소프트웨어 역공학을 통한 HeliScope OFP 의 테스트 케이스 생성

이종훈, 이동아, 유준범, 송승화, 김두현

건국대학교 정보통신대학
서울 광진구 화양동 1 번지 건국대학교
{kirdess, ldalove, jbyoo, sshtel, doohyun}@konkuk.ac.kr

요약: HeliScope OFP(Operational Flight Program)는 건국대학교 무인비행체 소프트웨어 융합연구센터에 의해 개발된 소형 무인 헬리콥터의 비행 제어를 위한 실시간 내장형 소프트웨어이다. 무인 헬리콥터의 비행 제어 컴퓨터에서 동작하며, 비행 상태에 대한 정보를 받아 비행 제어를 위한 연산을 수행한다. 따라서 OFP 에 결함이 발생할 경우 비행 상태와 직결되어 오동작을 일으킬 수 있으며, 이는 임무 실패 및 기체 손실 등의 피해를 야기할 수 있다. 이러한 위험 상황 발생을 방지하기 위해 HeliScope OFP 를 대상으로 강도 높은 검증 활동이 이루어져야 할 필요가 있다. 본 논문에서는 이미 개발이 완료된 HeliScope OFP 를 대상으로 검증 활동을 수행하기 위하여 소프트웨어 역공학을 적용하였으며, 이를 통해 소프트웨어의 정보를 추출 및 복원하는 내용과 이들 정보를 이용하여 HeliScope OFP 를 위한 테스트 케이스를 개발한 결과를 소개한다.

핵심어: HeliScope Project, OFP(Operational Flight Program), Software Reverse Engineering, Software Testing

1. 서 론

소프트웨어의 동작의 정확성 안정성을 확인하고 검증(Validation & Verification)하는 것은 소프트웨어 개발에 있어 중요한 문제이다. 특히 항공이나 철도와 같은 분야에서 사용되는 소프트웨어는 결함 발생이 인명 피해와 같은 사고를 유발할 위험성을 지니며, 소프트웨어 검증 활동이 굉장히 중요한 항목으로 요구되고 있다. 그러나 일부 소프트웨어의 경우 개발 과정에 체계적인 검증 활동이 구성되어 있지 않거나, 개발이 끝난 뒤에 이를 적용하고 있다. 이들 소프트웨어를 대상으로 검증 활동을 적용하기 위해서는 소프트웨어의 정보를 검증 활동의 입력으로 활용하기에 충분한 수준인지 확인하고, 충분하지 않을 경우 이를 보완해야 할 필요가 있다.

HeliScope OFP 는 HeliScope Project[1]의 소형 무인 헬리콥터의 비행 제어를 위해 개발된 소프트웨어로, 비행 상태를 실시간으로 파악하고 이를 제어한다. 실시간으로 동작하며 헬리콥터의 비행 상태를 제어하기 때문에 결함이 발생할 경우 헬리콥터의 오동작 및 추락 등의 높은 위험도를 지니는 상황을 유발할 수 있다. 때문에 이러한 위험 상황 발생을 방지하기 위해 엄격한 검증 활동이 이루어져야 할 필요가 있다. 또한 HeliScope OFP 는 이미 개발이 완료되었기 때문에 검증 활동을 적용하기 위해 소프트웨어의 정보를 분석하여 검증을 수행하기에 충분한 수준인지를 확인하고, 부족하다면 이를 보완해야 할 필요가 있다. 본 논문에서는 이전 연구에서 제시하였던 HeliScope OFP 를 위한 검증 절차[2]에 기반하여, 소프트웨어 역공학 기술을 적용하여 설계와 요구사항 정보를 보완하고, 이들 자료를 기반으로 소프트웨어 테스팅을 위한 테스트 케이스를 작성한 결과를 소개한다.

본 논문은 다음과 같이 구성된다. 제 2 장에서는 HeliScope Project 의 OFP 와 이전 연구에 대해서 간략히 소개하고, 적용한 활동들에 대하여 설명한다. 제 3 장에서는 적용된 소프트웨어 역공학 기술들과 산출물을, 제 4 장에서는 3 장에서 작성된 산출물을 이용하여 작성한 테스트 케이스에 대하여 설명한다. 마지막으로 제 5 장에서는 결론 및 향후 연구 방향에 대하여 정리한다.

2. 배경 및 관련 연구

2.1 HeliScope Project

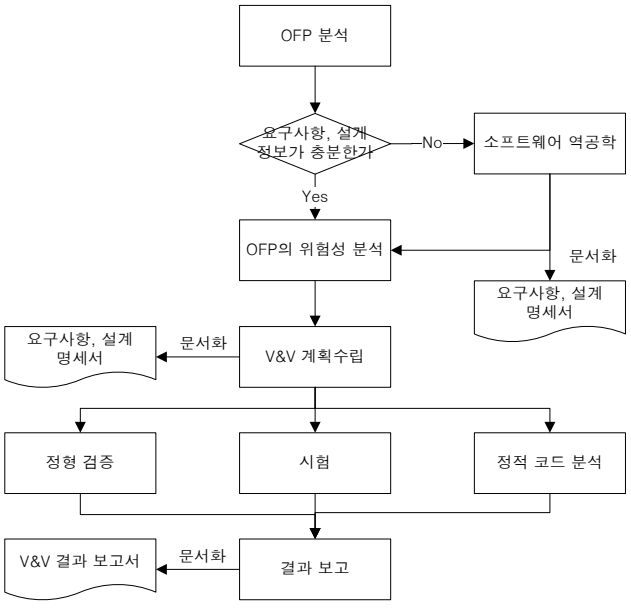
HeliScope Project 는 무인 헬리콥터에 장착된 카메라를 이용하여 방재, 재난 복구 등의 중요 임무를 수행할 소형 무인 헬리콥터를 개발할 목적으로 진행된 프로젝트이다. 무인 헬리콥터는 재난 현장으로 빠르게 이동하여 현장의 영상을 촬영하여 GCS(Ground Control System)에 중계한다. GCS 에서는 재난 현장의 정보를 파악함과 동시에 헬리콥터의 현재 상태를 확인하고, 카메라에 대한 조작을 할 수 있게 한다. 이런 사항들을 만족하기 위해 헬리콥터에 비행 제어 컴퓨터와 멀티미디어

통신 컴퓨터를 설치하여 무인 헬리콥터를 제어하는 방법에 대한 연구가 진행되어 왔다.

