Available online at www.sciencedirect.com

SCIENCE DIRECT®

Reliability Engineering and System Safety 89 (2005) 248–260

RELIABILITY ENGINEERING & SYSTEM SAFETY

www.elsevier.com/locate/ress

An effective technique for the software requirements analysis of NPP safety-critical systems, based on software inspection, requirements traceability, and formal specification

Seo Ryong Koo[a,*], Poong Hyun Seong[a], Junbeom Yoo[b], Sung Deok Cha[b], Yeong Jae Yoo[c,1]

[a]Department of Nuclear and Quantum Engineering, Korea Advanced Institute of Science and Technology, 373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, South Korea

[b]Division of Computer Science and AITrc/SPIC/IIRTRC, Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, 373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, South Korea

[c]BNF Technology Inc., 150, Duckjin-dong, Yuseong-gu, Daejeon 305-353, South Korea

Received 2 April 2004; accepted 24 August 2004
Available online 19 November 2004

Abstract

A thorough requirements analysis is indispensable for developing and implementing safety-critical software systems such as nuclear power plant (NPP) software systems because a single error in the requirements can generate serious software faults. However, it is very difficult to completely analyze system requirements. In this paper, an effective technique for the software requirements analysis is suggested. For requirements verification and validation (V&V) tasks, our technique uses software inspection, requirement traceability, and formal specification with structural decomposition. Software inspection and requirements traceability analysis are widely considered the most effective software V&V methods. Although formal methods are also considered an effective V&V activity, they are difficult to use properly in the nuclear fields as well as in other fields because of their mathematical nature. In this work, we propose an integrated environment (IE) approach for requirements, which is an integrated approach that enables easy inspection by combining requirement traceability and effective use of a formal method. The paper also introduces computer-aided tools for supporting IE approach for requirements. Called the nuclear software inspection support and requirements traceability (NuSISRT), the tool incorporates software inspection, requirement traceability, and formal specification capabilities. We designed the NuSISRT to partially automate software inspection and analysis of requirement traceability. In addition, for the formal specification and analysis, we used the formal requirements specification and analysis tool for nuclear engineering (NuSRS).

Keywords: Software requirements analysis; V&V; Safety-critical systems; Inspection; Traceability; Formal method

1. Introduction

The use of digital systems has been increasing in the nuclear industry in recent years. The importance of software verification and validation (V&V) is therefore emphasized for nuclear safety. Inspection is widely believed to be an effective technique of software V&V. It can greatly increase productivity and product quality by reducing development time and by removing defects. Inspection can be applied to the whole software life cycle. By inspecting products as early as possible, major defects will be revealed sooner and will not be propagated through to the final product.

However, software inspection is labor-intensive and its introduction is difficult to justify in terms of the investment of time and money. This labor-intensive feature is compounded because software inspection uses little technology. Software inspection does not fit in well in a development environment that is more technology-oriented.

These problems are mainly due to a lack of understanding. Many nuclear engineers have heard of software

* Corresponding author. Fax: +82 42 869 3810.
E-mail addresses: srkoo@kaist.ac.kr (S.R. Koo), phseong@kaist.ac.kr (P.H. Seong), jbyoo@salmosa.kaist.ac.kr (J. Yoo), cha@salmosa.kaist.ac.kr (S.D. Cha), yjyoo@bnftech.com (Y.J. Yoo).
[1] Fax: +82 42 868 4384.

inspection, but few know enough to implement it. They are aware of its often-quoted benefits but are unwilling to risk implementing it. They feel much more secure with traditional testing methods.

Software inspection is also difficult to implement properly. Incorrect implementation will produce poor results. When this happens, people are discouraged from using software inspection again, and apocryphal tales of how 'inspection was a disaster for us' soon spread. Nonetheless, software inspection is gaining in popularity. More people are using it and benefiting from it. At the same time, new variations are being created or tailored for certain types of products or for use under certain circumstances.

For more concrete inspection, requirements traceability analysis is considered a component of software inspection in this work. This point of view is the motivation for integrating the capability of requirements traceability analysis into software inspection. Requirements traceability analysis identifies requirements that are either missing from, or added to, the original requirements. Applying requirements traceability to the software architecture phase can aid in identifying requirements that have not been accounted for in the architecture. Stepwise refinement of the requirements into the architecture produces a natural set of mappings from which requirements traceability can be derived.

Although formal methods such as Statechart [1], CPN [2], RSML [3], and SCR [4] are considered effective V&V activities, their proper use is difficult in the nuclear fields due to their mathematical nature. It is because the software developers in nuclear fields are more familiarized in 'single and intuitive' notations such as ladder-logic or function block diagram programming. Formal specification, however, can lessen requirement errors by reducing ambiguity and imprecision and by clarifying instances of inconsistency and incompleteness.

To promote the application of software inspection, requirement traceability, and the formal specification method in this work, we developed an IE approach for requirements which is an effective technique for the software requirements analysis of nuclear power plant (NPP) safety-critical systems in Section 3. Our approach integrates software inspection, requirements traceability, and formal methods in order to support systematic requirement analysis.

We also developed the nuclear software inspection support and requirements traceability tool (NuSISRT), which comprises three views: an inspection view, a traceability view, and a structure view. The inspection view of the NuSISRT was designed to partially automate the software inspection process to reduce the burden of software inspection. Requirements traceability analysis, which is considered an important activity of software V&V, is supported through the traceability view of the NuSISRT. In addition, the structure view of the NuSISRT enables the analyzer to easily specify a system using a formal specification method. Moreover, the NuSRS [5], which is another tool for formal requirements specification and analysis for nuclear fields, supports the formal requirements specification using the results of the structure view.

2. Related works

2.1. Software requirements inspection

Fagan inspection [6] was an attempt to improve software quality using systematic team-orient review. Since Fagan first defined the software inspection process in 1976, there have been many variations of software requirements inspection. The major goal is to detect as many errors as possible, not to suggest corrections or examine alternative solutions.

We describe here the original method.

An inspection team generally consists of four to six people. Each person has one of the following well-defined roles.

Moderator. A moderator is the person overall in charge of the inspection. The moderator's task is to invite suitable people to join the inspection team, distribute source materials and to organize and moderate the inspection meeting.

Author. An inspection requires the presence of the author of the product under inspection. The author can give invaluable help to the inspectors by answering questions pertaining to the intent of the document.

Reader. During an inspection meeting, the reader's job is to paraphrase out loud the document under inspection.

Recorder. The recorder's duty is to note all defects, along with their classification and severity. Although Fagan says this task can be accomplished by the moderator, another member of the team is usually chosen because the workload can be quite high, though mainly secretarial. The recorder is often known as the scribe.

Inspector. Any remaining team members are cast as inspectors. Their only duty is to look for defects in the document.

For effective use of software inspection, Fagan describes the five stages of the inspection process as follows.

Overview. The entire team is present during the overview. The author describes the general area of work and then gives a detailed presentation on the specific document. The document is then distributed to all members and any necessary work is assigned to the members.

Preparation. Each team member carries out individual preparation and studies the document. Errors in the document will be found during this stage but, in general, more errors will be found at the next stage. Checklists of common types of defects can help the inspectors concentrate on the most beneficial areas of the inspection. Each inspector produces a list of comments on the document, indicating defects, omissions and ambiguities.

Inspection. The inspection meeting involves all team members. The reader paraphrases all areas of the document.

During this process, inspectors can stop the reader and raise any issue until a consensus is reached. If members agree that a particular issue is a defect, the issue is classified as missing, wrong or extra. The severity of the defect is also classified as major or minor. At this point the meeting moves on. No attempt is made to find a solution for the defect; this step is carried out later. After the meeting, the moderator writes a report detailing the inspection and all defects. The report is then passed to the author for the next stage.

Rework phase. During the rework phase, the author corrects all the defects that are found in the document and detailed in the moderator's report.

Follow-up. After the document has been corrected, the moderator ensures that all required alterations have been made. The moderator then decides whether the document should be re-inspected, either partially or fully.

2.2. Requirements traceability analysis

Requirements traceability (RT) is to trace system, software requirements through requirements, design, code, and test materials RT is concerned with the relationships between requirements, their sources and the system design.

