KAERI/TR-1871/2001

기술보고서

차세대 원자로 디지털 계측제어 소프트웨어 요구명세 평가절차서

(Evaluation Procedure of Software Requirements Specification for Digital I&C of KNGR)

KAERI

2001년 6월

한국원자력연구소

제 출 문

한국원자력연구소장 귀하

이 보고서를 2000년도 "차세대 원자로 설계관련 요소기술 개발" 과제의 기술보고서로 제출합니다.

제 목: 차세대 원자로 디지털 계측제어 소프트웨어 요구명세 평가절차서

2001년 6 월

과 제 명	: 차세대 원자로 설계관련 요소기술 개발
주 저 자	: 이 장 수
공 동 저 자	: 박 종 균, 이 기 영,
	김 장 열, 천 세 우

요 약

원전 디지털 계측제어계통을 개발할 때 소프트웨어 요구명세의 정확도, 완전성과 안전성 보장은 최종 시스템 개발성공과 인허가 획득에 아주 중요한 요소가 된다. 원전 계측제어계통의 디지털화 추세에 따라 원자력산업의 특수성인 안전성 확보와 컴퓨터 소프트웨어 안전성 심사 및 개발기준이 되는 국제표준 정립을 위해 국제원자력기구 (IAEA), 국제전기기술위원회 (IEC), 국제전기전자공학회 (IEEE) 등에서 표준화 노력을 기울이고 있으며 현재 한국원자력연구소에서는 소프트웨어 공통모드고장 문제에 대한 대책의 일환으로 일련의 필수안전 소프트웨어 평가 방법론들을 개발하고 있다. 본 보고서에서는 새롭게 개정되는 국제표준의 요건과 한국 원자력 안전기술원의 차세대 원자로 안전 규제지침을 만족하고 소프트웨어 공통모드고장의 대책이 될 수 있는 차세대 원자로 디지털 계측제어 소프트웨어 요구명세 평가 절차를 개발하였다. 현재 원자로 안전 규제지침에서는 미국 원자력 안전규제 위원회 (NRC)의 규제지침 (Reg. Guide) 1.172에 따라 원전 안전 소프트웨어 요구명세 관련 기준을 제시하고 있다. 이 보고서는 차세대 원전 안전 소프트웨어 요구명세를 평가 할 때 상위법과 표준들에서의 요건의 만족여부를 평가할 수 있도록 인도하는 지침서이다. 이 지침서 1장에서는 원전 소프트웨어 요구명세 공학에 대한 소개와 정형적 요구 명세 기법과 요구명세 안전성 분석 기법 등을 종합 정리하였다. 2장에서 차세대 원전 소프트웨어 요구명세를 평가하기 위해 개발한 평가항목 별 평가 절차와 요구명세 단계에서의 안전성 분석을 위한 평가 절차를 기술하였다.

Abstract

The accuracy of the specification of requirements of a digital system is of prime importance to the acceptance and success of the system. The development, use, and regulation of computer systems in nuclear reactor Instrumentation and Control (I&C) systems to enhance reliability and safety is a complex issue. This report is one of a series of reports from the Korean Next Generation Reactor (KNGR) Software Safety Verification and Validation (SSVV) Task, Korea Atomic Energy Research Institute, which investigates different aspects of computer software in reactor I&C systems, and describes the engineering procedures for developing such a software. The purpose of this guideline is to give the software safety evaluator the trail map between the code & standards layer and the design methodology & documents layer for the software important to safety in nuclear power plants. Recently, the requirements specification of safety-critical software systems and safety analysis of them are being recognized as one of the important issues in the software life cycle, and being developed new regulatory positions and standards by the regulatory and the standardization organizations such as IAEA, IEC, NRC, and IEEE. For example, NRC endorsed the IEEE standard 830 by Regulatory Guide 1.172, Software Requirements Specifications for Digital Computer Software Used in Safety Systems of Nuclear Power Plants. We presented the procedure for evaluating the software requirements specifications of the KNGR protection systems. We believe it can be useful for both licenser and licensee to conduct an evaluation of the safety in the requirements phase of developing the software. The guideline consists of the requirements engineering for software of KNGR protection systems in chapter 1, the evaluation checklist of software requirements specification in chapter2.3, and the safety evaluation procedure of KNGR software requirements specification in chapter 2.4.

Table of Contents

List of Figures

List of Tables

1. Software Requirements Engineering

1.1 Introduction

The accuracy of the specification of requirements of a digital system is of prime importance to the acceptance and success of the system. It has been shown that a significant portion of the problems with software-based systems can be traced back to incorrect or incomplete requirements specification. When the software-based system is a replacement for an existing analogue system, it is a mistake to just take the requirements specification for the old system and use it for the new digital system, since the latter system has different characteristics to the former. The analogue system requirements specification can be used as a starting point, but the digital characteristics must be taken into account when preparing the new requirements specification.

It is important to make sure that the requirements specification is accurate and complete. Software-based equipment brings with it unique opportunities and unique concerns compared to analogue equipment. Therefore, it is important to make sure that these differences are taken into account when the requirements specification is developed.

The requirements specification needs to completely define the functions of the system and the qualification requirements, including acceptance criteria. The requirements specification must also address system issues including defining the external and internal interfaces and hardware-software interactions.

We have developed an improved process for evaluating the requirements specification to assure correctness and completeness. This includes making sure that the coverage of potential abnormal condition and events is adequate and properly addressed in the requirements specification.

1.1.1 Classification

Requirements can be classified from several kinds of viewpoints. The good classification of requirements can reduce the implementation and assessment cost of the software-based I&C systems. That is, the requirements must be well classified by the criticality level of the target system, by system structures, by subject areas, and by whom required them. First of all, by classification of criticality level, the requirements can be classified as follows:

- Requirements for software-based safety I&C systems
  - ✓ Requirements for new software-based I&C systems
  - ✓ Requirements for upgrades using software-based I&C systems
- Requirements for software-based non-safety I&C systems
  - ✓ Requirements for new software-based I&C systems
  - ✓ Requirements for upgrades using software-based I&C systems

This classification should follow the general classification scheme. The important factor to be cost-effective when implementing and assessing the requirements is the complete separation of these two requirements.

Second, in general, the requirements can be classified by the system structures which consist of the software-based I&C system, as follows:

- System functional requirements
- Hardware requirements
- Software requirements
- Human factors(HMI) requirements
- Interface requirements among above requirements

Third, by subject areas, as follows:

- Safety requirements
- Reliability requirements
- Availability requirements
- Level of automation requirements
- Security requirements
- Timing requirements
- Environmental requirements (EMI, RFI, Seismic)

Last, by who's requirements, as follows:

- Utility requirements (e.g. EPRI URD)
- Regulatory requirements (Member country's)
- IAEA/IEC requirements

1.1.2 Good requirements

Characteristics of good requirements should be a) Correct, b) Unambiguous, c) Complete, d) Consistent, e) Ranked for importance and/or stability, f) Verifiable, g) Modifiable, h) Traceable.

To the utilities, independent assessors, and regulators, a good SRS should provide several specific benefits, such as the following:

- ✓ Establish the basis for agreement between the customers and the suppliers on what the software product is to do.
- ✓ Reduce the development effort.
- ✓ Provide a basis for estimating costs and schedules.
- ✓ Provide a baseline for validation and verification.
- ✓ Serve as a basis for enhancement.

1.1.3 Formal methods and tool support for requirements elicitation, specification, and validation

A good practical formal method and its tool support can be a direct solution for the cost problem of the software-based I&C systems. But, according to my limited knowledge, I have not found such a practical and technically sound formal method for specifying the requirements of a safety-critical I&C systems. Under this topic, this chapter will cover the assessment of the existing formal methods and their available tools. There are many controversial factors in this topic, and so we will follow the definitions and requirements on formal methods in IEC 880 supplement 1.

The objective of the section on formal methods is to discuss the impact the use of formal methods, in particular in the requirement phase, have on the assessment of a system. It is claimed that formal methods promote the finding of errors, inconsistencies and missing elements in the requirements. They also facilitate the understanding of the functioning of a planned system at an early stage. The problem which will be discussed in the chapter on formal methods is how the use of formal methods in an actual system specification will influence the assessment of the system. E.g. to which degree can one state that the use of formal method will increase the safety integrity level of a system, as claimed in certain standards an guidelines. A potential advantage of using formal specification methods is that they are often facilitated with tool support, and an evaluation of such tools should also be included in the assessment process.

1.2 Requirements Engineering for Software-based I&C Systems of NPP

Software-based I&C systems of nuclear power plants must deal with continuous state changes as well as discrete and instantaneous state changes. Most of process control systems are hybrid systems where digital devices, usually modeled by finite automata, must sense the changes in analog physical phenomena, which is typically modeled by differential equations. It is the interaction of continuous and discrete changes that make hybrid systems interesting and nontrivial targets for formal analysis. Majority of hybrid systems also possesses real-time and safety-critical characteristics, as is the case for a computer-controlled safety system for a nuclear power plant. We refer to such systems as hybrid real-time safety system or HRTS in short. Safety is a system property, and the requirements should describe the program behavior as a relation between entities in the system's environment. Some of these entities are continuous and others are discrete, and an ideal safety analysis technique must be able to validate the logical contribution of the software elements to the physical hazard.

There are many problems associated with requirements engineering for HRTS, including problems in completeness, expressiveness, and analyzability. These problems may lead to poor specifications and the development of a system that is later judged unsatisfactory or unacceptable.

Requirements engineering can be decomposed into the iterative activities of requirements elicitation, specification, and validation. Most of the requirements techniques and tools today focus on the specification, that is, the representation of the requirements. This report concentrates instead on a conceptual specification that can be located between an elicitation and a quantitative specification, and a safety analysis by causality information that is produced from the conceptual specification. This safety analysis is one of the validation activities.

Requirements engineering is a key problem area in the development of complex HRTS. The development and modification of requirements specifications has long been identified as an important and difficult part of software development. Many of the problems in creating and refining a system can be traced back to elicitation issues [Chri92]. Research efforts for software engineering should be directed towards methods and tools needed to improve the requirements analysis process, and, in particular, to those providing more support to the elicitation of requirements. The list of ten elicitation problems given in one source [McDe89] could be classified according to this framework as follows:

- Problems of scope
  - The boundary of the system is ill defined
  - Unnecessary design information may be given
- Problems of understanding
  - Users have incomplete understanding of their needs
  - Users have poor understanding of computer capabilities and limitations
  - Analysts have poor knowledge of problem domain
  - User and analyst speak different languages
  - Ease of omitting “obvious” information
  - Conflicting views of different users
  - Requirements are often vague and untestable
- Problems of volatility
  - Requirements evolve over time

1.3 Formal Methods for Requirements Engineering

Formal methods are perceived by the community as a way of increasing confidence in software for safety critical systems. They are mathematically based techniques, often supported by reasoning tools, that can offer a rigorous and effective way to model, design and analyze computer systems. Evidence shows that effective ways of using formal methods for safety critical systems are still an open problem [Leve90, Barr92, and Liu95]. This is because during the process of software development for safety critical systems, not only the functional behavior of software has to be considered carefully, but we must also demonstrate that the developed software satisfies the overall safety requirements.

It is possible to distinguish five types, or classes, of formal methods that can be roughly characterized as follows:

1. Model based approaches --- giving an explicit, albeit abstract, definition of system

(program) state and operations that transform the state.

2. Algebraic approaches --- giving an implicit definition of operations by relating the behavior of different operations without defining state.
3. Process algebra --- giving an explicit model of concurrent processes and representing behavior by means of constraints on allowable observable communication between the processes.
4. Logic based approaches --- a variety of approaches using logic to describe properties of systems, including low level specification of program behavior and specification of system timing behavior.
5. Net based approaches --- giving an implicitly concurrent model of the system in terms of (causal) data flow through a network, including representing conditions under which data can flow from one node in the net to another.

