한국철도학회논문집
제11권 제2호 2008년 ▌pp. 139-144

열차제어 소프트웨어 안전성 평가도구의 설계

Design of Train Control Software Safety Evaluation Tool

황종규* · 조현정** · 김형신†
Jong-Gyu Hwang · Hyun-Jeong Jo · Hyungshin Kim

Abstract Recent advances in embedded system technology have brought more dependence on automating train control. While much efforts have been reported to improve electronic hardware's safety, not so much systematic approaches to evaluate software's safety, especially for the vital software running on board train controllers. In this paper, we propose a new software tool to evaluate software safety for the train controller. We have reviewed requirements in the international standards and surveyed available tools in the market. From that, we identified necessary tests to meet the standards and proposed a tool that can be used during the whole software life cycle. We show the functional architecture and internal components of the tool. Our tool is unique in that it is a comprehensive tool specifically designed for software safety evaluation while other tools are not.

Keywords : Software safety, TCS(Train Control System), RAMS, Software testing, Vital software

요 지 최근 임베디드 시스템 기술의 발전에 따라 열차제어시스템의 자동화가 촉진되고 있다. 이를 위한 하드웨어의 안전성 검증을 위한 연구는 활발히 진행되고 있으나, 소프트웨어의 안전성을 검증하기 위한 체계적인 노력은 거의 이루어지지 않고 있다. 이 논문에서는 열차제어 시스템 소프트웨어의 안전성을 자동으로 평가할 수 있는 소프트웨어 도구를 제안한다. 이를 위하여 관련 국제표준을 분석하였으며, 기존의 사용 소프트웨어 테스팅 도구들을 조사하였다. 이로부터 국제표준에서 요구하는 주요 요건을 테스팅 할 수 있는 도구를 제안하였으며, 이 도구는 소프트웨어 전 개발주기에서 이용이 가능하다. 이 도구는 기존의 테스팅 도구들과는 달리 소프트웨어의 안전성을 검증할 수 있다는 점에서 그 의의가 크다.

주 요 어 : 소프트웨어 안전성, 열차제어시스템, 램즈, 소프트웨어 테스팅, 바이탈 소프트웨어

1. 서 론

열차제어 시스템은 최근 기존의 기계적 장치로부터 컴퓨터시스템으로 전환되고 있으며, 소프트웨어에의 의존성이 급격하게 증가하고 있다. 대표적인 시스템으로는 일본의 EJTC[1] 자동열차제어시스템(ATC)을 들 수 있다. 차상 탑재 컴퓨터의 소프트웨어는 열차운행의 자동화와 자율화 추세에 따라 그 역할이 더욱 중요해 지고 있으며, 따라서 소프트웨어가 열차제어 시스템 전체에 미치는 영향도 커지고 있다. 차상 탑재 컴퓨터는 마이크로 프로세서 시장의 급속한 발달에 따라 점차 고성능화 하고 있으며, 사용하는 프로그래밍 언어도 단순한 어셈블리어로부터 최근 상위수준 언어인 Ada가 이용되기도 하였다[2]. ATC 소프트웨어는 스웨덴 Teknogram사의 경우 1만 라인의 어셈블리어, 약 30 킬로바이트의 바이너리 이미지를 갖는다[2]. 열차제어 시스템소프트웨어의 크기와 복잡도는 하드웨어의 발달 속도보다는 느리지만, 점차적으로 규모가 커지며, 복잡도도 증가할 것을 예상된다. 이와 같이 초기의 기계적, 수동 차량신호시스템에서부터 최근의 무인 자동 열차제어 시스템으로 변천되어 오면서, 다수의 컴퓨터들이 차상장치로 사용되기 시작했으며, 이들 컴퓨터에 탑재되는 소프트웨어의 안전성을 검증하는 것이 중요한 문제도 대두되기 시작했다.

