"3+3 PROCESS" FOR SAFETY CRITICAL SOFTWARE FOR I&C SYSTEM IN NUCLEAR POWER PLANTS

JAE-CHEON JUNG*, HOON-SUN CHANG and HANG-BAE KIM
NSSS Engineering and Development Division, Korea Power Engineering Company
150 Duckjin-dong, Yuseong-ku, Daejeon-city, 305-303, Korea
*Corresponding author. E-mail : jcjung@kopec.co.kr

Received May 23, 2008
Accepted for Publication October 9, 2008

The "3+3 Process" for safety critical software for nuclear power plants' I&C (Instrumentation and Control system) has been developed in this work. The main idea of the "3+3 Process" is both to simplify the software development and safety analysis in three steps to fulfill the requirements of a software safety plan [1]. The "3-Step" software development process consists of formal modeling and simulation, automated code generation and coverage analysis between the model and the generated source codes. The "3-Step" safety analysis consists of HAZOP (hazard and operability analysis), FTA (fault tree analysis), and DV (design validation). Put together, these steps are called the "3+3 Process". This scheme of development and safety analysis minimizes the V&V work while increasing the safety and reliability of the software product. For assessment of this process, validation has been done through prototyping of the SDS (safety shut-down system) #1 for PHWR (Pressurized Heavy Water Reactor).

KEYWORDS : Safety Critical Software, Validation and Verification, Safety Analysis, Model Driven Architecture

1. INTRODUCTION

KINS (Korea Institute of Nuclear Safety), the nuclear regulatory agency in Korea, defines safety critical software for the NPP (nuclear power plant) as software that is used either in the reactor protection system or engineered safety actuation circuitry.

The safety critical software must meet strict technical standards, and developers must adopt engineering practices in compliance with its safety level [2]. IEEE Std. 1012 adopts a software integrity level approach to classify software criticality as a range of values that represent software complexity, criticality, risk, safety level, security level, desired performance, reliability, or other characteristics unique to the project [3]. IEC Std. 61508-5 defines safety integrity as the probability of a safety-related system satisfactorily performing the required safety functions under the stated conditions within a stated period of time [4].

Therefore, developing a safety critical I&C (instrument and control) system for the NPP means building a system that satisfies the highest level of safety standards.

Unlike hardware, the cause of a software failure is more vulnerable to human cognitive errors caused by either an error of omission, an error of commission or an operational error [5]. These errors adversely affect either the software environment or result in the production of ambiguous specification.

According to the Standish group report in 2001, 49% of software projects are cancelled before completion or are never implemented. In addition, 23% are completed and become operational, but suffer from budget over-run and project schedule delays and include fewer features and functions than initially specified [6].

Misunderstanding of requirements due to lack of communication between the design engineer and verification engineer causes unexpected results such as functionality errors of the system.

IEEE Std. 1228 defines specific contents of the software safety plan, including safety goals [1]. To satisfy this plan, a structured model has been used. However, the structured model is too complicated to manage the software life cycle because;

- Too many documents are required for each phase of development,
- Requirements analysis is a lengthy and tedious job,
- And it is difficult to fully understand detailed specifications due to a complicated work structure.

Complexity is a major obstacle to safety. Complex software is more likely to contain design errors. Software errors can't be identified by system testing alone. Experience has shown that the use of a formal,

deterministic and automated solution reduces cost and improves quality [7].

Software safety analysis using a structured process is too complicated to implement. In addition, this process depends upon the expertise of the analyst. Generally, safety analysis is performed by a third-party and the quality of the analyst directly affects the quality of the analysis result.

Considering the vulnerability of software failures due to complexity, we developed the "3+3 Process" for the safety critical software.

2. "3+3 PROCESS" FOR SAFETY CRITICAL SOFTWARE DEVELOPMENT

The main idea of the "3+3 Process" is to simplify both the software development and safety analysis in three steps to fulfill the requirements of IEEE Std. 1228.

"3-Step" software development process consists of;

- formal modeling and simulation
- automated code generation
- model coverage analysis

"3-Step" safety analysis consists of;

- HAZOP (hazard and operability analysis)
- FTA (fault tree analysis)
- DV(design validation)

These steps put together are called the "3+3 Process". Figure 1 shows the overall scheme of the "3+3 Process".

