2001 춘계학술발표회 논문집
한국원자력학회

Digital Plant Protection System내 내장형 소프트웨어의 테스팅 방안

Testing Methodology of Embedded Software in Digital Plant Protection System

성아영1), 최병주1)
이나영2), 황일순2)

1) 이화여자대학교
서울특별시 서대문구 대현동 11-1

2) 서울대학교
서울특별시 관악구 신림동 산 56-1

요약

원전보호계통(RPS; Reactor Protection System)은 사고 시 치명적 피해를 입을 수 있다는 점에서 안전에 대한 중요도가 가장 높은 Safety 1E class로 분류되며, 이러한 보호계통을 디지털라이즈 하는데 있어서 높은 신뢰도에 대한 보장이 필요하다. 따라서 본 논문에서는 DPPS(Digital Plant Protection System) 내에서 작동하는 내장형 소프트웨어에 높은 신뢰성을 보장할 수 있으면서 효율적이고 체계적인 테스팅 방법론을 제시하고자 한다. DPPS에서 작동하는 내장형 소프트웨어를 효율적으로 테스트 하기 위한 방법은 크게 두 가지로 나누어진다. 첫번째 단계는 절차중심의 프로그램에서 객체 지향적인 클래스를 추출하는 재공학의 단계이다. 두 번째 단계는 이러한 클래스들을 이용하여 레벨별 테스팅을 수행하기 위한 테스트 아이템을 추출하고, 추출된 테스트 아이템을 이용하여 테스트 케이스를 선정하는 단계이다. 이렇게 각 레벨별로 선정된 테스트 케이스를 이용하여 단위 테스팅, 통합 테스팅, 시스템 테스팅 이렇게 3단계의 레벨별 테스팅을 수행함으로써 테스팅에 드는 시간과 비용을 줄 일 수 있다.

Abstract

It is necessary to assure the reliability of software in order to digitalize RPS(Reactor Protection System). Since RPS causes fatal damage on accidental cases, it is classified as Safety 1E class. Therefore we propose the effective testing methodology to assure the reliability of embedded software in the DPPS(Digital Plant Protection System). To test the embedded software effectively in DPPS, our methodology consists of two steps. The first is the Re-engineering step that extracts classes from structural source program, and the second

is the Level of Testing step which is composed of Unit Testing, Integration Testing and System Testing. On each testing step we test the embedded software with selected test cases after the test item identification step. If we use this testing methodology, we can test the embedded software effectively by reducing the cost and the time.

1. 서론

사고의 피해가 치명적인 시스템을 Safety-Critical System[1]이라고 하며, 이 시스템에서 사용되는 소프트웨어는 안전성 검증을 통해 높은 신뢰도를 보장 받아야 한다. Safety-Critical System 중 하나로 원자력 발전소의 RPS(Reactor Protection System)의 경우, 원전 안전계통 중에서도 Safety 1E class로 안전에 대한 치명도가 가장 높다. [2] 따라서 이러한 시스템에서 사용되는 소프트웨어의 신뢰도 문제가 중요하게 부각된다. 이러한 신뢰도를 정량화하기 위해서 소프트웨어에 대한 테스트가 필요하며 고신뢰도의 보장에 필요한 많은 수의 테스트를 수행하기 위해서는 많은 시간과 노력이 요구된다.

특히 이러한 소프트웨어는 특정 기능을 수행하기 위해 하드웨어와 조합되어 하나의 시스템으로 나타내지며, 이런 시스템에서 사용되는 소프트웨어를 내장형 소프트웨어라고 한다. [3] 내장형 소프트웨어의 경우 다른 소프트웨어에 비해 수정하는 것이 용이하지 않으며, Real-time-ness적인 특성과 Robust-ness적인 특성이 있는데, 특히 안전도가 높은 원전 보호계통 시스템 내에 들어갈 내장형 소프트웨어이기 때문에, Robust-ness가 높게 요구된다.[4,5] Robust-ness가 높게 나타나기 위해서는 높은 Fault-tolerance가 요구되며 이를 위해서는 철저한 테스팅을 통해 안전성과 신뢰성을 보장 받아야 한다.

본 논문에서는 원자력 발전소에서 사용한 기존의 아날로그 방식의 RPS를 기기 낙후에 따른 부품 조달의 어려움과 한번 구현된 시스템에 대한 수정의 어려움 때문에 이런 안전에 대한 중요도가 가장 높은 원전보호계통 시스템을 디지털화 하려는데 있어, 이 시스템에서 사용되는 소프트웨어의 안전성 검증을 위해 체계적이고 효율적인 테스트 방법론을 제안 하고자 한다.

