한국철도학회논문집
제12권 제4호 2009년 | pp. 450-456

열차제어시스템 소프트웨어 Metrics 분석 자동화 도구 개발

Development of Automatic Tool for Software Metrics Analysis for Railway Signaling System

황종규† · 조현정* · 김용규**
Jong-gyu Hwang · Hyun-jeong Jo · Yong-kyu Kim

Abstract In accordance with the development of recent computer technology, the dependency of railway signaling system on the computer software is being increased further, and accordingly, the testing for the safety and reliability of railway signaling system software became more important. This thesis suggested an automated analysis tool for S/W metrics on this railway signaling system, and presented its result of implementation. The analysis items in the implemented tool had referred to the international standards in relation to the software for railway system, such as IEC61508 and IEC 62279. This automated analysis tool for railway signaling system can be utilized at the assessment stage for railway signaling system software also, and it is anticipated that it can be utilized usefully at the software development stage also.

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 안전성과 신뢰성에 대한 테스팅이 더욱 중요하게 되었다. 본 논문에서는 이러한 열차제어시스템 소프트웨어를 위한 Metrics 분석 자동화 도구를 제안하였으며, 또한 구현결과를 제시하였다. S/W Metrics는 철도시스템 소프트웨어관련 국제표준에서 언급되고 있는 Metrics를 대상으로 하였으며, 철도 소프트웨어 소스코드의 Metrics를 자동으로 분석하여 그 결과를 사용자에게 다양한 형태로 제시할 수 있도록 구현하였다. 자동화 도구는 열차제어시스템 소프트웨어 평가단계에서도 활용될 수 있고 또한 소프트웨어 개발단계에서도 유용하게 활용될 수 있을 것으로 예상된다.

주 요 어 : 소프트웨어 테스팅 자동화 도구, S/W Metrics 분석, S/W 안전성 검증

1. 서 론

열차제어시스템은 최근 기존의 기계적 장치로부터 컴퓨터시스템으로 전환되고 있으며, 소프트웨어에의 의존성이 급격하게 증가하고 있다. 최근의 컴퓨터 기술의 발달에 따라 지능화 및 자동화를 위해 열차제어시스템의 소프트웨어가 더욱 복잡해지게 되면서, 시스템에서 소프트웨어가 차지하는 비중이 더욱 증대되고 있다. 열차제어시스템 소프트웨어의 크기와 복잡도는 하드웨어의 발달 속도보다는 느리지만, 점차적으로 규모가 커지며, 복잡도도 증가할 것을 예상된다. 이에 따라 임베디드화 된 철도시스템 소프트웨어의 신뢰성과 안전성을 검증하는 것이 중요한 문제로 대두되기 시작했다.

철도시스템 소프트웨어 안전성 요구사항들이 최근 들어 IEC 61508[1]과 IEC 62279[2]에 의해 국제표준화 되었고, 또한 국내에서도 철도안전법이 제정되어 이러한 철도시스템 관련 국제표준에서 요구하는 각종 테스팅 및 검증활동을 요구하기 시작했다. 하지만 국내에서는 철도 소프트웨어에 대해 국제 표준에 따른 테스팅이나 검증을 위한 기준이나 이에 부합하는 기술에 대한 연구가 이제 막 시작되고 있다[3].

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

열차제어시스템 소프트웨어의 안전성은 주로 소프트웨어의 개발초기 단계인 소프트웨어 설계 단계에서의 안전성 활동을 통해 이루어진다. 이처럼 소프트웨어의 개발초기 단계에서의 안전성 활동에는 다양한 기법들이 적용되고 있으나, 소프트웨어의 개발이 완료된 이후 검증은 개발과정에 대한 문서의 검증에 주로 의존하며 일부의 경우 테스팅에 의한 정량적인 분석이 이루어지게 된다[3-5].

철도시스템 소프트웨어관련 국제표준에서는 소프트웨어 안전성 평가를 위해서 개발과정의 안전성 활동에 대한 문서의 검증뿐만 아니라 소프트웨어 테스팅을 통한 정량적인 분석 및 측정도 요구하고 있으며, 특히 대부분 SIL(Safety Integrity Level) 등급이 3 또는 4로 분류되는 ATP나 전자연동장치와 같은 바이탈한 열차제어시스템 소프트웨어의 경우 소프트웨어 Metrics에 대한 분석이 관련 국제 표준에서 안전성 검증을 위한 'M : Mandatory' 조건이며 ATO나 TCMS 같은 SIL 1 또는 2 등급으로 분류되는 설비는 'HR: Highly Recommend' 조건으로 규정되어 있다[1,2]. 이러한 소프트웨어 Metrics의 분석은 문서의 검증이나 정성적인 검증 방법을 통해 이루어질 수는 없고 자동화된 도구에 의한 정량적인 분석을 통해서만 가능하다.