The system requirements are described in a natural language. High level requirements in system requirements specification are formalized using a formal model. The high level requirements that can not be directly obtained by formalization of the system requirements are manually coded and defined as a dedicated node of the formal model. The formal model and the high level requirements are linked for traceability analysis. This model is simulated concurrently to ensure that it contains the intended function.

Through the coverage analysis, the completeness of coverage is checked to ensure that inactive parts of models are not included.

Code generators automatically generate C code, implementing the requirements and architecture defined in the formal model.

Requirements specification is used for HAZOP analysis as well. HAZOP is used to define the safety constraint or corner bugs in the requirements specification.

FTA identifies the root cause of the undesired event using HAZOP results.

Through design validation (DV), the safety constraints of the software requirements specification are proven mathematically to determine if the situation to be avoided will not occur [7].

If the generated code is validated through DV, the code is integrated with target hardware and a verification test and reliability test are performed for system acceptance.

The "3+3 Process" complies with Regulatory Guide 1.173/ IEEE Std. 1074, software life cycle documentation requirements[8][9]. Documents are automatically generated from a formal model.

Through automatic document generation, the following benefits result in reduction of the software phase documents, increased understandability of user needs and reduction of cost and man-power for development.

This scheme of development and safety analysis provides benefits such as increased safety and reliability

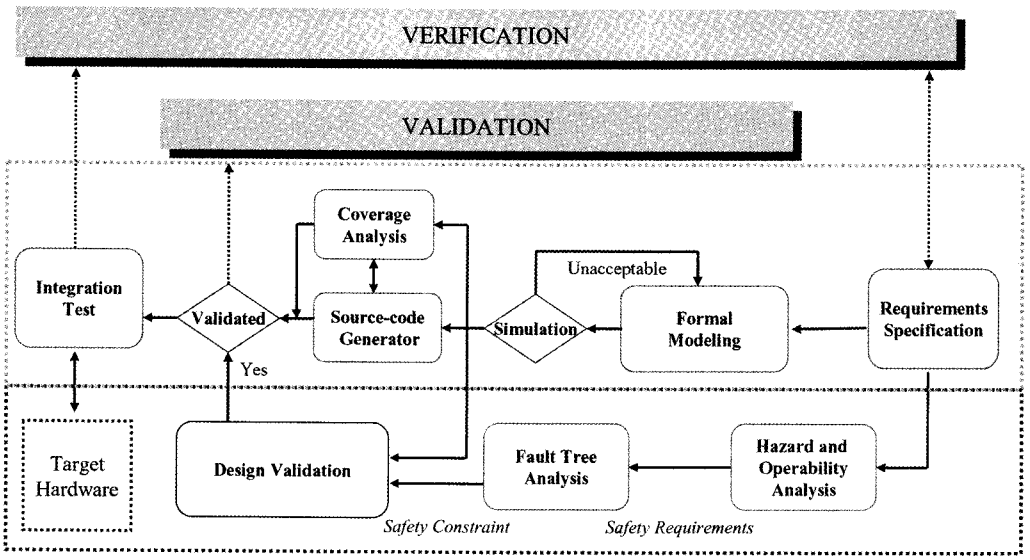


Fig. 1. Overall Scheme of the "3+3 Process"

of software products, requirements error reduction, V&V work load decrease, as well as coding work saving in the development process. Moreover, it saves a lot of safety analysis process by way of simplifying the complicated process into a "3- Step" process.

2.1 "3-Step" Software Development Process

Historically, the following issues have been raised in software development;

- Testing always starts late, compounded by the fact that massive rework is typically necessary after the first tests.
- The correct analysis of errors is very difficult due to the fact that during testing the software is embedded into the entire target system.
- Test coverage measurement is very complicated, and complete coverage may be nearly impossible to predict [7].

Safety critical software shall comply with such characteristics as modularity and simplicity, minimizing the number of subroutines and excluding the interrupt routine. In addition, the crosslink fault and erroneous function shall be eliminated. Ease of repair work after installation shall be achieved as well [10].

In regard to the issues stated above, the "3-Step" software development process consists of formal modeling and simulation, automated code generation and coverage analysis of the model.

In this process, requirements and design specifications are written in the formal model.

The formal method relies on the use of unambiguous formalisms for specifying systems. It brings a description of a system using fixed notation based on mathematics [5] [11]. Therefore, it ensures detecting and preventing incompleteness, inconsistencies and incorrectness of the software [5] in the design phase. Moreover, it allows development of larger and more complex software systems.

