Journal of the KIMST, Vol. 20, No. 5, pp. 688-699, 2017
ISSN 1598-9127
DOI http://dx.doi.org/10.9766/KIMST.2017.20.5.688

Research Paper 정보 · 통신 부문

모델기반 테스트 기법 및 무장통제장치 적용 사례

배정호[*,1)] · 장부철[1)] · 구봉주[1)]

[1)] 국방과학연구소 제1기술연구본부

A Model-based Test Approach and Case Study for Weapon Control System

Jung Ho Bae[*,1)] · Bucheol Jang[1)] · Bongjoo Koo[1)]

[1)] *The 1st Research and Development Institute, Agency for Defense Development, Korea*

(Received 13 March 2017 / Revised 2 June 2017 / Accepted 8 September 2017)

ABSTRACT

Model-based test, a well-known method of the black box tests, is consisted of the following four steps : model construction using requirement, test case generation from the model, execution of a SUT (software under test) and detection failures. Among models constructed in the first step, state-based models such as UML standard State Machine are commonly used to design event-based embedded systems (e.g., weapon control systems). To generate test cases from state-based models in the next step, coverage-based techniques such as state coverage and transition coverage are used. Round-trip path coverage technique using W-Method, one of coverage-based techniques, is known as more effective method than others. However it has a limitation of low failure observability because the W-Method technique terminates a testing process when arrivals meet states already visited and it is hard to decide the current state is completely same or not with the previous in the case like the GUI environment. In other words, there can exist unrevealed faults. Therefore, this study suggests a Extended W-Method. The Extended W-Method extends the round-trip path to a final state to improve failure observability. In this paper, we compare effectiveness and efficiency with requirement-item-based technique, W-Method and our Extended W-Method. The result shows that our technique can detect five and two more faults respectively and has the performance of 28 % and 42 % higher failure detection probability than the requirement-item-based and W-Method techniques, respectively.

Key Words : Extended W-Method(확장 W-Method), Model-based Testing(모델기반 테스팅), Weapon Control System(무장통제장치)

1. 서 론

무장통제장치는 대상을 발사하기 전 상태를 점검하

* Corresponding author, E-mail: jhbae@add.re.kr

고 정상 발사를 위한 절차를 진행하는 역할을 수행한다. 발사 대상마다 발사 절차 및 점검 요소가 다르므로 정해진 절차에 따라 점검을 수행하며 비정상 상황이 확인되면 발사를 중지해야 한다. 무장통제장치에 문제가 발생하여 비정상 상황임에도 발사되는 경우에는 막심한 피해가 발생하기 때문에 높은 신뢰성이 요구된다.

무장통제장치에 내장된 소프트웨어는 무기체계 소프트웨어로 방위사업청에서 발간한 "무기체계 소프트웨어 개발 및 관리 지침"에 따라 정적 시험 및 동적 시험을 실시한다. 정적시험은 코드를 실행하지 않고 코딩 규칙, 실시간 오류 등을 검출하는 시험이다. 동적시험은 실제로 소프트웨어를 실행하여 입력과 출력을 관찰하여 오류를 검출하는 시험으로, 코드의 실행률을 확인하는 화이트박스 테스트와 요구사항의 테스트 여부를 확인하는 블랙박스 테스트로 나뉜다. 화이트박스 테스트는 단위시험에 적합하며, 블랙박스 테스트는 통합 또는 시스템 테스트에 적합하다.

블랙박스 테스트 방법은 요구사항을 모두 만족할 수 있도록 경험 기반으로 테스터가 테스트 케이스를 작성하여 진행된다. 이러한 상황에서는 추상화 레벨이 높은 수준인 요구사항을 기준으로 테스트 케이스를 작성하기 때문에 테스터의 역량에 따라 테스트 케이스의 품질이 차이가 달라질 수 있다. 또한 개별 요구사항을 대상으로 테스트 케이스가 작성되기 때문에 기능들의 선/후행 관계를 고려한 테스트가 수행되는 것을 보장할 수 없다. 뿐만 아니라 각 테스트 케이스들을 표 형태로 나열하여 관리하는 경우에는 가독성 및 이해용이성이 떨어지므로 재사용이 힘들다[1].

더욱 체계적인 테스트를 위하여 다수의 모델 기반의 테스트 생성 연구가 진행되었다[2-4]. 모델 기반 테스트 케이스 생성은 요구사항을 기반으로 모델을 만들고 이를 이용하여 특정 조건을 만족하는 테스트 케이스를 생성하는 기법이다. 일반적으로 서술 형태로 기술되어있는 요구사항을 정형적이고 가독성과 유지보수성이 높은 모델을 이용하여 기술함으로써 고품질의 테스트 케이스를 적은 비용으로 생성할 수 있도록 도와준다.

무장통제장치와 같은 이벤트 기반 임베디드 시스템에서는 상태 기반 모델을 이용한 상태 기반 테스트(State-Based Testing, SBT)가 사용된다. 상태 기반 테스트는 모든 상태를 한번이상 방문하는 상태 커버리지, 모든 전이를 한번이상 실행하는 전이 커버리지, 기 방문한 상태가 아닌 상태까지 전이 시퀀스를 실행하는 라운드 트립(round-trip path) 커버리지 등의 기준을 만족하기 위한 방향으로 진행된다. 이중 라운드 트립 커버리지를 만족하는 테스트가 가장 성능이 우수하다고 알려져 있다[5-6]. 라운드 트립 커버리지를 만족하는 테스트 케이스 생성 방법은 W-Method[7]를 사용한다.

블랙박스 테스트는 시스템이 제공하는 출력 값만을 이용하여 정상판단을 하므로 오류의 발견가능성(observability)의 한계가 존재한다. W-Method는 기방문한 상태를 만나면 테스트를 중단하는 특징을 가지고 있다. 따라서 마지막 상태에서 오류가 발견되지 않는다면 오류가 없는 것으로 판단한다. 복잡한 임베디드 시스템, 특히 GUI기반 시스템은 모든 요소(attribute)를 외부로 출력하지 않기 때문에 이미 발생한 오류가 다른 기능을 수행하여야만 발현될 수도 있다.

본 논문에서는 오류의 발견 가능성 향상을 위한 확장 W-Method(Extended W-Method) 기법 제안한다. 무장통제장치의 행위를 표현하는 모델은 UML의 상태머신(State Machine)[8]을 사용하였고, 요구항목 기반 기법, W-Method와 테스트 결과를 비교하여 보여준다.

