상상을 현실로,
소프트웨어가 미래다

2018 SW 산업

자동차 SW 안전가이드

2018 SOFTWARE SAFETY GUIDE

nipa 정보통신산업진흥원
National IT Industry Promotion Agency

Contents

Contents / 그림

Contents / 표

PART 1

가이드 개요

1 배경 및 필요성

2 목적 및 범위

3 가이드 구성

4 관련 문서

SOFTWARE

상상을 현실로,
소프트웨어가 미래다

I. 가이드 개요

1. 배경 및 필요성

전기전자기술의 발달로 차량의 기능 및 운행 제어에 대한 전자시스템 및 SW 에 대한 의존성이 높아지고 차량 내부 통신망이 발달함에 따라 비 기계적 요소에 의한 차량의 안전성 저해 가능성이 높아진 상황이며 이러한 기술적 변화에 따라 차량에 탑재되는 전기전자시스템의 안전성을 확보하기 위한 새로운 개발 패러다임으로 기능안전(functional safety) 개념이 차량 전자제어 시스템 개발에 필수적으로 적용되고 있다.

기능안전 모표준인 IEC 61508 을 근간으로 ISO TC22/SC32/WG8 에서 2011 년 11 월 ISO 26262 를 제정하였으며 현재 2018 년 완성을 목표로 2 판 개정 작업이 진행되고 있다. ISO 26262 는 강제법규는 아니지만 전세계 완성차 제조사(OEM)와 부품사가 참여하여 개발되었으며 전 세계 주요 OEM 은 기능안전을 확보한 시스템 개발을 위해 ISO 26262 에 대한 요구사항을 포함하여 부품사에게 부품 개발을 요구하고 있다. 이를 위해 OEM 은 Safety Goal 및 기능안전 요구사항과 이들의 ASIL (Automotive Safety Integrity level, 안전등급)을 정의하고 부품사들이 이에 따라 ISO 26262 표준을 준수하여 제품을 개발하여 공급하도록 요구하고 있다.

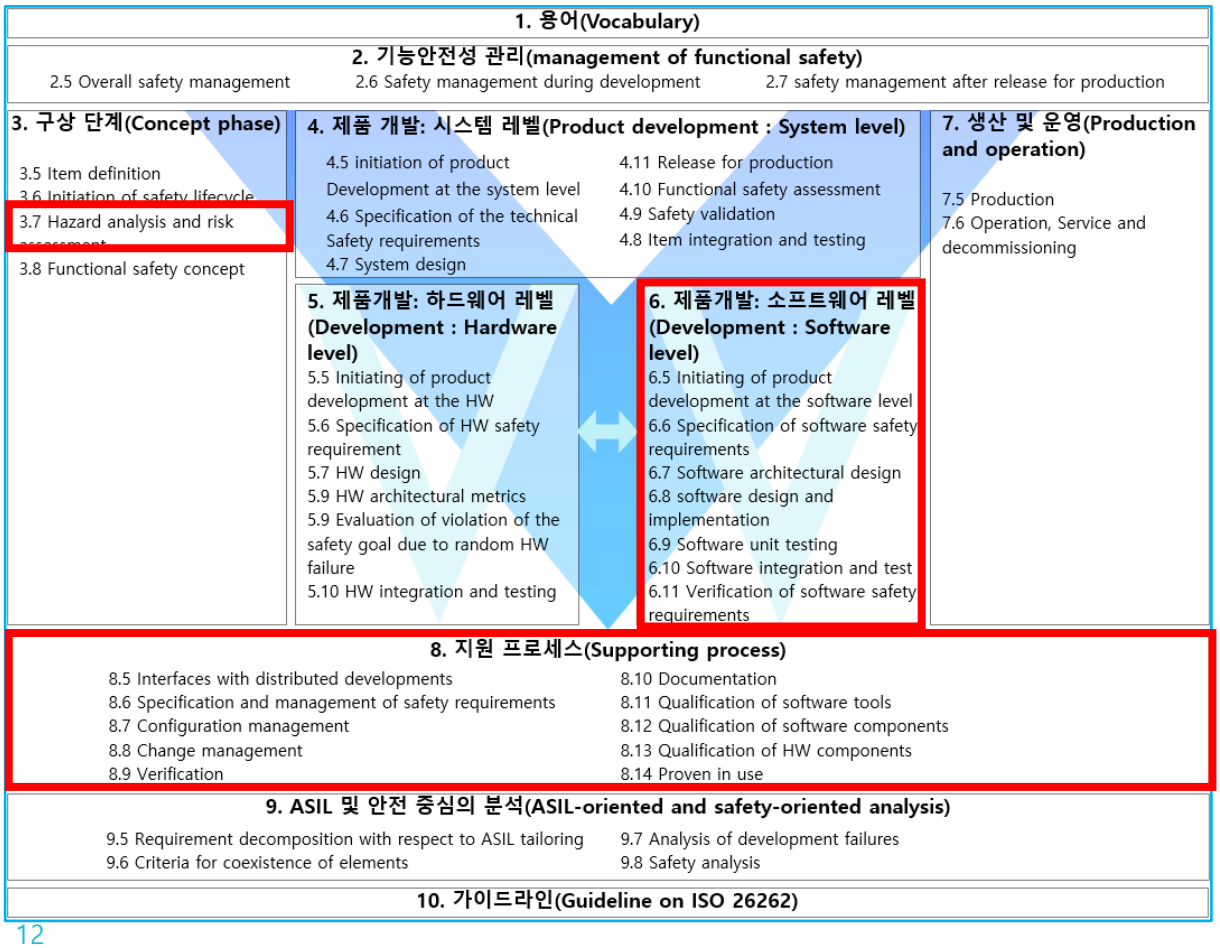
하지만 ISO 26262 는 기능안전 확보를 위한 기법과 목표 기준만을 제시하고 구체적인 도구와 방법은 제시하지 않는다. OEM 이나 대형 부품사들은 고가의 컨설팅을 통해 프로젝트를 진행하고 있지만 국내 중소 소프트웨어 기업들의 경우 자체적으로 ISO 26262 표준을 해석하는데 어려움을 호소하고 있다. 이런 문제를 극복하고 중소 소프트웨어 기업들이 ISO 26262 에 대해 이해하는 것을 돕기 위한 실례 중심의 가이드가 필요한 상황이다.

2. 목적 및 범위

ISO 26262 의 전체 프로세스 중 ASIL A, B 기준으로 HR (Highly Recommended)의 필요성이 있는 주요 항목들을 예제를 통해 쉽게 설명하여 자동차 전자제어장치 분야 국내 SW 중소기업들이 실용적으로 참고할 수 있는 가이드를 만드는 것을 목적으로 한다. 본 가이드를 이용하면 중소기업들이 적어도 ASIL B 수준의 자동차 전자제어장치를 개발할 때 차량 제조사나 부품사와 효율적으로 협상하고 의사 소통할 수 있고 궁극적으로 안전한 SW 개발을 위해 필요한 역량을 갖추도록 돕고자 한다.