The formal model is easy to understand. Therefore, rigorous checking is possible, which can eliminate human errors [11]. Two specification formalisms; block diagrams for continuous control and state machines for discrete control, are used. This reduces the pain in splitting the logic of the application because of different support for algorithm and control.

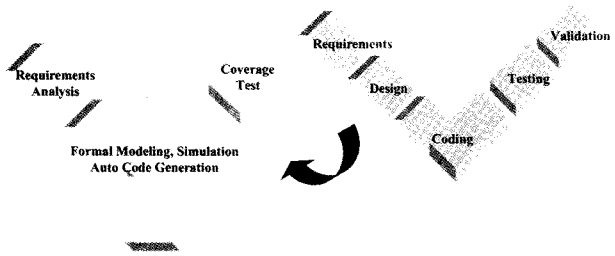


Fig. 2. Modification of the Development Process by Adapting the "3-Step" Process (V-Cycle Transforms to the Y-Cycle)

In addition, specification and design error can be corrected simultaneously with modeling and simulation. The formal model is exercised using its function and operability by model simulation.

The simulation checks specification integrity at the model level. It allows the developer to verify the semantics, data type, data dependencies and cycle detection [12].

Automatic code generation ensures that what is validated on the model is verified on the source code. The safety of the generated code is ensured by rigorous constraints; no pointer arithmetic, no dynamic memory allocation, no operating system call, loops only for delays, and so on. [12].

Unintended functions in traditional processes are eliminated by the coverage analysis. Coverage analysis shows how thoroughly the formal model has been translated into source code. In addition, it reveals inadequacies in system requirements.

The major impact on the software development cycle by adapting the "3-Step" software development process is to simplify and shorten the usual development and validation cycle.

The classical software development cycle is called the V cycle. Development flows down from requirements to coding, while validation flows up from source code unit tests to validation tests. The down stream steps of the V cycle are particularly difficult and expensive.

This V cycle can be transformed to the Y cycle as shown in Figure 2 using the "3+3 Process". Validation can concentrate on the highest level of requirements in the Y cycle. So, the performance analysis for the lower part of the development cycle is made easier.

The big savings in the Y cycle in particular come from automated code generation that simplifies coding and testing steps. It is one of the most expensive parts of the whole development cycle. According to the projection of Airbus, around 30% of development time savings can be achieved when a Y cycle is adopted [13].

2.2 "3-Step" Safety Analysis

As shown in Figure 3, "3-Step" safety analysis consists of HAZOP (hazard and operability analysis), FTA (fault tree analysis) and DV (design validation). HAZOP and FTA are performed respectively in the requirements phase and design phase. In the testing phase, design validation is done with a formal verification tool.

By adopting the "3-Step" safety analysis, tasks and output documents are reduced.

HAZOP provides an interdisciplinary analysis of system failures, both accidental and intentional, and the severity of the associated consequences [5]. HAZOP works from the fault both forwards to possible consequences and backwards to possible causes [14]. In HAZOP, a series of "guide words" is used to define particular types of deviation such as: no, more, less, as well as, part of, reverse, and other than. HAZOP results are interfaced

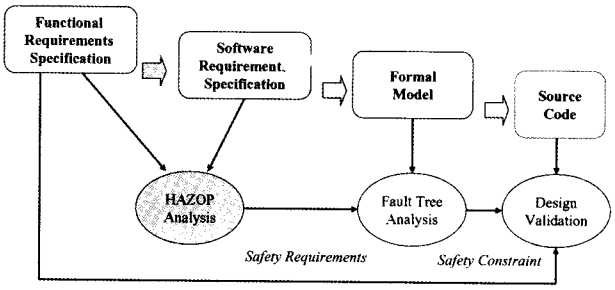


Fig. 3. Proposed "3-Step" Safety Analysis Process

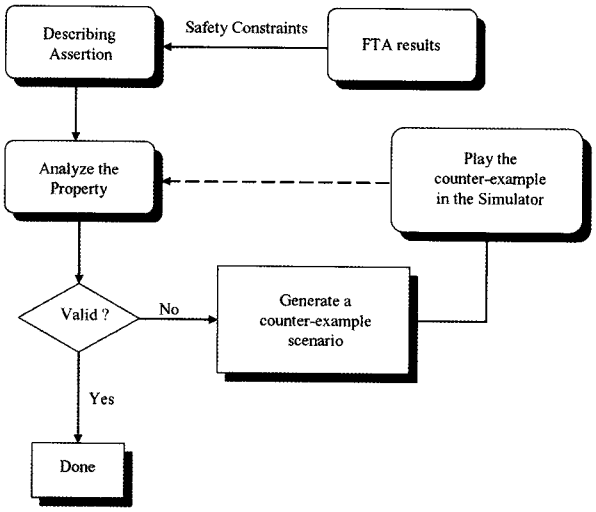


Fig. 4. Scheme of DV Cycle

with FTA to detect corner bugs or prove safety properties. The success or failure of the HAZOP depends on several factors [14];