HeliScope OFP 는 무인 헬리콥터의 비행 제어를 위해 개발된 내장형 소프트웨어로서, 헬리콥터에 장착된 비행 제어 컴퓨터에서 동작한다. OFP 는 실시간 운영체제에서 동작하는 제어 소프트웨어이며, 헬리콥터에 부착된 GPS/INS, AHRS 등과 같은 센서들을 통하여 실시간으로 변화하는 헬리콥터의 비행 상태를 파악하고, 안정된 비행을 위해 다음 움직임에 대한 제어를 위한 연산을 수행하는 역할을 한다.

2.2 HeliScope OFP 를 위한 검증 절차

HeliScope OFP 를 대상으로 검증을 수행하기 위해, OFP 의 특징을 분석하고 이에 적합한 검증 절차를 수립하는 연구[2]가 진행되었다. 이 연구에서는 IEEE 1012-2004 소프트웨어 검증과 확인 절차 표준[3]과 DO-178B 항공 소프트웨어 개발 지침[4]을 참고하여 소프트웨어 역공학 과정과 정적 코드 분석 과정, 정형 검증 과정, 소프트웨어 테스팅 과정을 포함하는 [그림 1]과 같은 절차를 제시하였다. 본 논문에서는 소프트웨어 역공학을 통해 검증을 위한 정보를 보완하고, 이를 가지고 여러 검증 기법들 중 소프트웨어 테스팅을 적용한 내용을 소개한다.



[그림 1] HeliScope OFP 를 위한 V&V 절차

2.2.1 소프트웨어 역공학

소프트웨어 역공학은 대상 소프트웨어의 구성 요소와 요소들 간의 관계를 파악하고, 이를 가지고 고수준의 추상화 표현을 사용하여 소프트웨어의 구조와 사양 등을 분석하는 방법이다 [5]. 소프트웨어 역공학은 일반적으로 소프트웨어의 유지보수와 재개발, 재구조화 등을 위한 목적으로 이용된다.

소프트웨어 검증 활동을 수행하기 위해서는 소프트웨어의 요구사항 및 설계에 대한 정보들이 필요한데, HeliScope OFP 는 이미 개발이 완료된 소프트웨어로 요구사항과 설계 정보가 검증을 수행하는데 충분치 않을 가능성이 있다. 따라서 기존 정보를 보완할 필요성이 있으며, 이를 위한 방법으로 소프트웨어 역공학을 이용하였다. HeliScope OFP 에 소프트웨어 역공학 기술을 적용하여 기존의 문서들과 소스 코드를 분석하고, 요구사항과 설계 정보를 보완하여 검증 활동을 위한 입력 자료로 활용하였다. 소프트웨어 역공학 과정과 이를 통해 복원된 자료들에 대해서는 3 장에서 자세히 소개한다.

2.2.2 소프트웨어 테스팅

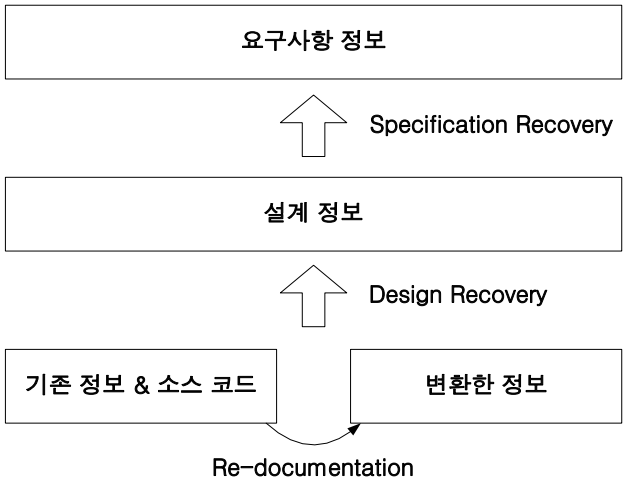
소프트웨어 테스팅은 대표적으로 사용되는 검증 활동으로서 소프트웨어의 동작을 확인할 수 있는 테스트 케이스를 작성하고, 소프트웨어를 실제로 실행시켜 기능과 동작의 정확성을 보장하는 검증 방법[6]이다. 소프트웨어 테스팅은 테스트 케이스의 작성 방법에 따라 크게 기능적 테스트와 구조적 테스트로 분류할 수 있다. 기능적 테스트는 소프트웨어의 요구사항에 기반하여 테스트 케이스를 작성하며, 구조적 테스트는 소프트웨어의 구조에 기반하여 테스트 케이스를 작성한다. 기능적 테스트는 소프트웨어의 요구사항에 기반해서 테스트 케이스를 작성하므로 개발된 소프트웨어가 요구사항에 부합하는지를 확인할 수 있으나, 소프트웨어의 일부분만을 테스트할 가능성이 있다. 구조적 테스트는 소프트웨어의 구조에 기반하여 테스트 케이스를 작성하기 때문에 잠재해있는 결점을 발견할 기회가 크지만, 소프트웨어 개발의 목적인 요구사항 만족 여부를 확인하기가 어렵다는 단점이 있다. 따라서 이 두 시험은 상호 보완적인 관계이며, 양쪽 모두의 접근법을 사용하는 것이 바람직하다. 본 논문에서는 구조적 테스트와 기능적 테스트 양쪽의 접근법을 모두 사용하여, 각각 유닛 레벨과 시스템 레벨에서의 테스트를 계획하였다. 각 테스트와 작성한 테스트 케이스에 대해서는 4 장에서 더 자세히 다룬다.

3. 소프트웨어 역공학 적용

검증을 위한 소프트웨어 역공학 과정은 먼저 기존 자료들을 수집하여 분석하는 단계부터 시작된다. 기존의 설계와 요구사항 문서들과 소스 코드를 수집하여 수집한 정보들이 검증 활동을 진행하기 충분한지를 확인한다. 소프트웨어 역공학 적용의

필요성이 확인되면 수집한 자료들로부터 소프트웨어 구조에 대한 정보를 추출하고, 이를 바탕으로 설계와 요구사항 정보를 분석한다.

소프트웨어 역공학 활동의 세부적인 절차는 다음과 같이 이루어진다[7]. 먼저 수집한 정보들을 분석하고, 이를 좀 더 이해하기 쉬운 형태로 변환하는 Re-documentation 과정을 거친다. 이 과정을 통한 산출물들은 소프트웨어의 구조 및 동작에 대한 기본적인 정보를 담고 있으며, 소프트웨어의 설계 정보를 분석하기 위한 기본 자료가 된다. 다음 과정은 이들 정보를 가지고 설계 정보를 복원하는 Design Recovery 과정이다. 이 과정에서는 소프트웨어의 구조와 각 요소들의 기능과 동작을 확인하고, 소프트웨어가 사용되는 영역에 대한 지식 등을 참고하여 설계 정보를 복원한다. 마지막 Specification Recovery 과정에서는 소프트웨어의 동작과 복원된 설계 정보를 확인하고, 개발자나 연구 참여자 등 전문가의 도움을 얻어 요구사항에 대한 정보를 복원한다. 이렇게 복원된 정보들을 문서화하여 검증을 수행하기 위한 입력 자료로서 사용하게 된다. [그림 2]는 소프트웨어 역공학 활동의 전체적인 흐름을 나타낸 것이다. 각 세부 활동을 적용한 내용과 산출물에 대해서는 아래에서 더 자세히 설명한다.



