Journal of KIIT. Vol. 24, No. 4, pp. 63-71, Apr. 30, 2026. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2026.24.4.63

비대상 프로그래밍 언어로 개발된 위성 비행소프트웨어의 소프트웨어 신뢰성 확보 방안 및 분석

윤수성*, 오상훈**

Software Reliability Assurance Approach for Flight Software of Satellite Developed in a Non-Target Programming Language

Soosung Yoon*, Sanghoon Oh**

Agency for Defense Development

요 약

위성의 비행소프트웨어는 우주 환경에서 실시간 위성 제어 및 임무 수행을 담당하는 핵심 요소로써, 높은 신뢰성과 안정성이 필수적으로 요구된다. 하지만 방위사업청의 무기체계 소프트웨어 개발 및 관리 매뉴얼에서는 소프트웨어 신뢰성 시험 대상 언어를 C 계열 언어로 한정하고 있어, 이외 언어로 구현된 소프트웨어는 신뢰성을 확보하는데 한계가 존재한다. 본 논문에서는 이러한 제약을 극복하기 위해 특정 비대상 프로그래밍 언어로 개발된 위성 비행소프트웨어를 대상으로 신뢰성 확보 방안을 수립하고, 이를 통해 상용 도구를 활용하여 정적/동적 시험을 수행하고 분석하였다. 본 연구는 비대상 언어 기반 비행소프트웨어에도 신뢰성 확보가 가능함을 보여줌으로써 여러 비대상 언어의 무기체계 소프트웨어 신뢰성 시험 표준화의 기초자료로 활용될 수 있을 것이라 기대된다.

Abstract

The Flight Software (FSW) of a satellite is a core component responsible for real-time satellite control and mission execution, for which high software reliability and stability are essentially required. However, the Defense System Software Development and Management Manual limits software reliability test to C-family languages, making it difficult to apply standardized software reliability verification process to software implemented in other programming languages. In this paper, to address this limitation, we propose a software reliability assurance process for satellite FSW implemented in a non-target programming language and perform static and dynamic analyses using commercial off-the-shelf tool. The results showed that reliability assurance is achievable even for flight software implemented in the non-target language. We believed that the proposed approach is expected to provide foundational material for the future standardization of software reliability test in defense system software developed using non-target languages.

Keywords

flight software, artificial satellite, software reliability, software stability

* 국방과학연구소 선임연구원
- ORCID: http://orcid.org/0000-0001-9823-1840
** 국방과학연구소 선임연구원(교신저자)
- ORCID: https://orcid.org/0009-0004-5449-6723

· Received: Dec. 04, 2025, Revised: Jan. 09, 2026, Accepted: Jan. 12, 2026
· Corresponding Author: Sanghoon Oh
1st R&D Institute - 1st PMO, Agency for Defense Development
34186 Yuseong P.O.Box 35, Daejeon, Republic of Korea
Tel.: +82-42-821-0768, Email: oshang55@add.re.kr

Ⅰ. 서 론

우주에서 임무를 수행하는 위성의 비행소프트웨어(FSW, Flight Software)는 궤도상에서 실시간으로 위성의 상태를 감시하고, 자세제어, 전력관리, 임무탑재체(Payload) 운용 등을 통합적으로 제어하는 핵심 구성요소이다. 특히, FSW는 궤도상에서 인간의 개입 없이 지상 명령에 따라 자율적으로 동작해야 하며, 극한의 온도 변화, 방사선 환경, 통신 지연 등과 같은 우주 환경 제약 속에서도 지속적으로 안정적인 기능을 수행해야 한다. 이러한 특성으로 인해 위성의 FSW는 일반적인 임베디드 소프트웨어보다 훨씬 높은 수준의 결정론적(Real-time deterministic) 성능과 신뢰성을 요구받기 때문에 지상에서 충분한 검증이 이루어져야 한다[1][2]. 또한, FSW 경우, 일반적인 지상 제어용 소프트웨어 달리 OBC(On-Board Computer) 상에서 RTOS(Real-Time Operating System) 기반으로 동작하며 다수의 하드웨어 구성품과 상호작용한다. 이뿐만 아니라 위성 내 주요 임무 기능(자세제어, 온도제어, 전력제어, 데이터 통신 등)을 실시간으로 조정하므로, 단 하나의 논리적 오류도 시스템 전반의 기능 마비로 이어질 수 있다. 이러한 구조적 특성은 FSW의 신뢰성 확보 활동을 단순한 기능 검증이 아닌 안정성 및 결정론적 타이밍 보장 측면에서 접근해야 함을 의미한다.