There are significant differences between the expressive power and analyzability of the methods. Formal methods can be used in two distinct ways. First, they can be used for production of specifications that are then used as the basis of a fairly conventional system development. Second, formal specifications can be produced as above, and then used as a basis against which the correctness of the program is verified.

Formal methods can assist people to do requirements analysis thoroughly and to express precise requirements specifications. Many formal notations and methods have been used so far in industry for the purpose of requirements analysis and functional specification of safety critical systems. One of the main objectives for requirements safety analysis is that requirements specification does not allow executions that would lead to catastrophic failure in its intended operational context.

Demonstrating to the complete satisfaction that a method has achieved this objective is generally accepted to be impossible [Leve86]. In essence the difficulty is that we do not have any way of knowing that we have identified all the possible threats to, or failure modes of, the system so we can never be sure that our specifications are complete. However it is possible to apply techniques that reduce the likelihood that the specification is catastrophically flawed

We cannot have complete confidence that we have achieved safety integrity. Instead we need to achieve assurance, or confidence. Assurance is based on a number of issues including the level of trust we have in the individuals carrying out the development, etc.

However, one of the main contributing factors to assurance is the evidence produced during software development, and this in turn derives from the verification and validation activities that we carry out throughout the software development process

Safety assurance can be thought of as confidence based on objective evidence. We can say that the greater is our comprehension of some artifact, the greater is our confidence about the dependability of the artifact.

The results of requirements analysis are the primary basis for communication with the user and customer. For this reason it is desirable that the representation should be as precise as possible, e.g. formal. It is also necessary that requirements be intelligible to the customers as one of the primary forms of validation.

The use of formal methods in the requirements phase has added the possibility of animation to the already noted advantages of unambiguity, completeness and consistency [Jaff88]. Notations become more complete, addressing not only functionality but also non-functional requirements such as timing. However, they have not yet been able to combine power with expressiveness and intuitiveness, and there is still a long way to go to make the notations presentable to the user without loss of precision.

1.3.1 Strengths of Formal Methods

- Precision: Formal specifications can be very precise definitions because the semantics of the notations are well defined. The direct benefit of the precision is that it reduces, or even eliminates, the risk of ambiguity and misinterpretation of specifications.
- Abstraction: Abstraction is one of primary intellectual tools for coping with complexity and it aids clarity by ‘drawing away from’ details that are not germane to our interests.
- Concise: Clarity also arises from conciseness. Formal notations can be much more compact than equally clear natural language descriptions whilst normally being more precise.
- Manipulability: There are well-defined rules for analyzing and perhaps transforming formal specifications. This manipulability property can be used to show consistency of specifications and to derive important consequences of specifications.

Errors due to misunderstandings are reduced. As formal specifications are unambiguous, communication between people involved in requirements analysis, specification construction, design and implementation via the formal specifications is enhanced. Therefore, errors due to misunderstandings are reduced.

Validation of requirements specifications becomes easier. Because of the precision of formal requirements specifications, every task specified can be precisely interpreted thus enhancing the clients' ability to scrutinize the correctness of formal requirements specifications.

1.3.2 Weaknesses of Formal Methods

- Clearly it is extremely valuable to remove doubts associated with software development but, unfortunately, most evidence suggests that the primary source of (significant) software errors is the specification and safety critical systems are more prone to this sort of problem [Leve86]. This means that the mathematics, of itself, is insufficient to assure safety. It should be noted that we can use proof techniques to assist in validation, e.g. by deriving safety properties from a specification, but this simply reduces the ‘gap’ between formalisms and the ‘real world’, and doesn’t eliminate it.
- Another major limitation is to do with interpretation of specifications. Formal specifications do not just have an interpretation in terms of the underlying mathematics, they are also interpreted by software engineers in terms of a computational model and by system users in terms of a model of the use of the system in its operational environment. The issue of ambiguity then becomes not one of the existences of a unique model for the specification in the underlying logic but of compatibility of interpretations made in different domains by individuals with differing backgrounds and knowledge.
- Formal specifications are difficult to read. The reasons for this limitation are twofold. The first is that the majorities of people working in the computing industry at present are accustomed to traditional informal methods and are not well trained in formal notations. The second reason is that mathematical notations are usually more difficult to understand than informal descriptions.
- Correctness proofs are resource-intensive. This is because considerable time is required to produce formal specifications. Furthermore, since there is intrinsic difficulty in performing correctness proofs automatically, proofs have to be done manually or interactively with machines, which is resource-intensive. Development costs increase. The main reason for this limitation is that many companies and projects need to invest more money for training their staff in formal methods technology.

- Formal specifications can still have errors. No formal method so far can provide automatic support to semantic consistency checking for formal specifications.
- Environments to support the use of formal methods are not available. Tools to support the use of some formal methods do exist. However, none of them are powerful enough to support the whole activity of using formal methods.

1.4 Related Works

1.4.1 Formal Methods

There are much kinds of formal notations, methods, and supporting tools for specifying and verifying the requirements of real-time systems. Most of them are methods for safety-critical, embedded, and reactive systems. There are pointers to information on formal methods available around the world on the World Wide Web (WWW).[1] Many researchers had conducted the comparison and evaluation of existing methods [Barr92, Liu95, and Stol98]. According to the survey, the main conclusion was that most of the existing formal methods were both over-sold and under-used, and the major reason not to used widely in industry was their complexity and difficulty. The mathematics of a formal verification is not sufficient to assure the safety of the physical system because the primary source of software error is the requirements specification itself. We focus our survey into the subset of formal methods, those for hybrid real-time safety system, in this report.

More and more real-life processes, from elevators to aircraft, are controlled by programs. These reactive programs are embedded in continuously changing environments and must react to environment changes in real time. Obviously, correctness is of vital importance for reactive programs. Yet traditional program verification methods allow us, at best, to approximate continuously changing environments by discrete sampling. A generalized formal model for computing systems is needed to faithfully represent both discrete and continuous processes within a unified framework. Hybrid automata present such a framework.

A hybrid system consists of a discrete program within an analog environment. Hybrid automata are generalized finite-state machines for modeling hybrid systems. As usual,

[1] In the web address, http://www.comlab.ox.ac.uk/archive/formal-methods.html, there are pointers to information on formal methods available around the world on the World Wide Web (WWW).

the discrete transitions of a program are modeled by a change of the program counter, which ranges over a finite set of control locations. The global state of a system changes continuously with time according to the laws of physics. For each control location, the continuous activities of the environment are governed by a set of differential equations.

In a linear hybrid automaton, for each variable the rate of change with time is constant --- though this constant may vary from location to location --- and the terms involved in the invariants, guards, and assignments are required to be linear. An interesting special case of a linear hybrid automaton is a timed automaton [Male92]. In a timed automaton each continuously changing variable is an accurate clock whose rate of change with time is always one unit. Furthermore, in a timed automaton all terms involved in assignments are constants, and all invariants and guards only involve comparisons of clock values with constants.

Most of formal methods [Alur93, Chao91, Male92, Ravn93, Heit96a, Henz95, Henz95a, Ostr89, and Henz94] for requirements engineering of HRTS are effective in a partial correctness verification, but have cognitive problems in elicitation and specification including the problem of cognitively unbalanced steps in requirements engineering phase, as follows:

- When system and software engineers elicit and specify the requirements of HRTS through the qualitative and causal thinking, the formal methods have a difficulty to be a communication tool for a mutual understanding of the target systems because they require rigorous and quantitative thinking.
- In a process control loop for HRTS, because it is impossible to model mathematically the non-linear and continuous properties of the controlled process, an approximation approach is indispensable. A physical approximation approach by the qualitative physics is better than the mathematical approximation approaches by fuzzy, probability, and abstract interpretation in eliciting and specifying the physical properties of the controlled process and the controller in HRTS process-control loop.
- The informal and semi-formal methods are cognitively appropriate for elicitation and specification. But, they can produce incorrect analysis result because of the cognitive burden in the validation and safety analysis.
- Most of existing methods, such as Hybrid Automata [Alur93], Timed Automata [Male92], Statechart [Pnue89], RSML [Leve94], SpecTRM-RL [Leve97], Four-variable methods [Parn90] are based on the black box specification of the information hiding principle that is similar to the Whole-Part decomposition in a cognitive systems engineering. The Whole-Part decomposition emphasizes to elicit and specify the HOW from the WHAT

only. It is difficult to analyze the safety by tracing the Why information because the WHY is included in the WHAT implicitly.

Formal methods for HRTS attempt to describe non-linear physical phenomena using quantitative differential equations and time functions [Ostr89, Alur93, Male92, Ravn93, Henz95a, Lemo96]. While these are effective in formal verification of the specification in part, they make the elicitation and specification difficult. The rigorous and quantitative formal languages are not appropriate for understanding the problem of HRTS in a conceptual requirements engineering phase.

In order to overcome the difficulty and the computational complexity problems, recently, many researchers are trying to approximate the quantitative formal methods. For example, Henzinger and Ho proposed the model checking and abstract interpretation strategies for the hybrid automata [Henz95b, Henz94]. Puri and Varaiya suggested the verification method for the hybrid system using abstraction [Parn90]. The duration calculus that requires the quantitative specification and analysis [Hans98, Ravn93] is approximated into the probabilistic duration calculus [Liu93].

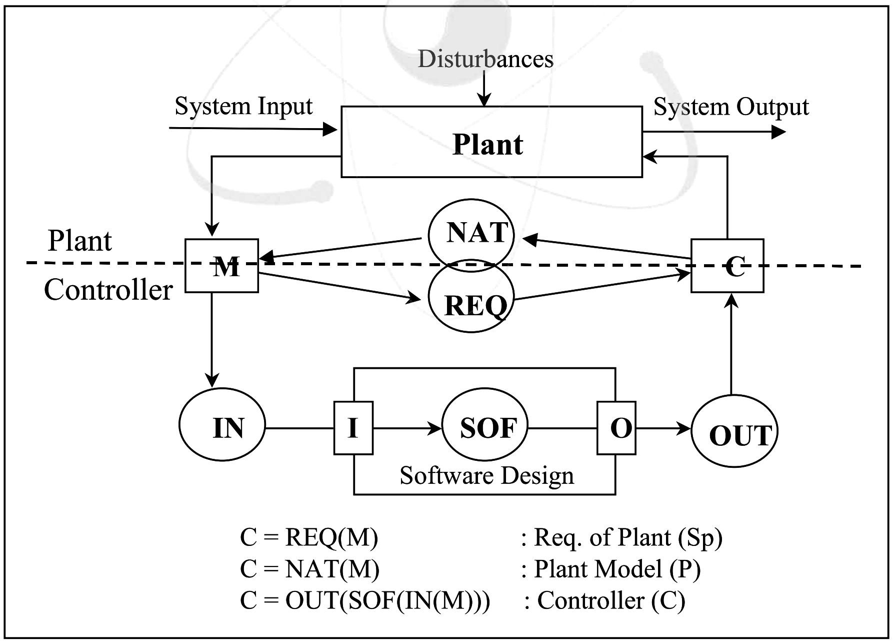
Disturbances
System Input
System Output
Plant
Plant
Controller
M
NAT
REQ
C
IN
I
SOF
O
OUT
Software Design
C = REQ(M) : Req. of Plant (Sp)
C = NAT(M) : Plant Model (P)
C = OUT(SOF(IN(M))) : Controller (C)

Fig. 1 Four-variable method

The discrete approximation was the most successful technique in the practical industry. The Four-variable approach [Parn90], one of the SCR (Software Cost Reduction) methods [Heit96a, Heit96b, Cour93, Wols93], has been used to state requirements for hybrid systems such as the Wolsong SDS2 and the Darlington shutdown system. Requirements are stated as mathematical relations involving the monitored (M), controlled (C), input (I), and output (O) variables as shown in Fig. 1.