본 가이드의 적용 범위는 ISO 26262 1 판의 적용 범위와 동일하게 3.5 톤 미만의 승용차용 전기전자장치를 위한 소프트웨어 개발로 한정한다. ISO 26262 2 판에는 상용차와 모터사이클도 그 대상으로 포함되지만 본 가이드는 ISO 26262 1 판을 기반으로 하고 있다. 또한 ISO 26262 전체 10 개 파트 중에서 그림 1 에 표시된 "파트 6. 소프트웨어 수준의 제품 개발"과 이를 지원하는 "8. 지원 프로세스"의 소프트웨어 관련 내용을 주된 설명 대상으로 한다. "3-7. 위험원 분석 및 리스크 평가"는 주로 완성차 회사에서 수행하기 때문에 본 가이드의 대상인 중소기업에서 직접 수행하지는 않는다. 하지만 ISO 26262 를 적용할 때 매우 중요한 절차이고 이 결과에 따라

그림 1. 본 가이드의 범위



소프트웨어 개발사 수행해야 하는 업무의 양과 성격이 크게 달라지기 때문에 본 가이드에서 추가로 수행 주체, 주요 차량 제어 시스템에 대한 일반적인 리스크 평가 결과 예제 등을 중심으로 설명한다.

본 가이드는 자동차 전자제어 시스템만을 그 대상으로 하며 그 외 항공, 철도 등 타 산업 분야의 소프트웨어에 대한 안전 가이드는 해당 산업 분야에 특화된 가이드를 참고하도록 한다. 본 가이드는 "SW 안전가이드 공통분야"와 같이 사용하는 것을 가정하고 개발되었다. 그렇기 때문에 공통분야 가이드에 자세히 설명된 내용을 본 가이드에 다시 되풀이하기 보다는 자동차 분야에 특화된 내용을 주로 서술하고 필요시 공통분야 가이드를 참고하도록 기술하였다.

본 가이드는 완성차 제조사가 판매하는 차량에 탑재되는 전자제어시스템을 대상으로 하며 그 외 아래의 경우는 그 적용 대상으로 하지 않는다.

- 애프터마켓에서 판매되는 제품(예, 블랙박스)
- 모바일폰에서 실행되는 앱 (예, 네비게이션, 음악스트리밍)
- ISO 26262-3 의 리스크 평가 결과 최소 ASIL A 등급의 안전성 요구사항을 받지 않는 제품

3. 가이드 구성

본 가이드는 4 개의 파트로 구성된다. 각 파트의 내용은 아래와 같다.

파트 I. 가이드 개요는 가이드 개발의 배경과 필요성, 목적과 적용 범위, 가이드 문서의 구성, 용어 정의로 구성된다.

파트 II. 자동차 SW 안전 현황은 1 장에서 자동차 SW 의 특징에 대해서 설명하고, 2 장에서 자동차 SW 관련 산업의 개괄적인 구조를 제시한다. 3 장은 자동차 SW 안전성 확보 실패로 인한 사고 사례를 제시하고 4 장은 ISO 26262, MISRA C, A-SPICE 등 자동차 SW 와 안전 관련 표준 동향을 설명한다. 5 장은 국내외의 자동차 SW 관련 법제도와 규제에 대한 분석 결과를 설명하고 마지막 6 장은 국내 자동차 SW 기업들의 대응 현황을 분석한다.

파트 III. 자동차 SW 안전가이드는 1 장에서 V 수명주기에 기반한 ISO 26262 의 전체 구조와 그 중에 "파트 6. 소프트웨어 수준의 제품 개발"과 이를 지원하는 "8. 지원 프로세스"에서 수행해야 하는 활동들을 예제를 통해 설명한다. 그 후 ISO 26262 파트 6 의 전체 진행 절차를 차량전원 인터페이스를 타겟 시스템으로 하여 산출물 중심으로 설명한다.

파트 IV. 부록은 용어집을 포함한다.

4. 관련 문서

- ISO 26262-1:2011 Road vehicles -- Functional safety -- Part 1: Vocabulary
- ISO 26262-2:2011 Road vehicles -- Functional safety -- Part 2: Management of functional safety
- ISO 26262-3:2011 Road vehicles -- Functional safety -- Part 3: Concept phase
- ISO 26262-4:2011 Road vehicles -- Functional safety -- Part 4: Product development at the system level
- ISO 26262-5:2011 Road vehicles -- Functional safety -- Part 5: Product development at the hardware level
- ISO 26262-6:2011 Road vehicles -- Functional safety -- Part 6: Product development at the software level
- ISO 26262-7:2011 Road vehicles -- Functional safety -- Part 7: Production and operation
- ISO 26262-8:2011 Road vehicles -- Functional safety -- Part 8: Supporting processes
- ISO 26262-9:2011 Road vehicles -- Functional safety -- Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses
- ISO 26262-10:2011 Road vehicles -- Functional safety -- Part 10: Guideline on ISO 26262
- KS R ISO 26262-1:2012 도로 차량 – 기능 안전 – 제 1 부: 용어
- KS R ISO 26262-2:2012 도로 차량 – 기능 안전 – 제 2 부: 기능 안전 관리
- KS R ISO 26262-3:2012 도로 차량 – 기능 안전 – 제 3 부: 개념 단계
- KS R ISO 26262-4:2012 도로 차량 – 기능 안전 – 제 4 부: 시스템 수준의 제품 개발
- KS R ISO 26262-5:2012 도로 차량 – 기능 안전 – 제 5 부: 하드웨어 수준의 제품 개발
- KS R ISO 26262-6:2012 도로 차량 – 기능 안전 – 제 6 부: 소프트웨어 수준의 제품 개발
- KS R ISO 26262-7:2012 도로 차량 – 기능 안전 – 제 7 부: 생산 및 운영
- KS R ISO 26262-8:2012 도로 차량 – 기능 안전 – 제 8 부: 지원 프로세스
- KS R ISO 26262-9:2012 도로 차량 – 기능 안전 – 제 9 부: 자동차 안전무결성 수준(ASIL) 및 안전 기반 분석

- KS R ISO 26262-10:2012 도로 차량 – 기능 안전 – 제 10 부: KS R ISO 26262 에 대한 지침
- IEC 61508-1:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements
- IEC 61508-2:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems
- IEC 61508-3:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 3: Part 3: Software requirements
- IEC 61508-4:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 4: Definitions and abbreviations
- IEC 61508-5:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 5: Part 5: Examples of methods for the determination of safety integrity levels
- IEC 61508-6:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 6: Guidelines on the application of IEC 61508-2 and IEC 61508-3
- IEC 61508-7:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 7: Part 7: Overview of techniques and measures
- ISO/IEC/IEEE 29119-1:2013 Software and systems engineering -- Software testing -- Part 1: Concepts and definitions
- ISO/IEC/IEEE 29119-2:2013 Software and systems engineering -- Software testing -- Part 2: Test processes
- ISO/IEC/IEEE 29119-3:2013 Software and systems engineering -- Software testing -- Part 3: Test documentation
- ISO/IEC/IEEE 29119-4:2015 Software and systems engineering -- Software testing -- Part 4: Test techniques
- ISO/IEC/IEEE 29119-5:2016 Software and systems engineering -- Software testing -- Part 5: Keyword-Driven Testing
- SW 안전가이드 공통 분야
- SW 안전가이드 의료기기 분야

