2001 춘계학술발표회 논문집
한국원자력학회

SMART MMIS 소프트웨어에 적용을 위한 소프트웨어시험 개념
A Concept of Software Testing for SMART MMIS Software

서용석, 성승환, 박근옥, 허섭, 김동훈
한국원자력연구소

요 약

고품질의 소프트웨어를 생산하기 위해 체계적으로 정립된 소프트웨어시험이 요구된다. 본 논문은 SMART MMIS 소프트웨어에 적용을 위한 소프트웨어시험 개념을 소프트웨어시험 조직, 문서화, 절차, 방법으로 분류하여 설정하였다. 소프트웨어시험 방법으로는 소프트웨어 정적분석과 동적시험으로 구분하여 제시하였으며 동적시험은 white-box 시험과 black-box 시험을 중심으로 기술하였다. 본 논문을 통해 제시된 소프트웨어시험 개념을 SMART MMIS 소프트웨어시험에 적용함으로써 고품질의 소프트웨어를 생산할 수 있을 것으로 판단된다. 추후 본 논문을 통해 제시된 시험방법을 SMART MMIS 시험검증설비 구축 시 적용 및 수행하여 소프트웨어고장 데이터를 취득할 예정이다.

Abstract

In order to achieve high quality of SMART MMIS software, the well-constructed software testing concept shall be required. This paper established software testing concept which is to be applied to SMART MMIS software, in terms of software testing organization, documentation, procedure, and methods. The software testing methods are classified into source code static analysis and dynamic testing. The software dynamic testing methods are discussed with two aspects: white-box and black-box testing. As software testing concept introduced in this paper is applied to the SMART MMIS software, the high quality of the software will be produced. In the future, software failure data will be collected through the construction of SMART MMIS prototyping facility which the software testing concept of this paper is applied to.

1. 서론

전력생산 외에도 담수화 목적을 위해 설계되는 일체형원자로인 SMART(System-integrated Modular Advanced ReacTor)를 제어하고 감시하는 MMIS(Man-Machine Interface System)는 입력신호를 취득하는 계측계통부터 제어 및 보호, 감시, 제어실에 경보 및 정보를 표시하는 모든 계통이 컴퓨터기반의 디지틀시스템으로 설계된다[1]. 고품질의 SMART MMIS 소프트웨어를 보장하기 위한 소프트웨어 개발 개념에서 소프트웨어시험 지침 및 절차의 필요성을 언급하였다[2]. 소프트웨어시험 지침 및 절차를 작성하기 앞서 전반적인 소프트웨어시험 개념이 설정되어 이를 자체적으로 개발된 소프트웨어시험뿐만 아니라 COTS(Commercial Off-the-Shelf) 소프트웨어시험에도 적용할 필요가 있다.

소프트웨어시험에 대한 일반적인 인식은 개발된 소프트웨어가 그 소프트웨어의 상위 요건을 만족하는지 소프트웨어시험을 통해 확인하려는 노력으로 이해된다. 이것은 소프트웨어 개발자 관점에서 소프트웨어시험을 이해한 것이고 독립적인 그룹에서 수행되는 소프트웨어시험에 대한 정의는 위의 내용을 포함할 뿐만 아니라 시험대상 소프트웨어에는 결함 또는 오류가 반드시 존재하며, 이것을 소프트웨어시험을 통해 밝혀내려는 노력임을 정의에 포함하여야 한다. 즉, 독립된 그룹에서 수행하는 소프트웨어시험은 개발된 소프트웨어가 정상적으로 동작하리라는 시각보다 고장을 유발하는 결함 또는 오류가 반드시 존재하리라는 부정적인 시각과 비정상적인 상황이 주어졌을 경우 소프트웨어의 반응이 완전하지 않을 것이라는 가정에 많은 비중을 두고 출발한다. 따라서, 소프트웨어 시험자는 도전적이며, 파괴적인 성향을 갖게 되는데 이는 소프트웨어를 다양한 관점에서 접근하고 시험할 수 있는 긍정적인 요소로 작용한다. IEEE-1012 소프트웨어 표준에 의하면 소프트웨어 검증 및 확인 작업이 효율적으로 수행되기 위해서는 요건, 설계, 코드와 같은 생산물이 추적성(traceability)과 그에 따른 시험성(testability)이 가능한 내용으로 기술되어야 함을 언급하고 있다[3]. 이를 바탕으로, SMART MMIS 소프트웨어시험의 목표를 추적성과 시험성이 가능한 요건, 설계, 소프트웨어의 내용 가운데 잠재되어 있는 결함 및 오류를 소프트웨어시험을 통해 찾아내는 것으로 설정한다.

소프트웨어 시험을 통해 소프트웨어고장 데이터를 취득할 수 있다. 소프트웨어고장 데이터는 그 소프트웨어의 신뢰도를 계산하는데 있어서 필수적인 입력요소로 활용된다. 예를 들어, 소프트웨어 신뢰도 계산을 하는데 있어서 포아송(poisson) 확률분포를 사용하거나 고장밀도함수(failure density function)를 얻고자 하는 경우, 소프트웨어고장 데이터가 절대적으로 필요하며, 이는 시험을 통해 고장 데이터가 취득되어야 가능하다. 이와 같이 소프트웨어시험은 소프트웨어의 품질과 신뢰도를 제고하는데 있어서 중요한 역할을 한다.

