KIPS Tr. Software and Data Eng.
Vol.7, No.7 pp.249~258 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2018.7.7.249

Analysis on Dynamic Software Defects for Increasing Weapon System Reliability

Jihyun Park† · Byoungju Choi††

ABSTRACT

The importance of software in military weapon systems is increasing, and the software structure is becoming more complicated. We therefore must thoroughly verify its reliability. In particular, the defects from the interaction of the software components that make up the weapon system are difficult to prevent only with static testing and code coverage level dynamic testing. In this paper, we classify dynamic software defect types and analyze the issues reported in the Open Source Software (OSS) used in the US department of defense weapon systems. The dynamic defects classified in this paper usually occur after integration, and it is difficult to reproduce and identify the cause. Based on this analysis, we come to the point that the software integration test must be enhanced in order to verify the reliability of the weapon system.

Keywords : Dynamic Software Defects, Defense Weapon Systems, Open Source Software

국방 무기체계 소프트웨어 신뢰성 향상을 위한 소프트웨어 동적 결함 분석

박 지 현† · 최 병 주††

요 약

국방 무기체계에서의 소프트웨어 비중이 증가하고 있으며 구조도 점차 복잡해지고 있어, 이에 따른 신뢰성 검증이 매우 중요하다. 특히 무기체계 시스템을 구성하는 소프트웨어 구성 요소 들의 상호 작용에 따라 발생하는 결함은 정적 시험 및 코드 실행률 점검 수준의 동적 시험만으로는 예방하기가 어렵다. 본 논문에서는 소프트웨어의 동적 결함 유형을 분류 개발한다. 미 국방부 무기체계에 사용하는 오픈 소스 소프트웨어(OSS)에서 보고된 이슈 분석을 통하여 본 논문에서 분류한 동적 결함이 실제로 발생하며, 이들은 대체로 통합 이후 발생하는 결함이며, 재현이 어렵고, 결함 원인 파악이 어려웠음을 보인다. 이 분석 결과를 기반으로 무기체계 신뢰성 검증을 위하여 소프트웨어 통합 시험 개선의 중요성을 도출한다.

키워드 : 소프트웨어 동적 결함, 무기체계 소프트웨어, 오픈 소스 소프트웨어

1. 서 론

무기체계 소프트웨어는 무기체계에 탑재되어 무기체계를 제어하고 동작시키는 소프트웨어이다. 무기체계는 정해진 시간 안에 임무를 완수하여야 하고 어떠한 상황에서도 중단 없이 동작하여야 하므로 고 신뢰성이 요구된다[1]. 특히 무기체계에서 소프트웨어가 차지하는 비중이 증가하고 있으며 구조도 점차 복잡해지고 있어, 무기체계 소프트웨어에 대한 신뢰성은 반드시 검증되어야 한다. 이를 위해서는 무기체계 소프트웨어의 개발 단계별로 적합한 검증이 필요하다[2].

시스템, 소프트웨어, 그리고 하드웨어의 검증(V&V, Verification and validation)에 관한 표준에 따르면 소프트웨어의 구현 이후 소프트웨어 구조 V&V, 소프트웨어 통합 V&V를 수행하도록 한다[3]. 소프트웨어의 구조 V&V는 소프트웨어의 단위 시험에 해당하는 부분으로 이 때 수행해야 하는 활동은 소스 코드 및 문서를 이용한 정적 분석과 컴포넌트 시험이 있다[4, 5]. 소프트웨어 통합 V&V에서는 소프트웨어 컴포넌트들이 제대로 통합되었는지를 검사하는 통합 시험을 수행하여야 한다.

무기체계 소프트웨어 개발 프로세스에 따르면 소프트웨어 신뢰성 시험을 통한 소프트웨어 검증을 반드시 수행하도록

※ 본 연구는 방위사업청과 방위산업기술지원센터의 지원 (계약번호: UC160002D)하에 수행되었음.
† 준 회 원 : 이화여자대학교 컴퓨터공학과 박사과정
†† 정 회 원 : 이화여자대학교 컴퓨터공학과 교수
Manuscript Received : February 26, 2018
First Revision : March 16, 2018
Accepted : April 20, 2018
* Corresponding Author : Byoungju Choi(bjchoi@ewha.ac.kr)

되어 있다[6]. 무기체계 소프트웨어의 신뢰성 시험은 소프트웨어 정적 시험과 동적 시험으로 구성한다. 정적 시험에서는 소프트웨어를 실행하지 않은 상태에서 잠재적인 결함 검출을 위하여 코딩 규칙 검증, 취약점 점검 및 소스 코드 메트릭 점검을 수행하고, 동적 시험에서는 요구 사항을 기반으로 실제 하드웨어(Target)에 탑재하여 코드의 문장(statement), 분기(branch) 실행률, MC/DC (Modified Condition/Decision)등의 실행률(coverage)을 점검한다.

문제는 무기체계 소프트웨어 동적 시험이 시스템을 구성하는 모듈을 단계적으로 통합하여 검증을 수행하는 방식이 아니라 전체 개발 소프트웨어를 단번에 묶어(일명 빅뱅 통합) 검증하는 방식이라는 점이다. 즉, 현재 무기체계 소프트웨어는 소프트웨어 시험 표준의 단위 시험에 해당하는 부분은 기준이 수립되어 있으나 통합 시험 프로세스에 대해서는 제대로 된 기준이 수립되어 있지 않다. 개발 이후 타겟에서의 동적 시험은 스터브(stub)와 드라이버(driver) 등의 장치(test harness)가 별도로 필요 없다는 매력적인 점은 있다. 그러나, 단위 소프트웨어(CSU), 구성품(CSC) 및 소프트웨어 형상항목(CSCI)이 통합되어 서로 호출하며 실행하는 과정에서 발생하는 결함에 대한 소프트웨어 통합단계의 신뢰성 검증 방식으로는 한계가 있다는 뜻이다. 특히, 소프트웨어 시험에 관한 국제표준에 따르면, 코드의 문장, 분기, MC/DC 는 단위 시험을 위한 실행률 기준이며, 통합시험을 위한 기준은 아니라는 점에서도 알 수 있다[5].

실제로도 무기체계 소프트웨어의 개발 후반에서 결함이 발생하고 있다[7]. USS Vincennes 함 사건이나 패트리어트 방어 체계 오류와 같은 결함이 대표적이다[8]. 이 때 발생하는 결함들은 현재의 전체 소프트웨어를 대상으로 실행하는 빅뱅 통합 스타일의 동적 시험 방식만으로는 탐지하기 어렵고, 탐지되더라도 그 원인 파악 및 해결이 매우 어렵다.