• SW 안전가이드 철도 분야

PART 2

자동차 SW 안전 현황

1 자동차 SW 개요

2 자동차 SW 관련 산업구조

3 자동차 SW 안전의 중요성

4 자동차 SW 안전 표준 동향

5 자동차 SW 법제도와 규제 분석

6 자동차 SW 국내 기업 현황

SOFTWARE

상상을 현실로,
소프트웨어가 미래다

II. 자동차 SW 안전 현황

1. 자동차 SW 개요

1.1 자동차 SW 특징

자동차는 처음에는 순수한 기계 장치로 탄생했지만, 현재 자동차 회사들은 기계 기술로 해결하지 못하는 안전, 환경 규제와 사용자 편의성 제공을 위하여 전기전자 기술과 소프트웨어 기술을 적극적으로 사용하고 있다. 정밀한 엔진 제어 소프트웨어 없이는 최근의 목표 연비와 환경 규제를 만족시킬 수 없다. 차선 유지 보조 (Lane Keeping Assist) 시스템이나 자동 긴급 제동 (Autonomous Emergency Braking) 시스템과 같은 첨단 운전자 보조 (Advanced Driver Assistance) 시스템은 모두 소프트웨어로 구현된다.

자동차 소프트웨어는 일반 패키지 소프트웨어와 달리 제어기에 내장되어 있기 때문에 부품의 일부로 동작하고 사용자의 눈에는 보이지 않는 특징을 가진다. 그렇기 때문에 소프트웨어의 정상 동작 여부를 확인하기 어렵고 소프트웨어 검증에 많은 노력과 시간이 필요하다. 또한 고속으로 운행하는 자동차를 제어하는 제어 소프트웨어의 경우 소프트웨어의 오동작이 치명적인 인명 손실로 이어질 수 있기 때문에 자동차 소프트웨어는 안전성이 매우 중요하다. 직접 자동차의 움직임에 영향을 주지 않는 네비게이션 등의 인포테인먼트 소프트웨어의 경우도 운전자와 상호작용 과정에서 간접적으로 운전자의 운전 행위를 방해할 수 있기 때문에 안전을 고려한 세심한 설계가 필요하다.

산업 구조 측면에서 자동차 소프트웨어는 완성차 회사 (OEM) 혹은 부품사 (Tier-N)의 발주에 의해서 맞춤형으로 개발되는 경우가 대부분이다. 비슷한 기능을 하는 소프트웨어인 경우에도 차종과 연식별로 상이한 요구사항을 가지고 있기 때문에 소프트웨어 개발사는 다양한 요구사항 변화에 유연하게 대응해야 한다. 그렇기 때문에 소프트웨어 개발사는 요구사항을 정교하게 관리하는 기술을 가져야 하며 동시에 모든 요구사항 변화를 수용하면서 안전성을 확보하는 어려운 문제를 해결해야 한다.

소프트웨어 개발자 관점에서 자동차 소프트웨어는 다른 소프트웨어 분야에 비해 진입장벽이 매우 높은 분야이다. 개발 도구 확보와 개발 환경 구축이 비교적 쉬운 다른 산업 도메인에 비해 자동차 소프트웨어는 고가의 개발 도구와 개발 환경을 구축해야 하기 때문에 개별 개발자가 쉽게 진출하기가 어려운 분야이다. 개발 방법론 측면에서도 소스 코드를 개발자가 직접 작성하지 않고 다이어그램 형태의 모델로부터 자동 생성해서 사용하는 MBD (Model-Based Design) 방법을 많이 사용한다. 또한 고속으로 움직이는 자동차와 연결되어 동작하기 때문에 디버거를 이용해

프로그램의 동작을 멈추고 디버깅하는 방식을 사용할 수 없고 대신에 소프트웨어의 내부 변수들의 값을 실시간으로 모니터링하고 조정하는 Calibration & Measurement 기반의 평가 방식이 발달했다. 특히 안전성 검증을 위해 테스트와 평가 조직이 강하고 전체적인 제품 개발 라이프사이클이 최소 3-5 년 이상으로 매우 길다. 이와 같은 특징들은 자동차 소프트웨어 개발을 대단히 독특한 분야로 만들고 있다.

1.2 자동차 SW 분류

자동차에 탑재되는 소프트웨어는 크게 (1) IVI (In-Vehicle Infotainment) 시스템을 위한 소프트웨어와 (2) 차량제어 시스템을 위한 소프트웨어로 구분할 수 있다. IVI 시스템은 오디오, 비디오와 같은 멀티미디어 기능과 GPS 네비게이션 등을 포함한다. 차량제어 시스템은 엔진, 조향장치, 제동장치 등 자동차의 움직임을 제어하는 기계 부품들과 연결되어 동작한다. 일반적으로 IVI 시스템보다는 차량제어 시스템이 더 높은 수준의 안전 요구사항을 필요로 한다. 예를 들어 멀티미디어 기능의 경우 상대적으로 낮은 안전 요구사항을 가지는 반면에 인스트루먼트 클러스터 (Instrument Cluster)의 경우 더 높은 안전성이 요구되고 제동 장치를 제어하는 제어기는 이보다 훨씬 높은 안전성이 요구된다. 이러한 제어기에 탑재되는 소프트웨어를 개발할 때는 가장 높은 수준의 안전성을 보장하기 위해 최선의 노력을 다해야 한다.

