KIPS Tr. Comp. and Comm. Sys.
Vol.5, No.9 pp.275~284 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2016.5.9.275

The Software Reliability Evaluation of a Nuclear Controller Software Using a Fault Detection Coverage Based on the Fault Weight

Young-Jun Lee† · Jang-Soo Lee†† · Young-Kuk Kim†††

ABSTRACT

The software used in the nuclear safety field has been ensured through the development, validation, safety analysis, and quality assurance activities throughout the entire process life cycle from the planning phase to the installation phase. However, this evaluation through the development and validation process needs a lot of time and money, and there are limitations to ensure that the quality is improved enough. Therefore, the effort to calculate the reliability of the software continues for a quantitative evaluation instead of a qualitative evaluation. In this paper, we propose a reliability evaluation method for the software to be used for a specific operation of the digital controller in a nuclear power plant. After injecting weighted faults in the internal space of a developed controller and calculating the ability to detect the injected faults using diagnostic software, we can evaluate the software reliability of a digital controller in a nuclear power plant.

Keywords : Software Reliability, Fault Injection, Fault Weight, Fault Detection Coverage

가중치 기반 고장감지 커버리지 방법을 이용한 원전 제어기기 소프트웨어 신뢰도 평가

이 영 준† · 이 장 수†† · 김 영 국†††

요 약

원자력분야에서 사용되는 안전관련 소프트웨어는 계획단계부터 설치단계까지의 전 생명주기 공정을 통해 개발과 확인검증, 안전성 분석, 그리고 품질보증 활동을 수행해 소프트웨어의 안전성을 보장하고 있다. 그러나 이러한 개발과 검증공정을 통한 평가는 시간과 비용을 많이 필요로 한다. 또한, 소프트웨어의 품질을 향상시키기 위해 다양한 활동을 수행했다고 주장하지만, 어느 정도의 품질이 향상되었는지 확인하기에는 한계가 있다. 이러한 한계를 극복하기 위해서 정량적인 평가를 수행할 수 있는 소프트웨어 신뢰도 계산 방법을 제안한다. 특히, 소프트웨어가 사용하는 메모리 공간에 고장을 주입하여 소프트웨어의 고장을 모사하고, 주입된 고장에 가중치를 부여하여 고장 민감도에 차이를 두고, 감지능력을 평가하여 소프트웨어 고장율을 계산한다. 이러한 고장율을 활용하여 소프트웨어 신뢰도 계산을 수행하면 정량적인 평가결과를 획득할 수 있게 된다.

키워드 : 소프트웨어 신뢰도, 고장 주입, 고장 가중치, 고장 감지 커버리지

1. 서 론

원자력 발전소의 다양한 계통에서 사용되는 제어기 대부분이 디지털 기반으로 개발을 완료하고 있고, 기존의 아날로그 제어기기도 디지털 제어기기로의 교체를 점점 진행해가고 있다. 따라서 디지털 제어기기 내부에서 동작하는 소프트웨어의 중요성이 더욱 높아지고 있는 것이 현실이다. 원전에서 사용되는 소프트웨어의 안전성을 보장하기 위해서 미국 규제기관인 United State Nuclear Regulatory Commission (US-NRC)는 Software Review Plan (SRP)[1]를 발표하였고, Regulatory Guide 1.152, IEEE Standard 7-4.3.2[2] 요건에 맞추어 디지털 컴퓨터 안전 계통을 개발하고 검증할 것을 요구하고 있다. Fig. 1은 디지털 컴퓨터 안전 계통의 안전요건, 개발공정, 그리고 확인 및 검증 공정의 대표적인 지침과 표준문서를 보여준다.

† 정 회 원 : 한국원자력연구원 선임연구원
†† 비 회 원 : 한국원자력연구원 책임연구원
††† 종신회원 : 충남대학교 컴퓨터공학과 교수
Manuscript Received : August 17, 2016
Accepted : August 29, 2016
* Corresponding Author : Young-Kuk Kim(ykim@cnu.ac.kr)

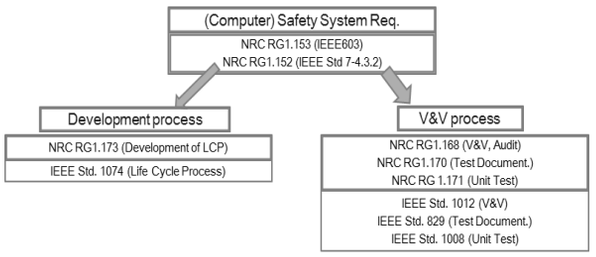


Fig. 1. Guidelines and Standards for nuclear safety software

이러한 규제요건을 만족하기 위해서 원자력분야에서 사용되는 안전관련 소프트웨어는 계획단계부터 설치단계까지의 전 생명주기 공정을 통해 개발과 확인검증, 안전성 분석, 그리고 품질보증 활동을 수행해 소프트웨어의 안전성을 보장하고 있다[3]. 그러나 이러한 개발과 검증공정을 통한 평가는 시간과 비용을 많이 필요로 한다. 또한, 소프트웨어의 품질을 향상시키기 위해 다양한 활동을 수행했다고 주장하지만, 어느 정도의 품질이 향상되었는지 확인하기에는 한계가 있다. 소프트웨어 확인 및 검증에 관련한 Regulatory Guide 1.168 지침에서도 소프트웨어의 신뢰도를 측정할 것을 요구하고 있다. 따라서 정성적인 평가와는 별개로 정량적인 평가를 위해 소프트웨어의 신뢰도를 구하려는 시도가 계속되고 있다.

본 논문은 원전 디지털 제어기기의 특수한 동작을 위해 사용되는 소프트웨어에 대한 신뢰도 평가방법을 제안하고자 한다. 개발공정에 따라 완료된 제어기기의 소프트웨어 내부 공간에 임의의 고장을 주입하고, 주입된 고장을 진단 소프트웨어가 얼마나 잘 감지해 내는 지 계산해내어 소프트웨어 신뢰도를 평가해보고자 한다. 본 논문은 다음과 같이 구성되어 있다. 2절에서는 기존의 원전 분야에서 연구하고 있는 소프트웨어 신뢰도 평가방법을 기술하고, 3절에서는 본 논문에서 제안하려고 하는 신뢰도 평가방법을 설명한다. 3.1절에서는 원전 제어기기의 동작특성을 살펴보고, 3.2절에서는 새로운 신뢰도를 계산하기 위해 고려할 사항들을 정의하며, 3.3절에서는 제안된 신뢰도 계산식을 설명한다. 4절은 시험방법과 결과를 기술하고, 5절에서 결론을 맺는다.

