한국철도학회논문집
제12권 제1호 2009년 | pp. 81-87

열차제어시스템 소프트웨어 안전성 확인을 위한 코딩규칙 테스팅 자동화 도구의 개발

Development of the Design & Coding Standard of Railway Signaling Software as a Automatic Testing Tool

황종규† · 조현정*
Jong-gyu Hwang · Hyun-jeong Jo

Abstract Recent advances in computer technology have brought more dependency on software to railway signalling system. While much efforts have been reported to evaluate embedded software safety for railway signalling systems, not so much systematic approaches to evaluate software safety testing. In this paper, we propose a adaption of automatic software testing tool in terms of the design & coding standard for railway signalling system. The test items for the design & coding standard suggested in this study related international standards and MISRA-C. It is anticipated that it will be greatly helpful for the evaluation on the software for railway signalling system.

Keywords : Software Testing Tool, Design & Coding Standard, S/W Safety Assessment

요 지 최근의 컴퓨터 기술의 발달에 따라 열차제어시스템들이 컴퓨터 소프트웨어에 의존성이 더욱 증가되고 있으며, 이에 따라 이러한 열차제어시스템 소프트웨어의 안전성에 대한 테스팅이 더욱 중요하게 되었다. 본 논문에서는 이러한 열차제어시스템 소프트웨어를 위한 코딩규칙의 테스팅 자동화 도구의 적용을 제안하였으며, 코딩규칙 테스팅 자동화 도구의 구현 및 적용결과를 제시하였다. 구현 툴에서의 테스팅 항목은 철도시스템 소프트웨어 관련된 국제규격과 MISRA-C 기준을 참조하였다. 본 열차제어시스템 테스팅 자동화 도구는 열차제어시스템 소프트웨어 평가단계에서도 활용될 수 있고 또한 소프트웨어 개발단계에서도 유용하게 활용될 수 있을 것으로 예상된다.

주 요 어 : 소프트웨어 테스팅 자동화 도구, 설계 및 코딩 규칙, S/W 안전성 테스팅

1. 서 론

열차제어시스템은 최근의 전자 및 컴퓨터 기술의 발달에 따라 기존의 기계·전기적 장치로부터 컴퓨터시스템으로 전환되고 있다. 이에 따라 열차제어시스템은 컴퓨터 소프트웨어에의 의존성이 급격하게 증가하고 있다. 최근의 컴퓨터 기술의 발달에 따라 지능화 및 자동화를 위해 열차제어시스템 소프트웨어가 더욱 복잡해지게 되면서, 열차제어시스템에서 소프트웨어가 차지하는 비중이 더욱 증대되고 있다. 열차제어시스템 소프트웨어의 크기와 복잡도는 하드웨어의 발달 속도보다는 느리지만, 점차적으로 규모가 커지며, 복잡도도 증가할 것을 예상된다. 이에 따라 열차제어시스템처럼 임베디드화 된 바이탈 소프트웨어의 신뢰성과 안전성이 매우 중요한 문제로 대두되기 시작했다.

이처럼 열차제어시스템 소프트웨어의 안전성 이 중요해지고, 또한 IEC 61508[1]과 IEC 62279[2]에 의해 열차제어시스템 소프트웨어 안전 요구사항이 최근 들어 국제표준화 되었다. 이러한 철도 소프트웨어관련 국제 표준에서는 소프트웨어 안전성 확인을 위해 다양한 테스팅 기법의 적용을 요구하고 있으며, 특히 IEC 62279의 "A.4 소프트웨어 설계와 개발"에서 SWSIL (Software Safety Integrity Level) 3~4등급의 소프트웨어 설계 및 개발에 있어서 "Design & Coding Standards"를 반드시 준수하도록(M : Mandatory) 요구하고 있다.

† 책임저자 : 정회원, 한국철도기술연구원, 선임연구원, 공학박사
E-mail : jghwang@krri.re.kr
TEL : (031)460-5438 FAX : (031)460-5449
* 정회원, 한국철도기술연구원, 열차제어통신연구실, 주임연구원

철도이외의 산업분야에서는 대표적인 임베디드 시스템 중의 하나인 자동차 분야의 제어장치들의 코딩규칙 및 기준들이 "MISRA-C(Motor Industry Software Reliability Association-C)" 형태로 규격화되어 있다. 즉, 자동차 분야의 제어장치에서 C언어로 프로그래밍된 소프트웨어가 준수하여야 할 코딩규칙들이 정의되어 있다[3]. MISRA 코딩규칙처럼 자동차 분야는 국제적으로 공인된 코딩규정을 리스트화해서 적용하고 있지만, 자동차 분야 이외의 분야인 항공이나 철도분야 등에서는 아직 이와 같은 형태의 정형화된 코딩규칙을 정의하고 있지 않다. 따라서 현재 항공, 철도를 포함한 대부분의 산업용 소프트웨어는 자동차분야의 코딩규칙인 MISRA를 준용하여 사용하고 있다.