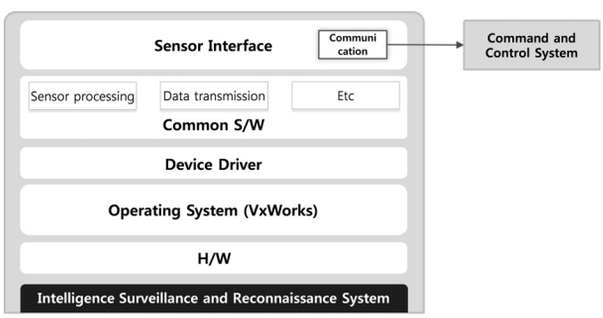
본 논문의 주요 기여점은 **1)소프트웨어의 동적 결함 유형에 대해 분류**하고, **2)이들 결함이 실제로 무기체계 소프트웨어에서 발생하는지 분석**한다. 소프트웨어의 동적 결함은 소스 코드가 아닌 실제 동작하는 프로그램에서 발견할 수 있는 결함이다. 대부분의 동적 결함은 시스템의 고장이나 중단과 같이 심각한 상황을 유발할 수 있으므로 반드시 해결되어야 하는 결함이다. 이러한 결함이 현재의 동적 시험 방식으로 탐지할 수 있는지를 분석함으로써 향후 무기체계 소프트웨어의 신뢰성을 향상시킬 수 있도록 하는 방향을 도출하는데 기대하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 무기체계 소프트웨어의 특성을 분석, 3장에서 동적 결함을 분류하고, 4장에서는 이러한 결함들이 실제로 존재하는지를 분석한다. 5장에서는 무기체계 소프트웨어의 신뢰성을 위한 검증 개선 방안을 제시하고 6장에서는 결론과 향후 연구 방향을 기술한다.

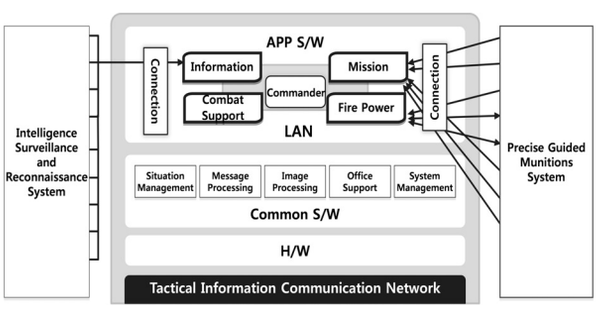
2. 무기체계 소프트웨어의 특성

무기체계라 함은 유도무기・항공기・함정 등 전장에서 전투력을 발휘하기 위한 무기와 이를 운영하는데 필요한 장비・부품・시설・소프트웨어 등 제반 요소를 통합한 것으로서 지휘통제·통신무기체계, 감시정찰무기체계, 기동무기체계, 함정무기체계, 항공무기체계, 화력무기체계, 방호무기체계, 모의분석훈련 소프트웨어/장비 등 그 밖의 무기체계로 분류된다[9]. [9]에 따르면, 무기체계 소프트웨어의 특성은 다음과 같이 요약할 수 있다.

- 실시간성: 정해진 시간 안에 임무를 완수해야 함
- 신뢰성: 어떠한 상황에서도 중단 없이 동작
- 효율성: 제한된 자원의 최적화된 활용
- 상호운용성: 다양한 소프트웨어 계층 구성요소와 연동
- 안전성: 안전 우선
- 정확성: 목표물을 정확하게 탐지하고 정밀하게 타격하여야 함
- 보안성: 정보가 외부로 유출되지 않아야 함



(A) Embedded software



(B) Battle management system

Fig. 1. Example of Weapon System Software Structure [10]

특히 본 논문에서는 무기체계 소프트웨어가 다양한 계층의 구성요소와 연동되어 동작하는 것에 주목하였다. 무기체계 소프트웨어는 실제 무기체계에 탑재되는 내장형 소프트웨어와 전장관리 정보체계로 구분할 수 있다[9]. 그림 1은 2006년 방위사업청의 C4I 체계에 대한 보도 자료를 바탕으로 작성한 감시정찰체계와 전장관리 정보체계의 구조 예제이다[10]. 내장형 소프트웨어는 실제 무기체계에 탑재되어 동작하는 소프트웨어로 하드웨어와 소프트웨어가 통합된 시스템 관점에서 개발이 된다. 어플리케이션 단독으로는 실행이 불가능하며 Fig. 1(A)와 같이 어플리케이션에서부터 하드웨어에

이르기까지 다양한 계층이 연관되어 동작하므로 내부 구성 요소들 간의 상호 작용이 많이 발생한다.

전장관리 정보체계는 전장 상황에 대한 파악 및 의사결정을 위해 지휘관의 작전 지휘를 지원하는 소프트웨어이다. Fig. 1(B)와 같이 어플리케이션이 하드웨어 독립적으로 동작하나, 외부 센서로 부터의 신호와 같은 외부 체계들과의 상호 작용이 많이 발생한다. 이와 같이 소프트웨어의 목적과 개발 방식이 달라도 내부/외부 구성 요소들 간의 상호 작용이 빈번하게 일어난다는 점은 공통적으로 나타나는 특징이다. 따라서 무기체계 소프트웨어를 검증할 때에도 다양한 구성 요소들 간의 상호 작용과 관련된 결함이 발생하는지를 중점적으로 검증하여야 한다.

3. 소프트웨어 동적 결함

동적 결함이란 시스템이 실행 중인 상태에서 발생하는 결함을 의미한다. 무기체계 소프트웨어의 동적 결함은 무기체계 소프트웨어가 동작하고 있을 때 발생하는 결함이다. 앞서 분석한 무기체계 소프트웨어의 특성 중 다양한 구성요소들 간의 상호 작용과 관련된 주요 결함은 메모리 결함, 병행성 결함, 예외처리 결함이 대표적이다.

메모리 결함은 프로세스 내부의 메모리 할당, 해제 및 접근 과정에서 발생하는 결함이다. 병행성 결함은 한 프로세스 내부에서 멀티 스레드가 수행될 때 스레드 간에서 발생하는 결함과 프로세스 간 자원 공유 및 동기화 과정에서 발생하는 결함이다. 예외처리 결함은 예외가 발생하였으나 예외처리 루틴이 정상적으로 동작하지 않는 경우를 의미한다. 이러한 결함들은 정적 분석을 통해서는 탐지가 어렵고, 탐지가 되더라도 그 원인을 분석하기가 매우 어려운 결함이다. 메모리 누수나 널 포인터 접근과 같은 일부 결함의 경우 정적 분석을 통해 찾아낼 수는 있으나, 무기체계 소프트웨어는 다양한 계층 구조로 이루어져 있고, 내·외부 구성요소들 간에 상호 연동될 때 결함이 발생하는 경우가 많기 때문에 오탐율이 높고, 탐지가 되더라도 원인 파악이 어려운 경우가 많다[11, 12]. 또한 동적 결함 대부분은 실행 환경 및 조건에 따라 발생 여부가 달라지는 경우가 있는데, 이러한 결함들은 원인 파악이 어렵고 결함이 재현되지 않는 경우도 많아 디버깅이 어려운 것이 특징이다.