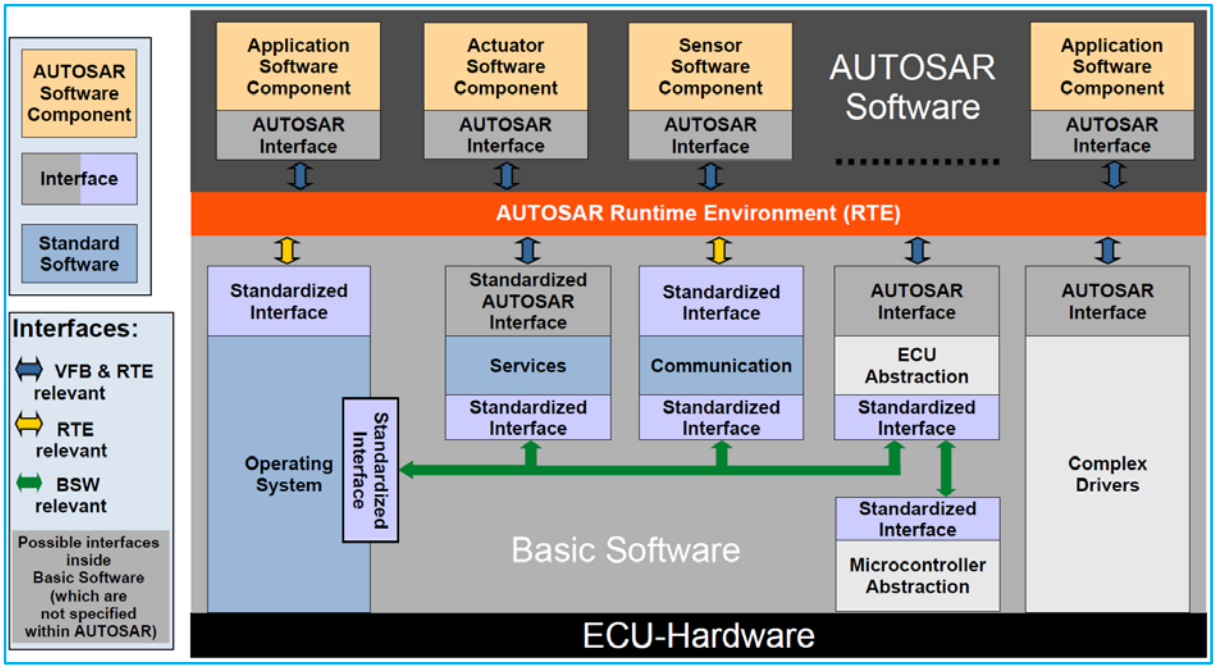
차량 제어 시스템은 (1) 파워트레인(Powertrain) 도메인, (2) 샤시(Chassis) 도메인, (3) 바디(Body) 도메인으로 다시 크게 분류할 수 있다. 파워트레인 도메인은 동력을 생성하고 이를 바퀴에 전달하는 역할을 하며 엔진과 변속기가 여기에 속한다. 샤시 도메인은 조향, 제동, 현가(서스펜션)에 대한 제어 기능을 포함한다. 바디 도메인은 도어, 윈도우, 와이퍼 등의 편의기능을 포함한다. 파워트레인 도메인은 엔진을 제어하는 컴퓨터인 EMS (Engine Management System)와 자동변속기를 제어하는 컴퓨터인 TMS (Transmission Management System)를 중요한 구성 요소로 포함한다. 특히 EMS 의 경우 환경 규제 대응을 위해 이미 1980 년대부터 자동차 안에 탑재되기 시작했기 때문에 자동차 내부에 탑재된 가장 역사가 긴 컴퓨터 시스템이라고 할 수 있다. 샤시 도메인의 경우 차량의 횡방향 안정성을 강화하는 ESC (Electronic Stability Control) 시스템이 대표적이다. 이 시스템은 차량이 코너에 진입했을 때 오버스티어 혹은 언더스티어가 발생하여 원하는 곡률로 회전이 되지 않는 경우 개별 바퀴의 브레이크를 독립 제어하여 횡방향 안정성을 확보하는 시스템이다. 샤시 제어 기능들은 최근에는 자동 조향 제어 시스템이나 스마트 크루즈 컨트롤(Smart Cruise Control)과 같이 적극적으로 제어에 개입하여 운전자를 보조하는 첨단 운전자 보조 시스템으로 발전하고 있다.

1.3 자동차 SW 구조

자동차 소프트웨어 중에서 AVN (Audio, Video, and Navigation) 시스템의 경우 스마트폰의 모바일 응용 소프트웨어와 유사한 구조를 가진다. 하지만 차량 제어 소프트웨어의 경우 운영체제부터 응용 소프트웨어까지의 구조가 일반적인 모바일 혹은 PC 소프트웨어와는 전 다른 구조를 가지고 있다. 차량 제어 소프트웨어의 구조는 보쉬(Robert Bosch GmbH), 컨티넨탈(Continental AG)과 같은 독일 부품사들의 인하우스 (In-House) 연구를 통해 오래전부터 회사별로 독자적으로 발전되어 왔다. 이러한 중복 투자를 줄이기 위한 목적으로 2002 년에는 독일을 중심으로 BMW, 보쉬, 다임러크라이슬러, 폭스바겐 등의 회사들이 모여서 자동차 제어 소프트웨어의 구조를 표준화는 방안을 논의하는데 이와 같은 노력을 통해 2005 년 AUTOSAR (AUTomotive Open System Architecture)라는 자동차 제어 소프트웨어 개방형 표준이 만들어진다. AUTOSAR 표준의 목적은 자동차 소프트웨어의 재사용성을 높이고 플랫폼 소프트웨어에 투입되는 연구개발 투자를 줄이고 대신에 응용 소프트웨어에 역량을 집중하도록 하는데 있다.

AUTOSAR 표준의 기본 구조는 그림 2 와 같다. 하드웨어 위에 표준화된 BSW (Basic Software) 계층과 RTE (Runtime Environment) 계층을 올려서 그 위에서 동작하는 ASW (Application Software)는 AUTOSAR 표준에 맞춰서 개발되면 하드웨어가 변경되더라도 호환성을 유지할 수 있다. 또한 ASW 수준의 소프트웨어 컴포넌트 간의 통신 방식도 표준화되어 있기 때문에 소프트웨어 컴포넌트도 독립적으로 재사용할 수 있다.

그림 2. AUTOSAR 소프트웨어 구조 (Source: AUTOSAR Consortium)



AUTOSAR 는 실행 아키텍처 외에도 ASW 설계 방법도 표준화하고 있다. AUTOSAR 의 ASW 는

가상의 통신 방식인 VFB (Virtual Function Bus)로 연결된 여러 개의 소프트웨어 컴포넌트의 집합으로 설계된다. 소프트웨어 컴포넌트들은 구현시에는 같은 ECU 에 배치될 수도 있고 서로 다른 ECU 에 배치될 수도 있다. 각 경우에 대해 VFB 는 자동 생성되는 RTE 코드로 구현된다. 같은 ECU 내에서의 통신은 메모리 복사로 코드가 생성되고 다른 ECU 사이의 통신은 CAN 과 같은 네트워크 통신 코드로 자동 생성된다.

AUTOSAR 의 운영체제는 2005 년에 제정된 OSEK/VDX 운영체제에 기반하며 현재는 AUTOSAR OS 라고 명명된다. AUTOSAR OS 는 우선순위 기반 스케쥴링을 이용하는 실시간 운영체제이다. 고정우선순위 스케쥴링을 지원하며 우선순위 역전 현상과 데드락을 방지하기 위하여 PCP (Priority Ceiling Protocol)을 사용한다. 모든 소프트웨어 객체(태스크, 인터럽트 서비스 루틴 등)는 모두 정적으로 생성되기 때문에 시스템 동작 중에 새로운 소프트웨어 객체가 생성되지 않는다. OSEK/VDX 표준이 AUTOSAR OS 로 확장되는 과정에서 소프트웨어 안전성 강화를 위해 타이밍 보호, 메모리 보호 기능 등이 추가되었다.