2. 원전 소프트웨어의 신뢰도 평가 연구

원전 분야에서 소프트웨어의 신뢰도를 평가하기 위한 연구는 현재 진행형이다. 소프트웨어는 하드웨어 부품과는 달리 시간이 지나도 노후화가 진행되지 않는다는 전제 때문에 소프트웨어의 신뢰도를 정량적으로 계산하는 것은 불가능하다고 주장한다. 따라서 소프트웨어 신뢰도는 직접 계산하는 것이 아니라 일정 목표 수준의 신뢰도를 확보하기 위해서 필요한 시험 또는 다른 행위들이 어느 정도 수행되어야 하는 지에 초점이 맞춰진다. 이러한 목적에 맞추어 지금까지 연구되어온 방법은 Software Reliability Growth Model (SRGM)[4]과 Bayesian Belief Net (BBN)[5] 방법들이다. 이러한 방법들이 연구되고는 있으나 원전 소프트웨어의 특수성을 고려할 때 현장에 직접 적용하기에는 아직 시기상조이다. 연구 방법론이 검증되고 결과가 객관적으로 입증된 이후에야 적용여부를 고려할 수 있기 때문이다.

현재 원전 분야에서 연구하고 있는 소프트웨어 신뢰도 평가방법을 간략하게 소개하면 다음과 같다.

2.1 소프트웨어 신뢰도 성장 모델 (Software Reliability Growth Model)

소프트웨어 신뢰도 성장모델(SRGM: Software Reliability Growth Model)은 소프트웨어 고장 발생 시 소프트웨어에 내재하고 있던 결함을 제거함으로써, 이를 유발할 수 있는 무작위(random) 조건이 발생하더라도 이러한 결함에 의해서는 소프트웨어 고장이 다시는 발생하지 않기 때문에 소프트웨어 신뢰도는 향상되었다는 가정을 수립하는 것으로써, 소프트웨어 고장데이터를 기반으로 통계적 방법에 의하여 가정한 모델의 모수를 추정하여 소프트웨어 신뢰도를 파악하는 기법이다.

소프트웨어 결함은 모터나 밸브와 같이 시간이 지남에 따라 마모나 손상에 의하여 발생되는 것이 아니라, 소프트웨어를 개발하거나 구현 시 소프트웨어 설계자나 프로그래머에 의하여 발생하며 이는 결정론적 방식으로 고장을 유발한다. 그러나 소프트웨어 결함에 의하여 발생하는 고장현상은 운전환경이나 입력 값의 변화에 의하여 결함이 외형상으로 나타날 조건이 되었을 때, 소프트웨어에 의한 고장이 발생하며, 이러한 발생현상은 통계적으로 처리 가능한 현상으로 간주될 수 있다. 따라서 기존 통계처리에 의한 소프트웨어 신뢰도 기법을 소프트웨어 고장에 적용할 수 있게 된다.

일반적으로 SRGM 에는 두 개의 주요 변수가 있다. 하나는 소프트웨어에 잔존하는 소프트웨어 결함 수이고 또 다른 하나는 소프트웨어 고장율이다. 이러한 변수는 소프트웨어 모델에 모수로 되어 있어서, 지금까지 발생한 소프트웨어 고장 데이터를 기반으로 통계적 방법을 통하여 이러한 모델 모수를 추정한다.

SRGM 은 수학적인 모델이고, 고장이 검출되고 회복되는 것에 따라 소프트웨어 신뢰도가 어떻게 향상되는 지를 보여준다. SRGM 은 주어진 신뢰도 수준을 획득하기 위해 시험을 언제 멈추어야 하는 지를 결정하는 데에 사용되기도 한다. 소프트웨어 성장모델은 다양한 방법론이 존재한다. 가장 흔하게 사용되는 소프트웨어 신뢰도 모델은 JM(Jelinski-Mornada) 모델, Go(Goel-Okumoto) 모델, MO(Musa-Okumoto) 모델, Sch 모델, S-Shape 모델 등이 있다[6]. 이러한 모델들의 예측능력을 평가하기 위해서는 의미 있는 측정치를 사용하는 것이 필요하다. 여기에는 두 가지 기준이 존재하는데 평균제곱근 오차(RMSE: Root Mean Square Error)와 평균 에러(AE: Average Error)이다. RMSE 는 추정 값 또는 모델이 예측한 값과 실제 환경에서 관찰되는 값의 차이를 다룰 때 흔히 사용되는 측도이다. 정밀도(precision)를 표현할 때 적합하다. 각각의 차이값은 잔차(residual)라고도 하며, 평균 제곱근 편차는 잔차들을 하나의 측도로 종합할 때 사용된다. 이

러한 기준은 실제 값과 예측 값 사이의 차이를 측정하기 위해 사용되는 것이다. 두 가지 공식은 다음과 같이 표현될 수 있다.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(c(k) - \hat{c}(k)\right)^2} \quad (1)$$

$$AE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{c(k)-\hat{c}(k)}{c(k)}\right| \times 100 \quad (2)$$

여기서 n은 고장 데이터의 그룹 개수이고 $c(k)$는 고장데이터의 각각 그룹에 있는 실제 고장들의 개수이며, $\hat{c}(k)$는 예측된 고장들의 개수이다. RMSE와 AE가 더욱 작아질수록 모델예측 능력은 더욱 더 강해진다.

2.2 Bayesian Belief Net

Bayesian Belief Net (BBN)[7] 은 대상 시스템의 관련된 변수들을 인과관계에 의해 모델링하고 변수들 간의 종속성 정도를 조건부 확률로 나타낸 다음 관찰된 여러 가지의 증거를 만들어진 BBN 모델에 입력한 후 베이스(Bayes) 확률 정리를 비롯한 확률 법칙을 적용하여 계산하고 정량적 결과를 이끌어 내는 방법론이다.

BBN은 그래프 상에서 원으로 표시되는 노드(Node)와 노드들 사이를 연결하는 연결선(arcs 또는 directed edges) 그리고 각 노드에 속한 확률 테이블(Node Probability Tables: NPT 또는 Conditional Probability Table: CPT)로 구성되어 있다. 노드는 모델에 포함된 변수들을 나타내며 노드 연결선은 노드간의 인과관계를 나타낸다. 각 노드는 무작위 변수로서 몇 개의 상태를 가지고 있으며(예: "Yes"와 "No"의 상태) 각 상태의 확률 값의 합은 1이 된다. 각 노드에 연결된 노드 확률 테이블은 노드간의 연결 강도를 결정하며 모 노드(parent node)의 각 상태에 대한 조건부 확률로 표현된다.

BBN의 최대 특징은 인과관계에 의한 모델링과 불확실성의 명시적 모델링 그리고 다양한 유형의 증거(정성적 평가 증거와 정량적 평가 증거 모두)를 일관된 프레임 안에서 정형적으로 처리하여 정량화 시킬 수 있다는 점이다. 안전 소프트웨어의 신뢰도나 안전성 평가에는 인지적 한계와 현실적 제약이 수반되므로 필연적으로 불확실성이 포함되고 또 평가에 필요한 모든 증거를 얻을 수 없는 것이 일반적 상황이다. BBN은 이런 모든 상황을 포함한 모델링을 할 수 있다는 점에서 기존 방법에 비하여 장점을 가진다.

BBN 방법론을 개발중인 시스템에 적용한 사례가 있다. 이 사례는 한국원자력연구원에서 수행한 연구이며 연구보고서 KAERI-RR-2794[8]에 기술되어 있다. 인허가 및 개발단계에서 사용되고 있는 소프트웨어의 정성적인 신뢰도 평가 체계를 BBN으로 모델링하고, 기존의 SW 엔지니어링 척도에서 얻어진 정량적 결과 및 V&V와 같은 정성적 평가 결과를 정량화한 입력을 사용하여 최종적으로 소프트웨어의 정량적 신뢰도 정보를 획득하는 것이다.

