특집 07

ISO/DIS 26262를 대비한 자동차 제어기 테스팅 기법

박인권·강해달 (슈어소프트테크(주))

목 차 »

1. 서 론
2. 자동차 제어기 테스트 방법
3. 실용적 제어기 테스트 기법
4. 자동차 제어기 소프트웨어 테스트 자동화
5. 결 론

1. 서 론

경제성과 친환경으로 대변되는 미래의 자동차, 첨단 장치를 장착하고 청정에너지를 이용하는 것에 주력해 왔던 자동차업계의 이슈는 2010년 한 사건으로 인해 급속하게 안전과 품질로 바뀌었다. 탑승자의 편리함과 고연비를 위한 첨단제어 장비의 결함으로 인해 발생된 연이은 리콜사태는 제조사와 소비자에게 미래자동차의 청사진에 안전과 품질 문제가 결코 배제될 수 없다는 것을 깨닫게 하였다.

이에 아직 Draft 버전임에도 불구하고 국제적으로 자동차제품의 안전성을 규정한 표준인 ISO/DIS 26262에 대한 관심도 세계적으로 높아졌다. 특히 국내 자동차 업계는 기존의 완제품 제조와 함께 21세기형 고부가가치를 창출할 부품산업을 발전시키기 위해 IT기술을 융합하여 첨단부품을 제조하기 위한 노력을 해온 상황에서, 안전과 품질이라는 이슈를 해결하고 세계의 고급자동차 시장을 공략하기 위해서 필수적으로 받아들여야 하는 표준으로 인식하고 있다.

자동차는 기계적인 장치 이외에도 40여 개(2008년 기준)의 ECU라고 불리는 제어기와 그에 임베디드 된 소프트웨어로 이루어지며, 이 수는 지속적으로 늘어나는 추세이다[1]. 이와 함께 탑재되는 소프트웨어 안전성을 어떻게 검증할 지에 대한 관심도 증가하고 있다.

이를 반영한 듯 ISO/DIS 26262 는 자동차 제어 SW의 안전등급(ASIL: Automotive Safety Integrity Level)을 A, B, C, D 네 등급으로 분류하고 있으며, 등급별로 사용을 권장하는 기법을 구분하고 있다. 다음의 <표 1>은 SW 개발 단계별 적용 가능한 기법수를 정리하고 있다. 총 93개의 소프트웨어 안전과 품질 기법중 중 소프트웨어 요구사항 도출, 설계, 개발과 관련된 기법은 64개 이며, 단위테스팅, 통합테스팅, 검증과 관련된 기법은 26개이다. 이 외에 형상관리와 관련된 기법 3가지가 Appendix로 소개되고 있다[2].

〈표 1〉 ISO/DIS 26262 소개된 소프트웨어 안전과 품질 관련 기법의 개수

개발단계	관련항	제목	기법수
Initiation of product development at the software level	5.4.6	Topics to be covered by modeling and coding guidelines	8
Specification of software safety requirements	6.4.6	Methods for the verification of requirements	4
Software architectural design	7.4.1	Notations for software architectural design	3
	7.4.3	Principles for software architectural design	7
	7.4.15	Mechanisms for error detection at the software architectural level	5
	7.4.15	Mechanisms for error handling at the software architectural level	4
	7.4.18	Methods for the verification of the software architectural design	7
Software unit design and implementation	8.4.1	Notations for software unit design	4
	8.4.5	Design principles for software unit design and implementation	10
	8.4.5	Methods for the verification of software unit design and implementation	6
	8.4.5	Methods for the informal verification of software unit design and implementation	6
Software unit testing	9.4.2	Methods for software unit testing	5
	9.4.3	Methods for deriving test cases for software unit testing	4
	9.4.4	Structural coverage metrics at the software unit level	3
Software integration testing	10.4.3	Methods for software integration testing	5
	10.4.4	Methods for deriving test cases for software integration testing	4
	10.4.5	Structural coverage metrics at the software architectural level	2
Verification of software safety requirements	11.4.3	Test environments for conducting the software safety requirements verification	3
(Appendix)Software Configuration	C.4.9	Mechanisms for the detection of unintended changes of data	3
합 계			93

물론 여기서 소개된 기법을 모두 사용하도록 강제된 것은 아니다. 표준에서는 7가지의 개발 단계별로 개발하고자 하는 아이템의 ASIL과 기업과 개발 제품의 상황, 개발 비용을 고려하여 적극추천(Highly recommendation) 하는 기법을 중심으로 개발 및 검증을 실시하도록 하고 있다.

