2003 추계학술발표회 논문집
한국원자력학회

모델 기반 설계 도구를 이용한 계측제어계통 소프트웨어 생산 기법 개발

Development of the Software Generation Method using Model Driven Software Engineering Tool

장훈선, 정재천, 김재학, 한희환, 김도연, 장영우
Hoon-Seon Chang, Jae-Cheon Jung, Jae-Hack Kim
Hee-Hwan Han, Do-Yeon Kim, Young-Woo Chang
한국전력기술주식회사

문왕식
Wang Sik, Moon
넥스텍 주식회사

요약

UML(Unified Modeling Language) 기반의 모델링 도구를 사용하여 원전 계측제어시스템 소프트웨어용 소스 코드를 생성하고 통합된 시험 도구를 이용, 코드 감사(Audits) 및 모듈 시험을 수행하는 방법론에 대해 기술하였다. 객체 지향 언어의 개발을 위해 먼저 UML 1.4 에서 정한 모델링 규약 중에서 클래스 다이어그램을 이용하여 관련되는 변수 및 모듈을 객체화, 액티비티 다이어그램을 이용한 자료흐름도의 구성, 그리고 시퀀스 다이어그램을 이용한 알고리즘의 중요 로직을 자동으로 생성하였다. 완성된 소프트웨어 모듈은 시험 도구에 의해 순환 복잡도, 코드 및 패스 커버리지등을 시험하도록 정의되었다. 모델링 도구에 의해 개발된 소프트웨어는 원자력발전소 계측제어 소프트웨어 중 원자로보호계통의 가압기 저압력 트립 알고리즘을 대상으로 실시하였으며 표준 모델링 언어인 UML 을 사용하여 객관적인 모델의 구축 및 시스템의 기능구현에서 소스코드의 생산까지를 일관적으로 행할 수 있으며, 도구에서 제공하는 SRE(Simultaneous Round-trip Engineering)에 의하여 생산과정에서 발생하는 오류를 줄일 수 있었다. 시험도구에 의한 테스트는 소스코드의 감사 및 모듈 시험을 효율적으로 수행할 수 있었다.

Abstract

The methodologies to generate the automated software design specification and source code for the nuclear I&C systems software using model driven language is developed in this work. For qualitative analysis of the algorithm, the activity diagram is modeled and generated using Unified Modeling Language (UML), and then the sequence diagram is designed for automated source code generation. For validation of the generated code, the code audits and module test is performed using Test and QA tool. The code coverage and complexities of

example code are examined in this stage. The low pressure pressurizer reactor trip module of the Plant Protection System was programmed as subject for this task. The test result using the test tool shows that errors were easily detected from the generated source codes that have been generated using test tool. The accuracy of input/output processing by the execution modules was clearly identified.

1. 서 론

원전 계측제어계통의 디지털화에 따라 Computer Aided Software Engineering (CASE)와 같은 검증된 도구에 의한 소프트웨어의 효율적인 생산과 V&V 의 핵심기술 개발표준 방법론이 활발히 검토되고 있다. 이와 병행하여 체계적인 소프트웨어의 개발과 개발기간의 단축은 물론 소프트웨어의 품질향상, 생산성 향상, 재사용성 및 인적 오류 절감 등을 위하여 적극적인 CASE 도구의 도입 등이 추구되고 있다. [1]

소프트웨어 개발은 공식적으로 정의된 소프트웨어 수명주기(Software Life Cycle)에 따라 수행된다. 따라서 수명주기에 따라 생산되는 설계문서 또는 소프트웨어 제품들을 결정해야 하며, 이와 아울러 원자력발전소에 적용되는 규제요건 및 기술기준 등을 준수하여야 한다.

소프트웨어 공학도구를 크게 분류하면 소프트웨어 개발도구와 소프트웨어 확인도구로 분류할 수 있다. 소프트웨어 개발도구는 그 출력물이 프로그램의 일부로 변환되므로 오류에 직접 영향을 줄 수 있다. 반면 소프트웨어 확인 도구는 오류를 확인하는 것이 목적이므로 오류를 직접 생성하지는 않지만 오류 발견을 실패할 수 있다.

산업기술기준 7-4.3.2 는 소프트웨어 도구는 그 도구에 의해 생산된 출력문서 또는 코드가 확인 및 검증활동을 통해 그 도구로 인하여 발생한 오류를 찾아낸다면 도구자체에 대한 입회, 검토 및 시험과 같은 확인 및 검증이 요구되지않는 다고 기술하고 있다.

본 논문에서는 요건추적도구를 문서작성의 기반으로 하는 방법을 선택하고 모델링도구, 시험도구, 형상관리도구를 연계하는 방법을 사용한 통합 환경 중 모델링 도구와 시험도구에 대해 중점적으로 고찰하였다.