이와 같은 우주 환경 제약하에서는 단일 검증 기법만으로 FSW의 신뢰성을 확보하는 데 한계가 있으며, 실시간 제어 실패나 예외 상황에서의 비정상 동작은 위성 임무 실패로 직결될 수 있다. 이에 따라 개발 단계에서의 정적 분석을 통한 사전 오류 제거와 함께 실행 환경을 고려한 동적 분석을 병행함으로써 실행 경로 및 상태 전이에 대한 검증을 수행하는 체계적인 신뢰성 확보 방법론이 필수적으로 요구된다.

특히, 국방 및 민간의 위성 운용 환경에서는 임무 실패가 단순한 기능 결함을 넘어 막대한 경제적 손실과 전력적 위험으로 이어질 수 있다. 따라서 FSW의 소프트웨어 신뢰성 확보는 단순한 소프트웨어 품질 문제를 넘어 시스템 생존성과 임무 성공률을 결정짓는 필수 조건으로 인식된다.

이에 따라 국내외 항공우주 및 방위산업 분야에서는 FSW의 소프트웨어 신뢰성을 보장하기 위한 다양한 검증 절차와 표준을 마련해 왔다[3]-[6].

국내에서는 방위사업청(DAPA)에서 「무기체계 소프트웨어 개발 및 관리 매뉴얼」[3]을 통해 무기체계 소프트웨어의 신뢰성 시험 기준 및 절차를 명확히 규정하고 있다. 해당 지침은 C 계열 언어를 중심으로 MISRA C:2012, CWE, CERT C 규칙을 기반으로 한 정적 및 동적 분석을 수행하도록 요구한다. 그러나 실제 개발에서는 플랫폼의 구조적 제약, 레거시(Legacy) 코드의 사용, 유지보수, 국제 공동 개발 환경 등에서의 상호운용성, 또는 신뢰성이 높은 컴파일러 활용 등으로 인해 C 계열 언어 이외의 언어가 사용되는 사례가 존재한다. 이러한 언어로 개발된 FSW는 높은 형식성과 실시간 검증 가능성을 지니고 있음에도 불구하고, 무기체계 소프트웨어 개발 및 관리 매뉴얼 등 기존 표준 검증 방법론에서는 명확한 시험 범주에 포함되지 못하는 한계가 있다. 다만, 비대상 프로그래밍 언어로 개발된 FSW는 이미 유럽 등의 우주개발 프로그램에서 실제 적용 및 운용되고 있으며, 문제의 핵심은 해당 소프트웨어의 부재가 아니라 C 계열 언어와 달리 명문화된 소프트웨어 신뢰성 시험 기준과 절차가 정립되어 있지 않다는 점에 있다. 특히 일부 강타입(Strongly typed) 언어 기반 FSW는 안정성과 결정론적 제어에 유리한 특성을 지니지만 무기체계 소프트웨어 개발 및 관리 매뉴얼에서는 공식적인 시험 대상으로 포함되지 못하는 구조적 공백이 발생하고 있다.

본 논문에서는 특정 강타입 언어로 구현된 위성 FSW의 소프트웨어 신뢰성을 확보하기 위한 절차적 방법론을 제시하고 실제 위성에 탑재된 FSW에 이를 적용하여 정적/동적 분석 결과를 제시한다. 특히 상용 자동화 도구인 LDRA를 활용하여 정적 및 동적 분석을 병행함으로써 기존에 적용받지 못한 비대상 언어에 대한 소프트웨어 신뢰성 확보 방안을 제시하였다. 정적 분석 단계의 코딩 규칙(Coding rule) 점검 시에는 먼저 무기체계 소프트웨어 개발 및 관리 매뉴얼에 포함된 시험 규칙 중에서 적용 가능한 코딩 규칙들을 개발 컴파일러 특성과 비교 및 분석했다.

이 과정에서 시행착오(Trial and error) 방식으로 적용 가능한 규칙들을 도출하였다. 또한 LDRA의 호환성을 검토하고 필요한 수정을 수행했으며, FSW 코드 리뷰를 병행하여 해당 FSW의 특성에 적합한 새로운 코딩 규칙을 추가로 정의하였다. 또한, 취약점 점검의 경우 자동화 도구가 보유한 취약점 점검 규칙인 CWE(Common Weakness Enumeration) 중 매핑이 가능한 항목을 매핑하여 취약점 점검을 수행하였다. 동적 분석 단계에서는 실 하드웨어 타겟 보드(Target board) 기반에서 LDRA의 문장(Statement) 및 분기(Branch) 커버리지 측정 기능을 이용하여 동적 분석을 병행함으로써 커버리지 완성도를 평가하고자 하였다. 적용 분야 특수성에 의한 탑재 하드웨어의 성능 제약으로 단일 바이너리 기준의 동적 분석이 상용 자동화 도구에서 지원되지 않아 각 태스크(Task)에 대해 다수 바이너리로 분할하여 분석을 수행하였다. 그 외 정적 분석 단계의 코드 메트릭(Code metric)은 기존 표준 검증 방법에 따라 분석을 수행하여 신뢰성 확보를 완료하였다.