열차제어시스템을 포함한 철도시스템을 위한 소프트웨어에 필요한 안전 요구사항이 IEC 62279에 의해 규격화되어 있으며, 이 규격에서는 열차제어시스템과 같은 바이탈 소프트웨어의 안전성 확보를 위해 시스템 개발 수명주기 단계별 수행하여야 할 활동들을 설명하고 있다. 또한 이 규격에 의하면 열차제어시스템 시스템의 소프트웨어는 소스코드가 안전성 등급별 적절한 코딩규칙에 적합하도록 개발하고 또한 이를 확인하도록 되어있다[1,2]. 그리고 이 규격에서 철도 소프트웨어가 준수해야할 기본적인 요구사항들이 서술형으로 제시되어 있다.

국내뿐만 아니라 국외에서도 아직 열차제어시스템 소프트웨어 평가를 위한 코딩규칙 테스팅 자동화 도구는 없으며, 일반 산업용 임베디드 소프트웨어 코딩규칙 검사를 위한 자동화 도구가 일부 상용화되어 있다. 하지만 이러한 산업용 임베디드 소프트웨어 코딩규칙 검사 도구는 대부분 MISRA-C 코딩규칙만을 검사하거나, 일부 테스팅 도구 개발사에서 임의로 지정한 몇 가지 규칙들만 검사 가능하도록 되어 있다. 따라서 열차제어시스템과 같은 바이탈 소프트웨어를 위한 코딩규칙 테스팅 자동화도구는 아직 개발되고 있지 않다[4,6].

본 연구에서는 열차제어시스템 소프트웨어의 안전성 확인을 위해 관련 국제 표준에서 M 조건으로 요구하고 있는 코딩규칙의 적합여부 확인을 위한 테스팅 자동화 도구를 구현하였다. 이를 위해 산업계 임베디드 소프트웨어에 일반적으로 적용되고 있는 MISRA-C 코딩규칙과 철도시스템 소프트웨어 안전관련 표준인 IEC 61508-3과 IEC 62279 표준을 통해 열차제어시스템 소프트웨어가 준수하여야 할 코딩규칙을 도출하였으며, 이를 자동으로 테스팅 할 수 있는 도구를 구현하였다[5]. 본 연구를 통해 개발한 코딩규칙 테스팅 도구는 소프트웨어 평가 단계에서도 활용되어질 수도 있지만, 시스템 개발단계에서 소프트웨어 디버깅 단계에도 활용되어질 수 있을 것으로 기대한다. 이 논문은 열차제어시스템 소프트웨어 코딩규칙 테스팅 자동화 도구 개발의 결과를 정리한 논문으로, 논문의 구성은 다음과 같다.

2장에서는 제안하는 열차제어시스템 소프트웨어 테스팅 자동화 도구의 구조를 설명하고, 3장에서는 제안한 도구의 화면설계 내용을 정리하였다. 4장에서는 도구에서 구현한 코딩규칙을 설명하고 5장에서는 구현한 테스팅 자동화 도구를 설명하고, 6장에서 결론을 맺는다.

2. 코딩규칙 검사도구의 구조

본 연구에서 제안하고 구현한 코딩규칙 테스팅 자동화 도구를 "Design & Coding Standard 검사 도구"라 정의하였다. 이는 철도의 열차제어시스템 소프트웨어를 위한 설계 및 코딩 규정이 아직 없지만, 국외에서 일반적으로 적용되고 있는 자동차분야의 코딩규칙과 철도시스템 소프트웨어 안전관련 IEC 규격의 분석을 통해 소프트웨어 코딩규격을 도출하였으므로, 이와 같이 명명하여 제안 및 구현하였다.