While NAT describes any constraints on behavior, such as those governed by physical laws, REQ defines the additional black-box constraints to be enforced by the controller. There are three additional relations, IN, OUT, and SOF, relevant to the controller specification [Parn90], and the proposition (1) can be rewritten as follows:

$$\text{NAT(M) and OUT(SOF(IN(M)))} \rightarrow \text{REQ(M)} \qquad (2)$$

There is a common semantic basis among Four-variable model, Hybrid Automata, and QFM. That is, all these approaches provide a tool to describe the physics of the plant, NAT relation in the Four-variable approach, quantitative differential equations in HA, and qualitative differential equations in QFM. NAT must be carefully interpreted whenever it is modeled by a discrete approach like SCR by Heitmeyer, or continuously modeled by HA, or qualitatively approximated by QFM.

Parnas [Cour93] and Heitmeyer [Heit96a] successfully used a modified Four-variable approach in which discrete approximation is used where tabular and formal notations are used to document the REQ relations. The same approach has been used to document the Wolsong SDS2 requirements. Recently, Heitmeyer and her colleague [Bhar97] tried to perform model checking of hybrid systems using timed automata and PVS [Arch97]. They, however, need human guidance in model checking for reasoning about non-deterministic automata.

These approaches are primarily intended to reduce a computational complexity by approximating the continuous physical phenomena mathematically. The QFM, however, tries to approximate the physical phenomena of HRTS qualitatively and physically, by using the idea of qualitative reasoning based on the physical knowledge on HRTS, in order to reduce not only the computational complexity but also the difficulty.

Our work is motivated by Coombes and Moffet's causal logic-based approaches [Coom93, Coom95, Moff96]. Causal reasoning is an effective and natural approach [Find96] when documenting and analyzing behavior for complex systems. Cognitive approach to system engineering task has been the subject of extensive research as documented in [Rasm94]. Software engineering researchers have recently begun to investigate cognitive aspects of requirements engineering, and the intent specification approach, proposed by Leveson [Leve98, Modu96], is such an example. The QFM is an approach where we try to combine advantages offered by formal specification and qualitative reasoning techniques in order to allow systematic software safety analysis in the early phases of requirements engineering for HRTS. The causal reasoning techniques in qualitative physics domain are introduced in next chapter.

In summary, a formal method for HRTS should have the following features in order to be a good formalism for specifying and validating the software requirements;

- Formality

In order to have the precision of the specification and the analysis, a method should have an appropriate formality. However, a formal method should have a different formality for each phase of the software development. It can be classified as informal, semi-formal, and formal.

- Cognitive approach

A formal method should be balanced cognitively in each phase of the requirements engineering: elicitation, specification, and validation. It should be easy to use, but precise enough to catch all the important information required at the phase. For example, there are cognitive approaches for requirements elicitation and specification, such as goal-based approach, intent specification, causality-based specification and analysis.

- Abstraction

Abstraction is one of primary intellectual tools for coping with complexity. Because it is impossible to model mathematically the non-linear and continuous properties of the plant, an approximation approach is indispensable. The physical approximation approach is better than a mathematical approximation one because it do not need re-interpretation of the approximated specification and analysis.

- Completeness

A formal method for HRTS should have the ability to model the continuous plant and the discrete controller at the same time, and the ability to validate the satisfaction of the proposition (1). Furthermore, it must have the mathematical rigorousness to analyze the timing property and its safety property of the HRTS.

- Integrated approach

A formal method for HRTS should be able to integrate with other formal analysis methods such as FTA semantically or procedurally.

Even though the formal methods for HRTS have their own usage and purpose, we can roughly compare the existing methods using above features as follows:

Table 1 Comparison of formal methods for HRTS

Features Notations	Formality	Cognitively Balanced	Abstraction	Completenes s	Integratio n
DC	HH	LL	LL	HH	HH
HA	HH	L	LL	HH	L
Linear HA	H	L	L	HH	L
Abstracted HA	H	L	M	HH	L
TA	H	L	M	HH	L
Petri nets	M	H	H	H	H
Statechart	M	H	H	M	H
FSM	M	H	H	M	H
FVA	L	H	H	M	L
QFM	H	HH	HH	HH	H
NL	LL	LL	HH	LL	L

HH: High-High, H: High, M: Medium, L: Low, LL: Low-Low
DC: Duration Calculus
HA: Hybrid Automata
TA: Timed Automata
FVA: Four Variable Approach
QFM: Qualitative Formal Method
NL: Natural Language

1.4.2 System and Software Safety Analysis Techniques

A significant problem of developing software for safety critical systems is how to guarantee that the functional behavior of developed software will satisfy the corresponding functional requirements and will not violate the safety requirements for the associated overall system. In order to solve this problem, it is important to analyze thoroughly the safety properties of the overall system, to achieve accurate software functional requirements and to verify properly the implementation of the software.

The interface between system and software, requirements analysis, is a key activity in the process of software development for achieving the safety goal of systems. The quality of requirements analysis determines the quality of requirements specifications, which directly affects the quality of the developed system.

It is hard to bound precisely the environment that should be considered in requirements analysis, but it should cover at least those systems that interact directly with the target system. In the case of safety critical systems the environment model should cover sources of threats to the system and other systems or equipment in which hazards could arise due to failure in the target system.

We believe that identification of appropriate safety requirements is a prerequisite for any useful safety critical application of formal methods. Therefore, safety analysis methods must be incorporated in the lifecycle of formal methods applications. It is important to realize that formal methods are not alternatives to safety analysis; the latter gets as close to analyzing physical reality as possible, while the former deals in models and abstractions.

Because software safety can be analyzed from the relationship between a logical fault of software and a physical hazard of a system, the software safety process should be a subset of the system safety process. However, current approaches for analyzing the software safety, originated from system safety techniques, do not provide formal basis of conducting systematic safety analysis of software, in particular, for HRTS software requirements.

Several techniques for safety analysis have been used by industry for decades, and some have attracted great attention in the research community. They include Fault Tree Analysis (FTA), Failure Modes, Effects and Criticality Analysis (FMECA), Failure Propagation and Transformation Notation (FPTN), Hazard and Operability (HAZOP), and Preliminary Hazard Analysis (PHA). In Leveson’s book, “Safeware” [Leve95], there is an excellent summary on techniques for system safety and computers. It is important to recognize that no analysis technique can guarantee completeness. FTA provides one single structure for specifying software, hardware, and human actions and interfaces with the system. Because FTA is already familiar to system engineers, we have chosen it as a basic tool for CRSA.

Fault Tree Analysis (FTA) is an analytical technique used in the safety analysis of electromechanical systems. An undesired system state is specified, and the system is then analyzed in the context of its environment and operation to find credible sequences of events that can lead to the undesired state. A fault tree thus depicts the logical inter-relationships of basic events that lead to the hazardous event.

FTA has been used for the assessment of system reliability and safety for decades and has been developed into an well-understood, standardized method with wide applications throughout the discipline of safety and reliability engineering. A comprehensive introduction to fault tree analysis is the extensive and authoritative Fault Tree Handbook [Vese81].

Traditional fault tree analysis is a probabilistic method in which potential causes of some failure (“top event”) are organized in a tree structure reflecting causality. High-level events can be caused by various combinations of lower-level events, with the principal logical connectives used in the tree being AND and OR gates, which have meanings analogous to those traditionally used in electronic circuit design. Priority-AND gates, exclusive-OR gates and INHIBIT are also available for use.

Existing software FTA techniques can be grouped as their application phases of the software lifecycle. That is, FTA of software requirements [Hans94 and Liu96], FTA of the software design specification [Cha91, Fene93 and Subs95], and FTA of the software code [Frie95, Clar93, Leve83, Leve87, Cha88]. However, its industrial practice depends heavily on technical expertise of human analysts, the understandability of the system physics, and is often ad hoc.

Leveson and colleagues [Leve83] were the first to apply fault trees to the safety analysis of software at a statement level. Software fault trees are derived from the software (programs) based on the semantics of statements (e.g. sequential, conditional and iteration statements). Unfortunately, the informal nature of the technique is its major weakness because the success of the technique is highly dependently on the ability of the analysts.

Template-based FTA [Cha88] is given for each major construct in a program, and the fault tree for the program (module) produced by composition of these templates. The templates are applied recursively, to give a fault tree for the whole module. As they are applied, the fault tree templates are instantiated, e.g. in the above template the expressions for the conditions would be substituted, and the event for the THEN part would be replaced by the tree for the sequence of statements in the branch. Software FTA can go back from a software hazard, through the program, and stop with leaf events that are either “normal events” representing valid program states, or external failure events. If the hardware failure event probabilities are known, then the top event probability can be determined. Note that this does not rely on a statistical analysis of software reliability.

The semantics of fault trees is closely linked to that of Dijkstra's weakest precondition (wp) calculus [Clar93]. One of the practical advantages of software FTA seems to be that it has the rigor of the wp calculus, but it is presented in a form that is familiar to safety engineers. The difficulty with sequential composition is thus a major drawback, although the links to wp calculus suggest there may be a fruitful area of research in linking formal verification and fault tree analysis. A recent book [Frie95] treats sequential composition in a rather different way, which may effectively address this semantic problem using the concept “program segment prefix”, but there is still a challenge. They interpret the semantic of FTA as a Hoare’s logic [Best96] rather than Dijkstra’s wp calculus [Dijk76].

Clark and McDermid proposed a more traditional view of the application of fault trees to software [Clar93]. It is suggested that weakest preconditions are used for program specification and validation, and software fault tree analysis is employed for a system-wide analysis of hazards. The scope of software fault trees can be increased to include, for example, compiler errors, control errors, and memory errors, as well as logical errors. Thus a more realistic view of the software's role in system hazards can be given.

Hansen and colleagues have recently developed fault trees into a notation for describing software safety requirements for design specifications [Hans94]. Specifications are given in a real-time, interval logic, based on a conventional dynamic systems model with a state changing over time. Fault trees are interpreted as temporal logic formulae giving a cause effect relationship between states. It is shown how such formulae can be used for deriving safety requirements for design components. Similar work on formalization of fault trees is also described in [Gors95].

Fenelon and colleagues proposed an integrated safety analysis method that consists mainly of Hierarchical FTA (HFTA) and Failure Propagation and Transformation Notation (FPTN) [Fene93]. They insist that while Leveson’s template-based FTA is a depth-first and bottom-up approach, HFTA is a top-down and breadth-first method although compatible with Leveson's methods. FPTN is a new notation to integrate software FTA and FMECA, and is somewhat analogous to traditional data flow-based design notations, although instead of showing normal data flow between elements in a system, it describes the propagation and transformation of failures.

Subramanian and colleague proposed ideas to analyze the safety of software in requirements and design phases, Software Requirements Safety Analysis (SRSA) and Software Design Safety Analysis (SDSA) [Subs95]. SRSA and SDSA are also software FTA techniques based on the statechart models [Hare86, Hare87]. They suggested a lot of rules to generate and verify the fault trees, but there are still questions how to verify the completeness and consistency of the rules. Also there is a limitation to apply these ideas to HRTS domain because the statechart is a discrete model.

Liu and McDermid also proposed a model-based software FTA method [Liu96]. They modeled the physical system behavior using an entity-relationship concept. But, there is a drawback of this approach in analyzing the causality of failure behavior, the causal relations of system behavior are modeled by the structural information of system and static relations between components, such as connection, contain, control, input, output, and so on.