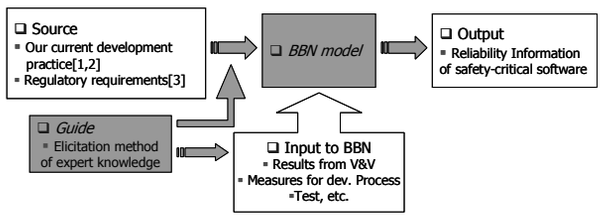


Fig. 2. In/Out information for BBN model

3. 원전 제어기기에 특화된 소프트웨어 신뢰도 평가방법 제안

지금까지 살펴본 소프트웨어 신뢰도 방법은 원전 소프트웨어에 적용하기 위해 연구되어 왔으나 실제로 적용하여 평가한 사례는 존재하지 않는다. SRGM 은 소프트웨어가 완성된 이후의 고장 및 해결사례가 있어야 신뢰도가 성장되고 있음을 보일 수 있으나 적용을 위한 데이터가 충분히 확보되지 않았고, BBN방법론은 정성적인 판단요소가 많은 부분을 차지하고 있어 정량적인 데이터를 계산해내는 데에 한계가 있다. 이러한 단점들을 극복하고, 원전에서 사용하는 소프트웨어의 정량적인 신뢰도 값을 구하기 위해 새로운 방법을 제안하고자 한다. 간략하게 설명하면, 이 방법은 시스템 신뢰도를 계산하기 위해 사용된 방법에 원전 제어기기에서 사용하는 소프트웨어의 특성을 고려한 맞춤형 신뢰도 계산 방법이라고 할 수 있다. 원전 제어기기에서 사용하는 소프트웨어는 동작 특성과 기능, 그리고 동작 주기가 일정한 임베디드 프로그램이므로 프로그램의 구조는 설계명세서를 통해 확인할 수 있고, 프로그램이 사용하는 리소스들 무엇인지 파악할 수 있다. 따라서 신뢰도를 계산하기 위해 필요한 고장데이터를 정확한 위치에 주입할 수 있고, 이를 이용하여 소프트웨어의 고장감지율을 계산해낼 수 있게 된다. 원전 제어기기의 소프트웨어 동작 특성을 파악하고, 신뢰도를 계산하기 위해 고려되어야 할 사항, 그리고 신뢰도 평가를 위해 제안된 방법을 살펴보고자 한다.

3.1 원전 제어기기 소프트웨어의 동작 특성

원전 안전계통에서 사용하는 Programmable Logic Controller (PLC) 제어기기는 프로세서(CPU) 모듈, 입출력(I/O) 모듈, 통신(Communication) 모듈, 전원(Power) 모듈, 버스(Bus) 모듈 등 다양한 모듈로 구성되어 있다. 이러한 모듈들은 외부로부터의 입출력 데이터를 송수신하고, 통신 모듈을 통해 타 채널 계통시스템과 정보를 교환한다. 프로세서모듈은 수신된 데이터를 이용해 안전 응용프로그램을 동작시키면서 보호계통 시스템의 안전을 위한 제어신호를 생성한다.

소프트웨어가 위치해 있는 프로세서모듈은 외부 모듈들과 공유메모리를 통해 데이터를 송수신하고 있고, 공유메모리의 영역을 서로 분리하여 할당하고 있다. 또한 제어기기를 구성하고 있는 하드웨어 모듈들은 자제적으로 자가진단 기능을 가지고 있어서 내부적인 오류에 대해 진단 및 처리를

수행하고 프로세서모듈의 특수메모리를 이용하여 그 결과를 통보한다. 공유메모리는 외부모듈의 입력 값, 연산결과 값, 그리고 하드웨어의 상태정보를 포함하고 있다.

제어기기는 반복적인 일들을 시간주기에 따라 수행하기 때문에 소프트웨어가 사용하는 메모리 영역이 어느 곳인 지 알 수 있다. 이는 소프트웨어가 사용하는 메모리의 공간에 임의의 고장을 주입하였을 경우 소프트웨어의 고장이 발생하였다고 유추할 수 있다. 소프트웨어의 고장을 유추할 수 있다면, 소프트웨어 고장에 대한 감지능력을 계산할 수 있고, 이는 소프트웨어 신뢰도를 계산하기 위한 중요한 요소로 사용될 수 있게 된다.

3.2 신뢰도 계산 고려 사항

1) 원전 제어기기 내부의 고장 정의

원전 제어기기 소프트웨어에서 발생하는 고장에 대한 새로운 정의가 필요하다. 고장이란 그 결과로 인해 시스템의 실패를 가져오기 전 단계를 의미하므로, 고장이 발생하였다고 해서 시스템이 무조건 실패하는 것은 아니다. 고장이 안전기능을 수행하는 프로그램이나 시스템 태스크에 영향을 주어 시스템의 실패로 결론날 수도 있으나 그렇지 않을 수도 있다. 예를 들어 메모리의 특정공간에 고장이 발생하였을지라도, 해당 메모리를 소프트웨어가 사용하지 않는다면, 발생된 고장은 고장이라고 정의할 수 없다.

Program Counter (PC)	0xD000111E								•	•	•					•
	0xD000111F														•	•
	0xD0001120													•	•	•
	0xD0001121	•	•	•								•	•	•		
	0xD0001122										•		•			

Fig. 3. Effectiveness of the injected faults

Fig. 3에서 살펴보는 것처럼, 프로그램 카운터(PC)가 가리키는 0xD0001120, 0xD0001122 위치는 소프트웨어가 사용하지 않은 메모리 공간이다. 이 위치에 내재되어 있는 오류는 프로그램의 동작에는 전혀 영향이 발생하지 않는다.

따라서 원전 제어기기의 고장을 정의할 때는 제어기기내에서 수행 중인 응용프로그램이 사용하는 위치인지, 사용하지 않는 위치인지 구분하는 것이 필요하다. 응용 프로그램이 사용하지 않는 위치의 고장은 고장에서 배제한다. 실제로 고장이 발생하였다고 하더라도 소프트웨어 동작에 전혀 영향을 주지 않는다면 그 고장을 감지해내는 것이 현실적으로 어렵기도 하고, 불필요한 부분까지의 고장을 감지할 수 있는 진단 프로그램을 개발하는 것도 쉽지 않다. 또한 복잡한 진단 알고리즘으로 구현되었을 경우 고장에 대한 실시간 감지를 보장할 수 없고, 동작하기 위해서는 많은 시간과 비싼 비용이 필요하기 때문에 감지기능을 제대로 수행하지 못할 수도 있다. 고장은 그 고장이 제어기기 소프트웨어의 동작에 영향을 줄 수 있는 지에 대한 여부를 가지고 고장의 유효성을 부여하게 된다.

유효한 고장인지를 판단하기 위해서는 임의의 고장 개수를 가지고 표본을 이용해 확률적으로 계산할 수도 있지만, 실제 실험을 통해 프로그램 카운터(PC)가 가리킨 물리적인 주소를 파악하여 고장의 유효성을 판단하는 것도 가능하다.