본 논문에서는 대표적인 결함 군 세 가지에 대하여 관련 자료를 토대로 Table 1과 같이 세부 결함 유형을 분류하였다[13-18].

3.1 메모리 결함

메모리 결함이란 프로세스에서 프로세스 내부의 메모리에 대해 할당/해제/접근하는 과정에서 발생하는 결함을 의미한다[13]. 세부적으로는 11가지 유형으로 분류한다.

메모리 할당 결함은 메모리를 할당하는 과정에서 발생하는 결함이다. 직접적으로 시스템의 다운과 같은 심각한 상황을 야기시키지는 않으나 잠재적으로는 그 원인이 될 수 있는 결함으로 세부 유형은 다음과 같다.

- **메모리 할당 실패**: 메모리를 할당하고자 하였으나 시스템의 일시적인 장애와 같은 이유로 할당이 이루어지지 않아 NULL 값이 반환되는 결함
- **Zero size 메모리 할당**: 메모리를 할당할 때 할당 크기를 '0'으로 하는 결함
- **메모리 누수**: 동적으로 메모리를 할당한 후, 메모리 해제를 하지 않고 종료함으로써 프로세스가 종료된 이후에도 메모리 자원을 점유하게 되는 결함

메모리 해제 결함은 메모리를 해제할 때 발생하는 결함으로 대부분 시스템 다운을 발생시킨다. 세부 유형은 다음과 같다.

- **NULL 포인터 해제**: 메모리 해제를 실행할 때 해제하고자 하는 값이 NULL인 결함
- **해제된 메모리 재 해제**: 이미 해제된 메모리에 대해서 다시 해제하고자 하는 결함
- **할당되지 않은 포인터 해제**: 해제하고자 하는 메모리 포인터 주소가 동적으로 할당된 메모리 주소가 아닌 결함

메모리 접근 결함은 메모리의 값을 읽거나 쓰는 것과 같이 메모리를 사용할 때 발생하는 결함이다. 이 결함 유형 역시 시스템 다운을 발생시킬 수 있는 심각한 결함으로 세부 유형은 다음과 같다.

- **NULL 포인터 접근**: 사용하고자 하는 메모리 주소 값이 NULL인 결함
- **해제된 포인터 접근**: 사용하고자 하는 메모리 주소 값이 이미 해제된 메모리인 결함
- **할당되지 않은 포인터 접근**: 동적으로 할당되지 않고 유효한 주소가 아닌 메모리 공간을 사용하고자 하는 결함
- **할당된 범위를 벗어난 메모리 접근**: 동적으로 할당된 메모리 크기를 벗어난 메모리 공간에 접근하고자 하는 결함
- **다른 변수의 메모리 공간과 충돌**: 접근하고자 하는 메모리 주소의 크기가 동적으로 할당된 메모리 크기보다 커서 다른 변수의 메모리 공간을 침범하는 결함

메모리 결함 11가지 유형은 단일 어플리케이션에서의 메모리에 대한 결함으로 실행 기반의 동적 시험을 통해서 탐지할 수 있다. 앞에서도 언급한 바와 같이 일부 메모리 결함의 경우 정적 분석을 통해 탐지가 되기도 하나 소스 코드나 바이너리 코드의 분석만으로는 오탐율이 높을 뿐만 아니라 탐지가 되더라도 문제의 원인을 찾기 힘들다.

3.2 병행성 결함

병행성 결함은 멀티스레드, 멀티프로세스, 다중 CPU 환경에서 서로 상호작용하는 과정에서 발생하는 결함이다[14]. 공유되는 자원의 사용 및 동기화와 관련된 결함으로 크게 데드락, 원자성 위반, 순서 위반으로 분류할 수 있으며 좀 더 세분화하여 Table 1의 21가지 결함 유형으로 분류한다.

데드락이란 두 개 이상의 스레드가 대기 대기 상태이며, 각 스레드가 서로 다른 스레드가 가진 자원을 기다리거나 작

Table 1. Dynamic Software Defects

Category	Defect			ID
Memory	Memory allocation		Memory allocation failure	WF_M_001
			Zero-size memory allocation	WF_M_002
			Memory leak	WF_M_003
	Memory deallocation		NULL pointer free	WF_M_004
			Duplicated free	WF_M_005
			Unallocated pointer free	WF_M_006
	Memory access		NULL pointer access	WF_M_007
			Released pointer access	WF_M_008
			Unallocated pointer access	WF_M_009
			Out of range memory access	WF_M_010
			Conflict with other variables	WF_M_011
Concurrency	Deadlock	Resource deadlock	Synchronized object deadlock	WF_C_001
		Message deadlock	Synchronized object receive failure	WF_C_002
			Synchronized object processing failure	WF_C_003
			Synchronized object send failure	WF_C_004
			Loss of synchronized message transmission	WF_C_005
			Asynchronous message transmission failure	WF_C_006
			Asynchronous message processing failure	WF_C_007
	Atomicity violation	Variable	Variable use atomicity violation	WF_C_008
		Memory read/write	Perform a write operation on other threads while performing a read operation on shared memory	WF_C_009
			Perform write operation on other threads while performing read after write operation on shared memory	WF_C_010
			Perform write operation on other threads while performing read after write operation on shared memory	WF_C_011
			Perform read operation on other threads while performing a write operation on shared memory	WF_C_012
			Perform write operation on other threads while performing a write operation on shared memory	WF_C_013
	O.rder violation	Uncreate resource	Using uncreated objects	WF_C_014
			Using uncreated shared memory	WF_C_015
		Uninitialized resource	Using uninitialized objects	WF_C_016
			Using uninitialized shared memory	WF_C_017
		Released resource	Using released objects	WF_C_018
			Using released shared memory	WF_C_019
		Shared memory access	Read and write operation on shared memory from different threads	WF_C_020
			write operations on shared memory from different threads	WF_C_021
Exception handling	Device driver exception handling	Device connection	Device connection failure	WF_E_001
			Device open failure	WF_E_002
		Device disconnection	Device disconnection failure	WF_E_003
			Device close failure	WF_E_004
		No Data	Fail to read data from device	WF_E_005
			Fail to write data to device	WF_E_006
			Fail to seek data on the device	WF_E_007
			Fail to transfer IOCTL data to/from device	WF_E_008
		Illegal Data	Reading illegal data from the device	WF_E_009
			Writing illegal data to the device	WF_E_010
			Seeking illegal data on the device	WF_E_011
			Transferring illegal IOCTL data to/from device	WF_E_012
		Power	Fail to power on device	WF_E_013
			Fail to power down device	WF_E_014
	OS structured exception handling		Interrupt handler exception handling failure	WF_E_015
	Application structured exception handling		Try-catch exception handling failure	WF_E_016

