한국시뮬레이션학회 논문지
Vol. 27, No. 1, pp. 25-32 (2018. 3)
http://doi.org/10.9709/JKSS.2018.27.1.025
ISSN 1225-5904

모델 기반 내장형 소프트웨어의 효율적 신뢰성 시험 기법

박장성 † · 조성봉 · 박현룡 · 김도완 · 김성균

An Efficient Software Reliability Testing Method for the Model based Embedded Software

Jang-Seong Park† · Sung-Bong Cho · Hyun-Yong Park · Do-Wan Kim · Seong-Gyun Kim

ABSTRACT

This paper presents an efficient software reliability testing method for the model based auto-generated code and reify a dynamic test procedure. The benefits of executing the model-based each static/dynamic reliability test before the code-based static/dynamic reliability test are described. Also, The correlations of code/model based reliability test are demonstrated by using model testing tool, Model Advisor and Verification and Validation, and the code testing tool, PolySpace and LDRA. The result of reliability test is indicated in this paper.

Key words : Software Reliability Test, Static Test, Dynamic Test, Model Based Design, Auto-generated Code, Weapon System Embedded Software

요 약

본 논문은 모델 기반 내장형 소프트웨어의 자동 생성 코드에 대한 효율적인 신뢰성 시험 절차와 구체화된 동적 시험 방안에 대해서 제시하고 있다. 모델 정적/동적 시험 각각을 코드 정적/동적 시험 전에 수행함으로서 코드 신뢰성 시험 수행의 이점이 있음을 기술하였다. 또한, 모델과 코드의 신뢰성 시험 상관관계를 모델의 경우 Model Advisor와 Verification and Validation tool, 코드의 경우 Polyspace와 LDRA를 이용하여 살펴보고 제시한 절차대로 수행한 신뢰성 시험의 결과를 보여주고 있다.

주요어 : 소프트웨어 신뢰성 시험, 정적 시험, 동적 시험, 모델 기반 설계, 자동 생성 코드, 내장형 소프트웨어

1. 서론

내장형 SW(Software) 개발 시 항공, 군수, 선박, 자동차 등 여러 분야에서 알고리듬을 설계 및 모델링하고 모의(Simulation)하여 성능을 검증하는데 있어 사용자 입장에서 다양한 편의성 때문에 Simulink와 같은 모델 기반 SW 개발도구를 사용하는 사례가 증가하고 있다(Kim, Y.G. 등, 2014a, 2014b; Kim, Y.G. 등, 2015; Han, D.G. 등, 2016).

군수 장비의 SW 중 제어 및 항법 등을 담당하는 비행조종컴퓨터(Flight Control Computer)에 탑재된 GC (Guidance & Control) SW 개발 시 Simulink 등과 같은 모델 기반 SW 개발도구를 사용한다(Kim, Y.G. 등, 2014a, 2014b; Kim, Y.G. 등, 2015; Han, D.G. 등, 2016). 모델 기반으로 설계된 군수 장비의 SW의 신뢰성과 품질확보는 중요한 요소로서, 방위사업청(이하 DAPA(Defense Acquisition Program Administration))에서는 '무기체계 소프트웨어 개발 및 관리 매뉴얼'(DAPA, 2016)을 통해 신뢰성 시험항목에 대하여 지침을 제시하고 있다.

Kwon, K.Y.(2015)에서는 코드의 전반적인 SW 신뢰성 시험 관련 동향에 대해 설명하고 있다. 또한 기존 DAPA에서 수행하는 SW 신뢰성 시험 프로세스에 추가적으로 품질검증체계를 도입하여 양산 단계에서 발생하는 소프트웨어 문제를 최소화 하는 방안을 주장하고 있다.

Kim. J.W.(2015)에서는 코드 신뢰성 시험을 수행할

Received: 8 June 2017, **Revised**: 13 December 2017, **Accepted**: 21 December 2017

† **Corresponding Author**: Jang-Seong Park
E-mail: jangseong.park@lignex1.com
LIG NEX1 Co. Ltd.

때 높은 코드 실행률을 달성하기 위해 동적 시험 보다 정적 시험을 먼저 수행하는 것을 권하고 있다. 이는 정적 시험을 통해 Unreachable Code, Redundant Condition등과 같이 동적 시험을 수행하더라도 통과하지 못하는 코드들을 미리 걸러주기 때문이다. 또한 개발 상황에 따라 코드 동적 시험 시 유용한 방식을 비교 분석하고 있다.