확장 W-Method의 효용성을 확인하기 위하여 현재 개발 중인 무장체계에서 사용하는 통합모의기를 대상으로 실험을 진행하였다. 통합모의기는 무장제어기, 다수의 무장연동기, 수직발사제어기 다수의 무장통제장치들간의 통신 메시지를 확인하기 위한 용도로 개발되었으며 인터페이스를 확인하기 위한 용도이므로 실제 장비의 모든 기능이 구현되어 있지 않다. 즉, 다수의 실제 기능적인 오류들이 존재한다.

본 논문의 구성은 다음과 같다. 2장에서는 연구 배경으로 기존의 테스트 방법과 UML 상태 머신, 모델기반 테스트 생성 기법을 설명하고, 3장에서는 구체적인 테스트 모델 생성과 테스트 케이스 생성 절차를 소개한다. 4장에서는 제안한 기법으로 무장통제장치를 테스트한 결과에 대하여 논의하고, 5장에서 결론 및 향후 연구를 기술한다.

2. 연구배경

2.1 요구항목 기반 블랙박스 테스트

무장통제장치와 같은 무기체계 소프트웨어는 방위사업청에서 발간한 "무기체계 소프트웨어 개발 및 관리 지침"과 이 지침의 적용을 지원하기 위한 "무기체

계 내장형 소프트웨어 획득 및 관리 실무 지침서"를 따라 개발되고 있다. 이 지침서에는 무기체계 소프트웨어를 요구사항분석, 구조 설계, 상세설계, 구현, 통합시험 등의 단계를 수행하도록 규정되어 있다.

이 지침서에 따르면 구현 및 통합시험 단계에서 동적시험을 하도록 명시되어 있다. 동적시험의 완료 기준은 화이트 박스 테스트에서는 코드 커버리지를 요구하며, 블랙박스 테스트에서는 요구사항의 항목들을 모두 테스트 하였는지를 확인하도록 되어 있다. 블랙박스 테스트는 요구사항 추적성 표를 작성하여 모든 요구사항들이 테스트 되었는지를 확인한다.

요구사항 표를 이용하여 완료 조건을 판단할 경우에는 세 가지 한계가 존재한다. 첫 번째는 구체성이다. 요구사항은 개발 단계의 가장 초기에 작성하는 것으로 실제 구현되었을 때는 하나의 요구사항이 반영이 되었는지 확인하는 구체적인 테스트 입력이 하나 이상이 될 수 있다. 이 중 하나의 입력 방법만 테스트 된다면 테스트 되지 않은 다른 입력 방법에 존재하는 오류를 발견하지 못할 수 있다.

두 번째는 요구사항 항목들 간의 연속성 확인의 어려움이다. 요구사항 추적성 표를 이용하여 확인하는 경우에는 개별 요구사항들은 확인할 수 있지만 요구사항들 간의 선/후 관계는 확인하기 어렵다. 예를 들어, '전원 켬'과 '전원 끔'에 대한 요구사항이 있다고 가정하자. 각각에 대한 테스트는 잘 된다고 하다라도, '전원 켬 → 전원 끔 → 전원 켬' 과 같은 테스트는 수행하였는지 확인할 수 없다.

세 번째는 재사용의 어려움이다. 시스템을 개발하는 중에는 새로운 기능의 추가, 기존 기능의 수정 등 많은 변경이 발생한다. 이러한 변경이 발생할 때 마다 관련된 테스트 케이스를 찾아서 수정하여야 한다. 이러한 작업은 테스트의 수가 많아질수록 많은 노력이 소요된다. 또한 무장통제장치의 특성상 발사대상이 유사한 경우가 많은데 기존의 문서 형태로 관리되는 테스트 케이스는 가독성 및 이해용이성이 떨어지므로 대상이 변경되면 새로 테스트 케이스를 작성하는 경우가 대부분이다.

본 연구에서는 이러한 요구항목 기반 테스트 기법의 한계를 극복하기 위하여 모델 기반의 블랙박스 테스트 기법을 적용한 사례를 소개한다.

2.2 UML 상태 머신

본 연구에서는 무장통제장치의 행위를 나타내는 테스트 모델로서 UML 상태 머신(State Machine)을 채택하였다. UML 상태 머신의 정의는 다음과 같다.

정의 1. 컴포넌트의 동적 행위는 상태 머신 $SM = (S, s_0, S_\psi, I, O, \sigma)$로 표현된다.

- S : 공집합이 아닌 상태의 집합.
- s_0 : 초기 상태로 S의 원소.
- S_ψ : 최종상태로 S의 부분집합.
- I : 입력 이벤트의 집합.
- O : 출력 이벤트의 집합.
- $\sigma : S \times I \rightarrow O \times S$. 두 상태 간의 전이 함수. O는 생략 가능.

Fig. 1은 상태 머신에 대한 간단한 예를 보여준다. 이 예의 상태 머신에는 2개의 상태와 하나의 전이가 있다. 상태 집합 $S = \{IDLE, PWR_ON\}$이고, 전이 집합 $\sigma = \{\{IDLE, PwrOn, wating(20), PWR\_ON\}\}$ 이다. 상태는 모서리가 둥근 사각형으로 표현하며, 전이는 상태들 간의 화살표로 표현한다. 전이의 화살표 위에는 (입력 / 출력) 의 형태로 전이가 천이되기 위한 입력과 그에 따른 출력을 표현하며, 입력은 하나 이상이 될 수 있고 출력은 생략할 수 있다. 초기 상태는 작은 점이 가리키는 상태 $s_0 = IDLE$ 이고, 최종 상태는 이중 원으로 향하는 상태 집합 $S_\psi = \{ PWR\_ON \}$ 이다. 이 컴포넌트가 처리할 수 있는 입력은 $I = \{ PwrOn \}$ 이고 수행할 수 있는 출력은 $O = \{ wating(20) \}$이다. *IDLE* 상태에서 받아들일 수 있는 입력은 *PwrOn*이고, *PwrOn*이 입력되면 20초간 대기(*wating*(20))한 후에 *PWR_ON* 상태가 되고 종료된다.