소프트웨어의 안전성은 주로 소프트웨어의 개발초기 단계인 소프트웨어 설계 단계에서 안전성 활동을 수행함으로써 이루어진다. 대표적인 안전성 활동으로는 HAZOP[5],

† 책임저자 : 정회원, 충남대학교 컴퓨터공학과 부교수
E-mail : hyungshin@cnu.ac.kr
TEL : (042)821-5446 FAX : (042)822-9959
* 정회원, 한국철도기술연구원 전기신호연구본부 선임연구원
** 정회원, 한국철도기술연구원 전기신호연구본부 주임연구원

FTA[3,4], FMECA[6] 등을 들 수 있다. 소프트웨어의 개발 초기에는 이와 같은 기법들이 활동되고 있으나, 개발이 완료된 이후에는 추가적인 안전성에 대한 검증은 자동화되거나 정형화된 방법이 적용되지 않고 있다.

최근 소프트웨어의 모델을 구현하고, 모델 검증을 통하여 소프트웨어의 안전성을 검증하는 모델기반 소프트웨어 개발 방법론[12]이 소프트웨어 안전성 개선을 위한 주요 기법으로 주목을 받고 있다. 그러나, 이 방법은 [13]에서 지적한 것과 같이 모델의 정확성을 보장하는 것이 가장 큰 과제로 남아 있다. 모델 기반 개발을 지원하는 개발도구들로는 Mathworks 사의 Simulink[14], Stateflow[15], Esterel 사의 Esterel[16], SCADE[17], I-Logics사의 Statemate[18], Safeware사의 SpecTRM[19] 등이 있으나, 이들 도구들은 소프트웨어의 안전성을 검증하는 도구라기보다는 소프트웨어 개발을 지원하는 도구들로 본 연구에서 제안하는 안전성 검증 도구와는 다르다.

소프트웨어의 안전성을 인증하는 경우에는 피인증기관의 안전성 활동을 평가할 때, 피인증기관이 제공하는 문서에 의존하여 평가하는 경우, 소프트웨어 안전성 활동의 충실성을 파악하기가 어려우며, 경우에 따라서는 추가적인 검증이 필요하게 된다. 이런 경우 자동으로 소프트웨어의 안전성을 평가할 수 있는 도구가 있다면, 평가의 신뢰성을 높일 수 있어 큰 도움이 된다.

본 연구에서는 열차제어 시스템 소프트웨어의 안전성을 자동으로 평가할 수 있는 소프트웨어를 제안한다. 이 연구의 의의는 다음과 같다. 첫째, 본 연구에서 제안하는 도구는 기존의 소프트웨어 개발도구들과는 달리 소프트웨어의 안전성을 전문적으로 검증할 수 있는 도구이다. 기존의 소프트웨어 개발 도구들은 안전성을 평가하기 위한 도구라기보다는 신뢰성 있는 소프트웨어 개발을 위한 개발지원 도구들이다. 둘째, 소프트웨어 안전성 관련 인증도구로 이용이 가능하도록 설계되었다. 이를 위하여 관련 국제표준을 분석하였으며, 국제표준에서 요구하는 다양한 요구조건 중에서 자동화가 가능한 평가항목들을 도출하였다. 이로부터 IEC61508[7] 및 IEC62279[8] 표준에서 요구하는 주요 요건을 테스팅 할 수 있는 도구를 제안하였다. 이 도구의 개발이 완료되면 향후 공인기관을 통한 객관적 소프트웨어 안전성 검증에도 이용이 될 수 있을 것으로 기대된다. 이 논문은 도구의 설계단계의 결과를 정리한 것으로, 논문의 구성은 다음과 같다.

2장에서는 본 논문에서 제안하는 열차제어 소프트웨어의 안전성 평가 방법을 설명하고, 3장에서는 이를 자동화한 테스팅 도구의 요구사항을 정리하였다. 4장에서는 제안하는 도구에서 구현해야 하는 주요 테스팅 항목을 설명하고, 5장에서는 테스팅 도구의 구조 및 주요 기능을 제안하고, 6장에서 결론을 맺는다.

2. 열차제어 소프트웨어의 안전성 평가

본 연구에서는 일반 소프트웨어 안전성 국제 표준인 IEC61508[7]과 열차제어 분야의 국제 표준인 IEC62279[8]를 분석하여 소프트웨어 안전성 테스트 방법을 제안하였다. IEC62279에서는 소프트웨어 안전 무결성 수준(Software Safety Integrity Level, SWSIL)을 정의하고, 정형화된 개발프로세스를 제안하고, 각 프로세스별로 SSIL 수준에 따라 검증기법을 제시하고 있다.