AUTOSAR 소프트웨어 구조는 전통적인 제어기의 신호처리와 제어 알고리즘을 실시간으로 실행하는데 최적화되어 있지만 최근의 자율주행 시스템이나 카메라 기반의 안전기능을 구현하는데는 적합하지 않다. AUTOSAR 소프트웨어 구조는 영상처리, 인공지능, Connectivity, OTA (Over-The-Air) 업데이트 등 최근에 빠르게 발전하고 있는 기술 분야에 대한 지원을 하지 못하기 때문이다. 이에 대한 대응으로 리눅스와 같은 POSIX 표준에 따르는 운영체제를 가정한 새로운 AUTOSAR 표준인 Adaptive AUTOSAR 표준이 2017 년에 발표되었다. 2018 년 현재 아직 Adaptive AUTOSAR 상용 제품은 판매되고 있지 않으나 많은 AUTOSAR 소프트웨어 벤더들이 제품을 개발 중이다.

2. 자동차 SW 관련 산업구조

자동차 산업은 완성차 회사(OEM)를 정점으로 부품사(Tier-N)들이 협력하는 피라미드 구조를 이루고 있다. 이 구조 안에서 완성차 회사에 직접 부품을 공급하는 회사를 Tier-1 이라 하고 Tier-1 에 부품을 공급하는 회사를 Tier-2 라고 한다. 결과적으로 그림 3 과 같은 형태의 산업 구조를 가진다.

그림 3. 피라미드 형태의 자동차 산업 구조

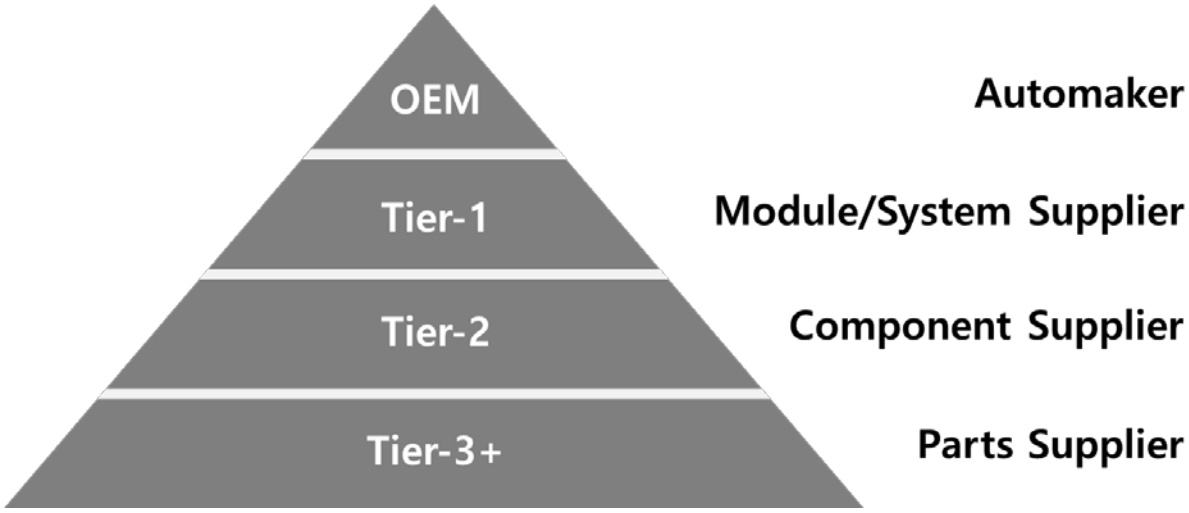


그림 4. 세계 10 대 부품사 (Source: Automotive News)

1	**Robert Bosch GmbH** (49) 711-811-0; bosch.com
2	**Denso Corp.** (81) 566-25-5511; denso.com
3	**Magna International Inc.** (905) 726-2462; magna.com
4	**Continental AG** (49) 511-938-01; conti-online.com
5	**ZF Friedrichshafen AG** (49) 7541-77-0; zf.com
6	**Aisin Seiki Co.** (81) 566-24-8441; aisin.co.jp
7	**Hyundai Mobis** (82) 2-2018-5114; mobis.co.kr
8	**Lear Corp.** (248) 447-1500; lear.com
9	**Valeo SA** (33) 1-40-55-20-20; valeo.com
10	**Faurecia** (33) 1-72-36-70-00; faurecia.com

2017 년 기준 세계 10 대 부품사는 그림 4 와 같다. 여기에는 독일 3 개사, 일본 2 개사, 프랑스 2 개사, 캐나다 1 개사, 미국 1 개사, 한국 1 개사가 포함되어 있다.

자동차 소프트웨어 기술은 지금까지는 부품사들을 중심으로 발전해 왔다. 특히 독일의 거대 부품사들은 오랜 기간의 자동차 제어 소프트웨어 개발 경험을 바탕으로 AUTOSAR, ISO 26262, Automotive SPICE 등의 자동차 소프트웨어 관련 표준에 막대한 영향력을 행사하고 있으며 이를 바탕으로 계속적으로 입지를 강화하고 있다. 특히 자동차 소프트웨어 관련 운영체제나 소프트웨어 개발 도구를 개발하는 회사들도 상당수는 독일 부품사의 자회사이거나 스핀오프한 회사인 경우가 많다. 근래에는 반대로 전문 소프트웨어 개발사를 대규모 부품사들이

인수하는 등 소프트웨어 기술을 오랫동안 부품사들이 주도하고 있다.

하지만 근래에는 전통적인 부품 공급 사슬 바깥에 존재하는 기업들이 자동차 소프트웨어의 핵심 기술을 개발하여 자동차 분야에 진출하는 일이 많이 생기고 있다. 대표적인 기업은 구글이다. 구글은 2009 년부터 자율주행 자동차 연구를 시작하여 2016 년에는 웨이모로 분사했으며 웨이모는 2017 년 기준 80 조원의 기업 가치를 평가받고 있다. 차량 공유 서비스를 제공하는 우버의 경우도 자율주행 기술 개발에 많은 투자를 하고 있다. 그 외에도 중국의 바이두 등 IT 기업들도 자율주행 기술 개발에 뛰어들고 있다.