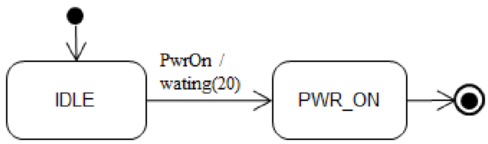

Fig. 1. An example of state machine

2.3 W-Method

상태 기반 테스트(State-Based Testing : SBT)는 상태 머신 다이어그램으로부터 테스트 케이스들을 도출하고, 각 테스트 케이스를 수행함으로써, 소프트웨어의 오류를 검출하는 것이다. SBT의 대표적인 테스트 케이스 생성 기준에는 모든 전이 커버리지(AT), 모든 전이쌍 커버리지(ATP), 모든 라운드 트립 커버리지(RTP)

등이 있다[9].

W-Method는 RTP를 만족하는 테스트케이스를 생성한다. RTP는 모든 라운드 트립 패스(round-trip path)를 수행하는 것으로, 라운드 트립 패스는 시작 상태와 끝 상태가 동일하며, 시작/끝 상태를 제외하고는 중복되는 상태가 없는 패스를 의미한다. 이 기준을 만족하는 테스트 케이스 세트를 생성하는 방법으로는 깊이 우선(depth-first) 혹은 너비 우선(breadth-first) 탐색 알고리즘을 사용하여 전이 트리(transition tree)를 만드는 것이다. 이때 초기상태에서 기 방문한 상태를 만날 때까지 테스트를 수행함으로써 RTP를 만족하는 테스트 케이스를 생성할 수 있다. 이 전이 트리에서 각 패스가 테스트 케이스가 된다.

비용은 적게 들되, 오류 검출은 많이 하는 테스트케이스 세트가 좋은 것이라 할 수 있는데, 준비시간, 실행시간, 테스트 크기, 뮤테이션 점수를 기준으로 AT, RTP, ATP 등을 평가했을 때, RTP가 가장 우수하다고 알려져 있다[10].

무장통제시스템은 최종임무를 끝까지 수행 가능한지를 확인하는 것이 중요한데, 기존 기준들은 최종상태까지 진행되지 않을 수 있으므로 이를 확인하기 힘들 수 있다. 본 연구에서는 이러한 한계를 극복하기 위하여 W-Method를 확장하는 기법을 제안한다. 본 연구에서 제안하는 방법은 W-Method를 기반으로 전이트리를 만들, 각 패스에서 기 방문 상태에서 끝나는 것이 아니라, 발사절차 소프트웨어의 최종상태까지의 패스를 추가하는 방법이다.

2.4 테스트 케이스 평가 방법

본 연구에서는 생성한 테스트 케이스를 비교하기 위하여 커버리지 분석, 테스트 케이스 실패율, 발견한 결함 수, 테스트 효율성의 4가지 평가 방법을 사용하였다.

커버리지는 테스트의 종료 기준으로 주로 사용된다. 화이트박스 테스트에서는 코드의 실행율을 기준으로 측정하며 구문, 분기 커버리지 등이 있다. 블랙박스 테스트에서는 요구사항의 테스트 여부를 기준으로 측정하며 모델 기반이 아닐 경우에는 요구사항 항목, 상태 기반 모델에서는 상태, 전이, 전이쌍, 라운드 트립 커버리지 등을 측정한다. 본 연구에서는 블랙박스 테스트를 수행하므로 요구사항 항목, 전이, 라운드 트립 커버리지를 측정하여 비교한다.

본 연구에서는 결함은 3단계로 구분하였다. 1단계는 색깔이 틀리거나 오타 등의 경미한(trivial) 오류이다. 2단계는 오류가 있으나 절차에 지장을 주지 않거나, 우회방법이 존재하는 보통(minor) 오류이다. 3단계는 정상절차가 진행될 수 없는 치명적인(critical) 오류이다.

테스트 케이스의 전반적인 효용성을 확인하기 위하여 테스트 케이스 실패율을 비교하였다. 테스트 케이스 실패율은 전체 테스트 케이스 중 실패한 테스트 케이스의 비율을 나타낸다. 실패한 테스트 케이스가 많을수록 결함을 발견할 확률이 높아진다. 따라서 테스트 케이스 실패율이 높을수록 테스트 케이스가 좋을 수 있다. 하지만 동일한 결함에 의해서 다수의 테스트 케이스가 실패할 수 있음을 주의해야 한다.

테스트 케이스의 효과성을 확인하기 위하여 발견한 결함의 수를 분석하였다. 발견한 결함 수는 실패한 테스트의 근본 원인의 수이다. 하나의 테스트 케이스가 다양한 결함에 의해서 실패할 수도 있고, 하나의 결함에 의해서 다수의 테스트 케이스가 실패할 수도 있다. 실패 원인들을 분석하여 결함을 도출하였다.

본 논문에서 제안하는 확장 W-Method는 기존의 W-Method로 생성한 테스트 케이스 중 일부에 입력을 추가하는 방법이다. 따라서 기존 기법에 비해서 테스트 케이스의 수는 동일하고, 전체 입력의 길이는 늘어난다. 테스트의 입력을 늘리는 것이 오류를 발견하는 것이 더욱 많은 오류를 발견할 수는 있지만 들이는 노력에 비해 효율적이지 못할 수 있다.

이러한 의문을 해소하기 위하여 테스트 케이스 당 오류발견 효율성과 더불어 테스트 입력당 오류발견 효율성을 함께 분석하였다. 테스트의 효율성을 평가하는 일반적인 방법은 *테스트 케이스 효율성*으로 [발견한 오류의 수 / 테스트 케이스 수] 로 측정한다. 테스트의 준비 혹은 정리의 노력이 많이 소요될 때는 테스트 케이스 효율성이 효과적인 평가 방법이 될 수 있다. 하지만 이 방법은 테스트 케이스마다 그 길이가 다르므로 공정한 비교 방법이 되지 않을 수 있다. 따라서 추가적으로 [발견한 오류의 수 / 테스트 입력 수]로 계산하는 *테스트 입력 효율성*을 측정하였다.

3. 확장 W-Method를 이용한 모델 기반 테스트 케이스 생성

이 장에서는 무장통제장치를 테스트하기 위하여 테스트 모델을 생성하고 이를 이용하여 테스트 케이스

를 생성하는 절차를 예를 활용하여 설명한다.

3.1 요구사항 정의

이 장에서는 장비 상태 관리, 발사 리스트 관리, 발사제어 등 무장통제장치의 다양한 기능 중 핵심 기능인 발사제어 컴포넌트를 대상으로 예를 들어 설명한다. 간소화된 발사제어 요구사항은 Table 1의 다섯 가지이다.