2. CASE 도구 적용을 통한 원전 계측제어 시스템 개발

소프트웨어 공학 도구 통합 모형은 그림 1 과 같다. 통합 환경이란 모델링도구, 개발 도구 및 검증 도구로 대별할 수 있다. 세부적 단위 도구로서, 개발 도구는 프로그래밍 도구를 포함하고, 검증 도구는 요건추적도구, 테스트도구, 형상관리도구로 구성된다.

그림 1 과 같은 통합 모형은 변경되는 요건 관리를 효율적으로 관리할 수 있으며 하나의 요건에 부여되는 종속의 충족 요건들을 확인할 수 있다는 장점을 가지고 있다.

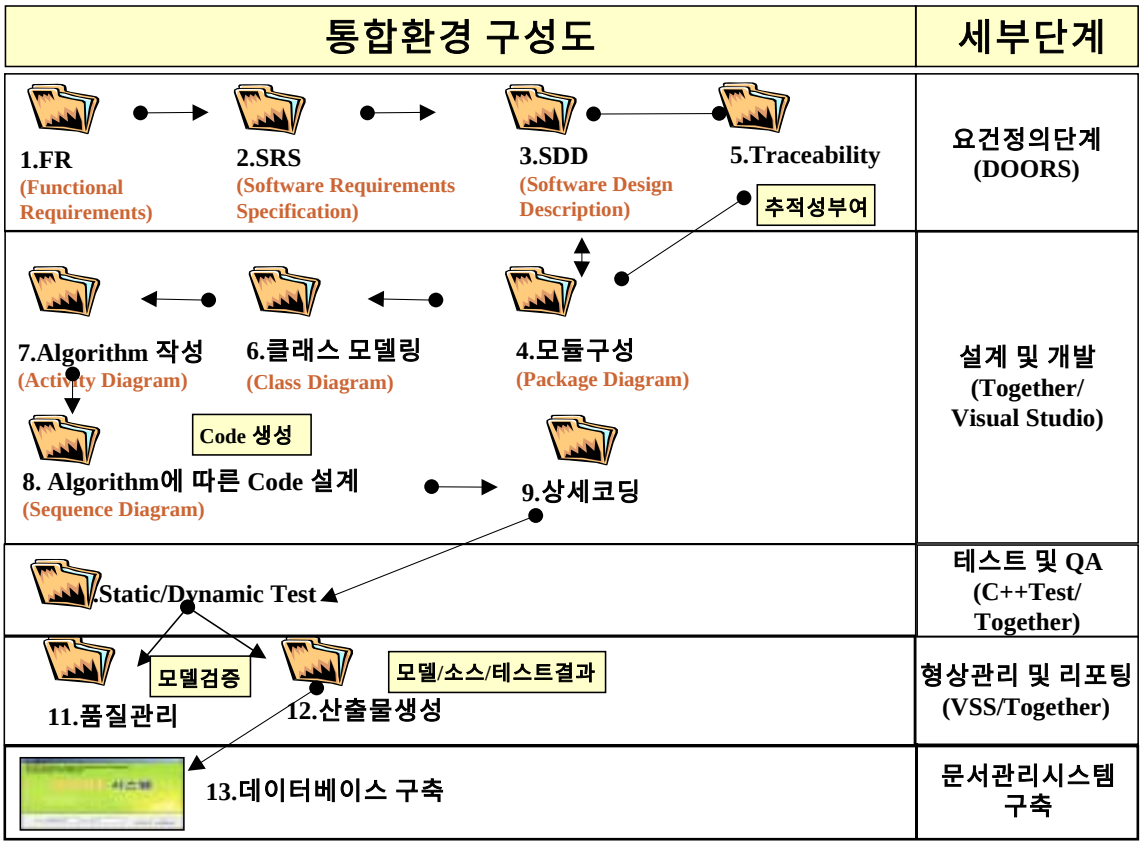


그림 1. CASE 도구를 기반으로 한 소프트웨어 통합개발 모형도

2.1 모델 기반 소프트웨어 설계 도구 적용

모델 기반 소프트웨어 설계를 위해 본 논문에서는 대표적인 모델링 기반 언어인 UML(Unified Modeling Language) 1.4 가 사용되었다. UML은 소프트웨어 개발 과정에서 산출되는 산출물들을 명시, 개발, 문서화하기 위한 모델링 언어로서 객체지향형 언어인 C++와 함께 클래스 다이어그램 등 개발 용도에 따른 다이어그램을 작성할 수 있다.

