JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 29 No. 8, pp. 67-75, August 2024
https://doi.org/10.9708/jksci.2024.29.08.067

A Study on the System Management CSCI Software Modularization in Naval Combat Management System

Hyeon-Tae Ha*

*Engineer, SW Team(Naval), Hanwha Systems, Gumi, Korea

[Abstract]

Frequently changeable functional requirements in System Management CSCI Software make it difficult to reuse an overall application, but only the partial class codes repeatedly as a new version of Naval Combat Management System is developed. This structural environment leads to increasing development time and expenses. This is why modularization for System Management CSCI Software is proposed as a solution, leveraging the advantages of proper standardization and functional expandability offered by Standard Interface Architecture. This paper outlines the comparisons of modified class ratios as well as software reliability test runtime results between before and after implementing the modularization for System Management CSCI Software. The findings demonstrate there is sufficient improvement in areas, such as higher maintenance and reusability, supporting the application of modularization for System Management CSCI Software with the implementation of Standard Interface Architecture.

▸ **Key words**: Naval Combat Management System, System Management, Software Modularization, Software Reuse, Standard Interface Architecture, Feature Model

[요 약]

함정전투체계의 필수 CSCI(Computer Software Configuration Item) 중 하나인 체계관리 CSCI 소프트웨어는 타 CSCI보다 함형 별 기능 요구사항 변경이 자주 발생한다. 빈번한 요구사항 변경은 소프트웨어의 완전한 재사용을 어렵게 만들고, 신규 함형 개발 시 소스 코드 수준의 재사용을 반복하게 한다. 그리고 이러한 구조적 환경은 비효율적인 공수 활용에 따른 개발 시간 및 비용 증가를 일으키는 원인이다. 이를 개선하기 위해 본 논문에서는 체계관리 CSCI 소프트웨어의 재사용을 높이기 위한 모듈화를 적용하였다. 모듈화 적용의 수단으로는 표준화의 용이성과 기능 확장성의 장점이 있는 표준 연동 아키텍처 방법을 활용하였다. 비교 평가 방법으로는 체계관리 CSCI 소프트웨어 모듈화 적용 전/후의 기능 요구사항 수정 시 변경되는 클래스 비율의 비교, 그리고 소프트웨어 신뢰성 시험의 수행 시간 비교 등의 실험 평가를 수행하였다. 이러한 실험 결과를 통해 본 논문은 체계관리 CSCI 소프트웨어의 모듈화 적용 시 기존보다 높은 유지보수성과 재사용성을 가짐을 검증하였다.

▸ **주제어**: 함정전투체계, 체계관리, 소프트웨어 모듈화, 소프트웨어 재사용, 표준 연동 아키텍처, 휘처 모델

• First Author: Hyeon-Tae Ha, Corresponding Author: Hyeon-Tae Ha
*Hyeon-Tae Ha (mypimang2003@hanwha.com), SW Team(Naval), Hanwha Systems
• Received: 2024. 06. 05, Revised: 2024. 07. 19, Accepted: 2024. 07. 26.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

유럽의 군사적 긴장이 대만을 포함한 아시아로 확대되면서, 동북아 국제정세는 시시각각 변하고 있다. 이러한 상황 속에서 대한민국 해군은 군사적/외교적/민사적/경찰 역할을 가지는 주체[1]로서, 창군 이래 강한 해군력 확보를 위해 다양한 수단들을 발전시켜왔다. 그리고 그 수단 중 하나인 함정전투체계(Combat Management System) 또한 그 존재감을 확실히 다져왔다.

이러한 함정전투체계의 발전에도 불구하고 그 소프트웨어는 발전성이 비교적 느린 편이었다. 국내 방위원가 계산의 구조적 특징과 매출 감소 시 일반관리비 중 인건비를 절감하는 경향이 있는 방산기업의 특성이 맞물려[2], 소프트웨어의 투자에 비교적 관심도가 적었기 때문이다. 그리고 이는 함정전투체계 소프트웨어 모듈화의 기반이 충분히 마련되지 못한 이유가 되었다. 이러한 배경 속에서 일부 함정전투체계 소프트웨어는 모듈 재사용이 아닌 기존에 검증된 소스 코드 일부가 재활용되는 수준으로 개발되었다. 소스 코드 변경은 함형 별 특화 기능 요구사항과 함정전투체계 적용 베이스라인의 개선 등에 따라 필연적이므로, 기존 소프트웨어를 완전히 재사용하는 것이 불가능했기 때문이다.

이러한 소프트웨어의 불완전한 재사용은 이해관계자들의 효율적이지 못한 공수 활용을 야기할 수 있다. 개발자로서는 기존 소프트웨어 구조를 답습해야 하는 제한적인 상황이므로, 코드 스멜[3]의 영향 하에 비효율적으로 코드 수정 작업을 수행하게 된다. 방위산업체의 입장에서는 함형 별 차이에 따른 형상관리와 신뢰성 시험 기간 증가에 따라 추가적인 공수 소모의 여지가 있다. 그리고 운용자의 입장에서는 개선점을 검토하고 기술변경이 필요한 요소를 식별하는 시간이 필요하다. 이러한 비효율적인 공수 활용은 개발 기간과 개발 비용의 증가로 이어지기 쉽다.

따라서 이를 해결하기 위한 표준 연동 아키텍처의 개발은 함정전투체계 개발 이해관계자들의 시대적 요구사항이라고도 할 수 있었다. 이에 따라 2021년에는 Naval Shield Component Platform과 같은 표준 연동 아키텍처에 대한 연구에 가시적인 성과가 있었다[3]. 그리고 표준 연동 아키텍처를 함정전투체계 소프트웨어에 적용하는 연구하는 결과도 점차 활발해지는 추세다[4][5].

