철도시스템 고안전 소프트웨어 검증을 위한 도구 개발 방안 및 실제

| 윤 광 식 |
슈어소프트테크(주)
공공 및 제조사업부 부사장

| 배 현 섭 |
슈어소프트테크(주)
대표이사

Ⅰ. 개요

전자 및 컴퓨터 기술의 발달에 따라 기존의 기계, 전기적 장치에 의해 제어되던 열차제어시스템을 비롯한 많은 인간의 생명 및 재산에 큰 손해를 끼칠 수 있는 고안전 시스템(Safety-critical system)들이 점점 더 컴퓨터화되고 있다. 고안전 시스템에서 컴퓨터의 활용의 증가는 특히나 소프트웨어 활용의 증가로 이루어지고 있다. 이는 하드웨어에 비해 상대적으로 다양한 경우를 처리할 수 있는 소프트웨어의 유연성과 비용 및 시간 측면의 효율성으로 인한 것이다. 철도의 열차제어시스템 소프트웨어는 이러한 소프트웨어의 예로서 이러한 고안전 시스템의 제어 소프트웨어의 신뢰성과 안전성은 전체 시스템의 신뢰성과 안전성에 직접적인 영향을 끼치게 되므로 고안전 시스템 소프트웨어의 신뢰성과 안전성을 검증하기 위한 체계적인 방법론 및 검증 도구의 지원의 필요성이 매우 크다고 할 수 있다.

이러한 필요에 따라, 국제적으로 안전성 중요 소프트웨어의 개발 및 검증, 평가에 관한 표준화가 이루어지고 있으며, 이러한 노력의 결과 IEEE 1012[1] 와 같은 소프트웨어 검증에 관한 표준 및 IEC 61508[2], IEC 62279[3], RTCA/DO-178B[4] 등이 활용되고 있다. 또 이러한 표준에 대한 검토를 바탕으로 고안전 소프트웨어 개발을 위한 설계 및 개발 방안이 제시되었다.[5] 하지만, 이러한 개발 방안에 따라 검증 도구를 개발하기위해서는 설계에 관한 연구들에서 고려되지 못한 세부 사항들을 고려하여 도구를 개발할 필요가 있다.

본 기고에서는 철도시스템 고안전 제어 소프트웨어의 검증 도구를 개발하기 위하여 표준을 분석하여 각 도구의 기능 요구사항을 도출하는 과정을, 사례를 중심으로 소개하며 이를 바탕으로 개발된 검증 자동화 도구의 주요 기능을 소개한다.

Ⅱ. 철도시스템 고안전설 소프트웨어 검증 활동 및 지원 도구 개발 방안

고안전 시스템 소프트웨어의 검증을 위해서 많은 국제 표준이 제안되었으며, 이들 국제 표준들은 고안전 소프트웨어를 개발하기 위한 활동외에도 고안전 소프트웨어를 검증하기 위한 활동도 제안하고 있다.

IEEE 1012[1]는 소프트웨어의 검증 기준에 관한 표준으로서 기타 고안전 소프트웨어의 검증 요구 사항을 결정하기 위한 가장 기본이 되는 표준이다. IEEE 1012는 ISO/IEC 12207[6] 의 획 득 (Acquisition), 공 급 (Supply), 개 발 (Development), 운 용 (Operation), 유 지 보 수 (Maintenance)등 5개의 유형의 프로세스에 대해 필요한 검증 활동(Activity)과 상세 검증 작업(Task)를 정의하고 있다. 또 소프트웨어를 오동작이 미치는 영향에 따라 4개의 SIL(Software Integrity Level)로 구분하고, 각 SIL 수준의 소프트웨어의 검증에 필요한 활동과 검증 작업들을 정의하고 있다.

표 1. Selected Test Item[5]

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

또, 고안전 소프트웨어 국제 표준인 IEC 61508[2]과 철도 시스템 분야의 국제 표준인 IEC62279[3] 에서도 소프트웨어 안전성 테스트 방법을 제안하고 있다. IEC 62279에서는 소프트웨어 안전 무결성 수준(Software Safety Integrity Level, SWSIL)을 정의하고, 정형화된 개발프로세스를 제안하고, 각 프로세스별로 SSIL 수준에 따라 채택할 수 있는 검증기법을 제시하고 있다. 5)에서는 이러한 국제 표준, 특히 IEC 61508과 IEC 62279을 분석하여, 고안전 소프트웨어의 평가항목을 도출하였다. 평가항목의 도출을 위하여 소프트웨어의 구현물이 작성되거나 혹은 작성된 이후에 검증되어지는 행위와 관련한 단계를 중심으로 크게 ST1~ST6까지의 총 6 단계로 구분하였다.