본 연구의 최종 목표는 실제로 개발 및 탑재된 특정 비대상 프로그래밍 언어 기반 위성 FSW를 대상으로 정적 및 동적 분석을 병행한 신뢰성 확보 방안과 그 적용 가능성을 제시하는데 있다. 즉, 기존 소프트웨어 신뢰성 확보 방법의 언어 종속성을 극복하고, 방위산업 소프트웨어 신뢰성 시험 범위를 확장할 수 있음을 보여준다. 이를 통해 본 연구는 여러 비대상 언어의 소프트웨어 신뢰성 확보 방법론에 대한 기초자료로 활용될 수 있을 것이라 기대된다.

본 연구에서 다루는 비대상 프로그래밍 언어는 기존 소프트웨어 신뢰성 시험 기준에서 직접적으로 다루어지지 않는 특정 언어를 의미하며, 보안상 해당 프로그래밍 언어의 명칭은 명시하지 않는다.

본 논문의 구성은 다음과 같다. 2장은 기존의 소프트웨어 신뢰성 시험 관련 연구 및 신뢰성 확보 표준 동향을 분석한다. 3장에서는 LDRA 기반 정적 및 동적 분석 절차를 포함한 신뢰성 확보 방법론을 제시한다. 4장에서는 신뢰성 확보 방법론을 통해 실제 위성 FSW에 적용한 결과를 통해 신뢰성 향상 효과를 정량적으로 분석한다. 5장에서는 본 연구의 의의와 한계, 향후 연구 확장 방향을 기술한다.

II. 관련 연구 및 표준 동향

소프트웨어 신뢰성 확보에 관한 연구는 항공우주, 자동차, 원자력 등 고신뢰 분야를 중심으로 활발히 수행되어왔다. 특히 위성 FSW는 시스템 오작동이 임무 실패로 직결되기 때문에, 국내외에서는 이를 위한 다양한 표준과 검증체계가 마련되어 있다. 대표적으로 유럽우주국(ESA)의 ECSS-E-ST-40C (Software Engineering)와 미항공우주국(NASA)의 NASA-STD-8739.8(Software Assurance Standard)은 개발 전주기에 걸쳐 품질보증(SQA, Software Quality Assurance) 및 검증활동(V&V, Verificiation and Validation)을 수행하도록 규정하고 있다. 이러한 표준은 정적 분석과 동적 분석을 병행하여 코드의 정확성, 일관성, 추적성을 확보하고, 기능 안정성과 임무 신뢰성을 정량적으로 평가할 것을 요구한다[1][2].

자동차 분야의 ISO 26262, 산업제어 시스템의 IEC 61508, 그리고 항공전자 소프트웨어 개발 표준인 DO-178C 역시 소프트웨어의 안전등급(SIL, Safety Integrity Level) 또는 신뢰성 수준을 정의하고 있으며, 정형기법, 코딩 규칙 준수, 단위시험과 구조적 커버리지(Statement, Branch, MC/DC 등) 분석을 필수적으로 수행하도록 한다[3][4]. 이러한 표준들은 주로 C 계열 언어 환경을 가정하여 규칙 체계를 구성하고 있으며, 대표적으로 MISRA C:2012는 산업계에서 사실상의 표준으로 사용되고 있다. MISRA 규칙은 코드의 안정성과 유지보수성을 보장하기 위한 규칙 집합으로, 정적 분석 도구를 통해 컴파일 전 단계에서 오류와 취약점을 사전에 식별하도록 한다. 또한 CWE는 소프트웨어 보안 및 품질 취약점을 분류하는 국제적 분류체계로, 방위 및 항공 분야의 검증 규칙 설계에도 널리 활용된다[5].

이와 관련하여 안전이 필수적으로 요구되는 시스템을 대상으로 한 기존 국외 연구들에서도 프로그래밍 언어의 종류와 무관하게 정적 분석과 동적 분석을 병행하는 검증 접근이 소프트웨어 신뢰성 확보의 핵심 요소로 제시되어왔다[6]-[8]. 기존 국외 연구들은 안전 필수 시스템을 대상으로 소프트웨어 신뢰성 확보의 중요성을 강조하며, 정적 분석과 동적 분석을 병행한 검증 접근이 필요하다는 점을 공통적으로

제시하고 있다. 그러나 이러한 연구들은 주로 개념적 논의나 일반적인 검증 원칙을 중심으로 전개되어, 특정 소프트웨어 사례를 대상으로 실제 시험 환경을 구성하고 구체적인 신뢰성 확보 절차와 적용 결과를 체계적으로 제시하는 데에는 한계가 있다.