Kim, Y.G. 등(2015)에서는 모델 기반으로 설계된 무인기 비행조종 컴퓨터 에 탑재되는 GC 알고리듬의 신뢰성 시험 절차를 제시하였다. Cho, S.O. 등(2009)에서는 Simulink 기반으로 설계된 무인기용 자동비행 프로그램의 모델 동적 시험에 대해서 Case study 형태로 수행하고 그 결과를 제시하였다. 그 외에 모델에서 자동 생성된 코드의 정적 시험 시 생기는 문제점들을 분석하거나(Kim, Y.G. 등 2014b), Simulink의 자동생성 코드가 코드 정적 시험 시 생길 수 있는 위배 항목 등을 분석한(Yoon, S.H. 등 2015) 논문들도 있으며, Kim, Y.G. 등(2014a)에서는 동적 시험 신뢰성 향상을 위한 모델 검증 방안을 다루고, Han, D.G. 등(2016)은 모델 검증 자동화 방안을 다루는 등의 논문들이 존재한다.

기존 연구들은 모델 기반 내장형 SW 개발 시 자동 생성 코드의 코드 신뢰성 시험 시 생길 수 있는 문제점을 분석하거나, 모델의 동적 시험 방안을 제시하였다. 또한, 코드의 신뢰성 시험 방안을 구체화 고 그 결과에 대해 제시하기도 하였다. 하지만 모델 기반의 내장형 소프트웨어가 모델과 연동하여 코드 동적 시험을 수행하는데 있어서 효율적이고 구체적인 시험 방안에 대해서 분석한 것들은 없다.

본 논문에서는 군수 장비 중 비행조종컴퓨터에 탑재되는 GC 알고리즘의 모델 기반 내장형 SW 개발 시 효율적인 신뢰성 시험절차와 구체화된 동적 시험 방안에 대하여 제시하였다. 시험 절차는 참고문헌과 실제 군용장비에 탑재되어 개발 중인 프로젝트에 적용한 결과를 바탕으로 만들었으며, DT(Developmental Test)전 신뢰성 시험에도 제안된 시험 절차대로 신뢰성 시험을 수행할 예정이다. 동적 시험 방안은 모델과 코드의 동적 시험 관계와 그 결과의 유사성을 확인하고 만들어진 절차로 현재 진행 중인 프로젝트에 적용되어 시험을 수행하였고, DT전 신뢰성 시험에도 제안한 방법으로 수행할 예정이다.

본 논문의 구성을 보면, 2.1절에서는 효율적 시험을 위한 시험 절차를 설명하고, 2.2절에서는 모델/코드의 정적 시험 관계와 그 결과를 설명한다. 2.3절에서 모델/코드의 동적시험 관계와 그 관계를 이용한 효율적인 동적 시험 방안을 제시하고, 그 결과를 설명한다. 또한, 제시한 절차를 현재 군용장비에 탑재되어 개발 중인 프로젝트에 적용한 신뢰성 시험 결과 및 분석결과를 포함하고 있다.

2. 본 론

2.1 모델 기반 내장형 SW 신뢰성 시험 절차

Fig 1은 본 논문에서 적용한 모델 기반으로 개발 된 알고리듬의 모델/코드 시험의 관계를 보여주고, Fig 2는 본 연구에서 적용한 모델 기반 내장형 SW 신뢰성 시험 절차이다.