Through the RT analysis, we can get the following benefits:

- completeness/omissions
- identification of most important or riskiest paths
- locations of interactions
- discovery of root causes of faults and failures

For more efficient RT analysis, the calculation algorithm for similarity between Korean sentences was developed in this work. That for English sentences was adopted from the Luhn's work [7]. So, the calculation algorithms can support both the documents written in English and Korean. In this section, we introduce the basic concepts of the calculation algorithms for similarity.

2.2.1. Statistical analysis for English documents—cosine vector similarity formula

In 1957, Luhn [7] noted that an information retrieving system could be made by comparing specific words with the words of a query. Once certain important terms are extracted through analysis of the documents subject of a search, each document can be expressed in the following vector form according to the existence of the terms

$$D = (t_1, t_2, t_3, \cdots, t_n) \tag{1}$$

where D is term vector of a specific document and t_k is 0 or 1 depending on whether the document contains specific terms ($k = 1, 2, \ldots, n$).

An information request or a query can also be expressed by a term vector as follows

$$Q = (q_1, q_2, q_3, \cdots, q_n) \tag{2}$$

where Q is the term vector of a query and q_k is 0 or 1 depending on whether the query contains specific terms ($k = 1, 2, \ldots, n$).

The simplest method of defining similarity is to compute the value of similarity with the number of terms that coexist in a document and a query. In this case, similarity of document written in English is represented by the following formula:

$$\text{Similarity}\ (D, Q) = \sum_{k=1}^{n} t_k q_k \tag{3}$$

Nevertheless, by assigning different weighting factors to each term, the measurement of similarity can be more effective than merely using 0 and 1. Such weighting factors can be assigned by many different methods. The normalization of vectors is a common method. By this procedure, the similarity between a document and a query is given from the following cosine vector similarity formula:

$$\text{Similarity}\ (D, Q) = \frac{\sum_{k=1}^{n} w_{qk} w_{dk}}{\sum_{k=1}^{n} (w_{qk})^2 \sum_{k=1}^{n} (w_{dk})^2}, \tag{4}$$

where w_{qk} is the weighting factor of the kth term in the query and w_{dk} is the weighting factor of the kth term in the document ($k = 1, 2, \ldots, n$).

2.2.2. Linguistic analysis for Korean documents—case grammar

Unlike English, the Korean language has a free word order (scrambling) and inflections. In addition, it has many ellipses of essential sentence components. Furthermore, Korean has an agglutinative characteristic in which a word is formed with an essential morpheme and a formal morpheme. Consequently, linguistic methods are favored over statistical methods for the analysis and processing of Korean. By modifying and simplifying the grammar of natural languages, linguistic methods enable languages to be analyzed. Some of the typical types of grammar are as follows:

- phase structure grammar (Chomsky)
- unification grammar
- dependency grammar (Tesniere)
- case grammar (Fillmore).

Among these types of grammar, dependency grammar and case grammar are favored for their suitability to scrambling and omitting. There have been many studies, particularly in the Department of Computer Science at the Korea Advanced Institute of Science and Technology (KAIST), on the processing systems of natural language using these types of grammar. A methodology for analyzing traceability based on the concepts of case grammar was proposed in [8]. In analyzing Korean, it is possible to grasp the cases of substantives with the information about the case

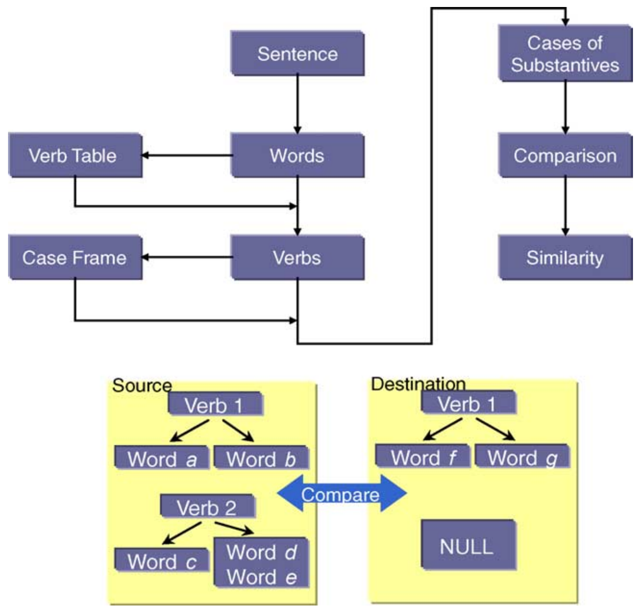


Fig. 1. The similarity calculation algorithm for Korean documents.

frames of verbs and the postpositions added to the substantives. Once analyzed with case grammar, sentences are expressed in vector form. Then their similarities can be computed using Cosine Vector Similarity Formula. In this manner, substantives of the same cases (i.e. semantic roles) can be compared to obtain the similarity of sentences.

Fig. 1 shows a schematic diagram of the similarity calculation algorithm for Korean documents presented in [8]. The proposed method can be considered a more semantic-oriented method than the statistical method, because it re-structures sentences based on the semantic roles of words or phrases in the sentences.

2.3. NuSCR approach

The Atomic Energy of Canada Limited (AECL) approach specifies a methodology and format for the specification of software requirements for safety-critical software used in real-time control and monitoring systems of nuclear generating systems. The AECL approach is a SCR-style software requirement specification (SRS) verification method based on the Parnas four-variable method [9]. A nuclear system reads environmental states through monitored variables that are transformed into input variables. The values of the output variables are calculated and changed into control variables. The AECL provides two different views of the software requirements: the function overview diagram (FOD) presents a large view, and the structured decision table (SDT) presents a small view of each function depicted in the FOD. The AECL specifies all the requirements of the nuclear control system in the FOD and SDT notations. This specification is somewhat complex if timing and history-related requirements are considered, though the difficulty of specification is alleviated in the software cost reduction (SCR) for nuclear engineering.

To maximize safety of NPP safety-critical software, proven-effective formal methods are being used. For example, SCR-style notation was previously used to specify software requirements for Wolnsung SDS2 [10], a shutdown system currently in service at a different plant in Korea. Experts who performed critical analysis on SCR and other formal specification languages came to the conclusion that SCR-like notation is well-suited for specifying and verifying requirements for NPP but that the notation in its current form is too verbose to be effectively used. Furthermore, availability of SCR* toolset was unsatisfactory from the viewpoint of KNICS [11] project management office. Therefore, an effort was initiated to: (1) customize SCR so that characteristics unique to nuclear engineering domain are best reflected in the design of a specification language; and (2) develop a tool suite to integrate graphical editing capability and formal verification environment.

The NuSCR approach [12], as noted earlier, customizes SCR to nuclear engineering industry. The NuSCR approach, based on SCR-style AECL notation [9] used in specifying requirements for Wolsung SDS2, uses function overview diagram (FOD) to capture high-level data flows. In addition, three basic constructs—function variable, history variable, and timed history variable—are defined by structured decision table (SDT), finite state machine (FSM), and timed transition system (TTS), respectively [13]. NuSCR improves the readability of specification and enhances expressiveness by supporting intuitive notations. Details on formal definition of NuSCR syntax and semantics are found in [12].

In the approach of AECL, the state-based operations such as trip setpoint hysterisis are specified by functions, although they have originally state-based features. It is because that the basic specifying concept of AECL approach is to specify all aspects by functions. Timing-based operations such as delay timer are also specified by special timer functions, which are too hard to define and understand. In the NuSCR approach, we adopt FSM for specifying state-based parts, and a kind of TTS for timing-related parts in software requirements.

The NuSCR formal software requirements specifications can be verified by theorem prover PVS [14,15]. Using PVS, we can verify the structural properties such as input/output completeness, consistency, and circular dependencies in NuSCR specification. NuSCR specifications can also be verified by model checker such as the SMV [16], based on the formal semantics of NuSCR presented in this work. We are developing an automatic translator that translates NuSCR specification into SMV inputs. Therefore, we can say that the NuSCR approach is an extended formal specification and verification method of the existing SCR-style of the AECL approach.

3. An effective technique for the software requirements analysis

In this work, we developed an IE approach for requirements which is an effective technique for the software requirements analysis; the technique enables easy inspection by combining requirements traceability and effective use of a formal method. There are some difficulties in putting inspection into practice. Because software requirements inspection is labor-intensive, it is difficult to implement properly. If implemented wrongly, they will produce poor results, in comparison with the effort expended. Also, there are difficulties in using formal method. Because of mathematical nature of formal method, it is difficult to understand the syntax and semantics. Usually, most nuclear engineer tends to like simple and intuitive techniques because they are more familiarized with the ladder-logic or function block diagram style simple logic. So, formal methods are not easy to be used properly in nuclear fields.