열차제어시스템 소프트웨어의 Metrics 분석을 지원하는 도구는 국내에 아직 개발된 적이 없으며, QAC나 Poly-Space 같은 일부 외국 제품들이 상용화되어 있다. 본 논문에서는 이러한 열차제어시스템 소프트웨어 안전성 확인을 위한 방법의 하나인 Metrics 분석 지원을 위한 자동화 도구를 설계 및 개발하였다. 본 논문은 2장에서는 열차제어시스템 소프트웨어 테스팅 개요를, 3장은 소프트웨어 Metrics 분석 모듈의 설계 내용, 4장은 3장의 설계 내용을 바탕으로 한 자동화 도구의 구현 결과를 설명하고 마지막으로 5장에서 결론을 설명하는 것으로 구성한다.

2. 열차제어시스템 소프트웨어 테스팅

열차제어시스템 임베디드 소프트웨어는 개발초기부터 테스팅 과정을 통해 버그를 확인하여 품질비용을 낮출 수 있으며, 또한 개발 완료 후 검증과정에서도 소프트웨어 테스팅 과정이 필수적으로 요구되고 있다. 열차제어시스템 소프트웨어는 하드웨어에 의존적이고 높은 안전성이 요구되는 바이탈 소프트웨어여서 자동화된 도구를 통한 테스팅이 필요하지만 아직 이의 지원을 위한 도구들에 대한 연구가 많이 진행되고 있지 않고 있으며, 산업용 임베디드 시스템의 소프트웨어를 대상으로 일부 항목에 대한 테스팅 자동화 도구들이 소개되고 있다.

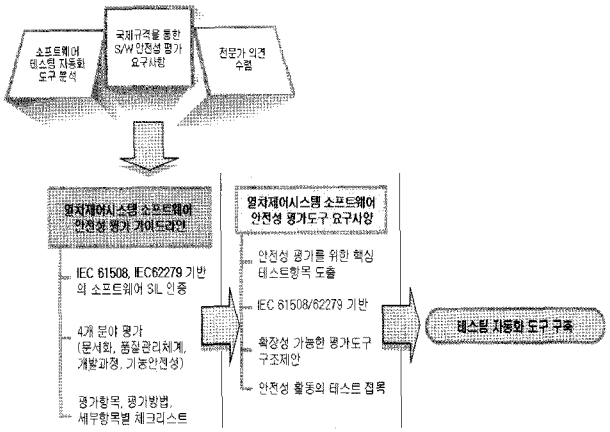

Fig. 1. Overview of automatic testing tool for railway signaling software

IEC 61508과 IEC 62279 같은 국제표준들에서 열차제어시스템 소프트웨어에 관련된 안전관련 사항들이 요구되고 있지만, 일반 산업용 제어시스템을 위한 테스팅 도구들은 철도 관련 국제 표준의 요구사항을 일부 만족시키지 못하고 있다[3]. 본 논문에서의 열차제어시스템 소프트웨어 테스팅을 자동화 도구를 개발하기 위해 Fig. 1처럼 철도시스템 소프트웨어의 안전관련 국제 표준의 분석을 통해 열차제어시스템 소프트웨어 안전성 평가를 가이드라인을 작성하였으며, 이 가이드라인에서의 테스팅 항목 중 반드시 자동화를 하여야할 아이템들을 분석하였다.

이처럼 테스팅을 자동화하여 정성적인 분석이 필요한 아이템 중 1장에서와 같이 소프트웨어 Metrics 분석을 위한 지원도구가 안전성 확인을 위해 반드시 필요하지만 아직 국내에서 이의 지원을 위한 도구가 개발되어 있지 않음을 확인하였다. 국제 표준에서 요구하는 바이탈 열차제어시스템 소프트웨어 Metrics 분석은 소프트웨어의 구조적 특성을 평가하고, 복잡도 같은 특성들이 요구되는 값을 만족하는지를 확인하기 위한 수단으로 활용되어진다.