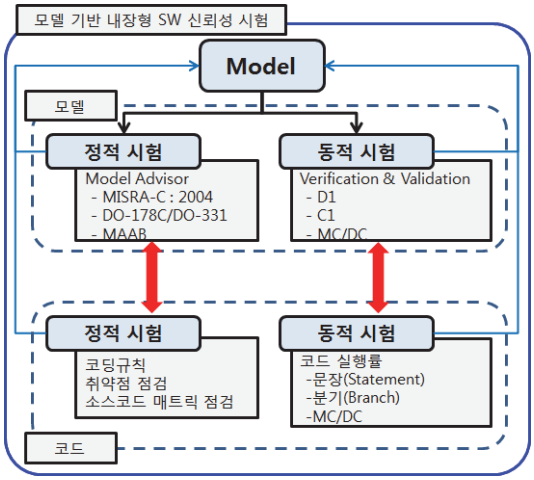


Fig. 1. Relationship of model and source code verification

절차에 대해 설명하면, 알고리듬 초안이 만들어진 상태에서 모델 기반의 정적 시험을 수행하고 모델 수정 필요 시 모델을 수정한다. 모델 기반 정적 시험이 완료되면, 코드 기반의 정적 시험을 수행하고 모델 수정 필요 시 모델 수정을 수행한다. 다음 단계로 모델 기반의 동적 시험을 수행하고, 필요 시 모델을 수정하고(수정 시 모델/코드 기반 정적 시험 재 수행), 코드 기반의 동적 시험을 수행한다. 해당 시험 절차는 DAPA.(2016) 의 기준을 충족할 때까지 반복한다.

위 절차를 준수하면, 일차적으로 모델 기반에서의 검증 과정을 거친 후 코드 기반에서의 정적 및 동적 시험을 수행하기 때문에, 코드기반에서 문제의 가능성이 있는 부분 중 상당 부분을 모델 기반에서 걸러낼 수 있다. 이는 코드 기반 신뢰성 시험 수행 전에 위배 요소를 최소화 할 수 있고 시험에 소요 되는 시간을 줄일 수 있다.

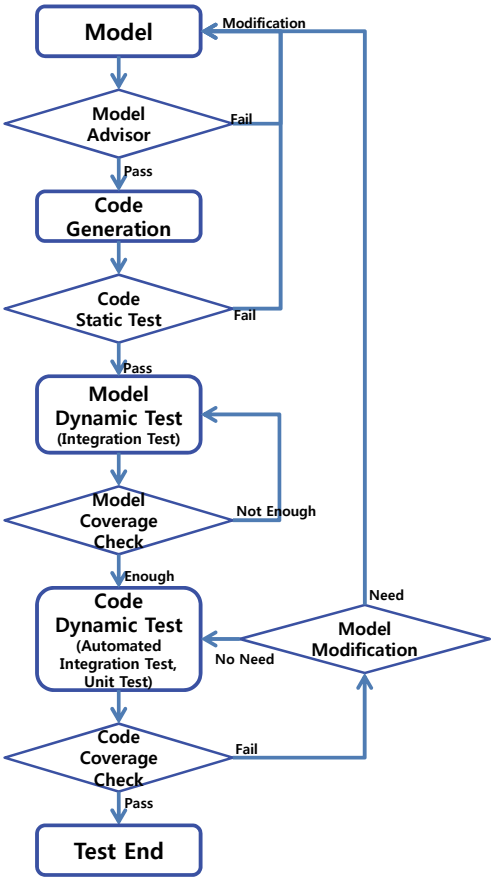


Fig. 2. Model/code test procedure

2.2 모델 기반 내장형 SW 정적 시험

DAPA.(2016)에 따르면 무기체계 내장형 SW의 정적 시험은 코딩 규칙 검증, 취약점 점검, 소스코드 메트릭 점검 3가지 항목으로 구분이 가능하다. 코딩 규칙은 DAPA.(2016)의 부록 6의 코딩 규칙 준수 여부를 확인하고, 취약점 점검은 사용언어(C/C++/JAVA 등)에 따라 CWE (Common Weakness Enumeration)-658/ 659/ 660 등 해당 취약점 점검 목록에서 자동화 도구가 지원하는 점검 항목은 모두 적용하도록 되어있다. 소스코드 메트릭은 함수를 기준으로 하며 여러 종류와 제한 값을 가진다. 단, 표준 적용이 제한되는 사업의 경우 66개의 프로그램 작성 규칙과 취약점 점검을 따르도록 되어 있다.

본 절에서는 모델/코드 기반 정적 시험의 관계를 설명하고, Simulink로 설계된 모델의 Model Advisor 수행결과와 Polyspace에 Misra C. 2004 기반의 66개의 프로그램 작성 규칙과 CWE-658을 적용한 시험 결과를 보여준다.