본 연구에서 제안하는 소프트웨어 안전성 평가는 소프트웨어 설계 단계 이후인 구현, 확인(Verification) 및 시험, 하드웨어 결합시험, 검증(Validation), 평가(Assessment) 단계에서 수행해야 하는 평가 항목들을 추출하였다. 이 절에서는 IEC62279에서 요구하는 소프트웨어 안전성 요건에 관해 설명한다.

2.1 소프트웨어 안전 무결성 수준

소프트웨어 안전성의 평가는 소프트웨어 설계시에 부여되는 소프트웨어 안전 무결성 수준(Software Safety Integrity Level, SWSIL)을 개발된 소프트웨어가 만족하는지를 검증함으로써 이루어진다. SWSIL은 소프트웨어 자체적으로 정의되지 않으며, 소프트웨어를 적용하는 시스템의 안전 무결성 수준(SIL)과 동일하도록 결정된다. 그러나, 소프트웨어의 오류가 시스템으로 전파되지 않도록 할 수 있는 경우에는 이보다 낮은 수준으로 정할 수 있도록 하였다.

SWSIL은 시스템의 위험도에 따라 다음과 같이 5 등급으로 분류한다.

SWSIL	안전 무결성
4	매우 높음
3	높음
2	보통
1	낮음
0	비안전성 등급

2.2 개발단계별 안전성 검증방법

IEC62279에서 제시하고 있는 소프트웨어 개발 프로세스는 그림 1과 같이 개발 프로세스와 검증 프로세스들로 구성되어 있다. 표준에서는 각 개발 단계마다 만족해야 하는 요건들을 제시하고 있으며, 주요 요건들에 대해서는 SWSIL에 따라 의무(M, Mandatory), 강한권고(HR, Highly

recommend), 권고(Recommend), 비권고(Not Recommend) 등으로 구분하여 요건의 검증기법을 제시하고 있다. 테스팅에 기법을 적용함에 있어서 자동화된 테스팅 도구의 사용을 권장하고 있다. 다음은 표준에서 제시된 검증기법들 중에서 최상위 안전등급인 SWSIL 4 등급의 경우 의무(M) 및 강한권고(HR)로 요구된 검증기법들을 중심으로 정리한 것이다.

2.1.1 소프트웨어의 설계 및 구현

소프트웨어의 설계 및 구현과정에서는 코딩 규칙준수와 블랙박스 테스팅을 의무검증 기법으로 요구하고 있다. 요구하고 있는 코딩 규칙으로는 동적객체 및 변수의 사용금지, 포인터 및 재귀함수 사용의 제한, 무조건적 점프 사용 금지를 제안하였다. 블랙박스 테스팅과 함께, 성능시험과 인터페이스 시험을 강한권고로 구분하였다.

2.1.2 소프트웨어 확인(Verification) 및 테스팅

확인 및 테스팅 기법들로는 정형증명(Formal Proof), 확률시험, 정적분석, 동적 분석, 소프트웨어 오류효과분석(Software Error Effect Analysis) 등을 강한권고 항목으로 제안하고 있다.

2.1.3 소프트웨어/하드웨어 통합시험

소프트웨어와 하드웨어 통합시험단계에서는 기능시험, 블랙박스 시험, 그리고 성능시험을 강한권고 수준으로 요구하고 있다.

2.1.4 소프트웨어 검증(Validation)

검증단계에서는 성능시험과 기능시험, 그리고 블랙박스 시험을 의무, 확률시험을 강한권고 항목으로 제시하고 있다.

2.1.5 소프트웨어 평가(Assessment)

이 단계에서는 개발 프로세서들과 개발된 소프트웨어가 설계초기에 정의된 SWSIL을 만족하는지 최종적으로 검증하는 것을 목적으로 하고 있다. 사용해야 하는 검증기법들로는 체크리스트, 정적분석, 동적 분석, 고장트리 분석(Fault Tree Analysis), 소프트웨어 오류 영향분석, 공통원인 고장분석(Common Cause Failure Analysis) 등을 강한권고 항목으로 제시하고 있다.

