디지털산업정보학회 논문지 제12권 제4호-2016년 12월
http://dx.doi.org/10.17662/ksdim.2016.12.4.013

C# 프로그래밍 무기체계 소프트웨어에 대한 신뢰성 시험 기준 연구*

신 봉 득** · 오 혁 준***

A Study of Criteria of the Reliability Test for C# programming software in Weapon System

Shin Bongdeug · Oh Hyukjun

〈Abstract〉

Defense Acquisition Program Administration's weapon system software development and management guideline specifies the criteria of software reliability tests including static and dynamic tests mainly on C/C++ languages. Recently, Defense Acquisition Program Administration expanded the scope of software reliability test for the various languages including C#, java etc. but specific criteria for them are not established. This study suggests the reliability test procedures and standards on C# programming software in weapon system. For the static test, considering the nature of the C# which depends on .NET framework, this paper introduces applying coding rules recommended by Microsoft Corp. Visual Studio 2012. Block coverage provided by Visual Studio is applied on dynamic tests and the achievement objectives for block coverage according to the software levels(A, B, C) are suggested. Also, the software reliability test procedures and standards proposed by this paper are properly verified through the case study.

The result of this study can be used for establishing the specific criteria of the software reliability test for C# programming software in weapon system.

Key Words : C#, Static Test, Dynamic Test, Coding Rule, Block Coverage

Ⅰ. 서론

현대 무기체계는 대부분의 성능이 소프트웨어에 의해 구현되고 있으며 소프트웨어의 결함 발생 시 전투력 손실은 물론이고 심각한 재산 및 인명 피해를 초래할 수 있으므로 그 중요성과 관심이 높아지고 있다. 최근 무기체계는 하드웨어 중심의 시스템에서 소프트웨어 중심의 내장형 시스템으로 점차 발전되고 있고, 내장형 소프트웨어의 신뢰성과 품질 확보에 대한 필요성과 중요성이 증대되고 있다 [1-4].

* 이 논문은 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터육성지원사업의 연구결과로 수행되었음 (IITP- 2016-R2718-16-0014)
* 본 연구는 2016년도 광운대학교 교내 학술연구비의 지원을 일부 받아 수행된 연구임.
** 광운대학교 전자통신공학과
*** 광운대학교 전자통신공학과 교수(교신저자)

따라서 방위사업청은 무기체계 소프트웨어 개발 및 관리 매뉴얼을 통해 무기체계 연구개발사업, 핵심기술(시험개발)사업 등 개발 착수 시부터 소프트웨어에 대해 강도 높은 시험을 수행하도록 요구하고 있으며 이에 대한 신뢰성 시험(정적시험, 동적시험) 절차 및 기준을 제시하고 있다[5].

이번 2016년 개정판에서는 요구사항 기반시험, 실타겟 시험, 시험 적용 대상 언어 확장 등 여러 항목이 추가되었으며, 특히 C#, java 등 다양한 언어에 대해 시험 범위를 확장하였다. 하지만, 여전히 C/C++언어 기준으로 서술되어 있고 C#, java 등 다른 개발언어에 대한 구체적인 기준이 확립되어 있지 않아, 이에 대한 신뢰성 시험 기준 및 방안이 필요하다.

이에 따라 본 논문에서는 C/C++ 위주의 기존 무기체계 소프트웨어 신뢰성 시험 절차 및 기준을 근거로, 개발언어 C#에 대한 소프트웨어 신뢰성 시험 기준 및 방안을 제시하고자 한다. 정적시험에 대하여는 코딩규칙 검증을, 동적시험에 대하여는 코드 실행률 점검에 대한 방안과 절차를 제시한다. 또한, 소프트웨어 등급에 따른 동적시험 달성에 대한 목표값 방안도 제시한다.

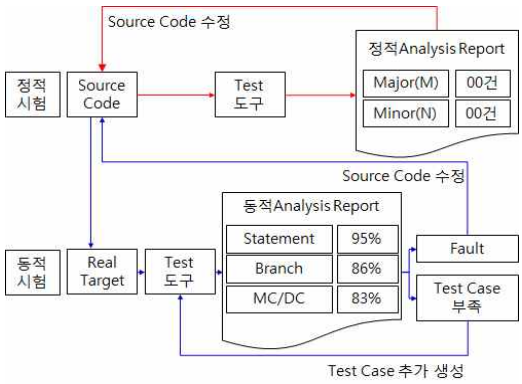
본 논문은 총 5절로 구성되어 있다. 2절에서는 무기체계 소프트웨어 개발 및 관리매뉴얼의 소프트웨어 신뢰성 시험 절차 및 기준과 Microsoft사의 Visual Studio 2012의 개발언어 C#에 대한 Code Analysis 및 Block Coverage 산출 절차에 대하여 소개한다. 3절에서는 C# 소프트웨어에 대한 신뢰성 시험 수행 기준 및 방안과 동적시험 달성에 대한 목표값 방안에 대해 제시한다. 4절에서는 무인기의 지상체계 주장비 운용 소프트웨어에 대하여 3절에서 제시한 기준에 따라 수행한 신뢰성시험 결과를 사례로 기술하고 분석하여 본 연구의 결과에 대한 타당성을 제시하며, 5절에서는 결론을 맺고 향후 연구과제에 대해서 검토한다.