For more effective software requirements analysis, IE approach for requirements which integrates inspection, requirements traceability and formal specification was proposed in this work. By using IE approach for requirements, we can support the inspection of all software development documents written in natural language. The inspection results, which are the main requirements elicited from the documents, can then be used not only for requirements traceability analysis but also for software formal specification.

Because of the difference in domain knowledge between a developer and a requirements analyzer, the analyzer has difficulty in completely understanding design documents written in a natural language instantly and in formally specifying the software requirements of the documents, making it difficult to assure the quality of a specification. As a result, reducing the difference in domain knowledge is critical for software development. Furthermore, because requirement documents are mostly written in a natural language, the analyzer needs considerable time and effort to understand and formally specify the documents.

Generally, a software life cycle consists of a concept-phase, a requirements phase, a design phase, an implementation phase, and a test phase. Each phase is defined so that its activities are distinct from the activities of the other phases. As shown in Fig. 2, in the IEEE Standard 1012-1998, titled 'Software Verification and Validation,' [17] minimum V&V tasks for safety-critical systems are defined for each phase. The V&V tasks should be traceable back to the software requirements, and a critical software product should be understandable for independent evaluation and testing.

Document evaluation and traceability analysis are major tasks of the concept and requirements phases. In the concept-phase, which is the initial phase of a software development project, the needs of the user are described and evaluated through documentation. The requirements phase is the period in the software life cycle when requirements such as the functional and performance capabilities of a software product are defined and documented.

In this work, IE approach for the software requirements analysis focuses on the concept and requirements phases. To fill the gap between the natural language document phase and the formal specification phase, our approach helps the user to easily perform an inspection and to efficiently compose a formal specification.

Fig. 3 shows a schematic diagram of the approach proposed in this work. Our effective technique for the software requirements analysis consists of four steps for document evaluation and requirement analysis throughout the concept and requirement phases.

3.1. Document evaluation in IE approach for requirements

The first stage of supporting V&V activity involves evaluation of the concept and requirement documents written in natural language. In this stage, to increase the quality of the design documents written in natural language, our approach supports software requirements inspection (Step 1) and requirements traceability analysis (Step 2). In the first stage, which focuses on document evaluation, our

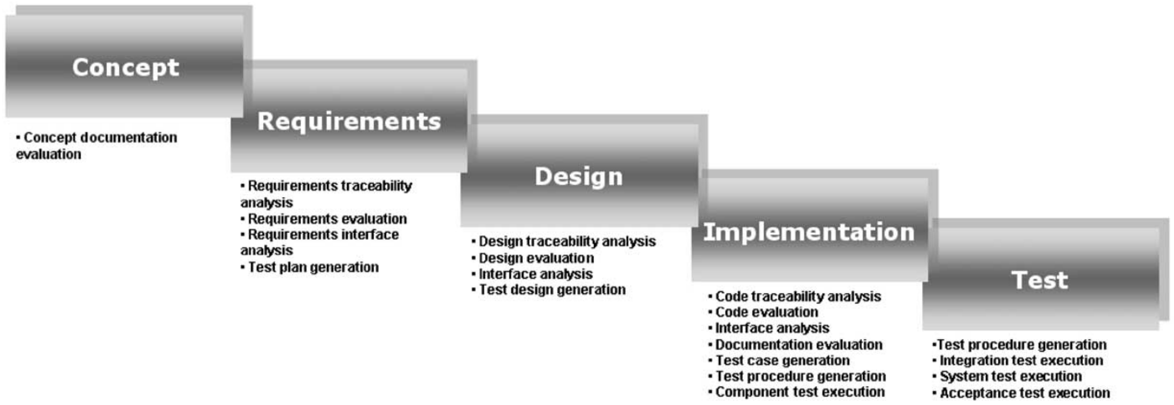

Concept
• Concept documentation evaluation
Requirements
• Requirements traceability analysis
• Requirements evaluation
• Requirements interface analysis
• Test plan generation
Design
• Design traceability analysis
• Design evaluation
• Interface analysis
• Test design generation
Implementation
• Code traceability analysis
• Code evaluation
• Interface analysis
• Documentation evaluation
• Test case generation
• Test procedure generation
• Component test execution
Test
•Test procedure generation
• Integration test execution
• System test execution
• Acceptance test execution

Fig. 2. Software V&V tasks during the life cycle.

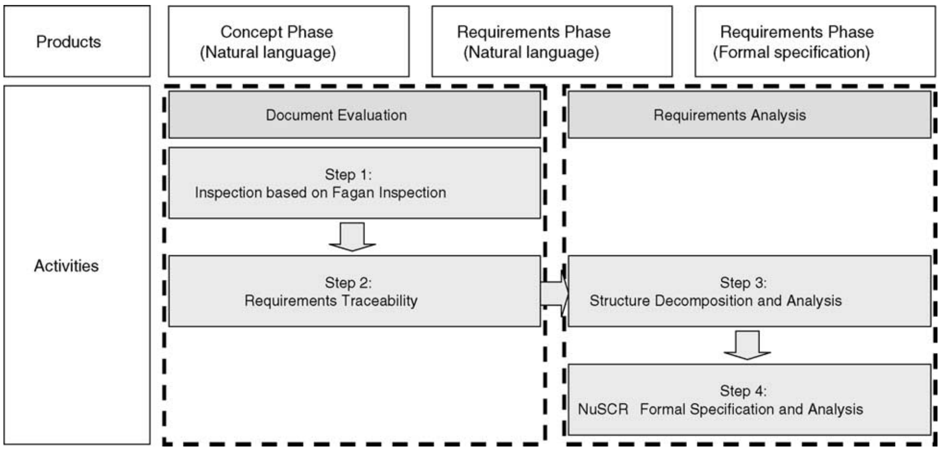

Products
Concept Phase (Natural language)
Requirements Phase (Natural language)
Requirements Phase (Formal specification)
Activities
Document Evaluation
Step 1: Inspection based on Fagan Inspection
Step 2: Requirements Traceability
Requirements Analysis
Step 3: Structure Decomposition and Analysis
Step 4: NuSCR Formal Specification and Analysis

Fig. 3. Schematic diagram of the IE approach for requirements.

approach enables the user to easily conduct inspections and requirements traceability.

3.1.1. Step 1: inspection based on Fagan inspection

In Step 1, software requirements inspection is based on the Fagan inspection process mentioned in Section 2: the system requirements are elicited on the basis of Fagan's inspection checklists. Rigor is a desirable inspection attribute. Repeatability is also essential if feedback is to be used to improve the process. In addition, it is important to identify requirements that are either missing from, or added to, the original requirements. Through the systematic inspection process, these attributes should be supported in Step 1 of IE approach for requirements. The major role of Step 1 is document handling, which supports on-line browsing of documents and elicitation of requirements. And then, composition of checklist is also an important role.

Actually, in one document, there are many sentences but all of them are not requirements. Therefore, we have to elicit adequate requirement sentences for more effective inspection. And then, software requirements inspection based on checklist can be performed by each inspector. As a simple example, we performed inspection for ATWS mitigation system (AMS) based on our approach. We examined the functional requirements (FR) document [18] in concept-phase and software requirements specification (SRS) document [19] in requirement-phase. In view of three V&V criteria such as completeness, consistency, and correctness, the checklist was composed by authors for checking of S/W function definition, I/O variable definition, S/W behavior definition, and interface definition. As shown in Table 1, we could find some comments for AMS according to V&V items in the checklist.