Table 1. An example of fire control requirement

#	Title	Description
1	Power Control	Power on and off of target weapon
2	Countdown Control	Countdown start and liftoff detection
3	Cooldown	Emergency cooldown sequence execution
4	Ready Monitoring	Ready status monitoring
5	Countdown Monitoring	Countdown status monitoring

3.2 테스트 모델 생성

상태 모델 기반 테스트를 위하여 Table 1의 요구사항을 바탕으로 Fig. 2와 같이 상태 머신 모델을 작성하였다. 이 예의 상태머신은 6개의 입력 *I* = {*Pwr On, Pwr Off, Start Cnt Dn, Fire Inhibit, Fired, Failed*}를 받을 수 있으며 그에 따라 5개의 상태 *S* = {*IDLE, PWR ON, COUNT DOWN, FIRED, ABORTED*}로 구성된다. 초기 상태인 *IDLE* 상태에서 *Pwr On* → *Start Cnt Dn* → *Fired* 를 입력함으로써 정상발사를 수행할 수 있으며 이 과정에서 *Pwr Off*, *Fire Inhibit*, *Failed*의 입력을 통해 상태를 제어할 수 있다.

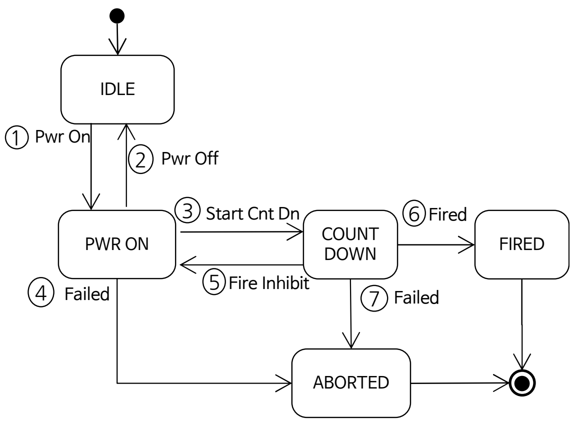

Fig. 2. An example of simplified fire control component test model

3.3 테스트 케이스 생성

이 절에서는 1) 요구항목 기반 테스트 케이스 생성 기법, 2) W-Method를 이용한 테스트 케이스 생성 기법, 3) 본 연구에서 제안하는 확장 W-Method를 이용한 테스트 케이스 생성 기법을 설명한다.

3.3.1 요구항목 기반 테스트 케이스 생성

요구항목 기반 테스트 케이스 생성은 모델과 관련없이 요구사항만을 이용하여 테스트 케이스를 생성하는 테스트 케이스 생성 기법이다. 요구항목 기반 테스트 케이스의 생성 기준을 만족하기 위해서는 모든 요구사항을 테스트할 수 있는지를 확인하여야 한다. Table 1의 발사절차 요구사항을 만족하기 위하여 생성한 요구항목 기반 테스트 케이스는 총 3개로 Table 2에서 보여준다.

Table 2. An example of req. item-based test cases

#	Name	Test Case	Covered Req.
1	Power Control	①→②	Req. 1, Req. 4
2	Fire Success	①→③→⑥	Req. 2, Req. 5
3	Fire Failed	①→③→⑦	Req. 2, Req. 3, Req. 5

3.3.2 W-Method 테스트 케이스 생성

W-Method를 이용한 테스트 케이스 생성은 트리 생성과 테스트 케이스 생성의 두 단계로 진행된다. 트리 생성 단계에서는 상태 머신의 초기 상태에서 너비 또는 깊이 우선 탐색을 하면서 트리를 구성한다. 이때, 기 방문한 상태를 만나는 라운드 트립 패스가 되거나 최종상태를 만나면 해당 상태를 단말노드로 설정한다.

Figure 3은 W-Method를 사용하기 위하여 Fig. 2의 상태 머신으로부터 생성한 트리를 보여준다. 이 예에서 ①→②와 ③→⑤가 시작상태와 끝상태가 각각

*IDLE*과 *PWR ON*으로 동일한 라운드 트립 패스이다.

생성한 트리에서 초기 상태에서 각 단말 노드까지의 패스가 W-Method를 이용한 테스트 케이스가 된다. Table 3은 W-Method를 이용하여 생성한 5개의 테스트 케이스를 보여준다.

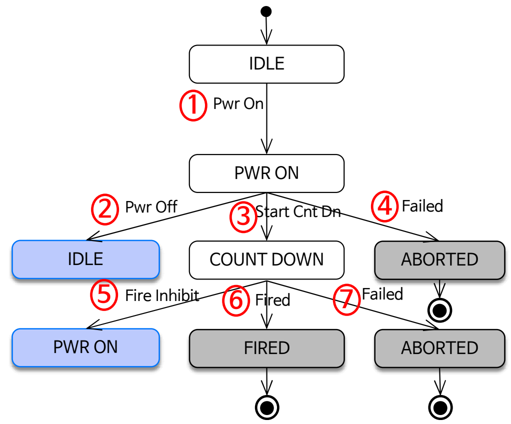

Fig. 3. An example of test case generation using W-Method

Table 3. An example of W-Method test cases

#	Name	Test Case	Covered Req.
1	Power Control	①→②	Req. 1, Req. 4
2	Fire Inhibit	①→③→⑤	Req. 2, Req. 5
3	Fire Success	①→③→⑥	Req. 2, Req. 5
4	Fire Failed	①→③→⑦	Req. 2, Req. 3, Req. 5
5	Power On Failed	①→④	Req. 1, Req. 4