국내에서도 무기체계 소프트웨어 등의 신뢰성 확보를 위해 이러한 국제표준을 참조한 소프트웨어 품질관리 지침이 마련되어 있으며, MISRA C, CERT C, CWE 등과 연계한 소프트웨어 신뢰성 시험 절차를 규정하고 있다. 그러나 실제 현장에서는 서론에서 언급한 바와 같이 이러한 절차에 적용을 받지 못하는 비대상 언어로 개발된 FSW가 사용될 수 있다. 이러한 환경에서는 기존 규칙 체계를 그대로 적용하기가 어려워, 검증 규칙을 부분적으로 변형하거나 자체 개발 규칙을 적용해야 하는 소요가 발생할 수 있다. 특히 비대상 언어는 정형적 문법구조를 가지고 있음에도 불구하고 정적 분석 도구의 규칙 매핑이 부족하거나 지원되지 않는 경우가 많아 신뢰성 확보 활동이 체계화되지 못하는 문제가 발생한다[9].

이와 관련하여 해외에서는 비표준 언어 기반 시스템에 대한 검증 연구가 제한적으로 이루어져 왔으며, 일부 연구에서는 코드 메트릭과 커버리지 기반 분석을 결합하여 비대상 언어의 신뢰성 확보 가능성을 검토하기도 하였다. 예를 들어 유럽우주국의 소형위성 FSW 연구에서는 Python 및 Fortran 기반 모듈에 대해 코드 복잡도와 분기 커버리지를 조합한 검증 기법을 제안하였으며, NASA JPL에서는 Ada 계열 언어의 형식적 검증을 통한 신뢰성 평가를 수행한 바 있다[10][11].

국내의 경우 비대상 프로그래밍 언어로 개발된 위성 FSW 대한 신뢰성 확보 연구는 거의 전무하며, 표준 검증 절차의 공백으로 인해 프로젝트별 자율 규칙에 의존하는 실정으로 예상된다.

III. 신뢰성 확보 방안

본 장에서는 특정 비대상 프로그래밍 언어로 개발된 위성 FSW의 신뢰성 확보를 위해 수립된 절차와 방안을 기술한다. 제안된 절차는 표준 신뢰성 확보 절차를 참조하고 자동화 도구인 LDRA 기능을 최대한 활용 및 보완하여 객관적 신뢰성 평가를 가능하게 하였다. 해당 방안은 정적 분석과 동적 분석 두 단계로 구성된다.

3.1 정적 분석

정적 분석 단계에서는 소스코드 수준의 코딩 규칙 위반 여부, 취약점 점검(CWE 매핑 항목) 및 소스코드 메트릭에 대하여 자동화 점검을 수행하였다.

코딩 규칙 점검에서는 방위사업청 무기체계 소프트웨어 개발 및 관리 매뉴얼의 C/C++ 규칙을 개발 컴파일러 특성과 비교 분석하고, LDRA 도구사와의 협업을 통해 시행착오 방식으로 적용 가능한 15개 항목을 도출하였다. 또한, 개발된 FSW의 개발자 코드 리뷰, LDRA와의 호환성 검토 및 자동화 도구 수정을 병행함으로써 본 FSW 특성에 맞는 62개의 신규 코딩 규칙을 도출하였다. 최종적으로 도출된 77개의 기존 및 신규 코딩 규칙의 목록은 각각 표 1 및 표 2와 같다.

표 1. 적용된 표준 코딩 규칙 목록
Table 1. List of the applied standard coding rules

No.	Description
1	Types shall be specified when declaring functions and variables
2	Meaningless statements shall not be used
3	External functions shall be identified before use
4	The use of the goto statement shall be prohibited
5	Each function shall have a single exit point
6	Each line shall contain only one statement
7	An if - else if statement shall include a final else clause
8	All variables shall be initialized before use
9	Names of functions or objects declared with external linkage shall be unique
10	The data types of functions or objects defined with external linkage shall match those specified in their declarations.
11	Equality comparison operations shall not be performed on floating-point types
12	Conditional statements whose results are always TRUE or FALSE shall not be written as conditional statements
13	Labels in switch case statements that cannot be satisfied shall not be used
14	Unexecuted (dead) code shall not be implemented
15	When a variable is used as a divisor, a check shall be performed to ensure that it is not zero