3.1.2. Step 2: requirements traceability

In Step 2, requirements traceability is another important V&V activity. Using the inspection results of Step 1, we can more easily map the source requirements (concept-phase document) and the destination requirements (requirement-phase document). In this work, we also developed techniques for calculating the similarity between requirement sentences. Based on the cosine vector similarity formula and the case grammar mentioned in Section 2,

Table 1
Inspection results of AMS

FR (Concept)	Completeness		Consistency		Correctness	
	V&V item	Comment	V&V item	Comment	V&V item	Comment
S/W function define	1	0	1	0	2	1
I/O variables define	25	5	3	1	5	2
S/W behavior define	15	3	1	0	5	3
Interface define	1	0	1	0	1	0
SRS (Requirements)	**Completeness**		**Consistency**		**Correctness**	
	V&V item	Comment	V&V item	Comment	V&V item	Comment
S/W function define	1	0	1	0	2	0
I/O variables define	25	10	3	2	5	1
S/W behavior define	15	4	1	0	5	1
Interface define	1	0	1	0	1	0

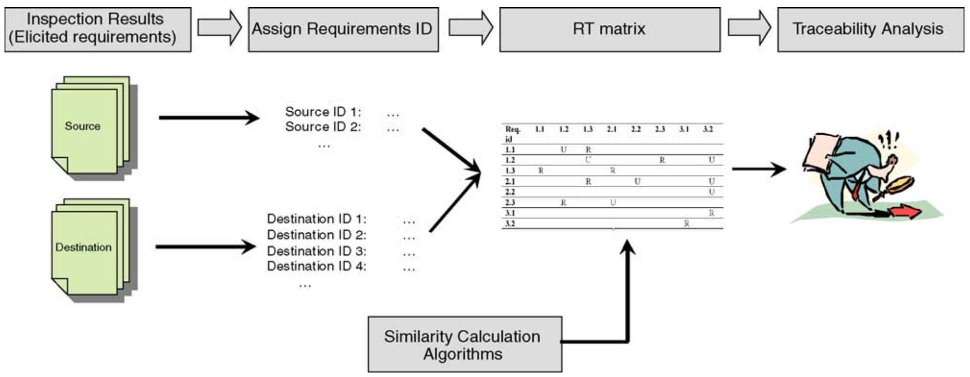


Fig. 4. Schematic diagram of requirements traceability.

the similarity of design documents written in English and Korean can be calculated for efficient traceability analysis.

In Step 2 of IE approach for requirements, ID number of requirements should be assigned to each requirement sentence elicited from Step 1. And then, in the RT matrix, the relation between the source requirements and the destination requirements should be described for requirements traceability analysis. Using this result of relation, we can analyze the traceability between documents. Fig. 4 shows a schematic diagram of requirements traceability.

In Fig. 4, similarity calculation algorithms support the analyzer to make a relation between documents in the RT matrix. Using the similarity calculation algorithms, we can represent the similarity as percentage in the RT matrix and the result of similarity will be helpful to the analyzer in traceability analysis. Table 2 shows the results of traceability analysis for the very early version of AMS. Based on the results of Step 1, there were 96 requirement sentences in FR document and there were 142 requirement sentences in SRS document. Among them, 60 requirement sentences of FR were reflected in SRS document and we could not find the relation about 21 requirement sentences of FR. On the other hand, we could not decide the relation about 15 requirement sentences of FR.

3.2. Requirements analysis in IE approach for requirements

The second stage in Fig. 3, which focuses on formal requirements analysis, enables an effective transition from a natural language specification into a formal specification in the requirements phase. Because of the difficulty of directly generating a formal specification from a natural language document, we need to extract useful information from the design documents. In this stage, our approach supports structure decomposition and analysis (Step 3) and NuSCR formal specification and analysis (Step 4).

3.2.1. Step 3: structure decomposition and analysis

Step 3 supports the structure decomposition and analysis of a system through an input-process-output type of structure. Based on the document evaluation in the first stage of IE approach for requirements, structural information such as inputs, functions, and outputs is extracted and formed directly into an input-process-output structure. This structural decomposition is useful for the user in analyzing the completeness of Input/Output variables and the consistency between system functions. Through the requirement analysis stage, we can therefore obtain a refined document based on structural decomposition, and this document is very useful for the analyzer in generating the formal specification. The formal methods that the analyzer uses for specifying the software requirements in Step 4 affect the type of structure found in the analysis.

Fig. 5 shows a schematic diagram of Step 3. After the review of each document in the first stage, structures of system is decomposed by input-process-output type and the user analyzes the structures for the increasing quality of system. Consequently, NuSCR formal specification in Step 4 can be started from this result of structure decomposition. Table 3 shows an example of structure decomposition for AMS.

3.2.2. Step 4: NuSCR formal specification and analysis

In the approach, Step 4 supports formal specification and analysis based on the NuSCR approach. In this work, we selected the NuSCR approach mentioned in Section 2 because it provides an environment for verifying

Table 2
Requirements traceability results of AMS

AMS	Number of requirement	Relation	No relation	Questionable
FR	96	60	21	15
SRS	142	N/A	N/A	N/A

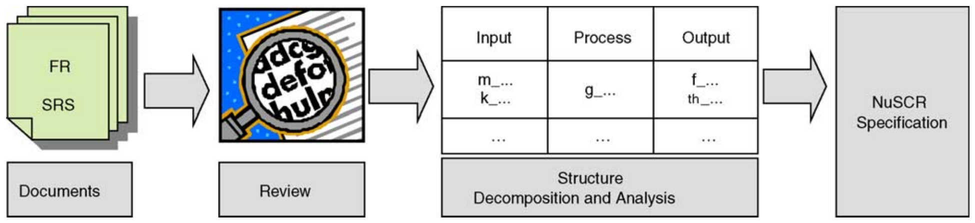


Fig. 5. Schematic diagram of structure decomposition and analysis.

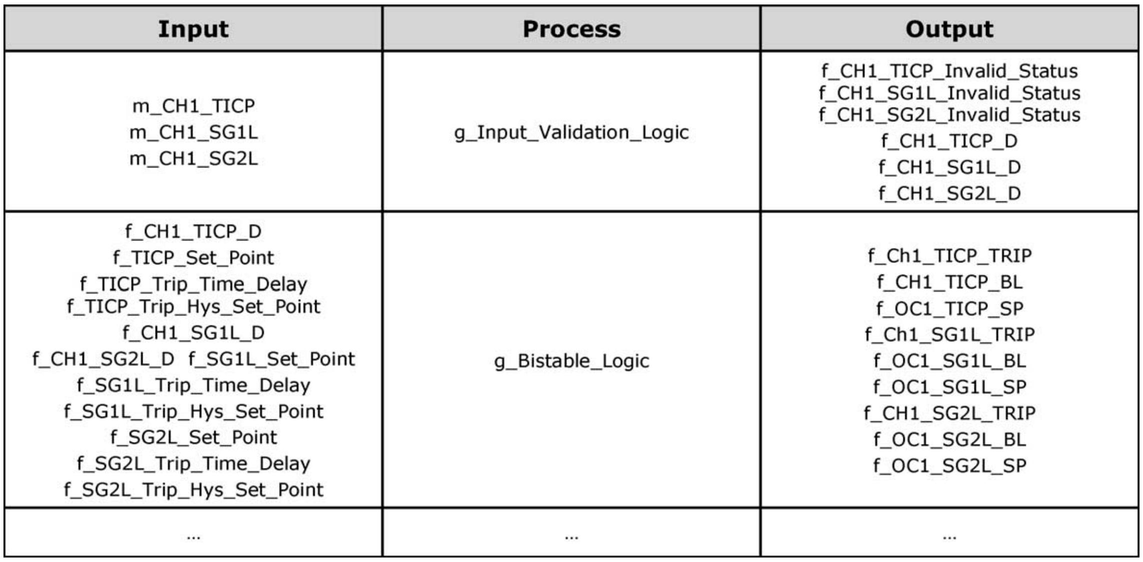
the functional requirements of a nuclear control system. Besides providing a specification approach to specifying requirements, it also provides a verification environment. Fig. 6 shows overview of the NuSCR approach. The NuSCR approach supports formal specifications based on FOD, SDT, FSM, and TTS and formal verifications using SMV and PVS.

For the NuSCR specification, basic information of input/output variables and functions should be imported from the result in Step 3. With the Step 3 input-process-output type of structure, we have successfully used the NuSCR approach to draw an FOD and an SDT.