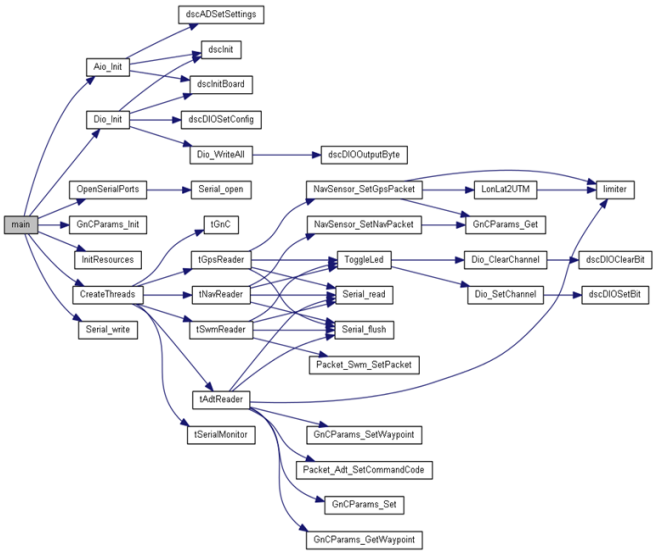
[그림 2] 소프트웨어 역공학 과정

3.1 Re-documentation

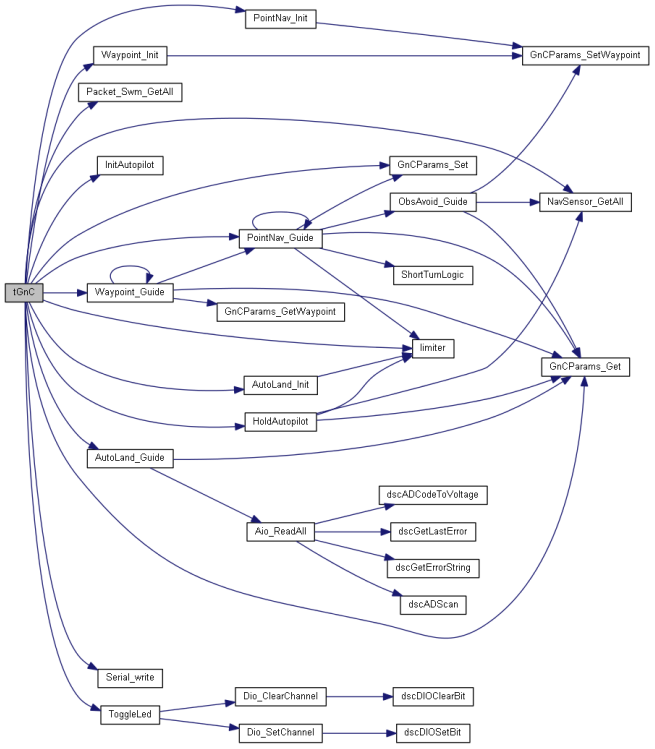
Re-documentation 은 한 표현 형태를 분석하여 동일한 의미를 지니는 다른 형태의 표현으로 변환하는 과정이다. 특정 정보를 좀 더 이해하기 쉬운 형태로 바꾸어 표현함으로써 소프트웨어의 구조 및 동작의 분석을 수행하기가 용이해진다. Re-documentation 은 소스 코드, 설계 정보, 요구사항 정보 등 모든 수준의 정보들을 대상으로 수행할 수 있으나 일반적으로 소스 코드 분석하여 이해하기 쉬운 형태로의 변환에 많이 사용된다. 소스 코드를 분석한 형태로는 대표적으로 Call Graph 나 CFG(Control Flow Graph) 등이 있다. 본 논문에서는 C/C++/JAVA 소스 코드 자동화 분석 도구인 Doxygen[8]을 사용하여 HeliScope OFP 의 소스 코드의 정보를 추출하였다. 이 도구를 통해 변수들과 함수들의 정보를 추출하고 이를 분석하여 Data Description 목록을 작성하였으며, 함수들의 호출 관계인 Call Graph 를 추출하였다. Data Description 을 통하여 각 변수들이 어떤 목적으로 사용되고, 어떤 타입으로 선언되어 있는지를 확인할 수 있다. 또한 Call Graph 를 통해 함수들의 호출 관계를 다이어그램 형태로 표현함으로써 OFP 의 구조와 동작의 흐름을 쉽게 확인할 수 있다. [표 1]은 작성한 Data Description 의 일부를 나타낸 것이며, [그림 3]과 [그림 4]는 도구를 통해 추출된 HeliScope OFP 의 일부 모듈의 Call Graph 이다.

[표 1] Data Description(일부)

변수 명	정의	Type
fdAdt	GCS 통신 시리얼 포트 접근 변수	int
fdDbg	Debug 모드 시리얼 포트 접근 변수	int
fdGps	GPS 센서 시리얼 포트 접근 변수	int
fdNav	Nav 센서 시리얼 포트 접근 변수	int
fdSwm	서보 상태 파악 및 컨트롤을 위한 시리얼 포트 접근 변수	int
semAdtReader	Adt 시리얼 포트로부터 데이터를 읽기 위한 Semaphore	struct sem_t
semAdtReader	Adt 시리얼 포트로부터 데이터를 읽기 위한 Semaphore	struct sem_t
semGpsReader	Gps 시리얼 포트로부터 데이터를 읽기 위한 Semaphore	struct sem_t
semNavReader	Nav 시리얼 포트로부터 데이터를 읽기 위한 Semaphore	struct sem_t
semSwmReader	Swm 시리얼 포트로부터 데이터를 읽기 위한 Semaphore	struct sem_t
…	…	…



[그림 3] HeliScope OFP 의 main 모듈의 Call Graph



[그림 4] HeliScope OFP 의 tGnC 모듈의 Call Graph

3.2 Design Recovery

Design Recovery 는 대상 소프트웨어를 분석하여 해당 영역의 지식, 외부 정보 등을 포함하여 구조와 동작을 파악할 수 있는 형태의 상위 설계 정보로 표현하는 방법으로, 기존 정보와 Re-documentation 을 통해 작성된 정보를 바탕으로 추상화된 고수준의 설계 모델로 표현한다. 본 논문에서는 기존의 정보들과, Re-documentation 과정을 통해 작성된 Data Description 과 Call Graph 를 가지고 HeliScope OFP 의 구조를 분석하여 Structure-chart 의 형태로 표현하고, 이를 기반으로 DFD(Data Flow Diagram)을 작성하였다. [그림 5]는 OFP 의 구조를 분석하여 표현한 Structure-chart 이다. 각 모듈과 데이터 저장 영역을 표시하고, 이들 간의 명령 및 데이터 이동을 분석하였다. [그림 6]는 Structure-chart 에서 분석한 내용을 통해 작성한 DFD 이다. DFD 를 통해 OFP 의 모듈들의 기능을 분석하여 프로세스로 표현하였고, 이들 간의 데이터 흐름을 분석하였다.