3. 소프트웨어 안전성 평가도구 요구사항

IEC62279에서는 열차제어시스템의 소프트웨어 안전성을 검증하기 위해서 자동화된 테스팅 도구의 사용을 권장하고 있다. 소프트웨어 안전성을 평가하는 자동화된 도구를 설계하기 위하여 다음과 같은 요구사항들을 도출하였다.

1) 열차제어 소프트웨어 관련 국제표준에서 요구하는 안전성 요건을 검증할 수 있어야 한다.

본 연구에서는 IEC61508와 IEC62279를 분석하여 국제표준에서 요구하는 소프트웨어 안전성 검증요건들을 도출하였다. 소프트웨어 수명주기의 개발 단계부터 요구되고 있는 테스팅 및 검증기법들을 분석하여 자동화할 수 있는 기법들을 도출하였다. 테스트 도구는 소프트웨어 매 개발단계에서 사용이 가능해야 하며, 표준에서 단계별로 요구하는 검증항목들을 자동으로 적용할 수 있어야 한다. 따라서, 개발되는 도구를 사용하면 IEC61508과 62279의 준수 여부를 객관적으로 검증할 수 있게 되어, 개발과정의 신뢰성을 높일 수 있다. 도출된 평가항목들은 4절에서 기술한다.

2) 평가자의 선택에 따라 다양한 형태의 테스트 조합이 수행 가능해야 한다.

개발하는 도구는 소프트웨어 개발자, 평가자 및 인증기관에서도 사용할 계획이며, 개발과정에서 사용한 테스트 기법을 인증기관에서 확인을 목적으로 적용할 수 있도록 정형화된 테스트를 수행하는 것이 아니라, 테스트들을 사용자의 선택에 의해 구성할 수 있어야 한다. 테스트하는 안전성 수준에 따라서, 개발단계에 따라서 필요한 테스트들을 조합하는 것이 가능해야 한다.

3) 소프트웨어의 정적 분석과 동적 분석이 가능해야 한다.

기존 소프트웨어 자동 테스트 도구는 소스코드 분석에 의

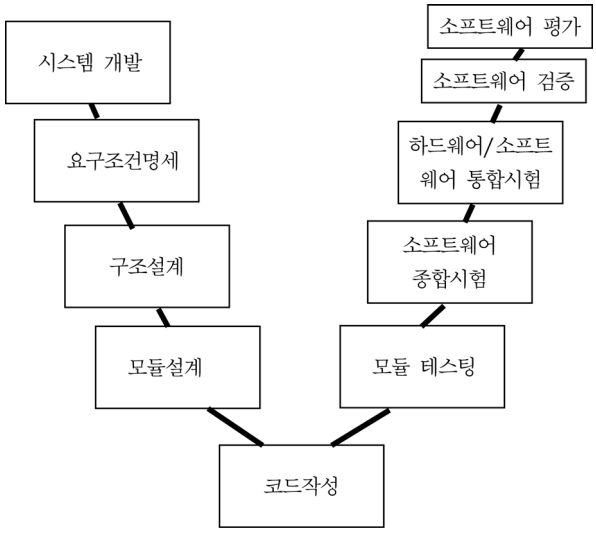

Fig. 1. Software Development Life Cycle in IEC61508-3 [7]

한 정적분석 도구들이 대부분이다. 그러나, 정적분석으로 확인할 수 없는 런타임 특성들의 검증을 위해 동적 분석의 중요성이 최근 부각되고 있다[10]. 동적 분석을 위해서는 범용성이 높은 임베디드 타겟 보드에서 실행 가능한 에이전트 프로그램들을 개발해야한다.

4) 설계단계의 주요 안전성 활동 결과물을 테스트 도구에서 활용할 수 있어야 한다.