본 논문에서 언급하는 소프트웨어시험은 개발그룹으로부터 독립된 시험그룹에서 수행되는 소프트웨어시험을 의미한다. 소프트웨어시험은 소프트웨어 확인 및 검증 활동의 일환으로 수행되며, 확인 및 검증 활동은 독립적으로 수행되어야 소프트웨어를 객관적으로 평가할 수 있다. 일반적으로 시험과정은 단위모듈시험, 기능시험, 계통시험, 통합시험, 시운전시험으로 구분된다. 소프트웨어 개발자가 자신의 프로그램에 대해 단위모듈시험 및 기능시험을 수행하거나 소프트웨어 개발그룹에서 자체적으로 시험지침을 수

립하여 수행할 수 있으나 이는 개발자의 주관적인 관점이 강조되어 수행될 수 있는 결함을 내포하고 있다. 따라서, 시험그룹에 의해 위의 시험과정을 포괄하는 독립시험이 수행될 필요가 있다.

본 논문의 목적은 위의 시험과정을 포괄하는 시험개념을 설정하기 위함이다. 본 논문의 소프트웨어시험 개념은 소프트웨어시험 수행을 위한 조직설정, 소프트웨어시험과 관련된 문서생산, 소프트웨어시험 수행절차수립, 그리고 구체적인 소프트웨어시험방법으로 분류하여 기술하였다. 소프트웨어시험 방법을 소프트웨어 정적분석과 동적시험 방법으로 분류하여 기술하였으며, 소프트웨어 동적시험을 white-box 시험과 black-box 시험으로 구분하여 기술하였다. 부가적인 소프트웨어시험과 소프트웨어시험 시 고려되어야 할 사항을 본 논문에서 언급하였다.

2. 소프트웨어시험 조직

독립적인 소프트웨어시험을 수행하기 위해서 소프트웨어시험 조직구성은 필수적이다. 소프트웨어시험 조직은 소프트웨어 개발조직으로부터 예산 및 관리측면에서 완전히 독립되어야 시험활동의 다양성이 보장될 수 있다. 소프트웨어시험 조직의 최소 구성인원은 시험관리자, 시험분석가, 시험수행원이다. 시험관리자는 시험계획을 설정하며, 소프트웨어 품질보증그룹, 개발그룹과 시험그룹간의 조정역할을 담당한다. 개발그룹과 시험그룹간의 상충된 의견이 발생한 경우 시험관리자는 이를 중재하고 일정을 재조정한다. 시험분석가는 테스트케이스(test case)를 작성하여 시험수행원이 테스트케이스를 수행할 수 있도록 입력으로써 제공한다. 또한, 시험수행원이 수행한 시험결과를 분석하고 자신의 평가와 결론을 내린 시험보고서를 작성한다. 시험수행원은 테스트케이스의 내용을 수행하며 완료된 테스트케이스를 시험분석가에게 제출한다.

3. 소프트웨어시험 문서화

IEEE-829는 소프트웨어시험 문서 작성에 대한 지침을 제공한다[4]. 본 논문은 IEEE-829를 참조하여 SMART MMIS에 적용하기 위한 소프트웨어시험 문서를 다음과 같이 정의하였다: 소프트웨어시험 계획서, 소프트웨어시험 사양서, 소프트웨어시험 보고서이다.

소프트웨어시험을 수행하기 위해 가장 먼저 생산되어야 하는 문서는 소프트웨어시험 계획서이다. 소프트웨어시험 계획서는 소프트웨어시험에 대한 전반적인 계획을 수립하는 문서로써 활용되며, 소프트웨어시험 전체의 윤곽이 기술된다. 소프트웨어시험 계획서에 포함될 내용은 <표 1>과 같다. 하나의 소프트웨어시험 계획서가 모든 시험을 포괄할 수 없는 경우, 소프트웨어시험 계획서는 세분화될 수 있다. 이를 위해, 시험대상 소프트웨어는 영역별로 분리가 가능하여야 한다. 소프트웨어를 서로 독립적으로 분리할 수 있는 경우, 이들에 대한 소프트웨어시험 계획은 각각 작성될 수 있다. 예를 들어, 소프트웨어 구성 가운데 데이터입력부와 MMI부는 서로 독립적으로 시험이 수행될 수 있는 부분이 있으므로 이들은 각각 별도의 소프트웨어시험 계획서가 작성될 수 있다.

소프트웨어시험 사양서의 핵심은 테스트케이스 작성에 있다. 테스트케이스에는 시험환경과 신호입력 방법 및 결과출력방법 그리고 예상결과가 기술된다. 입력방법은 모사소프트웨어에 의해 모사신호를 발생하는 방법과 하드웨어장비로 입력신호를 생성하는 방법이 있다. 가능한 현장과 동일한 환경을 만들어 모사신호를 발생하는 방법을 추구한다. 이것은 시스템이 현장에 설치되면 입력신호의 값이 시험 시 예상했던 것 이상으로 변화되기 때문이다. 계산결과를 확인하는 방법으로써 출력값을 프린터로 출력하거나 데이터파일 형태로 작성하는 방법과 하드웨어장비를 통해 출력상태를 확인하는 방법을 선택한다. 전형적인 테스트케이스 양식은 <표 2>와 같다. <표 2>에 나타난 테스트케이스 양식은 SMART MMIS의 모든 소프트웨어 시험에 공통적으로 사용된다.