원자력발전소용 소프트웨어 개발 단계에서 본 모델 기반 소프트웨어를 이용할 수 있는지 여부를 검증하기 위해 원자로 보호계통 알고리즘중 가압기 저압력 트립 알고리즘을 자연어 명세로 제공하고 모델 기반 설계도구만을 이용하여 소스코드 단계까지 작성하여 보았다.

우선 가압기 저압력 트립 모듈에 필요한 클래스를 Class Diagram 을 이용해 설계하였으며, 구체적 Algorithm 은 Activity Diagram 을, Algorithm 에 따른 Code 설계는 Sequence Diagram 을 사용하였다. Activity Diagram 은 해당 모듈이 가지는 Algorithm 을 설계하기 위한 목적으로 사용되며, Sequence Diagram 은 Algorithm 에 따른 실제 Code 를 생성하기 위한 목적으로 사용된다. 따라서, 상세 코딩에 들어가기 전에 작성된 Sequence Diagram 을 통해 Algorithm 의 중요한 로직을 코드로 자동 생성함으로써, 모든 코드를 수작업 해야 하는 어려움을 없앴다.

그림 2 에 가압기 저압력 트립 알고리즘 구현을 위한 설계 및 개발 단계를 명시하였다.

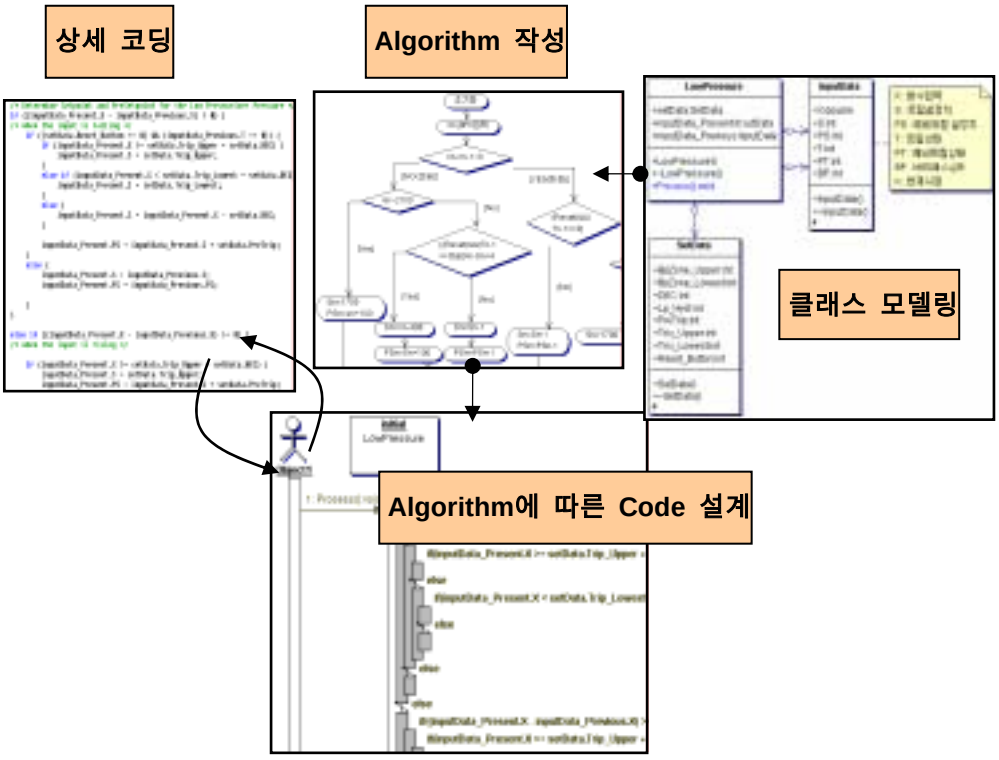


그림 2. 가압기 저압력 트립 알고리즘 구현을 위한 설계 및 개발 단계

코드 생성 자동화를 지원하는 CASE 도구를 사용하지 않던 기존 작업은 작성된 Algorithm 을 참고하여 개발자가 수작업을 통해 전체 코드를 작성해야만 했으나, Sequence Diagram 을 이용해 코드를 자동 생성할 경우 Algorithm 을 따르는 분기문, 조건문 등을 자동으로 생성할 수 있다. 대상으로 한 시스템의 경우 전체 코드 분량의 약 70% 가량을 자동으로 생성할 수 있었으며, 코드 작업 시간을 현저히 줄일 수 있었다. 뿐만 아니라 Algorithm 에 대한 개발자의 주관적 이해로 인한 오류를 줄일 수 있는 장점도 얻을 수 있었다.