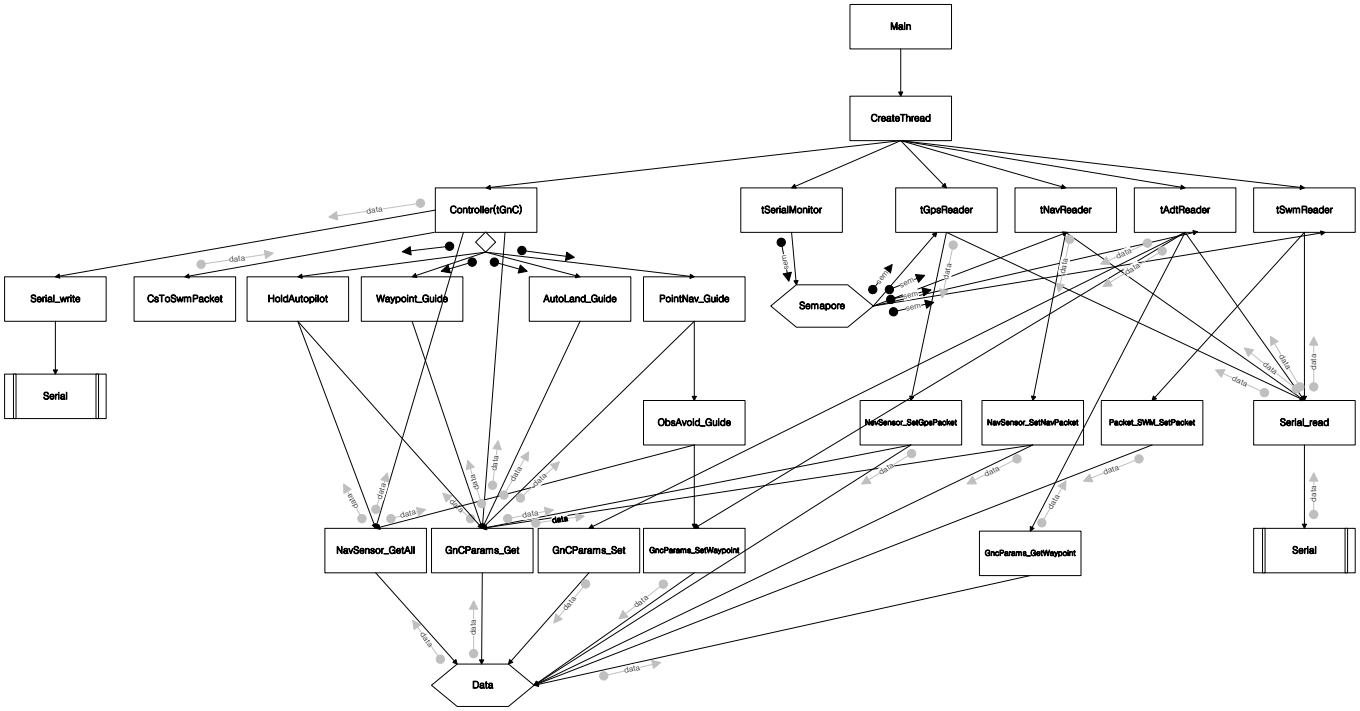
3.3 Specification Recovery

Specification Recovery 는 소프트웨어의 요구사항 정보를 분석하는 작업이다. 소프트웨어의 동작을 확인하고, 설계 정보를 바탕으로 소프트웨어의 기능을 분석하여 Use-Case Diagram 과 같은 요구사항 모델이나 자연어 형태로 정의한다. 본 논문에서는 DFD 에서 정의한 각 프로세스들의 기능을 분석하여 정의하였다. [표 2]는 결과의 일부를 나타낸다. 이를 통해 소프트웨어의 요구사항을 일부 분석할 수 있다.