- The completeness and accuracy of drawings and other data used as a basis for the study
- The technical skills and insights of the team
- The ability of the team to use the approach as an aid to their imagination in visualizing deviations, causes, and consequences
- The ability of the team to concentrate on the more serious hazards which are identified.

FTA is a deductive, top-down method of analyzing system design and performance. It identifies the root cause of a major undesired event [5]. FTA works back from consequences to causes that will lead to a hazard [14].

Figure 4 shows the scheme of the DV (design validation) cycle. DV increases the reliability and quality of designs by checking the safety requirements. It starts with describing the safety properties from the FTA result. DV determines mathematically if the situation to be avoided will not occur in the system [7]. The safety property is expressed as a truth table to validate whether the safety property is satisfied. If valid, the DV activity is finished. Otherwise, a counter example scenario is generated and plugged into the simulator. Then, the property analysis is performed until the counter-example is valid.

3. DEVELOPMENT OF SDS #1 USING THE "3+3 PROCESS"

In order to verify that the "3+3 Process" can be implemented on the safety critical I&C system for the NPP, the SDS#1(number one safety shutdown system)

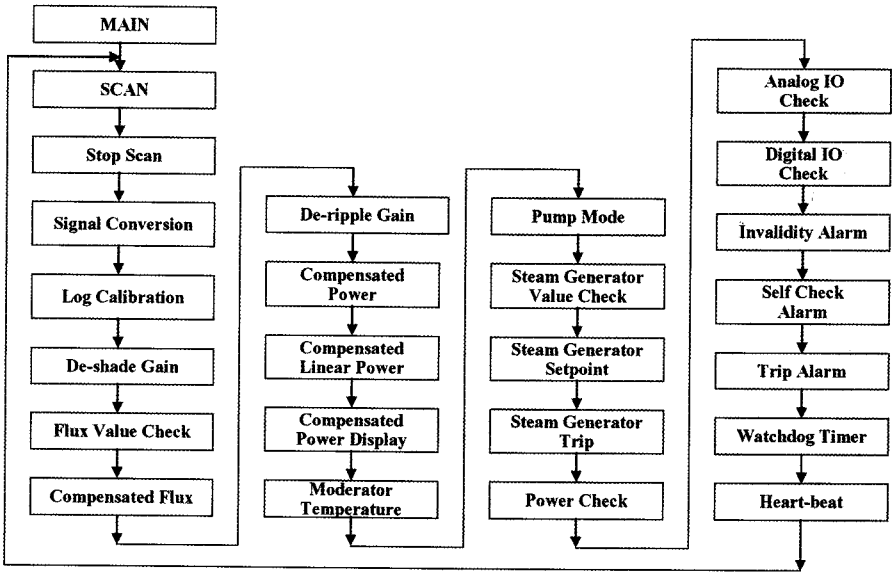
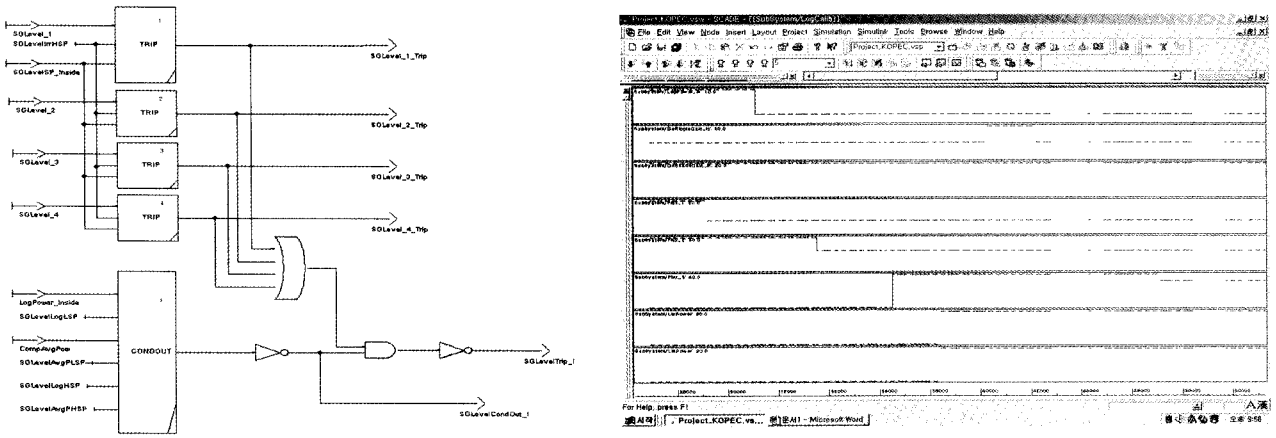