따라서, 곧 세계적인 자동차 핵심 기술의 개발 표준이 될 것으로 보이는 ISO/DIS 26262에 대비하는 자동차 업계의 기업들은 제조하고자 하는 제품에 따라 각 기법들을 비용과 효용성을 따져 단계적으로 도입하는 것이 중요하다.

본 연구에서는 자동차의 제어기에 탑재되는 소프트웨어에 대해, 소프트웨어 단위 테스팅(Software unit testing)과 소프트웨어 통합 테스팅(Software integration testing)을 효과적으로 수행할 수 있는 방법에 대해서 다루고자 한다.

2. 자동차 제어기 테스트

2.1 자동차 제어기 소프트웨어의 현황 및 특징

2.1.1 자동차 시장의 변화

자동차에 사용되는 제어기 소프트웨어는 탑승자의 사용성과 보행자의 안전 등을 위해 점점 복잡해지는 추세이다. 또한, 치열해지는 경쟁과 끊임없이 변화하는 소비자의 취향에 발맞추어 다양한 라인업을 갖추고 새로운 모델의 출시가 잦아지고 전자기기를 이용한 편의기능이 확충되면서 소프트웨어 개발 기간도 단축되고 있다.

안전성 검증이 필요한 소프트웨어가 점점 늘어나고 복잡해지고 있으나 검증할 때 사용할 수 있는 시간은 줄어들고 있는 것이다. 이에 대응하여 비용을 고려한 소프트웨어의 개발 및 검증 프로세스를 최적화하고 자동화하는 작업은 자동차 시장의 글로벌 경쟁에서 앞서 나가기 위한 필수적인 작업이다.

2.1.2 자동차 제어기 소프트웨어의 특징

자동차 제어기 소프트웨어는 자체적으로도 다음과 같은 특징을 가지고 있다.

첫 번째는 소프트웨어가 동작하는 하드웨어의 자원 환경이 다양하다는 것이다. 자동차는 인포테인먼트, 바디, 보안, 안전, 샤시, 파워트래인 등 각 부분별 부품으로 구성되며, 각각 부분에는 그에 상응하는 기능을 가진 임베디드 시스템이 탑재된다. 이 때, 인포테인먼트에 사용되는 자원은 비교적 고사양으로 변화하는 추세이지만, 나머지 부분은 대다수 CPU성능, 메모리 등에 심각한 제약을 가지고 있다.

두 번째는 제어기 소프트웨어의 운영환경이 특정되지 않았다는 것이다. 최근 OSEK 기반의 운영체계를 도입하는 움직임이 있지만, 일부 프로젝트에만 적용되는 것으로 대다수의 ECU에는 운영체계를 사용하지 않는다.

세 번째는 제어기 소프트웨어의 동작환경변수가 연속적으로 변화한다는 것이다. 이는 제어기에 입력되는 값이 타이어의 온도처럼 상당수 센서에 의해 측정되는 실제 자연 현상인 경우가 많기 때문에 생겨난 특징이다.

네 벤째는 오랜 시간을 지속적으로 동작하는 임베디드 시스템에 다양한 기능을 동작시키기 위해 외부 입력 데이터 외에 소프트웨어 자체적으로 생성되는 주기적인 신호가 있다는 것이다. 이 신호들은 종종 인터럽트의 형태로 소프트웨어에 특정 기능을 동작시킨다. 예를 들면 네비게이션이 주기적으로 GPS 신호를 받기 위한 활동을 하는 것이다.

위의 네가지 특징으로 인해 일반적인 소프트웨어 개발환경에서 사용되는 테스팅 자동화 기술과 도구는 바로 적용되는데 한계가 있다.

