IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 4, 2017, accepted October 15, 2017, date of publication October 26, 2017, date of current version March 12, 2018.

Digital Object Identifier 10.1109/ACCESS.2017.2765698

Input-Domain Software Testing for Failure Probability Estimation of Safety-Critical Applications in Consideration of Past Input Sequence

HEE EUN KIM[1], HAN SEONG SON[2], BO GYUNG KIM[3], JAEHYUN CHO[4], SUNG MIN SHIN[4], AND HYUN GOOK KANG[5]

[1]Korea Advanced Institute of Science and Technology, Daejeon 34141, South Korea
[2]Joongbu University, Chungnam 32713, South Korea
[3]Korea Institute of Nuclear Safety, Daejeon 34142, South Korea
[4]Korea Atomic Energy Research Institute, Daejeon 34057, South Korea
[5]Rensselaer Polytechnic Institute, Troy, NY 12180, USA

Corresponding author: Hee Eun Kim (heeeun.kim@kaist.ac.kr)

This work was supported in part by the project of Evaluation of Human Error Probabilities and Safety Software Reliabilities in Digital Environment under Grant L16S092000, in part by the Central Research Institute, Korea Hydro and Nuclear Power company, and in part by the Nuclear Research & Development Program of the National Research Foundation of Korea through the Ministry of Science and ICT (MSIT) under Grant 2017M2A8A4015291.

ABSTRACT Software failure probability quantification is an important aspect of digital system reliability assessment. Several quantification methods currently available in the software reliability field have characteristics unsuitable for application to safety-critical software. In this paper, a software test framework in consideration of input trajectory is developed, and a software failure probability quantification method is also suggested. The test input cases consist of the states and present inputs, where input trajectory is represented by the state. To obtain the input domain, which represents realistic plant behavior, digital system characteristics and plant dynamics are considered. This allows software failure probability to be estimated by using the result of each representative test case, thus reducing testing efforts. The proposed framework was applied to a nuclear power plant reactor protection system as an example to show its effectiveness. The method provides a practical and relatively simple way to test software and estimate software failure probability.

INDEX TERMS Safety, safety-critical software, software reliability, software safety, software testing.

I. INTRODUCTION

Nuclear power plants (NPPs) employ several safety systems to protect the public from the release of radioactive material in case of an accident. These safety systems are manipulated by the instrumentation and control (I&C) systems, which provide the control and monitoring functions of the various and diverse components and equipment that are essential to maintain safe operation. Existing I&C systems are currently being replaced with microprocessor-based digital systems because of the obsolescence of analog-based I&C equipment and a lack of vendor support. It is true that digital systems provide better performance such as improved accuracy, computation capabilities, and data handling, as well as the potential for improved capabilities such as fault tolerant techniques [1]. Nevertheless, the use of digital I&C systems has triggered a big challenge in terms of incorporating their characteristics into the probabilistic safety assessment (PSA) model traditionally used for evaluating the safety level of NPPs.

Various unique features of digital I&C systems to be modeled in the PSA of digital systems were identified by Kang and Sung [2]. Among them, estimation of software failure probability is an important factor, and a sensitivity study on the digital reactor protection system (RPS) showed the relationship of system unavailability and software failure probability. A report on operating and maintenance experience described how software error caused a significant number of digital system failures during 1990–1993 [3], where 30 failures were caused by software error compared to 9 from

random component failure. The report also stated the possibility of common-mode or common-cause software failure, which can lead to significant safety threats.

In response, several software reliability quantification methods have been developed that can be adopted in the nuclear field. The software reliability growth method [4] models the decreasing failure rate of software following the identification and removal of faults during the development process to satisfy design and regulatory requirements. By applying a software reliability model and estimating its parameters using software failure data, assessment and prediction by extrapolation can be achieved. In regards to safety-critical software though, it is developed under a strict verification and validation (V&V) process so rarely produces appropriate failure data, and moreover the estimation result is highly sensitive to the data [5]. Furthermore, any software modification may introduce other faults which are not clearly modeled. The Bayesian belief network is another promising method for the quantification of software reliability; it models and integrates several aspects that affect the reliability of a software, such as design features, requirements, and V&V quality [6]. Several studies successfully applied the method to safety-critical software [7], [8], but the qualitative nature of the method prevents highly accurate results without consistency of expert judgment. Test-based methods can also be applied to software reliability quantification, which utilize the results of software testing. Tests are performed with sample test cases representing actual data to simulate the software demands, with results demonstrating whether the expected output is produced or not. Test-based methods are divided into two main categories: white-box and black-box testing. White-box tests consider the internal structure of a software, such as nodes and paths, to cover every part of the software. Popstojanova and Trivedi provided a comprehensive investigation of white-box based software reliability models and categorized them as state-based models, path-based models, and additive approaches [9]. Black-box tests do not consider internal structure, but rather test random samples from the software input space with results statistically analyzed. May et al. provided a method of generating a simulated sample from operational distribution [10]. Miller suggested a software failure probability estimation method for life-critical applications by binning the domain and estimating the failure probability of each bin using the Bayesian method [11].

The failure of software generally comes from design or coding faults, so the behavior of a software for a given input is deterministic. However, when the input is selected at random, the output also appears to be random. That is, the random nature of software failure can be explained by the uncertainty of the input sequences [11]. Finelli introduced the concept of an error crystal, which represents regions of the input space that cause a program to produce errors [12]; an input entering the error crystal leads to failure. In case of real-time control applications, a series of contiguous inputs might produce consecutive errors. On the other hand, by considering the operational profile, the failure probability of a software can be assessed from test results. Lyu [13] defined operational profiles and described their development and use in testing. An operational profile consists of disjoint operations that the software can execute along with the probability with which the operations will occur. The operations themselves are partitions of the software input space and are defined as a group of runs that typically involve similar processing. Here, a run represents the smallest division of software processing that is started by external demand.

Since the behavior of software is deterministic, the software will behave in the same way for a given set of inputs and internal state. Whereas thorough testing of the possible input space can assure the reliability of software, exhaustive testing is generally considered impossible on account of system complexity [14]. If the input space or domain can be specified, then test-based reliability estimation would lead to another promising method of software reliability estimation. Kang *et al.* suggested a test-based software reliability quantification method for an RPS that considers input profiles by reflecting digital system characteristics and plant dynamics [15]. By following this previous study, the input set covering all possible input spaces can be developed for real-time safety applications; however, it is assumed that the software responds only to its input, so it has a limitation in consideration of input trajectory which might activate the design faults [16]. Failures can be attributed to erroneous internal states, and the previous input sequence may change the internal state of a software during its processing [17], [18]. Therefore, the method proposed by Kang et al. needs to be extended by considering input trajectory which may change the internal state.

The objective of this study is to take previous input sequences into consideration to generate test cases by adopting the concept of the software internal state. Since the test cases reflect the effect of past input sequences, the software can be verified through testing. In addition, by considering the operational profile, the test set covers all possible demands for exhaustive testing. Since the test set covers all possible cases, the software failure probability can also be assessed from the test results. The proposed method integrates white-box and black-box testing as it considers the internal structure of the code as well as the operational profile. The result of the suggested method can be utilized as a reference for regulators and as input data for PSAs.