이러한 배경에서 본 논문에서는 함정전투체계 체계관리 CSCI(Computer Software Configuration Item) 소프트웨어에 모듈화를 적용하는 방안을 제시한다. 본 모듈화의 목적은 체계관리 CSCI 소프트웨어의 유지보수성과 재사용성을 높여, 빈번히 발생하는 요구사항의 변경에 따른 개발 기간과 개발 비용 증가에 대한 위험성을 줄이는 것에 있다. 그리고 실험 평가를 통해 함정전투체계 소프트웨어의 모듈화에 대한 유지보수성과 효율성 향상을 입증하였다.

본 논문의 구성은 다음과 같다. 2장에서는 체계관리 CSCI 소프트웨어에 모듈화와 관련한 연구와 배경지식을 설명한다. 3장에서는 표준 연동 아키텍처를 적용한 체계관리 CSCI 소프트웨어의 모듈화 전후 구조를 비교하여 설명한다. 그리고 4장에서는 실험 평가를 통해 제안한 방법에 대한 효과를 입증하였다. 5장에서는 결론 및 향후 연구과제로 이 논문을 마무리하였다.

II. Preliminaries

1. Related works & Background

1.1 Naval Combat Management System

함정전투체계(Naval Combat Management System)는 단일 무기체계를 위한 사격통제기술 뿐만 아니라, 다중 센서와 무장을 효율적으로 통제하고 제어하는 역할을 한다. 대한민국은 2000년대 초를 기점으로 국산 함정전투체계의 개발에 착수했다[6]. 현재에 이르러서 많은 수의 함정에 국산 함정전투체계가 채택되어 운용되고 있다.

함정전투체계 소프트웨어의 개발 프로세스에는 체계요구분석에 따른 기본/상세 설계가 선행되고, 이후 구성품의 설계와 제작이 수행된다. 이후 공장수락시험, 육상체계통합시험, 개발시험평가, 운용시험평가 등의 시험을 거쳐 체계검증시험 단계가 완료된다. 함정전투체계 소프트웨어를 형상 구분 기준에 따라 나누자면, 형상관리를 하는 단위인 CSCI(Computer Software Configuration Item)가 있고, 그 하위에는 독립적으로 배포가 가능한 CSC(Computer Software Component)가 있으며, 그 하위에는 독립적으로 테스트 가능한 단위인 CSU(Computer Software Unit)가 있다. CSCI, CSC, CSU는 개발 프로세스의 단계에 따라서 하위 단위에서부터 순차적으로 검증된다.

대표적인 함정전투체계 소프트웨어로는 센서/무장/통신 등의 다양한 외부 장비와 연결되어 해당 장비를 통제하는 연동 CSCI, 항해 관련 지원 기능 등 보조적인 기능을 담당하는 체계지원 CSCI, 체계를 제어하거나 상태를 감시하는 체계관리 CSCI 등으로 분류된다. Fig. 1.은 함정전투체계의 시스템 아키텍처를 간략하게 도식화하여 나타냈다. 함정전투체계 소프트웨어는 Fig. 1.에 있는 각 하드웨어 노드에 탑재되어 운용된다.

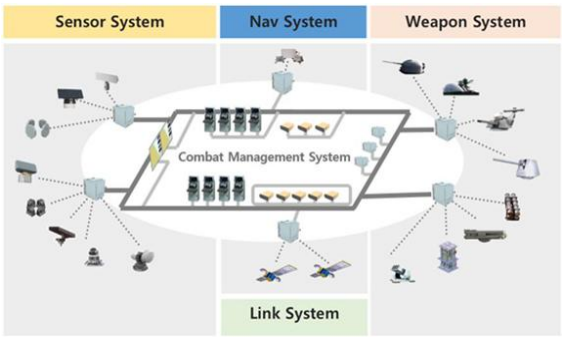

Fig. 1. System Architecture of Naval Combat Management System

국산 함정전투체계의 내부 노드 간 통신은 DDS(Data Distribute Service)을 통해 이루어진다. DDS는 비영리 기구인 OMG(Objective Management Group)에서 개발한 표준으로, 분산 시스템에서 데이터를 교환 및 공유하며, 미들웨어의 Publication/Subscribe 형식을 사용한다[7][8].

1.2 System Management CSCI Software

체계관리(System Management) CSCI 소프트웨어는 승조원의 운용에 따라 체계를 제어하거나 감시하는 다양한 기능을 포함한다. 체계관리 CSCI 소프트웨어에 포함되는 CSC에는 경고 관리, 날짜 및 시간 관리, 운용자 관리, 체계 상태 관리, 파라미터 관리 등이 있다.

체계관리 CSCI 소프트웨어를 포함한 함정전투체계 소프트웨어들은 함형 별 공통요소와 가변요소를 갖고 있다. 일반적으로 신규 함정전투체계 개발 시 신규 기능 요구사항이 추가되는 경향이 있는데, 이는 함정전투체계 도메인의 가변요소로 분류할 수 있다. 체계관리 CSCI 소프트웨어는 함정전투체계에 필수적으로 포함되는 CSCI이면서 새로운 기능에 대한 소요가 많아 신규 기능 요구사항이 추가되는 경우가 많고, 추가된 항목은 다음 함형에 승계된다. 이러한 특징으로 인해 체계관리 CSCI 소프트웨어에 표준화와 같은 구조적 최적화를 적용할 경우, 그 장점이 극대화될 수 있다.

1.3 Modularization

모듈화(Modularization)는 일반적으로 높은 유지보수성과 재사용성을 달성하기 위해 대상물을 독립적인 모듈로 분리하는 것을 의미한다. 모듈화를 위해서는 모듈 간 호환성이 필요한데, 이를 보장하려면 모듈과 모듈 사이의 상호작용을 단순화하여 적은 수의 인터페이스를 가지게 해야 한다. 그리고 이 인터페이스를 표준화하여, 모듈 내부 구성요소의 변경에 상관없이 타 모듈과 상호작용이 가능하도록 해야 한다[9].