Current approaches for analyzing the software safety, originated from system safety techniques, do not provide formal basis of conducting systematic safety analysis of software, in particular, for HRTS software requirements. Because software safety can be analyzed from the relationship between a logical fault of software and a physical hazard of a system, the software safety process should be a subset of the system safety process. Also, because the behaviors of HRTS are determined by the interaction between the continuous plant (P) and the discrete controller(C), we must consider the causal relations between the behavioral properties of the controller software and the behavior of the combined models (P and C).

In summary, a safety analysis method for HRTS should have the following features in order to be a good solution for analyzing the safety of the requirements;

- Formality

In order to have the precision of the analysis, a method should have an appropriate formality. However, a safety analysis method should have a different formality for each phase from system to software development.

- Cognitive approach

  A safety analysis method should be cognitively balanced in each phase from system to software, and also in requirements, design, and coding phases of the software. It should be easy to use, but precise. For example, there are cognitive approaches such as goal-based approach, causality-based safety analysis.

- Model-based systematic approach

  A safety analysis method for HRTS software should provide a model-based approach because the safety of software is tightly related with the plant and controller model. It is also preferable to provide a systematic solution such as template-based FTA.

- Behavioral safety analysis

  Most of the software safety analysis methods based on fault tree analysis are recognized as static method. However, the safety analysis method for HRTS software should be a dynamic analysis method in order to be able to find the cause of the physical hazard of the system from the behavioral aspects of the software.

- Integrated approach

  A software safety analysis method should be able to be integrated in all dimensions

Following is the result of the comparison on the existing software fault tree analysis methods according to the above features.

Table 2 Comparison of software fault tree analysis methods for HRTS

Features / Approaches	Formality	Cognitively Balanced	Model-Based Systematic	Behavioral Analysis	Integrated
[Leveson83]	L	H	L	L	L
[Cha88]	M	M	H	L	L
[Clarke93]	H	M	L	L	M
[Hansen94]	H	L	H	H	L
[Gorski95]	H	-	-	-	-
[Fenelon93]	M	H	H	L	H
[Subramanian95]	M	M	H	M	M
[Liu96]	M	M	H	H	M
CRSA	H	H	H	H	H

H: High, M: Medium, L: Low, -: Not applicable

2. Evaluation Procedure for Software Requirements Specification (SRS)

2.1 Introduction

A Software Requirements Specification (SRS) should establish the requirements for a software system. A Software Requirements Safety Analysis (SRSA) should confirm that the requirements do not pose a system hazard.

The SRS can be viewed as a bridge, or connection, between the overall safety system design and the software. The design of the safety system can be expected to impose certain requirements on the software. The SRS can be viewed as a translation of these requirements into language understandable to software engineers, and the addition of specific software requirements necessary to achieving the desired behavior of the software system.

There are two safety issues involved in a SRS. First, it is necessary to ensure that all system hazards that the software is expected to handle are indeed covered by the SRS. Second, it is necessary to ensure that the SRS does not add additional hazards to the system. The SRSA is one means of analyzing these safety concerns.

2.2 Review Techniques

There is no single assessment method that can, by itself, provide an adequate level of confidence in a SRS or a SRSA. Therefore, we recommend that two or more methods be used, where the methods are selected to compensate for one another’s weaknesses.
One method that should always be used is that of requirements reviews and requirements safety reviews. Other possible methods are:

- **Requirements modeling**
- **Prototyping**
- **Formal analysis**
- **Metrics**
- **Requirements testing**
- **HAZOP analysis**

The results of all the analyses can be combined into a single assessment of the quality and safety of the SRS. In general, this will be a qualitative result; in a few cases, a quantitative result may be possible.

2.2.1. Reviews

Reviews can be carried out using checklists. One such list for a SRS, based on the U.S. Standard Review Plan, BTP-14, is given in Section 3 below. In addition, there are questions about a safety analysis report in Section 4. The SRS review includes questions about safety. The first set of questions can be used to examine the SRS for desirable safety properties. The second set of questions, on the SRSA, can be used to examine a safety analysis that has been carried out on the SRS. Thus, two levels of analysis are possible. In a specific case, some choice should be made as to which sets of questions are to be used, based on the actual circumstances.

2.2.2. Requirements Modeling

Models can be used to acquire understanding of an SRS, and to demonstrate that the SRS has been correctly constructed. Such models abstract particular aspects of the requirements for more intense examination. Different models use different abstractions, and thus yield somewhat different sorts of information about the SRS. The following types of models are frequently used; details can be found in books on software engineering[2].

- Data flow diagrams
- Finite state machines
- Petri nets
- Queueing models
- Decision trees and decision tables
- Entity-relationship models
- Sequence diagrams
- Use-case diagrams
- Formal methods

[2] See, for example, Dean Leffingwell and Don Widrig, *Managing Software Requirements: A Unified Approach*, Addison Wesley,2000.

The Unified Modeling Language (UML)[3] can be used to create models combining several of these separate techniques. A valuable use of UML modeling is the activity of creating the model. If this can be done, and the result is sensible, then the portion of the requirements included in the model is likely to be correct. In any case, the action of creating the model is very helpful in understanding the requirements.

We recommend that the requirements review process include modeling of the SRS, and suggest that a UML model is appropriate. Creating such a model without adequate tool support is difficult, so such a tool should be acquired.

2.2.3. Prototyping

For a reactor safety system, prototyping is likely to be most useful in modeling data communications, particularly if a potential problem is anticipated. This might be the case where many separate computers share a single network; a prototype can be constructed of the network in order to ensure that there will be sufficient network capacity under worst case conditions. Specific guidance on data communications systems is presented in BTP-21. A communications system prototype is one method of demonstrating that the guidance here has been satisfied.

We recommend the development and use of a prototype only if necessary to accomplish the goals set forth in BTP-21. If these goals can be accomplished by some other form of analysis, then that will be sufficient.

2.2.4. Formal Methods

Mathematical analysis of requirements can be quite helpful in determining if requirements have been stated with precision and in discovering subtle inconsistencies. There are three primary methods in use: Vienna Definition Method (VDM), Z and Parnas Tables. If formal methods are used, we recommend the third. This has been used on reactor safety systems (at Darlington) with some success.

---

[3] See the following books for more information on UML: (1) Grady Booch, James Rumbaugh and Ivar Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1999; (2) James Rumbaugh, Ivar Jacobson and Grady Booch, *The Unified Modeling Language Reference Model*, Addison Wesley, 1999; and (3) Bruce P. Douglass, *Real-Time UML: Developing Efficient Objects for Embedded Systems*, Addison Wesley, 2000.

The Darlington effort was extremely expensive, but this was at least partially a result of imposition of the formal method after the code was finished. This has been found to be generally true of formal methods – they need to be used initially and then followed throughout the development, or they become too expensive and time consuming to be worth while. Perhaps because of the after-the-fact imposition, Parnas added many features to his method that appear unnecessary. We believe that a simplified version of this could prove helpful in verifying the functional characteristics of the requirements if a formal verification is used.

The functions required for reactor safety systems are often described in terms of Boolean equations before the software requirements are written. If this should occur, these equations are themselves formal mathematical descriptions, and nothing further should be necessary. The Boolean equations themselves, of course, are subject for formal verification.

A primary value of translating a requirements specification into any formal language is whether or not this can be done. If it can, then the requirements are probably correct. If not, or if there is difficulty, then the requirements need some additional work.

We are not recommending that formal methods be used for requirements analysis since we are not convinced that the resulting increase in safety is worth the cost when compared to other methods of analysis. Be aware, however, that this opinion is not widely shared in the academic safety community.

2.2.5. Metrics Collection and Analysis

We recommend that a small amount of metric information be collected during the requirements review activity. This will serve primarily as a baseline for further work, and can be useful in assessing the increasing capabilities of the reviewers and in comparing the efforts of different sets of verifiers. We suggest that the following three primitive measures be collected.

- Size of the requirements, which may be calculated using full function point analysis, number of pages, or any other method which can be used consistently
- The number and criticality of defects found by each assessment activity
- The number of staff-hours spent on each assessment activity

From the above it is possible to calculate several derived measures, *defect density* and *review efficiency*. Defect density is defined to be the number of defects (of each level of criticality) per unit of software size. For example, if function points are used as a size measure, then a derived measure could be number of safety-critical defects per function point. There is some ambiguity in interpretation of defect density. A low number might mean a good requirements specification, or a poor review team. (On the other hand, a high number always means a poor specification.) Consequently, some care is required in interpreting this number.

Review efficiency is defined to be the number of staff hours spent in reviews per defect found. There is also some ambiguity here, in that a high number might mean few defects (so that a long time is required to find each), a poorly written specification (so that it takes a long time to understand the specification well enough to identify defects) or a poor or inefficient review team.

Be particularly careful to focus on the insights gained from collecting metrics, not on the numbers. Early collection of metric information, particularly the derived measures, are useful mostly to establish a baseline from which to determine future process improvement (or the lack thereof). Effort is required in order to penetrate behind the numbers in order to discover what the numbers mean. Without a commitment to making this effort, metric collection is pointless and should be omitted.

2.2.6. Requirements Testing

At the end of development, the implementation of the software requirements will be validated by some type of testing. This can be very difficult if the requirements are not written so as to be testable. We recommend that, as part of the requirements verification activity, one or more specific test cases be written for each software requirement. This test should be able to objectively determine whether the requirement has been met or not – that is, it should be possible to write a test driver that will execute the software for each requirement, and compare the result with a pre-determined answer.

The generation of validation tests is independent of whether the software is developed in house by the vendor, whether pre-developed software is used, or whether Commercial Off-the-Shelf (COTS) software is used. The validation process is much the same in concept, though the test driver will be different in each case. It doesn’t really matter if the tests generated through this activity are the actual tests used later on, though of course it is to be hoped that most of them are usable. The important thing is to determine whether such test cases can, in fact, be developed for the requirements specification.

At this point, the test designer should not worry about practicalities such as the cost of carrying out the tests, whether some tests are redundant or about detailed test case definitions. For example, a test that a specified degree of reliability has been met may be very expensive to run, but the definition of such a test can reveal faults in the specification.

Tests can be generated in concept for the functional requirements (accuracy, functionality, reliability, robustness, safety, security and timing). The process characteristics discussed in BTP-14 are characteristics of the requirements specification itself, not characteristics of the resulting software, so are not testable in the sense meant here.

2.2.7. HAZOP Analysis

Hazard analysis is used extensively in reactor designs, and can be extended to the software design elements with some care. Little extensive experience with software hazard analysis has been reported in the literature. John McDermid, York University, U.K., has reported success in using HAZOP analysis on software systems, and I wrote a report in 1995 for the U.S. Nuclear Regulatory Commission proposing the use of this form of analysis on reactor safety systems.[4]

Software requirements hazard analysis investigates the impact of the SRS on system hazards. The analysis recommended in the cited report is to ask HAZOP-type questions about the requirements document, as the first step in the software hazard analysis. For example, the following questions should be asked about sensor input:

- What should the software do if the sensor is stuck at all zeros?
- What should the software do if the sensor is stuck at all ones?
- What should the software do if the sensor is stuck somewhere else?
- What should the software do if the sensor is below the minimum allowed range?
- What should the software do if the sensor is above the maximum allowed range?
- What should the software do if the sensor is within range, but wrong?
- What should the software do if the physical units are wrong?

[4] See J. D. Lawrence, *Software Safety Hazard Analysis*, NUREG/CR-6430, U. S. Nuclear Regulatory Commission, February 1996.

- What should the software do if the sensor data has a wrong data type or data size?

Similar types of questions should be asked about other functional properties of the software requirements. For more information on this topic, see the report.