"Design & Coding Standard 검사도구"는 열차제어시스템 소프트웨어가 IEC 61508과 IEC 62279와 같은 관련 규격 등에서 요구하는 코딩 규칙을 준수하는지 여부를 확인하여 요구되는 규칙에 어느 정도 적합한지를 판단하도록 할 수 있도록 하는 모듈로서, 기본적으로는 열차제어시스템 소프트웨어 평가단계에서 활용을 목적으로 하지만 소프트웨어 개발단계에서도 활용될 수 있도록 구조를 설계하였다. 본 절에서는 열차제어시스템 소스코드 분석 자동화 도구의 Design & Coding Standard 검사도구의 모듈의 구조를 기술한다.

이 모듈은 소스코드 자체에 대한 테스팅이 수행되게 된다. 또한 적용하는 코딩규칙은 다른 산업용 임베디드 시스템 소프트웨어에 일반적으로 적용되는 MISRA-C 코딩 규칙, 철도시스템 소프트웨어 RAMS 관련 규격인 IEC 61508-3, IEC62279를 통해 도출한 코딩규칙을 개발하고자 하는 테스팅 도구에 적용하였다. 그리고 추가적인 코딩규칙 등이 요구될 경우 등 개발하고자 하는 툴의 확장성을 위해 코딩규칙을 선택, 추가 등 편집할 수 있도록 하였다.

Fig. 1은 본 연구를 통해 제안한 열차제어시스템 소프트웨어를 위한 Design & Coding Standard 검사도구 구성도를 나타낸 것이다. 그림에서와 같이 테스팅 대상의 소스코드가 테스팅 자동화 도구로 입력되게 되면, 입력 소스코드를 분석하여 규정된 코딩규칙 검사기를 통해 입력된 대상 소스코드가 요구되는 코딩규격에 적합한지를 분석하여 리

포트 생성기를 통해 결과를 제시하도록 설계하였다.

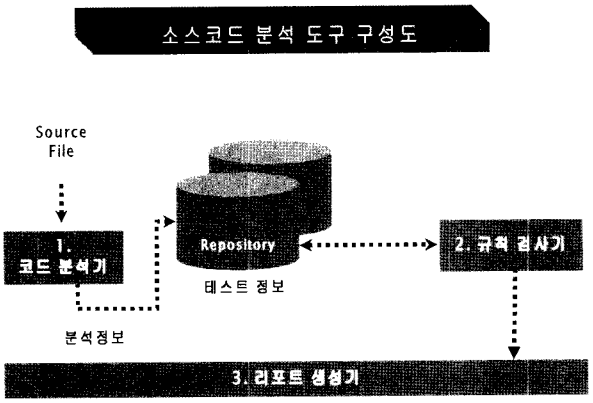

Fig. 1. Configuration of Design & Coding Standard Checking Module

Fig. 2는 제안하는 코딩규칙 검사도구의 기능 구성도를 나타낸 것으로, 크게 코딩표준 관리 기능과 코딩표준 적용 기능으로 구분되어지며 이들 각각은 다시 두 가지의 하부 기능을 갖도록 제안하였다. 즉, 아직까지 국내뿐만 아니라 해외에서도 열차제어시스템을 위한 규정화된 코딩규칙이 없기 때문에 테스팅에 적용할 코딩규칙을 조회하거나 편집할 수 있도록 확장성을 고려하여 기능구조를 설계하였다.