모듈화는 체계관리 CSCI 소프트웨어와 같이 공통요소의 비율이 높고 가변요소의 비율이 적은 제품군에 적용하면 그 유지보수성을 극대화할 수 있다. 모듈화에는 각 기능 요구사항을 클래스 단위로 분리하는 표준화가 포함되므로, 차기 함형 함정전투체계를 개발하더라도 기존 제품군에서 필요한 부분만 손쉽게 골라서 사용할 수 있기 때문이다. 이러한 모듈 형태의 소프트웨어 재사용은 함정전투체계의 개발 기간과 개발 비용을 줄이는 방안 중 하나다.

1.4 Naval Shield Component Platform

본 논문에서 언급되는 표준 연동 아키텍처는 NSCP (Naval Shield Component Platform)를 말하는 것으로, 함정전투체계 소프트웨어의 효율적인 개발을 위해 제시된 공용 아키텍처이다[3].

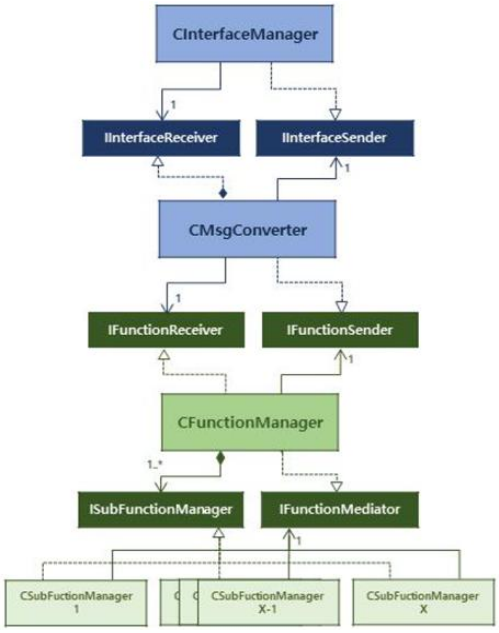

Fig. 2. Simplified Class Diagram of Naval Shield Component Platform

이 아키텍처는 객체 지향 프로그래밍과 그와 관련된 다섯 가지 기본 설계 원칙인 SOLID[10]를 준수하고, 연동 CSCI 소프트웨어를 위한 표준 연동 아키텍처로 시작하였으며, OO함에 적용되어 그 유용성을 검증하였다. 그리고 그 설계 구조상 연동 CSCI 소프트웨어뿐만 아니라 다른 CSCI 소프트웨어에도 적용할 수 있을 만큼 높은 범용성, 확장성 및 유지보수성을 가진다[3].

NSCP의 개략적인 클래스 다이어그램은 Fig. 2.로, 크게 4개 종류의 요소 클래스로 구성되고, 각 요소 클래스는 인터페이스 클래스를 통해 간접적으로 연결된다[3].

1.5 Feature Model

모듈화 패러다임 중 하나인 소프트웨어 프로덕트 라인(Software Product Line)은 하나 이상의 여러 제품을 재사용 가능한 소프트웨어 자산을 통해 개발하는 것을 의미한다. 소프트웨어 프로덕트 라인에서 가장 중요한 것은 공통성과 가변성을 어떻게 식별하고 표현할 것인가 하는 점이다[11]. 도메인 분석을 위한 다양한 기법 중 본 논문에서는 휘처 모델을 통해 도메인 모델링을 수행하고자 한다. 휘처 모델이란 휘처 간의 관계, 다시 말해 구성요소들의 관계를 활용하여 제품 정보를 표현하는 정보 모델이다[12]. 휘처 모델은 제품 기능을 중심으로 관계 표현이 가능하므로 함정전투체계의 체계관리 CSCI 소프트웨어가 포함하고 있는 다양한 기능을 도식화하는 데 적합하다.

본 논문에서 언급될 휘처 요소는 2가지로, mandatory 요소와 optional 요소다. mandatory 요소는 부모 휘처 입장에서 자식 휘처가 필수 요소인 경우이고, optional 요소는 부모 휘처 입장에서 자식 휘처를 선택하는 것이 옵션인 경우를 말한다[12]. 본 논문에서는 체계관리 CSCI 소프트웨어의 구성단위를 휘처 요소로 표현한다. 그리고 각 함형에 따라 코드 수정 없이 재활용할 수 있는 휘처 요소일 경우 mandatory 요소, 즉 공통요소로 판단한다. 그리고 각 함형에 따라 추가, 변경, 삭제될 수 있는 휘처 요소일 경우 optional 요소, 즉 가변요소로 판단한다.

III. The Proposed Scheme

본 장에서는 체계관리 CSCI 소프트웨어의 모듈화 적용 전후 구조를 비교하여 설명한다. 휘처 모델을 활용하여 공통요소와 가변요소를 식별 후, 모듈화의 수단으로써 표준 연동 아키텍처를 적용했다. 그리고 체계관리 CSCI 소프트웨어 중 경고 관리 CSC와 날짜 및 시간 관리 CSC 2개를 선행적으로 모듈화하였다.

1. Analysis of System Management CSCI SW with Feature Model

함정전투체계 도메인의 공통요소와 가변요소를 분리하여 구현하는 것이 중요한 이유는, 소프트웨어 내 공통요소를 최대화함으로써 개발 및 유지보수 비용 절감이 가능하기 때문이다. 따라서 본 논문의 목표는 체계관리 CSCI 소프트웨어의 공통요소와 가변요소를 판별하고, 기능 요구사항의 변경을 반영하기 쉽도록 모듈화를 적용하는 것이다.

Fig. 3.은 기존 체계관리 CSCI 소프트웨어 중 대표적인 CSC를 간추려 휘처 모델로 표현한 것이다. 황색 영역인 Common Factor는 공통요소를, 분홍색 영역인 Variable Factor는 가변요소를 말한다. CSC는 CSCI 내 개별 소프트웨어 응용 단위, CSU는 각 소프트웨어 응용의 기능 요구사항 단위로 정의한다. 그리고 각 CSU의 기능 요구사항의 동작에 필요하면서, 코드 내부에서 부분적으로 변경될 수 있는 코드 요소는 element라고 정의하였다. Fig. 3.에서는 다음과 같이 세 가지 타입의 CSC가 있다.