자동차 소프트웨어는 부품사와 그 협력사에 의해서 개발되어 완성차 회사에 부품, 모듈, 시스템의 형태로 납품되는 경우가 대부분이다. 한번 차량이 판매되면 소프트웨어에 치명적인 하자가 발견되지 않는다면 차량 폐차시까지 단일 소프트웨어가 계속 사용되고 차량 사용자는 소프트웨어의 존재를 알 수도 없다. 하지만 최근에는 자동차 소프트웨어 자체가 독립적인 기능으로 고객에게 가시적으로 제공되는 경우도 생기고 있다. 대표적인 것이 테슬라의 오토파일럿 기능이다. 오토파일럿은 미국자동차공학회 기준 레벨 2 수준의 운전자 보조 시스템이지만 대중에게 자율주행 소프트웨어로 인지되고 있는 기능이다. 이 소프트웨어는 OTA (Over The Air)로 소프트웨어가 차량에 자동으로 다운로드 되어 업데이트 된다. 그렇기 때문에 차량 소유자는 지속적으로 오토파일럿 소프트웨어 업데이트를 받게 되고 이를 통해 계속 향상된 기능을 제공받을 수 있다. 소프트웨어가 완성차 회사로부터 고객에게 직접 제공되는 이와 같은 방식은 자동차 산업에서는 완전히 새로운 개념이다.

3. 자동차 SW 안전의 중요성

3.1 도요타 ETCS 리콜 사태

2009 년 8 월 28 일 캘리포니아 샌디에고에서 2009 년형 Lexus ES350 의 급발진 의심 사고로 일가족 4 명이 사망한다 (그림 5[1]). 이 사고의 원인으로 처음에는 운전석 바닥 매트, 그리고 가속 페달이 지목되어 리콜된다. 하지만 소프트웨어 결함이 의심되면서 때문에 미국 도로교통안전국 NHTSA 와 NASA 가 공동으로 10 개월간 조사하고 그 결과가 2011 년 2 월 공개된다. 이 보고서는 사고의 원인은 기계적인 것이며 전자제어 시스템의 결함은 찾을 수 없다고 결론을 내린다.[2]

그림 5. Lexus ES350 사고 차량 (Source: NHTSA)

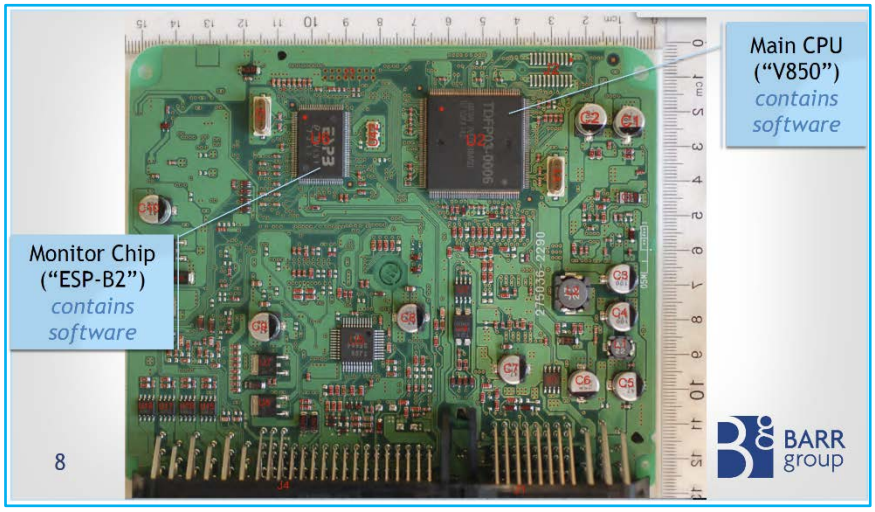
하지만 이보다 먼저 발생한 2007 년 9 월 오클라호마 사고의 경우 2013 년 10 월의 민사 소송 진행 과정에서 도요타 자동차의 ETCS (Electronic Throttle Control System) (그림 6) 소프트웨어 결함 가능성이 있는 것으로 밝혀진다. Barr 그룹의 Michael Barr 와 카네기멜론대학의 Phil Koopman 교수 등 전문가 증인들에 의해서 도요타 ETCS 의 소프트웨어가 분석되고 이들은 이

[1] REPORT: Vehicle and Crash Site Inspection of 2009 Lexus ES-350, VIN JTHBJ46G792282025, NHTSA

[2] NHTSA Report on Toyota Unintended Acceleration Investigation, NASA Report on Toyota Unintended Acceleration Investigation

소프트웨어에 급발진을 일으킬 수 있는 근본적인 결함이 있음을 밝히고 단지 메모리의 특정 비트 값을 변경함으로 급발진 상황이 재현되는 것을 실험으로 증명했다.

그림 6. 도요타 ETCS 시스템 (Source: BARR group)



이들이 발견한 ETCS 소프트웨어의 대표적인 결함은 다음과 같다.

- Failsafe 아키텍처 자체 결함
- 11,000 개 이상의 전역변수 사용
- 소스코드의 Cyclomatic Complexity Measure 가 검증불능 수준인 50 이상인 함수 67 개
- 재귀호출 사용으로 인한 스택 오버플로우 가능성
- 비표준 운영체제 사용
- MISRA C 코딩 가이드라인 미적용

안전성이 중요한 쓰로틀 제어 소프트웨어의 품질이 형편없는 수준이고 안전성을 보장하는 소프트웨어 구조도 결함이 있는 등 도요타의 소프트웨어 엔지니어링 프로세스는 중대한 결함을 가지고 있는 것으로 파악되었다.

3.2 자율주행 자동차 사고 사례

그림 8 테슬라 사망사고 피해차량 (Source: NTSA Preliminary Report, Highway

그림 7. 충돌한 화물 트레일러 (Source: NTSA Preliminary Report, Highway HWY16FH018)

부분 자율주행 자동차가 상용화되고 자율주행 시험 차량들이 공공도로에서 실증 테스트를 진행하면서 자율주행 자동차에 의한 사고 사례도 생겨나고 있다. 그림 7 은 2015 년 발생한 테슬라 오토파일럿 오작동에 의한 사망사고 피해 차량이다. 그림 8 의 트레일러의 측면을 장애물로 인식하지 못하고 그대로 주행하여 측면 하단을 통과하면서 운전자가 사망했다. 카메라를 이용한 인지 시스템을 이용한 오토파일럿은 무늬가 없는 트레일러 측면을 제대로 인지하지 못한 것으로 알려져 있다.

사고에 대한 NHTSA 의 최종 조사 결과가 2017 년 1 월 발표되었다(NHTSA PE 16-007). NHTSA 는 Tesla 의 Autopilot 이나 AEB 기능이 설계된 대로 동작하지 않았다는 증거를 찾지 못했으며 Autopilot 은 완전 자율주행 기능이 아니라 운전자 보조 기능이기 때문에 최종 책임은 운전자에게 있다고 결론을 내렸다. 실제 Autopilot 이 실행 중일 때 운전자가 핸들에서 손을 떼고 일정 시간이 흐르면 차례대로 시각적 청각적 경고가 발생한다. 그래도 운전자가 핸들을 잡지 않으면 차선을 유지한 상태에서 자동차의 속도를 서서히 줄이게 된다. 하지만 아주 가끔씩 핸들에 손을 올리는 방식으로 이 제한을 피해갈 수 있었다. Tesla 는 이 사고 이후 strike out 시스템을 만들어 안전 규칙을 지키지 않으면 아예 그 운행 사이클 동안 autopilot 이 작동 불가능한 더 강한 페널티를 주도록 만들었다.