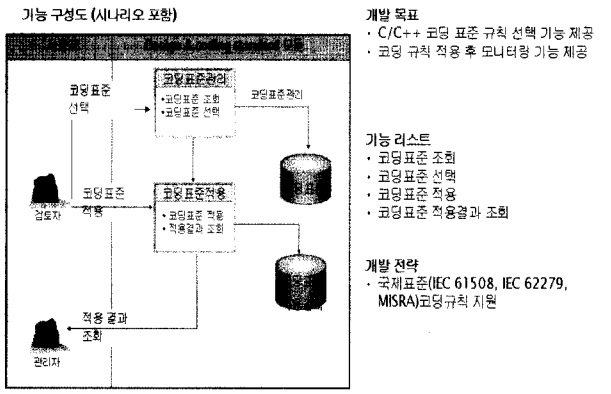

Fig. 2. Functional Architecture of Design & Coding Standard Checking Module

또한 제안하는 테스팅 자동화 도구는 크게 두 가지 방식으로 구성될 수 있도록 하였다. 첫 번째는 그림 3과 같이 클라이언트-서버 구조로서 품질 관리팀에서 설정한 규칙을 개발자들이 적용하는 방식이다. 이 구조에서는 개발자들이 자신의 규칙을 임의로 설정할 수 없으므로 개발 산출물의 품질을 일관되게 보장할 수 있는 장점이 있다. 두 번째는 별도의 서버를 두지 않고, 개발자가 규칙을 설정할 수 있는 지역(Local) 구조를 지원한다. 이 구조는 개발자가 개발을 진행하면서 개발 진행과 동시에 품질을 확인해 볼 수 있는 장점이 있다. 첫 번째 모듈은 소프트웨어의 평가단계에서 활용하기 적합한 구조이고, 두 번째 구조는 소프트웨어의 개발자가 활용하기 적합한 구조이다.

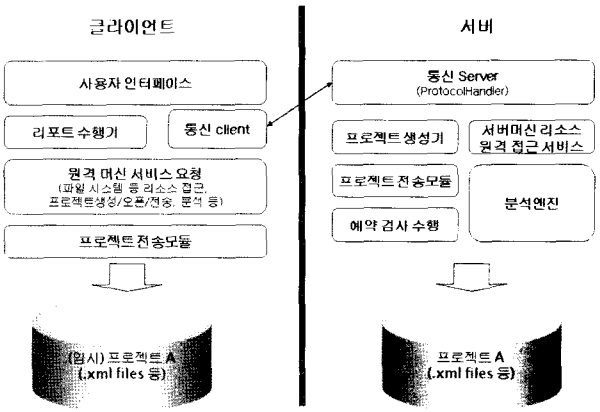

Fig. 3. Client-Server Architecture

3. 화면 설계

Design & Coding Standard 검사도구의 화면은 소프트웨어 평가자나 개발자가 사용하는 윈도우로서 앞 절에서 설명한 4가지 주요기능들이 GUI를 통해 구현되어져야 한다.

개발할 도구의 화면은 확장성을 고려하여 Eclipse RCP (Rich Client Platform)을 바탕으로 구성하였다. 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. 그림 4는 검증결과 설계 화면으로 프로젝트 뷰, 위배 리스트 뷰, 소스 뷰, 제어 흐름 그래프(CFG : Control Flow Graph) 뷰로 구성되도록 하였다.

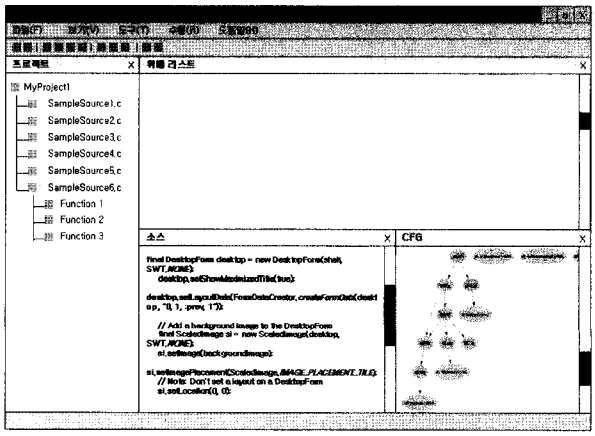

Fig. 4. Designed Checking Results Windows

Design & Coding Standard 검사 모듈의 사용자 인터페이스(UI : User Interface)는 Java언어로 작성되며, 그래픽 라이브러리는 스윙과 SWT를 혼합하여 사용하며, 클라이언트는 Microsoft Windows OS를 지원하도록 하였다. 사용자 인터페이스의 구동 계층은 Fig. 5와 같다.

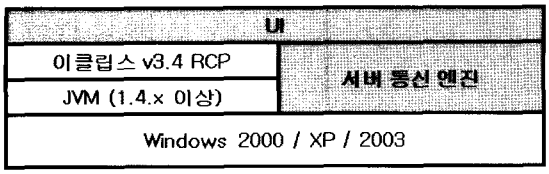

Fig. 5. Hierarchy of User API

개발하고자 하는 테스팅 도구는 검사대상 소스파일에 대해 정보 창에서 코딩규칙을 선택하여 적용할 수 있도록 하였다. 그리고 규칙 설정 기능을 통해 검사 대상 소스에 적용할 규칙 내용을 편집할 수 있으며, 검사 시작 기능을 통해 검사를 진행하고 그 결과를 리포트 함으로써 편집한 규칙 내용을 적용할 수 있게 한다. Table 1은 검사 결과를 출력하기 위한 항목을 나타낸 것으로서 표와 같이 적용 코딩 규칙에 위배되는 항목을 포함한 다양한 정보들을 출력하도록 설계하였으며, Fig. 6은 실제 구현하고자 하는 화면을 나타낸 것이다.