- 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

또 각 단계에서 수행하여야할 시험 기법들을 <표 1>과 같이 제안하였다[5].

이 기고에서는 기 제안된 열차제어시스템 고안전 소프트웨어 설계 방안과 부합하면서, 실제 도구 개발에 활용할 수 있는 요구 사항을 도출하기 위하여, 철도시스템 고안전 소프트웨어 검증 관련 국제 표준을 분석하는 방안을 제시하며, 제시된 방안에 따라 도출된 상세 항목들을 바탕으로 고안전 소프트웨어의 검증 도구를 개발하는 과정을 정적 분석 도구의 하나인 코딩규칙 테스팅 모듈의 사례를 들어 상세하게 제시한다.

Ⅲ. 정적 분석을 위한 코딩규칙 검사 도구 요구사항

IEC 61508은 표준에서 정의하고 있는 검증활동들에 대해 실제 수행하여야할 상세 활동들을 부록의 형태로 정의하고 있다[2]. 이러한 내용은 때로는 구체적인 항목을 포함하고 있고, 때로는 개념적인 내용을 포함하고 있어, 표준을 적용하는 사람이 여러 가지 선택 가능한 대안 중에서 적절히 선택하도록 허용하고 있다.

예를 들어, <그림 1>은 매우 구체적인 내용을 포함하고 있는 표준의 구절로부터 검증 규칙을 도출하는 과정을 도식화하였다.

C.2.6.6은 "Limited use of pointers" 에서는 "Aim: To avoid the problems caused by accessing data without first checking range and type of the pointer. To support modular testing and verification of software. To limit the

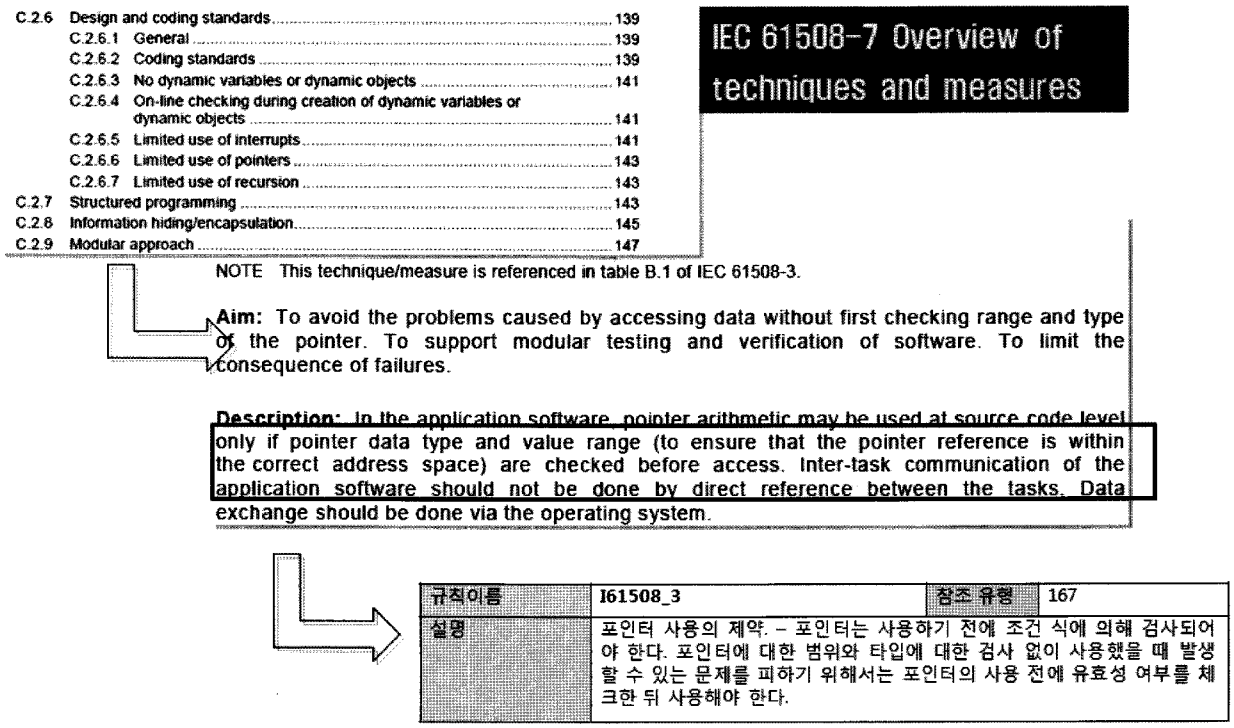

그림 1. IEC 61508-7 기반 정적 분석 규칙 도출 사례

consequence of failures.", "Description : In the application software, pointer arithmetic may be used at source code level only if pointer data type and value range (to ensure that the pointer reference is within the correct address space) are checked before access. Inter-task communication of the application software should not be done by direct reference between the tasks. Data exchange should be done via the operating system. "과 같이 정의하고 있다[2].