Class Diagram 에 그려진 모델은 실시간에 C++ Code 로 생성되며, C++ Code 에서 일어난 변경사항은 SRE(Simultaneous Round-trip Engineering) 기능을 이용해 바로 Class Diagram 에 반영된다. 따라서, 모델과 코드상의 불일치에 대한 위험성을 줄일 수 있다. 이와 같은 모델과 코드의 일치는 항상 자동으로 지원되며, 모델링 도구의 에디터뿐만 아니라 Visual C++를 이용했을 때도 지원이 가능하다. 따라서, 정확한 모델은 정확한 프로그램을 가능하게 하고, 프로그램을 정확하게 표현하고 있는 문서로 연결되어 시스템의 정확성과 개발의 효율성이 높아지게 된다. 그림 3 은 모델과 개발코드 및 산출물이 자동으로 일치되고 있음을 나타내고 있다.

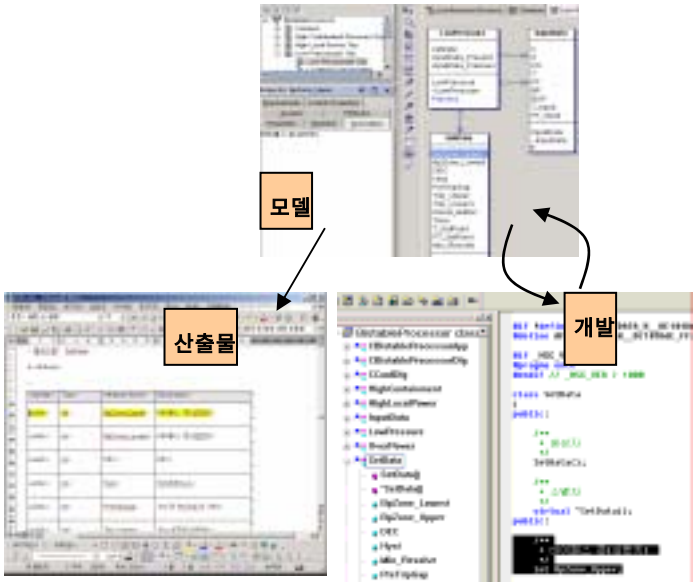


그림 3. 모델과 개발 코드, 산출물의 자동 일치

그림 3 에서 클래스 다이어그램을 이용한 설계 단계와 코드의 개발 단계는 연동성을 가지고 있어 소스코드 레벨에서의 변동은 클래스 다이어그램에서 자동으로 추적되며 이와 역 방향 관계로 성립된다. 아울러 모델에 속성을 비롯한 버전정보, 작업시간, 기능에 대한 설명 등을 입력해 놓으면 입력된 속성 정보는 코드의 주석으로 처리되고, 문서화 작업 시 자동으로 출력된다.

3. 모델링 도구에 의해 산출된 소프트웨어 모듈의 검증

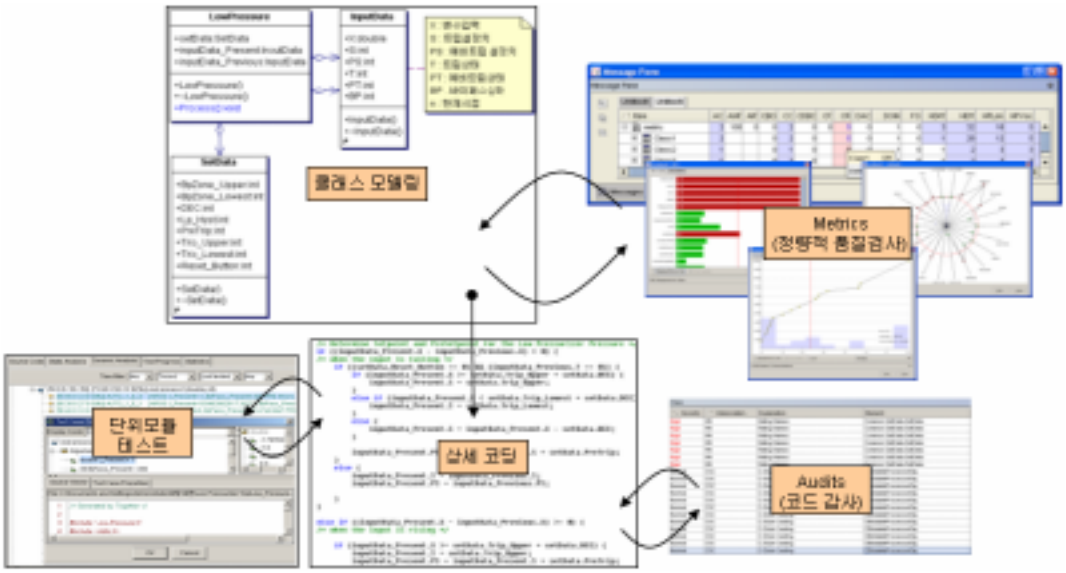


그림 4. 소프트웨어 검증 단계 모형도

안전성 필수 등급의 소프트웨어는 확인 및 검증 과정을 거치게 되며 품질보증을 비롯한 모듈 테스트는 핵심적인 과정의 하나가 된다. 본 장에서는 그림 2 와 3 의 단계를 거쳐 생성된 모델에 대한 정량적 품질검사, 소스코드를 위주로 수행한 단위 모듈 테스트와 코드 감사 단계의 모형도는 그림 4 에 표시되어 있다.