표 2. 적용된 신규 도출 코딩 규칙 목록
Table 2. List of the applied newly derived coding rules

No.	Description
1	Subprogram body exceeds limit of *** semicolons
2	Task name missing
3	Number Declaration
4	Uncommented begin in package body
5	More than one unit in source file
6	Improper return statement
7	USE clause
8	Renames exception
9	Renames object
10	Renames package
11	Overloaded operator symbol
12	Loop without iteration bound
13	Use of GOTO statement
14	Exit statement with no loop name
15	Unnamed loops
16	Package name missing
17	Subprogram name missing
18	Task body name missing
19	Subprogram name missing
20	Too many parameters. Max ***
21	Default parameter
22	Unconstrained array returned
23	Variable should be declared constant
24	Non static range
25	Only one literal in enumeration
26	Equality check on floating point type
27	Index constraint is not a type mark
28	Anonymous subtype in array declaration
29	The use of tasking should be avoided where possible.
30	Record field unconstrained array type
31	Pointer to task/protected
32	Use of non short circuit operator
33	Exit statement in if with else/elsif
34	Positional component in aggregate
35	Others choice in case statement
36	Enumeration range used in case statement
37	Improper return statement
38	Block statement
39	Generic unit definition in subprogram
40	Exception used as control flow
41	Raising external exception
42	Handler for predefined exception
43	Raise of a predefined exception
44	Handler for when others
45	Address clause imposed on subprogram
46	Address clause imposed on package
47	Abort statement
48	Select statement
49	Multiple entries in protected definition
50	Ada.Dynamic_Priorities withed
51	Ada.Asynchronous_Task_Control withed
52	Use of requeue
53	Calendar package withed
54	Asynchronous transfer of control
55	Task declared incorrectly
56	Relative delays
57	Task entry usage
58	Protected declared incorrectly
59	Protected entry with incorrect barrier
60	Ada.Task_Attributes withed
61	Ada.Interrupts withed
62	Pointer to task/protected

취약점 점검에서는 CWE 항목에 대해서 점검을 수행하였다. 이때 모든 CWE 항목 매핑 및 점검이 불가하여 LDRA에서 보유한 취약점 점검 86개 중 본 FSW와 매핑이 가능한 CWE 항목 39개를 선정하였으며 이는 표 3과 같다.

코드 메트릭 점검은 무기체계 소프트웨어 개발 및 관리 매뉴얼의 6가지 항목에 대해 지침 제한 값을 그대로 적용하여 결과를 도출하였다.

이와 같은 코딩 규칙, 취약점 점검, 코드 메트릭 기준으로 특정 비대상 프로그래밍 언어로 개발된 위성 FSW를 지속 수정하고 정적 분석을 수행하였다. 이를 통해 위반 건수를 감소시킴으로써 FSW의 소프트웨어 신뢰성을 확보하였다. 다만, 수정으로 인한 부수 효과(Side effect) 우려 부분 또는 개발자 의도 등으로 개발되었지만 정적 기준을 위반한 항목에 대해서는 충분한 코드 분석을 통해 예외적인 사유를 제시함으로써 정적 분석을 완료하였다.

표 3. 적용된 취약점 점검 매핑표
Table 3. List of the applied weakness check mapping table

No.	Description	CWE
1	Divide by zero found	190
2	Pointer not checked for null before use	232
3	Attempt to use uninitialized pointer	
4	UR anomaly, variable used before assignment	
5	Unused procedure parameter	235
6	Unused procedural parameter	
7	Raising external exception	248
8	Raise of a predefined exception	
9	Task body when others handler missing	
10	Potentially unused function return value	252
11	Potentially unused function return value	273
12	Array component is not a type mark	351
13	Index constraint is not a type mark	
14	Default Expression in record	
15	Record component is not a type mark	