본 논문에서 제안한 테스트 방법은 기존의 수 많은 테스트 케이스를 일일이 넣어서 테스트 하는 방법에 비해 적은 수의 테스트 케이스를 생성하고, 불필요한 테스트를 하지 않으며, 특히 오류를 잘 발견 할 수 있는 테스트 케이스를 생성 할 수 있어 테스트 하는데 있어 시간과 노력이 절감된다. 특히 기존의 테스트 방법에서는 어느 정도 테스트 커버리지가 만족된 후, 그 커버리지를 높이기 위해서는 많은 테스트 케이스를 넣어서 직접 값을 넣고 테스트를 해서 확인을 해 봐야 했지만, 도식화된 다이어그램을 보면서 소프트웨어에 관련한 클래스에서 하드웨에 관련한 클래스로 가는 메시지들 중점적으로 고려하여 테스트 케이스를 선정하기 때문에 특히 내장형 소프트웨어의 경우 오류를 잘 찾을 수 있는 좋은 테스트 케이스를 생성할 수 있기 때문에 보다 더 효율적인 테스팅을 수행한다.

본 논문의 구성은 다음과 같다. 2장에서는 원전 보호계통 시스템인 DPPS에 대한 배경과 설명을, 3장에서는 그러한 시스템을 효율적으로 테스트하기 위한 테스트 방법론에 대해서 기술하며 4장에서는 결론을, 그리고 5장에서는 향후연구과제를 제시한다.

2. DPPS(Digital Plant Protection System)

원자력 발전소의 RPS은 원전에 문제가 발생했을 때, 원전을 안전하게 정지시키기 위한 장치이다. 기존의 원전에는 릴레이 로직에 의해 구현된 아날로그 방식을 사용하였으나, 시스템의 낙후에 따른 부품 조달의 어려움과 한 번 구현된 시스템은 수정이 어렵기 때문에 신호의 drift 등의 문제에 대해 디지털 방식으로의 개발이 대두 되었다, 그렇지만, 원자력 분야에 신기술을 적용하기 위해서는 이전에 관한 까다로운 규제를 만족시켜야 한다. 특히 RPS은 원전 안전 계통 중에서도 Safety 1E class로 안전에 대한 치명도가 가장 높다. 따라서 소프트웨어의 신뢰도 문제가 중요하게 부각된다.

현재 하드웨어 부분에 대한 신뢰도 분석 방법론은 수립되어 있지만, 소프트웨어에 대한 방법론은 아직 확립되지 못하였다. 규제에서도 소프트웨어의 개발 방법론으로 정형기법을 이용한 V&V(Verification and Validation)의 수행이 권고 되고 있으며, 소프트웨어의 공통 모드 고장, 종속적인 고장, 검출되지 않은 고장 등에 대한 신뢰도 분석에 포함시킬 것을 요구하고 있지만, 디지털 안전계통의 적용 경험이 거의 없어 방법론적인 면에 대한 언급은 없다. 따라서 입증된 기술 중의 원전 DPPS에 적합한 방법론을 개발하여 테스트를 수행하고 이 결과를 신뢰도 분석에 활용 할 수 있도록 하고자 본 연구를 수행한다.

DPPS는 4개의 분리된 채널 장치로 구성되어져 있다. 각 채널 장치는 CPC(Core Protection Calculation) cabinet[2], PPC(Plant Protection Calculator) cabinet[2]과 인터페이스로 구성되어져 있다. PPC는 다시 Bistable processor[2], Coincidence processor[2], Interface and Test processor[2]로 구성되어져 있다.

보호 계통 케비넷의 Bistable processor는 10개의 아날로그 신호와 2개의 디지털 신호를 입력으로 받아 기존에 설정되어 있는 설정치와 비교하여, 입력 받은 값들이 설정치보다 높아지거나 낮아지게 되면 정지 신호를 발생시킨다. 이러한 정지신호는 2/4 Coincidence processor로 입력으로 들어가게 되며, 이때 Coincidence processor는 독립적인 4개 채널의 Bistable processor로부터 나오는 출력을 모두 비교하여, 두 개 이상의 채널에서 정지신호가 발생되면, Coincidence 프로세스는 원자로 정지와 공학적 안전설비계열을 동작하기 위한 트립 개시신호를 트립 개시 프로세서에 제공한다. 원자로 정지는 제어봉 구동장치의 코일로 가는 전원을 차단함으로써 제어봉을 노심내로 자중 낙하 시켜 달성되고, 공학적 안전설비는 신호 및 계열별로 할당된 펌프나 벨브의 작동을 위한 신호를 발생시킴으로써 동작된다.[2] 이 과정을 DFD(Data Flow Diagram)으로 표현하면 그림 1과 같다.