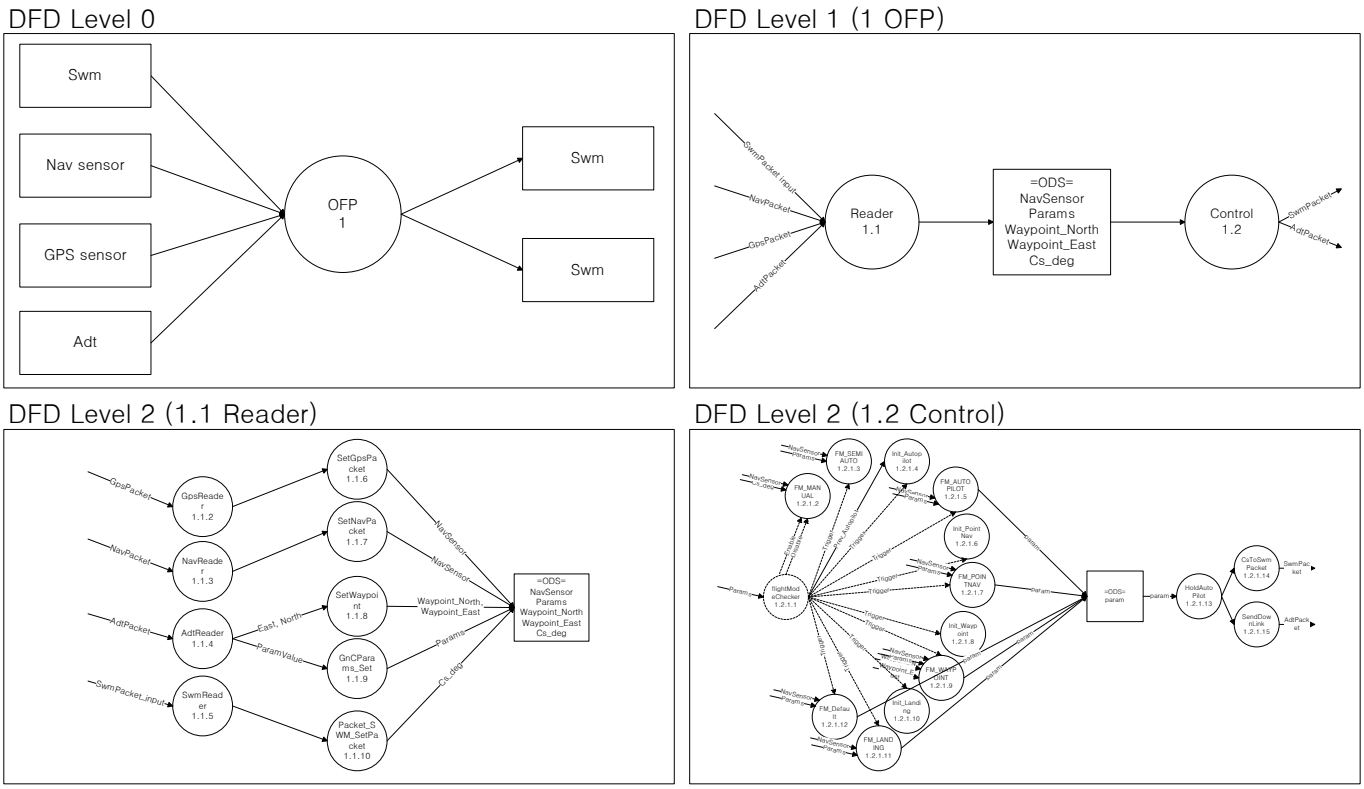
HeliScope OFP 를 대상으로 Design Recovery 과정을 통해 작성된 설계 정보를 이용하여 각 프로세스의 기능과 요구사항을 분석할 수 있었으나, 전체 시스템의 요구사항을 분석에 어려움이 있었다. 특히 기존 요구사항 정보에 기술되지 않은 기능의 경우 복원된 자료만으로는 분석하기에 어려운 점이 있었다. 이 과정에서는 부분적인 결과만을 작성하였으며, 정확한 요구사항 분석을 위해서는 OFP 의 개발자 혹은 전문가의 참여가 필수적인 것으로 판단된다.

[표 2] DFD 의 각 프로세스 기능 분석 결과(일부)

모듈 명	정의	입력 / 출력
SetGpsPacket	GPS 센서 값을 읽어 데이터 영역에 저장	GpsPacket / NavSensor
SetNavPacket	Nav 센서 값을 읽어 데이터 영역에 저장	NavPacket / NavSensor
SetWaypoint	GCS 로부터 전송 받은 Waypoint 데이터를 데이터 영역에 저장	AdtPacket / Waypoint_East, Waypoint_North
GnCParams_Set	paramValue 값(Adt 입력 중 GPS 제외)을 데이터 영역에 저장	paramValue / Params
Packet_SWM_SetPacket	Cs_deg 값(3 축 각도)을 데이터 영역에 저장	SwmPacket / Cs_deg
msgIdChecker	msgId 값 확인(GCS 의 커맨드 데이터의 타입 확인)	AdtPacket / msgId
…	…	…



[그림 5] 복원된 HeliScope OFP 의 Structure-chart



[그림 6] 복원된 HeliScope OFP 의 DFD(일부)

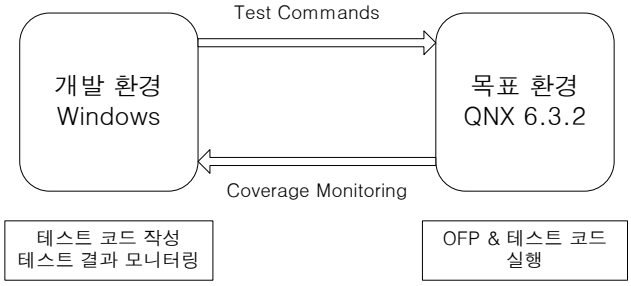
4. 소프트웨어 테스팅 적용

본 논문에서는 HeliScope OFP 를 대상으로 하는 소프트웨어 테스팅으로 구조적 테스트와 기능적 테스트 양쪽 모두를 계획하였다. 소프트웨어 테스팅은 소프트웨어의 실행을 수반하는 기법이므로

이를 위한 실행 환경을 구성하였으며, 구조적 테스트와 기능적 테스트를 위해 각각 HeliScope OFP 의 요구사항과 구조에 기반하여 테스트 케이스를 작성하였다. 구조적 테스트는 각 모듈을 대상으로 유닛 레벨의 테스트 케이스를 작성하였으며, 기능적 테스트는 두 가지 테스트 기법을 사용하여 접근하였다. 하나는 상태 전이 시스템을 기반으로 테스트 케이스를 작성하는 방법으로, 무인 헬리콥터의 비행 모드 전환에 관한 상태 전이 시스템을 작성하고 이를 기반으로 테스트 케이스를 작성하였다. 다른 하나는 데이터의 경계 값을 위주로 테스트 케이스를 작성하는 방법으로, HeliScope OFP 의 주요 데이터들을 선정하여 범위와 경계를 분석하고, 이를 가지고 테스트 케이스를 작성하였다.

4.1 구조적 테스트

구조적 테스트는 HeliScope OFP 의 각 모듈을 대상으로 하는 유닛 레벨의 테스트를 계획하였다. 테스트를 수행하기 위하여 먼저 HeliScope OFP 를 실행하기 위한 환경을 구성하였다. HeliScope OFP 는 실시간 내장형 소프트웨어로, 실시간 운영체제에서 동작하게 된다. 본 논문에서는 실시간 운영체제인 QNX Neutrino RTOS 6.3.2 [9] 버전을 사용하여 테스트를 위한 환경을 구성하였다. HeliScpe OFP 를 실행하기 위한 목표 환경과, 테스트 코드 작성을 위한 개발 환경을 [그림 7]과 같이 구성하였다. QNX 에서 제공하는 개발 도구의 기능을 사용하여 개발 환경과 테스트 환경간의 통신을 통해 시험을 수행하고 모니터링 하였다.