Fig. 7 shows an example of NuSCR specification. Fig. 7(a) is a FOD for *g_Fixed_Setpoint_Rising_Trip_with_OB*, fixed set-point rising trip logic in BP (bistable processor) of RPS, where *g_* denotes the group prefix. Boxed nodes represent inputs and outputs. SDT, shown in Fig. 7(b), defines function variable *f_X_Valid* appearing in the FOD. If the value of *f_X* is between *k_X_MIN* and *k_X_MAX*, the output value *f_X Valid* is 0, indicating normal case. Otherwise output value is 1. NuSCR allows multiple and related terms be written together on the same row. That is, in the AECL-notation, one would have no option but to divide into two rows: *(f_X>=k_X_MIN)* and *(f_X<=k_X_MAX)*. This example is too trivial for developer to appreciate the difference in expressiveness. However, in the Wolsung SDS2, which was considerably simpler in complexity than KNICS RPS, the most complex SDT consisted of 16 rows and 12 columns because complex equations had to be decomposed into 'primitive' fragments. Domain experts repeatedly emphasized that mathematical equations used in trip logics, no matter how complex they are, are well-understood and proven-correct as a whole to domain experts and that they need not be artificially fragmented in the specification.

Fig. 7(c), TTS for *th_X_Trip*, shows how behavior of timed-history variable node is captured. It is interpreted as follows: 'If condition *f_X*≥*k_X_Trip_Setpoint* is satisfied in state *Normal*, it enters *Waiting* state. If the condition remains true for *k_Trip_Delay* period while in *Waiting* state, system generates the trip signal 0. If *f_X_Valid*, *f_Module_Error*, or *f_Channel_Error* occur, then trip signal is immediately produced. In the *Trip_By_Error* or *Trip_By_Logic* state , if the trip conditions are canceled, system returns to *Normal* state and the output 1 is generated.' The TTS expression in *Cond_b* [*k_Trip_Delay*, *k_Trip_Delay*] means that the condition must remain true for *k_Trip_Delay* unit times. In AECL-style notation, behavior related to time-dependent state transition was written in tabular notation, and domain experts preferred automata notation to tabular notation.

Table 3
An example of structure decomposition for AMS



Input	Process	Output
m_CH1_TICP m_CH1_SG1L m_CH1_SG2L	g_Input_Validation_Logic	f_CH1_TICP_Invalid_Status f_CH1_SG1L_Invalid_Status f_CH1_SG2L_Invalid_Status f_CH1_TICP_D f_CH1_SG1L_D f_CH1_SG2L_D
f_CH1_TICP_D f_TICP_Set_Point f_TICP_Trip_Time_Delay f_TICP_Trip_Hys_Set_Point f_CH1_SG1L_D f_CH1_SG2L_D f_SG1L_Set_Point f_SG1L_Trip_Time_Delay f_SG1L_Trip_Hys_Set_Point f_SG2L_Set_Point f_SG2L_Trip_Time_Delay f_SG2L_Trip_Hys_Set_Point	g_Bistable_Logic	f_Ch1_TICP_TRIP f_CH1_TICP_BL f_OC1_TICP_SP f_Ch1_SG1L_TRIP f_OC1_SG1L_BL f_OC1_SG1L_SP f_CH1_SG2L_TRIP f_OC1_SG2L_BL f_OC1_SG2L_SP
...	...	...

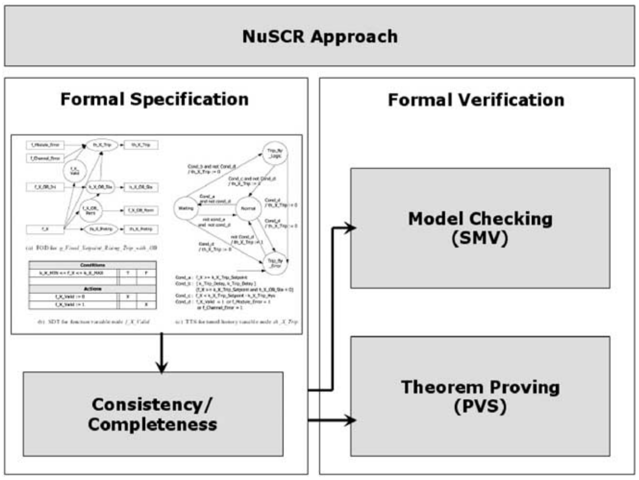


Fig. 6. Overview of the NuSCR approach.

Similarly, *h_X_OB_Sta*, shown in Fig. 7(a), is a history variable node defined as FSM. FSM is same as TTS except that time constraints are missing. All constructs in NuSCR, s.t. FOD, SDT, FSM, and TTS are familiar notations to domain engineers and software developers. NuSCR has been evaluated as being easy to specify and understand by domain engineers [20].

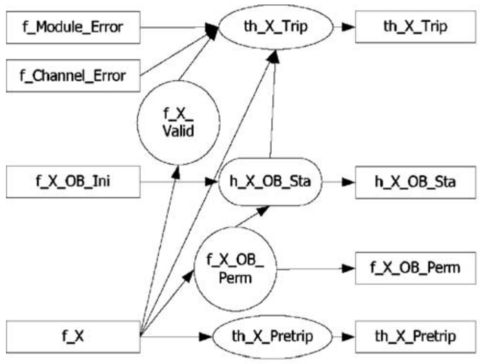
4. NuSISRT: an integrated tool for the software requirements analysis

In this section, we describe the NuSISRT, a computer-aided tool supporting software inspection and requirement traceability for nuclear engineering developed in this work. The NuSISRT totally supports the effective technique we proposed in this work for analyzing software requirements. By using the NuSISRT, we can straightforwardly conduct the phases of software requirement analysis based on IE approach for requirements described in Section 3 of this paper.

The NuSISRT comprises tools for document evaluation, traceability analysis, structural analysis and inspection meeting support. Designed to support the inspection of all software development documents, the NuSISRT is a PC-based application designed for anyone who needs to manage requirements. To support our approach systematically, the NuSISRT has three kinds of views: an inspection view, a traceability view, and a structure view. It also has a Web page for inspection meetings, though because of the similarity to general Web pages on the Internet it is not discussed here.

4.1. Inspection view

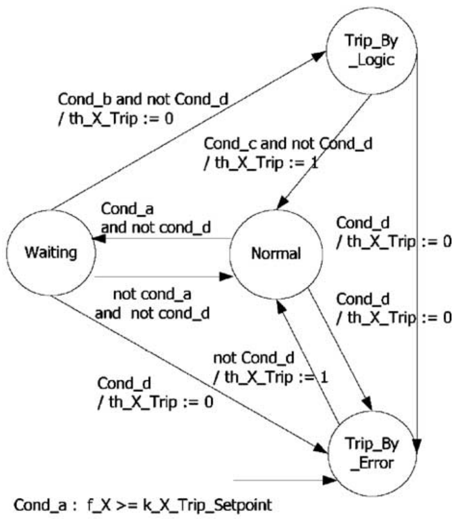
The support of document evaluation with the inspection view is a main function of the NuSISRT. It supports an elicitation function that reads a text file and copies paragraph numbers and requirement text to a NuSISRT file. It can read any text data that is convertible to '.txt' format. It also supports the manual addition of individual requirements and can import from various formats. The inspection view permits users to associate database items by defining attributes; the attributes attached to individual database items provide a powerful means of identifying subcategories or database items and of managing requirements.



(a) FOD forg*_Fixed_Setpoint_Rising_Trip_with_OB*

Conditions		
k_X_MIN <= f_X <= k_X_MAX	T	F
Actions		
f_X_Valid := 0	X	
f_X_Valid := 1		X

(b) SDT for function variable node *f_X_Valid*



Cond_a : f_X >= k_X_Trip_Setpoint
Cond_b : [k_Trip_Delay, k_Trip_Delay]
(f_X >= k_X_Trip_Setpoint and h_X_OB_Sta = 0)
Cond_c : f_X < k_X_Trip_Setpoint - k_X_Trip_Hys
Cond_d : f_X_Valid = 1 or f_Module_Error = 1
or f_Channel_Error = 1

(c) TTS for timed history variable node *th_X_Trip*

Fig. 7. NuSCR specification example.

Fig. 8 shows an example of the inspection view of the NuSISRT. The inspection view reads source documents, identifies requirements, and elicits main requirements for import into the database. The inspection view automatically finds and elicits requirements based on a set of keywords defined by the user. As the requirements are found, they are highlighted as shown in Fig. 8. The user can also manually select and identify requirements. The inspection view enables the production of an user-defined report that shows various types of inspection results. The user builds up the architecture of the desired reports on the right-hand window of Fig. 8. If the user writes down checklists in this window, the NuSISRT can directly support the software inspection with this functional window. The requirements to be found by the tool are located in a suitable checklist site using various arrow buttons in the window. In this way, each inspector can examine the requirements and generate the inspection result documents with the aid of the NuSISRT.