Fig. 5. Modularized Algorithm Node of the SDS#1 System



(a) (b)

Fig. 6. Modeling and Simulation Result of Steam Generator Level Trip Node

was developed. The safety shutdown system has similar functions to the RPS (reactor protection system) for PWR (pressurized water reactor).

The algorithm nodes as shown in Figure 5 are modeled, simulated, code-generated and validated. However, the hardware dependent nodes such as scan, stop scan, and heart-beat node are not implemented in the modeling. Assessment criteria of the developed system are: time for development and validation, accuracy in dealing with real time requirements, accuracy of generated source-code.

As a formal model and simulation as well as auto code-generator, SCADE® from Esterel technologies is used. For system modeling, the data flow diagram for the continuous function and the safety-state machine for the discrete function are used.

Using the formal modeling and simulation method, faults due to the requirements can easily be eliminated. SCADE® generates platform independent ANSI C code.

In Figure 6, the modeling and simulation result of the steam generator level trip node is shown.

Figure 6 (a) shows a model of the steam generator low level trip node. This node consists of four (4) trip modules. An individual trip module receives a steam generator level (SG Level) signal, high setpoint (SG Level IrrHSP) and inside setpoint (SG LevelSP_Inside). If one SG Level trip (TRIP) signal is triggered while the reactor power condition is above the setpoint (CONDOUT), a reactor trip is initiated. Figure (b) shows the simulation results using static test cases.

As shown in Figure 7, the path coverage is checked using the SCADE® coverage analysis tool. A set of test cases is created from the system requirements allocated to software. For each requirement, normal cases and robustness cases are considered.

After the coverage test, the ANSI-C code is generated using the qualified code generator. Generated source code is validated using a coverage analysis tool. In Figure 8,

2. Coverage Information

2.1. Overview

Coverage percentage of the project: **100.00 %**

2.2. Operator Instance Coverage

Instance Path	Coverage
SG_LevelTrip	59/59
SG_LevelTrip/CONDOUT 1	19/19
SG_LevelTrip/CONDOUT 1/RSR 1	5/5
SG_LevelTrip/TRIP 1	7/7
SG_LevelTrip/TRIP 2	7/7
SG_LevelTrip/TRIP 3	7/7
SG_LevelTrip/TRIP 4	7/7

Fig. 7. Coverage Analysis Result (Steam Generator Level Trip Node)

```
#define MUX3 MUX3_Orig
#include "MUX3.c"
#undef MUX3
void MTC_MUX3 ( real Dflt, real Sig2, bool Sel2, real Sig3, bool Sel3, real* Output1, bool*
MTC_BothTrue, bool* MTC_BothFalse, bool* MTC_Sel2TrueSel3False )
{
        MUX3_Orig ( Dflt, Sig2, Sel2, Sig3, Sel3, Output1 );
        *MTC_BothTrue = Sel2 && Sel3;
        *MTC_BothFalse = (!Sel2) && (!Sel3);
        *MTC_Sel2TrueSel3False = (Sel2) && (!Sel3);
}
```

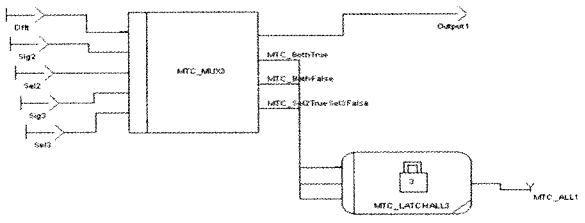


Fig. 8. Coverage Analysis for the Auto-Generated Source-Code

the upper part is generated C code. This code is assigned as an algorithm node, MTC_MUX3. Then, coverage analysis is performed in order to ensure that the generated