[그림 7] 구조적 테스트를 위한 환경 구성

구조적 테스트 케이스는 소프트웨어의 구조에 기반하여 작성하므로 테스트 케이스가 적절한지 판단할 기준이 필요하다. 일반적으로 구조적 테스트 케이스의 질을 확인하기 위한 방법으로 구조적 커버리지(Structural Coverage) 개념[10]을 사용한다. 구조적 커버리지를 측정하여 테스트 케이스가 적절한지를 파악하고 보완할 수 있다. 본 논문에서는 Statement 커버리지를 선택하여 측정하였으며, 여러 차례 테스트를 수행하며 테스트 케이스를 보완해 나갔다. 최종적으로 모두 54 개의 테스트 케이스를 작성하였으며 [표 3]은 그 일부이다. [표 4]는 최종적으로 측정한 각 모듈의 커버리지 결과를 나타낸 것이다. 테스트 수행 및 커버리지 측정 결과를 통해 대부분 모듈에 대한 테스트 케이스가 Statement 커버리지를 100% 만족함을 확인하였으며, 100%를 만족하지 않는 모듈의 경우 시리얼 포트 접근을 위한 코드와 같이 테스트 환경에서는 사용되지 않는 부분을 가지고 있거나, 개발 당시에 디버그 및 모니터링 등을 위하여 작성하였으나 실제로는 호출하지 않는 함수 등을 가지고 있음을 확인하였다.

[표 3] 구조적 테스트 케이스(일부)

ID	모듈 명	정의
TC_S01	tSerialMonitor	semAdt == true
TC_S02	tSerialMonitor	semAdt == false
TC_S03	tSerialMonitor	semNav == true
TC_S04	tSerialMonitor	semNav == false
TC_S05	tSerialMonitor	semSwm == true
TC_S06	tSerialMonitor	semSwm == false
TC_S07	tSerialMonitor	semGps == true
TC_S08	tSerialMonitor	semGps == false
TC_S09	tAdtReader	msgId == ADT_AP_ALT
TC_S10	tAdtReader	msgId == ADT_AP_U
TC_S11	tAdtReader	msgId == ADT_AP_V
…	…	…

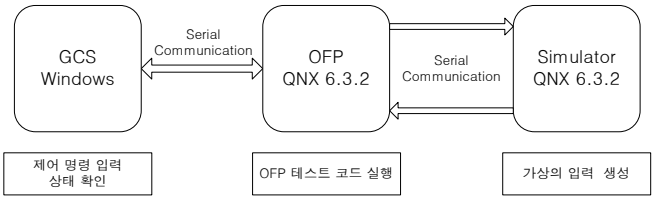
[표 4] 모듈 별 Statement 커버리지 측정 결과(일부)

모듈 명	# of branches	# of Test Case	Statement Coverage
tGnC	18	11	99.47%
GnCAutopilot	0	1	100%
GnCAutoLand	16	17	100%
GnCPointNav	9	4	100%
GnCWaypoint	3	3	93.33%
GnCObsAvoid	20	4	86.26%
…	…	…	…
전체	-	54	94.79%

4.2 기능적 테스트

기능적 테스트는 요구사항 정보에서 HeliScope OFP 의 비행 모드 전환에 관한 부분과, 주요 데이터들의 범위에 관한 부분을 테스트 하였다. 전체 시스템을 대상으로 하는 시스템 레벨의 테스트를 계획하였다. 기능적 테스트는 요구사항에 기반하여 테스트 케이스를 작성하므로, 테스트의 입력 값과 기대되는 결과 값을 작성하여 이를 확인해야 한다. 그러나 HeliScope OFP 는 실시간으로 동작하는

내장형 소프트웨어로, 센서 데이터를 입력 값으로 받으며 동작을 제어하기 위한 명령을 출력 값으로 내보낸다. 이들 데이터는 매우 짧은 간격을 가지고 반복적으로 입출력이 이루어지게 된다. 때문에 입력 값의 생성과 출력 값의 확인이 어렵다. 이런 문제를 해결하기 위해 현실 세계의 반응 값을 모델링한 시뮬레이터를 사용하여 [그림 8]과 같이 테스트 환경을 구성하였다. 시뮬레이터는 HeliScope OFP 개발 당시에 동작의 확인을 위해 함께 개발된 것이다. 이 시뮬레이터는 가상의 센서 값을 생성하여 OFP 에 전달하고, OFP 의 출력 값을 받아 다음 센서 값을 생성한다. 이와 함께 OFP 의 비행 모드 전환 및 수치 조작과, 동작 및 상태 확인을 위하여 GCS 를 추가하여 사용하였다. 본 논문에서는 환경 구성 후 테스트 코드를 작성하여 기능적 테스트를 수행하려 했으나, HeliScope OFP 와 시뮬레이터간의 동작에 이상이 있어서 환경 구성을 완료하지 못하였으며, 이 문제로 테스트 케이스를 작성하는 단계까지만 수행하였다.

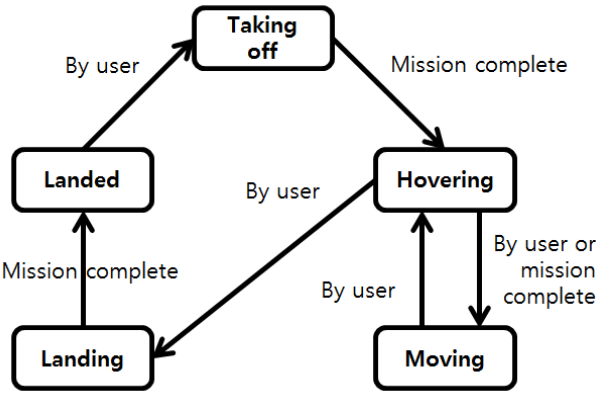


[그림 8] 기능적 테스트를 위한 환경 구성

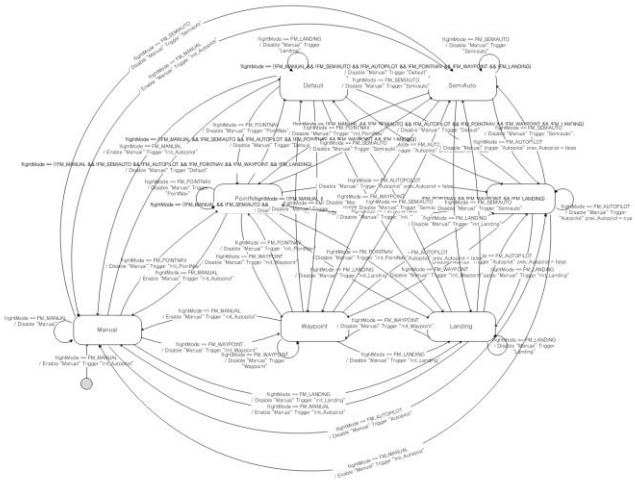
1) 상태 전이 시스템에 기반한 테스트

HeliScope OFP 는 무인 헬리콥터의 비행 상태를 제어하는 소프트웨어로, 수동 조작을 통한 비행 제어 외에도 자동으로 비행을 제어하는 모드가 존재한다. 자동 비행 제어 모드에는 이륙과 착륙, 호버링 모드와 웨이포인트 이동 등 여러 자동 비행 제어 모드가 존재하며, 지상으로부터 받은 명령에 따라 비행 모드의 전환이 이루어진다. 만약 비행 모드의 전환이 제 때 올바르게 이루어지지 않을 경우 오동작의 원인이 될 수 있으며 비행 상태에 큰 영향을 미칠 수 있다. 따라서 비행 모드의 전환이 제대로 이루어지는지에 대하여 검증이 필요하다.