소프트웨어 Metrics은 일반적으로 소프트웨어의 크기나 라인수로 분석되어진다. 하지만 이러한 피상적인 소프트웨어의 크기나 라인수로는 바이탈 소프트웨어의 특성 분석 및 평가가 불가능 하므로 국제규격에서는 특성을 보다 잘 나타낼 수 있는 소프트웨어의 복잡도(Complexity)나 응집도(Degree of Cohesion) 같은 소프트웨어공학(S/W Engineering) 적인 Metrics들이 요구되어진다. 본 논문에서 열차제어시스템 전용 소프트웨어 Metrics 분석 도구는 기본적인 라인수나 크기를 포함하면서 동시에 복잡도나 응집도를 자동으로 분석할 수 있는 도구를 설계 및 구현하였다. 이러한 국제 표준 및 소프트웨어공학에서 요구하는 보다 체계적인 소프트웨어 특성 분석이 가능한 열차제어시스템 전용 Metrics 분석 자동화 도구의 개발에 있어서 소프트웨

어의 검증단계에서 뿐만 아니라 소프트웨어의 개발단계에서도 활용될 수 있도록 분석결과를 다양한 형태로 출력되도록 하고자 하였다.

3. S/W Metrics 분석 모듈의 설계

3.1 소프트웨어 Metrics 분석

열차제어시스템과 같은 바이탈 소프트웨어의 Metrics를 분석하는 목적은 소프트웨어 개발의 각 단계에서 필요한 소프트웨어의 특성을 개발과정 또는 테스트 이력으로부터 습득하는 것이 아니라, 개발된 소프트웨어 자체의 구조적인 특징으로부터 분석 및 검증하기 위한 방법이다. 이러한 소프트웨어 Metrics 분석 기능은 소프트웨어의 구조적인 특성을 평가하여, 복잡도와 같은 특성들이 요구되는 값을 만족하는지 확인 및 검증하는 과정이 포함되며, 이런 결과물들은 소프트웨어의 효율적인 테스팅, 추가, 변경 작업에 기초자료로 사용되어질 수 있다.

본 연구를 통해 설계 및 구현한 열차제어시스템을 위한 Metrics는 소프트웨어 크기, 복잡도, 응집도 그리고 결합도(the Degree of Coupling)를 그 대상으로 하였다. 여기에서 복잡도나 응집도, 결합도를 분석 및 측정하기 위해서는 소프트웨어의 함수간 구성도 및 흐름도에 대한 분석이 이루어져야 하며, 이러한 소스코드의 분석을 바탕으로 이와 같은 소프트웨어공학적인 Metrics들에 대한 분석이 가능하게 된다. 이들 각 Metrics들에 대한 설명은 다음 절에 자세히 설명한다.

이러한 소프트웨어공학적인 Metrics들의 분석 및 측정은 소스코드의 양이 늘어남에 따라 소프트웨어 평가자가 매뉴얼로 작성하기 거의 불가능하다. 따라서 입력되는 소스코드에 대한 분석을 통해 함수들의 구성 및 흐름도를 자동으로 분석하여 이를 통해 다양한 형태의 Metrics들을 분석하고 표시할 수 있는 자동화된 도구를 구현한다.

3.2 Metric 분석 모듈의 설계

본 절에서는 앞에서 설명한 철도시스템 전용의 다양한 소프트웨어 Metrics들을 분석할 수 있는 모듈의 설계 내용을 Metrics 항목별로 설명한다. Metrics 분석 모듈은 입력되는 소스코드의 제어흐름 등의 분석을 바탕으로 Metrics들을 분석하고 사용자에게 분석결과를 제시하여 소스코드의 상태를 쉽게 파악할 수 있도록 그래픽 형태와 텍스트 형태로 다양하게 표시하도록 하였다.

설계 및 구현한 소프트웨어 Metrics 분석 모듈의 기능 구성도는 Fig. 2와 같으며, 주요한 기능은 표 1과 같이 정리하였다. 이들의 그림과 표에서와 같이 분석 및 측정 가능한 소프트웨어 Metrics들은 복잡도, 응집도, 결합도, 소프트웨어 크기, 인터페이스 형태, 모듈의 크기 측정의 6가지를 기본 Metrics로 선정하여 설계 및 구현하였다. 또한 본 자동화 도구가 소프트웨어 검증 단계 및 개발 단계에서도 활용이 가능하도록 하기 위해 다양한 형태로 출력하기 위해 구현 도구 내부에 별도의 분석결과 리포트 생성 모듈을 구현하였다. 즉, 그림에서와 같이 설계 및 구현한 자동화 도구는 Metrics 분석 모듈과 분석된 결과를 저장하고 표시하기 위한 모듈로 구성된다.