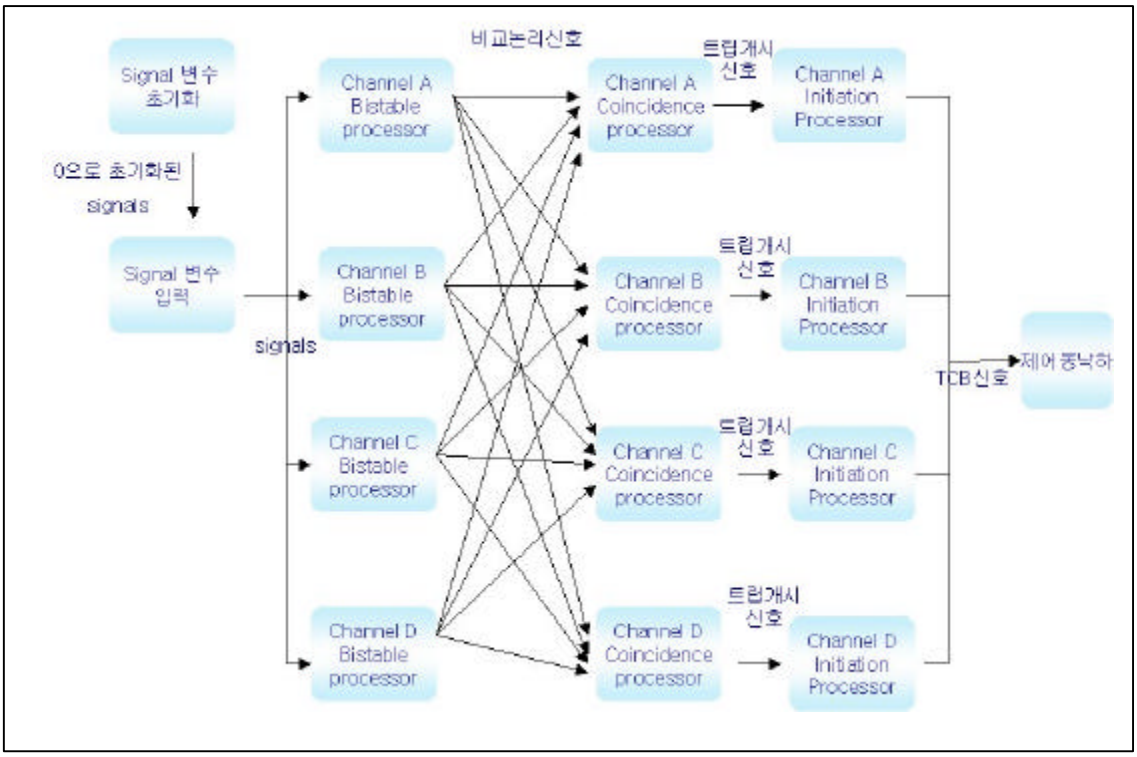

그림 1. DFD of DPPS

3. 테스팅 방안

DPPS에서 사용되는 소프트웨어는 절차중심의 언어인 C 언어로 구성되어져 있으며, 이러한 소프트웨어는 해당 시스템 안에 내장되어 있는 내장형 소프트웨어이다. 내장형 소프트에어는 시스템에 내장되는 컴퓨터 상에서 동작하는 소프트웨어로, 시스템의 하드웨어로부터 입력을 받아 적절한 제어 신호 발생 기능을 수행하는 소프트웨어이다.[3]

절차중심의 내장형 소프트웨어를 체계적이고 효율적으로 테스팅 하기 위해서 객체 지향적인 테스팅 방법론을 제안하고자 한다. 여기에서 말하는 효율적인 테스팅이란 좋은 테스트 케이스를 선정하여 테스팅 비용을 줄일 수 있는 테스팅을 의미한다. 좋은 테스트 케이스를 사용하여 테스팅을 수행하면 오류를 잘 발견 할 수 있으며, 선정된 테스트 케이스의 테스트 커버리지가 높게 나타나기 때문에 시간과 비용적인 측면에서 효과적이다.