2.3. Review Topics – Software Requirements Specification

Acceptance criteria for a SRS are divided in BTP-14 into two sets: functional characteristics and process characteristics, as shown in the following table.

Table 3. Review Topics of Acceptance Criteria of SRS

Functional characteristics	**Process characteristics**
Accuracy	Completeness
Functionality	Consistency
Reliability	Correctness
Robustness	Style
Safety	Traceability
Security	Unambiguity
Timing	Verifiability

An SRS that exhibits the functional and the software development process characteristics listed below should be produced. Reg. Guide 1.172, "Software Requirements Specifications for Digital Computer Software Used in Safety Systems of Nuclear Power Plants," which endorses IEEE Std 830, "IEEE Recommended Practice for Software Requirements Specifications," describes an acceptable approach for describing software requirements.

The remainder of this section gives each review topic, a discussion of the topic as quoted in BTP-14, and review questions that may be asked in order to determine if the topic has been satisfied. Some questions are accompanied by quotations from various sources and other comments.

2.3.1. Functional Characteristics

2.3.1.1. Accuracy

Accuracy requirements should be provided for each input and each output variable. Accuracy requirements should be stated numerically, and appropriate physical units and error bounds should be supplied. Accuracy requirements should include a description of data type and data size for each input and output variable.

- Does an accuracy requirement exist for each output variable that has a numerical value?
- Is each accuracy requirement stated quantitatively?
- Are the physical units stated for each accuracy requirement?
- Does each accuracy requirement include permissible error bounds?
- Do all accuracy requirements include data type and data size information?

Simple data types include integer, fixed point and floating point. These can be combined to create more complex data types - for example, vectors and matrices. Data size is generally given as the number of bits required to accurately represent the variable throughout its range.

- Does an accuracy requirement exist for each input variable that has a numerical value?

2.3.1.2. Functionality

Functionality requires that the operations that must be performed for each mode of operation be completely specified. Functions should be specified in terms of inputs to the function, transformations to be carried out by the function, and outputs generated by the function.

- Are termination requirements specified? Such as power down and shut-down sequences.
- Does the Software Requirements Specification specify completely the functional requirements for all modes of operation identified in the System Design Description and Safety Analysis Report? Such as refueling, system installation and commissioning, test, normal operation, start-up and shut-down, off-normal operation and emergency operation.
- Do functional requirements include starting conditions and system status at the

initiation of each function?

- Do functional requirements specify the input and output variables required by each function?
- Do functional requirements include task sequences, actions and events required to carry out each function?
- Do functional requirements include termination conditions and system status at the conclusion of each function?
- Do functional requirements include directly or indirectly the relevance of each function to system reliability and safety?
- Does the Software Requirements Specification identify those variables in the physical environment that the software must monitor and / or control?
- Such as temperatures and pressures.
- Does the Software Requirements Specification represent variables in the physical environment by mathematical variables?
- Does the Software Requirements Specification define the required behavior of the controlled variables in terms of monitored variables with the use of mathematical functions? Monitored variables are those the software has to measure, and controlled variables are those the software is intended to control. The entire set of monitored variables must be covered by these mathematical functions.
- Do functional requirements include the purpose of each function?
- Do functional requirements include trigger conditions which cause each function to operate?
- Are initialization requirements specified? Such as initial value of variables, start-up sequences and power up sequences.

2.3.1.3. Reliability

Reliability requires that all requirements for fault tolerance and failure modes be fully specified for each operating mode. Software requirements for handling both hardware and software failures should be provided, including requirements for analysis of and recovery from computer system failures. Requirements for on-line in-service testing and diagnostics should be provided.

- Are software reliability requirements derived from the reliability requirements of the System Design Description?
- Are software reliability requirements defined quantitatively?
- That is, in terms of failure rate or mean time to fail criteria.

- Are requirements for fault tolerance and graceful degradation defined?
- Are reliability and availability criteria given for each mode of operation?
- Does the Software Requirements Specification contain requirements for on-line in-service testing and diagnostics?

2.3.1.4. Robustness

Robustness requires that the behavior of the software in the presence of unexpected, incorrect, anomalous and improper (1) input, (2) hardware behavior, or (3) software behavior be fully specified. Of particular concern is the behavior of the software in the presence of unexpectedly high or low rates of message traffic.

- Does the Software Requirements Specification specify the behavior of the software in the presence of unexpected rates for message traffic? This includes both unexpectedly high rates and unexpectedly low rates.
- Does the Software Requirements Specification specify the behavior of the software in the presence of unexpected, incorrect, anomalous and improper input data and other anomalous conditions?
- This includes unexpected data types, formats, physical units, accuracies, sampling intervals, ranges, options, timing, and frequency of occurrence.
- Does the Software Requirements Specification specify the behavior of the software in the presence of unexpected, incorrect, anomalous and improper hardware or software behavior? The following factors must be considered: failures shall be identified to a reasonable degree of detail and isolated to the most narrow environment; fail-safe output shall be guaranteed as far as possible; if such a guarantee cannot be given, system output shall violate only less essential safety requirements; the consequences of failures shall be minimized; remedial procedures, such as fall back, re-try, system recovery should be considered for inclusion; reconstruction of obliterated or incorrectly altered data may be tried; and information on failures shall be provided to the operating staff.
- Does the SRS require (1) the checking of status after exit from any procedure where the status is provided and (2) appropriate action if an incorrect status is detected?

2.3.1.5. Safety

Safety requires that the software functions, operating procedures, input, and output be classified according to their importance to safety. Requirements important to safety should be identified as such in the SRS. The identification of safety items should include safety analysis report requirements, as well as abnormal conditions and events as described in Reg. Guide 1.152.

- Are the software conditions which can lead to a hazardous state identified in the Software Requirements Specification?
- Does the SRS specify the input conditions and the calculations necessary as a prelude to the software initiating protective actions?
- Does the Software Requirements Specification identify and define safe and unsafe (i.e., hazardous) reactor states?
- Are requirements for validity checks on operator and sensor inputs defined in the Software Requirements Specification?
- Does the Software Requirements Specification classify sensors and actuators according to their safety criticality?
- Does the Software Requirements Specification specify software actions which are necessary to prevent plant damage, including the necessary calculations and their physical background?
- Does the Software Requirements Specification classify software functions according to their safety criticality?
- Does the Software Requirements Specification specify software actions which are necessary to carry out emergency shutdown of the reactor, including the necessary calculations and their physical background?
- Are software actions specified for potential common mode failures as required by the System Design Description and Safety Analysis Report?

The following factors are relevant to avoiding common mode failures [IEC 880]: defense-in-depth, graceful degradation, management of failures in general, functional diversity and (if necessary) software diversity, spatial separation and modularization, decoupling, logical separation.

2.3.1.6. Security

Security requires that security threats to the computer system be identified and classified according to severity and likelihood. Actions required of the software to detect, prevent, or mitigate such security threats should be specified, including access control restrictions.

- Does the Software Requirements Specification impose requirements to prevent unauthorized personnel from interacting with the software system?

- Does the Software Requirements Specification impose access restrictions on operators, managers and other personnel?
- Are the security requirements, taken as a whole, mutually consistent?
- Are potential security threats to the computer system identified, classified according to severity and likelihood, and documented?
- Does the Software Requirements Specification impose requirements to prevent unauthorized changes to the software system?
- Does the Software Requirements Specification specify requirements to address security threats?

2.3.1.7. Timing

Timing requires that functions that must operate within specific timing constraints be identified, and that timing criteria be specified for each. Timing criteria should be provided for each mode of operation. Timing requirements should distinguish between goals and requirements. Timing requirements should be stated in such a way that the time delay between stimulus and response for safety actions is deterministic under normal and anticipated failure conditions. BTP HICB-21 provides additional guidance on real-time performance.

- Does the Software Requirements Specification specify storage tolerances?
- Does the Software Requirements Specification specify the time-critical functions and the timing criteria for each? Timing criteria include minimum times, maximum times, sampling frequencies, time intervals and timing tolerances, as appropriate. Criteria may differ according to the different modes of operation
- Are volume and throughput expectations given for the software?
- Do memory size requirements state explicitly which are merely target figures or goals and which are absolutely necessary for the software system?
- Is the software system required to have deterministic timing?
- Do timing requirements state explicitly which are merely target figures or goals and which are absolutely necessary for the software system?
- Does the Software Requirements Specification specify timing tolerances?
- Are timing requirements specified for each mode of operation?

2.3.2. Process Characteristics

2.3.2.1. Completeness

Completeness requires that all actions required of the computer system be fully described for all operating modes and all possible values of input variables (for example, the complete span of instrument inputs or clock/calendar time) * . The SRS should describe any actions that the software is prohibited from executing. The operational environment within which the software will operate should be described. All variables in the physical environment that the software must monitor and control shall be fully specified. Functional requirements should describe (1) how each function is initiated; (2) the input and output variables required of the function; (3) the task sequences, actions, and events required to carry out the function; and (4) the termination conditions and system status at the conclusion of the function. User interfaces should be fully described for each category of user.

- Are operator interfaces fully defined? Operator interface definitions can be given in terms of keyboard inputs; control panels; positioning and layout of controls and displays; human reaction and decision times; use of colors, bold face, underlining and blinking of displays; menu techniques;
- Does each functional requirement describe how the function is initiated? This includes the trigger conditions that cause the function to operate, the starting conditions at the initiation of the function and the required system status at the initiation of the function. The requirements should be written in such a way that a single requirement specifies no more than one function.
- Can the required safety-related functions be correctly implemented within the existing, available resources? Resources include budget, schedule, manpower, equipment, software tools.
- Are the specified models, algorithms and numerical techniques practical and within the state of the art?
- Are the quality attributes specified for the software achievable both for each software unit and the complete integrated software system? Such as accuracy, adaptability, availability, clarity, completeness, consistency, correctness, deterministic timing, integrity, maintainability, modularity, reliability, robustness, safety, security, serviceability, simplicity, stability, testability, traceability, understandability, uniformity, usability and validity.
- Is the relationship between the monitored and input variables, and the relationship between the output and controlled variables, precisely described?

- Are error conditions described, including required corrective actions?
- Do requirements exist to permit the operator to verify that basic system functions are operating?
- Do requirements for the use of colors, positions of information on the display screen, icons, flashing signals and alerting signals follow a consistent scheme?
- Are the variables in the physical environment that the software must monitor and control completely specified? The required behavior of the controlled variables shall be specified in terms of the monitored variables with the use of mathematical functions.
- Is there a requirement that the computer system will report its own defects and failures to the operator?
- Are requirements for control panels and display layouts specified?
- Are procedures required for introducing, modifying and displaying parameters to the operator exactly defined?
- Are requirements for human reactions to software-generated messages specified, including the amount of time available for making decisions?
- Is there a requirement that manual interactions shall not delay basic safety actions beyond specified safe limits?
- Is each possible input from each sensor completely described? The description should be in terms of the appropriate items in the following list: type of sensor (analog, digital); possible range of values; units of measurement; resolution of measurement; error bounds on measurements for the range of measurement; instrument calibration; and conversion algorithms. Examples of conversion patterns include analog to digital and bit patterns to physical units.
- Does the Software Requirements Specification specify the behavior of the software for anomalous inputs? This includes inputs received before startup, inputs received after shutdown and inputs received when the computer is temporarily disconnected from the process. The purpose is to cover spurious inputs which appear to come from the process under computer control, but do not actually come from there.
- Does each functional requirement specify the input and output variables required by the function?
- Are the actions required of the computer system for error recovery completely described?
- The description should include the type of error, the procedure (if any) for notifying the operator of the error, and the means of restoring service. The recovery procedure should not compromise safety; simply halting is generally not a satisfactory

response.