(1) 해당 CSC 내 요소 전체가 공통요소인 경우:
- 파라미터 관리 CSC (SYSPARAM CSC)

(2) 해당 CSC 내 요소 전체가 가변요소인 경우:
- 체계상태감시 관리 CSC (SYSSTAT CSC)
- 운용자 관리 CSC (ACCOUNT CSC)

(3) 해당 CSC 내 요소가 공통/가변요소 혼재인 경우:
- 경고 관리 CSC (WARNING CSC)
- 날짜 및 시간 관리 CSC (TIME CSC)

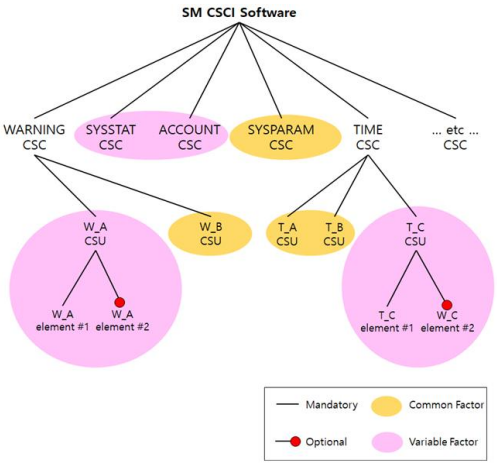

Fig. 3. Feature Model of System Management CSCI Software

(3)의 경우는 신규 기능 요구사항에 따라 기능이 새로 구현된 사례거나, 또는 함정전투체계 내 노드 수에 종속적인 기능이 있어서 함형에 따라 코드가 변경되는 경우이다. 경고 관리 CSC와 날짜 및 시간 관리 CSC는 후자로, 경고 관리 CSC는 함정전투체계 내 각 노드에서 발생하는 경고 정보 수신/관리 및 전시하고, 날짜 및 시간 관리는 각 노드에서 전시되는 시간 정보를 관리하는 역할을 하기 때문이다. Table 1.은 앞서 언급된 체계관리 CSCI 소프트웨어의 공통요소와 가변요소들을 간략하게 정리한 것이다.

Table 1. Classification of Factors

Classification	Function
Common Factor	SYSPARAM CSC
	WARNING CSC (W_B CSU)
	TIME CSC (T_A / T-_B CSU)
Variable Factor	SYSSTAT CSC
	ACCOUNT CSC
	WARNING CSC (W_A CSU)
	TIME CSC (T_C CSU)

2. Software Design & Implementation

함정전투체계 소프트웨어를 모듈화하기 위해서는 각 응용의 공통요소를 최대한 분리하고, 가변요소는 기능 요구사항 단위의 클래스로 나누는 작업이 요구된다. 본 논문에서는 특정 소프트웨어에 이 작업을 선행적으로 적용 후 그 성과를 확인하고자 한다. Table 1.에서 분류된 공통요소와 가변요소를 바탕으로, 체계관리 CSCI 소프트웨어 중 모듈화를 선행적으로 적용할 CSC는 날짜 및 시간 관리 CSC와 경고 관리 CSC로 결정되었다. Fig. 4. 및 Fig. 5.는 두 CSC의 기존 클래스 다이어그램 구조를 나타낸다.

Fig. 4.와 Fig. 5.을 통해 함정전투체계 내부의 DDS 송수신을 위한 CallBackHandler 및 CDataManager 클래스를 제외한다면 두 CSC는 1~2개의 클래스로만 기능함을 볼 수 있다. Fig. 4.에서 CWarningMgr은 소프트웨어 초기화와 기능 요구사항의 동작을 한 번에 구현한 메인 클래스다. Fig. 5.에서 MainSMTimeDateMgr은 소프트웨어 초기화를 구현한 메인 클래스이고, CSMTimeDateMgr은 기능 요구사항의 동작을 구현한 서브 클래스다.

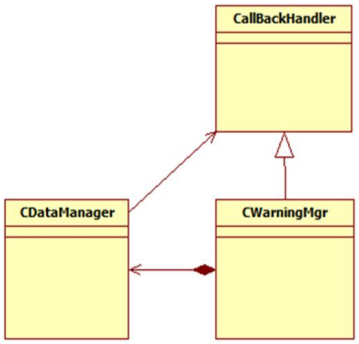

Fig. 4. Previous Class Diagram of WARNING CSC

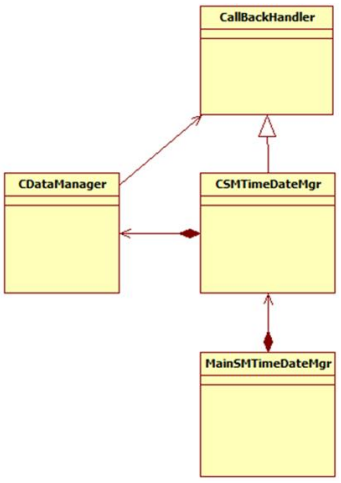

Fig. 5. Previous Class Diagram of TIME CSC

만약 각 CSC의 기능을 Fig. 4.와 Fig. 5.와 같이 소수의 클래스로 구현할 경우, 기능 요구사항이 추가, 변경, 삭제될 때 다수의 클래스가 변경될 확률이 높다. 변경 부분에 대해 재검증시험이 필요한 업계 특성상, 이러한 클래스 구조는 개발 기간 및 비용 증가의 원인 중 하나다.