4.2. Traceability view

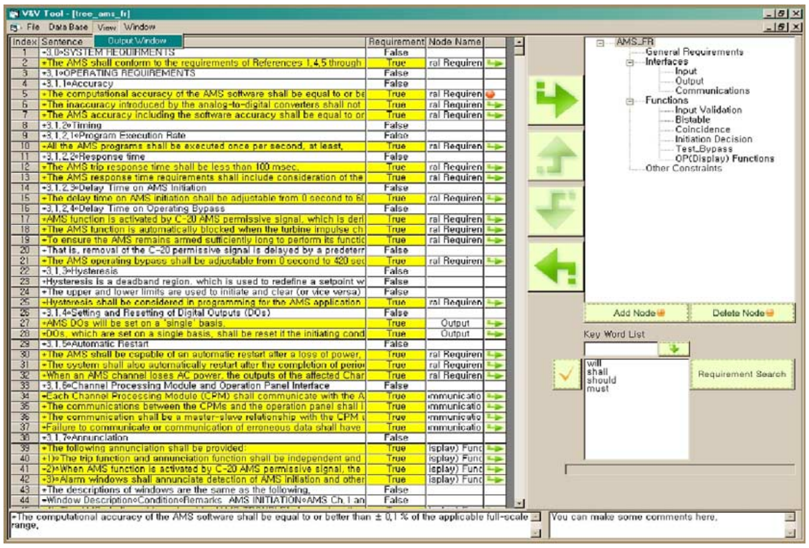
As mentioned, the NuSISRT supports normal parent-child links for managing requirements. Furthermore, it supports peer links between items in a database and general documents; the peer links provide an audit trail that shows compliance to quality standards or contractual conditions. This function is related to the requirement traceability analysis of our approach.

Fig. 9 shows an example of a traceability view. As shown in Fig. 9, the NuSISRT provides mechanisms that easily establish and analyze traceability through real-time visual notification of change in a matrix form. In Fig. 9, the column number of the matrix represents a requirement of the source file, and the row number of the matrix represents the destination file. The relationships between the source and the destination are expressed in a matrix window using linked chains and unlinked chains. The linked chains mean that source requirements are reflected in destination requirements. The unlinked chains indicate that source and destination requirements are changed. As a result, it is necessary to verify the change between source and destination documents. The question marks mean that the traceability between requirements is difficult to define. In such cases, another analyzer must verify the requirements.

As described in Section 3, we proposed a means of calculating the similarity between sentences to more easily support traceability analysis. The traceability view therefore supports the function of calculating the similarity between requirements based on each cosine vector similarity and the case grammar outlined in Section 2. The cosine vector similarity is for the analysis of English documents, and the case grammar is for the analysis of Korean documents. Through this function, the traceability view can automatically represent the similarity by percentage as shown in Fig. 10; this result is very helpful to the user and the analyzer. In this way, we can represent the traceability between documents for the support of Step 2 of our approach.

4.3. Structure view

For the translation of a formal specification, the NuSISRT supports Step 3 of our approach through the structure view. The structure view enables effective transition to the NuSRS, which is a tool for the NuSCR formal method. Fig. 11 shows an example of the structure view. Through the structure view, we can analyze design documents with respect to a system's structure. The results of analysis help generate a formal specification from a software requirement document written in a natural language. For structural analysis of systems, defining the input and output, along with the functions, is essential.



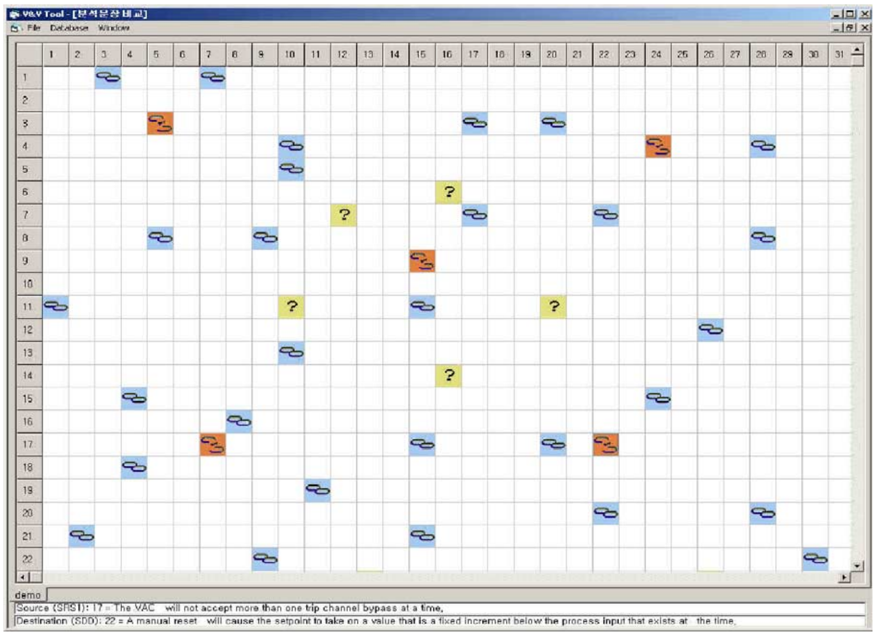
Index	Sentence	Requirement	Node Name	
1	•3.0•SYSTEM REQUIRMENTS	False		
2	•The AMS shall conform to the requirements of References 1,4,5 through	True	ral Requiren	➜
3	•3.1•OPERATING REQUIREMENTS	False		
4	•3.1.1•Accuracy	False		
5	•The computational accuracy of the AMS software shall be equal to or be	True	ral Requiren	●
6	•The inaccuracy introduced by the analog-to-digital converters shall not	True	ral Requiren	➜
7	•The AMS accuracy including the software accuracy shall be equal to or	True	ral Requiren	➜
8	•3.1.2•Timing	False		
9	•3.1.2.1•Program Execution Rate	False		
10	•All the AMS programs shall be executed once per second, at least,	True	ral Requiren	➜
11	•3.1.2.2•Response time	False		
12	•The AMS trip response time shall be less than 100 msec.	True	ral Requiren	➜
13	•The AMS response time requirements shall include consideration of the	True	ral Requiren	➜
14	•3.1.2.3•Delay Time on AMS Initiation	False		
15	•The delay time on AMS initiation shall be adjustable from 0 second to 6(	True	ral Requiren	➜
16	•3.1.2.4•Delay Time on Operating Bypass	False		
17	•AMS function is activated by C-20 AMS permissive signal, which is deri	True	ral Requiren	➜
18	•The AMS function is automatically blocked when the turbine impulse ch	True	ral Requiren	➜
19	•To ensure the AMS remains armed sufficiently long to perform its functic	True	ral Requiren	➜
20	•That is, removal of the C-20 permissive signal is delayed by a predeterr	False		
21	•The AMS operating bypass shall be adjustable from 0 second to 420 sec	True	ral Requiren	➜
22	•3.1.3•Hysteresis	False		
23	•Hysteresis is a deadband region, which is used to redefine a setpoint w	False		
24	•The upper and lower limits are used to initiate and clear (or vice versa)	False		
25	•Hysteresis shall be considered in programming for the AMS application	True	ral Requiren	➜
26	•3.1.4•Setting and Resetting of Digital Outputs (DOs)	False		
27	•AMS DOs will be set on a 'single' basis.	True	Output	➜
28	•DOs, which are set on a single basis, shall be reset if the initiating cond	True	Output	➜
29	•3.1.5•Automatic Restart	False		
30	•The AMS shall be capable of an automatic restart after a loss of power,	True	ral Requiren	➜
31	•The system shall also automatically restart after the completion of perio	True	ral Requiren	➜
32	•When an AMS channel loses AC power, the outputs of the affected Char	True	ral Requiren	➜
33	•3.1.6•Channel Processing Module and Operation Panel Interface	False		
34	•Each Channel Processing Module (CPM) shall communicate with the A	True	mmunicatio	➜
35	•The communications between the CPMs and the operation panel shall l	True	mmunicatio	➜
36	•The communication shall be a master-slave relationship with the CPM	True	mmunicatio	➜
37	•Failure to communicate or communication of erroneous data shall have	True	mmunicatio	➜
38	•3.1.7•Annunciation	False		
39	•The following annunciation shall be provided:	True	isplay) Func	➜
40	•1)•The trip function and annunciation function shall be independent and	True	isplay) Func	➜
41	•2)•When AMS function is activated by C-20 AMS permissive signal, the	True	isplay) Func	➜
42	•3)•Alarm windows shall annunciate detection of AMS initiation and other	True	isplay) Func	➜
43	•The descriptions of windows are the same as the following,	False		
44	•Window Description•Condition•Remarks AMS INITIATION•AMS Ch.1 an	False		

•The computational accuracy of the AMS software shall be equal to or better than ± 0.1 % of the applicable full-scale range,

will
shall
should
must

You can make some comments here.