2.2.1 모델 코드 정적 시험 관계

Model Advisor에서 제공하는 항목들이 모든 작성 규칙과 정확하게 일치하는 것은 아니다. 하지만 대다수의 C/C++전용 규칙과 스타일, 초기화, 식별자, 포인터/배열 항목에서 대다수 항목들은 자동 생성 되는 코드의 경우 준수하도록 생성이 되어서 문제의 소지가 적다. 또한, 취약점 점검도 마찬가지로 자동 생성 코드의 경우 Buffer Overrun, Leak, Double Free와 같은 메모리, 버퍼 할당 등에서 문제를 일으키지 않도록 생성된다.

Model Advisor를 통해서 주로 확인이 가능한 코딩 규칙들은 조건식, 변환, 연산자와 관련된 항목들이 많고, 취약점 점검에서는 Cast Alters Value, Division By Zero, Overflow/Underflow와 관련된 항목들이 주를 이룬다. 이중 Division By Zero, Overflow/Underflow의 경우 이런 부분이 있는지를 확인 해 주기 보다는 이런 문제의 소지가 있을 수 있는 부분에 방어코드의 생성여부를 설정할 수 있다. 소스코드 메트릭(Code Metrics) 점검은 6개 항목 중 5개의 항목이 확인이 가능하다(Mathworks, 2015c, Kim, Y.G. 2014a).

Model Advisor를 통해서 내장형 SW 정적 시험을 통과하기 위한 많은 부분을 모델 단위에서 사전 확인하여 수정할 수 있지만, 모든 항목을 충족 하는 것은 현재는 불가능 하다. 이것은 Model Advisor의 원래 목적인 모델 기반의 설계자들이 협업함에 있어서 생길 수 있는 문제들을 해결하고, 모델의 결함을 확인 하는데 있기 때문이다.

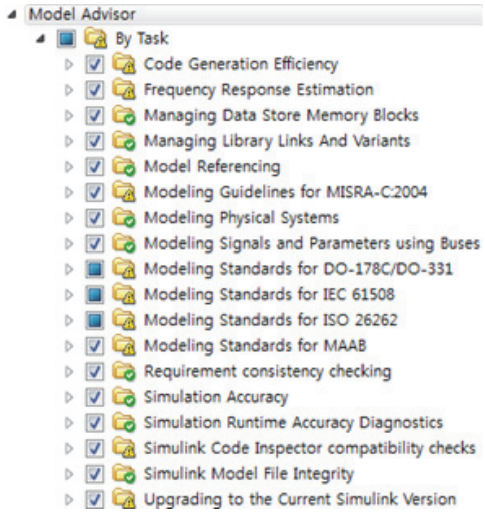
2.2.2 모델 기반 내장형 SW 정적 시험 결과

Table 1은 Matlab 2013a를 기준으로 수행한 Model Advisor 결과를 보여주고, Fig 3은 Task를 기준으로 적용한 대략적인 Model Advisor 설정이다.

Table 1의 Not run 항목은 주로 Model subsystem과 요구사항 문서들과의 Link를 확인하는 것들이거나, 결과 Report Display옵션, block 색깔, 글자 폰트들을 확인하는 것들 등에 해당한다. Warning 항목들은 코드 생성 최적화 관련 옵션 설정 여부와 이와 상충 되는 코드 생성시 안정성과 관련한 최적화 옵션 설정 여부, Code Inspector와 호환 불가능한 block 사용 여부 등에 해당한다.

Table 1. Model advisor result

Total	Pass	Fail	Warning	Not Run
231	157	0	49	25



Fig, 3. Applied model advisor options

Table 2는 코드기반 정적 시험 결과로 결함 0건에 False Alarm 6건이 나왔다.

Table 2. Code static test result

Tool	Defeat	False alarm
Polyspace Bug Finder	0	6

False alarm의 경우 if문의 조건이 항상 false라는 경고와 Division by zero의 가능성이 존재한다는 것이다. 하지만 해당 결함의 경우 Mathwork에서 제공하는 함수에서 나타나는 것들이고 발생할 수 있는 가능성이 존재하지 않았기에 False alarm 처리하였다.

2.3 모델 기반 내장형 SW 동적 시험

본 절에서는 내장형 SW 동적 시험을 수행하는 두 가지 방식 중 어떤 방식이 모델 기반 내장형 SW의 동적 시험에 유리 한지 설명하고, 그에 대한 Case Study와 실제 적용한 사업의 시험 결과를 포함하고 있다.