기존의 소프트웨어 자동 테스트 도구들은 신뢰성 향상을 목적으로 이용되어 왔으며, 안전성 검증에도 이용이 가능하지만, 안전성 검증을 위한 기능을 추가로 제공하지 않고 있다. 테스팅 도구들은 테스트 케이스들을 사용자가 직접 입력하거나, 무작위로 생성하도록 설계되어 있어서, 설계 단계에서 수행하는 안전성활동의 결과물과 개발된 제품과의 연계성을 보장하기가 어렵다. 이러한 문제점을 해결하기 위해서는 설계 단계에서 이용하는 고장트리분석(Fault Tree Analysis)이나, 해저드분석(HAZOP)의 결과를 개발단계에서 검증할 수 있는 기능이 필수적이다. 따라서, 새로이 개발하는 안전성 평가 도구에서는 기존의 소프트웨어 신뢰성 테스팅 도구들과는 차별화하여 설계단계의 안전성활동 결과물들을 자동으로 입력물로 받아들여 테스트 케이스를 생성하고, 검증하는 기능을 보유하도록 하였다.

5) 테스팅 도구는 국제 안전성 표준에 부합해야 한다.

본 연구에서 개발하는 안전성 테스트 도구는 국제표준의 인증에 이용될 수 있으며, 이를 위해서는 도구 자체도 인증을 받아야 한다. 따라서, 개발하는 도구도 IEC61508 또는 DO-178B 등의 인증을 받아야 한다.

4. 소프트웨어 안전성 평가도구의 평가항목

이 절에서는 IEC61508과 IEC62279에서 정의한 안전성 검증 요건들 중에서 안전성 평가도구에서 구현하기위해 선정된 평가항목들을 소개한다. 표준에서 요구하고 검증방법 중에서 자동화 도구로 구현 가능한 것들을 우선적으로 선정하였으며, 여러 기법 중에서도 의무(M)와 강한권고(HR)로 규정된 검증기법들 중심으로 추출하였다.

평가항목의 도출을 위하여 소프트웨어의 구현물이 작성되거나 혹은 작성된 이후에 검증되어지는 행위와 관련한 단계를 중심으로 크게 ST1~ST6까지의 총 6 단계로 구분하였으며, 이들은 다음과 같다.

ST1 : 소프트웨어 모듈 테스팅 단계
ST2 : 소프트웨어 통합 테스팅 단계
ST3 : 하드웨어와 소프트웨어의 통합 단계
ST4 : 소프트웨어 검증 단계
ST5 : 소프트웨어 변경 검증단계
ST6 : 소프트웨어 평가 단계

각각의 단계는 IEC61508의 소프트웨어 개발단계들로부터 소프트웨어 평가를 위한 항목을 포함하는 단계들을 추출한 것이다. 열차제어시스템 소프트웨어에 직접적으로 연관된 IEC62279에서는 주요 소프트웨어 개발단계별로 자동화된 테스트를 요구하고 있으며, 적용해야 하는 주요 개발기법, 테스팅 기법들이 소프트웨어의 안전무결성 수준에 따라 각각 정의되어 있다. 소프트웨어의 안전성 평가도구에서는 이들 중에서 최상위 안전성을 위하여 소프트웨어 안전무결성수준 4에 적용할 수 있으며, 해당 기술의 사용이 의무(M)이거나 강한권고(HR)로 구분된 기술에 한하여 도구에서 구현할 요구사항으로 도출하였다.

Table 1은 이와 같은 방식으로 도출된 12개의 핵심 평가항목을 보여준다. Table 1은 소프트웨어의 각 개발단계에 적용할 수 있는 시험항목을 규정하고 있다. 예를 들어, 성능시험(Performance testing)은 소프트웨어 모듈 테스팅, 통합테스팅, 하드웨어 통합테스팅, 그리고 변경 검증단계에서 사용할 수 있음을 의미한다. 평가항목들은 평가 기법에 따라 화이트박스 테스팅, 블랙박스 테스팅, 그리고, 소스코드 분석의 세 가지 기법으로 분류할 수 있다. 화이트박스 테스팅과 블랙박스 테스팅은 소프트웨어를 타겟에서 실행하며 분석하는 동적 테스팅 방법이며, 소스코드 분석 기법은 소프트웨어를 실행하지 않은 상태에서의 정적 분석방법이다.