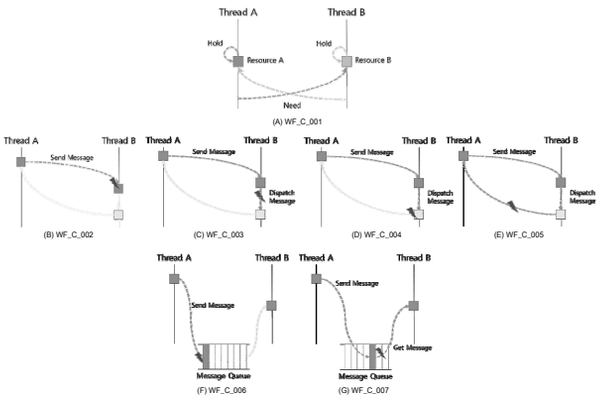


Fig. 2. Types of Deadlock Defects

업의 종료를 기다리고 있어 결과적으로는 아무것도 하고 있지 않은 상태를 의미한다[16]. 데드락은 주로 공유 자원 사용을 위한 동기화 객체인 소프트웨어 잠금(Lock) 혹은 하드웨어 잠금을 잘못 사용하거나, 스레드 간 통신 과정에서 시그널(Signal) 혹은 메시지(Message)가 손실되거나 손상되어 잘못된 값이 전달될 때 발생한다[17, 18].

- **동기화 객체 데드락**: Fig. 2(A)와 같이 동기화 객체의 잘못된 사용으로 인해 두 개 이상의 스레드가 무한 대기하는 결함
- **통신(메시지) 데드락**: Fig. 2(B)~(G)와 같이 스레드 간 통신 과정에서 시그널 혹은 메시지가 손실되거나 손상되어 잘못된 값이 전달되는 결함

멀티스레드 환경에서의 원자성이란 특정한 실행 단위가 다른 스레드로부터 방해 받지 않고 실행되는 것을 의미한다. 원자성 위반은 함께 수행되어야 하는 두 가지 이상의 작업이 안전하게 함께 수행되지 못하고 다른 스레드로부터 침범받아 데이터가 변경되는 것이다[14]. 이러한 원자성 위반은 한 스레드 내부에서 하나의 연산처럼 함께 수행되어야 할 read/write 연산 수행 도중 다른 스레드에서 끼어들어 동일한 대상, 즉 메모리나 변수에 대해 read 혹은 write 연산을 수행할 때 발생한다.

- **변수 사용 원자성 위반**: 특정한 변수에 대한 접근이 종료되기 전에 다른 스레드에서 끼어들어 동일한 변수에 대한 연산을 수행하는 결함
- **메모리 사용 원자성 위반**: Fig. 3과 같이 여러 스레드가 공유하는 메모리에 대해서 한 스레드의 접근이 종료되기 전에 다른 스레드에서 끼어들어 동일한 메모리에 대한 연산을 수행하는 결함

순서 위반은 멀티스레드 환경에서 개발자가 의도한 순서가 제대로 지켜지지 않아 발생하는 결함이다[14]. 멀티스레드 환경에서는 스레드의 실행 순서가 정해져 있지 않기 때문에 특정한 자원(객체나 공유 메모리)에 대한 생성/해제 및 접근이 서로 다른 스레드에서 일어날 경우 프로그래머가 의도하지 않은 순서로 실행될 수 있다. 순서 위반은 자원에 대한 생성, 초

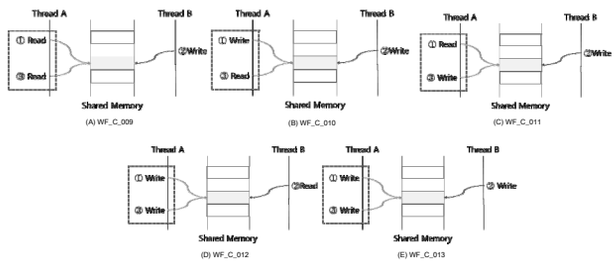


Fig. 3. Atomicity Violation

기화, 해제 및 접근과 관련한 순서 위반으로 나눌 수 있다.

- **생성되지 않은 자원 사용**: Fig. 4 (A), (B)와 같이 한 프로그램에서 동기화 객체나 공유 메모리와 같은 자원에 대해 사용하고자 할 때에는 반드시 해당 객체나 메모리에 대한 생성 및 초기화 과정이 수행되어야 하는데, 생성되지 않은 자원에 접근하여 시스템에 심각한 장애를 유발하는 결함
- **초기화 되지 않은 자원 사용**: Fig. 4 (C), (D)와 같이 생성은 되었으나 초기화되지 않은 자원에 접근하여 시스템에 심각한 장애를 유발하는 결함
- **해제된 자원 사용**: Fig. 4 (E), (F)와 같이 멀티스레드 환경에서 자원 사용이 종료되기 전에 자원을 해제한 후, 해제된 자원을 사용하여 시스템에 심각한 장애를 유발하는 결함
- **공유 메모리 접근 순서 위반**: 공유 메모리에 대한 접근 순서 위반은 공유 메모리를 두 개 이상의 스레드가 순차적으로 접근할 때 최소한 하나의 접근은 쓰기 연산인 경우, 읽기 연산과 쓰기 연산이 순차적으로 수행되어야 하는데 그 순서가 바뀌거나 공유 메모리에 대한 쓰기 연산이 각각 수행되어 먼저 수행된 쓰기 연산이 이후의 수행에 영향을 미치지 못하는 결함

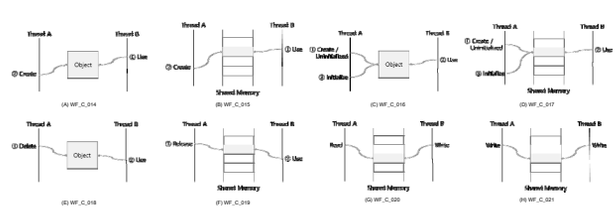


Fig. 4. Order Violation

병행성 결함은 실행 상황에 따라 결함의 발생 여부가 달라지므로 정적 분석으로는 탐지가 어려우며 동적 검증을 통해 탐지하여야 한다.

3.3 예외처리 결함

예외처리란 소프트웨어가 동작하는 동안 특정한 문제나 오류가 발생하였을 때 정상 처리를 중단하고 실행 흐름을 바꿔 다른 처리를 하는 행위를 의미한다. 무기체계 소프트웨어는 다양한 계층으로 이루어져 있어, 각 계층 별로 예외가 발생할 수 있으며 그에 따른 예외처리도 달라지게 된다. OS 계층에서 예외가 발생할 경우에는 OS 예외처리 핸들러가 동작한다. 장치 드라이버 계층에서는 장치의 입출력 관리자가 예

외처리를 하고, 어플리케이션 계층에서 예외가 발생할 경우에는 사용자가 정의한 예외 처리 루틴이 동작하게 된다. 예외처리 결함은 각 계층별 예외가 발생하였을 때 적절한 예외처리가 이루어지지 않는 결함이다[15].