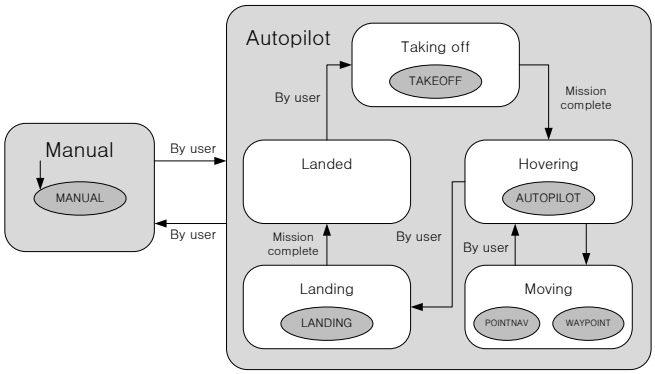
기존 요구사항 정보에서는 비행 모드 전환에 대한 내용을 상태 전이 시스템의 형태로 표현하고 있음을 확인하였다. 이 정보와, 3 장에서 소프트웨어 역공학을 통해 복원한 비행 모드 전환의 상태 전이 시스템을 가지고 새로운 상태 전이 시스템을 작성하였다. [그림 9]와 [그림 10]은 각각 비행 모드 전환에 대한 기존 요구사항 정보와 소프트웨어 역공학을 통해 작성된 정보를 나타내며, [그림 11]은 보완된 요구사항 정보이다.



[그림 9] 비행 모드 전환에 대한 기존 정보



[그림 10] 비행 모드 전환에 대한 복원된 정보



[그림 11] 비행 모드 전환에 대한 보완된 요구사항 정보

위의 과정을 통해 복원된 상태 전이 시스템을 기반으로 가능한 모든 전이를 만족할 수 있도록 하는 테스트 케이스를 작성하였다. 총 17 개의 테스트 케이스를 작성하였으며 이는 [표 5]와 [표 6]를 통해 확인할 수 있다. 작성된 테스트 케이스

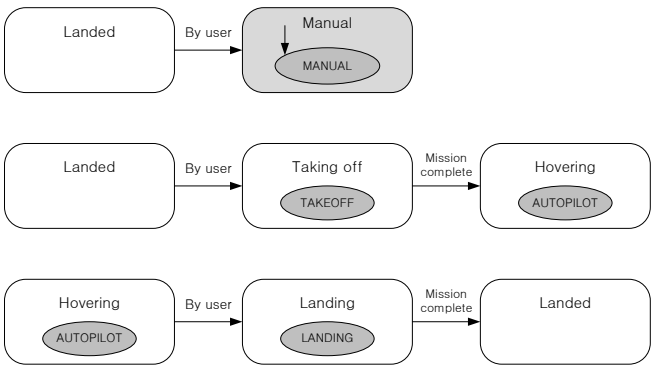
중 기존 정보에 정의되어 있는 상태 중 착륙 상태를 의미하는 Landed 를 포함하는 테스트 케이스는 제외하였다. 소프트웨어 역공학을 통해 복원된 상태 전이 시스템에는 착륙 상태를 의미하는 부분이 없었기 때문에 이들 테스트 케이스를 적용할 수 없다고 판단하였기 때문이다. [그림 12]은 제외한 테스트 케이스 들을 나타낸 것이다.

[표 5] 상태 전이 시스템에 기반한 테스트 케이스

ID	Prev State	Next State
01	FM_MANUAL	FM_TAKEOFF
02	FM_MANUAL	FM_LANDING
03	FM_MANUAL	FM_AUTOPILOT
04	FM_MANUAL	FM_POINTNAV
05	FM_MANUAL	FM_WAYPOINT
06	FM_TAKEOFF	FM_MANUAL
07	FM_LANDING	FM_MANUAL
08	FM_AUTOPILOT	FM_MANUAL
09	FM_POINTNAV	FM_MANUAL
10	FM_WAYPOINT	FM_MANUAL
11	FM_AUTOPILOT	FM_POINTNAV
12	FM_AUTOPILOT	FM_WAYPOINT
13	FM_AUTOPILOT	FM_WAYPOINT FM_POINTNAV
14	FM_POINTNAV	FM_AUTOPILOT
15	FM_WAYPOINT	FM_AUTOPILOT

[표 6] 상태 전이 시스템에 기반한 테스트 케이스

ID	Prev	Next	Next
16	FM_AUTOPILOT	FM_WAYPOINT	FM_AUTOPILOT
17	FM_AUTOPILOT	FM_POINTNAV	FM_AUTOPILOT

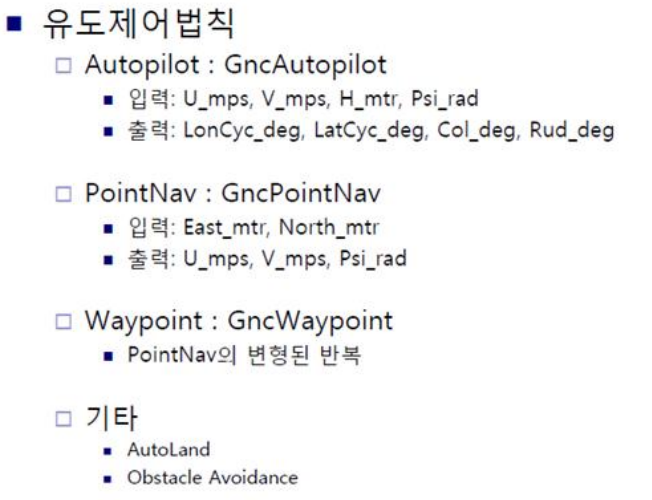


[그림 12] 제외된 테스트 케이스

2) 경계 값에 기반한 테스트

HeliScope OFP 는 비행 제어를 담당하기 때문에 많은 데이터를 입력으로 받아서 연산을 수행한다. 이들 데이터에 오류가 발생할 경우 비행 제어 연산의 결과에 오류가 생길 수 있다. 따라서 이들 데이터의 범위와 경계 값에 대한 테스트가 필요하다고 판단하였다.