C.2.6.6를 바탕으로 다음과 같은 포인터 사용의 제약에 관한 규칙을 도출할 수 있다 - "포인터는 사용하기 전에 조건 식에 의해 검사되어야 한다. 포인터에 대한 범위와 타입에 대한 검사 없이 사용했을 때 발생할 수 있는 문제를 피하기 위해서는 포인터의 사용 전에 유효성 여부를 체크한 뒤 사용해야 한다."

반면에 C.2.6.4 의 경우, ""if allocation is not allowed, appropriate action must be taken."와 같이 선언적인 규정을 담고 있다. 이런 경우, 도구 개발을 위해서는 적절한 해석이 가미되어야하며, 본 개발 도구에서는 할당시, 할당 후 반환된 메모리 값에 대해 반드시 오류를 검사하고 처리하는 루틴을 제공하도록 규정하고, 이러한 루틴이 존재하는지 검증하는 것으로 정의하였다.

이와 같은 과정을 거쳐 도출된 정적 분석을 위한 코딩규칙들은 표 2와 같다.

또, IEC 62279의 10. Software design and implementation', '11. Software verification and testing', '14. Software assessment' 로부터 다음과 같은 규칙들이 추출되었다.

IV. 정적 분석을 위한 코딩규칙 검사 도구 구현

<그림 2>는 III 절에서 정의된 정적분석을 위한 코딩규칙을 자동 검사하기 위한 검사도구 구성도를 나타낸 것이다. 그림에서와 같이 검증 대상의 소스코드가 검증 자동화

표 2. IEC 61508 Supported Rules

61508 항목	규칙 내용(관련 문장)	규칙 상대 내용
C.2.6.3	금지함수("If dynamic variables or objects are not used, these faults are avoided.")	동적 메모리 변수/메모리 생성 함수 금지
C.2.6.4	메모리 반환("the whole memory which was allocated to it must be freed.")	메모리 할당/반환 함수 쌍 매치 여부 체크
	오류처리루틴("if allocation is not allowed, appropriate action must be taken.")	
C.2.6.6	포인터 제한적 사용("pointer arithmetic may be used at source code level only if pointer data type and value range are checked before access.")	
C.2.6.7	직, 간접적인 재귀호출 금지("Limited use of recursion")	
C.2.7	cyclomatic complexity 제한 ("keep the number of possible paths through a software module small, and the relation between the input and output parameters as simple as possible.")	지정 값 : 10
	특정 종류의 문장 사용금지("avoid complicated branching and, in particular, avoid unconditional jumps(goto) in higher level languages.")	
	for문의 초기화, 제어, 증감 expression 은 모두 loop 제어와 관련 있는 지 검사("where possible, relate loop constraints and branching to input parameters.")	
C.2.8	전역변수 정의금지("The key data structures are "hidden" and can only be manipulated through a defined set of access procedure. This allows the internal structures to be modified or further procedures to be added without affecting the functional behaviour of the remaining software")	
C.2.9	function size(LOC)제한("subprogram sizes should be restricted to some specified value, typically two to four screen sizes")	지정 값 : 100
	함수가 하나의 exit point를 가졌는지 검사("subprograms should have a single entry and a single exit only.")	
	미사용 인자 검사("any software module's interface should contain only those parameters necessary for its function.")	

표 3. IEC 62279 Supported Rules

61508 항목	규칙 내용(관련 문장)	규칙 상대 내용
표1.10 Design and Coding Standards	금지함수("3. No Dynamic Objects 4. No Dynamic Variables")	동적 메모리 변수/메모리 생성 함수 금지
	포인터 제한적 사용("5. Limited use of pointers")	
	직, 간접적인 재귀호출 금지("6. Limited use of recursion")	
	특정 종류의 문장 사용금지 ("7. No unconditional jumps")	goto 사용 금지
표9.10 Modular Approach	file size(LOC)제한("1. Module Size Limited")	지정 값 : 320
	전역 변수 사용 금지("2. Information Hiding/Encapsulation")	
	함수의 파라미터 개수 제한 ("3. Parameter Number Limit")	지정 값: 5
	함수가 하나의 exit point를 가졌는지 검사("4. One Entry/One Exit Point in Subroutines and Functions")	
B.61 Structured Programming	Cyclomatic Complexity 제한	지정 값: 10
	for문의 초기화, 제어, 증감 expression 은 모두 loop 제어와 관련 있는 지 검사	