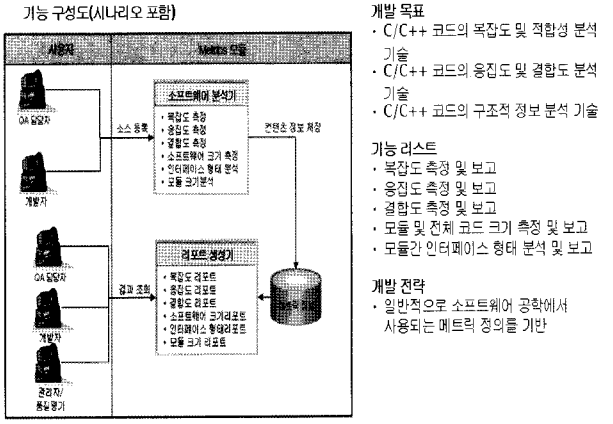

Fig. 2. Functional architecture for S/W Metrics Analysis Module

Table 1. Function Lists of Metrics Module

기능명	기능설명
복잡도 측정	등록된 소스를 기반으로 모듈별 또는 함수별 복잡도를 측정한다.
복잡도 Metrics 리포트	측정된 복잡도를 리포트로 보여준다.
응집도 측정	등록된 소스를 기반으로 모듈간 또는 함수간 응집도를 측정한다.
응집도 Metrics 리포트	측정된 응집도를 리포트로 보여준다.
결합도 측정	등록된 소스를 기반으로 모듈간 또는 함수간 결합도를 측정한다.
결합도 Metrics 리포트	측정된 결합도를 리포트로 보여준다.
소프트웨어 크기 측정	등록된 소프트웨어의 크기를 측정한다.
소프트웨어 크기 리포트	측정된 소프트웨어 크기를 리포트로 보여준다.
인터페이스 형태 분석	등록된 소스를 기반으로 모듈별 또는 함수별 인터페이스 형태 분석한다.
인터페이스 형태 리포트	분석된 인터페이스 형태를 리포트로 보여준다.
모듈 크기 측정	등록된 소스를 기반으로 모듈 크기를 측정한다.
모듈 크기 리포트	측정된 모듈 크기를 리포트로 보여준다.

3.2.1 복잡도 분석

소프트웨어에 대한 복잡도를 분석하고 측정하는 많은 방법들이 소프트웨어공학 분야에서 연구 및 제시되고 있다. 이러한 많은 방법들 중 임베디드 소프트웨어의 복잡도는 McCabe 복잡도와 Cyclomatic 복잡도에 의해 분석 및 측정하는 것이 일반적인 방법이다[6]. 이들 두 가지의 복잡도 Metrics는 Fig. 3을 기준으로 해서 다음과 같은 수식으로 표현되어진다.

- *McCabe 복잡도-1 = 전체 간선의 수- 전체 노드의 수+2*
- *McCabe 복잡도-2 = 영역의 수+1*
- *Cyclomatic 복잡도 = 전체 간선의 수-(전체 노드의 수-종료 노드의 수)+1*

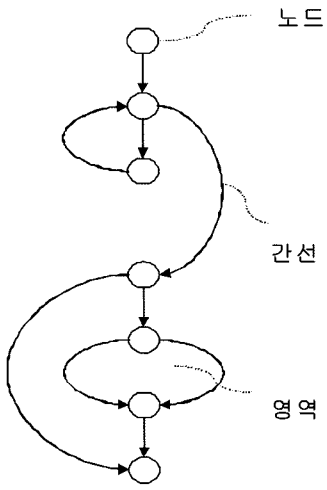

Fig. 3. Diagram for S/W Complexity Analysis

위에서 설명한 세 가지의 소프트웨어 복잡도 Metrics는 약간의 차이는 있으나 Fig. 3과 같이 소스코드의 제어흐름도 분석을 바탕으로 측정을 하는 것으로 기본적으로는 차이점이 거의 없다.

이들 세 가지의 복잡도 Metrics 모두 함수의 조건문이 많아질수록 간선이 많아져서 전체적인 복잡도가 높아지게 된다. 본 논문에서는 세 번째 방법인 Cyclomatic 복잡도를 기본으로 하여 소스코드의 복잡도 분석하는 기능을 구현하였다. 이 Metrics에서의 측정결과는 다음과 같이 복잡도의 값에 따라 진단 메시지를 사용자에게 제공하도록 하였다.