II. TEST SETS REGARDING SOFTWARE STATE

In this section, state variables are defined to reflect the state of a running software. The method of building test cases that include state variables is also described.

A. STATES OF REAL-TIME SAFETY-CRITICAL SYSTEMS

A finite state machine (FSM) is a mathematical model to design a logic device that has a limited or finite number of possible states. A typical FSM is defined as having six features: inputs from the outside, outputs to the outside, the state of the system stored in the memory elements, initial state,

a next-state decoder and output decoder where the next state and output are calculated as a function of inputs and current state. As a result, the state captures the essential properties of the history of the inputs and is used to determine the current output as well as the next state of the system. In other words, past inputs affect the behavior of the software in the form of the state, which is a combination of values of stored variables. The state transition is a physical change of the values in the memory. For example, let us assume that a vending machine dispenses a drink for 1 dollar. The memory of the vending machine stores the amount of money put into the coin slot; inputs can be 10-cent or 50-cent coins, and output is whether the drink is dispensed or not. The ''50-cent state'' can be reached from a 50-cent coin or five 10-cent coins, according to its next-state decoder. When an input of a 50-cent coin is given as an input to the ''50-cent state'', the state changes to 100 cents and the output will be a drink, according to its output decoder. A drink is served regardless of the type and the sequence of the coins, as output is only related to the present state and input.

A real-time safety-critical system can be modeled as an FSM because it has a limited number of states. A programmable logic controller program, typically employed in real-time safety-critical applications, periodically and indefinitely reads inputs, computes new internal states, and updates outputs in each scan cycle. From a safety point of view, testing needs to focus not on spurious operation but on safety signal generation failure, thus the possible number of states does not explode as that of general computers. In case of real-time safety-critical applications, variables for safety functions are saved in the designated memory address; these states change according to the input sequence, representing the past input sequence of the real-time safety-critical system.

B. DEVELOPMENT OF TEST SETS REGARDING SOFTWARE STATE

A test set is defined as a collection of test cases composed of inputs and expected results. In this study, the state of a software represents the past test input sequence. As the sequence of inputs to real-time safety-critical software continually changes the state every scan, the last state represents the effect of all past input sequences. Therefore, the state stored in the memory is included in order to generate a test set that considers input history.

Data from the previous cycle stored in the memory needs to be loaded into the current cycle, and only data stored in memory can affect the state and output of the next cycle. Therefore, state variables can be defined as stored variables in the memory, which are loaded every scan. Temporary variables used in one cycle will not be treated as internal state variables, because they are stored and loaded in one scan cycle but do not affect the next cycle. By inspecting the source code of the software, the list of state variables and input variables can be obtained. As a result, the test set consists of state variables, input variables, and corresponding expected output. The following subsections describe the characteristics

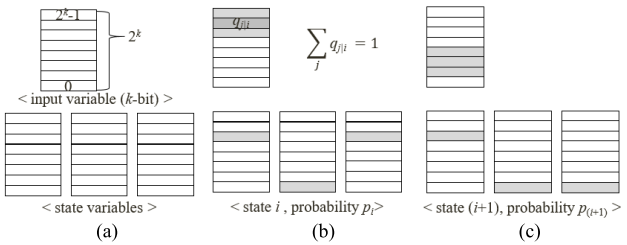

FIGURE 1. (a) Test case representing multi-dimensional space. (b), (c) Test cases which represent state i and $i+1$, respectively, and corresponding possible range of inputs.

of the test set that should be considered to identify each test case.

1) MULTI-DIMENSIONAL TEST INPUT SPACE

As the test set is a combination of input variables, state variables, and expected output, it is constructed from a multi-dimensional test input space. Figure 1 (a) shows a conceptual test set consisting of an input variable and three state variables. The empty boxes of Fig. 1 represent the range of a variable, from zero to 2^{k-1} where k is the resolution of the digital system. Combinations of values of each variable construct the test set. Therefore, the number of test cases is 2^{nk}, where n is the number of variables in the test input space. However, testing all these cases is almost impossible considering the long test time; for example, the number of two 16-bit test cases is greater than 4 billion, which would take more than 100 years if a test takes 1 second.

As some variables are related to each other though, the number of test cases can be less than 2^{nk}. The combinations of state variables represent the state of a plant, but some combinations do not appear in the real world. For example, we can expect that the next input will be a value around the present input value; testing therefore should be performed only for cases that can feasibly be entered as inputs. Possible combinations of variables derived from the operational profiles need to be tested for software reliability quantification. The state and corresponding range of each variable can be obtained by considering plant dynamics, digital system characteristics, and the relationships among the variables. Then the possible range of inputs for each state can be obtained. Figure 1 (b) and (c) conceptually describe states i and $i+1$, respectively, as sets of state variable values. State i with probability p_i is represented as combination of specific values, marked in color, among the possible values of each state variable, and the corresponding possible range of input is also marked in color. The probability of each possible j-th input for state i can be represented as $q_{j|i}$. Figure 1 (c) shows a different combination of values for state variables and a different range of corresponding possible input. Different values of state variables represent the different states, and the possible input range is also different according to the state.

2) PAIRED VARIABLES AND INDEPENDENT VARIABLES

As described above, the next state of a program is determined according to the present state and input. That is to say, the value of each state variable is determined by the value of the present state variables and input variables. Therefore, if an input variable determines the next value of several state variables, the values of these state variables are correlated to each other. These variables can be referred to as paired variables, and the values of a pair need to be identified together. Since the values of state variables are correlated, the range of each variable should be identified one by one from a variable of which distribution is known. Otherwise, state variables can be referred to as independent variables. The range of independent state variables can be determined by reflecting failure data or software design specification, etc.

C. PARTITIONED SAMPLING SPACE

From a safety point of view, testing needs to focus on safety signal generation for a given safety signal demand. The set of input variables and state variables represents the particular plant dynamics that generate safety signal demands in real operation, so the number of test cases can be limited. The test set should cover all possible safety signal demand situations, but the total test space should comprise the limited cases representing real operation. The probabilities of test cases are different as the probabilities that the actual state represented by the test case will occur are different.

These test cases are partitioned sampling spaces, as described in previous studies [11], [15], and represent different situations. The total failure probability (θ_t) of a software is expressed as the weighted sum of each partitioned sampling space, so the equation in the previous study can be extended as follows

$$\hat{\theta}_t = \sum (p_i \times q_{j|i}) \hat{\theta}_{j|i}, \tag{1}$$

where p_i is the probability of state i, $q_{j|i}$ is the probability of input j for state i, and $\hat{\theta}_{j|i}$ is the corresponding software failure probability. For a partitioned sampling space expressed by i and j, the probability can be obtained by calculating $p_i \times q_{j|i}$, and expected failure probability $\hat{\theta}_{j|i}$ can be obtained from the test result. The steps for calculating $p_i \times q_{j|i}$ will be described in Section III.