디바이스 드라이버 예외처리 결함은 디바이스의 연결/해제 혹은 디바이스 사용 과정에서 발생할 수 있는 예외 상황에 대해 적절한 예외처리가 되지 않은 결함이다.

- **장치 연결**: 시스템에 외부 장치가 처음 인식되면 장치에 대한 초기화 과정을 수행하는데, 그 과정에서 Fig. 5(A), (B)와 같이 장치 초기화가 실패하거나 장치를 여는데 실패하였으나 적절한 예외처리가 되지 않은 결함
- **장치 연결 해제**: 외부 장치에 대한 연결을 해제하고자 할 때는 장치를 위해 할당한 자원을 해제하고 장치와의 연결을 차단하게 되는데, 그 과정에서 Fig. 5(C), (D)와 같이 장치 연결 해제가 실패하거나 장치를 닫는데 실패하였으나 적절한 예외처리가 되지 않은 결함
- **No Data**: Fig. 5(E)~(H)와 같이 장치에 대한 읽기/쓰기/탐색/IOCTL 연산을 수행할 때 어플리케이션과 장치가 주고받는 데이터가 NULL이나 적절한 예외처리가 되지 않은 결함
- **Illegal Data**: Fig. 5(I)~(L)와 같이 장치에 대한 읽기/쓰기/탐색/IOCTL 연산을 수행할 때 어플리케이션과 장치가 주고받는 데이터가 잘못된 값이나 적절한 예외처리가 되지 않은 결함
- **전원**: 외부 장치의 전원에 대한 제어가 가능한 경우 Fig. 5(M), (N)과 같이 장치에 대한 전원 인가 혹은 종료를 수행하였으나 실패하였고 적절한 예외처리가 되지 않은 결함

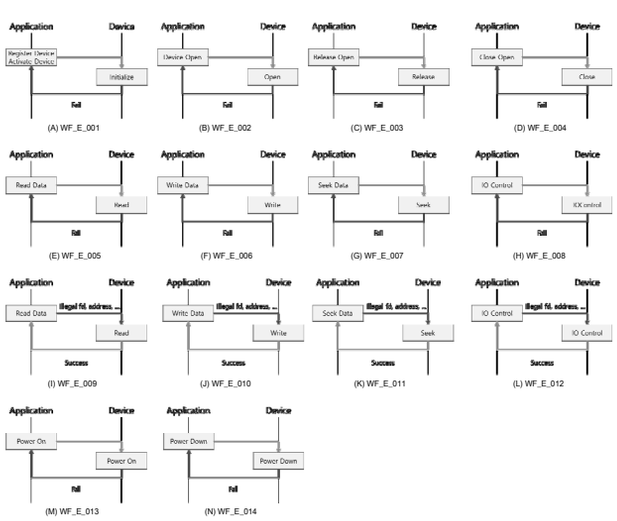


Fig. 5. Device Driver Exception Handling Defects

OS 예외처리 결함이란 OS에서 프로세스나 디바이스와 같은 자원을 관리하고 스케줄링 하는 과정에서 다양한 예외가 발생할 수 있는데, 예외가 발생하였음에도 적절한 예외처리를 하지 않는 결함이다.

- **구조적 예외처리**: OS에서는 발생할 수 있는 예외 상황에 대해 인터럽트 핸들러를 만들어 적절한 대응을 수행하는데, OS 예외가 발생하였음에도 적절한 인터럽트 핸들러가 동작하지 않는 결함

APP 예외처리 결함은 어플리케이션에서 지원하는 구조적 예외처리가 예외 상황에서 적절히 수행되지 않은 결함이다.

- **구조적 예외처리**: try/catch 키워드를 지원하는 어플리케이션에서 실행 도중 예외가 발생하여 throw 문이 호출되었으나, 적절한 catch문이 수행되지 않는 결함

4. 소프트웨어 동적 결함 분석

본 논문에서 분류한 표1의 동적 결함이 실제 무기체계 소프트웨어에서 얼마나 실재하는지를 파악하기 위하여 무기체계 소프트웨어에서 보고된 결함 이슈를 분석하였다.

4.1 분석 대상 무기체계 소프트웨어

본 논문에서는 미 국방부에서 무기체계에 사용하는 오픈 소스 소프트웨어(OSS) 중 6가지를 분석 대상으로 선정하였다[19]. 분석 대상OSS의 세부 명세는 다음 Table 2와 같다. 분석 대상 OSS는 모델링 혹은 시뮬레이션을 위한 소프트웨어로써 무기체계에 탑재되어 돌아가는 전체 소프트웨어는 아니다. 모델링 소프트웨어는 탱크와 같은 실제 무기를 모델링하기 위해 사용되기도 하나 Fig. 1(b)에 영상처리를 위한 구성 요소로도 사용된다. 또한 시뮬레이션 소프트웨어는 훈련용 소프트웨어로써 사용되며, GPS 신호와 같은 실제의 외부 입력을 받아 실제 상황처럼 훈련할 수 있도록 해 준다. 이와 같이 분석 대상 OSS는 무기체계를 구성하는 구성 요소들 중 하나로 다양한 외부 센서나 UI, DB와 같은 다른 구성 요소들과의 상호 작용을 한다는 점에서 분석의 의미가 있다.

4.2 분석 결과

2의 분석 대상 6종 OSS 오픈 소스 소프트웨어에 대한 오픈 소스 커뮤니티에 등록된 총 955 이슈를 Table 1의 결함과 비교 분석하였다. 정확한 원인 분석을 위하여 해결이 완료된 이슈 788건을 대상으로 하였고, 오픈 소스 커뮤니티에 등록된 결함 내용뿐만 아니라 결함이 해결되어 실제 코드 릴리즈에 반영된 내용을 함께 분석하였다. 다만 Table 1의 소프트웨어 동적 결함 중 디바이스 드라이버에 대한 예외처리 결함(WF_E_001~WF_E_014)은 OSS 어플리케이션 계층의 이슈 분석을 통해서는 확인하기 어려우므로 분석 대상 결함에 대해서는 제외하였다.