Fig. 8. Inspection view of the NuSISRT.



Source (SRS1): 17 = The VAC will not accept more than one trip channel bypass at a time.

Destination (SDD): 22 = A manual reset will cause the setpoint to take on a value that is a fixed increment below the process input that exists at the time.

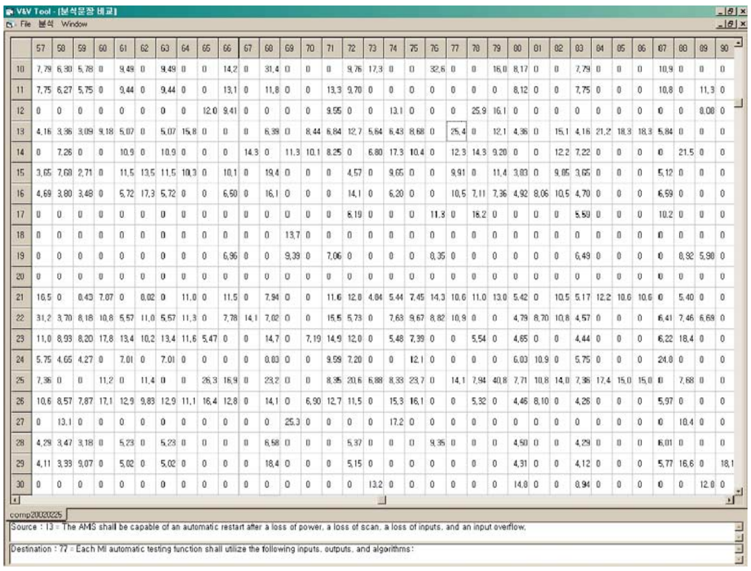
Fig. 9. Traceability view of the NuSISRT tool.

We therefore proposed in this work the input-process-output type of structure. In the structure view, several tabular forms help the user easily build up an input-process-output structure. Such a structure is represented in the right-hand window of the structure view as a tree type.

After structural analysis, the structure view generates a result file written in XML language and then it is transferred to the NuSRS. With this XML file, the FOD can be drawn automatically in the NuSRS. Fig. 12 shows an example of the NuSRS. The NuSRS, which is used in Step 4 of our approach, is described in Section 3. Using the NuSRS, a formal requirement specification can be generated in the software requirement-phase; the specification can then be helpful for formally analyzing the system requirements. The NuSRS is a platform independent tool made with JAVA for formally specifying the SRS of a nuclear system. It provides an environment for drawing the FOD and the SDT and allows automata diagrams to be built from the nodes of the FOD.

5. Conclusions

In this work, we propose an IE approach for requirements which is an effective technique for the software requirements analysis; the technique enables easy inspection by



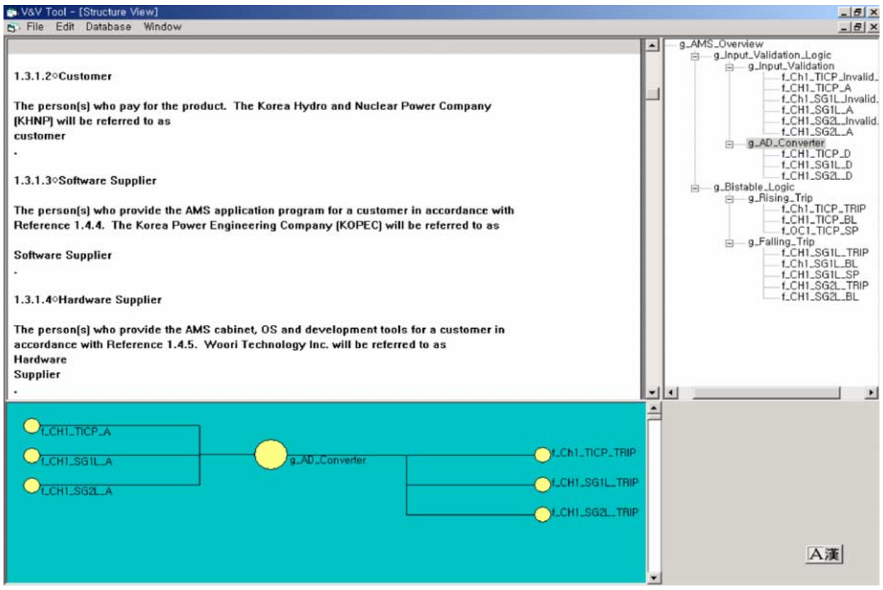
	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73
10	7,79	6,30	5,78	0	9,49	0	9,49	0	0	14,2	0	31,4	0	0	0	9,76	17,3
11	7,75	6,27	5,75	0	9,44	0	9,44	0	0	13,1	0	11,8	0	0	13,3	9,70	0
12	0	0	0	0	0	0	0	0	12.0	9.41	0	0	0	0	9.55	0	0
13	4,16	3,36	3,09	9,18	5,07	0	5,07	15,8	0	0	0	6,39	0	8,44	6,84	12,7	5,64
14	0	7.26	0	0	10.9	0	10.9	0	0	0	14.3	0	11.3	10.1	8.25	0	6.80
15	3,65	7,68	2,71	0	11,5	13,5	11,5	10,3	0	10,1	0	19,4	0	0	0	4,57	0
16	4,69	3,80	3,48	0	5,72	17,3	5,72	0	0	6,50	0	16,1	0	0	0	14,1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6,19	0
18	0	0	0	0	0	0	0	0	0	0	0	0	13,7	0	0	0	0
19	0	0	0	0	0	0	0	0	0	6,96	0	0	9,39	0	7,06	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	16,5	0	0,43	7,07	0	8,02	0	11,0	0	11,5	0	7,94	0	0	11,6	12,0	4,04
22	31,2	3,70	8,18	10,8	5,57	11,0	5,57	11,3	0	7,78	14,1	7,02	0	0	15,5	5,73	0
23	11,0	8,93	8,20	17,8	13,4	10,2	13,4	11,6	5,47	0	0	14,7	0	7,19	14,9	12,0	0
24	5.75	4,65	4.27	0	7.01	0	7.01	0	0	0	0	8.03	0	0	9.59	7.20	0
25	7,36	0	0	11,2	0	11,4	0	0	26,3	16,9	0	23,2	0	0	8,35	20,6	6,88
26	10,6	8,57	7,87	17,1	12,9	9,83	12,9	11,1	16,4	12,8	0	14,1	0	6,90	12,7	11,5	0
27	0	13.1	0	0	0	0	0	0	0	0	0	0	25.3	0	0	0	0
28	4,29	3,47	3,18	0	5,23	0	5,23	0	0	0	0	6,58	0	0	0	5,37	0
29	4,11	3,33	9,07	0	5,02	0	5,02	0	0	0	0	18,4	0	0	0	5,15	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13.2