2.3.1 동적 시험

DAPA.(2016)에 따르면 무기체계 내장형 SW의 동적 시험은 요구사항기반으로 소프트웨어 코드 실행률을 점검하는 것이다.

코드 동적 시험을 수행하기 위한 방식으로는 상향식(Bottom-up) 방식과 하향식(Top-down) 방식으로 나눌 수 있다. 상향식 방식의 경우 단위 시험(Unit Test)으로 동적 시험을 먼저 수행하고 통합 시험(Integration Test)를 수행하는 방식을 말한다. 상향식 방식은 코드 기반으로 새로운 SW Module 설계 시 오류를 줄일 수 있다는 측면에서는 유용하다. 하지만 SW Module 개별적으로 시험을 수행하기 때문에 시험에 소요되는 시간이 매우 길다. 하향식 방식의 경우 통합 시험을 먼저 수행하고 부족한 부분을 채우기 위해서 단위 시험을 수행하기 때문에 시험에 소요되는 시간이 상향식 방식 대비 짧다. 하지만 SW Module별 검증에 어려움이 따르는 것으로 알려져 있다(Kim. J.W, 2015).

본 논문에서 제시 하는 모델 기반 내장형 SW의 동적 시험 방안은 모델 단위에서 모듈별 기능이 모델 상에서 검증이 가능하고, 자동 생성 코드를 사용하기 때문에 상향식 방식 보다는 하향식 방식이 유리하다. 또한, 하향식 방식을 사용할 경우 통합 시험을 위한 Test case를 추가로 만들 필요 없이 본 논문에서 적용한 모델 동적 시험(2.3.2.절) 수행 시 사용하였던 Test case를 사용하여 코드 동적 시험 시 자동화하여 검증이 가능하다. 위와 같이 모델 기반 내장형 SW의 동적 시험은 하향식 방식을 사용 하면 비용, 효율성 및 편의성 측면에서 유리하다.

2.3.2 모델 기반 내장형 SW 동적 시험 절차

Simulink에서의 모델 기반 동적 시험 중 모델 동적 시험의 경우 Verification & Validation 기능을 통해 수행이 가능하며, D1(Decision Coverage), C1(Condition Coverage), MC/DC(Modified Condition/Decision Coverage) 세 가지 결과를 얻을 수 있다. 여기서 D1의 경우 코드 실행률의 문장 실행률(Statement), C1은 분기 실행률(Branch)와 같고, MC/DC는 동일하다.

Mathworks에서 제안하는 모델 동적 시험은 2가지 방식이 있다. 첫 번째는 Test Case 자동생성 방식으로 Simulink가 모델을 분석하여 Test Case를 생성하여 결과를 보여준다. 두 번째는 User Design 방식으로 User가 Signal Builder Block이나 Script등을 이용하여 모델 기반 동적 시험에 적합한 Case를 만들어서 수행하는 것을 말한다(Mathworks, 2015d).

상기 2가지 방식에는 각각 장/단점이 존재 한다. 이를 정리하면 Table 3과 같다.

Table 3은 자동생성과 User Design 방식에 대한 각각의 장/단점을 설명하고 있으며, 본 논문에서는 위 두 가지 방식의 단점을 보완할 수 있으며, 코드 동적 시험 시 재사용 및 자동화가 가능한 방식의 모델 기반 내장형 SW의 동적 시험 방안을 제안하고 적용하였다.

Table 3. Auto generation/user design advantage /disadvantage

	Advantage	Disadvantage
Auto Generation	1. Fast enough to reach a specific level of execution 2. Easy to use	1. If the model is complex, case is not created 2. Unable to Reach a 100% of execution 3. Additional analysis needs to be performed via User Design function
User Design	1. Able to Reach a 100% of execution	1. For many required variable models, it takes a lot of time to build the environment. 2. Time required for Test Case analysis in Complex model

본 논문에서 수행한 모델 동적 시험 방안은 Fig 4와 같이 성능확인을 위한 시뮬레이션 모델이 있고, 그 안에 GC Block 과 같이 일부 검증하고자 하는 subsystem이 존재할 경우 시뮬레이션 데이터를 모델 기반 동적시험과 코드 동적 시험 Test Case로 사용하는 것이다.