시험보고서는 시험결과를 분석하여 소프트웨어 개발자에게 제공하여야 하는 문서이다. 시험보고서에는 소프트웨어시험의 성공 또는 실패가 명시되어야 하며, 그 이유에 대해서도 기술되어야 한다.

4. 소프트웨어시험 절차

일관된 소프트웨어시험 수행을 위해 시험절차가 수립되어야 한다. SMART MMIS에 적용할 소프트웨어시험 절차는 <그림 1>과 같다. 시험그룹은 설계문서 가운데 계통설계요건서, 소프트웨어 설계요건서, 소프트웨어 설계서, 운전절차서와 소프트웨어 생산물인 소프트웨어 원시코드(source code)를 반드시 획득하여 시험계획과 테스트케이스를 작성하여야 한다. 원시코드를 획득할 수 없는 COTS 소프트웨어인 경우, 시험그룹은 계통설계요건서와 COTS 소프트웨어 기능요건서 그리고 COTS 소프트웨어를 획득하여 시험계획과 테스트케이스를 작성한다. 시험은 테스트케이스에 따라 수행되며, 모든 테스트케이스를 성공적으로 완료하였을 경우 그 소프트웨어를 시험관점에서 인수한다. 완전한 소프트웨어인수는 소프트웨어의 중요도에 따라 신뢰도분석과 같은 과정을 완료한 후 이루어질 것이다.

시험과정 중에 소프트웨어의 오류가 발견되었을 경우, 시험자는 그 오류가 소프트웨어의 결함 때문에 발생된 것인지 또는 테스트케이스의 오류에 의해 발생된 것인지를 분석한다. 테스트케이스가 잘못된 경우, 테스트케이스를 수정하여 시험을 다시 수행한다. 소프트웨어의 결함이라고 판단되면 소프트웨어 오류보고서를 작성한다. 소프트웨어 오류보고서는 개발자에게 전달되며 시험그룹은 잔여 시험에 대해 계속적인 수행여부를 결정한다. 이에 대한 기준은 발생된 오류가 다른 시험을 수행하는데 있어서 얼마만큼의 영향을 줄 것인가에 달려있다. 발생된 오류가 다른 시험에 대해 독립적이라면 다른 시험을 계속할 수 있지만 그렇지 못한 경우에는 시험은 중단되어야 한다. 오류가 발생된 소프트웨어가 개발자에 의해 다시 수정되고 그에 따른 소프트웨어 변경보고서가 시험자에게 전달될 때까지 시험은 중단된다.

5. 소프트웨어시험 방법

소프트웨어시험 방법은 크게 정적분석(static analysis), 동적시험(dynamic testing), 기타 부가적시험(supplementary testing)으로 분류하였다. 정적분석과 동적시험의 분류기준은 시험을 수행하기 위해 소프트웨어 실행이 요구되는가에 따라 분류하였다. 동적시험은 크게 white-box시험과 black-box시험으로

분류하였다. 본 논문은 NUREG/CR-6421[5]을 참조하여 SMART MMIS에 적용할 소프트웨어시험 방법을 기술하였다.

5.1 소프트웨어 정적분석

소프트웨어 정적분석은 원시코드를 실행시키지 않고 코드자체를 line-by-line 시각적으로 읽어가면서 검토하는 방식(peer review method)이다. 이는 소프트웨어 개발자가 발견할 수 없는 오류를 독립적인 검토를 통해 발견하기 위함이다. 정적분석을 수행하기 위해서는 소프트웨어 요건서, 소프트웨어 설계서, 원시코드가 반드시 제공되어야 한다. 정적분석에서 검토할 내용은 다음과 같다: 소프트웨어 코딩의 일관성, 소프트웨어 개발규범의 준수여부, 소프트웨어 설계서와 코드의 로직 일치성, 사용되지 않는 변수의 존재여부, 잘못된 계산방식, 잘못된 구문(syntax)의 사용 등이다.

본 논문에서 선정한 정적분석 방법은 Fagan이 제시한 3단계 검토방식을 채택한다[6]. 첫 번째 단계는 소프트웨어 요건서를 검토한다. 소프트웨어요건이 상위요건에서 요구하는 기능이 모두 반영되었는가를 검토한다. 두 번째 단계는 소프트웨어 설계서를 검토한다. 소프트웨어 설계서에는 구조, 로직, 데이터 등이 기술되어 있으므로 이들이 소프트웨어요건을 모두 표현하였는가를 검토한다. 세 번째 단계는 원시코드를 검토한다. 원시코드의 소프트웨어표준 준용성 및 로직의 일치성 등에 대해 검토한다. 검토구성원은 최소한 설계자, 코드개발자, 검토자, 중재자로 구성되어야 한다.