화이트 박스 기법은 시험의 대상인 소프트웨어의 내부구조에 대한 정보를 테스팅 시에 이용할 수 있는 경우에 해당하며, 블랙박스는 그 반대의 경우이다. 화이트 박스 테스팅으로 구현하게 되는 테스트 항목은 성능 테스트, 제어흐름 및 데이터 흐름 테스트이다. 성능 시험은 소프트웨어 구현 시에 요구되는 하드웨어 처리 능력과 자원상태를 동적 테스팅의 형태로 수행하는 것이다. 제어 및 데이터 흐름 테스팅에서는 소프트웨어에서 발생되는 제어흐름과 데이터 흐름을 추적하여 미사용 코드 영역이나 데이터 영역이 발생하는지를 시험한다.

블랙박스 테스팅은 경계값 분석(Boundary value analysis), 동일 클래스 테스팅(Equivalent class testing)으로 구성되어 있다. 경계값 분석 테스팅은 본 절에서 정의한 소프트웨어 개발 6단계 모두에 적용해야 하는 테스트로, 매개변수의 한계 또는 경계에서 발생하는 소프트웨어 오류를 검사한다. 동일 클래스 테스팅은 최소한의 시험데이터를 이용하여 입력변수에 의한 오류를 검출하는 시험이다. 이를 위

Table 1. Selected Testing Items

시험기법	수행단계						기법분류
	ST1	ST2	ST3	ST4	ST5	ST6	
Performance testing	×	×	×		×		Non-Functional
Boundary value analysis	×	×	×	×	×	×	Black-box
Equivalent classes	×	×	×	×	×	×	Black-box
Design & coding standard	×	×					Maintenance
Control flow testing	×	×	×	×	×	×	White-box
Data flow testing	×	×	×	×	×	×	White-box
Fagan inspection					×	×	Analysis
Symbolic execution					×	×	Analysis
Checklist						×	Analysis
Metrics	×	×				×	Analysis
Decision table						×	Analysis
FTA						×	Analysis

해서는 입력 값의 모든 범위가 포함되도록 시험 데이터를 조절하는 것이 중요하다.

정적분석방법으로는 기존의 소프트웨어 분석도구들에서 제공하는 일반적인 기능들을 포함한다. Fagan 검사에서는 외부 전문가들이 수행하던 일반적인 소프트웨어 검사를 자동화한 것이며, Metric 분석은 소프트웨어의 구조적인 특성을 분석하여 신뢰도나 복잡도와 같은 소프트웨어 고유의 특성을 측정하는 것을 말한다. 기존의 소프트웨어 신뢰성 분석 도구와는 달리 본 도구에서는 소프트웨어의 구현단계에서 안전성 검증을 위해 고장트리분석(FTA, Fault Tree Analysis) 도구를 구현한다. 일반적으로 FTA는 소프트웨어의 설계단계에서 이용되지만, 우리가 개발하는 테스트 도구에서는 설계단계에서 수행한 FTA의 결과를 입력으로 제공하여, 설계단계에서 구현된 소프트웨어 안전성이 최종 구현단계 이후에 정확히 구현되었는지 체크해 주는 기능을 제공한다. 아직까지 상용화된 제품 중에서 이와 같이 소프트웨어의 안전성까지 검증할 수 있는 도구는 발표되지 않았다.

5. 소프트웨어 안전성 평가도구의 구조

4절에서는 관련 국제표준을 분석하여 자동화가 필요한 소프트웨어 안전성 평가항목을 제시하였다. 이 절에서는 소프트웨어 안전성 평가도구의 구조설계에 대하여 설명한다.