16	Unprotected shared variable	362
17	Divide by zero found	369
18	Null statement in exception handler	390
19	Potentially unused function return value	391
20	Task body when others handler missing	
21	Handler for predefined exception	396
22	Handler for when others	
23	Raise of a predefined exception	397
24	Non-visible exception	
25	Raising external exception	431
26	Non-visible exception	
27	Storage_Error handler missing	
28	Task body when others handler missing	
29	Attempt to use uninitialized pointer	457
30	UR anomaly, variable used before assignment	
31	Raising external exception	460
32	Index constraint is not a type mark	
33	Null statement in exception handler	
34	Pointer not checked for null before use	476
35	Procedure not called in code analyzed	561
36	Unreachable code found.	
37	Exit statement not last statement of an if	
38	Inter-file recursion found.	
39	Procedure not called anywhere in system.	
40	No procedures in this file called from other files.	
41	Variables were declared but never used	563
42	Unreachable code found.	570
43	Unreachable code found.	571
44	Absolute addresses shall not be coded into source code	587
45	Procedure definition has no associated prototype.	628
46	Procedure call has no prototype declared.	
47	UR anomaly, variable used before assignment	665
48	Recursion in procedure calls found	674
49	Inter-file recursion found.	
50	Unchecked_Deallocation withed	676
51	Use of GOTO statement	691
52	Exception used as control flow	
53	Global used in procedure matches local parameter	694
54	Procedure name re-used in several files.	
55	Identical name in scope	
56	Declaration types do not match across a system.	
57	Ambiguous declaration of variable.	
58	Raising external exception	703
59	Object address clause missing Volatile	733
60	Hardware object missing Volatile	
61	Raising external exception	755
62	Non-visible exception	
63	Null statement in exception handler	
64	Others only choice in case statement	
65	Divide by zero found	758
66	Pointer not checked for null before use	
67	Attempt to use uninitialised pointer	
68	UR anomaly, variable used before assignment	
69	Potential side effect problem in expression	
70	Potential side effect from repeated function call	
71	Machine Code Insertion	
72	Expression has side effects.	763
73	Unchecked_Deallocation withed	
74	Allocator	770
75	Catenation operator	
76	Unconstrained array definition	
77	Allocator	789
78	Unconstrained array definition	
79	Standard type used in unconstrained array	
80	Attempt to use uninitialized pointer	824
81	UR anomaly, variable used before assignment	
82	Recursion in procedure calls found	834
83	Loop without iteration bound	
84	Procedure contains infinite loop	835
85	Potentially infinite loop found.	
86	UR anomaly, variable used before assignment	908

3.2 동적 분석

동적 분석 단계에서는 실행 경로를 기반으로 한 커버리지 측정을 통해 검증을 수행하였다. 자동화 도구의 문장 및 분기 커버리지 측정 기능을 활용하여 주요 기능단위 별 실행 경로를 분석하였다.

이때 하드웨어 성능에 기반한 실시간성 검증을 포함한 충분한 동적 분석을 위해서는 타겟 보드가 필요하다. 따라서 타겟 보드와 기능 및 성능이 동일한 탑재컴퓨터 공학모델(Engineering model) 하드웨어를 제작하였다. 또한, 위성 제작 및 검증에 사용하는 것과 동일한 위성 센서 및 구동기의 소프트웨어 모델을 활용하여 탑재컴퓨터 공학모델과 함께 STB(Software Test-Bed)를 구축하고 동적 분석을 진행하였다.

동적 분석의 경우 커버리지 측정을 위해 측정용

탐침 코드가 바이너리에 포함되어야 한다. 이때 위성 탑재컴퓨터의 메모리 성능 제약으로 인해 탐침 코드가 실행 코드에 포함될 경우, 단일 바이너리로 동적 분석이 진행되지 못하는 한계가 존재하였다. 따라서 탑재하여 분석이 가능한 바이너리 최소 수량을 도출하기 위해 STB 환경에서 지속적인 실험을 통해 동적 분석이 가능한 바이너리 개수를 도출하여 분석을 진행하였다. 실험적으로 도출된 다섯 개의 분할된 바이너리에 대해 각각 동적 분석을 별도 수행하고, 그 결과를 통합하여 앞서 언급한 문장 및 분기 커버리지 측정을 완료하였다.

Ⅳ. 결과 분석

본 장에서는 제안된 신뢰성 확보 절차를 실제 특정 소프트웨어 신뢰성 시험 비대상 언어 기반으로 구현된 위성의 FSW에 적용한 결과를 분석한다.

본 연구에서 분석된 FSW는 6개의 CSC(Computer Software Component)로 구성된다. 이때 개별 CSC의 명칭이나 기능은 보안상 생략한다. 정적 분석은 세 개의 신규 개발 CSC에 대해서 수행하였고, 동적 분석은 전체 구성요소를 대상으로 수행하였다. 신규 개발 CSC 대상으로만 정적 분석을 수행한 이유는 이외 CSC의 경우 레거시 코드로써 정적 분석을 통해 코드 수정을 수행할 경우 부수 효과 등으로 기존에 운용을 통해 확보한 안정성이 저하될 가능성이 존재할 수 있기 때문이다.

먼저 정적 분석 결과는 표 4 및 그림 1과 같다. 이때 FSW의 소프트웨어 신뢰성이 향상됨을 단계적으로 보이기 위해 총 4단계(4개 버전)로 소프트웨어 신뢰성 시험을 실시하여 결과를 제시하였고, 이는 동적 분석 결과에도 동일하게 적용하였다.