1) 대상OSS의 결함 이슈 분포

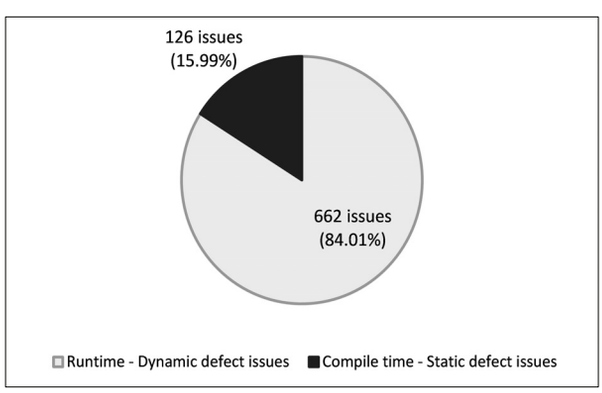
무기체계 소프트웨어 OSS에 대한 955건의 결함 이슈 중 원인이 아직 미해결된 “Open” 건수는 167건이다. 해결된 총 “Closed” **788 이슈를 분석하였을 때**, 소프트웨어가 동작하는 동안에 발생하는 런타임 동적 결함 이슈는 Fig. 6(A)에서처럼 전체의 84.01%인 662건이었고, 나머지는 컴파일 타임에

Table 2. Weapon System Open Source Software

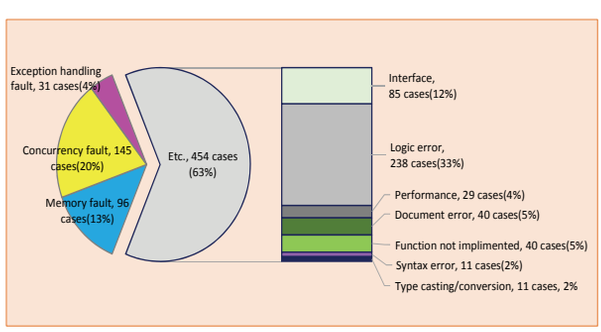
OSS: Open Source Software, R: Runtime, Cm: Compile time, O: Open, C: Close (단위: 건수)

OSS	Description	Supported platform	Issue	Open /Close	Time of occurrence	Non-reproduced
BRL_CAD	Solid modeling system used for modeling weapons such as tanks	BSD, Linux, Solaris, Mac OS X, Windows	344	102 /242	R 258 /Cm 86	O 4 /C 4
VSIPL	A vector, signal and image processing library that supported high-performance parallel processing of embedded weapon systems	BSD, Linux, Solaris, Mac OS X, Windows	5	1 /4	R5 /Cm 0	O 0 /C 0
Delta 3D	Game/simulation engine for simulation software	Windows, OS X, Linux	148	33 /115	R 126 /Cm 22	O 2 /C 5
OSSIM	Library for terrain image processing	Windows, Solaris, Linux	13	4 /9	R 6 /Cm 7	O 0 /C 0
Opticks	Software platform for remote sensing and image analysis	Windows, Solaris, Linux	372	0 /372	R 359 /Cm 13	O 0 /C 5
OMAR	Terrain image(used in OSSIM) management software using relational database	Windows, Solaris, Linux	73	27 /46	R 63 /Cm 10	O 0 /C 3
Total			955	167 /788	R817 /Cm138	O6 /C17

발견할 수 있는 정적 결함이었다. 동적 결함 이슈에 등록된 내용을 좀 더 상세히 분류를 하면, Fig. 6(B)와 같이 메모리 결함이 96건, 병행성 결함이 145건, 예외처리 결함이 31건이었고, 기타 결함이 454건으로 총 726건으로 집계되었다. 여기서 이슈 하나에는 다수의 결함이 있는 경우가 있어서, **662 이슈로부터 726 결함 건**이 파악되었다.



(A) Defect issue distribution



(B) Defect type distribution for dynamic defect issues

Fig. 6. Defect Issue Distribution of Target OSS

2) 동적 결함 발생 분포

Fig. 6(B)의 기타 결함을 제외한 메모리, 병행성, 예외처리 결함 군을 대상으로 Table 1의 소프트웨어 동적 결함 각 결함이 실제 얼마나 존재하는지 상세 분류한 것이 Fig. 7이다. 예외처리 결함에서는 앞서 언급하였듯이 본 적용대상 OSS가 어플리케이션 소프트웨어이므로 디바이스 드라이버 관련 예외처리는 관련이 없음을 알 수 있다. Fig. 7에서 볼 수 있듯이 Table 1의 모든 결함이 무기체계 소프트웨어 대상 OSS의 결함 이슈에 골고루 존재함을 알 수 있다. 개별 결함으로는 “Try-catch 예외처리 결함(WF_E_016)”이 29건, 할당되지 않은 포인터 해제(WF_M_006) 메모리 결함이 20건, “할당되지 않은 포인터 접근(WF_M_009)” 메모리 결함 16건, “생성되지 않은 공유 메모리 사용(WF_C_015)”으로 인한 병행성 결함 15건 순이었으며, 결함 군별로는 병행성 결함 군이 총 145건으로 가장 많이 발생하였다.

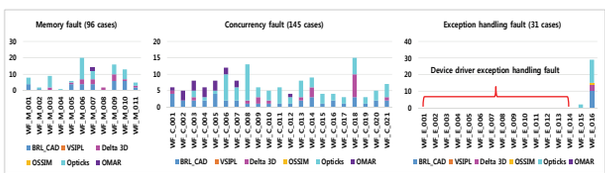


Fig. 7. Table 1 Dynamic Software Defect Occurrence Distribution

3) 결함 발견 가능 시점

Fig. 6(B)의 총 766개 동적 및 기타 결함을 언제 최초 발견 가능한 지 여부를 분석하였다. 이를 위하여 개발문서 검토단계, 정적 검증단계, 소프트웨어 단위시험단계, 소프트웨어 통합및시스템 시험 단계로 구분하여 분석한 것이 Table 3이다. 기타결함은 대부분 단위 시험 이전 설계 단계 검증에서 발견할 수 있는 것이다. 주목할 것은 분석 대상 OSS의 결함 이슈

Table 3. Test Level that Can Detect Defects for the First Time

Test level \ Defects	Dynamic defects			Etc. defects						
	Memory	Concurrency	Exception handling	Interface	Logic error	Performance	Document error	Function not implemented	Syntax error	Type casting/conversion
Review documents	0	0	0	92	12	1	37	40	0	2
Static verification	1	0	3	11	227	0	0	0	110	3
Unit test	1	0	9	2	4	6	3	0	0	7
Integration & system test	94	145	19	2	0	22	0	0	0	0

로 기록된 동적 결함들은 대부분 단위시험 이후 특히 통합 및 시스템 시험단계에서 발생 및 발견이 가능하였다는 점이다.

4) 재현 불가 결함

Table 2의 재현이 불가능한 이슈가운데, 결함이 해결된 이슈(closed issue)가 17건, 미해결된 이슈(open issue)가 6건으로 총 23이슈가 보고되었다. 이 가운데, 해결된 17건의 이슈에서 발생한 결함은 총 37건으로, 동적 결함이 36건, 기타결함으로 인터페이스 정적 결함이 1건이었다. 동적 결함 36건을 Table 1의 세부 결함 별로 나타낸 것이 Fig. 8이다. 메모리 결함 2건 이외 대부분이 병행성 결함이 재현이 안된 것으로 보고되었음을 알 수 있다.