2.1.3 자동차 제어기 소프트웨어 개발 프로세스 특징

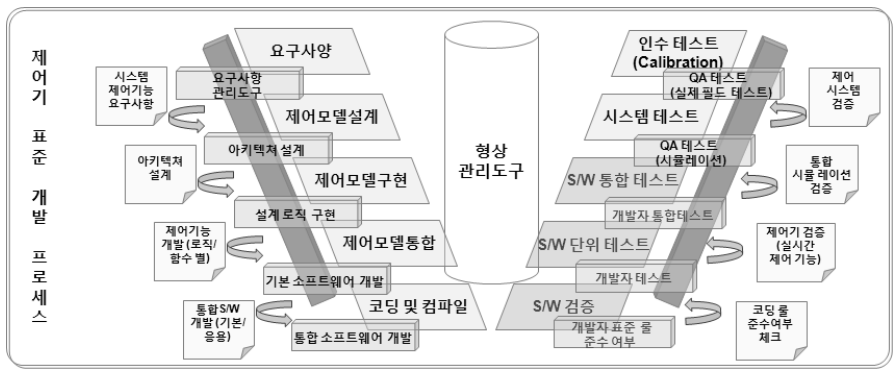
자동차 제어기는 (그림 1)의 표준 개발프로세스에 따라 진행된다. 표준 개발프로세스는 시스템 요구사항 분석, 설계, 구현, 테스트에 이르는 V모델로 이루어져 있다. 전체 개발프로세스중 소프트웨어 테스트에 대한 이슈는 다음과 같다.

가. 다수의 협력업체 인원에 의한 개발 수행

다수의 협력업체 인원이 개발 및 테스트에 참여함에 따라 개발 인원의 품질수준 및 사용 표준이 균질하지 않아 테스트에 소요되는 노력의 효율적인 분배가 어렵다.

나. HW와 SW 결합된 환경에서의 시험 필요

자동차 제어기는 HW와 SW가 결합된 환경에



(그림 1) 차량제어기 개발 표준 프로세스

서의 시험이 필요하다. HW등의 준비에 따른 시험 준비이 길어지고 HW자원의 제약에 따라 테스트 수행 노력이 증가하는 문제가 발생한다.

다. 실 환경 발견 오류의 재연의 어려움

다양한 센서 등의 입력이 포함된 HW환경에서 시험이 수행됨에 따라 문제점이 발견되어도 해당 문제를 야기한 지점에 대한 확정이 어렵고, 이 결함을 수정하기 위해 재현하기가 어렵다.

라. 테스트 완료의 기준 확정이 어려움

테스트에 배정된 기간 및 작성한 테스트케이스를 모두 사용하였더라도 어느 정도의 테스트가 수행되었는지를 정량적으로 판단할 수 없으므로, 테스트의 완료기준 확정이 어렵다.

2.2 소프트웨어 레벨의 단위/통합 테스팅의 필요성

제품의 품질 검증을 위한 테스트 활동은 전통적으로 실제품에 탑재하여 동작을 시험하는 방식을 선호한다. 최종제품의 동작은 일반인들도 확인할 수 있으며, 최소한의 동작을 보장할 수 있기 때문이다. 이런 검증과정은 자동차 분야에서도 이미 반영이 되어있는데, 제품을 양산하기 전에 행해지는 실차테스트가 바로 그것이다. 실차 이전에도 일부 부품에 대하여 HILS(Hardware-in-the-loop simulation)를 이용한 시뮬레이션 테스트를 하고 있다. 이를 반영하여 ISO/DIS 26262는 Hardware-in-the-loop, Electronic control unit network environments, Vehicles 세가지를 안전 요구사항 검사 단계의 환경으로 제시하고 있다.

그러나, 각종 제어 소프트웨어가 포함된 자동차를 일종의 시스템 테스트인 시뮬레이션 테스트와 실차테스트만으로 완벽하게 테스트하는 것은 재정적, 시간적인 소모가 크다. 자동차 각 부분이 내부적으로 연결되어 물리적으로 드러나지 않게 자동제어기능이나 각종 예외상황에 대한 대처 기능 등은 그 경우의 수와 유사환경구성의 어려움으로 인해 제품의 최종 테스트 단계인 시스템 테스트로 재현하고 시험하는 데 막대한 비용이 필요하게 된다.

이를 반영하듯, 자동차의 안전성을 강조한 ISO/DIS 26262 에서도 소프트웨어 소스코드 내부를 정밀하게 검증하여 제품 표면상에서 발견하기 힘든 오류들을 조기에 발견하기 위해 소프트웨어 단위테스팅 및 통합테스팅을 사용하도록 하고 있다. 단위/통합 테스팅에서 사용하도록 안내한 기법들은 <표 2>와 같다.