III. DETERMINING PROBABILITIES OF TEST CASES FOR QUANTIFICATION OF SOFTWARE FAILURE PROBABILITY

A. PROBABILITY OF A TEST CASE

As described in the previous section, if the probability of the states and corresponding input variables is identified, the total failure probability of software can be assessed. We denote the probability of a state i as p_i, and the conditional probability of the j-th input of state i as $q_{j|i}$ (Fig. 1 (b)), and then the probability of a test case is the product of the two probabilities, $p_i \times q_{j|i}$.

Probability p_i can be calculated by multiplying each probability of paired variables and independent variables, since the pairs and the independent variables are independent of each other. Therefore, possible variable pairs and their probabilities, as well as the range of each independent variable and its probability, need to be identified. When there are u pairs of paired state variables and several independent state variables, p_i could be presented as

$$p_i = r_{1,m} \times \ldots \text{ę} \times £ r_{u,n} \times r_{(u+1),l} \times \ldots \tag{2}$$

The probability of the m-th case of the first state variable pair is expressed as $r_{1,m}$, and the probabilities of the next variable pairs are multiplied in succession until the probability of the n-th case of the u-th pair is multiplied. Then the probability of the l-th case of independent state variable $r_{(u+1),l}$ is multiplied in succession to calculate p_i. Figure 2 shows a conceptual state i consisting of three state variables, two of which are paired variables. Possible state space consists of combinations of m-th and n-th cases of the pair and independent variables. Among $2^k \times 2^k$ possible value combinations for the pair, only a portion of value pairs could actually exist, and the probability of the m-th value pair is $r_{1,m}$. The sum of the probabilities of the possible value pair, $\sum_m r_{1,m}$, is one. Similarly, the sum of the probability of the possible independent variable $\sum_n r_{2,n}$ is also one. The probability of paired variable $r_{u,n}$ cannot be simply expressed as a multiplication of each state variable included in the pair; the possible state variable pairs need to be identified from the possible states of a program. The process of obtaining the possible state variable pairs is described below.

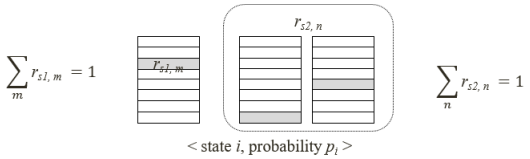

FIGURE 2. State variables representing state i.

B. OBTAINING THE PROBABILITY OF EACH VARIABLE

1) OBTAINING PROFILE OF INDEPENDENT VARIABLES

As independent variables are not related to the values of other variables, their distribution can be obtained by utilizing available data regardless of the other variables. By referring to plant operation strategy, design specification, plant dynamics, and so on, the possible range of variables can be obtained. For example, modes which bypass safety signal generation do not have to be tested from a safety viewpoint. Thus the bypass variable representing not-bypassed is included in the test set, with a probability of one. The probabilities of independent variables can be multiplied to the other probabilities of independent variables and pairs, as presented in Eq. 2.

2) OBTAINING PROFILE OF PAIRED VARIABLES

Distributions of paired variables need to be considered together, which can be obtained from plant dynamics,

simulation results, and so on. If the variables are related as input–output of a block, the range of one variable can be identified from that of the other variable. Since the function block diagram (FBD) program has no feedback loop, if the value of an output of a block is known, then the possible range of inputs of the block can be determined. From the known distributions, other distributions of variables can be obtained in sequence.

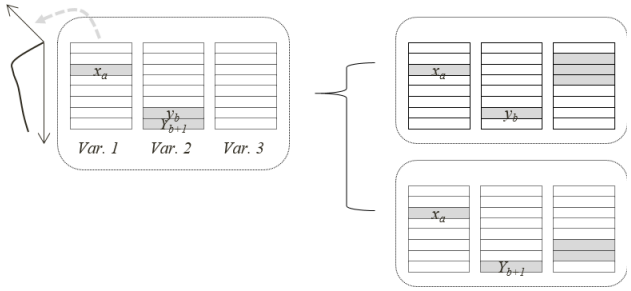

FIGURE 3. Conceptual diagram of the procedure for obtaining the profile of paired variables.

This procedure is conceptually depicted in Fig. 3, which illustrates a variable pair with three state variables. The process begins with variable 1 of which the distribution is known, with the range of other variables identified one at a time. For example, if variable 1 has a value of x_a, the possible range of variable 2 is limited to y_b and $y_{(b+1)}$. The conditional probability of variable 2 having value y_b when the value of variable 1 is x_a is $p_{y_b|x_a}$, which can be obtained from available data. Similarly, depending on whether variable 2 has a value of y_b or $y_{(b+1)}$, the possible values of variable 3 can be obtained. The conditional probabilities of variable 3, with variables 1 and 2 having x_a and y_b, is $p_{z_c|x_a y_b}$, which can also be obtained from available data. As this is the process to calculate the probability $r_{u,n}$ from Eq. 2, the probability $p_{x_a y_b z_c}$ of a pair with values of $p_{x_a y_b}$ can be expressed as Eq. 3 as a succession of multiplication of conditional probabilities,

$$r_{u,n} = p_{x_a y_b z_c} = p_{z_c|x_a y_b} p_{x_a y_b} = p_{z_c|x_a y_b} p_{y_b|x_a} p_{x_a}. \quad (3)$$

This process should be repeated starting from the first possible value of variable 1 to the last possible value of variables 1, 2, and 3. If there is insufficient data to obtain a distribution of a variable, test cases with all possible values need to be tested to obtain conservative results. In this case, the probability of each variable and each test case cannot be obtained. For example, if the conditional probability of variable 3 cannot be obtained, test cases with x_a, y_b, and all possible values of variable 3 need to be tested. In this case, the sum of probabilities of these test cases $p_{x_a y_b}$ can be utilized for failure probability quantification, which is described in Eq. 4. Similarly, if data is not available for variables 2 and 3, test cases with x_a and all possible values of variables 2 and 3 need to be tested, which is described in Eq. 5. The generation of test cases needs to be repeated for the possible range of values of variables 1 and 2, and probabilities of p_{x_a} and p_{y_b}.

$$\Pr\{\text{all test cases with value } x_a, y_b \text{ for variables 1, 2}\} = \sum\nolimits_c p_{z_c|x_a y_b} = p_{x_a y_b} \quad (4)$$

$$\Pr\{\text{all test cases with value } x_a \text{ for variable 1}\} = \sum\nolimits_b p_{y_b|x_a} p_{x_a} \quad (5)$$

C. TEST AND FAILURE PROBABILITY QUANTIFICATION PROCEDURE

The test step starts with obtaining a scenario which generates signal demand to the target software. Then input and state variables need to be verified to construct a test set by inspecting software code. The value and probability of each test set variable can be obtained by analyzing software design, plant dynamics, operation method, etc. It is recommended that obtained test cases be tested as per their probability, from high to low. When testing is processed without error, the probability of a test case $\hat{\theta}_{j|i}$ is updated from 1 to 0, because the test case is verified as an error-free portion. The total failure probability θ_t can also be updated. For safety software, if a failure is observed, it should be debugged and the test should be restarted from the first test case, and the result of the former version of the software should be disregarded. The overall procedure is depicted in Fig. 4, where w is the number of test cases.