표 4. 정적 분석 결과표
Table 4. Table of results for static analysis

Category	FSW V1	FSW V2	FSW V3	FSW V4
Coding rule	862 violations (19 rules)	0	0	0
Vulnerability assessment	345 violations (10 rules)	0	0	0
Code metric	39 violations	25 violations	25 violations	0

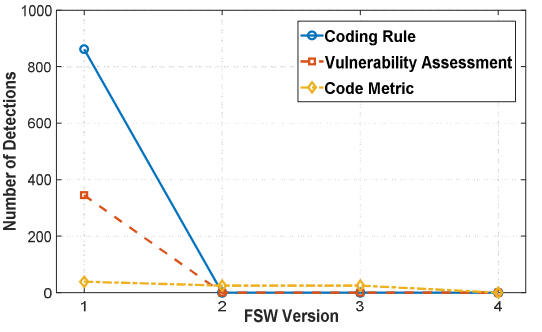

그림 1. 정적 분석 결과 그래프
Fig. 1. Graph of results for static analysis

코딩 규칙 및 취약점 점검의 경우 초기 버전인 V1에서 각각 862회 및 345회의 위반사항이 검출되었으나 검출된 위반사항에 대해 코드 리뷰를 수행하고 코딩 규칙과 취약점 점검 위반사항이 없도록 지속적으로 코드를 수정하였다. 코드 메트릭의 경우 표 4에 명시된 바와 같이 FSW V1에서 총 39개의 위반 사항이 존재하였다. 6개의 코드 메트릭 기준 중 결함이 검출된 5개의 코드 메트릭 “Number of Code Lines”, “Cyclomatic Complexity”, “Number of Calling Functions”, “Number of Called Functions”, “Number of Function Parameters”에 대해 결함수는 각각 4개, 3개, 19개, 11개, 2개로 총 39개가 위반되었다. 이에 따라 V2 및 V3에서 코드 수정을 진행하여 “Number of Calling Functions”, “Number of Called Functions” 항목에 대해 각각 18개 및 7개의 총 25개 검출 건수로 줄일 수 있었다. 위의 두 가지 메트릭 기준은 지상 명령의 개수 또는 함수 호출의 개수 등에 따라 발생하는 위반사항으로 위성을 운용할 때 매우 빈번히 발생하는 사항이다. 이를 제거하기 위해서는 동일 기능을 갖는 함수를 추가해야 하지만 시스템 안정성을 위해 별도 수정 없이 반복 기능시험을 통해 실제 운용에는 문제가 없음을 확인하였다. 이에 따라 최종 버전인 FSW V4에서는 정적분석 기준을 모두 충족한 것으로 보았다.

동적 분석의 경우 표 5 및 그림 2와 같이 최초 버전 V1에서의 실행률은 저조하지만, 코드 내 불필요한 코드 부분 제거 및 반복된 요구사항 기반 스크립트 수정을 통해 실행률을 달성하였다. 이때 버전 V4에서도 다음의 이유로 일부 미달성 사항이 존재한다.

표 5. 동적 분석 결과표(실행률)
Table 5. Table of results for dynamic analysis(Execution rate)

Category	FSW V1	FSW V2	FSW V3	FSW V4	Exception coverage
A CSC	36%	76%	81%	100%	19%
B CSC	34%	70%	81%	100%	19%
C CSC	36%	72%	83%	100%	17%
D CSC	56%	97%	97%	100%	3%
E CSC	54%	71%	74%	100%	26%
F CSC	56%	61%	61%	100%	39%

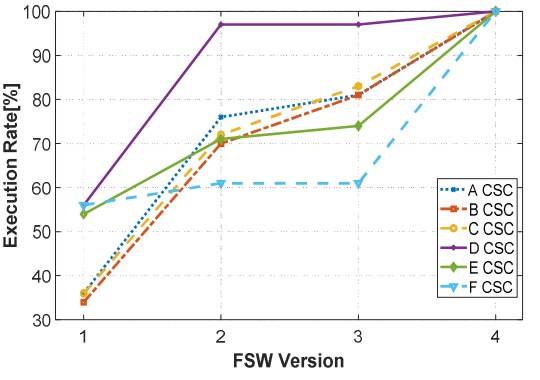

그림 2. 동적 분석 결과 그래프
Fig. 2. Graph of results for dynamic analysis

첫째, 실제 하드웨어 고장 상황 모의가 불가능하여 위성에서 필수적으로 요구되는 고장탐지, 분리 및 복구(FDIR, False Detection, Isolation and Recovery) 기능에 해당하는 코드에 진입할 수 없다.

둘째, 잘못된 입력값, NULL 포인터, 범위 초과 값 등 예상 불가한 입력 및 환경에서도 FSW가 안정적으로 동작하도록 구현된 방어 코드에 진입하도록 모의할 수 없다. 이에 따른 예외 처리율은 표 5와 같으며, 예외에 해당하는 모든 사항은 개발자 코드 리뷰 및 반복적 기능시험을 통해 운용에 문제가 없음을 확인하였다.