Table 2. Range of S/W Complexity

복잡도 값 범위	설명
1~0	단순함 (Simple Program)
11~20	약간 복잡함 (More Complex)
21~50	복잡함 (Complex)
51 이상	매우 복잡함 (High Risk)

3.2.2 응집도 분석

소프트웨어의 평가 Metrics에 있어서 앞에서 설명한 복잡도와 함께 중요한 Metrics 중의 하나가 응집도이다. 응집도는 소스코드 내에서 함수들과 변수들이 얼마만큼 독립적으로 작성되어 있는지를 분석하는 항목으로 앞에서 설명한 복잡도에서는 제어 흐름도를 바탕으로 측정한 것과는 달리 함수들과 이들 함수들이 사용하는 변수들과의 상관관계를 측정하는 Metrics이다.

열차제어시스템과 같은 바이탈 제어시스템 소프트웨어의 소스코드는 높은 응집도를 가져야 한다는 것은 일반적으로 통용되는 개념이다. 응집도를 측정하는 방법은 여러 가지가 있지만, 본 논문에서는 일반적으로 널리 사용되는 LCOM4 (Lack of COhesion in Methods) Metrics를 기반으로 분석하도록 하였다[7]. LCOM4는 객체지향 기법(OOP: Objected Oriented Programming) 소프트웨어에 대해 정의된 Metrics로써 한 클래스에서 각 메소드들이 얼마나 상관관계를 가지고 있는가를 평가하는 방법이다. 즉, 한 모듈(클래스) 내의 메소드간의 상호 연관성을 의미하며 동일한 데이터를 사용하는 메소드들의 집합(Connected components) 의 개수로서 정의한다. 모듈 내의 함수들이 동일한 데이터에 대해 연산할 때 그것들을 하나의 그룹으로 묶게 되는데, 그 이유는 그것들 간에는 관계가 강하게 놓여져 있기 때문이다. 어떤 모듈의 그 그룹개수가 1이면 가장 높은 응집도를 갖는 경우이며, 그보다 커질수록 응집도가 낮아진다고 본다.

Fig. 4는 본 논문에서 적용한 응집도를 분석하는 방법인 LCOM4를 나타낸 것이다. Fig. 4의 좌측 그래프를 보면 A함수가 B함수를 호출하고 B함수가 x변수를 사용하고 있는데, 이것이 하나의 Connected component 가 된다. C함수와 D함수가 y 변수를 사용하고 있는데 이 또한 하나의 Connected component가 된다. 결국 좌측 그림은 두 개의 Connected component가 존재함으로써 각각 독립적으로 나뉘어질 수 있는 형태가 되며 이 모듈의 응집도는 2로 나타낸다. 반대로, 우측 그림은 응집도가 1인 모습을 나타내고 있다. 좌측 그림에서 C가 x변수를 사용함에 따라 전체가 하나의 그룹으로 묶여졌기 때문이다.

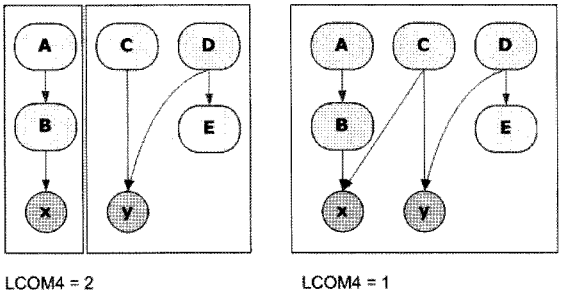

Fig. 4. Generals of LCOM4

이처럼 LCOM4는 본래 OOP에 적합한 소프트웨어 Metrics이지만 C 언어 프로그램에도 적용할 수 있도록 다음과 같이 재 정의하도록 한다. 하나의 C 소스파일 내에서 정의된 변수들(include한 헤더파일 내 구조체 등 변수 포함)을 사용하는 함수들을 함께 하나의 Connected component로 규정하며, LCOM4-c로 정의하여 사용하였다. 즉, 설계한 자동 분석도구에서의 응집도는 소스파일 단위로 응집도를 측정하도록 하였다.