- Does each functional requirement specify task sequences, actions and events required to carry out the function?
- Does each functional requirement specify the termination conditions and system status at the conclusion of the function?
- Are all output variables from functions completely described? The description should be in terms of the appropriate items in the following list: data types, formats, physical units, accuracies, update intervals, valid ranges, available options, timing, frequency of occurrence, message error rates and types, alerting signals and method of access by the software.
- Is each possible output to each actuator completely described? The description should be in terms of the appropriate items in the following list: type of actuator (analog, digital); possible range of values and units; units of measurement; resolution of measurement; calibration requirements; and conversion algorithms.
- Is each category of operator specified, including expected experience level for each category? The experience mentioned in the question refers to computer experience, not reactor experience. Less experienced computer operators may require fuller explanations on screens, less technical help messages, etc. The expected experience level can affect screen design, so should be mentioned in the SRS.
- Does the Software Requirements Specification completely specify the software interfaces? This includes interfaces to hardware, predeveloped software, commercial off-the-shelf software and operators.
- Are the actions required of the computer system for which fail-safe action must be taken completely described?
- Are all the operating modes within which the software must perform listed and described?
- Does the Software Requirements Specification describe the operational environment within which the program must run?
- Does the Software Requirements Specification state what the software must not do? The SRS must not contain requirements unless they are imposed explicitly or implicitly by the SyDD or the SAR. All requirements imposed by the SyDD and the SAR that specify actions which the software must not do must be contained in the SRS. Any "requirements" that the software not do something which are not contained in higher level documents are unlikely to be detected by the auditors.
- Are all actions required of the computer system for every mode of operation completely described? This includes start-up, shut down, initialization, termination,

normal operation, off-normal operation, degraded operation, emergency operation, etc.

- Are all inputs variables to functions completely described? The description should be in terms of the appropriate items in the following list: data types, formats, physical units, accuracies, sampling intervals, valid ranges, available options, timing, frequency of occurrence, alerting signals and method of access by the software.

2.3.2.2. Consistency

Consistency requires that the contents of the SRS be consistent with the safety system requirements, the safety system design, and documented descriptions and known properties of the operational environment within which the safety system software will operate. Individual requirements should not contradict other requirements. Timing requirements should be consistent with thermohydraulic analyses performed in the system safety analysis. Uniform and consistent terminology, notation, and definitions should be used throughout the SRS.

- Are the models, algorithms and computational techniques specified in the Software Requirements Specification mathematically mutually compatible?
- Are the accuracies required of input, computational and output data mutually compatible?
- Are the individual requirements consistent with the System Design Description and the Safety Analysis Report?
- Are requirements for similar functions mutually consistent?
- Do requirements for the use of color, positions of information on display screens, icons, flashing signals and alerting signals follow a consistent scheme?
- Has the SRS been analyzed for internal contradictions and has this analysis been documented?
- Do models, algorithms, and numerical techniques specified in the Software Requirements Specification agree with standard references where applicable?
- Are input and output specifications in the Software Requirements Specification consistent with interface requirements imposed by the hardware or predeveloped software?
- Specifications include data type, data size, data rate, accuracy, error bounds, and physical units.
- Are the individual requirements consistent with documented descriptions and known properties of the operational environment into which the program must fit?

- Have uniform and consistent terminology and definitions been used throughout the Software Requirements Specification?

2.3.2.3. Correctness

Correctness requires that the description of actions required of the computer system be free from faults and that no other requirements be stated. The operational environment within which the software will operate should be accurately described. All variables in the physical environment that the software must monitor and control should be properly specified. Functional requirements should accurately describe (1) how each function is initiated; (2) the input and output variables required of the function; (3) the task sequences, actions and events required to carry out the function; and (4) the termination conditions and system status at the conclusion of the function.

- Has an independent analysis of the algorithms specified in the SRS been done to verify the correctness of those algorithms?

2.3.2.4. Style

Style requires that the contents of the SRS be understandable. The SRS should differentiate between requirements placed on the software and other supplementary information, such as design constraints, hardware platforms, and coding standards. A precise definition of each technical term should exist, either in the SRS or in a separate dictionary or glossary. Each requirement should be uniquely and completely defined in a single location in the SRS.

- Does the Software Requirements Specification differentiate between requirements placed on the software and other supplementary information? Such as design constraints, hardware platforms, and coding standards.
- Is the functional requirements portion of the Software Requirements Specification organized in such a way that the requirements for each operating mode are grouped together?
- Is each requirement uniquely and completely defined in a single location in the Software Requirements Specification?
- Does the Software Requirements Specification contain a precise definition of each technical term and mnemonic that occurs in the Software Requirements Specification? Reference to a dictionary or glossary containing the required definitions will satisfy the intent of this question.

- Does the Software Requirements Specification conform to standards imposed by the reactor vendor or software developer?
- Does the Software Requirements Specification distinguish between requirements and design constraints?
- Is there written justification for each design and each implementation constraint contained in the Software Requirements Specification? Factors which limit the designer's options include, but are not limited to, regulatory and other legal policies; hardware limitations; interfaces to other applications; audit functions; use of specific operating systems, compilers, languages, and database management systems; use of specific communications protocols; critical safety considerations; and critical security considerations.
- Is the Software Requirements Specification complete in the sense that it contains no blank sections or paragraphs?

2.3.2.5. Traceability

Traceability requires that a two-way trace exist between each requirement in the SRS, and the safety system requirements and design. There should be a two-way trace between each requirement in the SRS and the software design, as well as a forward trace from each requirement in the SRS to the specific inspections, analyses, or tests used to confirm that the requirement has been met.

- Can each requirement be traced backward to specific elements in the System Design Description or Safety Analysis Report? The preferred technique is a requirements tracing matrix which identifies each requirement, the system component implementing the requirement, the system design element which generated the requirement, and the test used to confirm that the requirement has been met.
- Can each requirement be traced forward to specific tests or validation criteria which will be used to confirm that the requirement has been met?
- Can each requirement be traced forward to specific design elements?

2.3.2.6. Unambiguity

Unambiguity requires that each requirement, and all requirements taken together, have one and only one interpretation.

- Can every requirement be interpreted in one and only one way?

2.3.2.7. Verifiability

Verifiability requires that it be possible to construct a specific analysis, review, or test to determine whether each requirement has been met.

- Is each timing requirement testable?
- Is each security requirement testable?
- Is each reliability and availability requirement testable?
- Is each functional requirement testable?
- Is each safety requirement testable?

2.4. Review Topics – Software Requirements Safety Analysis

The software safety plan describes the safety analysis implementation tasks that are to be carried out by the applicant/licensee. The acceptance criterion for software safety analysis implementation is that the tasks in that plan have been carried out in their entirety. Documentation should exist that shows that the safety analysis activities have been successfully accomplished for each life cycle activity group. In particular, the documentation should show that the system safety requirements have been adequately addressed for each activity group; that no new hazards have been introduced; that the software requirements, design elements, and code elements that can affect safety have been identified; and that all other software requirements, design, and code elements will not adversely affect safety.
Review topics are divided into several sets.

2.4.1. Requirements Hazard Analysis

Software requirements hazard analysis investigates the impact of the software requirements specification on system hazards. Requirements can generally be divided into sets, each of which addresses some aspect of the software. These sets are termed qualities which are to be considered during software hazard analysis: accuracy, capacity, functionality, reliability, robustness, safety, and security. Some variations may be required to match special situations.

The general intent of software requirements hazard analysis is to examine each quality, and each requirement within the quality, to assess the likely impact on hazards. There are also numerous traditional qualities generally considered necessary to an adequate software requirements specification Completeness, consistency, correctness, traceability, unambiguity and verifiability are, of course, necessary, but should be handled as part of requirements analysis and verification, not as part of hazards analysis. However, software requirements hazard analysis will be hampered if the software requirements specification does not possess these qualities.

2.4.1.1 Inputs to Software Requirements Hazard Analysis

The following information should be available to perform the requirements hazard analysis.

- Preliminary Hazard List
- Preliminary Hazard Analysis
- Safety Analysis Report
- Safety System Design Description
- Software Requirements Specification

2.4.1.2 Analysis Procedures

The following steps may be used to carry out the requirements hazard analysis. The steps are meant to help organize the process. Variations in the process, as well as overlap in time among the steps, is to be expected.

1. Identify the hazards for which software is in any way responsible. This identification includes an estimate of the risk associated with each hazard.
2. Identify the software criticality level associated with each hazard and control category.
3. Match each safety-critical requirement in the software requirements specification (SRS) against the system hazards and hazard categories in order to assign a criticality level to each requirement.
4. Analyse each requirement using some guide.
5. Document the results of the analysis.

The information collected during this hazard analysis can be of considerable use later during software development. The combination of critica1ity level assigned to the various software requirements provides information that might affect the assignment of resources during further development, verification and testing. It can also suggest the need for redesign of the application system to reduce software-affected hazards.

It is possible that the Software Requirements Hazard Ana1ysis leads to the conclusion that some changes should be made to the system design. For example, it might be discovered that some system requirements assigned to software can be better met through hardware.

It is likely that the hazard ana1ysis wi11 conc1ude that some requirements do not pose hazards that is, there are no circumstances where failure to satisfy the requirements can cause a hazard. Such requirements probably do not need to be considered in subsequent analyses.

There are many ways to carry out the analysis of step 4. The technique most prominently documented in the literature is Fault Tree Analysis (FTA). Event Tree Analysis (ETA) should a1so be considered as top events in the tree and expanding the tree to consider consequences. The choice of technique depends on what information is known to the analyst and what information is sought.

2.4.1.3 Outputs of Software Requirements Hazard Analysis

The products of the requirements hazard ana1ysis consist of the following items:

- A list of software hazards.
- A criticality level for each hazard that can be affected by the software.
- A critica1ity level for each software requirement.
- An analysis of the impact on hazards of the software when it operates correctly or incorrectly with respect to meeting each requirement.

2.4.1.4 Review Topics for Requirements Hazard Analysis

- Does a software hazard list exist? This list should identify the hazards for which software is in any way responsible, and should include an estimate of the risk associated with each hazard.
- Has the degree of control been determined for each safety-critical requirement?

  For example: (1) software exercises autonomous control over potentially hazardous events such that failure of the software to prevent the event leads directly to the occurrence of the hazard; (2) software exercises control over potentially hazardous events, but there is time for independent safety systems to mitigate the hazard; (3) software displays information requiring immediate operator action to mitigate the hazard, and software failures will permit or fail to prevent the hazard's occurrence; (4) software issues commands over potentially hazardous events, but human action is required to complete the control function and independent safety measures exist for each hazardous event; (5) software generates information of a safety-critical nature used to make safety-critical decisions, but the software does not directly affect hazards (Mil-Std 882C). Another classification scheme is given in IEC 1226, as follows: (A) Functions, systems and equipment which play a primary role in the achievement or maintenance of nuclear power plant (NPP) safety, (B) functions, systems and equipment that play a complementary role to the category A items in the achievement or maintenance of NPP safety, and (C) functions, systems and equipment that play an auxiliary or indirect role in the achievement or maintenance of NPP safety.

- Has each safety requirement been matched against the system hazards and degrees of control in order to assign a criticality level to each requirement? A thread analysis should be performed on several requirements. "Criticality level" refers to the importance of the requirement with respect to safety. Standards in this area are ambiguous.
- Has each software requirement been analyzed to ensure that the requirements, taken both individually and as a whole, do not conflict with the system safety goals and requirements? Test the analysis on several requirements for reasonableness.
- If a conflict exists between the software requirements and the system safety goals, have mitigation measures been identified and carried out to resolve the conflict? The conflicts and their mitigation should be documented and the proposed mitigations should be reasonable. If no such conflicts exist this fact should be documented also.