Ⅱ. 소프트웨어 신뢰성시험

2.1 무기체계 소프트웨어 신뢰성시험

무기체계 소프트웨어 개발 관리매뉴얼은 소프트웨어 신뢰성시험을 정적시험과 동적시험으로 구분하고 있다.

정적시험은 소프트웨어를 실행하지 않은 상태에서 잠재적인 결함을 검출하는 시험을 말하며, 동적시험은 소프트웨어를 실제 하드웨어(Target)에 탑재한 상태에서 소프트웨어 통합시험 절차서에 기술된 시험절차를 이용하여 요구사항기반으로 소프트웨어 코드 실행률을 점검하는 것을 말한다[5]. 소프트웨어 신뢰성시험에서 정적시험과 동적시험을 구분하여 설명하지만 신뢰성 시험의 수행은 <그림 1>과 같이 상호 연관되어 수행된다. 먼저 정적시험을 실시하여 오류발생 시 소스코드를 수정하여 재 수행 후 동적시험을 실시한다. 동적시험은 정적시험을 수행하여 완료된 소스코드를 실제 하드웨어에 탑재하여 수행하며 목표값을 달성하지 못할 경우 소스코드를 수정한 후 정적시험-동적시험을 재수행하는 절차를 따른다.



<그림 1> 무기체계 소프트웨어 신뢰성시험 절차

2.1.1 정적시험

정적시험은 코딩 규칙(Coding Rule) 검증 시험, 취약점 점검 시험 및 소스코드 메트릭(Code Metrics) 점검으로 구분된다.

코딩 규칙은 소프트웨어 구현에 적용되는 소스 코드 작성 규칙으로서 무기체계 소프트웨어 개발 및 관리 매뉴얼【부록 6】"무기체계 소프트웨어 코딩 규칙"을 따른다. C/C++ 언어는 MISRA-C/C++을 관련 표준으로 제시하고 추가적으로 C/C++ 공통 45개, C 언어 전용 5개, C++ 언어 전용 15개 코딩규칙이 제시되어 있다. C# 언어는 Microsoft사 C# Coding conventions를 관련 표준으로 제시하고 있다.

취약점 점검은 소프트웨어 소스 코드가 CWE (Common Weakness Enumeration, CWE-658/659/660) 목록에 정의된 취약점을 포함하고 있는지 점검하는 활동을 말한다. 소스코드 메트릭 점검은 소프트웨어의 복잡도 감소, 유지보수 용이성 증대 등 소프트웨어 품질향상을 위한 소스코드의 품질 측정지표를 말하며 아래 표와 같다.

<표 1> 소스코드 품질 측정지표

구 분	내 용
측정 지표	- Cyclomatic Complexity - Number of Call Levels - Number of Function Parameters - Number of Calling Functions - Number of Called Functions - Number of Executable Code Lines

2.1.2 동적시험

동적시험은 코드 실행률이라고도 하며 문장 실행률, 분기 실행률, MC/DC 3가지 점검으로 구분한다. 문장 실행률(Statement Coverage)은 소스 코드 내의 문장 중 동적 시험 간 적어도 한 번 이상 시험된 문장의 비율(%)을 의미하고, 분기 실행률(Branch Coverage)은 소스 코드내의 분기문 중 동적 시험 간 참(True), 거짓(False)이 적어도 한 번 이상 시험된 비율(%)을 의미하며, MC/DC(Modified Condition/ Decision Coverage) 소스 코드 내 분기문에 있는 모든 조건식 중 개별 조건식의 독립적인 변화가 분기문의 참, 거짓에 영향을 미치는 모든 조합에 대해 동적 시험간 적어도 한 번 이상 시험된 비율(%)을 의미한다.

2.1.3 동적시험 목표값

무기체계 소프트웨어 동적시험 기준을 설정하는 방법은 <표 2>와 같이 결함 영향도 (Severity), 결함 발생빈도(Exposure), 결함 제어가능성 (Controllability)을 평가한 후 3요소의 조합에 따라 동적시험(S : 문장 실행율, B : 분기 실행율, M : MC/DC)에 대한 종류를 선정한다.

<표 2> 소프트웨어 동적시험 기준

결함영향도 (Severity)	결함발생빈도 (Exposure)	결함제어가능성 (Controllability)		
		C1	C2	C3
S1	E1	S	S	S
	E2	S	S	S
	E3	S	S	S
	E4	S	S	S
S2	E1	S	S	S
	E2	S	S	S
	E3	S	S	S
	E4	S	S	B
S3	E1	S	B	B
	E2	B	B	B
	E3	B	B	B
	E4	B	B	M
S4	E1	B	B	M
	E2	B	M	M
	E3	B	M	M
	E4	M	M	M

<표 2>의 무기체계 소프트웨어 동적시험 기준을 설정하는 방법을 요약해서 분석하면 소프트웨어 등급을 결정한 후에 각 등급별로 동적시험 항목을 설정하는 것과 같게 된다. 즉, 소프트웨어 등급을 A, B, C등급으로 분류하고 각각의 등급별 동적시험 항목 기준과 목표값을 설정하면 다음 <표 3>과 같다.