3.3.3 확장 W-Method 테스트 케이스 생성

프로그램의 버그는 기능의 후처리 과정에 존재할 수 있다. 이러한 버그는 해당 기능을 실행한 순간에는 발현되지 않지만 이후의 다른 기능에 영향을 미친다. 해당 기능만을 실행하여 테스트 하였을 때는 오류의 관찰 가능성(observability)이 낮을 수 있다. 예를 들어, Fig. 2에서 초기 상태인 *IDLE*과 *IDLE*에서 *Pwr On* → *Pwr Off*를 실행한 후인 *IDLE*이 동일함을 확인하기 위해서는 추가적으로 다른 기능들을 실행하여 확인하여야 한다.

```
function ExtendedW-Method
input: a state machine SM
input: a selected final state s'
output: a test suite TS
begin
  TS = {}
  Tree T = {init node n_0, a set of nodes N = {}, a set of edges E = {}}
  s = s_0 ∈ SM

  for each nonterminal leaf node n ∈ T
  begin
    for each outbound transition t of the state ss corresponding to n
    begin
      st = a target state of t
      n' = a node corresponding to st
      N = N ∪ {n'}
      e = an edge {n × I of t → O of t × n'}
      E = E ∪ {e}
      if st is already represented by another node then
        T = T ∪ shortest path from st to s'
        st = s'
      end if
      if st is a final state then
        set n' is terminal node
      end if
    end for
  end for

  for each event sequence TC from n_0 ∈ T to nonterminal leaf node n ∈ T
  begin
    TS = TS ∪ TC
  end for
end function
```

Fig. 4. An algorithm for Extended W-Method

본 논문에서는 오류의 발견가능성을 높이기 위하여 라운드 트립 패스에서 테스트를 끝내는게 아니라 그 이후에 최종 상태까지 실행하는 확장 W-Method를 제

안한다. Fig. 4는 확장 W-Method의 알고리즘을 보여준다. 입력은 테스트 대상의 상태 머신과 선택된 최종상태이고 출력은 테스트 케이스 집합이다. 9번-26번 줄은 테스트 트리를 구성하는 로직이고, 27번-32번 줄은 테스트 트리를 이용하여 테스트 케이스를 생성하는 로직이다. W-Method와의 차이점은 기 방문한 상태를 만났을 때 종료하는 것이 아닌 입력받은 최종 상태까지 도달하는 트리를 생성하는 것으로 알고리즘의 18번-21번 줄이 이에 해당한다.

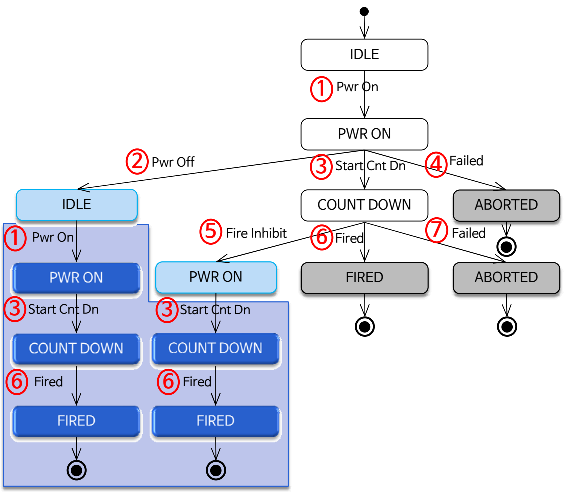

Fig. 5. An example of test case generation using Extended W-Method

Table 4. An example of Extended W-Method test cases

#	Name	Test Case	Extended
1	Power Control	①→② →①→③→⑥	✓
2	Fire Inhibit	①→③→⑤ →③→⑥	✓
3	Fire Success	①→③→⑥	
4	Fire Failed	①→③→⑦	
5	Power On Failed	①→④	

Figure 5는 Fig. 3의 트리의 라운드 트립 패스를 본 논문에서 제안하는 방법으로 확장한 테스트 트리를 보여준다. 이때, 최종 상태는 정상발사를 선택하였다.

이 테스트 트리에 의해 생성한 테스트 케이스는 Table 4와 같이 Table 3과 비교하여 1번과 2번의 테스트 케이스가 확장되었다.

3.3.4 테스트 케이스 평가

이 절에서는 2.4절에 기술된 테스트 케이스 평가방법으로 요구항목 기반, W-Method, Extended W-Method로 생성한 테스트 케이스를 비교한다. Table 5는 생성된 테스트 케이스들의 기술적인 값들을 보여준다. 요구항목 기반 테스트 케이스의 수 보다 W-Method와 Extended W-Method로 생성한 테스트 케이스의 수가 더 많다.

Table 5. Descriptive values of generated test cases

	Req. Item	W-Method	Extended W-Method
# of TCs	3	5	5
Total Input Length	8	13	18
Avg. TC Length	2.7	2.6	3.6

Extended W-Method는 W-Method로 생성한 테스트 케이스를 확장한 것이므로 테스트 케이스의 수는 동일하지만 평균 테스트 케이스의 길이가 더 길다. 우리는 이러한 추가적인 테스트가 버그의 발견 확률을 높여줄 것이라 기대한다. 그리고 그에따라 전체적인 테스트 효과성과 효율성이 높아질 것이라 기대한다.

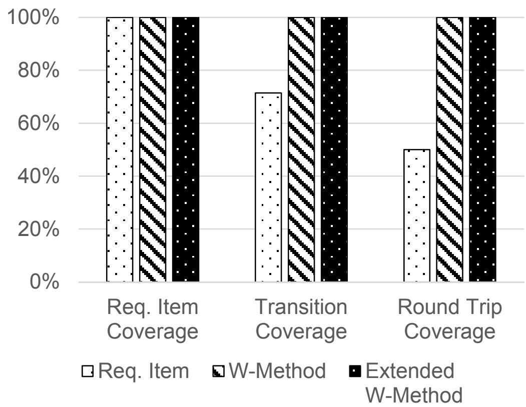

Fig. 6. Coverage evaluation result on the example

Figure 6은 각 기법으로 생성한 테스트 케이스의 커버리지를 평가한 결과를 보여준다. 요구항목 기반으로 생성한 테스트 케이스는 7개의 전이 중 ⑤, ④의 2개의 전이를 커버하지 못했으며, 2개의 라운드 트립 중 ③→⑤를 커버하지 못하였다.

이 장에서 사용한 예제는 설명을 위한 임의의 모델을 사용했으며 실제로 테스트할 대상이 없으므로 2.4절에 기술된 나머지 평가 방법은 적용할 수 없다. 테스트 케이스 평가 방법 중 테스트 케이스 실패율, 발견한 결함 수, 테스트 효율성은 다음 사례 연구 장에서 확인한다.

4. 사례 연구

이 장에서는 테스트 대상인 무장통제장비를 간략히 설명하고 테스트 모델과 각 기법으로 생성한 테스트 케이스의 평가 결과를 기술한다.