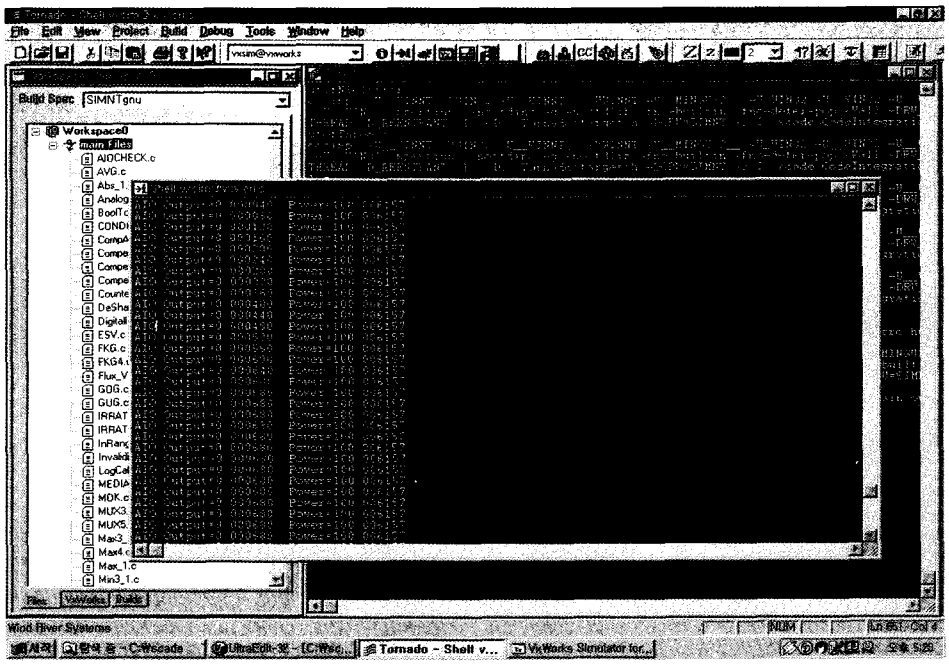


Fig. 9. Hyper Terminal Capture of Compilation Runs

source code satisfies its intended function.

The generated code is compiled in Vxworks® RTOS (real time operating system). As shown in Figure 9, the code is built well and it works as intended.

4. VERIFICATION OF "3-STEP" SAFETY ANALYSIS PROCESS

Table 1 shows the results of software HAZOP

Table 1. HAZOP Analysis Based on Guide Phrases

Hazard Description (guide phrases)	Hazard Cause	Detection Method	Potential Consequences	Hazard Impact	Hazard Mitigation	Safety Hazard Verification Method
Calculated result is outside acceptable range bounds (too low or high)	Programming Error	Visual Inspection, Periodic Tests	Channel Trip, Channel Alarm	Undetected problem in channel could give misleading indication	channel redundancy, channel trip	Software testing, code inspection
Function is not carried out as specified	Programming Error	Visual Inspection, WDT actuation	Channel Trip, Channel Alarm	Undetected problem in channel could give misleading indication	channel redundancy, channel trip	Software testing, code inspection
Software fails in the presence of unexpected input data	Entry error Inadequate design	Visual Inspection, WDT actuation	Channel Trip, Channel Alarm	Undetected problem in channel could leave failed channel in service	channel redundancy, channel trip code check	Software testing, code inspection error tracking