<표 3> 소프트웨어 등급별 동적시험 기준 및 목표값

동적시험 항목	소프트웨어 등급		
	A	B	C
문장실행률	100%	100%	100%
분기실행률	100%	100%	
MC/DC	100%		

<표 4> 코딩룰 항목별 내용 및 규칙 개수

항목	지원내용	규칙개수
Design	.NET Framework 디자인 지침을 준수할 수 있도록 지원	62
Globalization	라이브러리 및 응용프로그램 지원	11
Interoperability	COM 클라이언트와의 상호 작용을 지원	16
Maintainability	라이브러리 및 응용 프로그램 유지 관리를 지원	6
Mobility	효율적인 전원 사용을 지원	2
Naming	.NET Framework 디자인 지침의 명명 규칙 준수를 지원	23
Performance	고성능 라이브러리 및 응용 프로그램을 지원	16
Portability	여러 운영체제 간의 이식성을 지원	3
Reliability	올바른 메모리 및 스레드 사용과 같은 라이브러리 및 응용 프로그램 신뢰성을 지원	6
Security)	더 안전한 라이브러리 및 응용 프로그램을 지원 및 보안 결함 방지	44
Usage	.NET Framework를 올바르게 사용하도록 지원	43

2.2 C# Code Analysis 및 Block Coverage

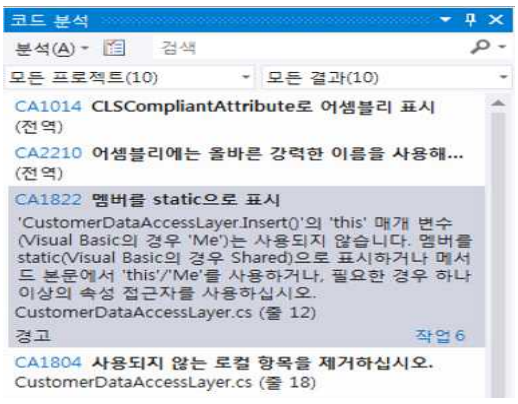
C#은 닷넷 언어 중 하나로써 Microsoft의 응용 프로그램 개발 및 구동환경인 닷넷 프레임워크(.NET Framework)를 기반으로 한 언어이다. 인터넷 환경에 부적합한 기존의 개발언어와 달리 네트워크 환경에 최적화된 언어로 플랫폼에 독립적이다. 따라서 C# 시험을 위해선 Microsoft의 IDE인 Visual Studio를 사용하여야 하며 단위시험은 Visual Studio 2010 Professional 이상부터 지원한다[6]. C# Code Analysis 및 Block Coverage에 대한 시험 절차도 <그림 1>의 무기체계 소프트웨어 신뢰성시험 절차와 같다.

2.2.1 Code Analysis

(1) 코딩룰 검사

Microsoft의 C# Coding Conventions은 규칙을 11개의 항목으로 분류하였으며 <표 4>와 같다. Visual studio에서는 C#을 위한 여러 가지 코딩 규칙과 코딩 규칙을 속성별로 모아놓은 규칙집합(Rule Set)을 제공하므로 정적분석 시 프로젝트에 맞는 규칙집합을 선택하여 정적분석을 수행할 수 있다. 원하는 규칙 집합이 없을 경우엔 사용자 정의 규칙집합을 생성하여 프로젝트의 특성에 맞게 규칙을 선택적으로 적용할 수 있다[7].

코딩 규칙검사는 프로젝트 빌드 시 자동으로 수행되며 수행 후에는 <그림 2>와 같이 코드분석 패널을 통해 위반된 코딩 규칙과 위반이 발생한 소스코드 라인 수를 확인할 수 있다.



<그림 2> 코딩규칙검사 절차

<표 5> Visual Studio에서 제공하는 코드 메트릭

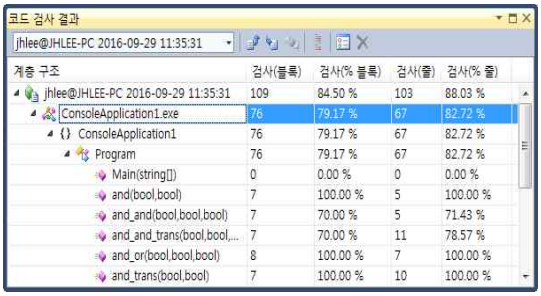
메트릭	의미
Lines of Code	컴파일된 소스코드 라인 수 수치가 작을수록 유지보수성이 높음
Depth of Inheritance	클래스의 상속 수준
Class Coupling	클래스의 변수, 속성, 매쏘드, 인터페이스, 상속성 등을 측정하며 수치 10 이하가 좋으며 수치가 작을수록 시험용이성이 높아짐
Cyclomatic Complexity	소스코드의 복잡도에 대한 매트릭으로 IF, While, Switch(Case Loop(For),문 등을 계산하여 소스코드 경로 수를 나타냄. 수치 10 이하가 좋음
Maintainability Index	0~100까지 값을 가지며, 100에 가까울수록 좋음 0~9 : 유지보수 불능, 10~19 : Warning, 20 이상: 유지보수 가능. 이 매트릭은 소스코드 라인수, Cyclomatic Complexity, Hallstead Complexity(연산자, 피연산자 수)를 합하여 측정

(2) 코드 메트릭 산출

Visual Studio에서는 코드품질을 위해 Code Metric 산출 기능을 제공하고 있는데, 도구에서 산출 가능한 메트릭으로는 Lines of Code, Depth of Inheritance, Class Coupling, Cyclomatic Complexity, Maintainability Index 5개가 있으며 각각의 메트릭은 함수, 클래스별, 프로젝트 별로 표시된다. 각 메트릭별 의미는 <표 5>와 같다.