안전 프로그램의 1 cycle 동안 90,020 개의 프로그램 카운터를 지났지만, 중복 경로를 제외하면 26,726개의 메모리 주소를 응용프로그램에서 사용한다. 따라서 이 위치에 발생된 고장만이 고장으로써 인정받을 수 있게 된다.

Fig. 4. Memory address used by application program

2) 고장 영향 인자

고장이 가지고 있는 특성에 따라 소프트웨어에 미치는 영향은 달라질 수 있다. 소프트웨어에 보다 많은 영향을 미치는 고장은 그렇지 않은 고장보다는 소프트웨어의 내구성을 더욱 약화시킬 수 있고, 고장율을 증가 시킬 수도 있다. 소프트웨어에 미치는 영향을 파악할 때 고려하여야 할 몇 가지 인자들은 다음과 같다.

Fault {type, duration, location, weight, recovery}

고장의 형태(fault type), 고장이 지속된 시간(fault duration), 고장이 생긴 위치(fault location), 고장이 가지고 있는 가중치(fault weighting), 발생한 고장의 회복력(recovery) 등이 고려해야 할 인자들이다. 이중에서 특히 가중치값 속성은 시스템의 고장감지 커버리지 계산에 가장 많은 영향을 줄 수 있다.

소프트웨어의 코드와 데이터가 위치하는 곳이 하드웨어 메모리이므로 메모리에 대한 특성을 고려할 필요가 있다. 4장에서 시험환경에 대해 기술한다.

디지털 시스템에서 사용되는 하드웨어는 서로 다른 재료 및 특성을 가지고 있다. 또한 안전 소프트웨어에 의해 접근되는 하드웨어의 종류가 다르며, 고장이 발생하였더라도 그 중요도는 서로 다를 수 있다. 이러한 조건을 반영하기 위해서 하드웨어마다 서로 다른 가중치 값을 정의할 수 있다. 고장에 대한 가중치 값은 하드웨어의 내구성, 그리고 접근 횟수 등 다양한 조건을 가지고 결정되나, 현재는 하드웨어의 물질적인 특성을 고려하는 것은 불가능하다. 따라서 이러한 가중치 값은 하드웨어의 고장으로 인해 안전소프트웨어가 받을 수 있는 영향의 정도에 따라 정의할 수 있다. 소프트웨어의 접근횟수, 안전데이터에 대한 입출력, 그리고 안전기능의 고장으로 인한 시스템적인 오류의 영향 정도를 가지고 가중치 값을 부여할 수 있다.

상기와 같은 전제에서 고장을 정량적으로 정의하면 다음과 같다.

- **Fault Type = 1 (0 fault, 1 fault)**

고장 형태는 0 또는 1 로 고착되는 고장을 고려한다. 소프트웨어는 코드영역과 데이터영역으로 구분되지만 메모리 공간에서 동작되고, 데이터의 입출력도 메모리공간에서 수행된다. 소프트웨어 오류를 하드웨어적으로 발생시키기 위한 고장주입 방법은 메모리에서 이루어지는데 반도체 특성상 0 또는 1로 고착될 수 있다. 하드웨어에 주입되는 메모리 고장은 두 개의 고장중 하나의 형태를 가지므로 해당 bit는 1/2 의 확률을 가지고 고장의 타입이 결정된다.

- **Fault Duration = 1**

고장이 지속되는 정도는 고장을 정의하기 위한 속성중의 하나이다. 고장이 발생해서 영구적인 고장으로 지속되기도 하지만, 고장이 일시적으로 발생하였다가 시간이 지나면서 다시 회복되는 경우도 존재한다. 본 연구에서 고려한 고장은 Stuck 고장이며 이는 일시적인 고장이 될 수 없고 영구적인 고장을 의미한다.

- **Fault Location = 가중치1**

고장의 위치도 고장을 정량화하기 위한 속성중의 하나이다. 여기에서 말하는 위치는 소프트웨어에서 변수 및 데이터를 위해서 사용하는 메모리의 bit 단위 위치이다. 제어기기 소프트웨어는 32bit 를 연산의 기본단위로 사용하므로 Fault 위치는 0bit~31bit 중 하나의 위치이다. 고장이 어느 위치에 발생되는 지 파악하는 것도 중요하다. 임의의 위치에 발생된 고장이 최상위 비트에 속해있는 지, 아니면 최하위 비트에 속해있는 지에 따라 고장의 정량화에 영향을 줄 수 있다. 고장의 위치는 가중치1의 항목으로 정의될 수 있고 표본을 이용한 정규화 분포를 통해 비트위치에 대한 가중치를 계산할 수 있다.

가중치1 의 데이터를 계산하기 위해서는 평균값을 이용한 정규화를 사용한다. 정규화 방법은 z-transformation을 사용한다. 이는 데이터에서 평균을 뺀 이후 표준편차로 나누는 방법인데 데이터의 normal 분포를 가정하지 않아도 된다. 평균값으로 정규화된 값은 다음과 같이 표현한다.

$$\tilde{d}_i = \frac{d_i - E(d)}{\sigma_d} \tag{3}$$

$$E(d) = \frac{1}{N}\sum_{i=1}^{n} d_i \tag{4}$$

$$\sigma_d = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(d_i - E(d))^2} \tag{5}$$

$\tilde{d}_i$ = 정규화값, $E(d)$ = 표본평균, σ_d = 표본표준편차

표본을 사용하여 계산한 비트위치의 가중치 값은 4.3절에 기술한다.

- **Fault Weight = 가중치2**

안전등급 제어기기에서 동작하는 소프트웨어는 동일한 동작을 반복하고, 반복된 동작에서 통신으로부터 받은 데이터를 이용해 계산하고, 진단동작을 수행한다. 1 cycle 동안 접근하는 메모리의 코드영역과 데이터영역은 고정되어 있다. 그러나 접근하는 횟수는 서로 상이하다. 많은 횟수로 접근하는 메모리의 공간에 주입된 고장이 더 많은 영향을 줄 수 있는 확률이 증가하기 때문에 접근하는 횟수에 따라 가중치를 부여하는 것이 타당하다. 가중치2는 정규화 연산을 사용하지 않는다. 정규화 연산은 모집단을 위해 표본을 사용하는 것이지만 가중치2는 전수 시험을 통해 가중치 값을 부여할 수 있다. 접근횟수에 따라 카테고리를 분류하였고, 분류된 카테고리별로 부여할 수 있는 가중치는 다음과 같이 계산한다.

$$W_{access(i)} = \frac{access(i)}{\sum_{i=1}^{n} access(i)} \tag{6}$$

$$\sum_{i=1}^{n} access(i) = 26{,}725 \tag{7}$$

Equation (7)은 실험을 통해 확인한 메모리 접근 총 횟수이다. 접근횟수의 구간평균 값을 접근 총 횟수를 이용해 가중치를 부여할 수 있게 된다. 따라서 발생되는 고장 하나하나가 가지는 민감도는 다음과 같이 계산할 수 있다.

고장 민감도 = TYPE × DURATION ×
LOCATION_WEIGHT1 ×
COUNT_WEIGHT2

3.3 제어기기 소프트웨어 신뢰도 계산 방법

신뢰도를 평가하기 위해서는 신뢰도 평가식과 고장율 데이터, 그리고 고장율을 계산하기 위해서 고려되어야 할 요소들이 필요하다.