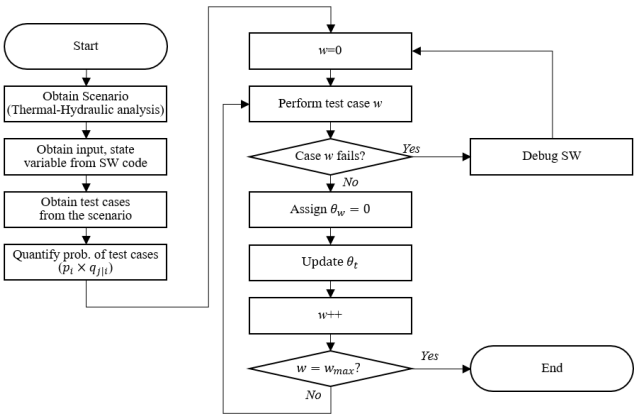

FIGURE 4. Overall software failure probability quantification procedure.

As previously mentioned, as the state represents the previous input sequence, the test input does not have to include lengthy past input sequences. Therefore, the test process can be simplified and the time required for testing can be decreased. Besides, for different scenarios that might lead to the same state, each related input sequence does not need to be individually tested. Several scenarios can be replaced with a representative case, considering the deterministic nature of software, so the number of test cases can be reduced.

IV. CASE STUDY

As a case study, the suggested method was applied to a target safety-critical software. Test cases and the probability of each

case were identified. The results of testing from the obtained test cases can be utilized for software failure probability quantification.

A. TARGET SYSTEM

Among the safety-critical applications in NPPs, a fully digitalized RPS, developed under the Korea Nuclear Instrumentation & Control Systems project (KNICS), was selected as a target system. The RPS generates trip signals when the plant deviates from normal conditions. Bistable processors (BPs) compare process variables with their trip setpoint (TSP), and coincidence processors perform two-out-of-four voting logic to determine whether the system generates a trip signal. Among the software modules, 19 modules for trip signals are defined in the BP. These trip logics can be categorized into several types, for example fixed TSP, variable TSP by manual reset, and variable TSP by automatic rate-limiting. Among the 19 trip signals, ''PZR_PR_Lo Trip'' (pressurizer pressure low trip) was chosen as the target logic, which has a variable TSP and operator bypass function. This trip logic is relatively more complex than the other logics, making it a good example to demonstrate the advantage of the suggested method. [19]

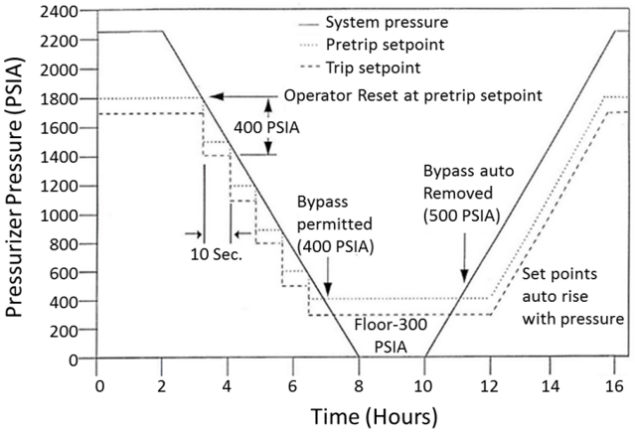

FIGURE 5. Configuration of the low pressurizer pressure trip (variable TSP).

Figure 5 shows the configuration of the low pressurizer pressure trip. It generates a trip signal if the pressure decreases below the TSP. When the plant is in full power mode, the TSP is fixed to 1762 psi. The TSP ranges between 1762 $\sim$ 300 psi during shut-down and start-up processes. The operator should manually decrease the TSP while the pressure slowly decreases during the shut-down process. When the pre-trip alarm occurs, where the pre-trip setpoint is set to 100 psi above the TSP, the operator has to push the reset button after which the TSP decreases 400 psi below the current pressure. Further decrease is not permitted within 10 s, and bypass is permitted under 400 psi. When the plant starts up, the TSP is automatically set to 400 psi below the current pressure, and bypass is canceled from 500 psi.

The design requirement of the KNICS RPS limits the scan time to less than 50 ms. As the scan time increases, the deviation of process variables increase, therefore we will assume 50 ms of scan time to find the maximum number of states. Process variables from the measuring instruments are analog signals that are converted into digital signals by a 12-bit analog-digital-converter, in the case of the OPR1000 NPP. A total of 212 digital values represent the input span of the target system, and we can derive the scale of one digital value. In the target logic, one step of digital value increase or decrease corresponds to a change of 0.703 psi.

B. STATES AND VARIABLES OF THE TARGET SYSTEM

By inspecting the source code of PZR-PR-Lo-Trip logic, we obtained three state variables: TSP, Previous-pressure, and Reset-delay-time, and five input variables: Current-pressure, Bypass-from-MCR, Bypass-from-RSR (remote shutdown room), Reset-from-MCR (main control room), and Reset-from-RSR. As the target is trip logic, the result of a test case should be ''trip''. Current-pressure is a process parameter obtained from the plant, with a value that cannot exceed a certain degree from Previous-pressure, because transition within the scan time is limited. TSP is calculated from Current-pressure and Previous-pressure, which is stored in the memory. Since the Previous-pressure and TSP are determined by using Current-pressure, these two variables are paired state variables. The Reset-delay-time is a counter which has to remember its previous value, so it is also a state variable. The value of this variable depends on the time difference between the accident and operator order. The timing of the accident and operator action are independent, so the Reset-delay-time variable does not depend on input variable (current pressure) or corresponding paired state variables. The remaining four variables represent whether the operator generates bypass or reset. Therefore, they do not determine the values of any other variables in this set, thus are independent of each other. All variables are summarized in Table 1. State variables are marked with *italics*.

TABLE 1. Variables in PZR-PR-Lo-Trip logic.

Type	Input variable	Related variable
Paired variables	Current-pressure	*TSP*
		Previous-pressure
Independent variables	*Reset-delay-time*	
	Bypass-from-MCR	
	Bypass-from-RSR	
	Reset-from-MCR	
	Reset-from-RSR	

A combination of state variables represents a certain state of running software. When the plant operates in full power mode, the TSP is fixed and the operator does not generate the reset signal, so the state can be characterized by changing Previous-pressure and other fixed variables during full power mode. By testing cases from this mode, almost all of the fault-free portion is covered because most pressure-drop-accidents and following trip signal generation demands occur

during full power mode. The TSP changes during the start-up process but the reset timer does not change, therefore the corresponding state is characterized by changing TSP and corresponding Previous-pressure. While the plant is in a shut-down process, the operator pushes the reset button, so changing Reset-delay-time needs to be considered with the corresponding state represented by TSP, Previous-pressure, and Reset-delay-time. In this case, Reset-delay-time is independent of TSP and Previous-pressure. For all three cases, the possible values of the paired input variables are determined by the current state. The value of Current-pressure should be slightly below the TSP, and smaller than Previous-pressure by the amount of transition during scan time. Test cases are composed of possible variations of these variables.