3.2.3 결합도(Degree of Coupling) 분석

앞 절에서 설명한 응집도와는 달리 결합도는 낮을수록 보다 좋은 소프트웨어 모델이다. 왜냐하면, 결합도는 하나의 모듈에서 다른 모듈로 접근하는 정도를 측정하는 것으로 정의하기 때문이다. 하나의 모듈은 다른 모듈에 의해 접근되는 잘 정의된 인터페이스를 가져야 한다. 한 모듈의 여러 개 객체를 통해 동작되는 대신 하나의 인터페이스를 통해 접근되는 것이 더 나은 구조를 갖게 되고 결합도가 낮아지게 된다. 낮은 결합도는 복잡성을 줄이고 유지보수성 및 프로그램 가독성을 높인다. 코드가 복잡해져서 결합도가 높아지면 코드 변경사항에 대한 Side-effect 범위가 커지면서 결함발생 소지가 더 높아질 수 있다.

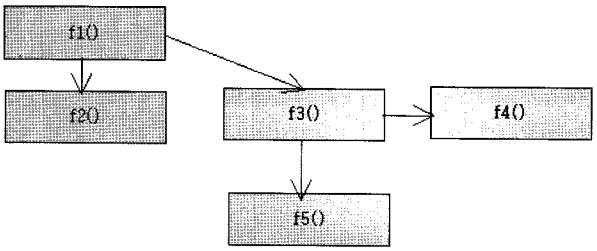

Fig. 5. Generals of Fan-In/Fan-Out

이 결합도를 분석하는 Metrics들이 여러 가지가 제안되고 있는데 대부분 OOP 프로그램에 적합한 성격을 지니고 있고, C 언어에는 적용이 어려운 Metrics들이다. 따라서, Safety-critical 소프트웨어의 개발에 많이 사용되는 C 언어를 위해서는, 결합도를 구조적 방법론에서 말하는 함수 각각의 Fan-In, Fan-Out 주요 Metrics로 분석 및 측정하도록 하였다[8].

- *Fan-In : 자신을 호출하는 모듈의 수 (caller)*
- *Fan-Out : 자신이 호출하는 모듈의 수 (callee)*

즉, 다음 그림에서 보면 f1()의 Fan-Out은 f2()와 f3()이 된다. f5()의 Fan-In은 f3()이 된다.

3.2.4 모듈 크기, 전체 코드 크기 측정

프로그램의 사이즈를 소스코드의 파일별, 함수별로 각각 분석 및 GUI를 통해 분석결과를 제공하도록 하였다.

3.2.5 모듈 간 인터페이스 형태 분석

모듈간 인터페이스 형태 분석 기능은 앞에서 설명한 복잡도나 응집도와 같은 Metrics 분석을 위해서는 소스코드의 제어흐름 그래프(Control Flow Graph)가 분석이 되어야 하므로, 이 분석된 결과를 사용자가 보다 시각적으로 이해할 수 있도록 비주얼하게 그래프로 형태로 표시하는 기능을 의미한다. 즉, 함수의 제어흐름 그래프에서 외부함수 호출관계를 나타냄으로써 그 함수의 의존성을 그래프 상에서 확인할 수 있다. 또한, 전체 함수간의 호출 관계를 분석하여 그래프로 제공하는데, 이를 통해 전체 함수들 간의 연관관계를 파악할 수 있도록 지원하도록 하였다.

4. Metrics 분석 자동화 도구의 개발

앞 장에서 설명한 열차제어시스템 소프트웨어를 위한 Metrics들을 자동으로 분석 및 측정할 수 있는 자동화 도구를 개발하였다.

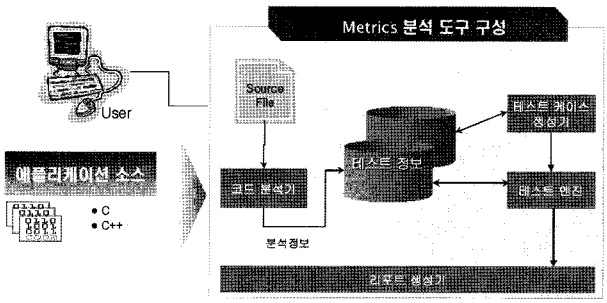

Fig. 6. Configuration of Metrics Analysis Tool

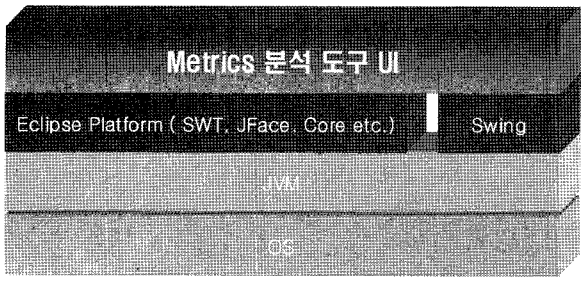

Fig. 7. Layer Architecture of Development Tool

Fig. 6은 개발한 Metrics 분석 자동화 도구의 구성을 나타낸 것으로 열차제어시스템 소프트웨어를 C나 C++ 언어의 소스코드로 개발한 도구에 입력하게 되면 입력되는 도구의 분석을 통해 제어흐름 및 함수의 분석을 통해 테스트 정보를 생성한다. 이를 바탕으로 사용자가 GUI를 통해 선택하는 Metrics 분석 요청에 다양한 형태의 Metrics 분석이