본 연구에서 제안하는 테스팅 방법은 그림 2와 같이 크게 2가지의 단계로 나눈다. 첫번째 단계는 재공학 단계로, 절차지향 중심의 C 프로그램에서 객체 지향적인 개념의 클래스를 추출하는 단계이다.[6,7] 두 번째 단계는 레벨별로 테스팅을 수행하는 단계로, 추출된 클래스를 이용하여 필요한 다이어그램으로 도식화 한 후 이를 토대로 레벨별로 테스팅을 수행하는 것이다.

본 논문에서 제안하는 테스팅 방안은 기존의 테스팅 방법, 즉 수 많은 테스트 케이스에 대해 일일이 프로그램을 돌려 테스트 하는 방법에 비해서 효율적이다. 왜냐하면 프로그램으로부터 추출한 클래스를 도식화하여 내장형 소프트웨어의 특징인 하드웨어 부분과 소프트웨어 부분에 대한 관련성에 대한 이해를 증가시키고, 체계적인 레벨별 테스팅을 통해서 테스트에 드는

시간과 비용이 절감되기 때문이다.

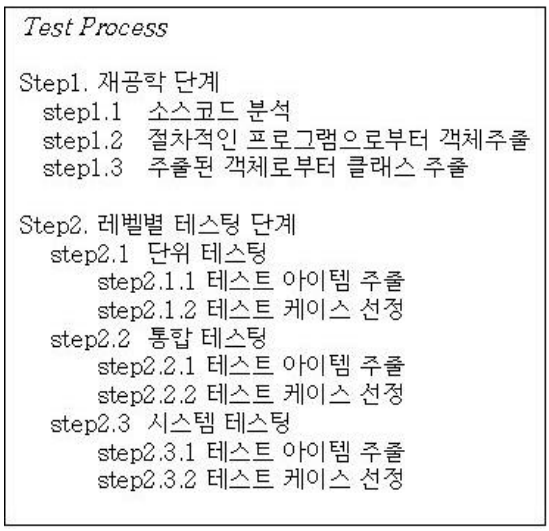

그림 2. 테스트 프로세스

3.1 재공학단계

이 단계에서는 절차중심으로 짜여진 프로그램으로부터 클래스를 추출하는 단계이다.

절차중심의 프로그램으로부터 재공학 단계를 거쳐 클래스를 얻는 이유는 다음과 같다. 첫번째, 추출된 클래스를 하드웨어 관련한 부분과 소프트웨어에 관련한 부분으로 나누고, 그것을 연관 있는 것끼리 그룹화 함으로서, 그들의 상관관계를 이해하는데 도움을 준다. 이러한 상관관계는 Step2 레벨별 테스팅 단계 중 통합 테스팅 단계에서 내장형 소프트웨어에서 중점을 두고 테스트 해야 하는 부분이다. 두 번째, UML(Unified Modeling Language)을 이용하기 위함이다. UML diagram으로부터 테스트 케이스를 얻기 위해서는 절차 중심의 프로그램으로부터 객체와 클래스를 추출한다. 추출된 클래스를 이용하여 UML 다이어그램을 작성하고, 이러한 다이어그램은 레벨별 테스팅 단계에서 테스팅을 수행할 때, 테스트 케이스를 선정하기 위한 입력이 된다.

원전 보호계통 내의 절차중심의 내장형 소프트웨어로부터 클래스를 얻기 위한 개략적인 과정은 그림 3과 같다. 첫번째 과정은 절차 중심의 프로그램으로부터 객체를 추출한다. 이때 절차중심의 C 소스 코드로부터 소프트웨어에 관련한 부분의 객체(객체I)와 하드웨어에 관련한 부분의 객체(객체II)를 추출한다. 두 번째 과정은 추출된 객체로부터 클래스를 추출한다. 이렇게 하여 추출된 소프트웨어에 관련한 클래스(클래스I)와 하드웨어에 관련한 클래스(클래스II)를 추출한다.

그림 3에서와 같이 객 그림 3에서와 같이 객체I, 클래스I과 객체II, 클래스II처럼 소프트웨어에 관련한 부분과 하드웨어에 관련한 부분 이렇게 두 개의 부분으로 나누는 이유는, 이 소프트웨어가 일반적인 소프트웨어하고는 다른 내장형 소프트웨어이기 때문에 소프트웨어에 관련된 부분과 하드웨어에 관련된 부분으로 나누어 줘야 한다. 이렇게 두 부분으로 나누어진 소프트웨어에 관련된 부분과 하드웨어에 관련된 부분 각각에 대한 단위 테스트를 수행하고, 이들을 통합했을 때 야기되는 문제점들과, 소프트웨어에 관련된 부분과 하드웨어에 관련된 부분 사이의

인터렉션을 테스트 하는 통합테스트를 수행할 수 있기 때문에 두 부분으로 나눈다.