Table 1. Designed Coding Rules

항 목	내 용
규칙	위배한 코딩 표준
진단 메시지	위배 상세 메시지
소스 파일	규칙을 위배한 코드가 정의된 소스파일 위치
함수	규칙을 위배한 함수
라인	규칙을 위배한 소스상의 라인
유형	위배된 규칙의 유형
범주	위배된 규칙의 카테고리

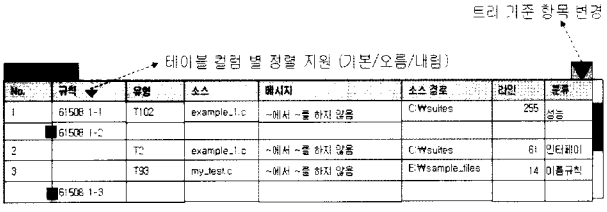

Fig. 6. Designed Coding Rules Applied Results Windows

4. 코딩규칙 설계

Design & Coding Standard 검사 도구는 크게 C 언어를 이용하여 내장형 소프트웨어를 개발할 때 준수하여야할 표준으로 자동차 업계에서 널리 사용되고 있는 MISRA-C 코딩 규정과 철도시스템 소프트웨어 개발 시 준수하여야할 국제 표준인 IEC61508-3과 IEC62279를 기본으로 하였다. IEC 61508-3과 IEC 62279의 경우 그 수준이 개념적, 포괄적이며 구체적인 설계 또는 코딩 표준을 정의하고 있지는 않고 있다. 따라서 본 연구에서는 관련 규격의 분석을 통해 코딩규칙을 도출 및 구현하였다.

4.1 MISRA-C

자동차 분야의 코딩규칙인 MISRA-C는 전체 141개의 코딩규칙을 제시하고 있다. 본 개발도구에서는 이러한 MISRA-C에서 제시하는 전체 코딩규칙 중에는 자동화 가능한 항목들에 대해 본 개발 도구에서 구현하였다. 표 2는 MISRA-C의 코딩규칙을 나타내고 있다.

Table 2. MISRA-C Supported Rules

MISRA-C 항목	규칙 내용
1.1	ISO 9989:1990 체크
2.1	assembly language 의 encapsulation 여부
2.2	C 주석 스타일만 허용
2.3	nested comment 사용금지(C style)
3.4	비표준 #pragma directive 사용 검사
4.1	허용 가능한 escape 문자
4.2	허용되지 않는 문자열 패턴
5.1	identifier 길이 제한
5.2	scope상에서 identifier 가리지 않기
5.3	typedef나 tag 이름이 unique identifier 인지 검사
...	...
15.1	switch label의 most closely-enclosing 문장이 switch 문인지 검사
15.2	switch가 하나 이상의 문장으로 이루어진 경우 break 가 있는지 검사
15.3	switch문에 default 존재 검사
15.4	switch 문의 조건식에 (in)equality 식 사용금지
15.5	switch 문은 적어도 하나의 case 문을 가졌는지 검사
...	...

4.2 IEC 61508 지원

IEC 61508은 바이탈 제어시스템 수명주기와 이에 따른 RAMS 요구사항을 정의하고 있으며, 전체 7파트로 구성되어 있다. 이 중에서, 파트 7이 시스템(HW/SW)을 측정하기 위한 방법을 다루고 있다. 파트 7의 구성은 다음과 같다.

Table 3. Configuration of IEC 61508 Part 7

목 차	비고
1. scope	
2. Normative reference	
3. Definitions and abbreviations	
Annex A Overview of techniques and measures for E/E/PES: control of random hardware failure	하드웨어 관점
Annex B (informative) Overview of techniques and measures for E/E/PES: avoidance of systematic failures	개념적인 설명 위주
Annex C (informative) Overview of techniques and measures for achieving software safety integrity	소프트웨어의 안전성 측정과 기술에 대한 설명

이중에서, 부록C에서 소프트웨어의 안정성 측정과 기술에 대한 설명들을 포함하고 있으며, 이를 바탕으로 다음과 같은 규칙들을 분석 및 도출하였다.