2.4.2. Requirements Properties

- Have the safety analysts ensured that the safety-related software requirements are complete?
- The method used to assure completeness should be documented and plausible.
- Have the safety analysts ensured that the safety-related software requirements are consistent with the System Design Description and the Safety Analysis Report?
- The method used to assure consistency should be documented and plausible.
- Have the safety analysts ensured that the safety-related software requirements are

internally consistent?

- The method used to assure consistency should be documented and plausible.
- Have the safety analysts ensured that the safety-related software requirements are correct?
- The method used to assure correctness should be documented and plausible.
- Have the safety analyst ensured that the safety-related software requirements are unambiguous?
- The method used to assure lack of ambiguity should be documented and plausible.
- Have the safety analysts ensured that the safety-related software requirements are accurate?
- The method used to assure accuracy should be documented and plausible.

2.4.3. Requirements Safety Analysis

- Have the safety analysts evaluated software safety requirements related to reliability by suitably rigorous methods?
- Have the safety analysts evaluated software safety requirements related to robustness by suitably rigorous methods?
- Have the safety analysts evaluated software safety requirements related to security by suitably rigorous methods?
- Have the safety analysts evaluated software safety requirements related to timing and sizing by suitably rigorous methods?
- Have the safety analysts evaluated software safety requirements related to functionality by suitably rigorous methods?
- This includes functionality relating to all modes of operation.
- Have the safety analysts evaluated software safety requirements related to instrumentation interfaces by suitably rigorous methods?
- Have the safety analysts evaluated software safety requirements related to operator interfaces by suitably rigorous methods?
- Has an ergonomic analysis been performed on safety-related information display requirements?
- Have acceptance criteria been defined for each safety-critical requirement?

IEEE 10.4 requires the acceptance criteria to be consistent with (1) results obtained from similar computer programs, (2) solutions of classical problems, (3) accepted experimental results, (4) analytical results pub. in tech. lit. and/or (5) solutions of benchmark problems.

2.4.4. Requirements Safety Analysis Results

- Have all safety-related deficiencies in the software requirements specification identified by analysts been formally discussed with the development team and formally documented?
- Have all safety-related deficiencies in the Software Requirements Specification been identified and have they all been corrected in such a way that the specification no longer has such deficiencies?
- This correction should not impact the functionality of the specification. All corrections should be documented.
- Was the safety analysis activity itself well documented?
- The analysis should follow the safety plan.
- Is there evidence that all specified corrective actions have actually taken place?

2.4.5. Safety Documentation and Records

- Is the safety process embedded in all of the software requirements documentation and not just tacked on with the software safety plan?
- This includes the following documents; others may be required in specific cases: project management; software requirements; software development standards, practices and conventions; and software requirements verification and validation documentation.
- Is the person responsible for software safety requirements records known by name and does he understand his job?
- Are all necessary requirements safety records under configuration control?
- This includes results of safety analyses, information on suspected or verified safety problems that have been detected in requirements technical documents, results of audits performed on software requirements safety activities, results of safety analyses carried out on the software requirements, and records on training provided to software safety personnel and software development personnel that relate to requirements activities. Verify that several documents, such as deficiency reports or other safety records, are preserved as configuration items.
- Do the requirements safety records identify the means used to track each hazard, the means of handling the hazard and the status of the hazard through the software requirements activities?
- Check several such hazards for proper identification, tracking, handling and status.

2.4.6. Safety Organization and Responsibility

- What are the names of the individuals who performed the requirements safety analysis?
- Talk with at least one such individual to find out his qualifications for this work and if he is conversant with some or all of the results.
- Were the analysts that carried out the Requirements Safety Analysis well qualified to undertake the analysis?
- Are the formal lines of communication for the Requirements Safety Analysis documented?
- This includes communication between system safety organization and software safety organization (if they are not the same); software safety organization and software development organization; software safety organization and software V&V organization; and software safety organization and software CM organization. These lines of communication should be consistent with the software safety plan.
- Do the software safety analysts have the authority to enforce software requirements compliance with system safety requirements and practices?
- The authority should be documented and unambiguous and should have adequate management authority behind it.
- Does a single individual have overall responsibility for the conduct of the Requirements Safety Analysis?
- Were sufficient resources made available to carry out Requirements Safety Analysis?
- Resources include financial, schedule, personnel, computers and other equipment and tools.

3. Conclusion

The accuracy of the specification of requirements of a digital system is of prime importance to the acceptance and success of the system. It has been shown that a significant portion of the problems with software-based systems can be traced back to incorrect or incomplete requirements specification. When the software-based system is a replacement for an existing analogue system, it is a mistake to just take the requirements specification for the old system and use it for the new digital system, since the latter system has different characteristics to the former. The analogue system requirements specification can be used as a starting point, but the digital characteristics must be taken into account when preparing the new requirements specification.

It is important to make sure that the requirements specification is accurate and complete. Software-based equipment brings with it unique opportunities and unique concerns compared to analogue equipment. Therefore, it is important to make sure that these differences are taken into account when the requirements specification is developed.

The requirements specification needs to completely define the functions of the system and the qualification requirements, including acceptance criteria. The requirements specification must also address system issues including defining the external and internal interfaces and hardware-software interactions.

4. References

[AECL93] AECL, “Software Work Practice: Procedure for Software Hazard Analysis of Safety Critical Software,” 00-68000-SWP-006, Rev. 0, September 1993.

[AECL94] AECL, “Wolsong NGS: SDS2 Software Hazard Analysis Report,” 86-68350-ANL-001, Rev. 0, October 1994.

[Alur93] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, “Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems”, Hybrid Systems Workshop, Lecture notes in computer science, vol. 736, Springer-Verlag, pp. 209-229, 1993.

[ANS86] ANS 8.3. “Criticality Accident Alarm System,” ANSI/ANS 8.3, 1986.

[Arch97] M. Archer, and C. Heitmeyer, “Verifying Hybrid Systems Modeled as Timed Automata: A Case Study,” HART’97, Grenoble, France, Lecture Notes in Computer Science 1201, Springer-Verlag, pp. 171-185, March 1997.

[ASM90] ASME Std NQA_2a_ 1990 Part 2.7. “Quality Assurance Requirements of Computer Software for Nuclear Facility Applications.” ASME, 1990.

[ASM94] ASME Std NQA_1_1994. “Quality Assurance Requirements for Nuclear Facility Applications.” ASME 1994.

[Bor90] Boris Beizer, “Software Testing Techniques,” 2nd Edition, Van Nostrand Reinhold, 1990.

[BTP19] NUREG-0800, BTP HICB-19. “Guidance for Evaluation of Defense-in-Depth and Diversity in Digital Computer-Based Instrumentation and Control Systems,” Rev. 4, U.S. Nuclear Regulatory Commission, June 1997.

[BTP97] Standard Review Plan, Appendix 7_A BTP HICB-14-27 Rev. 4-June 1997.

[Cha88] S. S. Cha, N. G. Leveson, and T. J. Shimeall, “Safety Verification in Murphy using Fault Tree Analysis,” ICSE-10, IEEE Computer Society Press, pp. 377-386, 1988.

[Cha91] Cha S. S. A Safety-Critical Software Design and Verification Technique. Ph.D. Dissertation, Information and Computer Science, UC Irvine, 1991.

[Chao91] Z. Chaochen, C. A. R. Hoare and A. P. Ravn, “A calculus of duration,” Information Processing Letter, vol. 40, no. 5, pp. 269-276, 1991.

[Chri92] M. G. Christel, K. C. Kang, “Issues in Requirements Elicitation,” Technical Report CMU/SEI-92-TR-12, Sep. 1992.

[Chu93] Chudleigh, M., Hazard Analysis Using HAZOP: A Case Study, in Proceedings of

SAFECOMP 93.

[Clar93] S. J. Clarke and J. A. McDermid, “Software Fault Trees and Weakest Preconditions: A Comparison and Analysis,” IEE Software Engineering, pp. 225-236, 1993.

[Cour93] P. J. Courtois, D. L. Parnas, “Documentation for Safety Critical Software,” 15th International Conference on Software Engineering, ICSE-15, pp. 315-323, 1993.

[Dijk76] E. W. Dijkstra, A Discipline of Programming, Prentice Hall, 1976.

[Fen93] Fenelon P, McDermid J. A. An Integrated Tool Set for Software Safety Analysis. J. Systems Software, vol. 21, 1993. p. 279-290.

[Frie95] M. A. Friedman and J. M. Voas, Software Assessment: Reliability, Safety, Testability, John Wiley & Sons, 1995.

[Gor95] Gorski J, Wardzinski A. Formalizing Fault Trees. Proc. of the Safety-Critical Systems Symposium. Eds. F. Redmill, and T. Anderson. Springer-Verlag. Brighton, UK. February 7-9, 1995. p. 311-327.

[Han94] Hansen K. M, Ravn A. P, Stavridou V. From Safety Analysis to Formal Specification. ProCos II Technical Report, ESPRIT, Department of Computer Science, Technical University of Denmark, 1994.

[Hans98] K. M. Hansen, A. P. Ravn, and V. Stavridou, “From Safety Analysis to Software Requirements,” IEEE Trans. on Software Engineering, 24, (7), pp. 573-584, July 1998.

[Hare86] D. Harel, A. Pnueli, J. Schmidt, and R. Sherman, “On the formal semantics of statecharts,” In Proc. First IEEE Symp., Logic in Computer Science, pp. 54-64, 1986.

[Hare87] D. Harel, “Statecharts: A visual formalism for complex systems,” Sci. Comp. Prog., pp. 231-274, 1987.

[Heit96a] C. L. Heitmeyer, “Requirements Specification for Hybrid Systems”, Hybrid Systems Workshop III, Lecture notes in computer science, Alur, R., Henzinger, T., and Sontag, E. (Eds.) Springer-Verlag, 1996.

[Henz94] T. A. Henzinger and P. H. Ho, “Model Checking Strategies for Linear Hybrid Systems,” Cornell Technical Report, CSD-TR-94-1437, 1994.

[Henz95] T. A. Henzinger and P. H. Ho, “A Note on Abstract Interpretation Strategies for Hybrid Automata”, Hybrid Systems Workshop II, Lecture notes in computer science, vol. 999, Antsaklis, P., Kohn, W., Nerode, A., and Sastry, S.(Eds.), Springer-Verlag, pp. 252-264, 1995.

[Henz95a] T. A. Henzinger and P. W. Kopke, “What’s Decidable About Hybrid Automata,” Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC), pp. 373-382, 1995.

[I1012] IEEE Std 1012. “IEEE Standard for Software Verification and Validation Plans,” ANSI/IEEE Std 1012, 1986.

[I1028] IEEE Std 1028. “IEEE Standard for Software Reviews and Audits.” IEEE Std 1028, 1988.

[I1063] IEEE Std 1063. “IEEE Standard for Software User Documentation.” ANSI/IEEE Std 1063, 1987.

[I1074] IEEE Std 1074_1991. “IEEE Standard for Developing Software Life Cycle Processes.” IEEE, 1991.

[I1228] IEEE Std 1228. “IEEE Standard for Software Safety Plans,” IEEE Std 1228, 1994.

[I379] ANSI/IEEE Std 379-1988. “Standard Application of the Single-Failure Criterion to Nuclear Power Generating Station Safety Systems.”

[I384] IEEE-384-1992. “IEEE Standard Criteria for Independence of Class 1E Equipment and Circuits.”

[I603] IEEE Std 603-1991. “IEEE Standard Criteria for Safety Systems for Nuclear Power Generating Stations.”