이러한 정적 및 동적 분석 결과를 통해 볼 수 있듯이 비대상 프로그래밍 언어에서도 제안된 방안을 기반으로 위반 항목 점검 및 실행률 점검이 가능하고, 이를 지속적으로 보완하여 달성 수치를 높일 수 있었다. 따라서 비대상 프로그래밍 언어로 개발된 위성의 FSW에서도 기존 C 계열 언어의 소프트웨어 신뢰성 시험과 같이 충분한 소프트웨어 신뢰성이 확보될 수 있음을 입증하였다.

Ⅴ. 의의 및 향후 연구 방향

본 연구는 특정 비대상 언어 기반으로 구현되어 실제 위성에 적용된 FSW에 대해 신뢰성 확보 절차를 수립하고, 이를 실증적으로 적용하여 효과를 입증하였다는 점에서 의의가 있다. 특히 기존 C 계열 언어 의존적으로 제시된 소프트웨어 신뢰성 시험 기준의 한계를 극복하고, 객관적인 규칙을 수립하였다. 또한 이를 기반으로한 정적 분석과 커버리지 기반 동적 분석을 수행함으로써 프로그래밍 언어의 제약을 벗어나 정량적 지표 확인을 통해 구현된 FSW의 신뢰성을 확보할 수 있었다.

향후 연구에서는 제안된 절차를 기반으로 다양한 프로그래밍 언어 및 플랫폼으로 확장할 수 있는 표준화된 신뢰성 확보 기준을 수립하는 것을 목표로 한다. 특히, 실시간 운용체제 및 실제 하드웨어 환경에서 보다 효율적인 동적 분석 방법, 그리고 정적 분석 규칙과 CWE 항목 간 매핑 자동화 알고리듬을 개발하여, 비대상 언어 기반 소프트웨어의 신뢰성 확보 활동을 표준화된 프로세스로 발전시키는 것을 장기적 연구 과제로 설정한다.

References

[1] NASA, "Software Assurance Standard and Software Safety Standard (NASA-STD-8739.8A)", NASA Technical Standard, pp. 1-59, Jun. 2020.

[2] European Cooperation for Space Standardization, "ECSS-Q-ST-80C: Space Product Assurance - Software Product Assurance", ECSS, pp. 1-132, Apr. 2025.

[3] Defense Acquisition Program Administration, "Weapon System Software Development and Management Manual", DAPA, pp. 1-128, Dec. 2024.

[4] MISRA Consortium, "MISRA C:2012 – Guidelines for the Use of the C Language in Critical Systems", 2013.

[5] The MITRE Corporation, "Common Weakness Enumeration(CWE) Database", 2024. https://cwe.mitre.org

[6] D. L. Parnas, A. J. Schouwen, and S. P. Kwan, "Evaluation of Safety-Critical Software", Communications of the ACM, Vol. 33, No. 6, pp. 636-648, Jun. 1990. https://doi.org/10.1145/78973.78974.

[7] J. C. Knight, "Safety-Critical Systems: Challenges and Directions", Proc. of the 24th International Conference on Software Engineering, ICSE 2002, Orlando, FL, USA, pp. 547-550, May 2002.

[8] N. Leveson, "The Role of Software in Spacecraft Accidents", Journal of Spacecraft and Rockets, Vol. 41, No. 4, pp. 564-575, July. 2004. https://doi.org/10.2514/1.11950.

[9] CERT Coordination Center, "SEI CERT C Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems", Carnegie Mellon University, pp. 1-534, Jun. 2016.

[10] ISO/IEC/IEEE, "ISO/IEC/IEEE 12207:2017 - Systems and Software Engineering — Software Life Cycle Processes", pp. 1-145, Nov. 2017.

[11] LDRA Ltd., "LDRA Tool Suite User Guide", LDRA, 2023.

[12] B. Boehm and V. R. Basili, "Software Defect Reduction Top 10 List", IEEE Computer, Vol. 34, No. 1, pp. 135-137, Jan. 2001. https://doi.org/10.1109/2.962984.

저자소개

윤 수 성 (Soosung Yoon)

2013년 2월 : 연세대학교 전기전자공학부(공학사)
2015년 2월 : 한국과학기술원(KAIST) 전기 및 전자공학과(공학석사)
2015년 3월 ~ 현재 : 국방과학연구소 선임연구원
관심분야 : 인공위성, 비행소프트웨어, 영상신호처리

오 상 훈 (Sanghoon Oh)

2007년 2월 : 경북대학교 전자전기컴퓨터공학과(공학사)
2009년 2월 : 포항공과대학교(POSTECH) 전자전기공학과(공학석사)
2009년 2월 ~ 2013년 3월 : 삼성전자 무선사업부 선임연구원
2013년 4월 ~ 현재 : 국방과학연구소 선임연구원
관심분야 : 인공위성, 비행소프트웨어