Table 4. IEC 61508 Supported Rules

61508 항목	규칙 내용(관련 문장)	규칙 상세 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.")	

Table 4. IEC 61508 Supported Rules(계속)

61508 항목	규칙 내용(관련 문장)	규칙 상세 내용
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.")	

4.3 IEC 62279 지원

IEC 62279는 17장으로 구성되며, 이중 '10. Software design and implementation', '11. Software verification and testing', '14. Software assessment'로부터 다음과 같은 규칙들이 추출되었다.

Table 5. IEC 62279 Supported Rules

62279 항목	규칙 내용(관련 문장)	규칙 상세 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	

5. 테스팅 자동화 도구의 개발

지금까지 설명한 열차제어시스템 소프트웨어를 위한 Design & Coding Standard 검사도구의 시스템 및 기능구조, 화면 설계 및 코딩규격 설계내용을 바탕으로 테스팅 자동화 도구를 구현하였다.

Fig. 7은 본 연구를 통해 개발한 검사모듈에서 신규규칙 생성 윈도우를 나타낸 것이다. 앞 절에서 설명하였듯이 아직 열차제어시스템 소프트웨어를 위한 코딩규칙이 규정되어 있지 않고 MISRA-C 규칙을 적용하거나, 아니면 IEC 61508이나 IEC 62279의 안전요구사항을 적용하도록 하고 있다.

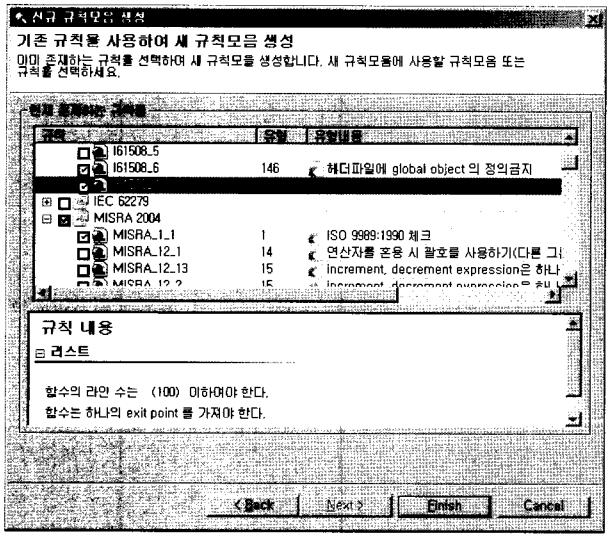

Fig. 7. Coding Rules Editing Windows

본 연구에서는 이에 따라 MISRA-C 코딩규칙과 IEC 61508, IEC 62279로부터 코딩규칙을 도출하여 구현하였다. 그리고 아직 명문화된 코딩규정이 없으므로 테스팅 과정에서 적절히 선택할 수 있도록 Fig. 7과 같이 기존 규칙 리스트에서 일부 취사선택하여 새로운 규칙을 생성할 수 있도록 하였다. 즉, 코딩규칙 설정 윈도우를 통해 시험대상이 되는 소스코드에 요구되는 코딩규칙을 선택할 수 있도록 할 수 있고, 또한 기존에 DB화 되어있는 코딩규칙 룰셋을 이용해서 프로젝트별 새로운 룰 셋을 편집할 수 있도록 하였다. 또한 개발한 도구에서는 향후 규정화될지도 모르는 코딩규칙을 위해 확장성을 고려하여 새로운 규칙을 입력할 수 있는 기능도 구현하였다.

Fig. 8은 개발한 검사모듈을 통해 임의의 소스코드를 대상으로 실제로 테스팅을 수행한 윈도우를 나타내고 있다. 그림의 상단 윈도우에 위반한 규칙 전체를 나타내고 있고, 그 중 하나를 선택할 경우 그림 우측하단의 윈도우처럼 위

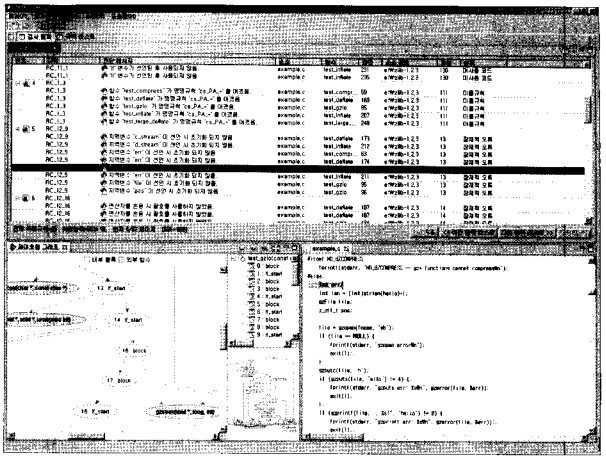

Fig. 8. Checking Results Windows

반한 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.