그리고 Fig. 6.과 Fig. 7.은 모듈화를 적용한 후, Fig. 2.의 형식에 맞추어 두 CSC 클래스 다이어그램 구조를 나타낸 그림이다. 이는 모듈화를 적용하여 공통요소와 가변요소가 분리되도록 재편한 클래스 구조로, 주요 클래스 위주로 개략적으로 나타냈다.

Fig. 4.와 Fig. 5.에서 각 CSC의 메인 클래스였던 CWarningMgr과 MainSMTimeDateMgr은 Fig. 6.과 Fig. 7.의 CTIMEMain과 CWARNINGMain으로 변경되면서 소프트웨어 초기화의 역할만 담당하게 되었다. 단, 두 메인 클래스는 Fig. 2.와 동일한 구조로 그리기 위해 다이어그램에서는 생략되었다.

Fig. 4.와 Fig. 5.에서 각 CSC의 기능 요구사항 구현 클래스였던 CWarningMgr과 CSMTimeDateMgr은 모듈화 후 기능 요구사항 수만큼 분리되어, Fig. 6.과 Fig. 7.와 같이 2~3개의 CSubFunctionManager로 매핑되었다.

Fig. 6.과 Fig. 7.의 클래스 중 CFunctionManager, CMsgConverter, CInterfaceManager 클래스는 모듈화를 적용하면서 추가된 클래스다. CFunctionManager는 각 기능 요구사항을 구현한 CSubFunctionManager를 관리한다. CMsgConverter는 전투체계 내 다른 소프트웨어와 DDS 송수신을 위한 메시지 포맷 변경을 수행하고, CInterfaceManager는 함정전투체계 내 다른 소프트웨어와 DDS 송수신을 중개한다.

Fig. 4.와 Fig. 5.의 함정전투체계 내부의 DDS 송수신을 위한 CallBackHandler 및 CDataManager 클래스는

Fig. 6.과 Fig. 7.의 CInterfaceManager의 하위에 연결되는 클래스가 되었다. 다만 이들 클래스는 Fig. 2.와 동일한 구조로 그리기 위해 Fig. 6과 Fig. 7.에서는 생략되었다. 마찬가지로 이외에도 또 다른 인터페이스와 추상 및 구현 클래스 등 다양한 클래스가 있으나[3], 제한된 지면으로 인해 생략되었다.

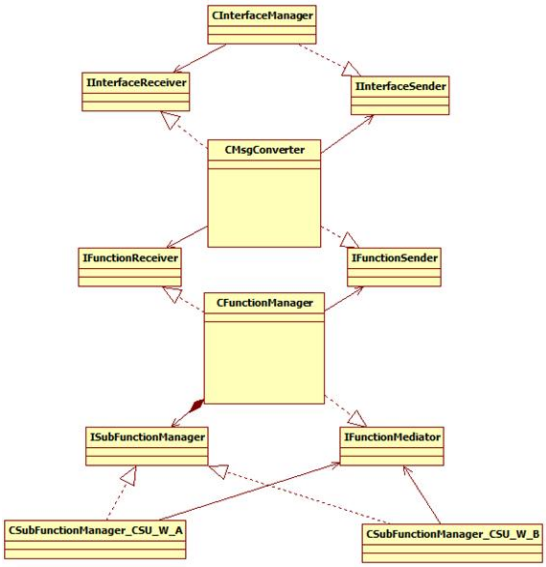

Fig. 6. Revised Class Diagram of WARNING CSC

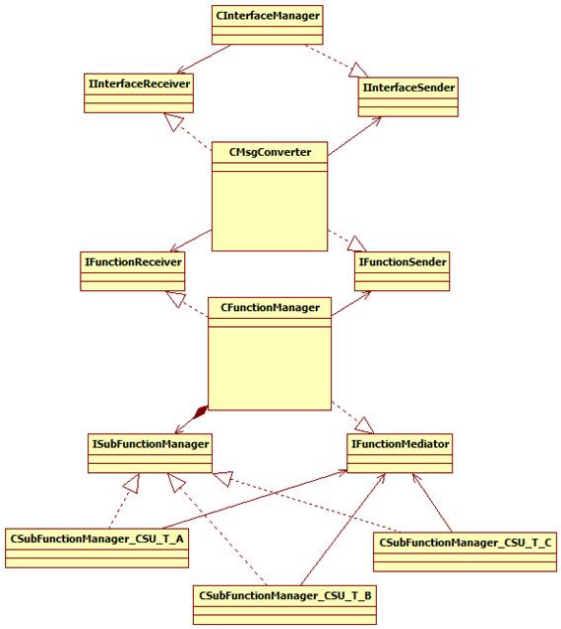

Fig. 7. Revised Class Diagram of TIME CSC

변경된 구조에서 핵심 기능을 수행하는 각 주요 클래스들은 인터페이스를 통해 서로 유연하게 연결되어 있다. 특히 주목할 점은 Fig. 3.의 CSU에 해당하는 각 기능 요구사항이 Fig. 6.과 Fig. 7.의 CSubFunctionManager 클래스로 명확히 분리되었다는 점이다. 이렇게 되면 각 CSCI 소프트웨어의 기능 요구사항에 변동사항이 있을 때 CSubFunctionManager 클래스를 추가, 변경, 삭제함으로써 수정사항을 손쉽게 반영할 수 있다. 이러한 소스 코드 수정의 최소화는 소프트웨어 신뢰성 시험 수행에 대한 범위를 축소하고 재사용률을 높이는 이점을 가진다[13].

IV. Evaluations

본 장에서는 함정전투체계 체계관리 CSCI 소프트웨어에 모듈화를 적용한 후 효율성을 평가하기 위해 실험 평가를 수행하였다. 개선 정도를 평가할 방법은 크게 두 가지다. 첫 번째는 기능 요구사항 수정 시 변경되는 클래스 비율의 비교고, 두 번째는 방위산업에서 필수적으로 수행하는 소프트웨어 신뢰성 시험 수행 시간 비교이다.