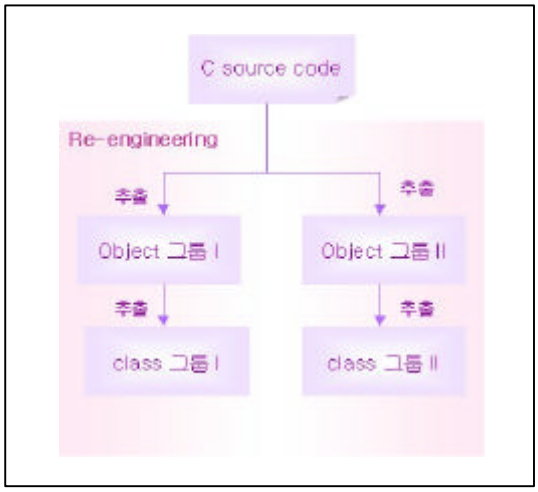

그림3. 재공학 단계

3.1.1 객체 추출 단계

절차 중심의 C 소스코드를 분석하여 전체 시스템의 논리적인 흐름을 그림 1과 같은 DFD(Data Flow Diagram)로 나타낸다. 절자 중심의 C 코드로부터 객체를 얻기 위한 첫번째 과정으로는 아래의 그림 4처럼 소스코드로부터 각 모듈의 기능별로 함수를 분리하여 간략화 하는 것이다.

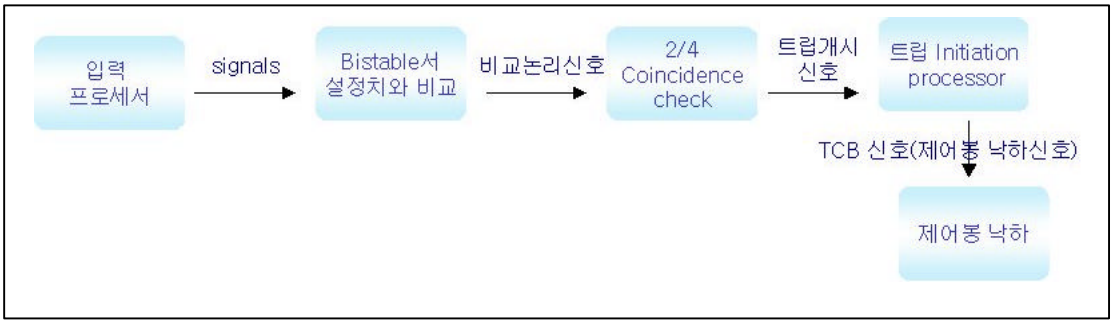

그림 4. 모듈의 기능별로 간략하게 표현된 DPPS

두 번째 과정은 그림4에서 각 함수들을 하드웨어에 연관된 부분과 그렇지 않은 부분으로 그룹화하여 나타내는 것으로 이는 표 1과 같이 나타내어진다.

그룹 I (S/W 관련 부분)의 함수	그룹 II (H/W 관련 부분)의 함수
Initialize(analog,digital); Initialize_measure(analog,digital); Input(i);	Bistable(analog,digital); Coincidence(analog,digital); Initiation(analog,digital);

표 1. 두개의 그룹으로 분리된 상태

표1에서 두 그룹으로 나누어진 함수들을 토대로 Initialize 객체, Initialize_measure 객체,

input 객체는 소프트웨어에 부분에 관련한 객체 그룹(객체 그룹I)이고, Bistable 객체, coincidence 객체, Initiation 객체는 하드웨어에 부분에 관련한 객체 그룹(객체 그룹II)으로 객체를 추출한다.

3.1.2 클래스 추출 단계

두 그룹으로 추출된 객체들을 수정이나 보완하여 클래스를 추출한다. 추출한 클래스를 가지고 그림 4에 대한 UML의 sequence diagram으로 표시하면 아래의 그림 5와 같다.