C. THE PROBABILITIES OF EACH TEST CASE

The range of independent variables can be obtained irrespective of other variables. In the target logic, Reset-delay-time counts up to 10 s. Since the scan time is 50 ms, this variable may have digital values from 0 to 200. Bypass-from-MCR and Bypass-from-RSR are Boolean type variables, so they are either 1 or 0. If an operator gives a bypass order, the system should not generate a trip signal, so these cases do not have to be examined. Accordingly, the values of Bypass-from-MCR and Bypass-from-RSR are always ''not bypassed''. Reset-from-MCR and Reset-from-RSR are also Boolean type variables and are either 1 or 0 depending on the situation. Previous and current input values can be obtained in the same manner as described in the previous work [15]. For a given resolution and scan time interval, the possible range of Current-pressure and Previous-pressure can be obtained.

TABLE 2. Categorization of the LOCAs and their frequencies at full power operation, start-up, and shut-down modes.

ID	Hole diameter (m)	Frequency (#/y)	Fraction
1	0.0127	5.44E-03	8.915E-01
2	0.0413	4.47E-04	7.325E-02
3	0.0762	1.18E-05	1.934E-03
4	0.1778	1.10E-06	1.803E-04
5	0.3556	8.40E-08	1.377E-05
6	0.761	1.50E-08	2.458E-06
Shut-down		1.14E-04	1.867E-02
Start-up		8.84E-05	1.449E-02

The probability of the TSP and Previous-pressure pair is denoted by $r_{x1,m}$, and that of Reset-delay-time is denoted by $r_{x2,n}$ where x denotes the three operation modes. The probabilities can be obtained based on the plant operation mode and their fraction of accident frequency. In this study, a loss of coolant accident (LOCA) is analyzed as a representative pressure transient accident. The accident frequencies are shown in Table 2, which were obtained from studies by the United States Nuclear Regulatory Commission and Lim [20], [21]. Since most LOCAs occur during full power operation, various LOCAs with different sizes during full power operation are analyzed. The fraction of test cases are calculated based on the relative fraction of accidents. The pressure drop data were obtained by performing thermal-hydraulic simulation.

1) FULL POWER OPERATION

At full power operation, pressurizer pressure is controlled to have constant values, and the TSP is set to its maximum value. As described above, the state is characterized as Previous-pressure and other variables with fixed values. In case of small LOCA (accidents #1, #2, #3 in Table 2), the transition of pressure is very slow, so the maximum Previous-pressure is one digital value above the TSP. Although the transitions of accidents #2 and #3 are faster than accident #1, they are treated as the same value because of the resolution of the digital system. As the size of the accident increases (#4, #5), the transition quickens. In case of a large LOCA (#6), Previous-pressure might have the maximum of five digital values below the TSP depending on the scan timing. If uniform distribution can be assumed for the Previous-pressure—that is, if the states are evenly distributed—frequency fraction can be divided into the number of states to obtain the probability of each state. Since the previous pressure is the only state variable in the full power operation test case, the probability of the m-th case of state variable pair $r_{f1,m}$ can easily be obtained. For example, in case of accident #6, the probability of having one of the five digital values can be calculated as $r_{f1,1\_\#6} = r_{f1,2\_\#6} = \ldots = r_{f1,5\_\#6} = \frac{(frequency\ fraction\ of\ accident\#6)}{5}$. The independent variable Reset-delay-time is fixed to its maximum value because the manual reset signal will not be generated. Therefore the probability of this independent state variable $r_{f2,1}$ is equal to one.

For each given state, we can obtain current pressure. The pressure changes linearly during the short scan time, so the current pressure is a transition within the scan time from the previous pressure. It corresponds to four or five digital values below the previous pressure, considering the timing of the scan. Not every conditional probability can be obtained, but the sum of the conditional probabilities is 1, $\sum_j q_{j|i} = 1 (1 \le j \le 2)$. Two test cases need to be conducted with state i and corresponding two possible inputs; then p_i of the error-free portion can be obtained. For each accident in Table 2, possible states (with probability p_i), corresponding input range (with probability $p_i \times q_{j|i}$), and the error-free portion covered by the case or a fraction of the case (with probability of $p_i \times \sum q_{j|i}$) were obtained as shown in Table 3. In this table, state i represents the value of Previous-pressure being i-digital values above the TSP, and input j represents the value of the input being j-digital values below the TSP.

Table 3 can be condensed as Table 4 according to the state and input. In this table, the probabilities of the common states and the inputs of different accidents are summed. For example, $r_{1,1}$ was calculated as the sum of the probabilities for each accident, $r_{f1,1} = \sum beobtained, butthesumoftheconditionallimits_{x=1}^{6} r_{f1,1\_\#x}$. A set of state and inputs in Table 4 represents one test case.

TABLE 3. Possible states and inputs for the full-power mode, according to the accident type.

Accident # (Frequency fraction)	State	Input	Fraction of a test case
#1 (8.915.E-01)	state 1	input 1	8.915.E-01
#2 (7.325.E-02)	state 1	input 1	7.325.E-02
#3 (1.934.E-03)	state 1	input 1	1.934.E-03
#4 (1.803.E-04)	state 1	input 2, 3	6.009.E-05
	state 2	input 1, 2	6.009.E-05
	state 3	input 1	6.009.E-05
#5 (1.377.E-05)	state 1	input 4, 5	2.753.E-06
	state 2	input 3, 4	2.753.E-06
	state 3	input 2, 3	2.753.E-06
	state 4	input 1, 2	2.753.E-06
	state 5	input 1	2.753.E-06
#6 (2.458.E-06)	state 1	input 4, 5	4.916.E-07
	state 2	input 3, 4	4.916.E-07
	state 3	input 2, 3	4.916.E-07
	state 4	input 1, 2	4.916.E-07
	state 5	input 1	4.916.E-07

TABLE 4. Test cases for full-power mode according to the state and input.

State	Input	Fraction of a test case	Software failure probability
state 1	input 1	9.666.E-01	3.335.E-02
	input 2, 3	6.009.E-05	3.329.E-02
	input 4, 5	3.245.E-06	3.329.E-02
state 2	input 1, 2	6.009.E-05	3.323.E-02
	input 3, 4	3.245.E-06	3.323.E-02
state 3	input 1	6.009.E-05	3.317.E-02
	input 2, 3	3.245.E-06	3.316.E-02
state 4	input 1, 2	3.245.E-06	3.316.E-02
state 5	input 1	3.245.E-06	3.316.E-02

Software failure probabilities $(1 - \sum p_i)$ are also shown, assuming the test is successfully finished. By testing a case set with state 1 and input 1, 96.67% of the input space can be covered. There are 15 test cases, corresponding to 96.68% of total input space, so software failure probability converges to 3.316E-02. The remainder (3.316E-02) will be covered in the LOCA of shut-down and start-up processes.