3.1 소프트웨어 감사 (Audit)

품질보증에 해당되는 작업으로 우선 개발된 시스템이 표준 코딩 지침에 얼마나 적합한지, 효율성이 떨어지는 코드는 없는지, 성능상 문제를 일으킬만한 코드는 없는지 등을 검사하는 Audits(코드 감사)이 있다. Audits 을 이용하면 안정성이 보다 높은 소프트웨어를 개발할 수 있게 되는 것이다. Audits 은 여러 가지 그룹으로 나눌 수 있는데, 이중에서 표 1 과 같은 항목을 적용하여 코드를 검사해 보고, 지적된 오류를 해결하였다.[2]

표 1. Audit 항목 및 내용

Audit 항목	내 용
Coding Style	코딩 스타일에 관련된 오류 검색
Critical Errors	중요한 에러를 발생할 수 있는 요소 검색
Declaration Style	Attribute, Operation, Class 등 선언시의 스타일을 정의하여 벗어난 요소 검색
Naming Style	Class, Exception Class, Operation, Variable 의 이름에 대한 규칙을 정의하여 벗어난 요소 감사
Performance	성능과 관련된 항목들로 구성
Possible Errors	잠재적 에러가 발생할 수 있는 사항들을 점검
Real-Time	실시간 모델 구성과 관련된 항목

그림 5 는 테스트 도구를 이용한 Audits(코드 감사) 기능을 나타낸다. Audits 를 수행하게 되면 공통적으로 해당 코드에 대한 오류를 찾아 리스트 형태로 보여준다. 해당 리스트 항목을 선택하게 되면 해당 코드가 나타나고, 해당 코드에 대한 설명 및 개선 방향을 알려준다.

그림 5 에서 지적된 오류는 Hiding Names 라는 오류로서, 클래스의 Attribute, Operation, 지역 변수의 이름이 비슷하게 혼용되어 사용될 경우 나타나는 오류이다. 여기서는 index 라는 Attribute 가 해당 오류를 범하고 있다.

Audits 를 이용하였을 경우 프로그램의 오류가 줄어 들게 되고, 프로그램의 성능이나 가독성 등이 높아지게 되며, 개발자의 코딩 실력 향상에도 큰 도움이 된다.

그림 5. 테스트 도구를 이용한 Audits(코드 감사) 기능

3.2 정량적 품질검사 (Metrics)

다음으로 개발 도구를 이용해 개발된 프로그램은 클래스와 Method 들이 원하는 요건대로 동작하는지 단위 모듈 테스트를 진행하게 된다. 이를 위해 사용하는 테스트 도구는 필요한 Test Case 를 자동 생성하여 White-box 테스트를 진행하게 되며, Line, Cumulative Line, Block, Branch, Path Coverage 분석 등이 가능하다. [3]

다음 그림 6 은 가압기 저압력 트립 모듈을 테스트한 통계 값을 보여주고 있다. 테스트 결과에서 Confidence Factor 는 소프트웨어 실패예측(Fault Estimation)의 정량적 척도이다. 한편 도구에서 표시하는 통계 값은 Testing 결과라기 보다는 수행한 Testing 에 대한 통계로 이해해야 한다.