3.3 국내 자동차 SW 안전성 의심 사례

국내에서도 자동차 SW 의 결함으로 인해 안전에 위협을 받는 사고가 발생하고 있다. 2018 년 1 월에 서비스센터에서 소프트웨어 업데이트를 받은 A 사 차량이 같은 해 4 월 주행 중 시동이 꺼져 전복되는 사고가 뉴스를 통해 보고되었다. 해당 차주는 소프트웨어 업데이트로 인해 차량의

안전성에 문제가 생겼다고 주장하고 있다.[3] 2018 년 여름에 시작된 B 사 차량 연쇄 화재 사고의 경우도 현재 원인 조사가 진행중이지만 전문가들을 통해 소프트웨어 결함이 아닌지 의심받고 있다. 디젤엔진을 위한 배출가스 재순환 장치(EGR)의 밸브 조작이 소프트웨어로 이루어지는데 이를 제어하는 소프트웨어의 결함이 있다는 것이 의심되고 있다.[4] 특히 민관합동조사단의 조사 결과 EGR 밸브의 문제인 것으로 결론이 나면서 계속적으로 소프트웨어 결함에 대한 조사가 진행되고 있다.[5] 해외 C 사의 경우도 최근 차량에서 인포테인먼트 시스템에서 많은 오류가 발생하여 큰 문제가 되고 있다.[6] 특히 후방주차 카메라, 센서 등과 같이 안전 관련 기능들이 소프트웨어 결함으로 동작하지 않으면서 사고 혹은 위험 사례들이 보고되고 있다. 국내 제조사인 D 사의 경우에도 2016 년 MDPS (Motor Driven Power Steering) 시스템의 소프트웨어 결함이 의심되었다.[7]

[3] https://news.sbs.co.kr/news/endPage.do?news_id=N1004873267

[4] https://news.joins.com/article/22871386

[5] http://news.khan.co.kr/kh_news/khan_art_view.html?art_id=201811072133025

[6] http://www.consumuch.com/news/articleView.html?idxno=41820

[7] http://news.khan.co.kr/kh_news/khan_art_view.html?art_id=201610092138005

4. 자동차 SW 안전 표준 동향

4.1 ISO 26262

ISO 26262 는 2011 년에 발표된 자동차 전기전자 시스템을 위한 기능안전 표준이다. 모표준인 IEC 61508 에서 파생된 표준으로 3.5 톤 이하 양산 승용차만을 대상으로 하고 있다. 전체 10 개 파트로 구분되어 있으며 여기에는 용어 정의부터 시작하여 안전 분석, 하드웨어, 소프트웨어 개발, 양산 후 안전 관리까지 제품의 전체 수명주기에 대해서 기능안전성을 확보하기 위한 표준을 제시한다.

현재는 완성차 회사가 부품사에 전기전자 시스템이 포함된 부품, 모듈, 시스템을 발주할 때 ISO 26262 준수를 의무화하는 것이 일반적이다. 보통 완성차 회사에서 안전분석을 하여 어느 정도의 안전 등급을 유지해야 하는지를 결정하고 이에 따라 부품사는 해당 등급별로 ISO 26262 에 명시된 의무사항들을 수행하고 그 결과를 완성차 회사에 제출해야 한다. 완성차 회사는 자동차의 제조사로서 제조물 책임법의 책임을 지는데 ISO 26262 를 따랐다는 사실이 제조물 책임법의 면책 조항이 될 수 있기 때문에 완성차 회사는 반드시 ISO 26262 를 준수하여 제품을 개발한다.

2011 년 1 판이 발표된 이후 7 년의 시간이 지났기 때문에 그동안의 기술 변화를 반영하여 2018 년 발표를 목표로 ISO 26262 2 판이 준비중이다. ISO 26262 2 판에는 기존의 3.5 톤 이하 승용차에 대해서만 적용되던 범위가 상용차 및 오토바이까지 확대된다. 이에 따라 Part1 에 PTO, Trailer, Body Builder 등 상용차 전용 용어를 추가하고, Part3 에는 트럭 및 버스의 상황에 맞는 위험분석 방법이 추가된다. 또한 Part 11 에 반도체 설계 분야에 대한 내용이 새롭게 추가되었으며 칩 성능과 패키지 형태, 동작 시간, 온도 환경에 따라 반도체가 고장 날 확률이 어느 정도인지 계산하는 고장율 예측에 대한 내용이 포함된다. ISO 26262 2 판의 주요 개정 내용을 요약하면 아래 표 1 과 같다.

표 1. ISO 26262 2 판의 변경 및 추가 내용

파트	변경/추가 내용
Part 1	• 트럭 및 버스를 포함한 차량 범위 관련 확대 적용 용어 추가(Base vehicle, body builder, tractor, trailer 등) • 반도체 관련 용어(Base failure rate, Processing Element, Multi-core) 추가 • 엘리먼트에 결함을 주입하여 결함의 영향을 평가하는 결함 주입(Fault Injection) 기법 용어 추가 • Fault Tolerant Time Interval 용어를 안전 매커니즘이 작동하지 않는 경우로 한정하여 재정의

Part 2	• 기능 안전에 부정적인 영향을 줄 수 있는 사이버시큐리티(Cybersecurity)에 대한 연계성 포함 (Guidance on potential interaction of functional safety with cybersecurity) • 재사용되는 엘리먼트에 대해서 영향 분석 수행하도록 함 • 확인 검토(Confirmation Review) 대상 변경 - 대상 삭제: 아이템 통합 및 시험 계획, 확인 계획, 소프트웨어 도구 평가 보고서 및 소프트웨어 도구 인정 보고서, 사용 실증 논거(Proven in use arguments) - 대상 추가: 기능 안전 개념, 기술 안전 개념, 아이템 통합 및 시험 전략, 안전 확인 명세, 안전 분석 및 종속 고장 분석
Part 3	• 트럭 및 버스 상황에 맞는 severity, exposure, controllability 정의 및 예제 추가 • 파생차종(Variance) 다양화에 따른 H&R 유의사항 명기 (Bus vehicle, configuration, operation 상황 고려 및 모든 파생 차종 조건을 고려하여 위험도 분석)
Part 4	• 기술 안전 개념 명세와 시스템 설계를 통합해 기술 안전 개념에 포함 • 트럭 및 버스의 파생 차종 다양화에 따른 대표 검증(V&V) 방법 포함 • 기능 안전 평가(Functional safety assessment)에 대한 내용을 삭제하고 Part 2 로 이동 • DFA (Dependent Failure Analysis)를 일관성 유지를 위해 Part 9 으로 이동
Part 5	• 하드웨어 소자에 대한 고장률/고장분포 산출 시 참고할 수 있는 표준으로 NPRD-2016, RIAC FMD-2016, SN29500, FIDES, IEC 61709 등 추가 • 하나의 아이템이 여러 가지 엘리먼트로 구성되는 경우 SPFM (Single Point Fault Metric)/LFM (Latent Fault Metric) 목표 값을 각 엘리먼트에 할당하는 경우의 예제와 산술식을 예제로 추가 • Scaling Factor 에 대한 요구사항 제거 • Diagnosis Coverage 에 대한 Proper Rationale 개념 정의 • Non-Single System 에 대한 PMHF (Probability Metric for Random Hardware Failures) 목표 값 설정 기준 수립
Part 6	• Multi-Core 시스템 상에서 Concurrent Software 실행 시 발생할 수 있는 고려 사항 추가 • 모델 기반 개발 방법론의 모델링 가이드라인으로 MISRA AC 추가 및 모델 기반 개발의 고려 사항 추가 • 소프트웨어 아키텍처 설계 시 시스템적 고장을 회피하기 위한 특성 및 설계 원칙 추가 • 단위 검증 방법으로 Pair-Programming 방법 추가 • 임베디드 소프트웨어의 시험 환경으로 HIL 이 모든 ASIL 에서 ++로