2) START-UP PROCESS

To estimate p_i, the probability of changing TSP and corresponding Previous-pressure should be identified. TSP changes during specific plant operational states (POS), so the frequencies of LOCA for each POS were applied from Lim's study [21]. During refueling and refilling of the reactor coolant system, the trip signal should be bypassed so the LOCAs during these POSs do not have to be considered.

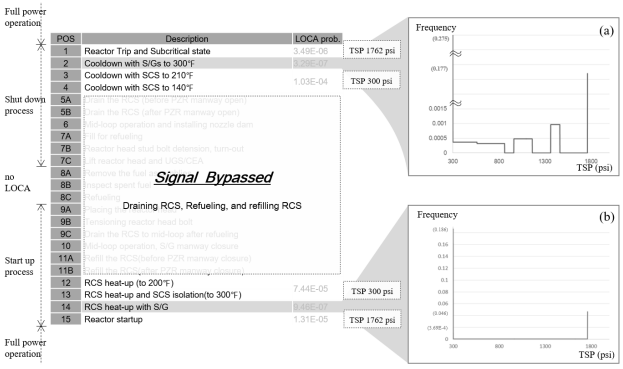

FIGURE 6. Profile of TSP during the (a) shut-down and (b) start-up processes.

When the pressure remains low, the TSP is set to 300 psi. When the pressure goes up, the TSP also increases, with 2080 digital values along the steady increase of TSP from 300 psi to 1762 psi. At the end of the start-up process, pressure remains high and the TSP is at 1762 psi. Based on the frequency of LOCA for each POS, the frequency fraction of LOCA for each TSP can be derived. If we assume that the pressure increases linearly, the probabilities of TSP between 300 psi and 1762 psi are the same. The lower graph in Fig. 6 shows the relative frequencies during the start-up process. A LOCA accident during start-up accounts for 1.449.E-02 among all LOCA accidents, as shown in Table 2, so the LOCA frequency for each TSP can be calculated. In this case, thermal-hydraulic simulation was done for large LOCA, assuming the worst case.

For each TSP, the range of previous pressure and corresponding input needs to be obtained to determine the possible states. In this case, obtaining detailed simulation results for all possible states is not realistic; therefore, a boundary of state variables was conservatively set. The maximum deviation does not exceed five digital values because the pressure drop of low pressure is less than that of full-power operation pressure. It can therefore be assumed that there are at most five states for each TSP. As described above, even though each probability of a state $r_{u1,m} = \Pr\{\text{possible previous pressure at TSP}^*|\text{TSP}^*\}$ cannot be exactly estimated, the sum of probabilities of the states with the same TSPs can be obtained as the probability of that TSP. If additional simulation results can be obtained, the maximum deviation and number of corresponding states would decrease. In this study, the number of states decreased to 8971 by utilizing additional simulation results. Representative test cases are presented in Table 5. The independent variable Reset-delay-time always has its maximum value, so the probability of independent state variable $r_{u2,1}$ again equals 1 and the state probability accordingly is $p_i = r_{u1,m} \times r_{u2,1} = r_{u1,m}$, $(6 \le i \le 8976)$. Current pressure can be obtained in the same manner as in the previous section.

Table 5 lists some possible states (with probability p_i), corresponding possible inputs (with probability $p_i \times \sum q_{j|i}$), the sum of error-free fraction with the

TABLE 5. Test cases and software failure probability for start-up process.

TSP	Possible previous pressure	State	Possible Inputs	Sum of error -free portion	Software failure probability
#1 (300 psi)	p.input 1	state 6	input 1	1.222.E-02	2.094.E-02
#2	p.input 1	state 7	input 1	5.718.E-08	2.094.E-02
	…				
#2079	p.input1	state 8967	input 1 to 5	5.718.E-08	2.082.E-02
	p.input2	state 8968	input 1 to 4		
	p.input3	state 8969	input 1 to 3		
	p.input4	state 8970	input 1 to 2		
	p.input5	state 8971	input 1		
#2080 (1762 psi)	p.input1	state 8972	input 1 to 5	2.148.E-03	1.867.E-02
	p.input2	state 8973	input 1 to 4		
	p.input3	state 8974	input 1 to 3		
	p.input4	state 8975	input 1 to 2		
	p.input5	state 8976	input 1		

same TSP (with probability $\sum p_i$), and the software failure probability (with probability $1 - \sum p_i$). Here, input j also means the j-th count from its TSP, so the j-th count of two different TSPs is not the same input. Based on the TSP profile during the start-up process as shown in Fig. 6, and the frequency of LOCA, each sum of probabilities of states ($\sum p_i$) with the same TSPs was calculated. During the start-up process, the most frequent TSP is 300 psi and 1762 psi, so test cases with these TSPs take up most of the probability. Testing all the test sets for the start-up process reveals $\sum beobtained, butthesumoftheconditionallimits_{i=6}^{8976} p_i = 1.449\text{E} - 02$ of fault-free portion.

3) SHUT-DOWN PROCESS

The shut-down process test set is represented by the TSP, Previous-pressure, and Reset-delay-time variables which need to be obtained to estimate p_i. In this case as well, the frequency of LOCA for each POS was considered. When the shut-down process is started, pressure remains high with the TSP at 1762 psi, and it is fixed at 300 psi at the end of the process. While draining the reactor coolant system and refueling, the trip signal should be bypassed so the LOCA during this period does not have to be considered. If we assume that the pressure decreases linearly, the distribution of the TSP during pressure decrease can be obtained. The TSP changes depending on the operator's order, so it does not decrease gradually. There are six possible TSPs: 1762, 1462, 1162, 862, 562, and 300 psi, if the operator immediately resets the TSP. However, if the operator does not reset it immediately, TSP is set to a relatively lower value because current pressure decreases continuously. The upper graph of Fig. 6 shows the possible distribution of TSP during the shut-down process, assuming uniform distribution between the resets as the exact distribution of human action was not available. The probabilities of the distribution need to be improved by reflecting more accurate data on the operator response to the alarm. Since the proportion of LOCA accidents during the shut-down process is 1.867.E-02, the accident probability for each TSP can be calculated.