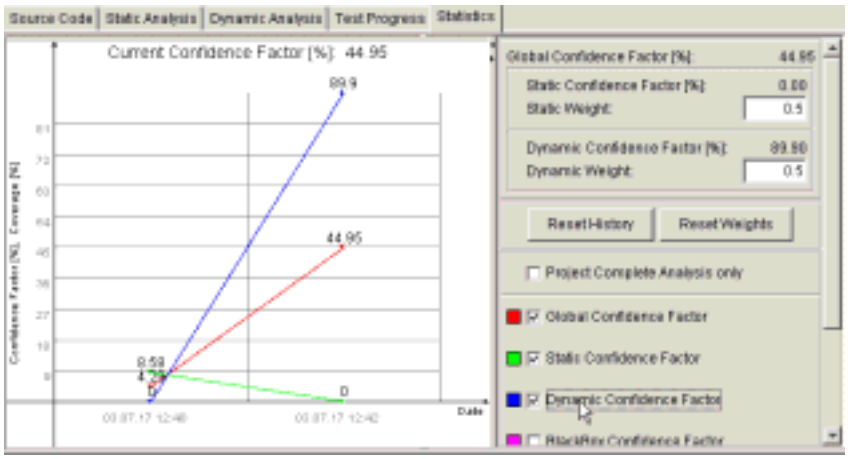


그림 6. Testing 통계-Statistics

Testing 결과는 Static 혹은 Dynamic (Module Test) Analysis 를 수행한 결과를 보여 주는 것으로 통상적으로 분석 결과 오류가 있으면 수정을 하는 절차를 진행하게 된다. 하지만 Testing 통계라는 것은 이와 같은 절차를 진행하는 동안 어느 정도(정량적 수치) Testing 이 수행이 되었는지를 보여주는 수치이다.

동적시험은 실행코드 부분에 대해 수행되는 것이 일반적이며 전체 코드의 약 50% 정도가 수행되는 것으로 알려져 있다. [4]

다음 표 2 는 시험도구에서 수행되는 통계항목을 설명하고 있다.

표 2. Statistics 항목 설명

값/데이터	설명
Project Complete Analysis Only	이 옵션을 선택하면 전체 프로젝트에 대한 complete analysis 명령을 수행했을 때만 값이 보여진다.
Global Confidence Factor	Global Confidence Factor 는 Static and Dynamic Confidence Factors 의 조합이다. 조합할 때 Weight(기본 값은 각각 0.5 로 되어 있어 일반적으로 평균이 된다)를 사용하게 된다.
Static Confidence Factor	Static Confidence Factor 는 발견된 static analysis violations 의 개수와 알려진 static coverage 값에 기초한다.
Dynamic Confidence Factor	Dynamic Confidence Factor 는 White-box 와 Black-box Confidence Factor 의 합으로 구해진다.
White-box Confidence Factor	White-box Confidence Factor 는 발견된 예외처리 되지 않은 런타임 예외 개수와 알려진 Dynamic Coverage 값에 기초한다.
Black-box Confidence Factor	Black-box Confidence Factor 는 발견된 specification/regression 에러 개수와 알려진 Dynamic Coverage 값에 기초한다.
Static Coverage	Static Coverage Confidence Factor 는 테스트된 파일 개수를 프로젝트의 모든 파일 수로 나눈 것이다.

그림 7 에 정량적 품질 검사 결과를 도시하는 그래프를 보여준다. 그래프의 각 방사상 라인은 Metrics 의 항목이 되며 이중, 순환 복잡도와 클래스의 가중 정적 복잡도 WMPC 를 개략적으로 설명하면 다음과 같다.

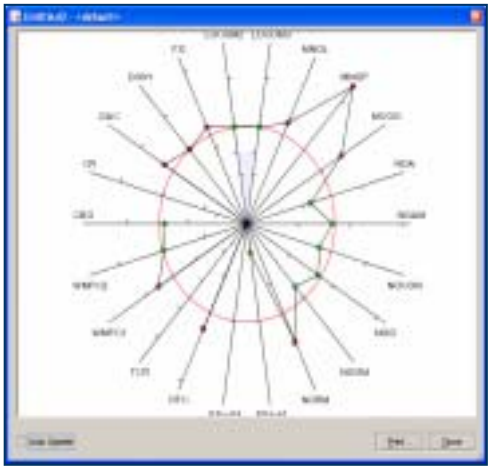


그림 7. 정량적 품질 검사 결과 (Kiviat Graph)

순환 복잡도 (Cyclomatic complexity, CC)는 프로그램의 논리적인 복잡도의 정량적인 측정을 제공해 주는 소프트웨어 매트릭스이다. 이 메트릭스가 기본 경로 테스팅 방법의 내용으로 사용될 때 순환 복잡도에 대해 계산된 값은 프로그램의 기본 집합 (Basis set)에 있는 독립적인 경로 (Independent path)들의 개수를 정의하고 또 모든 문장이 적어도 한 번은 실행되었음을 보장하기 위해 수행되어야 하는 테스트 횟수의 상한값을 제공한다. 새로운 독립적인 경로는 적어도 하나의 새로운 처리 문장이나, 하나의 새로운 조건을 도입시킨 프로그램을 통과하는 임의의 경로이다. [5]

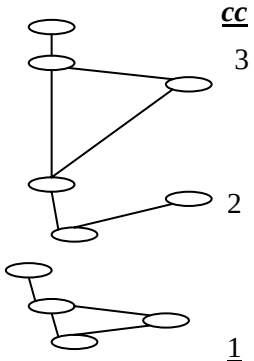
클래스의 가중 정적 복잡도(WMPC, Weighted Methods Per Class)는 클래스의 모든 방법에 대한 정적 복잡도의 합으로 정의된다. 따라서 WMPC 가 증가하면 개발 시간/비용이 증가, 하위 클래스에 대한 영향 증가, 테스팅의 어려움 증가, 클래스의 재사용이 감소됨을 의미한다.