Fig 4는 검증하고자 하는 GC의 입력이 되는 Measurement System의 출력을 Simulation 조건에 따라 Simulation을 수행하며 저장한다.

저장 된 출력은 Fig 5와 같이 검증하고자하는 모델만 따로 Simulink 모델로 구성한 검증용 모델의 입력으로 사용하여 모델 기반 동적 시험의 Test Case로 사용한다.

이 방법을 활용하면 모델이 복잡하고 검증 시 필요한 변수가 많더라도 이와 상관없이 손쉽게 동적 시험 환경을 구축할 수 있고 시험 결과 확인이 가능하다.

Test case를 늘려가며 모델 기반 동적 시험에서 일부 Coverage를 충족 하였다면, 모델 기반 동적 시험에서 사용한 Test case와 Fig 5와 동일한 모델에서 코드 동적 시험 수행이 가능하다.

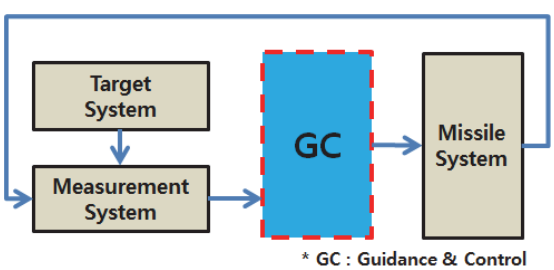


Fig. 4. Simulation and dynamic test object(GC)

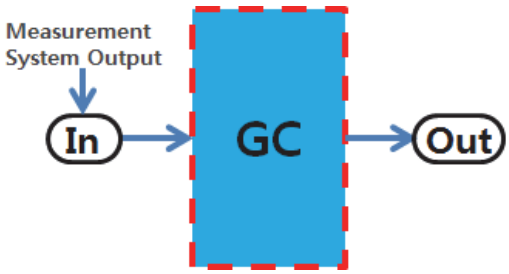


Fig. 5. Model based dynamic test

모델 동적 시험과 코드 동적 시험 절차가 매우 유사한데 이를 구분하는 이유는 저자가 적용한 알고리듬 모델의 경우 2.3.3절처럼 모델/코드의 동적 시험의 결과는 유사하지만, 소요되는 시간이 Table 4처럼 약 60배 이상 차이가 나기 때문이다.

Table 4. Model/Code static test result and used tool

	Model	Code
Necessary Time (1 Test Case)	1min	>60min
Tool	Verification & Validation	LDRA (Liverpool Data Research Associates)

즉, 동적 시험 간 소요되는 시간이 수배 이상 차이가 나기 때문에, 모델 동적 시험으로 Test case를 최소화 하며 Coverage를 늘릴 수 있는 Test set을 만들고 자동으로 코드 동적 시험(Fig 3. Automated Integration Test)을 수행하면, 개발자의 개입을 최소화 하며 코드 동적 시험에서 요구하는 Coverage까지 빠르고 효율적으로 수행이 가능하다. 저자가 수행한 프로젝트에서 동적 시험을 수행함에 있어서 소요되는 시간을 표로 나타내면 Table 5와 같다. 여기서 모델 동적 시험의 경우 모델 동적 시험을 수행하기 위해 Test case를 만들고 실제로 충족률 증가에 기여도가 있어 Test case 포함여부를 판단하는 등의 과정까지 포함된 시간이다.

Table 5. Working time of dynamic test result

	Model	Code Auto	Code Unit
Working Time	12hour	72hour	24hour

Automated Integration Test의 Working Time(Table 5

의 Code Auto)을 제외하고 개발자가 직접적으로 관여해서 시험을 수행하는데 걸리는 시간은 약 36시간 정도이다.

2.3.3 모델/코드 동적 시험 결과

모델 동적 시험의 결과와 코드 동적 시험의 결과의 유사성을 비교해 보기 위해 Test case를 늘려가며 두 시험의 결과를 비교 하고, 저자가 수행한 프로젝트의 결과를 제시 하겠다.

Table 6은 1개의 Test Case에 대한 모델/코드 동적 시험 결과이다.