도구로 입력되게 되면, 입력 소스코드를 분석하여 규정된 코딩규칙 검사기를 통해 입력된 대상 소스코드가 요구되는 코딩규격에 적합한지를 분석하여 리포트 생성기를 통해 결과를 제시하도록 설계하였다. 개발된 코딩규칙 검사 도구의 기능은 크게 코딩표준 관리 기능과 코딩표준 적용 기능으로 구분되어지며 이들 각각은 다시 두 가지의 하부

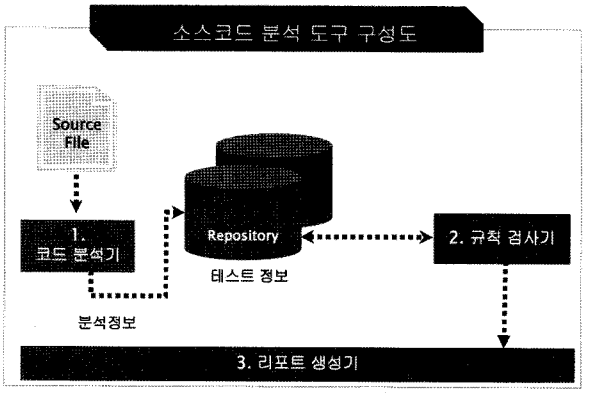

그림 2. 정적 분석 도구의 구조

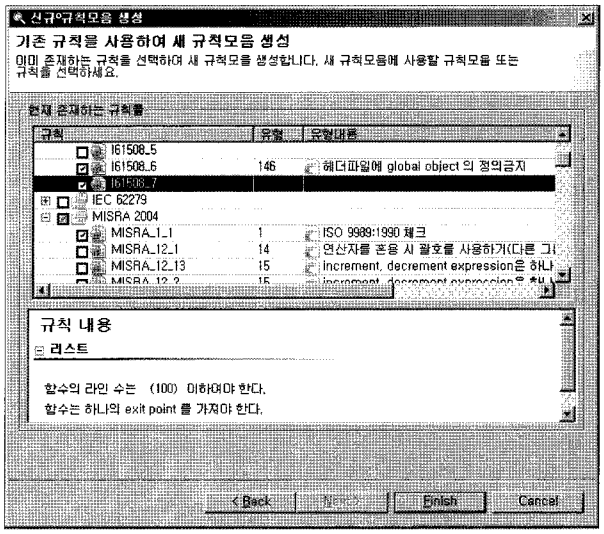

그림 3. Coding Rules Editing Windows

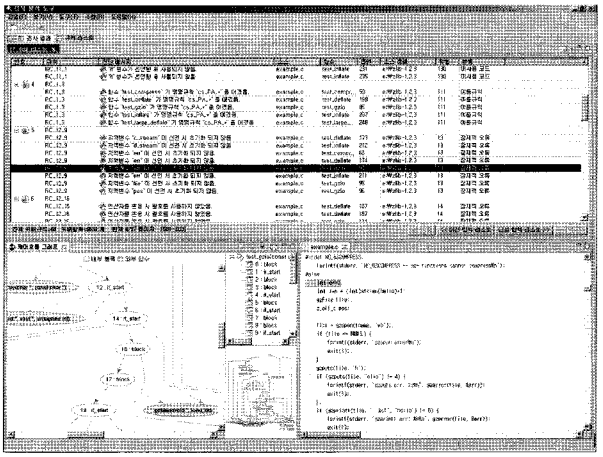

그림 4. Checking Results Windows

기능을 갖도록 개발하였다. 코딩 표준 적용을 위해서는, 검사대상 소스파일에 대해 정보 창에서 코딩규칙을 선택할 수 있도록 하였으며, 코딩 표준 적용 기능을 통해 검사를 진행하고 그 결과를 리포트 함으로써 편집한 규칙 내용을 적용할 수 있게 하였다. 그리고 코딩 표준 관리 기능을 통해 검사 대상 소스에 적용할 규칙 내용을 편집할 수 있다.