기존 요구사항에서 제어 연산의 입력으로 사용되는 데이터와, 소프트웨어 역공학을 통해 복원된 DFD 상의 데이터를 가지고 테스트할 변수들을 선정하였다. [그림 12]와 [그림 13]은 각각 기존 요구사항 문서와 복원된 정보의 일부를 나타낸 것이다. 이를 통해 제어 연산에 영향을 미친다고 판단되는 19 개의 변수를 선정하여, 변수의 타입에 따라 가질 수 있는 값의 범위와 요구사항에 정의되어 있는 값의 범위를 가지고 테스트 케이스를 작성하였다. [표 7]와 같이 7 개의 경계 값 타입을 정의하고, 각 변수 별로 전부 혹은 부분적으로 적용하여 84 개의 테스트 케이스를 작성하였다. [표 8]은 각 변수의 타입에 따른 범위에 대한 테스트 케이스를, [표 9]은 요구사항에 정의된 값의 범위에 대한 케이스를 나타낸 것이다.



[그림 12] 제어 연산과 관련된 데이터 (기존)

DFD – Data Dictionary

Variable	Description	Type or Format
SwmPacket_input	State of Servo	Data from Sensor
NavPacket	Navigation Data	Data from Sensor
GpsPacket	GPS data(North, East, ...)	Data from Sensor
AdtPacket	Data from GCS	Data from Sensor
East	Location Data	Double
North	Location Data	Double
ParamValue	Command data from GCS	Double
NavSensor	Navigation Data from NavPacket	Double array / static
Waypoint_North	Location data for Waypoints	Double array / static
Waypoint_East	Location data for Waypoints	Double array / static
Params	Data for Operation	Double array / static
Cs_deg	Data to represent the state of Servo	Double array / static

[그림 13] 제어 연산과 관련된 데이터 (복원된 정보)

[표 7] 경계 값 테스트의 입력 타입

타입 명	정의
Max boundary	최대 경계 값
Max-	최대 경계 미만의 값
Max+	최대 경계 초과의 값
Min boundary	최소 경계 값
Min-	최소 경계 미만의 값
Min+	최소 경계 초과의 값
Nominal	경계에서 떨어진 범위 내 값

[표 8] 변수 타입에 따른 경계 값 테스트 케이스

ID	데이터 명	입력 값
TC_D00	msgId	0 (Min boundary)
TC_D01	msgId	-1 (Min-)
TC_D02	msgId	65535 (Max boundary)
TC_D03	msgId	65536 (Max+)
TC_D04	flightMode	0 (Min boundary)
TC_D05	flightMode	-1 (Min-)
TC_D06	flightMode	255 (Max boundary)
TC_D07	flightMode	256 (Max-)
TC_D08	Nav_Sensor[]	1.7e-308 (Min boundary)
TC_D09	Nav_Sensor[]	1.7e308 (Max boundary)

[표 9] 요구사항에 따른 경계 값 테스트 케이스(일부)

ID	데이터 명	입력 값
…	…	…
TC_D10	msgId	5 (Min-)
TC_D11	msgId	6 (Min boundary)
TC_D12	msgId	16 (Nominal)
TC_D13	msgId	17 (Out of boundary)
TC_D14	msgId	18 (Nominal)
TC_D15	msgId	19 (Nominal)
TC_D16	msgId	20 (Out of boundary)
TC_D17	msgId	73 (Max-)
TC_D18	msgId	80 (Max boundary)
TC_D19	msgId	81 (Max+)
TC_D20	flightMode	0 (Min-)
TC_D21	flightMode	1 (Min boundary)
TC_D22	flightMode	7 (Max boundary)
TC_D23	flightMode	8 (Max+)
TC_D24	north	-5001 (Min-)
TC_D25	north	-5000 (Min boundary)
…	…	…

5. 결론 및 향후 연구

항공 분야의 소프트웨어는 결함이 유발할 수 있는 사고의 위험성이 매우 높아 검증 활동이 매우 중요한 항목으로 요구된다. 이들 소프트웨어는 체계적인 검증 활동을 적용할 필요가 있다. 그러나 이미 개발이 완료된 소프트웨어의 경우, 검증 활동을 적용하기 위해서 개발 단계에서 작성된 요구사항과 설계 정보들이 보완되어야 할 필요성이 있을 수 있다. 본 논문에서는 HeliScope OFP 를 대상으로 이전 연구에서 구성하였던 검증 절차의 일부를 실제 적용하여, 소프트웨어 역공학 과정을 통해 설계와 요구사항 정보를 보완하였으며 소프트웨어 테스팅을 적용하여 유닛 레벨과 시스템 레벨에서 각각 구조적 테스트와 기능적 테스트를 계획하고 테스트 케이스를 작성하였다. 향후에는 기능적 테스트 수행을 위한 환경 구성을 완료하여 작성한 테스트 케이스를 바탕으로 기능적 테스트를 수행할 계획이다.

Acknowledgement

본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT 연구센터 지원사업의 연구결과로 수행되었으며, (NIPA-2011-C1090-1131-0008) 또한 2011 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (2010-0002566).

참고문헌

[1] Doo-Hyun Kim, Kodirov Nodir, Chun-Hyon Chang, Jung-Guk Kim, “HELISCOPE Project: Research Goal and Survey on Related Technologies”, International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, pp.112~118, 2009.

[2] 이종훈, 이동아, 유준범, “HELISCOPE Project 의 비행 운용 프로그램을 위한 검증 절차”, 한국소프트웨어공학 학술대회, pp.357~365, 2011.

[3] IEEE 1012-2004 Standard for Software Verification and Validation. 2005.

[4] SW Considerations in Airborne Systems and Equipment Certification RTCA/DO-178B. 1994.

[5] Chikofsky, Elliot J., James H. Cross II, “Reverse Engineering and Design Recovery: A Taxonomy”, IEEE Software, Vol.7, Issue.1, pp.13~17, 1990.

[6] Mauro Pezze, Michal Young, Software Testing and Analysis: Process, Principles, and techniques, Wiley, 2007.

[7] B.B. Agarwal, S.P. Tayal, Mahesh Gupta, Software Engineering & Testing: An Introduction, Jones & Bartlett Pub, 2008.

[8] Doxygen, http://www.stack.nl/~dimitri/doxygen/index.html

[9] QNX Neutrino RTOS, http://www.qnx.com

[10] H. Zhu, P. Hall, J. May, “Software Unit Test Coverage and Adequacy”, ACM Computing Surveys, Vol.29, pp.366~427, 1997.