〈표 2〉 ISO/DIS 26262에 소개된 단위/통합 테스팅의 기법들

분류	기법	적용 단계
테스팅 방법	Requirement-based test	단위/통합 테스팅
	Interface test	
	Fault injection test	
	Resource usage test	
	Back-to-back test between model and code, if applicable	
테스트 케이스 도출 방법	Analysis of requirements	
	Generation and analysis of equivalence classes	
	Analysis of boundary values	
	Error guessing	
커버리지 메트릭	Statement coverage	단위 테스팅
	Branch coverage	
	MC/DC(Modified Condition/Decision Coverage)	
	Function coverage	통합 테스팅
	Call coverage	

2.3 기존 단위/통합 테스팅의 한계

기존의 단위/통합 테스팅은 환경을 구성하는 방법에 따라 두가지 형태로 진행되었다.

첫 번째 형태는 개발 환경(개발자 PC 나 개발 호스트) 상에서의 테스팅이다. 이 때 자주 쓰이는 방법은 개발도구에 XUnit 기반의 프레임워크를 적용하여 개발자가 테스팅 코드를 제작하여 직접 테스팅 하는 방법과 하드웨어 개발회사가 제공할 경우에 애뮬레이터를 통한 테스팅도 가능하다.

두 번째는 개발대상인 타겟 보드 상의 테스팅이다. 이 방법은 소프트웨어가 동작하는 환경을 테스트할 수 있는 테스팅 보드가 별도로 제공될 경우에 진행이 가능하며, 상용 디버깅 도구를 함께 이용하기도 한다.

두가지 방법은 각각 장단점을 가지고 있으며,

〈표 3〉 ISO/DIS 26262에 소개된 단위/통합 테스팅의 기법들

분류	테스팅 형태	장점	단점
개발환경 상 테스팅	XUnit 프레임워크	무료사용 가능 개발자의 자유도가 높음	테스팅 코드를 직접 제작해야함 실제 장비에 대한 적용성을 보장하기 힘듦 개발자의 수작업에 의존 다량의 실제 데이터 이용이 힘듦
	애뮬레이터 이용	개발하는 하드웨어에 가장 유사한 시뮬레이션 환경	개발플랫폼 및 HW 마다 별도 애뮬레이터 제작 필요
타겟 보드상 테스팅	디버깅 도구 이용	임베디드 제품의 실제 동작을 가장 유사하게 테스팅	테스팅 시점 지연 (HW개발 이후 테스팅 가능) 임베디드 장비 성능에 따른 하

이와 같은 내역은 <표 3>과 같다.

본 연구에서는 기존 테스팅 방법의 장점을 살리고 단점을 보완하면서 HW에 대한 의존성을 최소화하고 개발환경 상에서 테스트 가능한 실용적인 소프트웨어 단위/통합 테스팅 방법을 제시한다.

3. 실용적 제어기 테스트 기법

3.1 코드 컨버팅을 통한 시뮬레이션 테스팅 기법[3]

임베디드 시스템 소프트웨어는 개발자 PC와 Host환경에서 개발하고, 임베디드 환경에서 실제로 동작하게 된다. 이렇게 이중화된 환경으로 인해 임베디드 소프트웨어는 임베디드 환경에 의존적인 부분과 독립적인 부분으로 나뉠 수 있다. 코드 컨버팅을 통한 시뮬레이션 테스팅 기법은 임베디드 환경에 의존적인 부분을 제거 또는 변형하여 개발환경에서 테스트 할 수 있도록 하는 기법이다.