또한 필요시 Fig. 9와 같은 문서형태로 검사결과를 출력할 수 있도록 하였다. 즉, 최종적으로 시험대상 소스코드에 대한 테스트 결과가 종합적으로 표시되도록 하여 테스트 결과를 전체적으로 파악할 수 있도록 하고 있다. 즉, 입력 소스코드에 대해 요구되는 코딩규칙을 위반한 소스 부분을 표시하고 어떤 규칙을 위반했는지에 대해 리포팅 하도록 구현되어 있다.

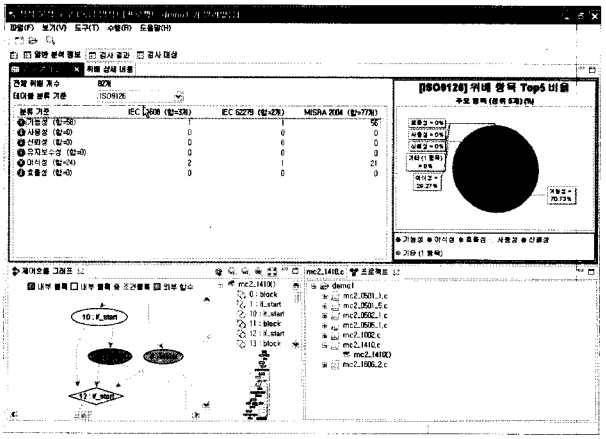

Fig. 9. Summarized Results Windows

6. 결 론

최근 들어 컴퓨터 기술의 발달에 따라 열차제어시스템들이 컴퓨터 소프트웨어에의 의존성이 급격하게 증가하고 있

으며, 이러한 기술발전에 따라 열차제어시스템 소프트웨어에 높은 신뢰성과 안전성이 요구되고 있다. 열차제어시스템을 위한 소프트웨어 코딩규칙 테스팅 자동화 도구는 소프트웨어의 안전성 확보를 위해 그 필요성이 요구되고 있다. 본 논문에서는 이러한 열차제어시스템 소프트웨어의 코딩규칙에 따른 테스팅 자동화 도구의 개발 내용을 설명하였다. 본 논문을 통해 제시한 열차제어시스템 소프트웨어 Design & Coding Standard 검사 자동화 도구는 기본적으로 소프트웨어 평가단계에 평가자 입장에서는 대상 소프트웨어에 요구되는 코딩규칙을 준수했는지 확인하는데 활용되어질 수 있을 것이다. 소프트웨어 개발과정에서 활용되어질 수도 있다. 즉, 소프트웨어 개발단계에서 디버깅 과정에 본 자동화 도구를 이용할 경우 코딩규격에 적합한 소스코드를 개발할 수 있고, 이를 통해 보다 높은 안전성을 갖는 소프트웨어가 확보될 수 있을 것으로 기대된다.

감사의 글

본 논문은 국토해양부가 출연하고 한국건설교통평가원에서 위탁 시행한 철도종합안전기술개발사업의 결과입니다.

참고문헌

1. IEC 61508, "Railway Applications: The specification and demonstration of RAMS", 1998.
2. IEC 62279, "Railway Applications : Software for railway control and protection systems", 2002.
3. MISRA-C Coding Standard, MISRA(Motor Industry Software Reliability Association), 2004.
4. Robyn R. Lutz and Robert M. Woodhouse, "Bi-directional Analysis for Certification of Safety-Critical Software", Proceedings of 1st International Software Assurance Certification Conference, Dulles, Virginia, February, 1999.
5. 황종규, 조현정, 김형신, "열차제어시스템 소프트웨어 안전성 평가도구의 설계", 한국철도학회 논문집, 제11권 제2호, pp.139-144, 2008. 4.
6. M. Fewstar, D. Graham, "Software Testing Automation: Effective use of test execution tools", ACM Press, Addison Wesley, 1999.

접수일(2008년 7월 7일), 수정일(2009년 1월 9일),
게재확정일(2009년 1월 20일)