평가에는 체계관리 CSCI 소프트웨어 중 경고 관리 CSC와 날짜 및 시간 관리 CSC를 활용한다. 평가를 위한 시험 환경 구성은 Fig. 8. 및 Table 2.와 같다.

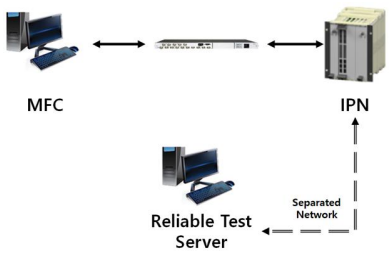

Fig. 8. Test Environment

Table 2. Test Environment

Item	MFC	IPN
CPU	Intel Core i7-6700 @3.40GHz	Intel Core i7-4700 EQ @ 2.40GHz
Memory	8 GB	8 GB
OS	Windows 7	RTST Linux

Fig. 8.의 다기능콘솔(MFC)과 체계관리 CSCI 소프트웨어가 운용되는 하드웨어 노드인 정보처리장치(IPN)는 스위치로 동일 네트워크에 연결되어 있다. 소프트웨어 신뢰성 시험 서버는 다기능콘솔 및 정보처리장치와 분리된 네트워크에 구성된다. 시험환경 운용자는 다기능콘솔을 조작함으로써 체계관리 CSCI 소프트웨어의 기능을 확인할 수 있는데. 만약 시험환경 운용자가 특정 CSC 응용의 기능을 수행하게 되면, 해당 노드 내에 소프트웨어 신뢰성 시험 분석용 raw 데이터가 생성된다. 신뢰성 시험 분석용 raw

데이터는 응용 코드 내 구문의 수행 여부를 판단하기 위해 활용된다. 시험환경 운용자는 보안 USB를 통해 이 raw 데이터를 분리된 망에 있는 소프트웨어 신뢰성 시험 서버로 옮기고, 소프트웨어 신뢰성 시험 서버에서는 해당 데이터를 기반으로 분석 결과를 도출한다. 개발자는 해당 결과를 바탕으로 수행할 수 없는 구문이 있는지 식별하고, 필요 시 코드를 수정한 후 신뢰성 시험을 추가 수행한다.

1. Comparison of and Modified Class Ratio

표준 연동 아키텍처에 관한 논문[3]에서는 표준 연동 아키텍처를 적용할 경우 기존 대비 구조적 복잡도는 상승하나, 기존 대비 공용 클래스 포함 비율의 향상과 높은 재사용성을 확보할 수 있다고 밝혔다. 본 논문에서도 이와 같은 경향성을 확인하기 위해, 경고 관리 CSC와 날짜 및 시간 관리 CSC에서 기능 요구사항 1개의 내용이 수정되었다고 가정하였다. 그리고 Table 3.과 Table 4.와 같이 표준 연동 아키텍처를 적용하기 전과 후의 클래스 수와 기능 요구사항 수정 시 변경되는 클래스 비율 등을 비교하였다.

Table 3. Comparison of Class Quantity

Class Quantity	WARNING CSC		TIME CSC	
	before	after	before	after
Number of Total Classes (α)	3	29	4	30
Number of Common Classes (β)	2	28	3	29
Number of Modified Classes after Functionality Addition (γ)	1	1	1	1

Table 4. Comparison of Ratio

Ratio	WARNING CSC		TIME CSC	
	before	after	before	after
Common Class Ratio (β/α)	66.6%	96.6%	75.0%	96.7%
Modified Class Ratio after Functionality Addition (γ/α)	33.3%	3.4%	25.0%	3.3%

Table 4.의 결과는 소수점 둘째 자리에서 반올림한 값이며, Table 3.과 Table 4.의 각 항목은 다음과 같다.

α : 전체 클래스 수
β : 공통 클래스 수
γ : 기능 요구사항 수정 시 변경되는 클래스 수
β/α : 공통 클래스 비율
γ/α : 기능 요구사항 수정 시 변경되는 클래스 비율

Table 3.에서 경고 관리 CSC와 날짜 및 시간 관리 CSC의 전체 클래스 수 α는 표준 연동 아키텍처 적용 전보다 각각 26개만큼 증가하였다. 이러한 클래스 수의 과도한 증가는 표면적으로는 단점처럼 느껴진다. 하지만 이러한 증가분은 표준 연동 아키텍처 라이브러리에 정의된 인터페이스와 추상 클래스 수를 모두 합한 수다. 그리고 이러한 클래스들은 각 요소 간의 상호의존성을 낮추어 유지보수성 및 확장성을 높이기 위한 모듈화의 기반으로, 개발자가 직접 구현해야 하는 요소는 아니다. 따라서 아키텍처 적용 후 구조에서 실질적으로 개발자가 소프트웨어 개발 및 유지보수를 해야 하는 클래스 수는 인터페이스와 추상 클래스를 제외한 일부 소수의 구현클래스라고 할 수 있다. 그 예로는 CInterfaceManager/CMsgConverter의 자식 클래스와 CFunctionManager, CSubFunctionManager 클래스가 있다. 이번 평가의 전제인 '기능 요구사항 1개의 내용 수정'과 같은 상황인 경우, 변경해야 할 클래스 수는 모듈화 전과 동일하게 CSubFunctionManager 1개 클래스만 변경하면 되는 상황이다.

한편, Table 4.에서는 경고 관리 CSC와 날짜 및 시간 관리 CSC의 공통 클래스 비율(β/α)이 각각 30.0 %p, 21.7 %p 증가했다. 그리고 기능 요구사항 수정 시 변경되는 클래스 비율(γ/α)이 각각 29.9 %p, 21.7 %p 감소했다. 이와 같은 증감은 체계관리 CSCI 소프트웨어가 기존보다 공용 클래스 비율이 늘고, 가변 클래스 비율이 줄었음을 나타낸다. 다시 말해 공통요소와 가변요소의 분리라는 모듈화의 특징이 반영되었음을 의미한다. 이 결과를 확장하여 기능 요구사항이 1개가 추가, 삭제되는 상황을 가정한다면, CSubFunctionManager 객체 1개의 추가, 삭제하면 된다. 그리고 해당 기능 요구사항에 필요한 DDS 메시지에 수정 소요가 있다면 DDS 송수신 관련 클래스의 소스코드 일부만 변경하면 된다.