정적 분석 도구의 구현은 확장성을 고려하여 Eclipse RCP(Rich Client Platform)을 바탕으로 진행하였다. RCP 기반의 화면은 하나의 윈도우에 하나 이상의 뷰(View)가 포함될 수 있어, 검사규칙들과 각 규칙들에 대한 간략한 설명이 하나의 윈도우에 표현될 수 있고, 또한 검사결과도 다양한 측면에서 동시에 보여줄 수 있는 장점을 가진다. 또, 정적 분석의 결과를 전달하기 위해 소스 프로그램과 관련된 파일 이름, 라인 정보 외에도 제어 흐름 그래프 바탕의 정보를 제공함으로써 개발자가 정적 분석 결과를 좀 더 쉽게 이해 가능하도록 지원하였다.

또, 각 표준 관련 규칙들 역시, 아직 명문화된 코딩 규정이 없으므로 실제 적용 과정에서 적절히 선택할 수 있도록 <그림 3>과 같이 기존 규칙 리스트에서 일부 취사선택하여 새로운 규칙을 생성할 수 있도록 하였다. 즉, 코딩 규칙 설정 윈도우를 통해 시험대상이 되는 소스코드에 요구되는 코딩 규칙을 선택할 수 있도록 할 수 있고, 또한 기존에 DB화 되어있는 코딩 규칙 룰 셋을 이용해서 프로젝트별 새로운 룰 셋을 편집할 수 있도록 하였다. 또한 개발한 도구에서는 향후 규정화될지도 모르는 코딩 규칙을 위해 확장성을 고려하여 새로운 규칙을 입력할 수 있는 기능도 구현하였다.

<그림 4>는 개발한 검사모듈을 통해 임의의 소스코드를 대상으로 실제로 검증을 수행한 윈도우를 나타내고 있다. 그림의 상단 윈도우에 위반한 규칙 전체를 나타내고 있고, 그 중 하나를 선택할 경우 그림 우측하단의 윈도우처럼 위반한 소스코드 부분을 가리키도록 되어 있다. 그림의 좌측하단은 입력된 소스코드의 분석을 통한 소스코드의 제어흐름도를 나타낸 것으로 테스팅 대상 소스코드의 전체 구조를 확인할 수 있도록 하였다. 이러한 기능으로 인해 본

도구는 소프트웨어 디버깅 단계에서도 유용하게 활용될 수 있을 것으로 기대된다.

V. 결론

최근 들어 컴퓨터 기술의 발달에 따라 열차제어시스템과 같은 고안전 시스템들에서 소프트웨어에의 의존성이 급격하게 증가하고 있으며, 이러한 기술발전에 따라 소프트웨어에 높은 신뢰성과 안전성이 요구되고 있다. 본 기고에서는 이러한 고안전 소프트웨어의 검증을 위해 국제 표준에서 정의하고 있는 항목에 대한 검증을 지원하는 도구의 설계 방안을 정적분석을 위한 코딩규칙 테스팅 모듈의 예를 들어 제안하고 검증도구의 구현 결과를 제시하였다.

본 논문을 통해 제안한 정적분석 도구는 IEC 61508등에서 규정하고 있는 "Design & Coding Standard" 검사를 자동 지원함으로써 대상 소프트웨어에 요구되는 코딩규칙을 준수했는지 확인하는데 활용되어질 수 있을 것이다. 또 도구의 검증 항목을 사용자가 선택가능하도록 지원함으로써 추후 정적 분석 항목이 규정화될 때 이를 유연하게 지원할 수 있도록 하였다.

향후에는 개발된 도구를 실제 열차제어시스템 고안전성 소프트웨어에 적용하여 유용성을 검증하고, 개선점을 도출하는 연구를 수행할 계획이다. 이러한 적용성 검토 및 개선이 완료된 이후, 개발된 고안전 소프트웨어 검증 도구를 실제 규정화하고, 적용함으로써, 고안전 소프트웨어의 개발 및 검증 생산성이 향상되고, 결과적으로 고안전 시스템의 안전성 목표가 달성 될 수 있을 것이다.

♣ 참고 문헌

1. IEEE 1012(2004), "IEEE Standard for Software Verification and Validation".
2. IEC 61508(1998), "Railway Applications :The specification and demonstration of RAMS".
3. IEC 62279(2002), "Railway Applications : Software for railway control and protection systems".
4. RTCA(1992), "Software Considerations in Airborne Systems and Equipment Certification".
5. 황종규, 조현정, 김형신(2008), "열차제어시스템 소프트웨어 안전성 평가도구의 설계," 한국철도학회 논문집, 제11권 제2호, pp.139-144.
6. IEEE 1220?(2008), "IEEE Standard for Information Technology-Software Life Cycle Processes".