그림 8 과 같은 샘플 WMPC 클래스의 one() 오퍼레이션의 CC 를 계산해보면 3(=7-6+2*1), two() 의 CC 는 2(=4-4+2*1), three(int i)의 CC 는 1(=1-2+2*1)이 된다. CC 를 계산하는 식은 식 (1)과 같다 (e:edge 의 개수, n: node 의 개수, p:컴포넌트의 개수).

$$CC = V(G) = e - n + 2p \qquad (1)$$

그림 8 과 같은 샘플 소스 코드의 WMPC 복잡도는 식 (2)와 같이 6 이 된다.

```
public class WMPC {
        public void one(){
                if(true) {
                        two();
                } else {
                }
                if(true && !false){
                }
        }
        public void two(){
                if(true){}
        }
        public void three(int i){
```



$$\mathrm{WMPC1} = \sum_{i=1}^{n} cc_i = 3 + 2 + 1 = 6$$

그림 8. WMPC 를 산정하기 위한 샘플 소스 코드와 경로로 표현한 값

WMPC (Weighted Methods Per Class)는 클래스의 모든 방법에 대한 정적 복잡도의 합으로 정의된다. 따라서 WMPC 가 증가하면 개발 시간/비용이 증가, 하위 클래스에 대한 영향 증가, 테스팅의 어려움 증가, 클래스의 재사용이 감소됨을 의미한다.

그림 8 의 샘플 소스 코드의 WMPC 복잡도는 6 이 된다.

$$WMPC = \sum_{i=1}^{n} CC_i = 3 + 2 + 1 = 6 \qquad (2)$$

그림 7 의 Kiviat Graph 상에 표시된 클래스의 가중 정적 복합도 역시 비교적 낮게 유지되고 있음을 알 수 있다.

다음 그림 9 는 Testing 시 어느 정도의 코드가 Cover 되었는지를 보여주는 Coverage 기능이다. 테스트를 수행하기 위해서는 다양한 테스트 케이스가 필요하고 각각의 테스트 케이스는 각각의 Testing Coverage 를 가지게 된다.

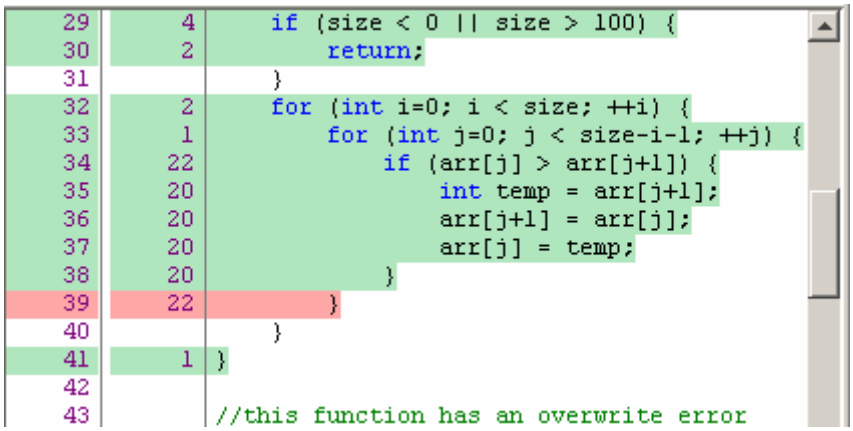


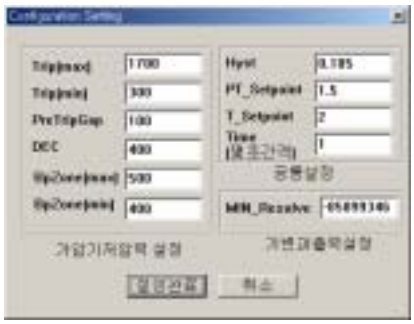
그림 9. Testing Coverage

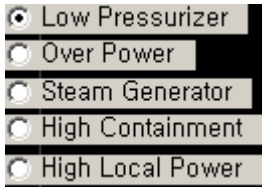
3.3 실행 파일의 생성 및 확인

시스템은 파일 형태로 입력되는 압력 값 등을 읽어 들여 다양한 로직을 수행하게 된다. 그림 10 은 테스트를 거친 소스파일을 C++ Compiler 를 이용하여 실행 파일로 만든 후 별도의 시뮬레이션 환경에서 실행 파일이 잘 동작하는지를 확인한 결과이다.