열차제어시스템 소프트웨어 안전성 평가도구는 테스트 케이스 자동생성도구, 테스트 자동수행 및 모니터링 도구, 그리고 타겟 테스팅 에이전트로 구성되어진다. 열차제어시스템은 임베디드 제어시스템의 특성을 가지고 있으므로, 응용 S/W가 포팅된 실제 타겟보드의 테스팅 에이전트 프로그램을 통해 테스팅 및 모니터링 되는 S/W 테스트 도구의 구조가 설계되어야 한다. 따라서, 테스트 도구는 소스코드와 입력자료 변환모듈을 이용하여 평가대상 소프트웨어의 안전성 분석 데이터를 변환하여 입력받고, 입력받은 소스코드와 안전성 분석 데이터를 바탕으로 테스트 데이터 자동생성모듈과 테스트 시나리오 자동생성모듈을 이용하여 테스트 데이터 및 시나리오를 생성한다. 이 생성된 테스트 케이스를 테스트 자동수행 및 모니터링 모듈과 타겟 테스팅 에이전트를 통해 테스팅을 수행하여 시험결과를 분석하고, 그 결과를 화면 및 파일로 저장하는 구조를 갖는다. Fig. 2는 제안하는 안전성 테스팅 도구의 사용과정을 보여준다.

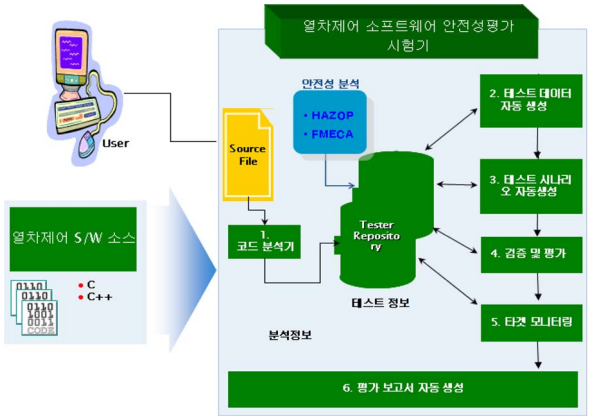

Fig. 2. Software Safety Testing Flow

테스팅 도구의 주요 단계별 기능정의는 다음과 같다.

- 프로그램 분석 : 프로그램 분석을 통해 테스트에 필요한 함수 정보, 타입정보, 제어흐름, 함수 간 호출 정보를 생성
- 입력 분할 : 프로그램 분석을 통해 얻은 정보 기반으로 데이터 타입 별로 입력 데이터 분할에 대한 정보를 생성
- 테스트 시나리오 생성 : 테스트 시나리오를 자동으로 생성. 테스트 시나리오(스크립트)는 사용자가 추가 생성할 수도 있도록 함
- 드라이버 생성 : 테스트 대상 코드와 테스트 엔진을 연결하는 드라이버와 테스트가 수행될 프로그램이 생성
- 실행 : 테스트를 수행하고, 테스트 커버리지, 구간별 테스트 내역, 및 테스트 결과를 요약. 오류 위치를 제시하고 테스트 케이스별로 상세한 결과를 사용자에게 보고하는 기능을 가지도록 함.
- 보고서 작성 : 모든 테스트 정보와 결과를 사용자의 옵션에 따라 리포트를 생성

Fig. 3은 테스팅 도구의 단계별 주요 기능을 보여주며, Table 2는 테스트 도구의 기능 및 출력물을 보여준다.

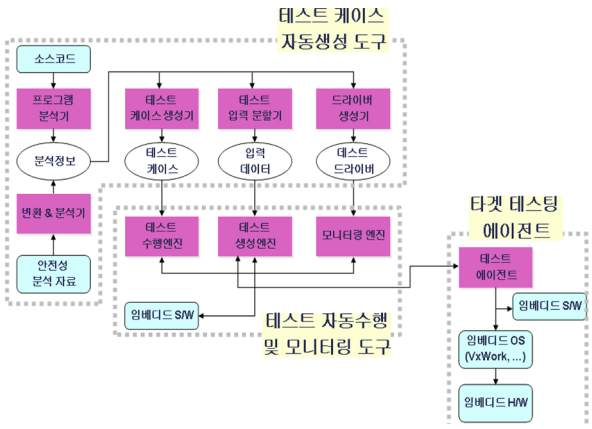

Fig. 3. Software Testing Tool System Design

Table 2. Functions of Software Testing Tool

구분	세부구분	출력물
프로그램분석	분석	함수 및 API 리스트
		함수 별 제어흐름 그래프
		함수 간 호출 그래프
		데이터 구조
테스트케이스 생성	타입 분할	자료형별 분할
	변수 분할	변수별 분할
	시나리오	테스트 시나리오
	드라이버	테스트 드라이버
컴파일 & 빌드	빌드	테스트 프로그램
		타겟 이미지
테스트 수행	실행	테스트 케이스별 수행결과
		테스트 커버리지
		에러진단
결과분석	보고	테스트 보고서