첫째 신뢰도 평가식은 일반적으로 사용하는 신뢰도 계산 방식을 사용한다. 이 방법은 전자부품에 대한 신뢰도 계산 방식이므로 노후되지 않는 소프트웨어 신뢰도에 적용하기에는 무리가 있어 보이나, 소프트웨어도 시간이 지나면서 버그에 노출되는 시간이 증가하므로 노후화되고 있다고 가정하였다.

둘째, 신뢰도 평가식에 사용할 고장율을 새로 정의해야 한다. 소프트웨어에 발생하는 고장은 시험방식을 통해 확보할 수 있다. 시험대상 소프트웨어가 사용하는 위치에 임의의 고장을 주입하고, 소프트웨어 진단기능을 통해 고장검출율을 파악하면 대상 소프트웨어의 고장율을 확인할 수 있다. 이 때, 고장율을 계산하기 위해서는 3.2절에서 정의한 고장영향 인자들을 주입된 고장에 적용하여야 한다. 고장의 민감도에 차등을 두어 소프트웨어에 주는 영향을 계산하는 것이 합리적이다.

신뢰도는 하드웨어의 부품이 일정한 시간 동안 지속적으로 동작하는 확률을 표현하는 방법으로 다음과 같이 표현한다.

$$R(t) = Pr(T \geq t) = 1 - Pr(T < t) = 1 - F(t) = 1 - \int_0^t f(t)dt \quad (8)$$

$$R(t) = 1 - F(t) = e^{-\lambda t} \quad (9)$$

여기에서 F(t)는 고장누적분포 함수이며 시스템이 t 시간 내에 고장 날 확률을 의미하고 다음과 같이 표현한다.

$$F(t) = Pr(T \leq t) = \int_0^t f(t)dt, t \geq 0 \quad (10)$$

$$F(t) = \int_0^t \lambda e^{-\lambda t}dt = 1 - e^{-\lambda t} \quad (11)$$

또한 고장누적분포 함수에서 사용되는 인자인 $\lambda(t)$는 단위시간 당 시스템의 고장 횟수를 의미한다.

신뢰도를 계산하기 위한 기본적인 방법에서 가장 중요한 인자는 고장율이다. 단위시간 당 시스템의 고장 횟수에 따라 신뢰도 계산값이 달라지기 때문이다. 상수값으로 표현되는 고장율을 계산하기 위해서는 다양한 방법이 있으나 시험데이터 및 분석데이터를 이용해 확률분석 방법을 통해 고장율 값을 추정하고, 추정된 값을 이용해 신뢰도를 계산하는 것이 일반적인 방법이다. $\lambda(t)$는 단위시간당 고장 횟수를 의미하지만 고장율을 구하면 고장 횟수로 사용할 수 있다.

고장율 계산은 다음과 같다.

$$\lambda(t) = 1 - C \quad (12)$$

$$C = Pr(\text{fault detected} \mid \text{fault existence}) = {\sum F\,iDi}/{\sum F\,i} \quad (13)$$

$$\lambda(t) = 1 - {\sum F\,iDi}/{\sum F\,i} \quad (14)$$

λ(t): 고장율, ∑F i: 주입고장, ∑F iDi: 감지고장,
C: 고장감지 커버리지

주입된 고장과 감지된 고장의 비율에 따라 고장감지 커버리지를 계산할 수 있고, 이를 근거로 고장율을 계산해낼 수 있다.

이러한 방법을 사용하기 위해서는 시험데이터 및 분석데이터가 충분해야 한다. 하지만 원전 제어기기가 안전계통에 적용되고 발전소가 동작되는 상황에서 시험데이터를 추출하기에는 한계가 있고, 그 표본도 극히 적은 수이므로 통계에 사용하기에 부적합한 면이 있다. 또한 개발과정에서 수행하는 시스템 시험은 개발이 완료되기 이전이므로 시험결과값을 고장율의 대표값으로 정하는 것도 합리적이지 않다. 따라서 이러한 단점을 극복하기 위해서 개발이 완료된 제어기기의 소프트웨어에 임의의 고장을 주입하고, 시스템의 진단기능을 통해 고장율을 계산하여 신뢰도를 구하기 위한 인자로 사용할 수 있게 된다.

4. 시험 방법 및 결과

4.1 고장 주입 환경 및 대상

지금까지 살펴본 고장에 대한 정의와 고려사항들은 메모리 공간에 주입되는 고장들을 의미하지만 소프트웨어가 사용하는 공간이므로 소프트웨어의 고장이라고 가정할 수 있다. 실제 소프트웨어가 사용하는 메모리의 고장은 소프트웨어 코드나 데이터의 변경이라고 볼 수 있다. 시험을 위해서 에뮬레이터를 활용한 고장 모사 장치 및 프로그램을 개발하였다. 고장 모사 장치는 소프트웨어를 이용해 메모리의 특정비트 공간에 Stuck-at-0 또는 Stuck-at-1 값을 프로그램이 시작해서 끝나는 시간까지 동일 값을 유지시켜서 영구고장을 일으키는 효과를 발생시킨다. Fig. 5는 고장을 주입하기 위한 시험환경으로 자체 개발하여 사용하였다.

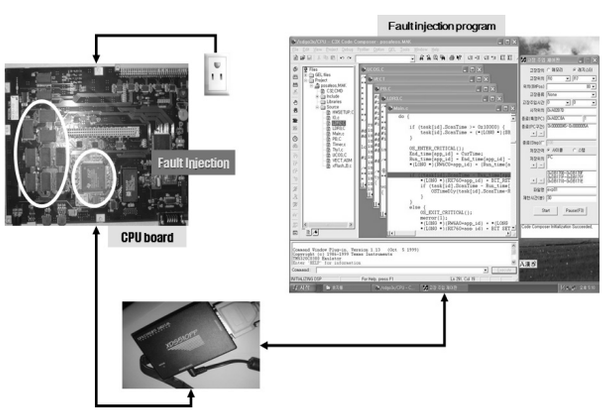


Fig. 5. Environment for injecting the faults into software memory

4.2 시험대상

고장을 주입하고 고장감지 커버리지를 계산하기 위해서 원자력 보호계통을 구성하고 있는 제어기기를 시험대상으로 이용하였다. 제어기기는 프로세서모듈, 통신모듈, 입출력모듈, 전원모듈, 버스모듈 등 다양한 모듈로 구성되어 있다. 이중에서 프로세서모듈의 응용 프로그램 실행 영역 메모리를 대상으로 하였으나, 할당된 전체영역에 대한 스캐닝 과정을 통해 실제로 접근하는 메모리 위치만을 선별하여 시험 대상으로 지정하였다. 응용 프로그램이 실제로 사용하는 영역은 26,725개의 주소영역이지만, 접근횟수가 20회 이내인 주소는 제외하고 20번 이상 접근한 주소 390개만을 시험대상으로 하였다. 26,725개의 전체주소를 시험하면 정확한 수치를 확보할 수 있으나, 산술적으로 전체 시험횟수를 계산하면 170만 번 이상의 시험이 필요하여 시험대상을 축소하였다.