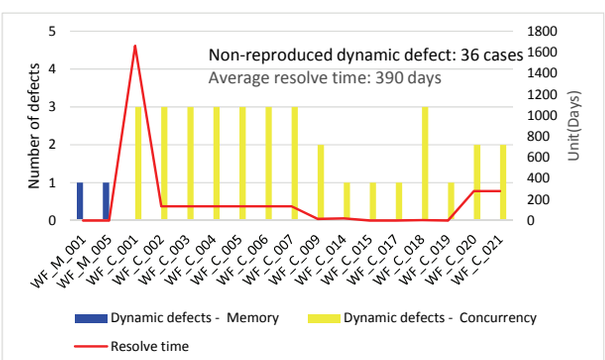


Fig. 8. Dynamic Defect Distribution and Debugging Time of Non-Reproduced Issues

재현 불가인 이슈가 해결되기 까지 소요된 시간도 보고되어 있었다. 이를 Fig. 8의 오른쪽 y축에 나타내었는데, 가장 오래 걸린 것은 “동기화 객체 데드락(WF_C_001)” 병행성 결함으로 1661일로 4년 5개월이었고, 전체적으로 평균 390일이었다. 대체로 동기화 관련 병행성 결함이 다수 원인이었고 결함 해결하는 데 대략 3일 정도 걸린 것을 알 수 있었다.

5. 무기체계 소프트웨어 동적 시험

서론에서도 언급한 바와 같이 현재 무기체계 소프트웨어는 소프트웨어 시험 표준의 단위 시험에 해당하는 부분은 기준이 수립되어 있으나 통합 시험 프로세스에 대해서는 제대로 된 기준이 수립되어 있지 않다. 특히 현재 무기체계 소프트웨어 개발 프로세스[6]에서 주목할 점은 서론에서 언급하였듯이 무기체계 소프트웨어 동적 시험이 전체 개발 소프트웨어를 단번에 묶어 검증하는 방식이라는 점이다. [6]에 따르면 동적 시험에서는 요구사항 기반으로 코드의 문장, 분기 실행률, MC/DC 실행률을 점검한다. 그러나 이러한 방식은 엄밀히 말하면, 전체 소프트웨어를 대상으로 실행하는 단위시험 실행률 측정이므로, 단위 소프트웨어(CSU), 구성품(CSC) 및 소프트웨어 형상항목(CSCI)의 상호 작용에 따라 통합 시험 단계에 발생하는 결함에 대한 신뢰성 검증 방식으로는 한계가 있다. 각 소프트웨어 구성 요소들이 단계적으로 결합하며 상호 작용하는 과정에서 발생할 수 있는 결함을 찾아내기 어렵고, 결함을 찾아내더라도 그 원인을 찾기 어렵기 때문이다. 따라서 기존의 검증 프로세스에 통합 시험에 대한 평가 기준의 추가가 필요하다.

4절에서 무기체계 오픈 소프트웨어 OSS 6종의 결함 이슈에서 파악된 동적 결함은 다양한 외부 센서나 UI, DB와 같은 다른 구성 요소 CSC 및 CSCI들과의 상호 작용으로 통합 이후 발생하는 결함임을 알 수 있다. 이 가운데에는 재현이 불가하여 결함 원인파악에 수일에서 수개월까지도 걸릴 수 있다는 점이다. 현재 무기체계 소프트웨어 산업현장에서 [6]을 준수하며 이루어지는 전체 개발 소프트웨어를 대상으로 하는, 특히 현재의 동적 시험 검증방식으로는 이러한 문제를 예방하는 데는 한계가 있다.

본 논문에서는 [6]의 무기체계 소프트웨어 검증 방안과 4절의 분석결과를 바탕으로 하였을 때, 통합 시험에 대한 평가 기준 수립에 도움을 줄 수 있는 두 가지 개선 방안을 도출할 수 있었다.

- CSC, CSCI 통합 단계의 상호 작용이 시험 평가 될 수 있도록 올바른 통합 시험 평가 기준의 개선 및 이를 지원하는 도구 적용
- 소프트웨어 동적 결함 발견을 지원하는 디버깅 자동화 도구 적용

대표적인 동적 시험 도구로는 크게 코드 실행률을 측정하는 도구와 결함 발견을 위한 디버깅 도구가 있다. 무기체계 소프트웨어의 실행률 측정 도구로는 LDRA tool, Cantata++, VectorCast, CodeSonar 등이 사용된다[20]. [6]에서는 이러한 도구를 사용하여 요구사항 기반의 문장(statement), 분기(branch) 실행률, MC/DC (Modified Condition/Decision) 실행률을 측정하도록 한다. 그러나 통합 세부 단계 별로 구분하여 올바른 통합 시험이 이루어질 수 있도록 통합 시험 평가 기준의 개선 및 사용 가이드 개발이 필요하다.

소프트웨어 동적 결함에 대한 디버깅 지원 도구로는 Table

Table 4. Dynamic Fault Debugging Tools

	Memcheck	Dr.Memory	Address Sanitizer	Helgrind	Thread Sanitizer	AMOS
Purpose	Fault detection	Fault detection	Fault detection	Fault detection	Fault detection	Fault detection & cause analysis
Target	1 Application	1 Application	1 Application	1 Application	1 Application	System
Test Level	Unit/integrated/system test	Unit/integrated/system test	Unit/integrated/system test	Integrated/system test	Integrated/system test	Integrated/system test
Environment	Virtual machine	Actual execution environment	Actual execution environment	virtual machine	Actual execution environment	Actual execution environment
Defect detection technique	Dynamic binary instrumentation	Dynamic binary instrumentation	Compile time instrumentation	Dynamic binary instrumentation	Compile time instrumentation	Dynamic binary instrumentation
Time of defect detected	Runtime	Runtime	Runtime	Runtime	Runtime	Runtime
Defect	Memory	Memory	Memory	Data race, order violation	Data race	Memory, inter process communication, exception handling

4가 대표적이다. Memcheck, Dr. Memory, Address Sanitizer는 대표적인 메모리 결함 디버깅 상용도구이다[21-23]. 하나의 어플리케이션을 대상으로 런타임에 발생하는 메모리 결함을 탐지하고 유용한 정보를 제공함으로써 메모리 결함 디버깅을 지원한다. 그러나 Memcheck의 경우에는 가상 환경에서 실행되므로 실제 실행 환경에서 발생하는 메모리 결함과 차이가 존재하며, Address Sanitizer의 경우에는 compile time에 결함 디버깅을 위해 추가된 구문으로 인해 결함 발생 여부가 달라질 수 있다는 한계를 가진다.

Helgrind와 Thread Sanitizer는 data race를 탐지하기 위한 상용도구이다[24, 25]. Data race는 공유 자원에 대한 어플리케이션 혹은 스레드 간의 경쟁 상태로 deadlock과 같은 병행성 결함의 원인이 되지만, 그 자체는 결함이 아니다. 또한 별도의 가상머신에서 동작하거나 컴파일 시 data race 탐지를 위한 코드 추가로 인해 원래의 실행 파일이 변경되는 것은 병행성 결함의 발생 여부를 달라지게 할 수 있다는 한계를 가진다. 상용화된 도구는 아니나 AMOS가 있다[26]. 전체 시스템을 대상으로 동적 결함을 탐지하기 때문에 통합 이후의 동적 시험 지원 도구로 적합하다. 그러나 Table 1의 병행성 결함보다는 IPC(Inter Process Communication) 결함을 지원한다.