먼저, 설계자는 전체적인 설계개념, 기능, 요건, 인터페이스 등을 설명한다. 코드개발자는 자신이 개발한 소프트웨어를 순차적으로 설명해 나간다. 검토자는 사전에 소프트웨어 개발에 입력으로 사용되었던 문서들을 충분히 숙지한 상태에서 설계자 및 코드개발자가 설명한 내용에 대해 질문을 던진다. 이때, 설계자, 개발자와 검토자간에 논쟁이 벌어질 수 있으므로 이를 중재자가 조절한다. 검토가 진행되는 과정에서 중재자의 역할이 가장 중요하며, 검토의 진행여부를 결정할 수 있는 권한을 갖는다. 검토시간은 1회에 2시간을 초과하지 않는다. 검토시간이 2시간을 초과하게되면 검토자의 집중력저하로 인해 검토효율이 떨어진다. 검토자는 check list를 작성하며, check list에 입각하여 요건, 설계, 구현이 이루어졌는가를 확인한다. check list에 포함될 전형적인 내용은 <표 3>과 같다.

소프트웨어 정적분석은 소프트웨어가 모두 개발이 완료된 후 수행되는 것이 아니라 소프트웨어 개발과정 중에 소프트웨어 시험그룹과 개발그룹이 함께 수행하는 것이 효과적이며, 검토시기는 소프트웨어 개발일정에서 결정한다.

5.2 소프트웨어 동적시험

5.2.1 White-box시험

white-box시험은 원시코드를 실행하면서 소프트웨어의 내부구조를 시험하는 방법이다. white-box시험을 소프트웨어의 구조적 시험(structured testing)이라고도 한다. white-box시험을 위해서는 원시코드가 반드시 제공되어야 하며, 원시코드를 구체적으로 설명한 소프트웨어 요건서 및 설계서가 제공되어야

한다. white-box시험은 경로시험(path testing)이 핵심이며 경로시험을 수행하면서 코드의 제어흐름(control flow)과 데이터흐름(data flow)에 대한 정확성이 확인된다.

경로시험 가운데 유일 경로시험(unique path testing 또는 exhaustive path testing이라고도 함)의 수행에 대한 필요성을 고려하여야 한다. 유일 경로시험의 예로써, 2개의 if문에 의해 4개의 분기가 형성되는 문장이 20번의 loop 순환문 안에 있는 프로그램인 경우 4^{20} 즉, 약 100만개 이상의 유일 경로(unique path)가 형성된다. 이와 같은 유일 경로를 시험하기 위해서는 100만개 이상의 테스트케이스를 만들어야 하는데 이는 현실적으로 많은 시간과 인력이 소모될 것이다. 따라서, 소프트웨어의 중요성에 따라 유일 경로시험 수행여부가 결정되어야 한다. 모든 유일한 경로에 대해서 발생되는 상황을 반드시 확인해야 하는 요건이 주어진 소프트웨어에 대해서는 유일 경로시험이 수행되어야 할 것이다.

유일 경로시험은 많은 인력과 시간을 필요로 하는 문제점과 시험의 효용성에 대한 의문을 갖고 있기 때문에 배제되는 경우가 많다. white-box시험을 다음과 같은 세 가지 시험방법으로써 구분하여 수행한다: 제어흐름시험, 순환(loop)시험, 데이터흐름시험이다.

제어흐름시험은 프로그램의 실행흐름을 확인하는 시험으로써 시험을 통해 모든 실행경로가 적어도 한 번 이상 실행되었다면 제어흐름시험은 완료된 것으로 간주한다. 시험방법은 프로그램의 흐름도를 도식하고 각 흐름을 결정하는 변수가 가질 수 있는 값에 대한 변화를 테스트케이스에 작성한다. 테스트케이스의 값은 프로그램이 실행되는 동안 입력되도록 한다. 각 흐름이 수행되었음을 알리는 출력값을 결정함으로써 출력된 결과를 분석하여 시험의 완료여부를 판단한다.

순환시험은 순환이 시작되고 종료되는 경계값 부근에서 소프트웨어오류가 발생할 가능성이 많다는 경험으로부터 비롯된다. 프로그램 개발자는 순환을 결정하는 변수의 초기값과 종료값이 비정상적으로 결정될 경우 순환문은 어떻게 반응할 것인가, 또는 초기값과 종료값이 정상적이라 하더라도 순환문이 요건에서 제시한 내용대로 수행하는지에 대한 시험을 간과할 우려가 있는데, 이를 순환시험을 통해서 확인한다. 시험방법은 순환을 결정하는 변수의 초기값에 대한 최소값(초기값-1으로 결정)과 최대값(초기값+1으로 결정)을 테스트케이스에 작성한다. 초기값의 최소값을 시험하고자 하는 순환문의 초기값으로 하고 최대값을 순환문의 종료값으로 하여 프로그램을 실행한다. 순환문에서 출력된 결과를 분석하여 소프트웨어오류 여부를 판단한다. 종료값에 대해서도 같은 방식으로 시험한다.

데이터흐름시험은 프로그램내의 변수가 그 시점에서 올바로 사용(변수의 정당성이라고 함)되고 있는지를 확인하는 시험이다. 특히, 파일 입출력과 관련된 변수의 사용에서 변수참조 오류가 발생할 가능성이 많다. 예를 들어, 파일이 열리지(open) 않았거나 또는 닫은 후에 파일을 읽거나 쓰려는 경우이다. 시험방법은 프로그램 흐름마다 변수의 사용에 대한 정당성을 확인한다. 변수의 정당성은 테스트케이스에 정의되어야 한다.

white-box시험은 소프트웨어 내부에서 사용되는 변수와 제어흐름이 복잡한 경우, 이에 대한 테스트케이스를 손으로 작성하고 수동으로 시험을 수행한다는 것은 무리이다. 따라서 소프트웨어의 복잡도에 따라 소프트웨어 역공학도구(reverse engineering tool) 또는 자동 테스트케이스 생성도구, 시험분석도구

등을 사용하는 것이 효과적이다.