Table 1. Reserved application program area and real used area

할당된 응용 프로그램 실행영역	1M Double Word	1,048,576
응용 프로그램이 실행을 위해 지나간 실제 주소 영역		90,020
중복된 주소를 제외한 응용 프로그램 사용 영역		26,725
시험을 위해 임의의 고장을 주입한 영역		390
고장을 주입한 횟수	390 × 32 × 2	24,960

4.3 시험결과

시험 및 분석을 통해 획득한 고장 민감도의 세부항목은 다음과 같다.

- **가중치1 = Location_Weight**

시험을 통해 계산한 표본 데이터는 다음과 같다.

- ✓ 고장주입 메모리 주소: 0x00C00E64 ~ 0x00C01139 (사용하는 58개 주소 대상)
- ✓ 고장위치: 0~31 번째 bit (각각의 주소는 32bit 공간으로 할당)
- ✓ 고장유형: Stuck-0

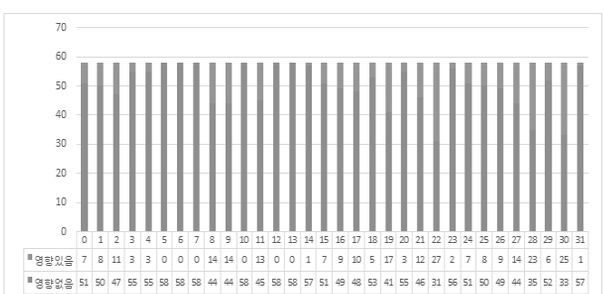


Fig. 6. Error effect statistics through 0 ~ 31 bit position

Fig. 6은 메모리 0번 비트부터 31번 비트의 위치에 고장을 주입하였을 경우 소프트웨어 프로그램에 영향을 주는 빈도수를 나타낸다. 예를 들면, 표본 데이터의 메모리 주소에 0번째 비트에 stuck-0 고장을 유발하였을 경우 7개의 주소는 소프트웨어에 영향을 주었고, 51개의 주소는 영향을 주지 않았다.

고장 영향 통계를 활용해 메모리의 비트위치에 따라 부여할 수 있는 정규화된 가중치 값은 Table 2와 같고 Fig. 7은 비트 위치에 따른 에러의 개수와 개수에 따른 가중치의 비율을 나타낸다. 정규화된 가중치 값은 3.2절에서 설명한 가중치1을 계산한 결과이다.

Table 2. Normalized weight according to bit position

Bit position	0	1	2	3	4	5	6	7
	8	9	10	11	12	13	14	15
	16	17	18	19	20	21	22	23
	24	25	26	27	28	29	30	31
가중치 1 (Normalize)	0.937	1.071	1.473	0.402	0.402	0.000	0.000	0.000
	1.875	1.875	0.000	1.741	0.000	0.000	0.134	0.937
	1.205	1.339	0.670	2.276	0.402	1.607	3.616	0.268
	0.937	1.071	1.205	1.875	3.080	0.803	3.348	0.134

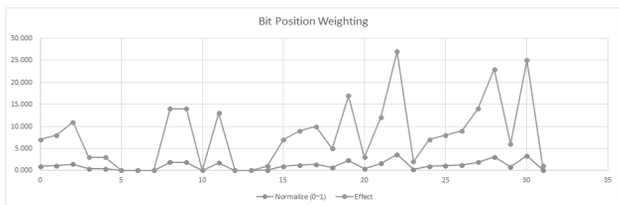


Fig. 7. Weight ratio according to bit position

- **가중치2 = Count Weight**

가중치 2는 응용 프로그램의 1 cycle 동안 접근한 소프트웨어 영역을 카테고리별로 분류하고, 각각의 카테고리 영역에 부여할 가중치 값을 계산하여 정의하였다. Fig. 8은 소프트웨어가 1 cycle을 수행할 때 접근하는 주소와 각 주소의 접근 횟수를 나타내며 Table 3은 접근횟수에 따른 가중치 값을 나타낸다.

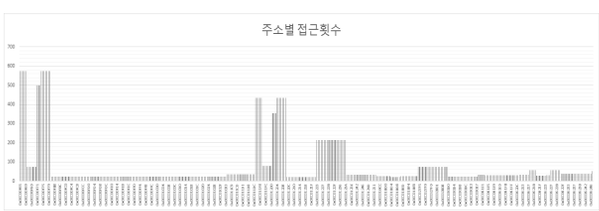


Fig. 8. Access count of each address

Table 3. Weighting value of categories

카테고리	A	B	C	D
접근횟수 구분	200~600	50~200	30~50	20~30
카테고리 소속 주소개수	51	48	102	189
카테고리별 총접근 횟수	19134	3340	3412	4472
카테고리별 가중치	0.6303	0.1100	0.1124	0.1473

카테고리 A-D까지의 오류 유형 비율은 Fig. 9와 같다. stuck-0 고장을 주입하였을 경우과 stuck-1 고장을 주입하였을 경우에 어떠한 오류가 발생하는 지 나타낸 실험결과이다. 에러를 판단하는 기준은 주입된 고장으로 인해서 에러임을 감지하지 못하고, 잘못된 출력 값을 생산하거나, 프로그램이 포인터를 잃어버려 Garbage 상태가 된 경우를 에러

라고 정의하였다. 오류의 유형은 다음과 같다

- ✓ No effect (주입된 고장이 SW의 정상적인 실행에 어떠한 영향도 없음)
- ✓ Infinite Loop (주입된 고장이 SW의 정상적인 실행을 방해하였으나 무한루프에 들어감으로 인해 SW의 자가진단 기능을 통해 감지되었으므로 후소조치가 가능함)
- ✓ Error but diagnosis (주입된 고장이 SW의 정상적인 실행을 방해하였으나 SW의 자가진단 기능을 통해 감지되었으므로 후속조치가 가능함)
- ✓ Warning but diagnosis (고장모사로 인해 SW의 정상적인 실행을 방해하였으나 SW의 자가진단 기능을 통해 감지되었으므로 후속조치가 가능함)
- ✓ Wrong Output (고장모사로 인해 SW 의 출력이 다른 값을 내었으며 SW의 자가진단 기능을 통해 감지되지 못한 상황임)
- ✓ Application Delete (고장모사로 인해 응용프로그램이 삭제되는 결과)