4.1 테스트 대상

본 연구에서 테스트 대상(SUT, subject under test)으로 사용한 프로그램은 무장통제장치를 구성하는 다양한 장비들의 통신을 점검하기 위하여 사용하는 무장통제장치 통합시뮬레이터이다. 무장통제장치는 다수의 장비들로 구성된다. 사용자 인터페이스인 통제콘솔, 발사제어 및 상태 확인을 수행하는 주제어부, 비행체와 연동을 담당하는 연동기, 비행체를 모의하는 비행체 시뮬레이터 등으로 구성된다. 이들 간은 이더넷/이산신호/시리얼통신 등 다양한 방법으로 복잡한 통신이 이루어진다. 이들간의 통신 메시지는 “무기체계 내장형 소프트웨어 획득 및 관리 실무 지침서”에 따라서 ICD(Interface Control Document)를 이용하여 관리된다. 개발이 진행됨에 따라 기능이 추가/변경되고, 이에 따라 ICD가 변경되는데 이 과정에서 오류가 만들어지기 쉽다. 통합 시뮬레이터는 각 장비들에 변경된 ICD가 정확하게 적용되었는지 확인하기 위하여 사용된다.

본 연구의 테스트 대상인 통합 시뮬레이터는 C++로 개발되었으며, Windows상에서 동작한다. 이 장비는 무장통제장치를 구성하는 일부 장비의 통신을 모의하거여 실장비와의 통신을 수행할 수 있다. 예를 들어, 통제콘솔을 테스트한다면 통제콘솔은 본 장비로 구성하고 나머지 주제어부, 연동기, 비행체 시뮬레이터의 통신은 모의해준다. 통합 모의기에 대한 기본 정보는 Table 6과 같다.

Table 6. SUT summary

	Overall	Per Module
NOM (No. Modules)	190	
LOC (Lines of Code)	14,122	74.326
CC (McCabe’s Cyclomatic complexity)	1,502	7.905

NOM은 모듈의 수를 측정하는 메트릭으로 클래스의 수를 나타낸다. LOC는 크기를 나타내는 메트릭으로 빈칸과 주석을 제외한 코드의 줄 수이다. CC는 복잡도를 나타내는 메트릭으로 함수당 조건문(if, for, while)의 수를 보여준다. 일반적으로 함수 당 10이하의 CC 값을 권장한다.

무장통제장치는 연동 장비들의 상태확인, 교전관리, 발사통제 등 다양한 기능들을 방위사업청의 “무기체계 소프트웨어 개발 및 관리 지침”에 따라서 컴포넌트 단위로 요구사항분석/설계/구현되고 있으며, 본 연구에서는 무장통제장치의 기능 중 핵심기능인 발사통제 컴포넌트를 대상으로 테스트를 진행하였다.

4.2 테스트 모델 생성

테스트 모델은 요구사항 및 상세 설계를 바탕으로 작성하였다. 작성된 테스트 모델에 대한 기본 정보는 Table 7과 같다.

Table 7. Test model summary

	Size
Number of States	9
Number of Transitions	17
Number of Input Events	11
Number of Round Trips	5

본 사례 연구에서 사용한 테스트 모델은 3장에서 예를 들고 있는 테스트 모델보다 상세한 수준의 상태머신이다. 구체적인 모델은 보안 관계상 생략한다.

4.3 테스트 케이스 생성

기존의 요구항목 기반 테스트 케이스와 W-Method, 본 연구에서 제안하는 확장 W-Method로 4.2절에서 생성한 테스트 모델을 대상으로 테스트 케이스를 생성하였다. Table 8은 생성된 테스트 케이스의 요약 정보를 보여준다.

Table 8. Test case summary

	Req. Item	W-Method	Extended W-Method
Number of TCs	4	11	
Total Input Length	14	39	54
Avg. TC Length	3.5	3.55	4.91
Number of Extended TCs	-	-	5

요구항목 기반 생성 방법은 4개의 테스트 케이스를 생성하였으며 W-Method로는 11개의 테스트 케이스가 생성되었다. W-Method를 이용하여 생성한 11개의 테스트 케이스 중 5개는 최종 상태까지 도달하지 않는 라운드 트립 패스이다. 확장 W-Method에서는 확장한 5개의 라운드 트립 패스와 최종상태까지 도달하는 나머지 6개의 테스트 케이스를 포함한 11개의 테스트 케이스를 생성하였다. 이때, 확장 W-Method로 생성한 테스트 케이스의 전체 길이는 확장한 5개의 테스트 케이스에 의해서 W-Method에 비해 15번이 많은 54번이다.

4.4 테스트 실행

본 실험의 테스트 대상인 통합 시뮬레이터에는 실제 미구현 기능들이 존재한다. 통합 시뮬레이터의 목적은 각 장비들간의 통신 기능을 확인하기 위한 것이므로 무장통제장치의 모든 기능이 구현되어 있지 않다. 즉, 누락된 기능이 있거나 실제로는 실행되어서는 안 되는 기능이 실행될 수 도 있다.

해당 기능을 실행하기 위한 입력 값이 다수 존재하는 경우에는 랜덤 값을 선택하였다. 예를 들어, 무장할당은 다수의 무장 중 하나를 선택해야 한다.

오류의 확인은 실제 무장통제장치와 비교하여 확인하였다. 실제 장비와 GUI를 비교하여 오류 확인하였으며, 다음 명령 입력 가능 여부, 출력 값, 색깔 등이 동일한지 확인하였다.

4.5 테스트 케이스 평가

이 절에서는 2.4절에서 기술한 4가지 테스트 케이스 평가 방법들로 각 테스트 케이스 생성 기법들을 평가한 결과를 보여준다.

4.5.1 커버리지 분석 결과

요구항목 기반, W-Method, 확장 W-Method 기법으로 생성한 테스트 케이스들이 요구사항과 상태 머신 모델에 기술된 내용을 얼마나 포함하고 있는지를 확인하기 위하여 커버리지 분석을 실시하였다. 본 연구에서 분석한 커버리지는 요구항목, 전이, 라운드 트립 커버리지 이다. Fig. 7은 커버리지 분석 결과를 보여준다.

