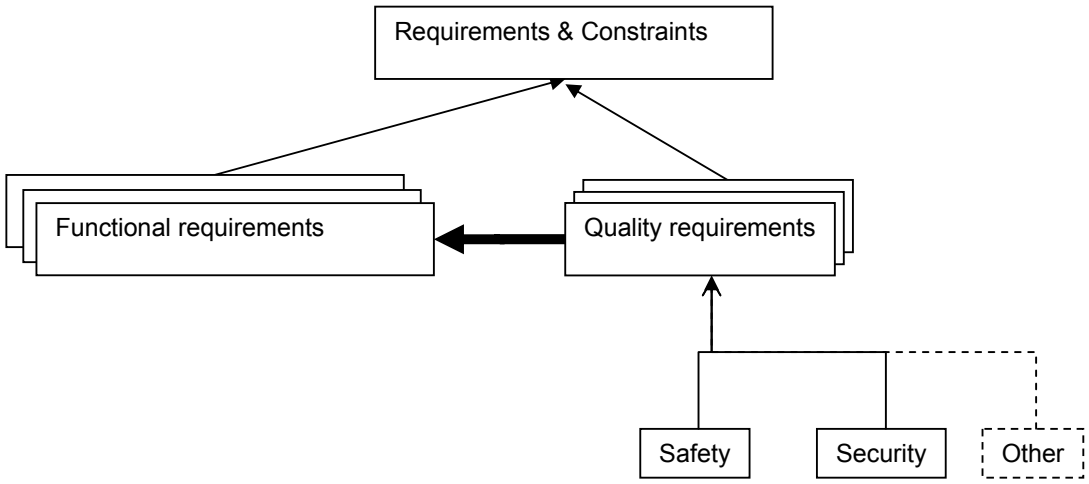

Figure 2: Quality requirements should be explicit

For a safety system, as shown in Figure 3, the "Assurability" property distinguishes it from systems that do not require similar assurance. Figure 3 also shows other quality attributes that contribute to or support "Assurability." The corresponding quality requirements may also be viewed as constraints to be satisfied by the digital safety system, that is, constraints on the solution space (also known as design space), such that system concepts that do not satisfy these constraints are eliminated from further consideration (i.e., the hazard space is reduced). Table 1 shows the logical derivation of further constraints (from those shown in Figure 3) to support the "Assurability" property. Following is an informal expression of the reasoning; the symbols are hyperlinked to entries in Table 1:

1. To be able to assure that a system is safe, one must be able to verify [H-S-1] that it meets all its safety requirements.
2. For a system to be verifiable, it must not be possible for one element of the system to interfere with another. [16]
3. If the conceived system is too complex, adequate verification is infeasible. [H-S-1.1]

4. If one cannot even understand it, how can one assure that it is safe? [H-S-2]
5. Verifiability is a required system property, flowing down from the system to its elements (constituents) progressing to the finest-grained element; it implies corresponding verifiable specifications. Verification also includes analysis at various phases in the development lifecycle, well before an artifact is available for physical testing. Examples of conditions for verifiability:
   5.1. Ability to create a test (or verification) case to verify the requirement.
   5.2. Observability
   5.3. Ability to constrain the environment of the object of verification.
6. For "analyzability" the system must have predictable and deterministic behavior (i.e., it must yield deterministic results). [H-S-1.2].