Similar to the start-up process, the probability of the paired variable $r_{d1,l} = \Pr\{\text{possible previous pressure at TSP}^*|\text{TSP}^*\} \times \Pr\{\text{TSP}^*\}$ cannot be exactly calculated either, but again in this case, the sum of the probabilities of the states with the same TSPs can be obtained as the probability of that TSP. All possible states, corresponding possible inputs, and their probabilities can be obtained in a manner similar to that described in subsection *2) start-up process*. In this case, the profile of the Reset-delay-time variable, which usually has a maximum value except for 10 s, should be considered. For example, if the pressure decreases with a rate described in Fig. 5 (2250psi/6h), the probabilities of Reset-delay-time for each state are $r_{d2,n} = 1.691\text{E} - 0.5 (0 \leq n \leq 199)$ and $r_{d2,200} = 9.966.\text{E} - 01$. To estimate p_i, $r_{d2,n}$ is multiplied to $p_i = r_{d1,m} \times r_{d2,n}, (8977 \leq i \leq)$.

TABLE 6. Test cases and software failure probability for shut-down process with a TSP of 1762 psi.

Reset counter	Previous pressure	State	Input	Sum of error	Software failure probability
200	p.input1	8977	input 1 to 5	5.816.E-04	2.983.E-05
	p.input2	8978	input 1 to 4		
	p.input3	8979	input 1 to 3		
	p.input4	8980	input 1 to 2		
	p.input5	8981	input 1		

TABLE 7. Test cases and software failure probability for shut-down process with a TSP of 300 psi.

Reset counter	Previous pressure	State	Input	Sum of error	Software failure probability
0	p.input1	13002	input 1 to 2	3.054.E-07	1.867.E-02
	p.input2	13003	input 1		
…					
199	p.input1	13400		3.054.E-07	1.867.E-02
	p.input2	13401			
200	p.input1	13402		1.800.E-02	6.114.E-04
	p.input2	13403			

Tables 6 and 7 describe all possible states, corresponding possible inputs, the sum of error-free fractions (with probability $\sum p_i$), and the software failure probability (with probability $1 - \sum p_i$) for 1762 psi and 300 psi, respectively. For certain TSPs, the Reset-delay-timer and previous input form a state, with corresponding input presented. Here, input j also means the j-th digital value below its TSP, and p.input i means the i-th digital value above its TSP. In case of 1762 psi, the reset button is not pushed, so the value of Reset-delay-timer is fixed to 200. Test cases can be identified for all TSPs

in the same way. As seen in Tables 6 and 7, test cases with a reset counter value of 200 are more frequent than the other cases.

D. DISCUSSION

In order to verify software integrity, all developed test cases should be tested. If an error occurs during the test, the software should be debugged and the test set should be performed again. If there are many changes during debugging such that the variables of the program change, a new test set should be obtained by repeating the same process.

TABLE 8. High-probability test cases, error-free portion, and software failure probability.

State	Input	Operating mode	Number of test cases	Error-free portion	Software failure probability
state 1	input 1	full power	1	9.666.E-01	3.335.E-02
state 13402 to 13403	corresponding input	3 shut-down	4	1.800.E-02	1.535.E-02
state 6	input 1 to 5	start-up	9	1.222.E-02	3.136.E-03
state 8972 to 8976	corresponding 15 input	start-up	24	2.148.E-03	9.878.E-04
state 8977 to 8981	corresponding input	5 shut-down	29	5.816.E-04	4.063.E-04
state 1	input 2, 3	full power	31	6.009.E-05	3.462.E-04
state 2	input 1, 2	full power	33	6.009.E-05	2.861.E-04
state 3	input 1	full power	34	6.009.E-05	2.260.E-04
state 1	input 4, 5	full power	36	3.245.E-06	2.228.E-04
state 2	input 3, 4	full power	38	3.245.E-06	2.195.E-04
state 3	input 2, 3	full power	40	3.245.E-06	2.163.E-04
state 4	input 1, 2	full power	42	3.245.E-06	2.130.E-04
state 5	input 1	full power	43	3.245.E-06	2.098.E-04
…					
(3,831,421 cases)					0

A total of 3,831,421 exhaustive test cases were identified, a part of which are listed in Table 8 in order of their probability. The number of test cases can be reduced if additional data is provided; with insufficient data, test cases were chosen with all possible values to obtain conservative results. For example, based on detailed thermal-hydraulic simulation, certain ranges of parameters related to the start-up and shut-down processes can be identified to be physically implausible, thereby reducing the number of test cases. As expected, the probabilities of test cases for full-power operation are much greater than the others because most accidents happen at full power. Software failure probabilities are presented in Table 8 with the assumption that each test is completed with a successful result.

As described above, the developed test cases can be tested starting from the highly probable cases in order to efficiently obtain a certain amount of failure probability. For example, to obtain 1.E-3 and 1.E-4 of software failure probability, at least 29 and 3156 test cases should be tested, respectively. This number is considerably smaller than that of test cases from other test-based quantification methods; consider that if the Bernoulli process is applied, 3.E3 and 3.E4 successful tests are required to obtain the same 1.E-3 and 1.E-4 of software failure probability with a confidence level of 0.95. Again, the number of required test cases may be reduced based on their probability of occurrence, as some cases have a much higher probability than others.

This method allows for relatively less test cases to satisfy certain software failure rate goals. However, to accomplish exhaustive testing, 6,179,415 test cases need to be verified without omission. The large number of test cases requires a long time to fully complete, even though the required effort for one test case has been reduced. If the test cases are tested from the highly probable cases and the testing is finished after accomplishing the failure probability goal, most cases of shut-down and startup with low probabilities will not be verified.

V. CONCLUSION

In this study, a software test framework in consideration of the input sequence was suggested. This framework provides a relatively simple way of generating test cases that considers the structure of the software and actual operational profile. The total effort required for testing can be reduced through the suggested method since several scenarios can be combined into a representative test case. In addition, the proposed method contains state variables which further reduce testing time as there is no need to test long input sequences. By performing testing with this test set, it can be verified that there is no error in the software logic. The testing results can be utilized for software failure probability quantification since the test set is exhaustive. If a certain degree of software failure probability is required, test cases with high probability can be tested first. From the case study in this work, to obtain a software failure probability of 10^{-3}, less than 100 test cases are required.

Despite these reductions in testing effort, the number of required test cases for zero failure probability is still very large, thus requiring a long time to exhaustively perform testing. This problem might be resolved by test automation, or by using processor simulators. The test cases in the case study example here were developed for the most complicated type of logic among the various RPS software logics. Other simple logics generate less number of test cases because they have less variables, so this method will be useful in these cases. This study also can be extended to obtain test cases considering several software modules. Since safety instrumentation systems usually include logic solvers, the suggested method is applicable to other safety instrumentation system in various facilities.

The proposed framework focuses on the verification of the correctness of the logic when demand arrives. Other causes of errors should be investigated and understood further to completely model software failure. For example, testing cannot reveal faults in the requirements, and the suggested software failure quantification process cannot reflect the quality of

development. The software running environment also needs to be considered, such as interactions with hardware, the environmental impact of the entire system, and the effect of other software. External causes of error such as wrong input from the operator or noise also need to be considered.