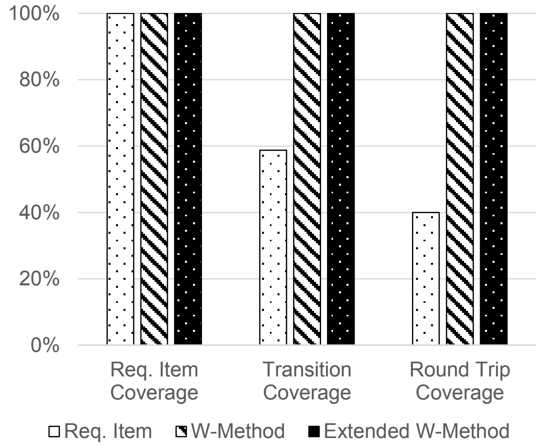

Fig. 7. Coverage evaluation result on the case study

W-Method와 확장 W-Method 기법으로 생성한 테스트 케이스는 세 커버리지를 100 % 만족하였다. 요구항목 기반으로 생성한 테스트 케이스는 8개의 요구사항은 모두 만족하였으나, 17개의 전이 중 10개만 실행하였고, 5개의 라운드 트립 중 2개만 실행하였다. 커버하지 못한 전이는 요구사항이 구현 단계에서 다양하게 구체화 될 수 있는 요구사항에 관한 것이었다. 예를 들어, 대상이 비가용 상태가 되는 것을 확인해야 되는 요구사항에 대하여, 전원이 켜진 상태와 카운트다운 상태가 더욱 세분화 된 상태로 나뉠수 있으나 단일 항목의 요구사항만으로 테스트 케이스를 생성할 때는 세분화된 상태를 고려하기가 어려웠다.

4.5.2 테스트 케이스 실패율 분석 결과

테스트의 목적은 오류를 찾아내는 것에 있으므로, 많은 테스트 케이스가 실패할수록 효과성과 효율성이 좋을 수 있다. Fig. 8은 세 기법으로 생성한 테스트 케이스의 테스트 케이스 실패율을 보여준다.

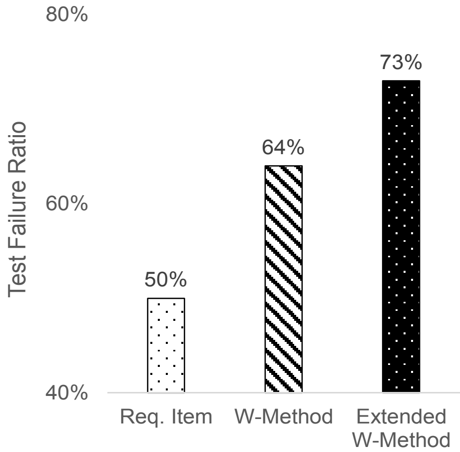

Fig. 8. Test failure ratio evaluation result on the case study

테스트 케이스 실패율 분석 결과, 본 논문에서 제안하는 확장 W-Method로 생성한 테스트 케이스의 실패율이 가장 높은 것으로 나타났다. 요구항목 기반 테스트 케이스는 4개 중 2개가 실패하였으며, W-Method와 확장 W-Method의 테스트 케이스의 수는 11개로 동일하지만 각각 7개와 8개의 테스트 케이스가 실패하였다. W-Method의 실패하지 않은 4개의 테스트 케이스 중 2개는 라운드 트립 패스로 확장 W-Method에 의해서 확장된 테스트 케이스이다. 이중 한 개의 테스트 케이스에서 W-Method로는 실패가 확인되지 않았으나 확장하여 테스트를 진행하는 동안 실패가 확인 되었다. 구체적으로 W-Method에서는 [무장할당 → 무장할당취소]로 끝나는 테스트 케이스에서는 오류를 확인할 수 없었다. 하지만, 확장 W-Method에의해 해당 테스트 케이스는 [무장할당 → 무장할당취소 → 무장할당 → ...] 의 재할당 기능을 테스트 하도록 확장되었고, 무장이 재할당되지 않으며 할당취소 기능에 치명적인 오류가 있음을 확인하였다.

4.5.3 결함 분석 결과

테스트 케이스 실패율이 높다는 것이 반드시 많은 오류를 찾아낸다는 것을 뜻하는 것이 아니다. 하나의 결함에 의해서 다수의 테스트 케이스가 실패할 수도 있다. 반대로 경미한(trivial) 오류나 보통(minor) 오류의 경우는 오류가 발견되더라도 이후의 절차가 더 진행 될 수 있으므로 하나의 테스트 케이스에 다수의 오류가 발견될 수 있다. Fig. 9는 테스트에 의해서 발견된 결함의 분석 결과를 보여준다.

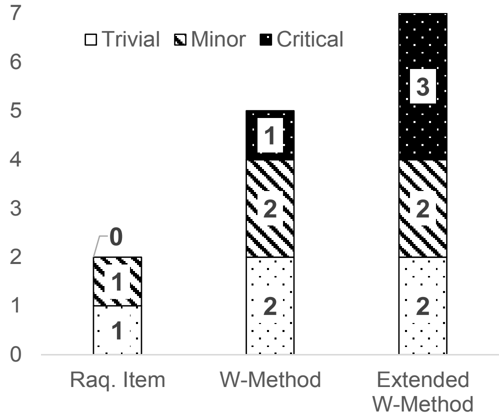

Fig. 9. Fault analysis result on the case study

발견한 결함 분석 결과 확장 W-Method로 생성한 테스트 케이스가 발견한 결함이 7개로 가장 많았고, W-Method 5개, 항목기반 2개 순으로 나타났다. 발견한 결함의 종류도 W-Method에 비해서 치명적인(critical) 결함을 두 개 더 발견하였다.

추가로 발견된 두 개의 치명적인 결함은 확장 과정에서 발견된 것으로 보통의 오류로 실패한 테스트에서 확장 테스트를 하는 과정에서 치명적인 오류를 추가로 발견하였다. 구체적으로, [... → 무장준비완료 → 패스워드입력 → 무장전원끔]으로 끝나는 W-Method에서는 무장의 상태 표시가 회색으로 되어야 하나 초록색으로 남아있는 오류가 있었다. 확장 W-Method에 의해 이 라운드 트립 패스를 확장한 테스트 케이스는 [... → 무장준비완료 → 패스워드입력 → 무장전원끔 → 무장전원켬 → ... → 패스워드입력 → ...]으로 다시 발사절차를 진행하며 이 과정에서 패스워드를 다시 입력받아야 하지만 패스워드를 입력받는 과정이 생략되어 넘어가는 오류를 발견할 수 있었다.