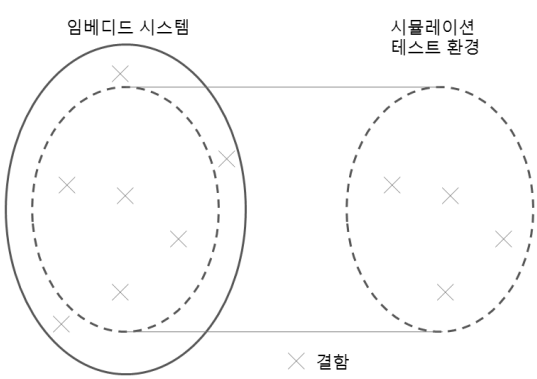
〈표 4〉 코드 컨버팅 항목

주요 변환 항목	변환내용
직접 주소 처리	Target 보드내의 주소를 직접 접근하는 경우, Host내의 메모리 주소로 변환
주소 변수 처리	실제 주소를 가리키는 변수인데 pointer 변수로 설정하지 않은 경우 해당 타입의 변수를 선언하여 공간을 할당하고 이의 주소를 설정하여 수행
Task 함수 처리	무한 루프 형태로 구성되어 있는 Task형태 함수를 하나의 case가 한번의 loop를 수행하는 것으로 수행
ASM 제거	소스코드상의 HW dependent ASM 코드 제거

이 때, 변형되는 주요항목 및 내용은 <표 4>와 같다.

이런 변환 항목은 하드웨어 의존적인 부분에 해당한다. 따라서, 컨버팅된 코드에서 발견된 결함은 실제의 소프트웨어에서도 존재하다고 볼 수 있다. (그림 2)의 소프트웨어 결함 분포는 이와 같은 포함관계를 설명해 준다.

이 기법은 컨버팅된 코드를 이용해 소프트웨어 테스트를 정밀하게 수행하는 것으로, 실 장비 탑재 이후에는 하드웨어 의존적인 부분을 추가로 테스트하여 소프트웨어 결함을 조기에 발견하도록 하고, 시스템 테스트에 드는 시간을 이전 개발 단계로 분산시켜 전체 개발 기간을 단축하고, 엄밀한 소프트웨어 테스트 수행을 가능하도록 한다.



(그림 2) 임베디드 소프트웨어 결함분포

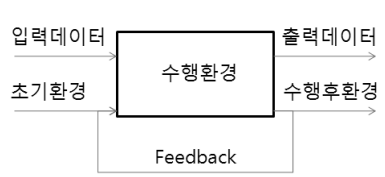
3.2 Continuous Test 및 Data Feedback 기법

자동차 제어기 시스템의 주요 특징중 하나는 다양한 종류의 데이터가 일정 기간이상 지속적이고 연속적으로 입력되는 환경이다. 예를 들어 자동차의 속도는 시속100km에서 바로 다음순간에 0km로 바뀌지 않고 브레이크의 효과에 의해 점차적으로 줄어든다. 이 때, 바퀴의 압력이나 차체의 기울기 등도 함께 연속적으로 바뀌게 된다.

그러나, 일반적인 소프트웨어 테스트 수행 기법은 프로그램 수행환경과 입력데이터, 출력데이터로 이루어지며 각 입력 데이터를 수행할 때마다 프로그램 수행환경을 초기화시킨다. 따라서, 실제 환경과는 다르게 시속 100km, 시속 50km, 시속 0km이 불연속적으로 입력되게 되며, 그에 따른 바퀴의 압력이나 차체의 기울기도 각각 독립적으로 입력된다. 이런 테스트 기법으로는 지속적인 데이터 수행환경에서의 결함을 발견할 수 없다는 문제가 있다.

이를 위해 수행된 환경을 지속적으로 유지하면서 대량의 데이터를 실 운영환경과 유사하게 하여 테스트를 수행하는 기법이 필요하다. 이에 적용되는 것이 제어기 소프트웨어 테스트 수행 후, 변화된 환경 값을 다음 테스트의 초기환경으로 세팅하는 Feedback 기법이다. (그림 3)은 이 기법을 간략하게 보여준다.

또한, 주기적인 소프트웨어의 동작을 시험하기 위해서는, 테스트데이터를 생성할 당시 실 운영



(그림 3) 테스트 환경 변수의 Feedback

〈표 5〉 테스트데이터 입력 기법

입력 기법	설명
범위값	최소, 최대값 등의 범위를 지정하면 범위내에서 지정된 개수의 데이터를 임의로 생성한다.
직접입력	사용자가 데이터 값을 직접 입력한다
등차수열	최소값, 등차, 개수의 정보를 지정하면 지정된 개수의 데이터를 등차로 생성한다.
주기수행	주기, 값 등을 지정하면 각 주기당 1회씩 데이터를 생성한다.