5.2.2 Black-box시험

black-box시험은 소프트웨어의 내부구조에 대해서는 전혀 관심을 두지 않고, 소프트웨어가 실행하면서 외부적으로 반응하는 결과가 소프트웨어요건에 부합되는가를 확인하는 시험이다. black-box시험의 핵심은 기능적 시험(functional testing)이다. 기능적 시험은 소프트웨어가 동작한 후 요구된 시간에 요구된 결과를 정확히 출력하는지 확인하는 시험이다. 기능적 시험을 수행하기 위해서는 소프트웨어 설계문서 및 운전절차서로부터 외부적 요소(운전원 행위, 센서, 작동기, 디스크, 프린터, 네트웍 등)에 대한 행위를 분석하여 시험하고자 하는 기능을 체계적으로 도출한다. 도출된 기능은 계층적으로 그룹화 되어야 시험이 효과적으로 수행될 수 있다. 일반적으로 프로그램 개발자는 프로그램 개발 시 정상적인 조건과 상황에서 자신의 프로그램이 반응해야할 결과에 중점을 두고 개발하는 경향이 있다. 프로그램 개발자는 비정상 상황 및 상태에서 자신의 프로그램이 반응해야 할 경우를 소홀히 하게되는데, black-box시험은 비정상 조건 및 상황을 유발하는 테스트케이스를 작성하여 소프트웨어를 시험하여야 한다.

black-box시험은 크게 입출력변수시험과 트랜잭션시험으로 구분되어 수행된다. 입출력변수시험은 소프트웨어의 외부입력으로 정의된 모든 변수의 입력값과 입력으로 인해 영향을 받는 외부출력의 결과값을 테스트케이스로 작성하여 수행하는 시험이다. 하나의 입력값이 가질 수 있는 영역이 구분되고 각 영역별로 출력값이 다를 경우 테스트케이스에는 각 영역이 분리된 입력값이 기술되어야 한다. 트랜잭션시험은 운전원 행위 관점에서 수행하는 시험이다. 운전원 행위는 소프트웨어의 외부 입력이 된다. 하나의 외부 입력에 대해 다수의 출력이 일어날 수 있다. 이러한 외부 입력과 소프트웨어 출력은 테스트케이스에 명시되어야 한다.

5.3 부가적 소프트웨어시험

부가적 소프트웨어시험으로써 통계적(statistical)시험, 과부하(stress)시험, 회귀(regression)시험을 들 수 있는데, 이들을 SMART MMIS에 적용하는 것은 부가적 소프트웨어시험에 대한 타당성이 연구된 후 고려될 사항이다.

통계적시험은 주로 소프트웨어의 고장율을 얻기 위해 수행되는 시험으로 알려져 있다. 통계적 시험의 입력은 난수발생기(random number generator)를 통해 이루어진다. 난수를 장시간 입력하여 소프트웨어의 고장율 데이터를 산출하는 통계적시험은 난수가 테스트케이스로써 타당성이 있는지 검토되어야 한다. 예를 들어, 어떤 소프트웨어의 고장율 목표치를 10^{-6}(per demand)로 설정하고 설정된 고장율에 대한 99%의 확신도를 얻고자 한다면 460만 테스트케이스를 수행하여야 한다[5]. 하나의 테스트케이스를 수행하는데 1초가 소요된다면 총 1.75개월 소요된다. 이때, 테스트케이스 입력값을 난수로 해도 무방한지 또는 사전에 정의된 값으로 해야하는지가 결정되어야 한다. 난수를 사용한다면, 난수발생을 극히 짧은 시간에 소프트웨어적으로 생성하여 시험을 수행할 수 있다. 이 경우, 시험시간이 오래 소요되지 않을 것이

다. 그러나 수동으로 시험입력을 만들어야 하는 경우, 통계적 시험은 무리가 있다. 이것은 통계적시험이 적용될 수 있는 환경과 소프트웨어의 특성에 따라 시험의 적절성이 결정되어야 함을 의미한다.

과부하시험은 소프트웨어가 수행되는 시스템에 과부하를 걸어 소프트웨어의 반응을 확인하는 시험이다. 과부하시험을 수행하기 위해서는 소프트웨어의 과부하 요소를 파악하여야 하는데, 그 예로써 입력스캐닝 시간, 프로세스의 수, 과도한 메모리사용, 빈번한 하드디스크 사용, 과대한 파일 또는 레코드 저장, 통신부하 등을 들 수 있다. 소프트웨어 과부하 요소는 테스트케이스에 명확히 기술되어야 한다. 이 시험의 문제점 중의 하나는 과부하를 발생시키는 부하발생기(load generator) 생성에 대한 것이다. 특히, 동적으로 과부하의 양을 조절하여야 하는 경우 부하발생기에 대한 사양은 더욱 복잡하게 된다. 과부하시험은 소프트웨어의 안전성 확인을 위해 필요한 시험이나 과부하 발생의 용이성이 확인된 후 가능한 시험이다.