4.5.4 테스트 효율성 분석 결과

이 절에서는 테스트 케이스 당 발견한 결함 수를 계

산하는 테스트 케이스 효율성과 하나의 입력 당 발견한 결함 수를 계산하는 테스트 입력 효율성에 대하여 논의한다. Fig. 10은 테스트 케이스 효율성을 보여준다.

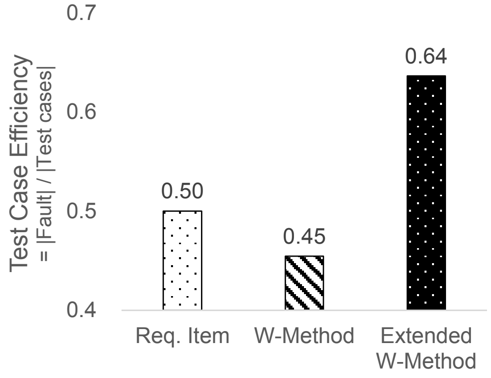

Fig. 10. Test case efficiency evaluation result

테스트 케이스 효율성 분석 결과 본 논문에서 제안하는 확장 W-Method가 테스트 케이스 하나 당 가장 많은 결함을 발견하는 것으로 나타났다. 확장 W-Method는 하나의 테스트 케이스 당 0.64개의 결함을 발견하였고, 요구 항목 기반은 0.5개 W-Method는 0.45개의 결함을 발견하였다.

Figure 11은 테스트 입력 효율성 평가 결과를 보여준다. 테스트 입력 효율성 평가에서도 확장 W-Method가 가장 우수한 것으로 나타났다. 확장 W-Method로 생성한 테스트 케이스는 입력 당 0.146개의 오류를 검출 하였고, 요구항목 기반 테스트 케이스는 0.143개, W-Method는 0.128개의 오류를 검출하였다.

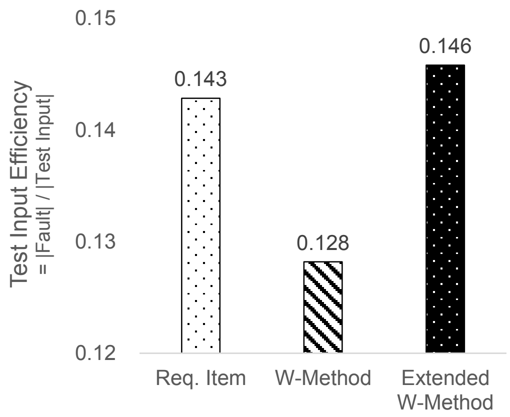

Fig. 11. Test input efficiency evaluation result

5. 결 론

무장통제시스템과 같은 국방, 항공 분야의 시스템은 높은 신뢰성이 요구되며, 체계적인 테스트를 통하여 신뢰성을 높일 수 있다. 이와 같은 이벤트 기반 임베디드 시스템은 상태 머신을 이용한 모델기반 테스트 기법이 유용하게 적용될 수 있다. 라운드 트립 커버리지를 만족하기 위한 W-Method 기법은 상태모델 기반 테스트 케이스 생성 기법 중 가장 효용성이 높다고 알려져 있다. 하지만 W-Method 기법은 최종 임무 단계까지 테스트를 수행하지 않으므로 관찰 가능성이 낮다. 따라서 본 연구에서는 W-Method를 최종 임무까지 수행하도록 수정한 확장 W-Method 기법을 제안하였다.

테스트 케이스 평가 결과 본 연구에서 제안한 확장 W-Method가 더 많은 오류를 더욱 효율적으로 발견하는 것으로 나타났다. 제안한 기법과 모델을 사용하지 않는 요구항목 기반 기법, 기존의 W-Method 기법으로 테스트를 수행하여 비교한 결과 요구항목 및 전이, 라운드 트립 커버리지에서 요구항목 기반 기법보다 우수하였으며, 테스트 케이스 실패율은 요구항목 및 W-Method에 비해 각각 23 %, 9 % 높았다. 발견한 결함 수는 각각 5개, 2개 더 많았으며, 테스트 케이스 효율성은 28 %, 42 % 더 높고, 테스트 입력 효율성은 각각 2 %, 14 % 더 높았다.

향후에는 본 기법을 다양한 종류와 크기의 시스템에 적용하고 뮤테이션 분석 등을 통하여 범용성과 일반성를 확인할 예정이다. 또한 테스트 자동화 도구의 입력 스크립트를 생성할 수 있도록 확장하여 테스트 용이성을 향상시킬 예정이다.

References

[1] El-Far, Ibrahim K., and James A. Whittaker, "Model-Based Software Testing," Encyclopedia of Software Engineering, 2001.

[2] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, "A Survey on Model-based Testing Approaches: a Systematic Revie," Proc. ACM Int'l. Workshop on Empirical Assessment of Sotware Eng. Languages and Technologies, pp. 31-36, 2007.

[3] S. Kansomkeat and R. Wanchai, "Automated-Generating Test Case using UML Statechart

Diagrams,” In Proc. 2003 Annual Research Conf. the South African Institute of Computer Scientists and Information Technologists on Enablement Through Technology, pp. 296-300, 2003.

[4] C. Mingsong, X. Qiu, and X. Li, “Automatic Test Case Generation for UML Activity Diagrams,” In Proc. the 2006 Int’l Workshop on Automation of Software Test, pp. 2-8, ACM, 2006.

[5] G. Antoniol, L. C. Briand, M. D. Penta, and Y. Labiche, “A Case Study Using the Round-Trip Strategy for State-Based Class Testing,” Proc. 13th Int’l. Symp. IEEE Software Reliability Eng., pp. 269-279, 2002.

[6] L. C. Briand, M. D. Penta, and Y. Labiche, “Assessing and Improving State-based Class Testing: A Series of Experiments,” IEEE Trans. Software Eng., Vol. 30, No. 11, pp. 770-783 , 2004.

[7] R. Binder, “Testing Object-Oriented Systems,” Addison-Wesley, 2000.

[8] UML, [Online]. Available: http://www.omg.org/spec/UML

[9] N. E. Holt, L. C. Briand, and R. Torkar, “Empirical Evaluations on the Cost-Effectiveness of State-based Testing: An Industrial Case Study,” Information and Software Technology, Vol. 56, No. 8, pp. 890-910, 2014.

[10] L. Briand, Y. Labiche, Y. Wang, “Using Simulation to Empirically Investigate Test Coverage Criteria based on Statechart,” Proc. 26th International Conference on Software Eng., pp. 86-95, 2004.