2.2.2 Block Coverage

C/C++과 C# 시험에서 가장 큰 차이점은 코드 실행률을 산출하는 동적시험이다. 네트워크 기반 언어 C#은 실타겟과 연동하는 환경을 구축하지 못하므로 원본 프로젝트와 연동되는 테스트프로젝트를 생성하는 작업이 필요하다[8]. 그 후 단위테스트를 수행한다면 stub기능인 shim/fake를 이용하여 연관되어 있는 클래스를 격리시키고 원본코드 실행을 위한 테스트 코드를 작성한다[9]. 테스트 코드를 실행시키면 원본코드의 실행된 블록(Block) 수 및 소스코드 라인 수는 <그림 3>과 같이 확인이 가능하다.



<그림 3> 테스트 실행 결과 확인

이와 같은 동적시험 프로세스를 통해 블록 실행률(Block Coverage)이 산출된다. 블록(Block)은 한 개의 진입점과 한 개의 출구점을 가지고 있는 명령어의 집합으로 정의하며, 블록에서 출구점은 분기 명령어(branch instruction), 함수 호출(function call), 반환 명령어(return instruction)나 관리코드(managed code), throw 명령어를 의미한다.

Ⅲ. 무기체계 C#소프트웨어 신뢰성 시험 기준 및 방안

본 연구에서는 C#으로 개발된 무기체계 소프트웨어 신뢰성 시험에 사용한 Visual Studio 2012를 기준으로 연구하였으므로, 지원하는 기능도 Visual Studio 2012기준으로 하였다. 무기체계 소프트웨어 신뢰성 시험절차에서 정적시험(Static Code Analysis, Code Metric) 및 동적시험(Unit Testing, Code Coverage)을 체크하기 위한 기능은 <그림 4>와 같다.

구분	기능	Ultimate	Premium	Professional	Test Professional
디버깅과 진단, 테스팅	IntelliTrace, 웹성능 테스트, 로드테스트	●	●		
	Microsoft Fakes (유닛테스트 isolation)	●	●		
	코드 메트릭, 코드 검사, 코딩된 UI 테스트	●	●		
	수동테스트, 에비테스트, 테스트 사례 관리, Lab management, Fast-Forward	●	●		●

<그림 4> Visual Studio 2012 모듈별 지원 기능

시험을 위해 추가적인 기능이 필요하거나 Visual Studio 2010 이전 버전에서 시험을 수행하고자 한다면 확장프로그램을 연동하여 시험이 가능하다. 확장프로그램은 NCover, PartCover, OpenCover, SD Test Coverage, JetBrains dotCover, NCrunch, NDepend 등이 있으며 확장프로그램마다 지원해주는 기능이 조금씩 다르므로 원하는 기능에 따라 도구를 선택적으로 적용하면 된다.

3.1 정적시험

3.1.1 코딩규칙 검증

무기체계 소프트웨어 개발 및 관리 매뉴얼에서 C# 언어에 대한 코딩규칙으로 Microsoft의 C# Coding Conventions이 제시되어 있으나, 특정 룰을 정하지는 않고 있다. Microsoft의 C# Coding Conventions의 해당 룰은 Visual Studio에서 검증이 가능하지만, 무기체계 소프트웨어 개발 및 관리 매뉴얼에서 특정하게 제시하고 있는 C/C++위주로 구성된 코딩규칙(C/C++공통 규칙 46개, C 전용 규칙 5개, C++ 전용규칙 15개)을 검증하는 데 한계가 있다.

C#에서 코딩규칙을 검증하기 위해서는 무기체계 소프트웨어 개발 및 관리 메뉴얼에서 제시한 주석 및 명명규칙과 C/C++ 공통규칙에 해당하는 규칙들을 C# Code Convention과 매핑하고 C#에서 추가적으로 적용해야 할 룰은 전용규칙으로 만들어 적용한다. <표 6>은 공통규칙에 해당하는 C# 코딩규칙을 매핑한 예제로, 코딩규칙 검증을 위해선 적용할 C# 규칙을 선택하는 것이 선행되어야 한다.

<표 6> 코딩규칙 매핑(예시)

무기체계 소프트웨어 코딩공통규칙	C# 코딩 규칙
5) 변환규칙 - 가)	CA2233
5) 변환규칙 - 나)	CA2241
5) 변환규칙 - 다)	CA2241
5) 변환규칙 - 라)	CA1800
5) 변환규칙 - 마)	CA1725
5) 변환규칙 - 바)	CA1725

3.1.2 취약점 점검

취약점 점검은 2016년 개정 이전의 '런타임 에러 점검'에서 변경된 용어로 무기체계 소프트웨어 개발 및 관리 매뉴얼의 취약점 점검기준은 CWE이다. 그러나 C#을 위한 CWE 항목이 제시되어 있지 않으며 Visual Studio도 CWE 규칙을 지원하지 않는다.

따라서 Microsoft의 C# Coding Conventions 11개의 항목에서 소스코드의 런타임 에러를 유발할 것으로 판단되는 항목인 호환성 (Interoperability), 이동성(Mobility), 성능(Performance), 보안성(Security)에 해당하는 규칙을 취약점 항목으로 매핑하여 코딩규칙검증 수행 시 해당항목의 시험 결과를 통해 취약점을 점검한다. 또는 C#은 JAVA와 같은 객체지향언어이므로 자바전용항목인 CWE 660 과 유사한 C# 코딩 규칙을 매핑하여 적용하는 방법도 있다.