결과적으로 Table 3.에서 보이는 클래스 수의 변화는 표준 연동 아키텍처를 적용한 모듈화에서 필연적으로 발생하는 특징이고, 실질적으로 개발자가 개발 또는 유지보수 시에 소모되는 추가 리소스가 없다. 그리고 Table 4.의 결과를 통해 기존 구조 대비 높은 유지보수성과 재사용성을 가지는 구조로 바뀌었다고 평가할 수 있다.

2. Comparison of Software Reliability Test

신뢰성 시험은 사전 설계를 통해 식별된 소프트웨어 기능 요구사항에 모두 충족하는 수행절차서를 작성 후 준비된 신뢰성 시험장에서 시험을 수행했을 때, 해당 CSCI 소

프트웨어의 모든 구문이 수행되는지 확인되면 수행 완료의 기준으로 판단한다[14]. 소프트웨어 신뢰성 시험은 MISRA, Code Sniper 기반의 정적 시험과 Quality Scroll Cover 기반의 동적 시험으로 진행된다.

모듈화 전/후의 신뢰성 시험 결과를 비교하기 위한 전제로, 각 CSC의 기능 요구사항 1개의 내용이 수정되면서 코드 변경사항이 생겨 추가적인 신뢰성 시험 소요가 발생한 상황을 가정하였다. Table 5.는 각 조건에 따라 소프트웨어 신뢰성 시험의 수행에 걸리는 시간을 정리한 표다. Fig. 9.의 항목은 각각 다음과 같다.

(1) 정적 시험을 처음부터 수행한 경우
(2) 정적 시험 완료 후, 기능 요구사항 1개의 내용이 수정되어 코드 변경 후 정적 시험을 추가 수행한 경우
(3) 동적 시험을 처음부터 수행한 경우
(4) 동적 시험 완료 후, 기능 요구사항 1개의 내용이 수정되어 코드 변경 후 동적 시험을 추가 수행한 경우

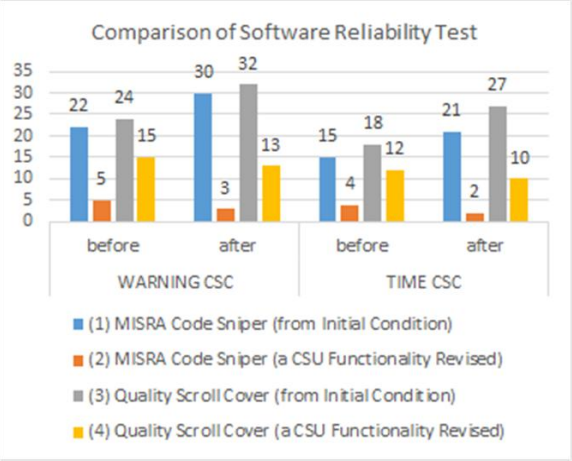

Fig. 9. Comparison of Software Reliability Test

Fig. 9.에서는 경고 관리 CSC와 날짜 및 시간 관리 CSC의 모듈화 전후의 신뢰성 시험 수행 시간을 비교할 수 있다. 정적 시험인 (1)에서는 변경 전 구조에 비해 소요 시간이 각각 36.4 %, 40 % 만큼 증가하였다. 그리고 동적 시험인 (3)에서 변경 전 구조에 비해 소요 시간이 각각 33.3 %, 50.0 % 만큼 증가하였다. 이는 표준 연동 아키텍처를 적용함으로써 기존보다 클래스 수 자체가 많아지고 구조가 복잡해지므로 생기는 불가피한 단점이다.

반면 각 신뢰성 시험을 완료 이후, 기능 요구사항 1개의 내용이 수정되어 코드 변경 후 추가 신뢰성 시험을 수행한 (2)와 (4)의 경우에서는 수행 시간이 감소하는 것을 알 수 있다. 추가 정적 시험인 (2)의 경우 변경 전 구조에 비해 소요 시간이 각각 40.0 %, 50.0 %만큼 감소하였다. 그리고 추가 동적 시험인 (4)의 경우 기존 구조에 비해 소요 시간이 각각 13.3 %, 16.7 %만큼 감소함을 확인하였다.

위 결과를 바탕으로 분석하자면, 표준 함정 아키텍처를 적용하여 모듈화된 체계관리 CSCI 소프트웨어의 최초 소프트웨어 신뢰성 시험에는 기존 구조보다 수행 시간이 오래 소요된다. 하지만 체계관리 CSCI 소프트웨어의 기능 요구사항의 변화가 있을 때 소프트웨어 신뢰성 시험의 재수행 시간은 적게 소요된다. 따라서 모듈화를 적용한 최초 시점에 소프트웨어 신뢰성 시험을 잘 수행 완료하였다면, 추후 기능 요구사항 변경 시 소프트웨어 신뢰성 시험으로 인해 소비되는 리소스를 줄일 수 있다.

그리고 이 분석 결과를 확장한다면 차기 함정전투체계를 개발할 때 베이스라인이 변경되는 상황에서도, 표준 연동 아키텍처를 준용한다면 클래스 구조가 바뀌지 않기 때문에 소프트웨어 신뢰성 시험의 소요 시간이 감소할 것으로 기대할 수 있다. 또한, 차기 함정전투체계에서 체계관리 CSCI 소프트웨어의 기능 요구사항이 추가, 삭제되는 경우를 가정했을 때에도 동일한 효과를 보일 것이다. 해당 기능 요구사항이 구현된 CSubFunctionManager 클래스와, 그와 관련된 DDS 메시지 송수신 클래스에 대해서만 추가 소프트웨어 신뢰성 시험을 수행하면 되기 때문이다.