[I60880] Draft Amendment 1 to IEC 60880. “Software for Computers Important to Safety For Nuclear Power Plants – First Supplement to IEC Publication 880,” Ed. 1, International Electrotechnical Commission, April 1999.

[I61513] Draft IEC 61513. “Nuclear Power Plants – Instrumentation and Control for Systems Important to Safety – General Requirements for Computer-based Systems,” International Electrotechnical Commission, April 1997.

[I7432] IEEE Std 7_4.3.2_1993. “IEEE Standard for Digital Computers in Safety Systems of Nuclear Power Generating Stations.” IEEE, 1993.

[I830] IEEE Std 830_1993. “IEEE Recommended Practice for Software Requirements Specifications.” IEEE, 1993.

[Jaff88] M. S. Jaffe, “Completeness, Robustness, and Safety in Real-Time Software Requirements and Specifications,” Ph.D. thesis, Univ. of California, Irvine, 1988.

[Lemo96] R. de Lemos and J. G. Hall, “Extended RTL in the Specification and Verification of an Industrial Press,” Hybrid Systems Workshop III, Lecture notes in computer science, vol. 1066, Alur, R., Henzinger, T., and Sontag, E. (Eds.), Springer-Verlag, pp. 114-125, 1996.

[Lev83a] Leveson N. G, Harvey P. R. Analyzing Software Safety. IEEE Trans. Software Eng. Vol. SE-9, No. 5, September 1983.

[Lev87] Leveson N. G, Stolzy J. L. Safety Analysis using Petri Nets. IEEE Trans. Software

Eng. Vol. 13, 1987. p. 386-397.

[Lev95] Leveson N. G. SAFEWARE: System Safety and Computers. Addison-Wesley Publishing, 1995.

[Leve83] N. G. Leveson and P. R. Harvey, “Analyzing Software Safety,” IEEE Trans. Software Eng. Vol. SE-9, No. 5, September 1983.

[Leve86] N. G. Leveson, “Software Safety: Why, What, and How,” Computing Surveys, vol. 18, no. 2, June 1986.

[Leve87] N. G. Leveson and J. L. Stolzy, “Safety analysis using Petri nets,” IEEE Trans. Software Eng. Vol. 13, pp. 386-397, 1987.

[Leve90] N. G. Leveson, “The Challenge of Building Process-Control Software,” IEEE Software, November 1990.

[Leve94] N. G. Leveson, M. P. E. Heimdahl, H. Hildreth, and J. D. Reese, “Requirements Specification for Process-Control Systems,” IEEE TSE, Vol. 20, No. 9. September, 1994.

[Leve97] N. G. Leveson, L. D. Pinnel, S. D. Sandys, S. Koga, J. D. Reese, “Analyzing Software Specifications for Mode Confusion Potential,” Presented at the Workshop on Human Error and System Development, Glascow, March 1997.

[Liu93] Z. Liu, A. P. Ravn, E. V. Sorensen, and C. Zhou, “A Probabilistic Duration Calculus.” Responsive Computer Systems, vol. 7, Dependable Computing and Fault-Tolerant Systems, Springer-Verlag, pp. 29-52, 1993.

[Liu95] S. Liu, V. Stavridou, and B. Dutertre, “The Practice of Formal Methods in Safety-Critical Systems,” Journal of Systems Software, Elsevier Science Inc., Vol. 28, pp. 77-87, 1995.

[Liu96] Liu S, McDermid J. A. Model-Oriented Approach to Safety Analysis Using Fault Trees and a Support System. J. Systems Software, vol. 35, 1996. p. 151-164.

[Male92] O. Maler, Z. Manna, and A. Pnueli, “From timed to hybrid systems,” Real Time: Theory in Practice, Lecture notes in computer science, vol. 600, Bakker, d.J.W., Huizing, K., Roever, d.W.P., and Rozenberg, G. (Eds.), Springer-Verlag, pp. 447-484, 1992.

[McD94] McDermid, J. A & Pumfrey, D. J., A Development of Hazard Analysis To Aid Software Design, COMPASS `94

[McDe89] J. A. Mcdermid, “Requirements Analysis: Problems and the START Approach,” In IEE Colloquium on “Requirements Capture and Specification for Critical Systems”, 4/1-4/4, IEE, Nov. 1989.

[MoD89] Defense Standard 00_55, Ministry of Defense, Glasgow, May 1989.

[MoD94] Feasibility study for MoD PES HAZOP, MoD Ref NSM 13C/1063, 1994.

[MSE93] Staff Requirements Memorandum. “SECY-93-087, Policy, Technical, and Licensing Issues Pertaining to Evolutionary and Advanced Light-Water Reactor (ALWR) Designs,” U.S. Nuclear Regulatory Commission, July 21, 1993.

[N6101] NUREG/CR_6101. “Software Reliability and Safety in Nuclear Reactor Protection Systems.” 1993.

[Ostr89] J. S. Ostroff, Temporal Logic for Real-Time Systems, Research Studies Press, 1989.

[Parn90] D. L. Parnas and J. Madey, “Functional Documentation for Computer Systems Engineering,” Technical Report 90-287, Queen's University, TRIO, Sep. 1990.

[Pnue89] A. Pnueli and M. Shalev, “What is in a step?: On the semantics of statecharts,” J.W. De Baker, L. Amicorum, J. Klop, J. Meijer, and J. Rutten (Eds.), Amsterdam:CWI, pp. 373-400, 1989.

[R1152] Regulatory Guide 1.152. “Criteria for Digital Computers in Safety Systems of Nuclear Power Plants.” Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1996.

[R1173] Regulatory Guide 1.173. "Developing Software Life Cycle Processes for Digital Computer Software Used in Safety Systems of Nuclear Power Plants." Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1997.

[Ravn93] A. P. Ravn, H. Rischel, and K. M. Hansen, “Verification of Hybrid Systems using Abstractions”, Hybrid Systems Workshop II, Lecture notes in computer science, vol. 999, Antsaklis, P., Kohn, W., Nerode, A., and Sastry, S.(Eds.) Springer-Verlag, pp. 359-369, 1995.

[SPM93] NPX80_SQP_0101.0, “Software Program Manual for NUPLEX 80+,” ABB_CE, January 21, 1993.

[Sub95] Subramanian S, Vishnuvajjala R. V, Mojdebakhsh R, Tsai W.T, Elliott L. A. Framework for Designing Safe Software Systems. COMPSAC’95, 1995. p. 409-414.

[Wols93] Wolsong NPP 2/3/4, “Software Requirements Specification for Shutdown System 2 PDC,” Design Document no. 86-68350-SRS-001, Rev. 0, June 1993.

서 지 정 보 양 식

수행기관보고서번호	위탁기관보고서번호	표준보고서번호	INIS 주제코드
KAERI/TR-**1871**/2001			

제목/부제	차세대 원자로 디지털 계측제어 소프트웨어 요구명세 평가절차서
연구책임자 및 부서명	이장수(미래원자력기술개발단, MMIS팀)
연 구 자 및 부 서 명	박종균, 이기영(이상 동력로기술개발팀) 김장열, 천세우 (이상 MMIS팀)

출 판 지	대전	발행기관	한국원자력연구소	발행년	2001. 6
페 이 지	59 p.	도 표	있음(○), 없음( )	크 기	21*29.6 Cm.

참고사항	'01 원자력 중장기 계획사업		
비밀여부	공개(○), 대외비( ), __ 급비밀	보고서종류	기술보고서
연구위탁기관		계약 번호	

초록 (15-20줄내외)

원전 디지털 계측제어계통을 개발할 때 소프트웨어 요구명세의 정확도, 완전성과 안전성 보장은 최종 시스템 개발성공과 인허가 획득에 아주 중요한 요소가 된다. 원전 계측제어계통의 디지털화 추세에 따라 원자력산업의 특수성인 안전성 확보와 컴퓨터 소프트웨어 안전성 심사 및 개발기준이 되는 국제표준 정립을 위해 국제원자력기구 (IAEA), 국제전기기술위원회 (IEC), 국제전기전자공학회 (IEEE) 등에서 표준화 노력을 기울이고 있으며 현재 한국원자력연구소에서는 소프트웨어 공통모드고장 문제에 대한 대책의 일환으로 일련의 필수안전 소프트웨어 평가 방법론들을 개발하고 있다. 본 보고서에서는 새롭게 개정되는 국제표준의 요건과 한국 원자력 안전기술원의 차세대 원자로 안전 규제지침을 만족하고 소프트웨어 공통모드고장의 대책이 될 수 있는 차세대 원자로 디지털 계측제어 소프트웨어 요구명세 평가 절차를 개발하였다. 이 보고서는 차세대 원전 안전 소프트웨어 요구명세를 평가 할 때 상위법과 표준들에서의 요구사항들의 만족여부를 평가할 수 있도록 인도하는 지침서이다. 이 지침서 1장에서는 원전 소프트웨어 요구명세 공학에 대한 소개와 정형적 요구 명세 기법과 요구명세 안전성 분석 기법 등을 종합 정리하였다. 2장에서 차세대 원전 소프트웨어 요구명세를 평가하기 위해 개발한 평가항목 별 평가 절차와 요구명세 단계에서의 안전성 분석을 위한 평가 절차를 기술하였다.

주제명키워드 (10단어내외)	차세대, 원자로, 소프트웨어 요구명세, 요구명세 안전성분석, 평가절차,

BIBLIOGRAPHIC INFORMATION SHEET

Performing Org. Report No.	Sponsoring Org. Report No.	Standard Report No.	INIS Subject Code
KAERI/TR-1871/2001			

Title / Subtitle	Evaluation Procedure of Software Requirements Specification for Digital I&C of KNGR
Project Manager and Department	Jang-Soo Lee (Man Machine Interface System Team)
Researcher and Department	Jong-Kyun Park, Ki-Young Lee (Power Reactor Technology Development Team) Jang-Yeol Kim, Se-Woo Cheon (MMIS Lab),

Publication Place	Taejon	Publisher	KAERI	Publication Date	2001. 6
Page	59p.	Ill. & Tab.	Yes(O), No ( )	Size	26 Cm.
Note	'01 Mid-and-Long Term Nuclear Research Project				
Classified	Open(O), Restricted( ), ___ Class Document	Report Type	Technical Report		
Sponsoring Org.	MOST	Contract No.			

Abstract (15-20 Lines)

The accuracy of the specification of requirements of a digital system is of prime importance to the acceptance and success of the system. The development, use, and regulation of computer systems in nuclear reactor Instrumentation and Control (I&C) systems to enhance reliability and safety is a complex issue. This report is one of a series of reports from the Korean Next Generation Reactor (KNGR) Software Safety Verification and Validation (SSVV) Task, Korea Atomic Energy Research Institute, which investigates different aspects of computer software in reactor I&C systems, and describes the engineering procedures for developing such a software. The purpose of this guideline is to give the software safety evaluator the trail map between the code & standards layer and the design methodology & documents layer for the software important to safety in nuclear power plants. Recently, the requirements specification of safety-critical software systems and safety analysis of them are being recognized as one of the important issues in the software life cycle, and being developed new regulatory positions and standards by the regulatory and the standardization organizations such as IAEA, IEC, and IEEE. We presented the procedure for evaluating the software requirements specifications of the KNGR protection systems. We believe it can be useful for both licenser and licensee to conduct an evaluation of the safety in the requirements phase of developing the software. The guideline consists of the requirements engineering for software of KNGR protection systems in chapter 1, the evaluation checklist of software requirements specification in chapter2.3, and the safety evaluation procedure of KNGR software requirements specification in chapter 2.4.

Subject Keywords (About 10 words)	Requirements Specification, Requirements Safety Analysis, Standards, Evaluation Procedure