Table 6. Model/code dynamic test result (1 Test Case)

	D1(Statement)	C1(Branch)	MC/DC
Model Based	80%	73%	43%
Code Based	80%	72%	40%

D1(Statement, 문장 실행률)의 경우 거의 동일한 결과가 나오고, C1(Branch, 분기 실행률)과 MC/DC의 경우 약간의 차이가 있는데 이는 모델 내의 Logical Operation이 코드 단위에서 검증되는 방식과 모델에서 검증하는 방식의 차이에 의한 것과, 코드 생성 시 Uint type등의 데이터의 Overflow를 막기 위해 추가 되는 if 구문 등으로 인해 발생한 것이다.

Table 7, 8은 Fig 4의 시뮬레이션 모델에서 초기조건을 바꾸어 10, 50개의 Test case를 생성하고 이를 이용하여 모델/코드 동적 시험을 수행한 결과를 나타내었다.

Table 7. Model/code dynamic test result (10 Test case)

	D1(Statement)	C1(Branch)	MC/DC
Model Based	90%	82%	53%
Code Based	90%	80%	47%

Table 8. Model/code dynamic test result (50 Test Case)

	D1(Statement)	C1(Branch)	MC/DC
Model Based	96%	93%	83%
Code Based	95%	91%	71%

결과를 통해 Test case를 늘려 시험 결과를 누적시키면 유사하게 변하는 것을 알 수 있다. 이를 통해 모델 동적 시험을 수행하고, 그때 사용한 입력들을 코드 동적 시험에 사용하면 매우 높은 수준의 결과를 자동으로 손쉽게 얻을 수 있다는 것이 확인 가능하다.

하지만, 이 방법의 경우 어느 정도 Test Case가 늘어나면, 복잡한 구조로 인해 통합 시험의 최상위 단에서 설정 변경을 통해 subsystem의 한 부분을 실행시킬 수 있는 Test Case를 만들기가 어렵다. 또한, 전체 실행률 상승에는 큰 도움이 되지 못한다는 것을 Table 6~8를 통해 볼 수 있다. 효율적인 신뢰성 시험을 위해서는 적정 수준 까지는 모델 기반 동적 시험의 Test case를 사용하여 자동으로 통합 시험 단위에서 코드 기반 동적 시험을 수행하고 단위 시험을 통해 Statement, Branch, MC/DC를 100% 달성한다.

아래 Table 9는 저자가 수행한 프로젝트의 동적 시험 결과이다. 저자가 진행한 프로젝트는 문장/분기 실행률 100%만을 요구하고 있어서 그에 대한 결과만 수록 하였다. Model은 모델 동적 시험 결과, Code는 코드 동적 시험 중 Automated integration test 결과를 말하며 Result는 코드 동적 시험(Automated integration test + Unit test)의 결과이다.

Table 9. Dynamic test result(52 Case)

	Model	Code	Result
Statement	96%	93%	100%
Branch	95%	92%	100%

3. 결론

본 논문에서는 모델 자동 생성 코드를 사용하는 내장형 소프트웨어의 신뢰성 시험을 수행하기 위한 절차와 시험을 효율적으로 수행하기 동적 시험 방안을 제시하고 그 과정과 결과를 기술하였다.

정적 시험은 모델 정적 시험을 Model Advisor로 수행 후 코드 정적 시험을 수행함으로써 시간 소요가 오래 걸리는 코드 정적 시험 전에 모델 단위에서 시험을 수행하여 모델을 수정함으로 써 정적 시험에 소요 되는 시간을 줄일 수 있었다.

동적 시험은 모델 기반 내장형 소프트웨어의 특성을 잘 활용하여 하향식(Top-down)방식을 적용하였다. 또한, 모델 동적 시험 결과와 코드 동적 시험 결과의 유사성과

코드 동적 시험 자동화가 가능하다는 점에 착안하여, 코드 동적 시험 중 통합 시험(Integration Test)을 자동화 하여 높은 충족률을 손쉽게 얻을 수 있다는 것을 보였다. 또한, 동적 시험 수행 시간을 보면, 모델 동적 시험을 수행함으로 써 개발자가 시험에 직접적으로 관여하는 시험 시간을 수배 줄일 수 있었다.