6. 결 론

본 논문에서는 열차제어 시스템소프트웨어의 안전성 평가도구를 제안하였다. 제안한 평가도구는 기존의 자동화된 소프트웨어 테스트 도구를 확장하는 형태를 가지고 있으며, 소프트웨어 개발주기에서 파생된 안전성 활동의 결과들을 입력으로 이용하여 표준에서 요구하는 평가항목들을 동적테스트 형태로 수행하도록 한다. 국제 표준에서 요구하는 핵심 평가항목들을 포함하도록 하였으며, 소프트웨어 개발 수명주기동안 이용될 수 있도록 하였다. 소프트웨어 설계단계에서 수행한 안전성 활동 결과를 테스팅 도구의 입력으로 이용하여 안전성 여부를 지속적으로 검증하는 기능을 추가하였다.

제안된 구조를 갖는 임베디드 소프트웨어 시험도구가 개발된다면 열차제어 시스템의 소프트웨어 안전성을 평가하는데 큰 도움을 줄 수 있을 것으로 기대된다.

참 고 문 헌

1. Matsumo, M. (2005), “The revolution of train control system in Japan”, Autonomous Decentralized Systems, ISADS Proceedings, pp.599 - 606
2. Lawson, H. W. et al., (2001), “Twenty years of safe train control in Sweden”, Engineering of Computer Based Systems, Proceedings. Eighth Annual IEEE International Conference and Workshop on the, p.0289.
3. Oh, Y.J. et al., (2005), “Software safety analysis of function block diagrams using fault trees”, Reliability Engineering & System Safety, Vol. 88, No. 3., pp. 215-228.
4. Weber, W. et al., (2005), “Enhancing software safety by fault trees: experiences from an application to flight critical software”, Reliability Engineering & System Safety, Vol. 89, No. 1., pp. 57-70.
5. Pirie, Ian B., (1999), “Software ? How do we know it is safe?”, Proceedings of the ASME/IEEE, pp.122-129.
6. Lutz, R. R. and Woodhouse, R. M. (1999), “Bi-directional Analysis for Certification of Safety-Critical Software”, Proceedings of 1st International Software Assurance Certification Conference, Dulles, Virginia, February.
7. International Electrotechnical Commission (IEC) (1999), “61508 - Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems”.
8. International Electrotechnical Commission (IEC) (2002), “62279 Railway Applications °© Communications, Signalling &. Processing Systems, Software for Railway Control & Protection”.
9. RTCA DO-178B/ED-12B (1992), “Software Considerations in Airborne Systems and Equipment Certification”.
10. Sammapun, U., Lee, I. and Sokolsky, O., (2005), “RT-MaC : Runtime monitoring and checking of quantitative and probabilistic properties”, Proceedings of IEEE RTCSA.
11. Leveson, N. G. (1986), “Why, what and how ?”, ACM Computing Surveys, 18(2), June.
12. France, R. and Rumpe, B. (2007), “Model-driven development of complex systems: A research roadmap”, Future of Software Engineering, ACM Press.
13. Mathworks Inc. (2004), Mathworks, http://www.mathworks.com
14. Mathworks Inc., Simulink, http://www.mathworks.com
15. Esterel Technologies (2004), http://www.esterel-technologies.com
16. Esterel technologies (2004), SCADE suite product, http://www.esterel-technologies.com
17. Harel, D. et al, (1990), “Statemate: A working environment for the development of complex reactive systems”, IEEE Transactions on Software Engineering, 16(4):403-414, April.
18. Leveson, N. G., Heimdahl, M. P. and Reese, J. D. (1999), “Designing Specification Languages for Process Control Systems: Lessons Learned and Steps to the Future”, In *Seventh ACM SIGSOFT Symposium on the Foundations on Software Engineering*, volume 1687 of *LNCS*, pages 127-145, September.

(2007년 12월 3일 논문접수, 2008년 4월 22일 심사완료)