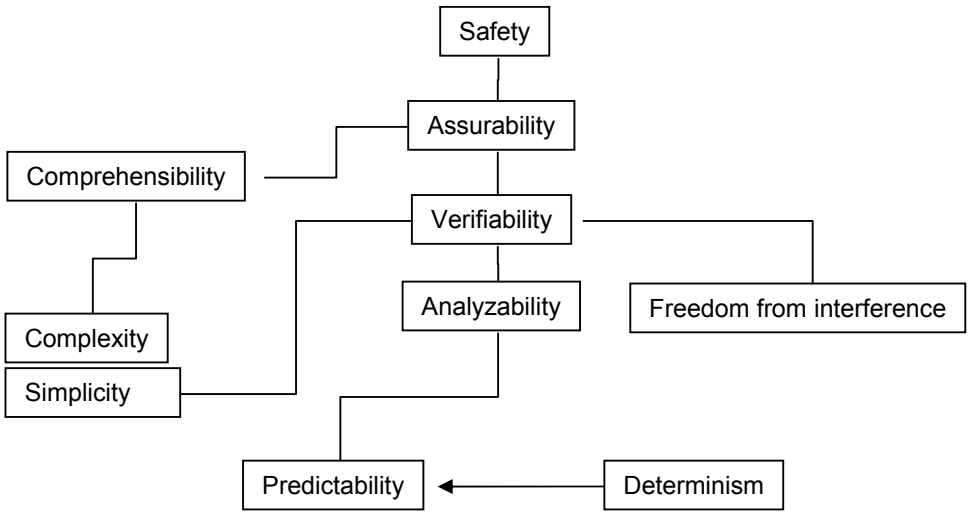

Figure 3: Quality characteristics to support safety

Table 1: Constraints derived from quality attributes: Scenario-based examples

Contributory hazard		**Conditions that reduce the hazard space**	
ID H-S-	**Generalized Scenario**	**ID H-S-**	**Description**
1	The system is not sufficiently verifiable and understandable, but this deficiency is discovered too late. Appropriate considerations and criteria are not formulated at the beginning of the development lifecycle; therefore, corresponding architectural constraints are not formalized and checked. When work products are available for testing, it is discovered that adequate testing is not feasible (e.g., the duration, effort, and cost are beyond the project limitations).	1G1	Verifiability is a required system property, flowing down from the system to its constituents progressing to the finest-grained element. [H-S-1.1G1↓]
		1G1.1	Verifiability of a work product is checked at every phase of the development lifecycle, at every level of integration, before proceeding further in the development. Examples of conditions for verifiability: Ability to create a test (or verification) case to verify the requirement; Observability; Ability to constrain the environment of the object of verification.
1.1	System is not verifiable (e.g., it is not analyzable or very difficult to analyze).	1.1G1	Avoidance of unnecessary complexity.
		1.1G1.1	The behaviour is unambiguously

Contributory hazard		Conditions that reduce the hazard space	
ID H-S-	**Generalized Scenario**	**ID H-S-**	**Description**
			specified for every combination of inputs (including unexpected inputs) at every level of integration in the system.
		1.1G1.2	The flow-down ensures that 1. Allocated behaviors satisfy the behavior specified at the next higher level of integration; 2. Unspecified behavior does not occur.
		1.1G1.3	The behaviour of the system is a composition of the behaviours of its elements, such that when all the elements are verified individually, their compositions may also be considered verified. This property is satisfied at each level of integration, flowing down to the finest-grained element in the system.
		1.1G1.4	Development follows a refinement process.
1.1.1	There are unanalyzed or un-analyzable conditions. For example, all system states, including unwanted ones such as fault states, are not known and not explicit. To that extent, verification and validation (V&V) of the system is infeasible. [H-S-1.1↑]	1.1.1G1	Static analyzability: System is statically analyzable. 1. All states, including fault conditions, are known. 2. All fault states, leading to failure modes, are known. 3. Safe state space of the system is known.
1.1.2	There is inadequate evidence of verifiability. [H-S-1.1↑]	1.1.2G1	Verification plan shows the coverage needed for safety assurance.
1.2	System behaviour is not deterministic (does not yield deterministic results). [H-S-1.1.1↑]	1.2G1	System has a defined initial state.
		1.2G2	System is always in a known configuration.
		1.2G3	System is in a known state at all times (e.g., through positive monitoring and indication): 1. Initiation of function 2. Completion of function 3. Intermediate state, where needed to maintain safe state in case of malfunction.
1.3	System behaviour is not predictable. [H-S-1.1.1↑]	1.3G1	Each transition from a current state (including initial state) to some next state is specified and known, including transitions corresponding to unexpected combination of inputs and transition conditions.
		1.3G2	A hazardous condition can be detected in time to maintain the system in a safe state.
2	Comprehensibility: System behaviour is not understood completely and correctly by its community of users (e.g., reviewers, architects, designers, and implementers), that is, the people	2G1	Behaviour is completely and explicitly specified.
		2G2	Behaviour is completely understandable.

Contributory hazard		Conditions that reduce the hazard space	
ID H-S-	Generalized Scenario	ID H-S-	Description
	and the tools they use. [H-S-1↑]	2G3	Behaviour is understood completely, correctly, consistently, and unambiguously by different users interacting with the system
		2G4	The allocation of requirements to some function and that function to some element of the system is bi-directionally traceable.
		2G5	The behaviour specification avoids mode confusion, esp. when functionality is nested.
		2G6	The architecture is specified in a manner (e.g., language; structure) that is unambiguously comprehensible to the community of its users (e.g., reviewers, architects, designers, implementers), that is, the people and the tools they use.

Considering that the state of practice is especially weak in the derivation of verifiable constraints from quality requirements, a careful review is needed. The architecture should satisfy these constraints, starting from the system concept phase and continuing at every successive phase of development, refinement and decomposition, including all phases of the software development lifecycle. An approach to organize information about the uncertainties in such an evaluation is discussed next.