회귀시험은 이미 시험이 수행되었던 소프트웨어가 수정되어 과거에 수행되었던 소프트웨어시험을 다시 수행하는 시험이다. 이 시험을 수행하기 위해서는 과거의 테스트케이스가 수정된 소프트웨어에 적용하여도 무방한 것인가를 먼저 판단하여야 한다. 수정된 소프트웨어를 위해 작성하여야 하는 테스트케이스가 과거의 테스트케이스를 크게 벗어나지 않은 경우 과거의 테스트케이스를 이용한 시험을 반복할 수 있다. 또는, 과거의 테스트케이스 가운데 수정된 소프트웨어에 적절한 테스트케이스를 추출하여 시험을 수행하는 방법이 있다. 이는 수정된 소프트웨어를 위해 테스트케이스를 재작성 해야 하는 비용을 절감하는 효과가 있다. 이것을 테스트케이스의 표준화라고 하며, 표준 테스트케이스는 그 소프트웨어가 수정되어도 재시험 시 사용될 수 있음을 의미한다.

6. 소프트웨어시험 시 고려사항

소프트웨어시험은 소프트웨어 자체만으로 수행될 수 없고 하드웨어의 동작상태와 밀접한 관계가 있다. 동일한 소프트웨어가 하드웨어 환경변화에 따라 시험결과를 다르게 출력할 수 있다. 이런 현상의 예로써, 동일한 테스트케이스에 대해 공장인수시험과 현장인수시험에서 다른 결과가 발생된다. 이는 현장의 환경에서 시험할 때 공장인수시험 환경에서 고려치 못했던 요소로 인해 시험이 영향을 받기 때문이다. 따라서, 공장인수시험 환경을 가능한 현장의 환경과 동일하게 구성하고, 현장 경험이 풍부한 시험요원을 투입하여야 한다.

소프트웨어시험은 하드웨어적 요소에 많은 영향을 받으므로 어떤 오류는 하드웨어에 의한 오류인지 소프트웨어에 의한 오류인지 구분하는데 많은 시간이 소요되는 경우가 있다. 이런 현상은 소프트웨어가 하드웨어의 결함을 밝히는 시험의 보조수단으로 사용되게 된다. 따라서, 소프트웨어시험 시작 전에 하드웨어의 건전성을 확인하는 작업이 테스트케이스에 기술되어 소프트웨어시험이 하드웨어오류에 의해 혼란이 야기되지 않아야 한다.

소프트웨어개발 일정 산정 시 소프트웨어시험 기간을 충분히 산정하지 않는 경우가 있다. 소프트웨어시험 과정에서 발견된 오류가 소프트웨어를 수정하게 하고 이를 다시 시험하는 과정이 예상치 못하게

자주 또는 길게 발생할 수 있다. 이에 대한 여유를 충분히 감안하여 일정을 산정하여야 한다. 소프트웨어 개발기간과 시험기간을 전체 기간에서 동일하게 산정하는 것이 바람직하다.

소프트웨어시험의 효율을 높이기 위해서는 설계사양서 및 소프트웨어 사양서가 추적성을 충분히 갖도록 정형적으로 기술하는 것이다. 사양서를 정형적으로 기술하는 것은 시스템의 특성에 따라 많은 인력과 시간이 필요하거나 심지어 불가능할 수 있을 수 있으나 가능한 사양서의 내용을 기능블럭도, 상태전이도 또는 흐름도와 같이 도식화(semi-formal인 형태임)할 수 있는 내용으로 기술하는 노력이 필요하다. 이것은 설계자, 개발자, 시험자 간의 의사소통에 오해를 최소화하는데 기여한다.

7. 결론

본 논문을 통해 SMART MMIS 소프트웨어시험에 적용하기 위한 소프트웨어시험 개념을 정립했다. 소프트웨어시험 개념을 소프트웨어시험 조직설정, 소프트웨어시험과 관련된 문서생산, 소프트웨어시험 절차수립, 소프트웨어 시험방법을 중심으로 설정하였다. 소프트웨어시험 방법을 소프트웨어 정적분석과 동적시험 방법으로 분류하였으며, 소프트웨어 동적시험을 white-box 시험과 black-box 시험으로 구분하여 기술하였다. 또한, 부가적인 소프트웨어시험과 소프트웨어 시험 시 고려되어야 할 사항을 본 논문에서 언급하였다. 본 논문에서 설정한 소프트웨어시험 개념을 SMART MMIS 소프트웨어 개발에 적용함으로써 고품질의 SMART MMIS 소프트웨어를 생산할 수 있을 것이다. 추후, 본 논문에서 설정한 개념을 바탕으로 소프트웨어시험 지침 및 절차를 구체화하여 SMART MMIS 시험검증설비 개발 시 적용함으로써 소프트웨어고장 데이터를 취득할 것이다.