본 논문은 위와 같이 첫 번째로 기능 요구사항 수정 시 변경되는 클래스 비율, 두 번째로 소프트웨어 신뢰성 시험 수행 시간 비교를 기반으로 한 실험 및 평가를 수행하였다. 그리고 이를 통해 체계관리 CSCI 소프트웨어에 표준 연동 아키텍처를 적용한 모듈화를 수행함으로써 유지보수성 및 재사용성 측면에서 개선되었다는 것을 확인하였다.

V. Conclusions

함정전투체계에서 체계관리 CSCI 소프트웨어는 필수적인 기능이며, 함형과 관계없이 다수의 공통요소를 공유하는 소프트웨어다. 그럼에도 불구하고 신규 함형 함정전투체계의 개발은 각 함정전투체계 소프트웨어 모듈의 재사용이 아닌, 이미 검증된 소스 코드의 일부를 재사용 수준으로 개발되어왔다. 이는 다양한 원인에 의해 모듈화적 관점이 소스 코드에 충분히 반영되지 않은 결과물로, 함정전투체계 개발 이해관계자의 비효율적인 공수 활용을 일으켜, 함정전투체계의 개발 시간과 개발 비용의 증가를 초래하기도 했다.

본 논문에서는 체계관리 CSCI 소프트웨어 중에서 경고관리 CSC와 날짜 및 시간 관리 CSC에 모듈화를 선행적으로 적용 후 그 성과를 연구하였다. 실험 평가를 수행하기 위해서, 기능 요구사항에 수정이 발생한 경우 변경되는 클래스 비율 감소, 소프트웨어 신뢰성 시험 수행 시간 감소와 같은 실험 결과를 도출하였다. 그리고 이 결과를 바탕으로 모듈화가 적용된 체계관리 CSCI 소프트웨어는 기존 대비 높은 유지보수성과 재사용성을 가짐을 입증하였다.

향후 연구과제로는 모듈화가 아직 적용되지 않은 함정전투체계의 타 소프트웨어 응용에 적용하는 방안 및 그 효과를 연구하고자 한다. 이러한 함정전투체계 소프트웨어를 안정화하고 발전시키고자 하는 노력의 방향은, 함정전투체계 개발 이해관계자의 공수 활용 효율화와 개발 기간 및 개발 비용 감소에 긍정적인 영향을 주리라 기대한다.

REFERENCES

[1] Han Jonghwan, "A Study on the Changes of Security Environment and the Roles of the ROKN," The Journal of Strategic Studies, Vol. 29, No. 1, pp. 233-261, 2022. DOI: 10.46226/jss.2022.04.29.1.233

[2] Dong-Uk Kim, Hyung-Rok Jung, and Young-Il Song, "A Study on Cost Behavior of Korean Defense Industry," KOREAN JOURNAL OF MANAGEMENT ACCOUNTING RESEARCH, Vol. 11, No. 2, pp. 55-82, 2011.

[3] Chi-Sun Baek, and Jin-Hyang Ahn, "A Study of the Standard Interface Architecture of Naval Combat Management System," Journal of the Korea Society of Computer and Information, Vol. 26, No. 1, pp. 147-154, 2021.

[4] Dong-Kwan Kim, Dong-Han Jung, Won-Seok Jang, Young-San Kim, and Hyo-Jo Lee, "A Study on Efficient Design of Surveillance RADAR Interface Control Unit in Naval Combat System," Journal of the Korea Society of Computer and Information, Vol. 28, No. 11, pp. 125-134, 2023.

[5] Yeon-Hee Noh, Dong-Han Jung, Young-San Kim, and Hyo-Jo Lee, "A Study on the Standard Architecture of IFF Interface SW in the Naval Combat Management System," Journal of the Korea Society of Computer and Information, Vol. 29, No. 1, pp. 139-149, 2024.

[6] Yeong-Ju Kim, "The Development Direction of Naval Weapon Systems According to Future Battlefield Environment Changes," Defense & Technology, No. 311, pp. 70-83, 2005.

[7] DDS, https://www.omg.org/omg-dds-portal/

[8] Data Distribution Service for Real-Time Systems Specification, OMG, March 2004.

[9] Chang-Mook Kang, "Complexity Cost and Modular Architecture," Industrial Engineering Magazine, Vol. 25, No. 1, pp. 35-41, 2018.

[10] Robert C. Martin, "Clean Code: A Handbook of Agile Software Craftsmanship," Prentice Hall, New Jersey, pp. 138-140, 2008.

[11] Minseong Kim, and Sooyong Park, "A Domain Analysis Method for Software Product Lines Based on Goals, Scenarios, and Features," Journal of KISS : Software and Applications, Vol. 33, No. 7, pp. 589-604, 2006.

[12] Chee-Yang Song, Eun-Sook Cho, and Chul-Jin Kim, "A Formal Specification and Checking Technique of Feature model using Z language," Journal of the Korea Society of Computer and Information, Vol. 18, No. 1, pp. 123-136, 2013.

[13] Ji-Yoon Park, Moon-Seok Yang, and Dong-Hyeong Lee, "A Study on IISS Software Architecture of Combat Management System for improving modifiability," Journal of the Korea Society of Computer and Information, Vol. 25, No. 2, pp. 133-140, 2020.

[14] Hwan-Jun Choi, "A Study on the Software Standardization and Simulator Design for Efficient Reliability Test in Combat System," Journal of the Korea Society of Computer and Information, Vol. 27, No. 12, pp. 151-159, 2022.

Authors

Hyeon-Tae Ha received the B.S. degree in Electronic Engineering from Kyungpook National University, Korea, in 2016. He is currently an engineer in Hanwha Systems. He is interested in Naval Combat Management System.