6. 결 론

무기체계 소프트웨어는 실시간성과 신뢰성, 상호운용성과 같은 특성을 가진다. 이러한 무기체계 소프트웨어의 특성에 맞는 소프트웨어 동적 시험 및 평가 개발은 매우 중요하다.

본 논문에서는 주요 소프트웨어 동적 결함으로 메모리 결함, 병행성 결함과 예외처리 결함을 도출하였다. 이 들 동적 결함이 실제 무기체계 소프트웨어에서 얼마나 실재하는지를 파악하기 위하여 미 국방부에서 무기체계에 사용하는 오픈소스 소프트웨어(OSS) 커뮤니티에서 등록된 이슈를 통하여 결함을 분석하였다. 실제 표1의 동적 결함이 보고되고 있음을 확인하였으며, 특히 이들은 다양한 외부 센서나 UI, DB와 같은 다른 구성 요소 CSC 및 CSCI들과의 상호 작용으로 통합 이후 발생하는 결함이며, 이 가운데에는 재현이 불가하여 결함 원인파악에 수일에서 수개월까지도 걸릴 수 있었다.

코드를 실행시키지 않고 수행하는 정적시험의 점검항목 및 도구의 보완 만으로는, 또한 소스코드 커버리지 측정 만으로는 이러한 문제를 모두 해결할 수는 없다. 이를 극복하기 위한 추가적으로 수행할 수 있는 무기체계 소프트웨어 검증 개선 방안으로써 1)통합 시험 평가 개선 방안으로 통합 세부단계 별 올바른 시험 실행률(coverage)의 개발 및 이를 지원하는 도구 적용과 2)소프트웨어 동적 결함 발견을 지원하는 디버깅 자동화 도구 적용을 들 수 있다.

향후 무기 체계 개선 방안을 구체화하여 적용 효과를 실험하는 연구를 추진할 계획이다.

References

[1] Kyeongyoun Kwon, Joonseok Joo, Taesik Kim, Jinwoo Oh, and Jihyun Baek, “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, pp.860-867, 2015.

[2] J. Kim, S. Jeong, I. Hwang, H. Cho, D. Kim, and Y. J. Jang, “M&S Verification, Validation and Accreditation Research Direction Considering the Characteristics of Defense M&S,” *Journal of Korean Institute of Industrial Engineers*, Vol.39, No.6, pp.486-497, 2013.

[3] IEEE Standard 1012-2016 IEEE Standard for System, Software, and Hardware Verification and Validation, 2016.

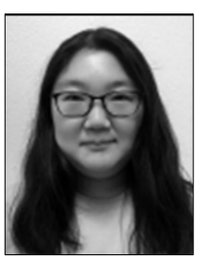
[4] ISO IEC IEEE 12207 System and software engineering - Software life cycle processes, 2017.

[5] ISO IEC IEEE 291199 Software and systems engineering - software testing, 2013.

[6] Weapon system software development and management manual, Defense Acquisition Program Administration Manual No.2017-8, 2017.

[7] 무기체계SW 발전방향 및 추진 전략 연구, Korea Defense Industry Association, 2016.

[8] D. A. Ormrod, “A ‘wicked problem’—Predicting sos behaviour in tactical land combat with compromised C4ISR,” *System of Systems Engineering (SOSE), 9th International Conference on*. IEEE, 2014.
[9] 전력발전업무훈령(개정2016.3.28 훈령 제1896호), 2016.
[10] Yonhap News, “국군 최초의 디지털군단 탄생[Defense Acquisition Program Administration]”, http://prlink.yonhapnews.co.kr/view.aspx?contents_id=RPR20060627008800353&from=search, (2006.06.27.)
[11] P. Emanuelsson and U. Nilsson, “A comparative study of industrial static analysis tools,” *Electronic Notes in Theoretical Computer Science*, Vol.217, pp.5-21, 2008.
[12] A. Fatima, S. Bibi, and R. Hanif, “Comparative study on static code analysis tools for C/C++,” *15th International Bhurban Conf. on Applied Sciences & Technology*, 2018.
[13] G. R. Luecke, J. Coyle, J. Hoekstra, M. Kraeva, Y. Li, O. Taborskaia, and Y. Wang, “A survey of systems for detecting serial run time errors,” *Concurrency and Computation: Practice and Experience*, Vol.18, No.15, pp.1885-1907, 2006.
[14] S. Lu, S. Park, E. Seo, and Y. Zhou, “Learning from mistakes: a comprehensive study on real world concurrency bug characteristics,” *ACM Sigplan Notices*, Vol.43, No.3, pp.329-339, 2008.
[15] F. Cristian, “Exception Handling and Tolerance of Software Faults,” Johy Wiley & Sons, Hoboken, JN, USA (Chapter 4). 1995.
[16] G. Blair, G. Coulouris, J. Dollimore, and T. Kindberg, “Distributed Systems: Concepts and Design,” Addison-Wesley, Boston. 2012.
[17] Padua, D. (Ed.). “Encyclopedia of parallel computing,” Springer Science & Business Media. 2011
[18] G. M. Schneider, & J. Gersting, “Invitation to computer science,” Cengage Learning. 2018.
[19] DoD CIO [Internet], http://dodcio.defense.gov/Open-Source-Software -FAQ/
[20] A study on software ILS applications, Defense Acquisition Program Administration, 2012.
[21] Memcheck [Internet], http://valgrind.org/docs/manual/mc-manual.html
[22] Dr. Memory [Internet], http://drmemory.org/
[23] Address Sanitizer [Internet], https://github.com/google/sanitizers/wiki/AddressSanitizer
[24] Helgrind [Internet], http://valgrind.org/docs/manual/hg-manual.html
[25] Thread Sanitizer [Internet], https://github.com/google/sanitizers /wiki/ThreadSanitizerCppManual
[26] Jooyoung Seo, Byoungju Choi, and Suengwan Yang. “A profiling method by PCB hooking and its application for memory fault detection in embedded system operational test,” *Information and Software Technology*, Vol.53, No.1, pp.106-119, 2011.



박 지 현
https://orcid.org/0000-0003-4478-7565
e-mail : pola0527@ewhain.net
2006년 이화여자대학교 컴퓨터학과(학사)
2011년 이화여자대학교 컴퓨터공학과(석사)
2011년~현 재 이화여자대학교 컴퓨터공학과 박사과정
관심분야 : 임베디드 소프트웨어 테스트, 결함 주입 테스트

최 병 주
https://orcid.org/0000-0003-3985-7645
e-mail : bjchoi@ewha.ac.kr
1983년 이화여자대학교 수학과(학사)
1990년 퍼듀대학교 컴퓨터학과(석・박사)
1995년~현 재 이화여자대학교 컴퓨터공학과 교수
관심분야 : 소프트웨어 검증 평가, 소프트웨어 테스트