	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
10	0	0	32,6	0	0	16,0	8,17	0	0	7,79	0	0	0	10,9	0	0	0
11	0	0	0	0	0	0	8,12	0	0	7,75	0	0	0	10,8	0	11,3	0
12	13.1	0	0	0	25.9	16.1	0	0	0	0	0	0	0	0	0	8.08	0
13	6,43	8,68	0	25,4	0	12,1	4,36	0	15,1	4,16	21,2	18,3	18,3	5,84	0	0	0
14	17.3	10.4	0	12.3	14.3	9.20	0	0	12.2	7.22	0	0	0	0	21.5	0	0
15	9,65	0	0	9,91	0	11,4	3,83	0	9,85	3,65	0	0	0	5,12	0	0	0
16	6,20	0	0	10,5	7,11	7,36	4,92	8,06	10,5	4,70	0	0	0	6,59	0	0	0
17	0	0	11,3	0	16,2	0	0	0	0	5,59	0	0	0	10,2	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	8,35	0	0	0	0	0	0	6,49	0	0	0	0	8,92	5,90	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	5,44	7,45	14,3	10,6	11,0	13,0	5,42	0	10,5	5,17	12,2	10,6	10,6	0	5,40	0	0
22	7,63	9,67	8,82	10,9	0	0	4,79	8,70	10,8	4,57	0	0	0	6,41	7,46	6,69	0
23	5,48	7,39	0	0	5,54	0	4,65	0	0	4,44	0	0	0	6,22	18,4	0	0
24	0	12.1	0	0	0	0	6.03	10.9	0	5.75	0	0	0	24.8	0	0	0
25	8,33	23,7	0	14,1	7,94	40,8	7,71	10,8	14,0	7,36	17,4	15,0	15,0	0	7,68	0	0
26	15,3	16,1	0	0	5,32	0	4,46	8,10	0	4,26	0	0	0	5,97	0	0	0
27	17.2	0	0	0	0	0	0	0	0	0	0	0	0	0	10.4	0	0
28	0	0	9,35	0	0	0	4,50	0	0	4,29	0	0	0	6,01	0	0	0
29	0	0	0	0	0	0	4,31	0	0	4,12	0	0	0	5,77	16,6	0	18,1
30	0	0	0	0	0	0	14.8	0	0	0.94	0	0	0	0	0	12.0	0

Source : 13 = The AMS shall be capable of an automatic restart after a loss of power, a loss of scan, a loss of inputs, and an input overflow.

Destination : 77 = Each MI automatic testing function shall utilize the following inputs, outputs, and algorithms:

Fig. 10. An example of similarity calculation.


V&V Tool - [Structure View]

1.3.1.2○Customer

The person(s) who pay for the product. The Korea Hydro and Nuclear Power Company (KHNP) will be referred to as customer
.

1.3.1.3○Software Supplier

The person(s) who provide the AMS application program for a customer in accordance with Reference 1.4.4. The Korea Power Engineering Company (KOPEC) will be referred to as

Software Supplier
.

1.3.1.4○Hardware Supplier

The person(s) who provide the AMS cabinet, OS and development tools for a customer in accordance with Reference 1.4.5. Woori Technology Inc. will be referred to as
Hardware
Supplier
.

Fig. 11. Structure view of the NuSISRT.

combining requirement traceability and effective use of a formal method. Our approach consists of two stages for effective V&V tasks: document evaluation and requirements analysis. In the document evaluation stage, there are two kinds of steps: Step 1, the inspection step; and Step 2, the traceability step. For formal analysis of the requirements, our approach suggests two kinds of steps: Step 3, the structural decomposition and analysis step; and Step 4, the NuSCR formal specification and analysis step. These steps are based on V&V tasks for safety-critical systems.

To support the approach in this work, we developed the NuSISRT which is computer-aided tool supporting software inspection and requirement traceability for nuclear engineering. The NuSISRT systematically supports software inspection and has the capability of analyzing requirement traceability. We also used the NuSRS, which is another tool for the NuSCR formal method. The NuSRS enables not only formal specification but also formal analysis.

Through our effective technique for the software requirements analysis, we can minimize some of the difficulties caused by the difference in domain knowledge between the designer and analyzer. The tools for our approach can also enhance V&V in the software requirement-phase. In the future, we plan to extend our approach to the next phases of the software life cycle: the design phase

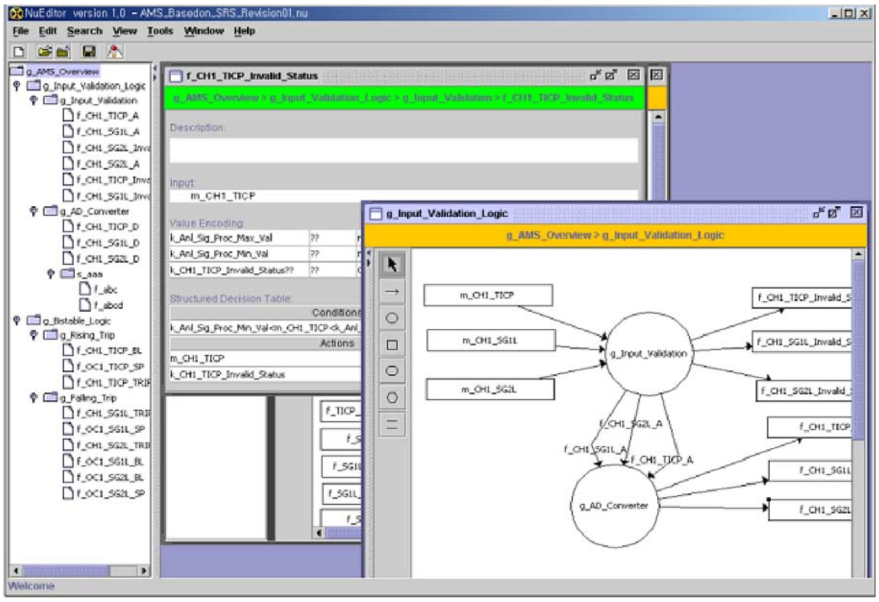


Fig. 12. The NuSRS.

and the implementation phase. In this way, we expect to develop an integrated environment throughout the life cycle for safety-critical systems.

Acknowledgements

This research was supported by KNICS (Korea Nuclear I&C System) R&D center and NRL project.

References

[1] Harel D. Statecharts: a visual formalism for complex systems. Sci Comput Program 1987;8:231–74.

[2] Jensen K. Coloured petri nets: basic concepts, analysis methods and practical use, 2nd Ed., vol. 1. Berlin: Springer; 1997.

[3] Leveson NG, Heimdahl MPE, Hildreth H, Reese JD. Requirements specification for process-control systems. IEEE Trans Softw Eng 1994;20(9).

[4] Heitmeyer C, Labaw B. Consistency checking of SCR-style requirements specification. International symposium on requirements engineering; March 1995.

[5] Cho J, Yoo J, Cha S. NuEditor—a tool suite for specification and verification of NuSCR. Second ACIS international conference on software engineering research, management and applications (SERA2004); 2004. p. 298–304.

[6] Fagan ME. Design and code inspections to reduce errors in program development. IBM Syst J 1976;15(3):182–211.

[7] Luhn HP. A statistical approach to the mechanized encoding and searching of literary information. IBM J Res Dev 1957;1(4):309–17.

[8] Yoo Y-J, Seong PH, Kim MC. Development of a traceability analysis method based on case grammar for NPP requirement documents written in Korean language. J Korean Nucl Soc 2004; 36(4):307–15.

[9] Schouwen Van AJ, Panas D, Madey J. Documentation of requirements for computer systems. In: Proceedings of IEEE international symposium on requirements engineering; 1993. p. 198–207.

[10] Wolsong NPP2/3/4. Software requirements specification for shutdown systems (SDS) 2 PDC, 86-68350-SRS-001; June 1993.

[11] KNICS (Korea Nuclear Instrumentation and Control System Research and Development Center) [http://www.knics.re.kr].

[12] Yoo J, Kim T, Cha S, Lee J, Son HS. A formal software requirements specification method for digital nuclear plants protection systems. J Syst Softw 2005;74(1):73–83.

[13] Henzinger TA, Manna Z, Pnueli A. Timed transition systems. In: Proceedings of REX Workshop; 1991. p. 226–51.

[14] Kim T, Cha S. Automatic structural analysis of SCR-style software requirements specifications using PVS. J Softw Test, Verif Reliab 2001;11(3):143–63.

[15] Kim T, Stringer-Calvert D, Cha S. Formal verification of functional properties of an SCR-style software requirements specification using PVS. Reliab Eng Syst Safety 2004 [in press].

[16] McMillan KL. Symbolic model checking. Dordrecht: Kluwer Academic Publishers; 1993.

[17] IEEE. IEEE standard 1012-1998 for software verification and validation. An American national standard; 1998.

[18] KOPEC. Functional requirements for ATWS mitigation system for KORI NPP UNIT 1; 2001.

[19] KOPEC. Software requirements specification for ATWS mitigation system for KORI NPP UNIT 1; 2001.

[20] Yoo J, Cha S, Kim CH, Oh Y. Formal software requirements specification for digital reactor protection systems. J KISS 2004; 31(6):750–9.