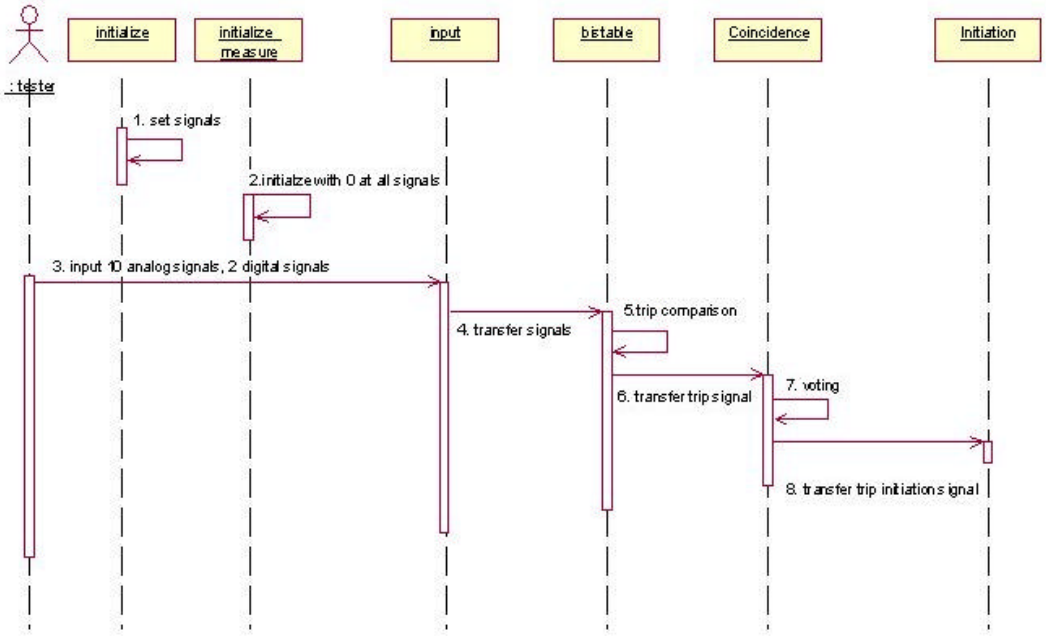

그림 5. Sequence diagram of DPPS

3.2 레벨별 테스팅 단계

레벨별 테스팅 단계에서는 레벨별로 테스팅 과정을 나눔으로써, 일일이 많은 테스트 케이스를 넣고 단순히 프로그램을 돌려 테스트를 수행하는 방법에 비해 테스트 과정을 체계화 하였으며, 또 불필요한 테스트를 다시 하지 않기 때문에 테스트의 비용과 시간이 감소된다. 또 도식화된 다이어그램들을 입력으로 받아서 테스트 케이스를 선정 하기 때문에, 단순히 프로그램을 돌려서 값을 넣어서 테스트하는 방법에 비해선 보다 오류를 잘 발견할 수 있는 테스트 케이스를 생성하기 때문에 테스트에 드는 노력과 시간이 절약된다.

레벨별 테스팅의 단계에서는 각 레벨별로 테스트 설계, 테스트 수행, 테스트 검토를 하게 되고 그 과정은 표 2와 같다.

활동	작업	절차
단위 테스트	단위 테스트 설계	테스트 항목 추출
		테스트 케이스 선정
	단위 테스트 수행	요구 환경 구성
		테스트 수행
		오류 수정 및 재테스트
	단위 테스트 검토	테스트 수행 결과 분석
		테스트 결과 승인
통합 테스트	통합 테스트 설계	테스트 항목 추출
		테스트 케이스 선정
	통합 테스트 수행	요구 환경 구성
		테스트 수행
		오류 수정 및 재테스트
	통합 테스트 검토	테스트 항목 추출
		테스트 케이스 선정
시스템 테스트	시스템 테스트 설계	테스트 항목 추출
		테스트 케이스 선정
	시스템 테스트 수행	요구 환경 구성
		테스트 수행
		오류 수정 및 재테스트
	시스템 테스트 검도	테스트 수행 결과 분석
		테스트 결과 승인

표 2. 각 레벨별 테스트 프로세스

레벨별 테스팅 단계에서는 그림 6의 점선처럼 단위 테스팅, 통합 테스팅, 시스템 테스팅처럼 3단계의 레벨로 나누어서 체계적으로 테스팅을 수행하는 단계이다. 이 단계의 목적은 체계적인 테스팅을 수행하기 위함이다. 이 단계에서는 Step1에서 추출한 클래스들을 이용하여 UML 다이어그램으로 도식화 한 후, 이 다이어그램들을 입력으로 받아 각 레벨별로 좋은 테스트 케이스 선정을 위해 먼저 테스트 아이템을 추출하고, 추출된 아이템을 기준으로 테스트 케이스를 선정한다. 이렇게 선정된 테스트 케이스를 가지고 레벨별 테스팅 단계에서 수행하는 내용은 다음과 같다.