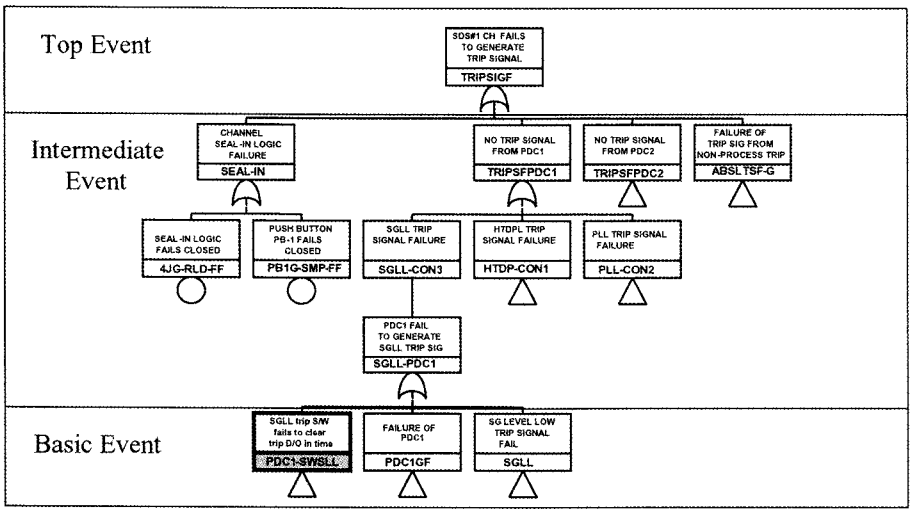


Fig. 10. FTA Result for the Steam Generator Level Trip Node

analysis performed in accordance with the guidelines of NUREG/CR-6430[15]. The guidewords to assess impact on hazards are suggested. HAZOP analysis was performed to evaluate the potential impact of possible software failures on identified hazards.

Figure 10 shows the FTA result for the steam generator level trip node. FTA is expressed in 3 events. A top event is SDS #1 channel failing to generate a trip signal. An intermediate event occurs when a component fails under conditions that are outside its designed operating range. A basic event occurs when a component fails [10].

Figure 11 shows a DV example for analyzing and validating the safety constraints of the steam generator level trip node.

In Figure 11, the safety properties are;

- Property 1: If the Signal is greater or equal to HSP, it implies that the Trip Signal must be true.
- Property 2: If the Signal is less than or equal to LSP, it implies that the Trip Signal must be true.

These properties can be described to system assertion

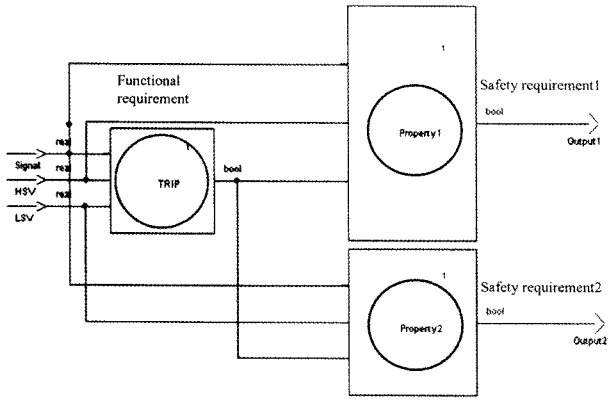


Fig. 11. Design validation of the steam generator level trip node

as; an abnormal SGLevel_* value_ (>SGLevelIrrHSP) always produces an SGLevel_*_Trip signal, or SGLevel_*> SGLevelIrrHSP implies SGLevel_*_Trip's value is true. This example shows how DV analyzes the boundary condition and safety constraint in a steam generator level trip case.

These properties are expressed as decision logic which has two input conditions. The inputs A and B must be Boolean signals. The output C shall be a Boolean signal as well. Therefore, the following algorithm is possible. "*If A is true then B must also be true.*" It means that if A is true and B is false, C is false; otherwise C is always true.

Figure 12 shows the feature of the safety shut-down system #1 for PHWR. Through an integration test, system reliability is proven.

5. CONCLUSIONS AND FURTHER STUDY

With the "3+3 Process", V cycle transforms to the Y cycle. This process has the following advantages and benefits. Advantages of the "3+3 Process" are: detecting software errors at an early stage, facilitating early detection and correction of software anomalies, and providing an early assessment of software and system performance. In addition, the "3+3 Process" generates life-cycle documents from a formal model. This feature resulted in the following benefits: reduction of the software phase documents, increasing the understandability of user needs, and reduction of cost and man-power for development while increasing software quality.

According to the projection of Airbus, around 30% of development time savings can be achieved when a Y cycle is adopted [13].