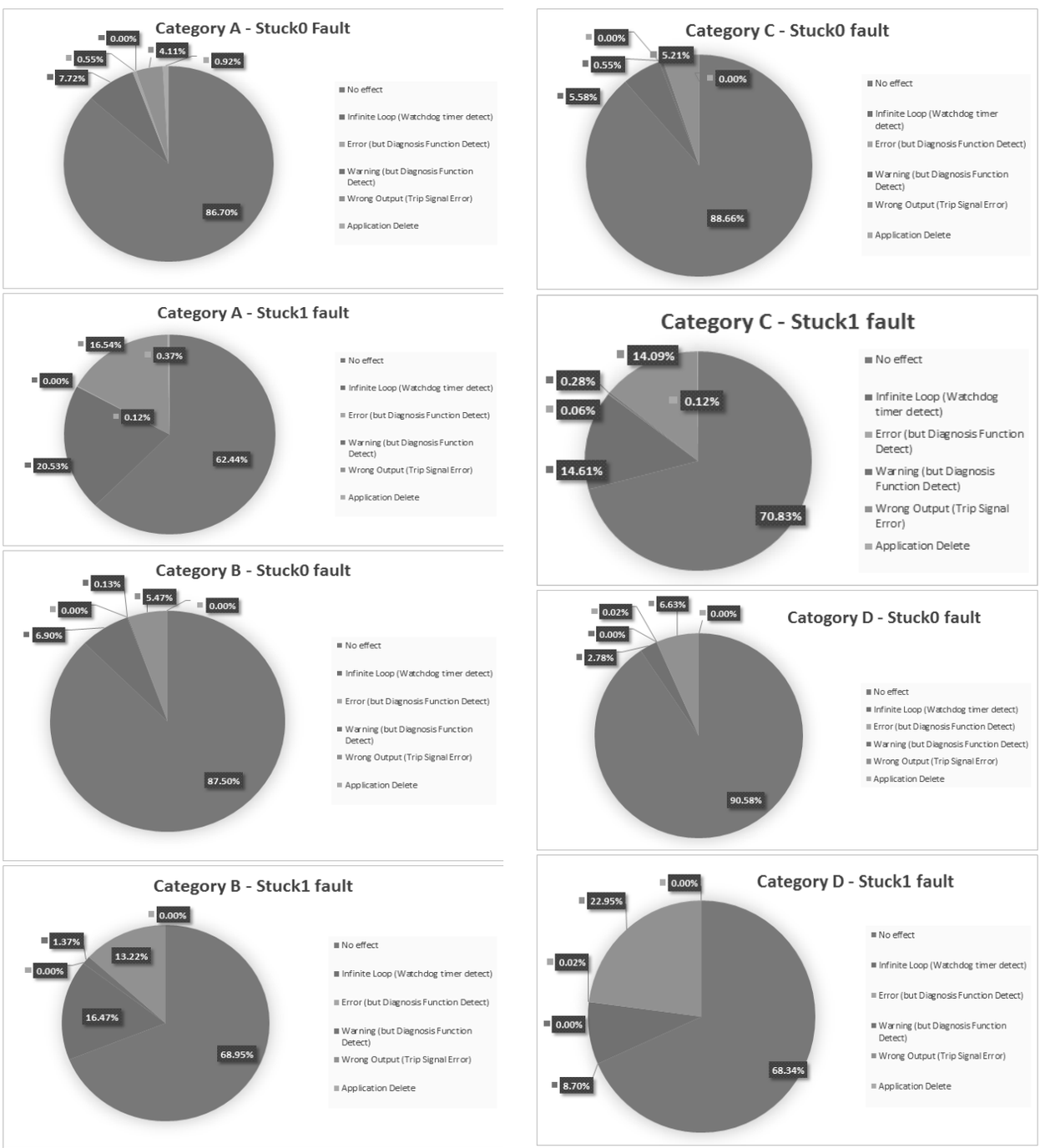


Fig. 9. Error rate of categories

Stuck0 고장과 Stuck1 고장으로 인한 오류 유형은 Fig. 10에서 나타내고 가중치1과 가중치2를 반영한 그래프는 Fig. 11에서 나타낸다. Fig. 12는 오류의 유형에 가중치를 합산한 결과를 보여준다.

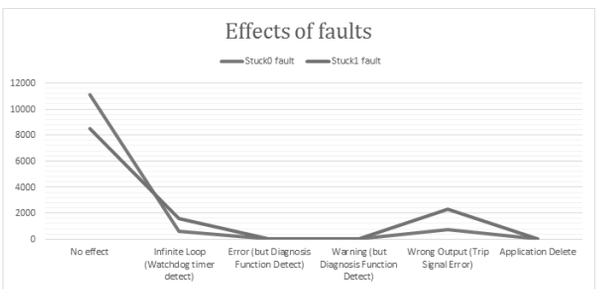


Fig. 10. Error type of Stuck0 fault and Stuck1 fault

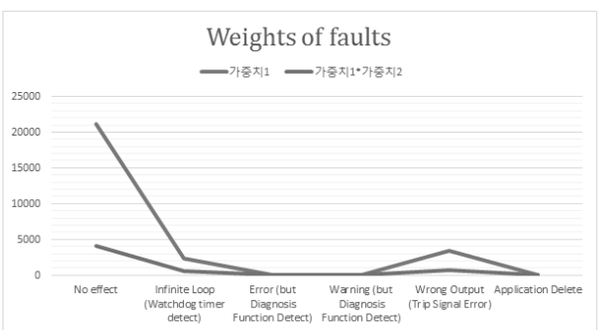


Fig. 11. Adaptation result of weight1 and weight2

	Weighting Sum
No effect	4095.610
Infinite Loop (Watchdog timer detect)	538.890
Error (but Diagnosis Function Detect)	13.990
Warning (but Diagnosis Function Detect)	7.168
Wrong Output (Trip Signal Error)	648.581
Application Delete	18.201
Detection Coverage	0.87472

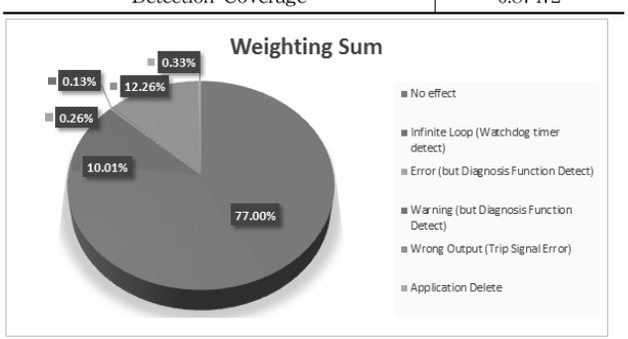


Fig. 12. Fault detection coverage using weighting values

제안된 방법을 통한 신뢰도를 계산하면 다음과 같다. 현재 소프트웨어에 있는 고장을 계산하기 위한 방법이므로 시간 T는 1로 고정하고 계산한다.

- ✓ C = 0.87472 : 고장감지 커버리지
- ✓ T = 1 : 고장누적분포함수의 매개변수
- ✓ $\lambda(t)$ = 1 - C = 0.12528 : 소프트웨어 고장율
- ✓ $F(t) = \int_0^t \lambda e^{-\lambda t} dt = 1 - e^{-\lambda t}$ = 0.11775 : 고장누적분포
- ✓ $R(t) = 1 - F(t)$ = 0.88225 = 88.225% : 소프트웨어 신뢰도

시험의 결과에 의해 원전 제어기기의 소프트웨어는 88.225%의 신뢰도를 가지고 있다고 추측할 수 있게 된다.