수행되게 된다. Fig. 7은 개발한 자동화 도구의 계층구조를 나타낸 것으로 소스코드 분석엔진인 Eclipse Platform과 사용자 인터페이스를 계층화시켜 개발하였다. 이는 향후 소프트웨어 테스팅을 위한 다른 기능들이 구현될 경우 본 논문의 도구와 통합하기 위해 각각 별개의 계층으로 구현하였다.

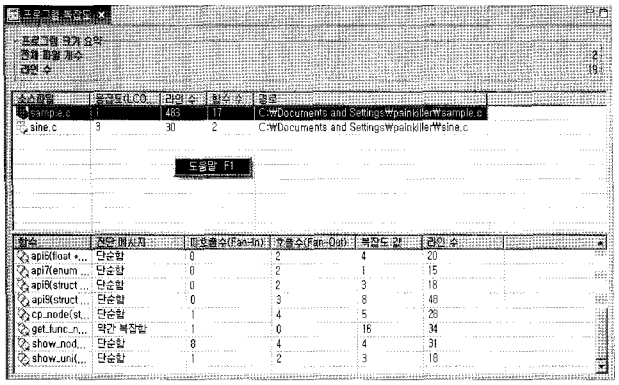

Fig. 8. Results of Complexity & Cohesion

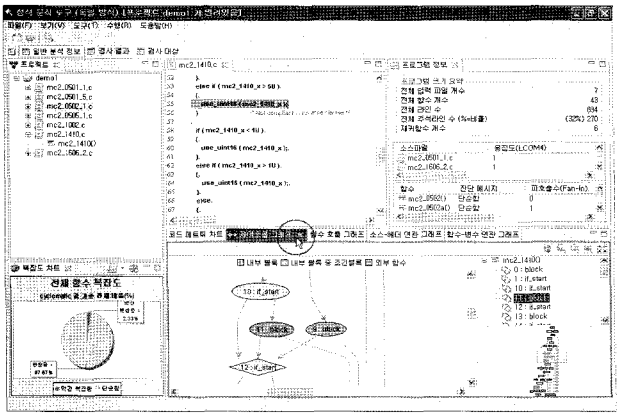

Fig. 9. Results of Complexity & Control Flow Graph

Fig. 8과 Fig. 9는 구현한 복잡도 Metrics와 응집도 분석 결과를 나타낸 윈도우이다. 그림에서와 같이 복잡도 Metrics에 대한 출력결과를 텍스트 형태나 그래픽 형태 등 다양하게 표시되도록 구현하였다. 그림에서와 같이 입력되는 소스코드를 분석하여 제어흐름 그래프를 그래픽 형태로 출력하고, 이를 바탕으로 복잡도나 응집도에 대해 측정하게 되며, 복잡도나 응집도는 소스파일별로 분석되어진다. Fig. 10은 응집도와 결합도의 결과 화면으로서 응집도는 Fig. 8과 같이 파일단위로 측정된 결과를 볼 수 있고 복잡도는 앞에서 설명한 바와 같이 Fan-In과 Fan-Out Metrics에 대해 함수별 측정되는 결과를 확인할 수 있다. Fig. 11은 복잡도나 응집도 같은 다양한 형태의 Metrics 분석이 가능하도록 하기 위해 입력된 소스코드의 제어흐름을 분석한 결과를 소스코드와 함께 그래픽 형태로 출력한 윈도우이다. Fig. 11과 같이 소스코드의 분석결과를 그래픽 형태와 소스코드 형태로 같이 제공하도록 하였다. 또한 전체 제어흐름 그래프에서 해당되는 소스코드가 어느 부분인지도 쉽게 확인할 수 있도록 GUI를 구성하여 본 개발 도구를 소프트웨어 검증위한 도구로서 뿐만 아니라 개발단계에서도 유용하게 활용될 수 있도록 하였다.