가. 단위 테스팅

각각의 모듈에 대한 테스팅을 수행하는 단계이다. 그림 2의 Step1의 재공학 단계에서 추출한 소프트웨어 부분과 하드웨어 부분 두개의 그룹 별로 얻어진 클래스들을 각각 테스팅 함으로써 각 그룹 내에 있는 클래스들의 신뢰성을 보장한다. 그림 5의 각 클래스들에 대해서 단위 테스팅을 수행한다.

나. 통합 테스팅

단위 테스팅을 거친 각 모듈들의 결합한 조합에 대해서 테스팅을 수행한다. 이때 모든 가능한 모듈의 조합에 대해 다 테스팅을 수행할 필요는 없다. 각 모듈들을 통합 시켰을 때 야기되는 문제들에 대한 테스팅을 수행하면 된다. 이 때 중점을 두고 테스팅 하는 부분은 소프트웨어 부분과 하드웨어 부분 두 그룹간의 클래스들을 테스팅 하는 것이다. 즉

소프트웨어에 관련한 부분의 클래스들과 하드웨어에 관련한 클래스들을 통합 테스팅 함으로써 둘 사이에서 일어나는 문제와 두 부분사이의 인터렉션을 테스팅 하는 것으로 내장형 시스템에 있어 필수적으로 테스트 해야 하는 부분이다. 이때 통합 테스팅에서 테스트 케이스를 선정 하기 위해 그림 5와 같은 sequence diagram을 보면서 소프트웨어에 관련한 클래스에서 하드웨에 관련한 클래스로 가는 메시지들 중점적으로 고려하여 테스트 케이스를 선정한다. 그렇기 때문에 도식화된 그래프를 통하여 오류를 찾을 수 있는 테스트 케이스를 생성할 수 있어 효율적인 테스팅을 수행한다.

다. 시스템 테스팅

시스템 테스팅 단계에서는 통합 테스팅 단계를 거친 후, 사용자 입력을 기준으로 전체 시스템이 목적에 맞게 제대로 작동하는지에 대한 테스팅을 수행한다.