본 시뮬레이션 환경은 가압기 저압력 트립 조건뿐만 아니라, 가변 과출력 트립, 증기발생기 저수위 트립등 다양한 조건에 대해 시험할 수 있도록 구성되어 있다.

확인 결과 본 모델을 통해 생성한 소프트웨어가 이상없이 동작함을 확인하였다.





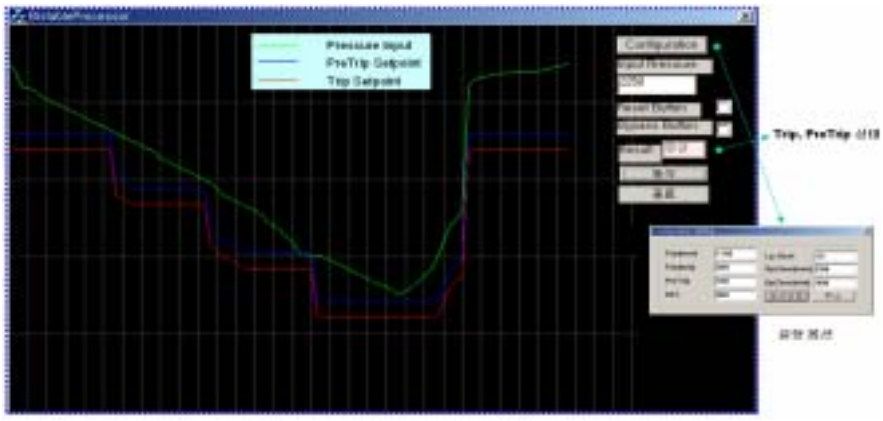


그림 10. 가압기 저압력 트립모듈 실행 화면

6. 결론

원자력발전소 계측제어시스템 소프트웨어 개발 방법론중의 일부로서 모델링 기반의 설계, 소스코드의 생성, 테스트 및 품질관리 그리고 실행파일의 작성을 소프트웨어 모델 기반 도구를 이용하는 방법에 대해 중점적으로 고찰하였다. 모델링 도구의 표준 모델인 UML 을 사용하여 객관적인 모델을 구축할 수 있었으며, 시스템의 기능구현에서 소스코드의 생산까지를 일관적으로 행할 수 있었다. 또한 모델링 도구에서 사용하여 소스코드에서 변경되는 변수 항목들에 대하여 역 엔지리어링인 SRE 방법에 의하여 소스코드의 변경으로 인한 오류를 줄일 수 있었다.

현재까지 발표된 모델기반 설계 도구의 경우, 시컨스 다이어그램을 이용해 코드를 자동 생성할 경우 알고리즘을 따르는 분기문, 조건문등을 자동으로 생성하여 전체 코드의 약 70%정도의 소스코드를 자동 생성하는 것으로 확인하였다.

이와 같은 모델 기반 설계를 통해 복잡한 계측제어 시스템 개발 시 인적 오류를 현저히 줄일 수 있을 것으로 판단된다.

테스트 도구를 이용한 소프트웨어 모듈의 검증결과, 높은 커버리지와 비교적 낮은 복잡도를 유지하고 있음을 확인하였다. 그러나 좀더 효율적인 시험을 수행하기 위하여는 시험도구에서 제공하는 Static Confidence Factor, Dynamic Confidence Factor 등의 테스트 결과의 대한 정량적 수치 계산에 대한 방법론등에 대한 연구가 향후 진행되어야 할 것으로 판단된다.

알리는 말씀

본 논문은 산자부가 지원하는 전력산업연구개발사업의 과제 수행으로 이루어졌습니다. 본 과제를 수행하는데 물심양면으로 지원하고 있는 산자부 및 한국전력연구원(KEPRI) 관계자 여러분과 한국전력기술㈜ 내의 관계자 여러분께 깊은 감사를 드립니다.

참고문헌

1. EPRI TR-105989-Vol. 1, “ Software Fault Reducing using Computer-Aided Software Engineering (CASE) Tools, 1995
2. Together Soft, “Together Control Center 6.0 User Guide.”, 2002

3. M.R, Woodward., D. Hadley, et. al., “. Experience with path analysis and testing of programs”, IEEE Trans. Software Engineering, Vol. 6, No. 3, 1980
4. Brian Henderson-Sellers, Object-Oriented Metrics: Measures of Complexity. Prentice Hall, 1995.
5. S. L. Pfleeger, “ Software Engineering-Theory and Practice”, Prentice-Hall International, Inc., 1998