Acknowledgement

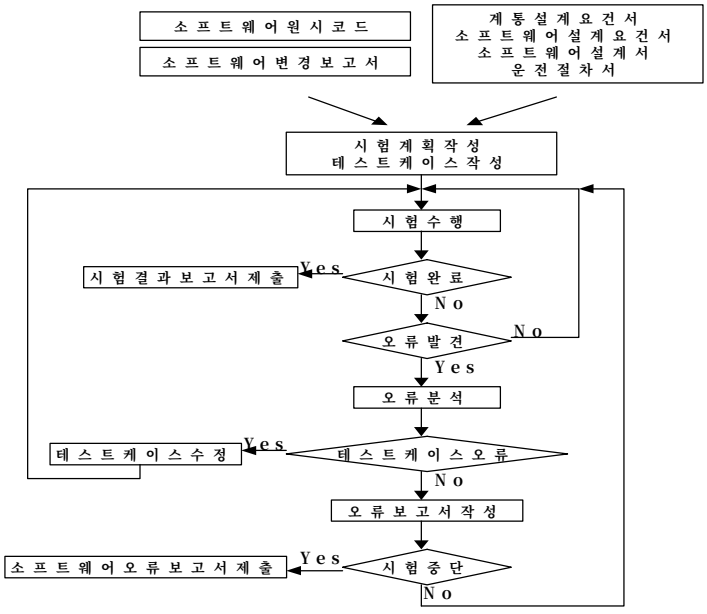
본 연구는 과학기술부의 원자력연구개발사업 일환으로 수행되었음.

[참고문헌]

1. KAERI/RR-1901/98, “일체형원자로 MMIS설계기술개발”, 한국원자력연구소, 1999. 3.
2. 서용석 외, "SMART MMIS 설계에 적용을 위한 소프트웨어 개발 개념", ‘00춘계원자력학회 학술발표회, 2000. 5.
3. IEEE Std 1012-1998, "IEEE Standard for Verification and Validation", IEEE, 1998.
4. IEEE Std 829-1998, "IEEE Standard for Software Test Documentation", IEEE, 1998.
5. NUREG/CR-6421, "A Proposed Acceptance Process for Commercial Off-the-Shelf(COTS) Software in reactor Applications", USNRC, 1996. 3.
6. M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", IBM System Journal, Vol. 15, No. 3, 1976, pp. 182-211.

<표 1 > 소프트웨어시험 계획에 포함될 항목 및 내용

항목	내용
식별자	본 시험계획에 부여된 식별자를 기술. 모든 시험계획은 고유의 식별자를 가져야 함.
개요	수행할 시험에 대한 전반적인 설명을 기술.
시험대상	시험할 요건서, 프로그램 모듈 이름을 나열.
시험할 기능	시험대상에 대해 세부적으로 시험할 기능을 설명. 시험대상의 세부기능이 별도의 문서에 기술된 경우, 그 문서의 이름을 기술.
시험하지 않을 기능	시험대상 가운데 본 시험계획에서 시험하지 않을 기능을 기술하고 그 이유에 대해 기술.
시험방법	시험을 위해 사전에 숙지해야 할 문서를 기술하고 시험에서 주안점을 두고 수행해야 할 항목 등을 기술. 시험을 위해 특별히 요구되는 장비 또는 기술(techniques) 등을 기술.
시험 합격 기준	시험이 통과될 기준을 전반적으로 기술.
시험 실패 기준	시험이 실패할 기준을 전반적으로 기술.
시험 중단/재시작 기준	시험이 중단될 조건과 중단된 시험이 재시작될 조건을 기술. 시험이 재시작될 경우 재시험할 대상을 기술.
시험 결과물	본 시험을 완료 후 인도할 결과물을 나열.
시험 일정	시험직무를 구분하고 각 직무의 일정을 기술.
환경 조건	시험을 수행하기 위해 갖추어져야 할 환경을 기술. 특히, test bed 구축의 필요성을 언급함.
책임 사항	시험에 참여하는 인원의 책임 사항을 기술.
제한 사항	시험을 수행하기 위해 갖추어져야 할 선결사항을 기술. 시험조직, 시험자의 훈련사항, 예산 등을 기술.
승인 절차	시험을 승인할 사람을 기술. 기본적으로 시험 수행자, 시험 분석가, 시험 관리자가 최소 승인자임.



<그림 1 > 소프트웨어시험 절차도

<표 2 > 전형적인 테스트케이스 양식

TEST CASE SHEET			
제목:			
LOG NO.:		작성자:	시험자:
시험시작 날짜/시간:		시험종료 날짜/시간:	
설명:			
초기정렬상태:			
기동방법:			
입력방법:			
측정방법:			
비상/일시 정지방법:			
재기동방법:			
정상정지방법:			
입력변수/TAG번호	입력값	출력변수/TAG번호	출력값

위 테스트케이스 양식의 각 항목에 대한 설명은 다음과 같다.