5. 결 론

원전의 계통을 구성하고 있는 제어기기가 점차 디지털로 변화하고 있고, 새롭게 개발되는 제어기기도 디지털 기술을 이용하고 있다. 이러한 디지털 기술의 핵심은 소프트웨어이다. 소프트웨어의 안전을 보장하기 위해서 규제기관 및 표준기관에서는 소프트웨어 요건을 정의하고, 개발 및 검증 공정에 관한 지침 및 요건을 배포하고 있다. 지금까지는 이러한 요건을 만족하기 위해서 필요한 모든 개발활동들, 즉 개발공정, 검증공정, 품질보증, 형상관리, 안전성 분석 등 소프트웨어의 안전을 보장하기 위한 다양한 활동을 수행하여 왔다. 이러한 방법으로 인해 소프트웨어의 안전성이 보장되었다고 주장하였지만 이러한 활동을 수행하기 위해서는 많은 시간과 노력, 그리고 비용이 많이 드는 단점들이 존재한다. 또한 정량적인 수치 데이터가 존재하지 않아 객관적인 평가를 수행하는 데에 어려움이 존재하기도 한다. 이러한 단점을 극복하기 위해서 소프트웨어 신뢰도 평가 연구를 진행하여 Software Reliability Growth Model, Baysian Belief Model 등의 적용 타당성을 분석하였다. 하지만 이러한 방법들도 개발 및 검증공정에서 생산한 결과물들을 이용해야 하는데 필요한 시험데이터를 확보하기가 쉽지 않고, 공정상 발생한 데이터를 사용해야 하기 때문에 정량적인 수치를 찾아내기에는 제약들이 존재한다.

따라서 본 연구에서는 소프트웨어의 개발을 완료하고, 실제 사용하는 응용프로그램의 특성을 분석한 후, 해당 소프트웨어가 사용하는 메모리 전 영역에 임의의 고장을 주입하여 소프트웨어의 반응을 살펴보고 고장감지 커버리지를 찾아내었다. 소프트웨어가 사용하는 메모리에 주입되는 고장을 소프트웨어 고장이라고 가정하고, 이러한 상황에 진단 소프트웨어가 어느 정도 고장 감내 능력이 존재하는 지 확인하여 신뢰도를 계산하기 위한 고장율 값으로 사용하였다.

본 연구는 주입되는 고장이 서로 동일한 민감도를 가질 수 없다는 동기를 가지고 고장에 대한 가중치를 부여하여 소프트웨어에 미치는 영향을 파악하고, 서로 가중치가 다른 고장을 감지해 내는 능력을 측정하여 고장율 계산에 사용하였다. 이러한 고장율을 확보하면 시스템 신뢰도 평가를 위한 계산식을 활용하여 소프트웨어의 신뢰도를 유추할 수 있게 된다. 이를 통해 소프트웨어의 진단 기능이 감지하지 못

하는 오류형태를 파악할 수 있고, 설계 또는 구현물의 개선을 통해 소프트웨어의 신뢰도를 향상시킬 수 있을 것이다. 설계개선을 통한 고장감지 커버리지 비율 향상은 제어기기 자체의 신뢰도 확률을 높일 수 있어 다양성 보호계통의 전체 신뢰도도 향상시킬 수 있을 것이다.

References

[1] BTP-7-14, Guidance on software reviews for digital computer-based instrumentation and control system. NUREG-0800, Standard Review Plan: branch technical position 7-14, Revision 5, Nuclear Regulatory Commission.

[2] The Institute of Electrical and Electronics Engineers, Inc., "Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations," IEEE 7-4.3.2.

[3] K. C. Kwon and M. S. Lee, "Technical Review on the Localized Digital Instrumentation and Control Systems," *Nuclear Engineering and Technology*, Vol.41, No.4, pp.447-454, 2009.

[4] Gaurav Aggarwal and V. K Gupta, "Software Reliability Growth Model," *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol.4, pp. 475-479, 2014.

[5] H. S. Eom, G. Y. Park, H. G. Kang, and S. C. Jang, "Reliability assessment of a safety - critical software by using generalized Bayesian nets," *6th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technology*, Knoxville, Tennessee 2009.

[6] Mohd. Anjum, Md. Asraful Haque, and Nesar Ahmad, "Analysis and Ranking of Software Reliability Models Based on Weighted Criteria Value," *Information Technology and Computer Science*, No.2, pp.1-14, 2013.

[7] B. A. Gran and A. Helminen, "The BBN methodology: progress report and future work. OECD Halden Reactor Project," HWR-693, 2002.

[8] "Development of Nuclear Risk Management Technology," Research Report, KAERI/RR-2794/2006.

[9] Yangyang Yu, Barry W. Johnson, "Fault Injection Techniques: A perspective on the state of Research," Fault injection techniques and Tools for Embedded System Reliability Evaluation, 7-39, 2003.

[10] H. G. Kang, "An Overview of Risk quantification Issues of Digitalized Nuclear Power Plants Using Static Fault Trees," *Nuclear Engineering and Technology*, Vol.41, pp.849-858, 2009.

[11] J. Duraes and H. Madeira, "Emulation of software faults, a field data study and a practical approach," *IEEE Trans. Softw. Eng.*. Vol.32, No.11, pp.849-867, 2006.

[12] M. C. Hsueh, T. K. Tsai, and R. KIyer, "Fault Injection Techniques and Tools," *IEEE Computer*, Vol.30, No.4, pp.75-82, April, 1997.

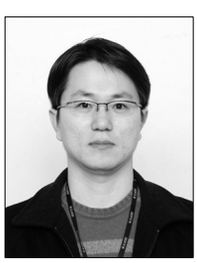
[13] Jean arlat et al., "Fault Injection for Dependability Validation: A Methodology and Some Applications," *IEEE Trans. On Soft. Eng.*, Vol.16, No.2, pp.166-182, Feb., 1990.

[14] PATENT, "Fault mode apparatus and method using software," 10-1222349, The Korean Intellectual Property Office, 2013.

[15] Y. Yu, "A perspective on the state of Research on Fault injection techniques," Research Report, University of Virginia, May, 2001.

[16] H. Madeira, D. Costa, and M. Vieira, "On the emulation of software faults by software faults by software fault injection," *Proceedings of International Conference on Dependable Systems and Networks*, pp.417-426, 2000.

[17] S. Richter and J. Wittig, "Verification and Validation Process for Safety I&C Systems," *Nuclear Plant Journal*, pp.36-40, May-June, 2003.



이 영 준

e-mail : yjlee426@kaeri.re.kr
2001년 충남대학교 컴퓨터과학과(학사)
2003년 충남대학교 컴퓨터과학과(석사)
2003～ 현 재 한국원자력연구원 선임연구원
관심분야 : 소프트웨어 시험, 확인 및 검증, 사이버 보안

이 장 수

e-mail : jslee@kaeri.re.kr
1983년 경북대학교 전자공학과(학사)
1986년 한국과학기술원 전자전산학과(석사)
2002년 한국과학기술원 전자전산학과(박사)
1986년～현 재 한국원자력연구원 책임연구원
관심분야 : 소프트웨어 공학, 안전성 분석, SW 확인 및 검증

김 영 국

e-mail : ykim@cnu.ac.kr
1985년 서울대학교 계산통계학과(학사)
1987년 서울대학교 계산통계학과(석사)
1995년 버지니아대 컴퓨터과학과(박사)
1996년～현 재 충남대학교 컴퓨터공학과 교수
관심분야 : 실시간 시스템, 데이터베이스, 스마트정보시스템