3.1.3 소스코드 메트릭 점검

무기체계 소프트웨어 개발 및 관리 메뉴얼에서 제시하는 소스코드 품질측정지표는 총 6개로 해당

메트릭은 <표 7>과 같이 Visual Studio에서 제공하는 메트릭과 대부분 매핑이 가능하다. 하지만 Visual Studio에서 Number of Function Parameters를 제공하는 메트릭가 없다는 것이 한계점이다. 해당 메트릭 검사를 위해선 소프트웨어 설계문서와 소스코드 간의 추적성을 확인하거나 개발자가 직접 코드를 검사 방법을 통해 확인하여야 한다.

<표 7> 소스코드 메트릭 매핑

방위사업청 제시	Visual Studio 제공
Cyclomatic Complexity	Cyclomatic Complexity
Number of Call Levels	Depth of Inheritance
Number of Function Parameters	(Not matched)
Number of Calling Functions	Class Coupling
Number of Called Functions	
Number of Executable Code Lines	Lines of Code
(Not matched)	Maintainability Index

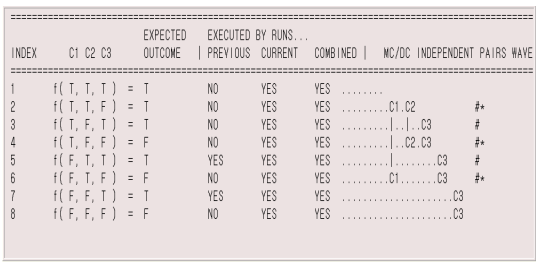
3.2 동적시험

무기체계 소프트웨어 개발 및 관리 매뉴얼의 동적시험은 문장 실행률, 분기 실행률, MC/DC이지만 Visual Studio에서는 동적시험을 위해 Block coverage가 제공된다. Block coverage는 문장 실행률과 분기 실행률 대체가능하나 MC/DC를 정확하게 대체할 수는 없다. 아래 <표 8>은 코드 실행률의 목표값 달성을 위해 필요한 테스트 케이스 수를 논리연산별로 비교하였다. MC/DC와 Block Coverage의 개수가 달라 블록 실행률을 달성(100%) 했다고 하지만 MC/DC를 100% 만족한 것이 아님을 확인할 수 있다.

<표 8> 목표값 달성을 위한 테스트케이스 수

논리연산	Block 수	테스트 케이스 수	
		MC/DC	Block Coverage
a b	4	3	2
a OR b	4	3	2
a AND b AND c	5	4	2
a OR b OR c	5	4	2
a AND b OR c	5	4	3
a OR b AND c	7	4	3
(a > 0) AND (a <100)	4	3	2
(a > 100) OR (a==100)	4	3	2

따라서 MC/DC를 위해서는 Block coverage 테스트 케이스(Block coverage Test Case : B_TC)와 <그림 5>와 같이 소프트웨어 신뢰성 시험을 지원하는 자동화도구를 통해 식별한 MC/DC 테스트 케이스(추가 테스트 케이스: 추가_TC)를 혼합하여 활용한다[10].



INDEX	C1 C2 C3	EXPECTED OUTCOME	EXECUTED BY RUNS... PREVIOUS	CURRENT	COMBINED	MC/DC INDEPENDENT PAIRS WAVE
1	f(T, T, T)	= T	NO	YES	YES	.....
2	f(T, T, F)	= T	NO	YES	YES	.....C1..C2 #*
3	f(T, F, T)	= T	NO	YES	YES	.....|...|...C3 #
4	f(T, F, F)	= F	NO	YES	YES	.....|...C2..C3 #*
5	f(F, T, T)	= T	YES	YES	YES	.....|.....C3 #
6	f(F, T, F)	= F	NO	YES	YES	.....C1.....C3 #*
7	f(F, F, T)	= T	YES	YES	YES	.....C3
8	f(F, F, F)	= F	NO	YES	YES	.....C3

<그림 5> 자동화 도구 MC/DC 테스트 케이스 테이블

3.3 동적시험 목표값

3.2절에서 살펴본 것처럼 동적시험은 문장 실행율과 분기실행률은 Block coverage로, MC/DC는 Block coverage 테스트 케이스와 자동화도구가 식별한 테스트 케이스를 혼합하여 대체할 수 있다.

따라서 2.1.3절의 방위사업청 기준을 그대로 적용하여 무기체계 소프트웨어 등급을 분류한 후 <표 9>와 같이 등급별 동적시험 기준과 목표값을 설정한다.

<표 9> C# 소프트웨어 등급별 동적시험 기준 및 목표값

동적시험 항목	소프트웨어 등급		
	A	B	C
Block Coverge	100%	100%	100%
MC/DC (BC_TC + 추가_TC)	100%		

Ⅳ. 신뢰성 시험 및 분석

최근 활발하게 개발되고 있는 국방 무인기의 지상체계 주장비에서 운용되는 소프트웨어는 대부분 C# 개발언어로 구현되고 있다. 본 논문에서의 신뢰성 시험 및 분석은 무인기의 지상체계 주장비에 사용된 C# 소프트웨어를 대상으로 정적 및 동적시험을 수행하였다.