제목: 테스트케이스의 제목을 기술.
LOG NO.: 테스트케이스에 할당된 고유의 번호를 기술. 시험계획은 모든 테스트케이스에 대한 LOG 번호를 산정해야 함.
작성자: 테스트케이스를 작성한 사람의 이름을 기술.
시험자: 시험을 수행한 사람의 이름을 기술.
시험시작 날짜/시간: 시험을 시작한 날짜 및 시간을 기술.
시험종료 날짜/시간: 시험을 종료한 날짜 및 시간을 기술.
설명: 시험내용의 전반적인 설명을 기술.
초기정렬상태: 시험을 시작하기 전에 하드웨어 및 소프트웨어의 초기정렬상태를 기술.
기동방법: 시험을 시작하기 위해 하드웨어 및 소프트웨어의 기동방법을 기술.
입력방법: 테스트케이스의 입력값을 주기 위한 하드웨어 및 소프트웨어 방법을 기술.
측정방법: 테스트케이스의 출력값을 측정하기 위한 하드웨어 및 소프트웨어 방법을 기술.
비상/일시 정지방법: 시험과정에서 비정상적인 상황에서 시험을 정지시키는 방법을 기술.
재기동방법: 중간에 정지된 시험을 재가동하기 위해 필요한 하드웨어 및 소프트웨어 기동을 기술.
정상정지방법: 시험완료 후 하드웨어 및 소프트웨어 정지 방법을 기술.
입력변수/TAG번호: 입력값을 넣을 변수 또는 하드웨어 TAG 번호 등과 같은 입력 식별자를 기술.
입력값: 입력변수/TAG번호에 영향을 줄 입력값을 기술.
출력변수/TAG번호: 출력값을 측정할 변수 또는 하드웨어 TAG 번호 등과 같은 출력 식별자를 기술.
출력값: 측정할 결과값을 기술.

<표 3 > 소프트웨어 정적분석 수행을 위한 check list

오류유형	확인내용
데이터선언오류	모든 변수(지역, 광역, 외부변수)는 명확히 선언되었으며 초기화 되었는가? (묵시적 선언 또는 초기화가 가능하여도 반드시 명시적으로 선언 및 초기화 한다.)
	변수의 초기값이 프로세스 또는 모듈이 기동될 때마다 정확히 사용되고 있는가?
	변수 선언은 모듈내의 지정된 장소에서 집합적으로 선언되었는가? (모듈 중간에서 선언이 가능하여도 모듈 시작부분에서 변수선언을 명시한다.)
	변수의 type, 크기(특히, 배열), class(static, auto 등) 등이 요건과 부합하는가?
	상수의 선언과 초기화 및 type이 요건과 부합하는가?
	변수 및 상수의 이름이 유사 또는 혼동되게 사용하고 있지 않은가?
	서브루틴 또는 함수 호출시 매개변수의 선언이 일치하는가?
데이터참조오류	선언되지 않은변수가 사용되거나 선언된 변수가 사용되지 않고 있는가?
	배열 또는 포인터가 적절한 메모리 영역을 참조하고 있는가?
	숫자 또는 문자배열의 참조가 변수의 선언영역을 벗어나지 않고 있는가?
	메모리 주소체계와 변수의 type이 일치하는가?
	배열의 첨자(subscript)가 정수(integer)인가?
	같은 메모리 영역을 서로 다른 변수가 공유(aliasing)하지 않고 있는가?
	서브루틴 또는 함수 호출 시 매개변수의 참조 순서가 일치하는가?
	외부(external) 참조 변수의 사용이 요건에 부합하는가?
계산오류	계산에 참여한 변수의 type, 크기가 일치하는가?
	계산 연산자의 사용과 순서가 요건에 부합하는가?(/와 %의 구별 등)
	논리 연산자의 사용과 순서가 요건에 부합하는가?(=와 ==의 구별, &와 &&의 구별, <와 <<의 구별 등)
	계산결과가 대입되는 좌측 변수는 계산결과를 충분히 수용하는가?
	계산결과에 underflow 또는 overflow가 발생하는가?
	실수연산의 근사치를 고려하였는가?
	나눗셈에서 젯수 또는 피젯수에 0(zero)이 발생하는가?
비교오류	비교에 참여한 변수의 type, 크기가 일치하는가?
	비교연사자의 사용과 순서가 요건에 부합하는가?(=와 ==의 구별, &와 &&의 구별, <와 <<의 구별 등)
	복합비교는 단일비교를 괄호로 묶은 후 사용하고 있는가?(예, a<b<c를 (a<b)&&(b<c)로 사용)
	실수값 비교 시 근사치를 고려하였는가?
제어흐름오류	분기변수의 값과 분기위치와 일치하는가? (즉, 분기변수의 값이 분기위치이외의 값은 갖지 않는가?)
	순환(loop)인 경우 무한(infinite)순환이 발생하는가?
	순환의 초기값이 거짓(false)인 경우 순환문 수행은 요건에 부합하는가?
	순환이 시작되거나 종료되는 경계값에 대한 오류 발생 시 예외사항은 마련되었는가?
	분기가 발생할 때마다 분기를 탈출하는 방안이 마련되어 있는가?
	프로그램 종료는 보장되어 있는가?
연계오류	호출(calling) 모듈과 피호출(called) 모듈간의 매개변수의 타입, 크기, 개수가 일치하는가?
	상수를 매개변수로 사용하고 있지 않는가?
	피호출 모듈의 진입점(entry point)이 하나인가?
	광역 및 외부변수에 대한 모듈간 연계는 명확한가?
입출력오류	파일 또는 외부장치는 open, read/write/rewind, close의 순서에 맞게 사용하고 있는가?
	파일 또는 외부장치를 사용하는 함수의 매개변수 사용은 요건에 부합하는가?
	파일 또는 외부장치와의 데이터포멧, 크기는 요건에 부합하는가?
	파일 또는 외부장치의 끝단(예, end-of-file)은 정확히 감지되고 있는가?
	이진(binay) 또는 문자(text) 형태의 데이터 사용은 요건에 부합하는가?