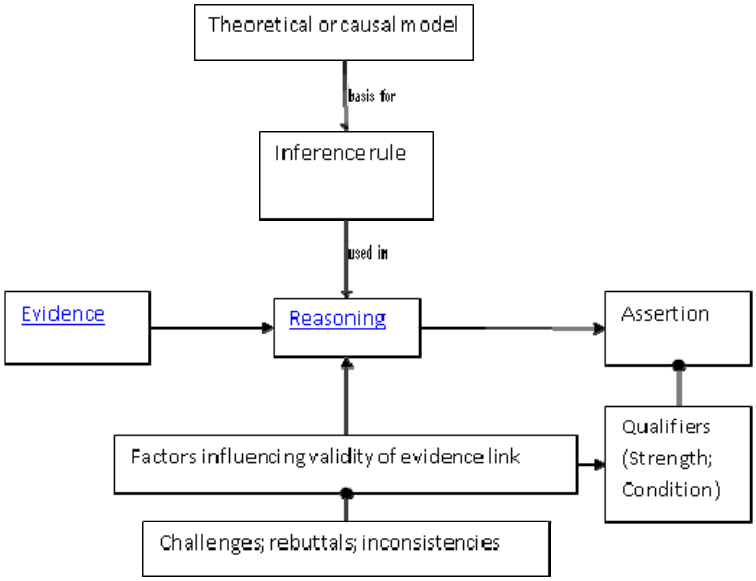

Figure 4: Structure to reason about the satisfaction of a constraint

Figure 4 depicts a structure for reasoning (adapted from [22]) to organize and reason about uncertainties encountered during the evaluation of the satisfaction of a constraint. Suppose that the evaluation team is considering an assertion that the result of their work (e.g., constraint on the object being analyzed) is satisfied. Then, the team clarifies its reasoning through discussion,

evoking challenges to the assertion and rebuttals to the challenges. The discussion may also reveal inconsistencies in the reasoning. In this manner, the team identifies factors affecting the validity of their assertion. Qualifiers are associated with the assertion; for example:

1. Condition(s) under which the assertion is supported.
   1.1. Uncertainties may be stated as assumptions, for which the truth has to be validated.
   1.2. Changes needed may be stated as constraints to be satisfied.
2. Degree or strength of the assertion: {Strong …. Weak}

The results are recorded, showing how the assertion is supported by the evidence, identifying the inference rule to assert the evidence-assertion link, and the technical basis for the rule such as a causal model.

5. Future directions to support independent safety evaluation

To support effective and efficient independent safety evaluation of a digital safety system for an NPP, it is attractive to align practices in the nuclear industry with mainstream standards for systems and software engineering, such as the ISO 25000 series. However, their quality model is too general (entailing much more work than necessary; diluting the focus on safety) and not well-aligned for a streamlined safety evaluation of a nuclear safety system. It is attractive to explore an NPP domain-specific quality model as discussed in Section 4.3. The supporting characteristics should be further decomposed to the granularity needed for verifiability, such as those introduced in Table 1. Appropriate measurement scales and methods should be explored [7]. An approach to organize information about uncertainty and reason about it should be developed such as one depicted in Figure 4.

6. References

[1] ISO/IEC/IEEE 24765 Systems and software engineering – vocabulary, 2010

[2] ISO/IEC/IEEE 29148:2011(E), Systems and software engineering – Lifecycle processes - Requirements engineering

[3] ISO/IEC 9126-1:2001), Software engineering — Product quality — Part 1: Quality model

[4] URL: http://airccse.org/journal/ijsea/papers/4113ijsea01.pdf

[5] URL: http://www.bth.se/com/besq.nsf/(WebFiles)/BFEFBED4E650D690C125706900324572/$FILE/chapter_4.pdf

[6] URL: http://smitaiddal87.wordpress.com/2013/01/27/9/

[7] Roberts, F. Measurement Theory with Applications to Decision Making, Utility, and the Social Sciences, Addison-Wesley, 1979

[8] ISO/IEC 25000: 2005(E) Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE.

[9] ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

[10] ISO/IEC 25030:2007, Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Quality requirements

[11] URL: http://www.sercuarc.org/app/webroot/uploads/files/7_Tradespace_Boehm.pdf

[12] URL: http://www.sei.cmu.edu/reports/95tr021.pdf

[13] URL: http://resources.sei.cmu.edu/asset_files/TechnicalReport/1994_005_001_16301.pdf

[14] Donald G. Firesmith, Engineering Safety- and Security-Related Requirements for Complex Systems, at the Kärnteknik-2011 *(Nuclear Technology 2011) Nordic Symposium* on 8 December 2011.

[15] URL: http://www.amazon.com/Method-Framework-Engineering-System-Architectures/dp/1420085751

[16] U.S. Nuclear Regulatory Commission, "Research Information Letter 1101: Technical basis to review hazard analysis of digital safety systems, URL: http://cps-vo.org/node/8758.

[17] ISO/IEC 15939:2007(E) Systems and software engineering – Measurement process

[18] ISO/IEC 15288:2008 (IEEE Std 15288-2008), Systems and software engineering — System life cycle processes

[19] ISO/IEC 12207:2008(E) (IEEE Std 12207-2008), System and software engineering – Software life cycle processes

[20] ISO/IEC 25040:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Evaluation process

[21] ISO/IEC 25041:2012, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Evaluation guide for developers, acquirers and independent evaluators

[22] Toulmin, Stephen. The Uses of Argument. Cambridge: University Press, 1958