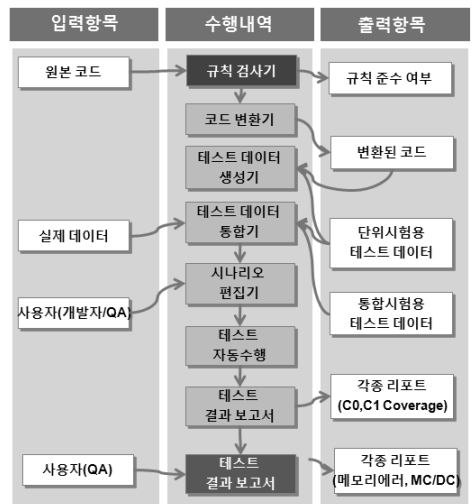
환경에서 발생할 수 있는 각종 주기데이터를 생성할 수 있어야 한다. 이를 위해 <표 5>와 같은 테스트데이터 입력 기법을 사용할 수 있다.

4. 자동차 제어기 소프트웨어 테스트 자동화

4.1 제어기 SW 동적시험 자동화 도구 (CodeScroll™ ControllerTester)[4]

CodeScroll™ ControllerTester는 코드컨버팅 기법과 기존의 동적 단위시험 기능이 결합된 제어기 SW용 동적시험 도구이다. 이 도구는 코드컨버팅, 프로그램 소스 분석, 테스트케이스 작성, 테스트 수행, 결함 분석 기능을 포함하고 있다.

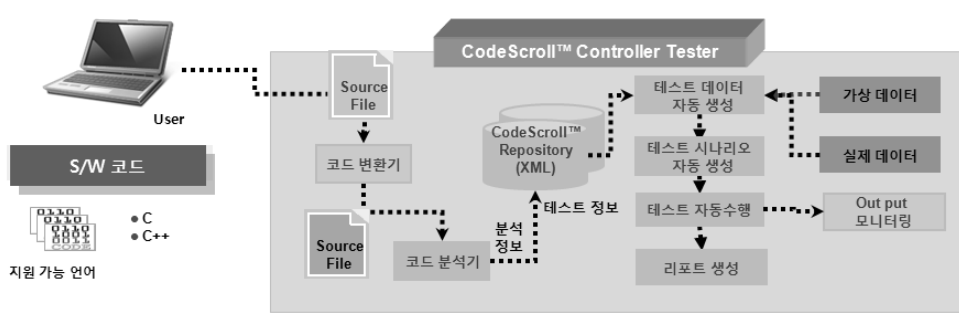
CodeScroll™ ControllerTester는 코드 변환기를 통해 개발환경에서 동작가능하도록 소스코드를 변환하고 이를 분석하여 테스트에 필요한 각종 정보를 추출한다. 또한 테스트데이터 생성 시에는 소스코드 분석을 통해 자동으로 생성한 데이터와 실제 차량의 데이터를 통합하여 테스트데이터를 생성한다. 생성된 테스트데이터에 대한 수행 시나리오를 작성하고 자동으로 수행하는 기능을 가지고 있다.



(그림 5) CodeScroll™ ControllerTester 수행도

4.2 제어기 SW 동적시험 자동화 도구를 이용한 적용 결과

제어기 SW 동적시험 자동화 도구인 CodeScroll™ ControllerTester는 기존 단위/통합 테스트의 문



(그림 4) CodeScrollTM ControllerTester 구조도

제점 개선하여 다음과 같은 사항을 지원하고 있다.

가. 다양한 하드웨어 환경을 단시간에 지원

다양한 하드웨어에 적용되는 CPU, OS 및 컴파일러 특성에 따른 커스터마이징 기간을 대폭 단축하여 자동차의 새로운 모델 라인업에 쉽게 대응할 수 있다.

나. 개발환경상에서 하드웨어와 가장 유사한 패턴의 테스트

자동차 제어기 하드웨어에 대한 애뮬레이터가 제공되는 것이 현실적으로 불가능한 상황에서 하드웨어와 가장 유사한 테스팅 방법을 제공한다.

다. 다양한 실데이터를 자동 테스트가 가능

기존에 측정된 자동차 운전상의 실데이터를 가공하여 다량의 데이터를 자동으로 테스트 할 수 있다.

라. 통일된 테스트 품질

자동화된 테스트케이스 작성 및 수행을 통해 개발자의 역량등에 따른 수준차를 상쇄하고 통일된 테스트 품질을 보장할 수 있다.