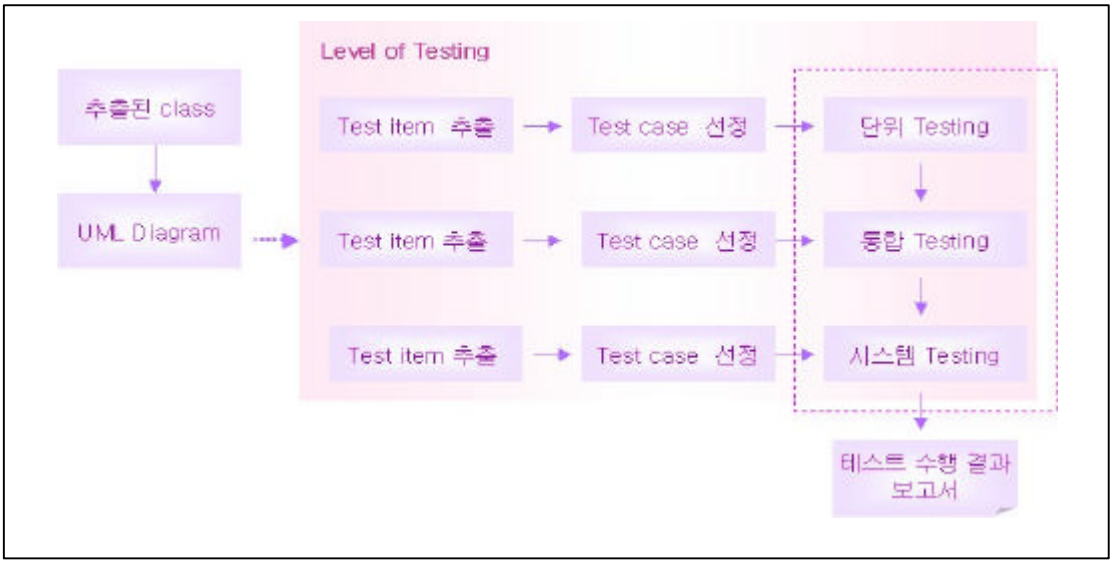

그림 6. 레벨별 테스팅 단계

4. 결론

본 연구에서는 안전에 대한 치명도가 Safety 1E class로 가장 높은 RPS를 디지털화 하려는데 있어, 이 시스템에서 사용되는 내장형 소프트웨어의 안전성과 신뢰도 보장을 위한 체계적인 테스팅 방법론을 제안했다. 이 방법은 기존의 수 많은 테스트 케이스를 일일이 넣어서 테스트 하는 방법에 비해 적은 수의 테스트 케이스를 생성하고, 특히 오류를 잘 발견 할 수 있는 테스트 케이스를 생성 할 수 있어 테스트 하는데 있어 시간과 노력이 절감된다.

이 방법에서 제안하는 방법은 크게 두 가지로 나누어 진다. 먼저 재공학 단계로, 절차중심의 프로그램으로부터 객체 지향적인 개념인 클래스를 추출하기 위하여, 소스코드로부터 소프트웨어 부분과 하드웨어 부분으로 나누어 객체를 추출하고, 이렇게 추출된 객체 그룹을 바탕으로 클래스를 추출한다. 다음 단계는 레벨별 테스팅 단계로, 이 단계에서는 이미 추출한 클래스들을 이용하여 필요한 UML 다이어그램을 작성하고 이것을 이용하여 각 레벨별로 테스트 아이템을 추출하고, 이것을 토대로 레벨별 테스트 케이스를 선정한 후, 레벨별로 테스팅을 수행 한다. 특

히 이때 통합 테스팅 단계에서는 소프트웨어 부분과 하드웨어 부분을 통합 하는데 있어 야기되는 문제점들을 테스팅 하는 단계로, 내장형 소프트웨어의 경우 필수적으로 이루어져야 하는 테스팅 단계이다.

레벨별로 테스팅 과정을 나눔으로써 일일이 많은 테스트 케이스를 넣고 단순히 프로그램을 돌려 테스트를 수행하는 방법에 비해 테스트 과정을 체계화 하였으며, 또 불필요한 테스트를 다시 하지 않기 때문에 테스트 비용과 시간이 감소된다. 또 사용되는 소프트웨어가 일반적인 소프트웨어와 달리 내장형 소프트웨어이기 때문에, 소프트웨어와 하드웨어로 두개의 그룹으로 나누어서 그것을 도식화하여 각 그룹에 대한 레벨별 테스팅을 수행하는데, 이 방법은 단순히 테스트 케이스를 일일이 넣어 프로그램을 돌려가면서 이 테스트 케이스가 오류를 찾을 수 있는지 없는지를 보는 방법에 비해, 도식화된 그래프를 통하여 오류를 찾을 수 있는 테스트 케이스를 생성할 수 있어 효율적인 테스팅을 수행한다.

5. 향후 연구 과제

향후 과제로는 이러한 테스팅 방법론에서 하드웨어와 관련된 부분에 대한 특징과 interaction을 다이어그램으로 나타낼 수 있도록 보완하여, 다른 내장형 시스템에다가도 이 방법론을 적용해 보는 것이며, 이 방법론을 지원 할 수 있는 내장형 소프트웨어 테스팅을 위한 자동화된 도구 구현을 하는 것이다.

6. 참고문헌

[1]Patrick R.H. Place, Kyo C.Kang , “Safety-Critical Software Status Report and Annotated Bibliography”, CMU/SEI-92-TR-5, ESC-TR-93-182

[2] EPRI TR-103331-V1 Research project 3093-01,"Guidelines for the verification and validation of expert systems software and conventional software", Vol. 1, 1995

[3] E. A. Lee, "What's Ahead for Embedded Software?", Computer, September, 2000

[4] Maier, T., "FMEA and FTA To Support Safe Design of Embedded Software in Safety-Critical Systems, " In CSR 12th Annual Workshop on Safety and Reliability of Software Based Systems, Bruges, Belgium, 1995.

[5]Mattias O'Nils and Axel Jantsch, "Communication in Hardware/Software Embedded Systems - A Taxonomy and Problem Formulation", Proceedings of the 15th NORCHIP Conference, November, 1997

[6] Robert S. Arnold, “Software Reengineering”, IEEE Computer Society Press, 1994

[7] J.A Zimmer “Restructuring for style”, Software Practice and Experience vol20(4), April, 1990