4.1 정적시험 수행

정적시험은 C# 언어의 특성상 닷넷 프레임워크 의존적인 언어임을 고려하여 분석 도구로는 Visual Studio 2012 Pro를 사용하고 코딩룰을 선정하여 적용하였다. 아래 <그림 6, 7>과 같이 코딩규칙과 취약성 점검 관련 규칙으로 나누어서 코딩룰을 선정하였으며, 코딩규칙 154개와 취약성 점검 규칙 78개로 총 232개의 규칙을 적용하여 정적시험을 수행하였다.

CheckId	TypeName
CA1000	정적 멤버를 제네릭 형식으로 선언하지 마십시오.
CA1001	삭제 가능한 필드가 있는 형식은 삭제 가능해야 합니다.
CA1002	제네릭 목록을 노출하지 마십시오.
CA1003	제네릭 이벤트 처리기 인스턴스를 사용하십시오.
CA1004	제네릭 메서드는 형식 매개 변수를 제공해야 합니다.
CA1005	제네릭 형식에 매개 변수를 너무 많이 사용하지 마십시오.
CA1006	멤버 시그니처에 제네릭 형식을 중첩하지 마십시오.
CA1007	적합한 제네릭을 사용하십시오.
CA1008	열거형에는 0 값이 있어야 합니다.
CA1009	이벤트 처리기를 제대로 선언하십시오.

<그림 6> 정적시험(코드분석) 코딩규칙 예시

CheckId	TypeName
CA1400	P/Invoke 진입점이 있어야 합니다.
CA1401	P/Invoke는 노출되지 않아야 합니다.
CA1402	COM 노출 인터페이스에서 오버로드를 사용하지 마십시오.
CA1403	자동 레이아웃 형식은 COM 노출이면 안 됩니다.
CA1404	P/Invoke 다음에 바로 GetLastError를 호출하십시오.
CA1405	COM 노출 형식의 기본 형식은 COM 노출이어야 합니다.
CA1406	Visual Basic 6 클라이언트에서 Int64 인수를 사용하지 않습니다.
CA1407	COM 노출 형식에 정적 멤버를 사용하지 마십시오.
CA1408	AutoDual ClassInterfaceType을 사용하지 마십시오.
CA1409	COM 노출 형식을 만들 수 있어야 합니다.

<그림 7> 정적시험(코드분석) 취약점 점검 예시

4.2 동적시험 수행

동적시험은 정적시험을 통해 도출된 문제점을 수정한 후 수행하였으며, <그림 8>과 같은 절차로 시험을 수행하였다.

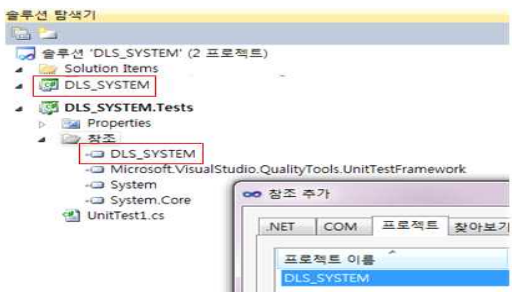
<그림 8> 동적 시험 수행절차

<표 10> CSC별 블록 개수

CSC명	등급	파일수	함수수	블록수
지상OO	A	17	123	5,896
운용OO	A	117	1870	21,137
DDS OO	A	710	32,054	927,725
운용통제OO	A	177	3,573	397,352
초기화OO	A	60	60	2,804
체계OO	A	22	322	8,189
실시간OO	B	3	8	916
비행체 OO	A	3	220	9,220
비행OO	B	50	657	10,131
비행상태OO	A	40	666	6,642
통신OO	A	17	403	6,241
데이터OO	B	48	596	19,041
경고 OO	C	32	313	8,170

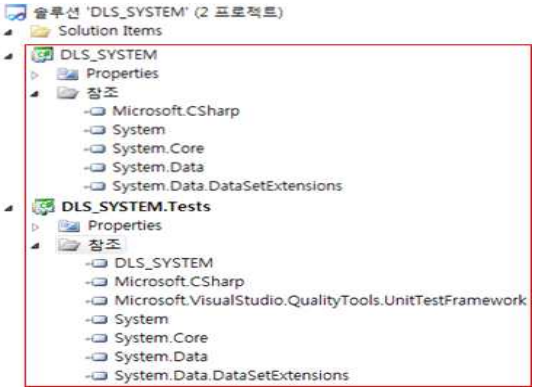
동적 시험 수행을 위한 첫 단계로 시험대상을 식별하였다. 비행안전심각도와 임무중요도에 따라 구분되는 소프트웨어 등급을 바탕으로 A, B, C등급 프로젝트를 대상으로 <표 10>과 같이 블록 개수를 파악하였다.

두 번째는 C# 소프트웨어를 테스트하기 위한 환경설정을 하였다. <그림 9>와 같이 테스트 대상 솔루션에 테스트용 프로젝트를 생성 후 테스트 대상 프로젝트를 참조하는 작업이 필요하다. 해당 작업이 선행되어야 테스트 프로젝트에서 작성하는 테스트 코드가 원본 코드를 시험할 수 있다.