마. 테스트오류재연

오류 발생 위치 및 재연 정보 제공함으로써 SW중 문제가 되는 부분을 파악함은 물론 문제수정후 정말로 해결되었는지를 알 수 있도록 하고 있다.

바. 테스트 완료기준 제공

소스코드에 대한 커버리지 결과를 제공함으로써 테스트가 미비한 부분에 대한 파악 및 테스트 완료 여부에 대한 객관적 기준 제공하고 있다.

4.3 향후 발전 방향

CodeScroll™ ControllerTester는 시뮬레이션 환경에서 테스트를 수행하여 사용 시점의 단축 및 성능 개선 등의 주요한 강점을 가지고 있다. 그러나 자동차제어기의 품질수준을 높이기 위해서는 실 환경에서만 발생하는 오류를 발견하기 위한 개선이 필요하다.

가. 하드웨어 시뮬레이션

현재의 방식은 하드웨어 의존적인 부분은 제거하는 방식을 주로 사용하기 때문에 하드웨어 연계된 부분에서의 결함을 발견하는데 한계가 있다. 하드웨어를 시뮬레이션하는 방식을 도입하여 실환경에 최대한 가까운 환경으로 테스트를 수행할 수 있도록 지원할 예정이다.

나. HW테스팅과 연계

시뮬레이션 환경에서 작성된 테스트케이스를 HW환경의 테스트에서 사용할 수 있도록 연계하는 것을 지원할 예정이다.

4. 결 론

IT융합 산업에 대한 투자와 기대가 모아지는 요즘, 나로호, 도요타 사태 등 일련의 사건으로 그 핵심이 되는 임베디드 제품의 품질에 대한 요구도 함께 증폭되고 있다. 또한 거의 전 산업에서 선진국에 의해 국제표준, 제품품질 강화라는 기술장벽이 등장하고 있으며, 특히 자동차 분야는 ISO/DIS 26262 라는 표준을 제정하여 타산업에 비해 발빠르게 움직이고 있는 중이다.

글로벌 기업은 이에 대응하기 위해 시장성있는 제품을 적기에 고품질로 출시할 수 있는 프로세스를 갖춰야 하며, 기존 제품과 차별화하고 고부

가가치를 창출할 수 있는 소프트웨어에 대한 검증 기술은 이 프로세스의 핵심이라고 할 수 있다.

여기에서는 국내 자동차 업계의 개발현황과 실제 현장 경험에 비추어, 향후 중요하게 대두될 소프트웨어 품질 문제에 대응할 수 있는 실용적인 테스팅 방안을 제시하였다. 이 방안은 자동차 제어기 소프트웨어를 검증하는 방법으로 하드웨어에 탑재하기 이전에 소프트웨어의 기능을 검증하는 방법으로 저렴한 비용으로 실환경에 유사한 테스팅을 수행할 수 있도록 하는 것이다.

물론, 제품의 신뢰에 대한 문제는 소비자의 눈과 상대적인 경쟁 기술에 따라 지속적으로 변화하고, 소프트웨어 테스팅 기술 역시 산업동향이나 소비자의 니즈에 따라 비용과 효과를 고려한 끊임없는 연구개발은 필수적인 요소이다.

참 고 문 헌

[1] SW Insight 2009 컨퍼런스

[2] Draft International Standard ISO/DIS 26262-6 Road vehicles-Functional safety- Part6 Product development: software level

[3] CodeScroll™ Controller Tester 소개자료, 슈어소프트테크(주), 2009

[4] http://www.suresofttech.com

저 자 약 력

박 인 권

이메일 : inkpark@suresofttech.com

- 1995년 한양대학교 전자계산학과(학사)
- 1995년~2007년 (주)LGCNS 차장
- 2007년~현재 슈어소프트테크(주) 기술기획팀장
- 관심분야 : 품질보증, 프로세스 개선, 소프트웨어 테스트 및 자동화

강 해 달

이메일 : rsun@suresofttech.com

- 1999년 포항공과대학교 물리학과(학사)
- 1999년~2005년 (주)위트정보통신, O1 Korea inc 프로그래머
- 2005년~2007년 (주)엔토시스 개발본부 과장
- 2007년~현재 슈어소프트테크(주) 기술기획팀
- 관심분야 : 품질보증, 프로세스 개선, 소프트웨어 테스트 및 자동화