본 연구를 통해 고안한 시험 절차가 빈번하게 수정이 일어나는 개발 초기의 모델 기반 무기체계 내장형 SW나 유도/제어 알고리듬에 적용하기에 적합한 방법인 것을 확인 할 수 있다. 수정이 생길 때마다 코드 정적/동적 시험을 수행 하는 것에 비해 모델 정적/동적 시험을 수행하여 SW의 신뢰성을 확보하고 특정 이벤트 또는 주기로 코드 정적/동적 시험을 수행하면 시간과 비용 측면에서 매우 효율적으로 SW의 신뢰성을 확보 할 수 있고, 개발/시험 기간 단축 효과도 기대할 수 있다.

References

1. Kim, J.W, “A Case Study on Reliability Test of Embedded Software in the Multi-Function Radar”. Journal of IKEEE, Vol. 19, No, 3, pp. 431-439, 2015.9
2. Kim, Y.G., Yoon, H.S, Kim, S.H., “A Model-Based Design and Testing Approach for the UAV Flight Control Computer”, KSAS Conference, pp. 630-633, 2015
3. Kwon, K.Y., Joo, J.S., Kim, T.S., Oh, J.W., Baek, J.H., “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, 2015.7
4. The MathWorks, “Control Algorithm Modeling Guidelines Using Matlab, Simulink, and Stateflow Version 3.0,” The Mathworks Inc., 2012
5. The MathWorks, “Do Qualification Kit, Model-Based Design Workflow for DO-178C,” The Mathworks Inc., 2015
6. The MathWorks, “Managing Model-Based Design”, The Mathworks Inc, 2015
7. The MathWorks, “Model Advisor Checks for High-Integrity Modeling Guidelines,” The Mathworks Inc., 2015.
8. The MathWorks, “Simulink Verification and Validation,” The Mathworks Inc., 2015.
9. Han, D.G., Kim Y.G., Yoon, H.S, “Reasearch for Automated Model Test Method of Model-based Software”, KSAS Fall Conference, pp. 1208-1209, 2016
10. Kim, Y.G., Yoon, H.S., Kim, S.H, “Research of the Effects of the Model Style Guidelines for Improving Software Reliability through Dynamic Testing in Model Based Design”, KSAS Fall Conference, pp. 1707-1710, 2014.11
11. Kim, Y.G., Yoon, H.S, “Research on Effects of MAAB Style Guidelines for Weapon System Embedded Software Reliability Improvement,” Journal of Korea Institude of Military Science and Technology, Vol. 17, No. 2, pp. 213-222, 2014
12. Yoon. S.H., Kim, Y.W., Hwang, S., “Validation and Verification of Model-based Development”, KSAS Fall Conference, pp. 1101-1104, 2015.11
13. DAPA, “무기체계 소프트웨어 개발 및 관리 매뉴얼 제2016-4호,” 방위사업청(DAPA), 2016
14. Cho, S.O., Choi, K.Y., “A Study on Validation of OFP for UAV using Auto Code Generation”, Journal of The Korean Society for Aeronautical and Space Sciences, Vol. 37, No. 4, pp. 359~366, 2009.4

박 장 성 (jangseong.park@lignex1.com)

2011 인하대학교 항공우주공학과 학사
2013 한국과학기술원 항공우주공학전공 석사
2013~ 현재 LIGNEX1 재직 중

관심분야 : 유도조종, 필터, 항법, 소프트웨어 신뢰성

조 성 봉 (chosb@add.re.kr)

2003 부산대학교 항공우주공학과 학사
2005 한국과학기술원 항공우주공학과 석사
2005~ 현재 국방과학연구소 재직 중

관심분야 : 유도, 제어, 궤적최적화, 무미익 무인기 비행제어

박 현 룡

2014 충북대학교 전자공학과 학사
2014~2017 LIGNEX1
2018~ 현재 스타트업

관심분야 : 임베디드 소프트웨어, 소프트웨어 신뢰성

김 도 완 (dkim.bkcg@gmail.com)

2008 인하대학교 항공우주공학과 학사
2010 인하대학교 항공우주공학과 석사
2010~ 현재 인하대학교 항공우주공학과 박사
2013~ 현재 국방과학연구소 재직 중

관심분야 : 조종안정성, 비행제어, 유도기법, 소프트웨어 신뢰성

김 성 균 (seonggyun.kim@lignex1.com)

2012 인하대학교 항공우주공학과 학사
2014 인하대학교 항공우주공학전공 석사
2014~ 현재 LIGNEX1 재직 중

관심분야 : M&S, 항법, 유도, 제어