Although we have not quantitatively confirmed the

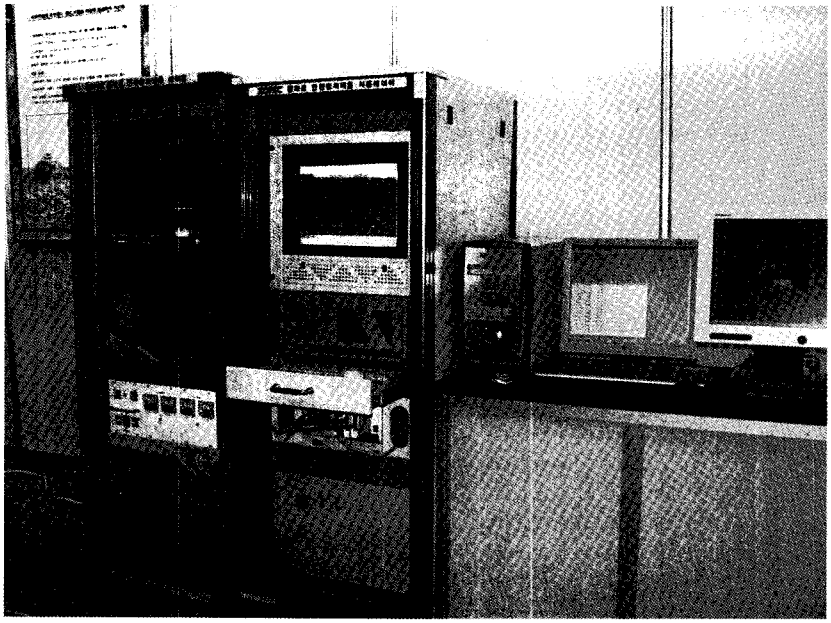


Fig. 12. Feature of the Safety Shut-Down System #1 for PHWR

savings through this work, the "3+3" process has the potential to reduce costs by: requirements error reduction, V&V decrease as well as coding work saving in the development process. Moreover, it cuts down a lot of the safety analysis process by way of simplifying the complicated process into the "3+3" process. Therefore, around 50% of development time savings is expected.

The prototype I&C system that is developed through the "3+3 Process" is running well and its performance satisfies the intended requirements. This scheme of development and safety analysis work minimizes V&V work while increasing the safety and reliability of the software product.

The "3+3 Process" does not propose the development of specific CASE (computer aided software engineering) tools for covering all software development and the entire safety analysis process that is defined in regulation guides. On the other hand, it suggests a methodology for integrating the commercial CASE tools to accomplish its intended goals as stated above.

After a nuclear power plant field test, this prototype will be implemented for the I&C system that hosts safety critical software.

REFERENCES

[1] IEEE Std. 1228-1994, "IEEE Standards for Software Safety Plans", IEEE, Aug. 1994

[2] Korea Institute of Nuclear Science, "Development of Safety Requirements and Guides for Digital Based I&C System Important to Safety in NPPs", KINS/RR-106, Mar. 2002

[3] IEEE Std. 1012-2004, "IEEE Standard for Software Verification and Validation", IEEE, June 2005

[4] IEC Std. 61508-5, "Functional Safety of Electrical/Electronics/ Programmable/Electronics Safety-related Systems - Part 5: Examples of Methods for the Determination of Safety Integrity Levels", IEC, 1998

[5] Debra S. Herrmann, "Software Safety and Reliability", IEEE Computer Society, 1999, pp. 28-32, 48-54.

[6] Standish Group International, Inc., "Extreme Chaos", 2001

[7] Wolfram Hohmann "Supporting Model based Development with Unambiguous Specifications, Formal Verification and Correct-by-construction Embedded Software", Society of Automotive Engineers, SAE international, 2004

[8] Regulatory guide 1.173, "Developing Software Life Cycle Processes for Digital Computer Software used in Safety System of NPPs", U.S. Nuclear Regulatory Commission, Sep. 1997

[9] IEEE Std. 1074-1995, "IEEE Standards for Software Life Cycle Processes", IEEE, Sep. 1995

[10] Neil Storey, "Safety-Critical Computer Systems", Addison-Wesley Publishing Company, 1996.

[11] Elizabeth Hull, Ken Jackson and Jeremy Dick, "Requirements Engineering", Springer, 2005

[12] Jean-Louis Camus, Bernard Dion, "Efficient Development of Airborne Software with Scade Suite™", Esterel Technologies white-paper, 2003

[13] Francois Pilarski, "Cost Effectiveness of Formal Methods in the Development of Avionics System at Aerospace", 17th Digital Avionics Conference, WA, Nov. 1998

[14] Felix Redmill, Morris Chudleigh, and James Catmur, "System Safety: HAZOP and Software HAZOP," John Wiley & Sons, 1999, pp. 25–26.

[15] NUREG/CR-6430, "Software Safety Hazard Analysis", U.S. Nuclear Regulatory Commission, Feb. 1996