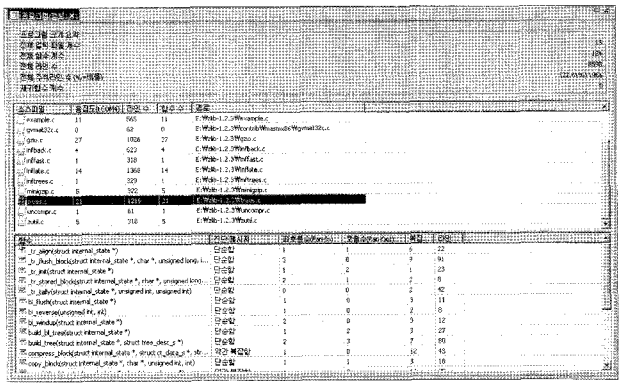

Fig. 10. Results of the Degree of Coupling & Cohesion

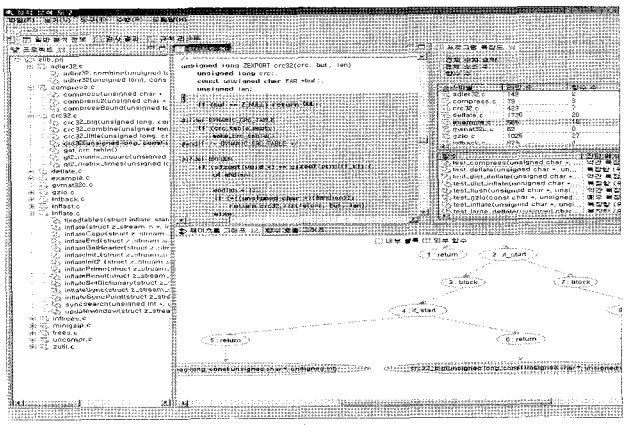

Fig. 11. Results of Control Flow Graph

5. 결 론

최근 들어 컴퓨터 기술의 발달에 따라 열차제어시스템들이 컴퓨터 소프트웨어에의 의존성이 급격하게 증가하고 있으며, 이러한 기술발전에 따라 소프트웨어에 높은 신뢰성과 안전성이 요구되고 있다. 열차제어시스템을 위한 소프트웨어 Metrics 분석 자동화 도구는 소프트웨어의 안전성 검증을 위해 필수적으로 준수해야 하는 관련된 국제표준에서 요구하는 항목들이다. 본 논문에서는 관련 국제 표준에서 요구하고 있는 열차제어시스템 소프트웨어 Metrics 분석 자동화 도구의 개발 내용을 설명하였고, 구현한 자동화 도구는 기본적으로 소프트웨어 검증 단계에 활용될 도구이며, 동시에 소프트웨어 개발과정에서 활용되어질 수도 있도록 화면을 구성하는 등 다양하게 활용될 수 있도록 개

발하였다. 즉, 소프트웨어 개발단계에서 디버깅 과정에 본 자동화 도구를 이용할 경우 적합한 소스코드를 개발할 수 있고, 이를 통해 보다 높은 안전성을 갖는 소프트웨어가 확보될 수 있을 것으로 기대된다.

감사의 글

본 논문은 국토해양부가 출연하고 한국건설교통평가원에서 위탁시행한 철도종합안전기술개발사업(철도안전C03)의 연구비에 의해 수행되었으며, 이에 감사드립니다.

참고문헌

1. IEC 61508(1998), "Railway Applications - The specification and demonstration of RAMS."
2. IEC 62279(2002), "Railway Applications - Software for railway control and protection systems."
3. 황종규, 조현정, 김형신(2008), "열차제어시스템 소프트웨어 안전성 평가도구의 설계," 한국철도학회 논문집, 제11권 제2호, pp.139-144.
4. Zage, W.M and Zage, D.M.(1993), "Evaluating design metrics on large-scale software," IEEE Trans. on Software Eng., Vol. 10, Issue 4, pp. 75-81.
5. Fewstar, M. and Graham, D.(1999), "Software Testing Automation: Effective use of test execution tools," ACM Press, Addison Wesley.
6. Wikipedia Doc., http://en.wikipedia.org/wiki/cycli matic_complexity
7. Wikipedia Doc., http://en.wikipedia.org/wiki/coupl ing _(computer_ science)
8. http://wwww.aivosto.com/project/help/pm-oo-cohe sion.htm

접수일(2008년 12월 17일), 수정일(2009년 1월 9일),
게재확정일(2009년 8월 3일)