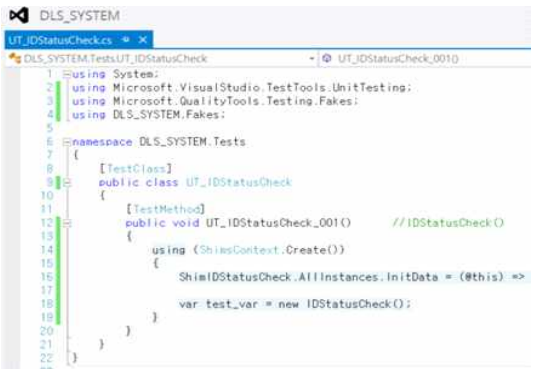
<그림 9> 테스트 생성 및 시험대상 프로젝트 연결

그 후 시험 대상 프로젝트에서 사용하는 코드를 사용하기 위해 <그림 10>과 같이 시험 프로젝트의 참조 폴더에 시험대상 프로젝트의 라이브러리를 모두 추가한다.



<그림 10> 시험대상 프로젝트 라이브러리 추가

세 번째는 원본 코드를 시험하기 위한 테스트코드를 생성한다. 다른 자동화 시험도구와는 다르게 Visual Studio는 테스트를 위한 코드를 직접 작성해야 한다. 테스트 코드는 <그림 11>과 같은 기본 틀을 따르며 TestMethod부분에 테스트 코드를 작성할 수 있다.

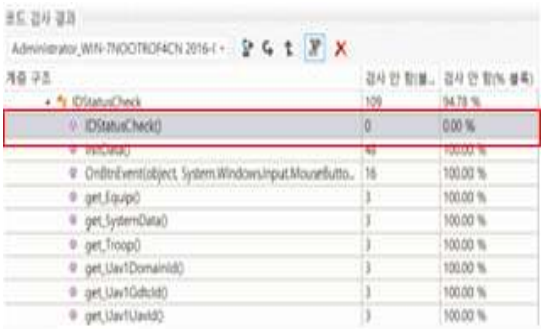


<그림 11> 테스트 코드 작성

Visual Studio는 단위 시험도 지원하고 있으며 이를 위해 해당 코드를 격리하는 기능이 필요하다. 해당 코드 격리기법은 다른 자동화 시험도구에서 Stub라고 불리는 기법으로 Visual Studio에서는 Shim또는 Stub라 부른다. 해당 기능을 사용하기 위해 Visual Studio는 Fake기능을 지원하며 <그림 12>와 같이 테스트 프로젝트의 라이브러리에서 Fake를 추가 하면 해당 라이브러리 코드에서 Shim또는 Stub 기능을 이용할 수 있다.

<그림 12> Visual Studio 테스트 기법(Fake)

네 번째는 테스트 실행 및 결과확인을 위해서 코드 검사 분석 결과를 확인한다. 테스트 프로젝트에서 테스트를 실행하면 <그림13>과 같이 코드 검사 결과를 확인할 수 있다. 해당 창에서 실행된 코드에 대한 검사 결과를 <그림14>와 같이 xml파일 형식으로 추출할 수 있다.



<그림 13> 테스트 검사 결과 확인

<그림 14> 코드검사 결과 내보내기

4.3 정적/동적시험 결과 분석

정적시험은 소스코드 내 결함이 존재하지 않아야 한다. 따라서 신뢰성 시험결과의 분석을 통해 <그림 1>과 같은 절차를 반복하여 결함을 수정하였다.

동적시험 결과는 크게 2가지 사항이 고려되었다. 첫째, 소프트웨어 등급 A에 대하여 MC/DC를 달성하기 위해서 블록 실행률이 달성된 이후에 추가 테스트 케이스(TC)를 도출하였다. 둘째, if문 조건을 만족하지 못하는 결함이 103곳이 발견되었으며 이는 소스 코드의 수정이 필요한 것으로 확인 되었다. 본 논문에서 제시하는 동적시험 기준과 같이 Block Coverage의 Test Case와 신뢰성 점검 도구를 통한 추가적인 Test Case를 활용하여 동적 시험을 수행하였고 목표값을 달성함으로서 소프트웨어 신뢰성을 확보할 수 있었다. 동적시험 결과는 <표 11>에서와 같이 확인할 수 있다.

<표 11> CSC별 동적시험 결과

CSC명	SW 등급	함수 수	대상함수 개수	결함수	추가 TC
지상OO	A	123	103	0	3
운용OO	A	1870	1,622	0	7
DDS OO	A	32,054	5,254	0	35
운용통제OO	A	3,573	3,310	101	24
초기화OO	A	60	47	0	6
체계OO	A	322	279	1	11
실시간OO	B	8	0	0	0
비행체 OO	A	220	170	0	7
비행OO	B	657	622	0	0
비행상태OO	A	666	540	0	2
통신OO	A	403	388	0	1
데이터OO	B	596	466	1	0
경고 OO	C	313	285	0	0

또한, Visual Studio 2012에서 지원하지 않는 환경적 제약으로 인해 동적시험 불가능한 함수는 시험에서 제외되었다. Visual Studio 2012에서 지원하지 않는 환경적 제약은 CLR제공파일인 경우, 함수 구현부가 없는 .cs 파일인 경우, Public Calss가 아닌 Class의 단위 시험인 경우, mscorlib.dall, System.dll 등의 코드 격리 기능이 제한된 경우, Resource 접근하는 경우이다.