NOMENCLATURE

NPP	Nuclear Power Plants
I&C	Instrumentation and Control
PSA	Probabilistic Safety Assessment
RPS	Reactor Protection System
V&V	Verification and Validation
FSM	Finite State Machine
FBD	Function Block Diagram
KNICS	Korea Nuclear Instrumentation & Control Systems project
BP	Bistable Processors
TSP	Trip SetPoint
LOCA	Loss of Coolant Accident
POS	Plant Operational States

REFERENCES

[1] National Research Council, *Digital Instrumentation and Control Systems in Nuclear Power Plants: Safety and Reliability Issues*. Washington, DC, USA: Academic, 1997.
[2] H. G. Kang and T. Sung, "An analysis of safety-critical digital systems for risk-informed design," *Rel. Eng. Syst. Safe*, vol. 78, no. 3, pp. 307–314, 2002.
[3] H. Ragheb, "Operating and maintenance experience with computer-based systems in nuclear power plants," in *Proc. Int. Workshop Tech. Support Licensing Issues Comput.-Based Syst. Important Safety*. Münich, Gemany, 1996, p. 9.
[4] *IEEE Recommended Practice on Software Reliability*, IEEE Standard 1633-2016, 2016.
[5] M. C. Kim, S. C. Jang, and J. Ha, "Possibilities and limitations of applying software reliability growth models to safety critical software," *Nucl. Eng. Technol.*, vol. 39, no. 2, pp. 145–148, 2007.
[6] N. Fenton, M. Neil, and D. Marquez, "Using Bayesian networks to predict software defects and reliability," in *Proc. Inst. Mech. Eng., O, J. Risk Rel.*, vol. 222, pp. 701–712, Dec. 2008.
[7] H. Eom, H. Son, H. Kang, and J. Ha, "A study of the quantitative reliability estimation of safety-critical software for probabilistic safety assessment," in *Proc. 4th ANS NPIC&HMIT*, Columbus, OH, USA, 2004, pp. 13–32.
[8] H. Eom, G. Park, H. Kang, and S. Jang, "Reliability assessment of a safety-critical software by using generalized Bayesian nets," in *Proc. 6th ANS NPIC&HMIT*, 2009.
[9] K. Goševa-Popstojanova and K. S. Trivedi, "Architecture-based approach to reliability assessment of software systems," *Perform Eval.*, vol. 45, no. 7, pp. 179–204, 2001.
[10] J. May, G. Hughes, and A. D. Lunn, "Reliability estimation from appropriate testing of plant protection software," *Softw. Eng. J.*, vol. 10, no. 6, pp. 206–218, Nov. 1995.
[11] K. W. Miller *et al.*, "Estimating the probability of failure when testing reveals no failures," *IEEE Trans. Softw. Eng.*, vol. 18, no. 1, pp. 33–43, Jan. 1992.
[12] G. B. Finelli, "NASA software failure characterization experiments," *Rel. Eng. Syst. Safe*, vol. 32, nos. 1–2, pp. 155–169, 1991.
[13] M. R. Lyu, "The operational profile," in *Handbook of Software Engineering and Knowledge Engineering*. Los Alamitos, CA, USA: McGraw-Hill, 1996, pp. 167–216. [Online]. Available: http://www.cse.cuhk.edu.hk/~lyu/book/reliability/
[14] *Software and Systems Engineering Software Testing Part 1: Concepts and Definitions*, IEEE Standard ISO/IEC/IEEE 29119-1:2013(E), 2013, pp. 1–64.
[15] H. G. Kang, H. G. Lim, H. J. Lee, M. C. Kim, and S. C. Jang, "Input-profile-based software failure probability quantification for safety signal generation systems," *Rel. Eng. Syst. Safe*, vol. 94, no. 10, pp. 1542–1546, 2009.
[16] J. C. Laprie and K. Kanoun, "Software reliability and system reliability," in *Handbook for Software Reliability Engineering*. New York, NY, USA: McGraw-Hill, 1996, pp. 27–69.
[17] P. Rook, "Software fault tolerance," in *Software Reliability Handbook*. New York, NY, USA: Elsevier, 1990, pp. 83–110.
[18] J. A. McDermid, "Fault-tolerant systems structuring concepts," in *Software Engineer's Reference Book*. New York, NY, USA: Elsevier, 2013, p. 61.
[19] G. Y. Park, K. Y. Koh, E. Jee, P. H. Seong, K.-C. Kwon, and D. H. Lee, "Fault tree analysis of KNICS RPS software," *Nucl. Eng. Technol.*, vol. 40, no. 5, pp. 397–408, 2008.
[20] R. Tregoning, L. Abramson, and P. Scott, "Estimating Loss-of-Coolant Accident (LOCA) frequencies through the elicitation process," U.S. Nucl. Regulatory Commission, Rockville, MD, USA, Tech. Rep. NUREG-1829, 2005.
[21] L. W. Sang, L. J. Sung, and H. J. Kyu, "A study on the low power and shutdown PSA during the standard design phase," in *Proc. 7th Korea-Japan PSA Workshop*.

HEE EUN KIM received the B.S. degree in electrical engineering and the M.S. degree in nuclear and quantum engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2011 and 2013, respectively, where she is currently pursuing the Ph.D. degree in nuclear engineering.

Her research interests include probabilistic safety assessment, and the safety and cyber security of nuclear power plant digital instrumentation and control systems.

HAN SEONG SON received the Ph.D. degree in nuclear engineering from KAIST in 2000. He has been an Assistant Professor with Joongbu University since 2008.

His research interests include software engineering, software reliability, and cyber security, among others.

BO GYUNG KIM received the Ph.D. degree in nuclear engineering from KAIST in 2016. She has been a Senior Researcher with the Korea Institute of Nuclear Safety since 2016.

Her research interests include probabilistic safety assessment and accident sequences of nuclear power plants, among others.

JAEHYUN CHO received the B.S. degree in nuclear engineering and the Ph.D. degree in energy system engineering from Seoul National University, Seoul, South Korea, in 2008 and 2013, respectively. He is currently with KAERI as a Senior Researcher.

His research interests include digital instrumentation and control, severe accident probabilistic safety assessment, and passive system reliability.

SUNG MIN SHIN was born in South Korea in 1984. He received the B.S. degree in mechatronics engineering from the Korea University of Technology and Education, Cheonan, South Korea, in 2010, and the M.S. degree in mechanical engineering and the Ph.D. degree in nuclear and quantum engineering from KAIST, Daejeon, South Korea, in 2012 and 2016, respectively.

He is currently a Senior Researcher with the Korea Atomic Energy Research Institute. His research interests include the field of safety component monitoring, probabilistic safety assessment, and digital instrumentation and control.

HYUN GOOK KANG received the Ph.D. degree in nuclear engineering from KAIST in 1999.

After receiving his Ph.D. degree, he was with KAERI as a Senior Researcher for more than ten years. A nuclear engineering expert, he is currently working as an Associate Professor with Rensselaer Polytechnic Institute.

His research interests include interconnected specialties that lie in innovations of the risk assessment of safety-critical systems, intrinsically safe nuclear power, intelligence of control and protection, and the design and evaluation of emergency procedures.