본 연구에서 제안한 C# 프로그래밍 소프트웨어 대한 정적 및 동적 신뢰성 기준을 적용하여 시험한 결과 방위사업청에서 제시하고 있는 정적 시험의 코딩규칙과 취약점 점검에 대한 기준을 충족하였다. 또한, 동적 시험의 경우는 방위사업청이 제시하는 기준을 만족하지 못하는 소스코드를 검출함으로서 본 논문이 제시하는 기준이 적합함을 확인할 수 있었다.

Ⅴ. 결론

본 연구는 무기체계 중 개발언어 C#을 사용하여 개발되는 소프트웨어 대하여 무기체계 소프트웨어 개발 및 관리 매뉴얼에서 제시하고 있는 신뢰성 시험(정적시험, 동적시험) 기준을 적용할 수 있는 방안을 제시하였으며 사례를 통해 그 기준의 적절성을 살펴보았다.

C# 언어 특성 상 Microsoft사의 Visual Studio 2012 환경의 고트분석, 메트릭 측정 및 블록 커버리지 등의 분석 기능을 신뢰성시험의 기준으로 적용하는 방안을 제시하였다. 정적시험은 C/C++ 공통 규칙들을 C# Code Convention과 매핑한 코딩규칙 기준, 소스코드의 런타임 애러를 유발할 수 있는 C# Code Convention 항목을 적용한 취약점 점검 기준 및 Visual Studio에서 제공한 메트릭을 활용한 소스코드 메트릭 점검기준을 제시하였으며, 동적시험은 문장실행율, 분기실행률 및 MC/DC를 Visual Studio의 Block Coverage와 MC/DC에 적용한 기준과 목표값을 제시하였다. C# 소프트웨어의 동적시험은 사전에 SW 등급을 분류하고 시험대상과 적용기준을 설정하고 수행되어야한다.

본 연구는 C# 프로그래밍 무기체계 소프트웨어의 신뢰성시험에 대한 세부기준이 마련되어 있지 않아 그 기준에 대한 방안을 제시하였고 사례를 통해 기준이 적절함을 확인할 수 있었다. 따라서 본 연구에서 제안한 C# 프로그래밍 무기체계 소프트웨어 신뢰성 기준은 C# 소프트웨어의 신뢰성 확보여부를 확인하기 위한 세부 기준을 수립할 수 있는 계기를 마련하였다.

향후 연구에서는 소프트웨어 신뢰성 시험 및 평가 분야의 발전을 위해 제시한 기준이 실제 적용이 가능하도록 국방의 관련분야들과 공동 연구가 필요하다.

참고문헌

[1] K. Y. Kwon, J. S. Joo, T. S. Kim, J. W. Oh, J. H. Beak, “A Study on Quality Assurance of Embedded Software Source Codes for Weapon Systems by Improving the Reliability Test Process,“ Journal of KIISE, Vol.42, No.7, 2015, pp. 860-867.

[2] J. W. Kim, Y. I. Bok, J. H. Lim, “The case study of software reliablilty assesment process based on IEEE Std. 1633l,“ KIIE, Autumn Conference, 2011, pp. 953-960.

[3] 김희철, “다항 위험함수에 근거한 NHPP 소프트웨어 신뢰성장모형에 관한 연구,“ 디지털산업정보학회 논문지, 제7권, 제4호, 2011, pp. 7-14.

[4] 신현철·김희철, “로그형 평균값함수를 고려한 소프트웨어 신뢰성모형에 대한 비교연구,“ 디지털산업정보학회 논문지, 제10권, 제4호, 2014, pp. 19-26.

[5] DAPA, Weapon system Software development and management manual, appendix 6, appendix 7, 2016.

[6] Alan Page, Ken Johnston, Bj Rollison, How We Test Software at Microsoft, Microsoft press. Washington, 2009, pp. 98-107.

[7] Patrick Desjardins, Visual Studio Condensed, Apress, 2013, pp. 137-141.

[8] Subashni, S., Satheesh Kumar, N., Software Testing using Visual Studio 2012, PACKT enterprise Ltd., Brimingham, 2013, pp. 90-142.

[9] Microsoft Corporation, Better Unit Testing with Microsoft Fakes(RTM) v1.2, 2013, pp. 12-16.

[10] LDRA, User‘s Manual for TBrun, LDRA Ltd., 2012.

■ 저자소개 ■

신 봉 득
Shin Bongdeug

2009년 3월~현재
광운대학교 전자통신공학과 박사과정
2006년 1월~현재 방위사업청 공군중령 재직 중
2005년 2월 연세대학교 전기전자공학과(공학석사)
1995년 2월 공군사관학교 항공공학과(공학사)

관심분야 : 무선통신, 레이더신호처리
E-mail : shinbd04@gmail.com

오 혁 준
Oh Hyukjun

2004년 3월~현재
광운대학교 전자통신공학과 교수
1999년 2월 한국과학기술원 전기및전자공학과 (공학박사)
1995년 2월 한국과학기술원 전기및전자공학과 (공학석사)
1993년 2월: 한국과학기술원 전기및전자공학과 (공학사)
관심분야 : 무선통신, 통신신호처리, 데이터신호처리
E-mail : hj_oh@kw.ac.kr

논문접수일 : 2016년 10월 19일
수 정 일 : 2016년 11월 2일
게재확정일 : 2016년 11월 9일