	변경되고 Vehicles 는 ASIL A, B 에서 기존 ++에서 +로 변경
Part 7	• 트럭 및 버스의 bodybuilding, re-building, multi-stage build 내용 추가
Part 8	• 분산 개발 계획의 사전 준비 사항에서 DIA 를 삭제하고 RFQ 유지 • 소프트웨어 도구 신뢰성 평가 수준 및 도구 인정 방법 변경
Part 9	• 공존 기준, 분할, 의존 고장 설명 추가 • 종속 고장 분석의 ASIL 별 권고되는 방법 정의
Part 10	• FTTI, PMHF 내용 일부 변경
Part 11	• 반도체에 ISO 26262 적용 가이드라인으로 신규 파트 추가(Guideline on application of ISO 26262 to semiconductors) - 기본 고장율(Base Failure Rate) 예측, 종속 고장 분석 방법, 결함 주입 시험, 결함 모델, 안전 분석 고려 사항 등
Part 12	• 오토바이에 ISO 26262 적용 가이드라인으로 신규 파트 추가 (Adaptation of ISO 26262 for motorcycles) - 안전 문화, 확인 수단, 위험원 분석 및 리스크 평가, 통합 및 시험, 안전 확인 등

4.2 MISRA C

MISRA C 표준은 안전성이 중요한 Safety-critical 시스템을 위한 C 프로그래밍 표준 가이드이다. 영국 MISRA (Motor Industry Software Reliability Association)에서 자동차 소프트웨어의 안전성 확보를 위해 1998 년 처음 발표하고 이후 2004 년과 2012 년에 새로운 버전을 발표했다. 자동차 소프트웨어를 적용 대상으로 개발되었지만 현재는 항공, 철도, 국방 등 다른 분야에서도 많이 사용되고 있으며 자동차 소프트웨어 분야에서는 사실상의 De Facto 표준으로 사용되고 있다.

1998 년에 출판된 "Guidelines for the use of the C language in vehicle based software"가 MISRA C:1998 로 알려져 있다. 총 127 개의 룰이 있으며, 93 개의 필수(Required) 규칙과 34 개의 권고(Advisory) 규칙으로 구성되어 있다. MISRA C 의 두 번째 버전은 2004 년에 출판된 "Guidelines for the use of the C language in critical systems"이다. MISRA C:1998 이 자동차 분야에만 적용되었다면 MISRA C:2004 부터는 자동차 외에도 안전에 민감한 시스템에 일반적으로 사용될 수 있도록 변경되었다. 총 142 개의 규칙 중에서 필수 규칙이 122 개이고 권고 규칙이 20 개이다. MISRA C 의 가장 최신 버전인 MISRA C:2012 는 C99 언어 표준을 지원하기 시작했다. 전체 규칙 수도 159 개로 변경되었으며, 그 중 MISRA C:2004 에서 10 개 규칙이 제거되고 새로운 규칙이 37 개 추가되었고 설명이 강화되고 이해를 돕기 위한 코드 예제도 많이 추가되었다. 규칙의 종류도 필수(Required), 권고(Advisory)에 더해서 의무(Mandatory)가 추가되었다. 의무 규칙은 반드시 지켜야 하는 규칙이며, 필수 규칙은 이유가 있을 시에는 지키지 않아도 되지만

사유에 해당하는 공식적인 문서가 필요하고, 권고 규칙은 프로젝트 안에서 자유롭게 적용할 수 있다. 현재는 MISRA C:2004 가 가장 많이 사용되고 있지만 MISRA C:2012 로 조만간 전환될 확률이 높다. 2008 년에는 C++ 언어에 대한 MISRA C++:2008 이 발간되었으며 이 표준은 응용 소프트웨어 개발에 C++ 언어를 사용하는 Adaptive AUTOSAR 표준에 포함되어 확장되었다.

4.3 A-SPICE

Automotive SPICE (A-SPICE)는 ISO/IEC 15504 (ISO/IEC 33000 Series:2015) 및 ISO/IEC 12207 을 기반으로 자동차 임베디드 시스템 개발에 맞게 특화하여 만든 국제 표준 프로세스 참조 모델 및 평가 모델이다. 자동차에서 소프트웨어의 비중이 커져가면서 소프트웨어 개발에 대한 신뢰성을 확보하기 힘들다는 완성차 회사들의 판단으로 자동차 소프트웨어 개발 프로세스의 정립 및 평가가 필요했고 CMMI 나 SPICE (ISO/IEC15504)와 같은 기존 프로세스 모델을 도입하여 이를 해결하고자 했다. 독일 자동차 협회(VDA: Verband Der Automobilindustrie)는 라이센스 비용의 부담, 변경의 용이성 등을 고려하여 2005 년 SPICE 를 기반으로 한 A-SPICE 1.0 을 발표하였다. 이후 개정을 거듭하여 2015 년 발표된 A-SPICE 3.0 은 기존의 소프트웨어 엔지니어링만을 고려한 SPICE 의 한계를 넘어 시스템 엔지니어링 영역이 추가되었다. 시스템 엔지니어링 영역으로의 확장은 명시적으로 하드웨어 엔지니어링에 대한 모델을 제공하지는 않지만, 하드웨어 엔지니어링 영역과의 통합의 가능성을 내포하고 있다. 또한, 모든 참조와 링크 간의 추적성을 강화하여 커버리지 분석(coverage analysis), 영향도 분석(impact analysis), 요구사항 기반의 구현 상태 추적을 가능하게 하였다. 2017 년 11 월에 HIS Scope 의 명칭을 VDA Scope 으로 수정하는 등의 소규모 변경 사항들을 적용한 A-SPICE 3.1 이 발표되었다.

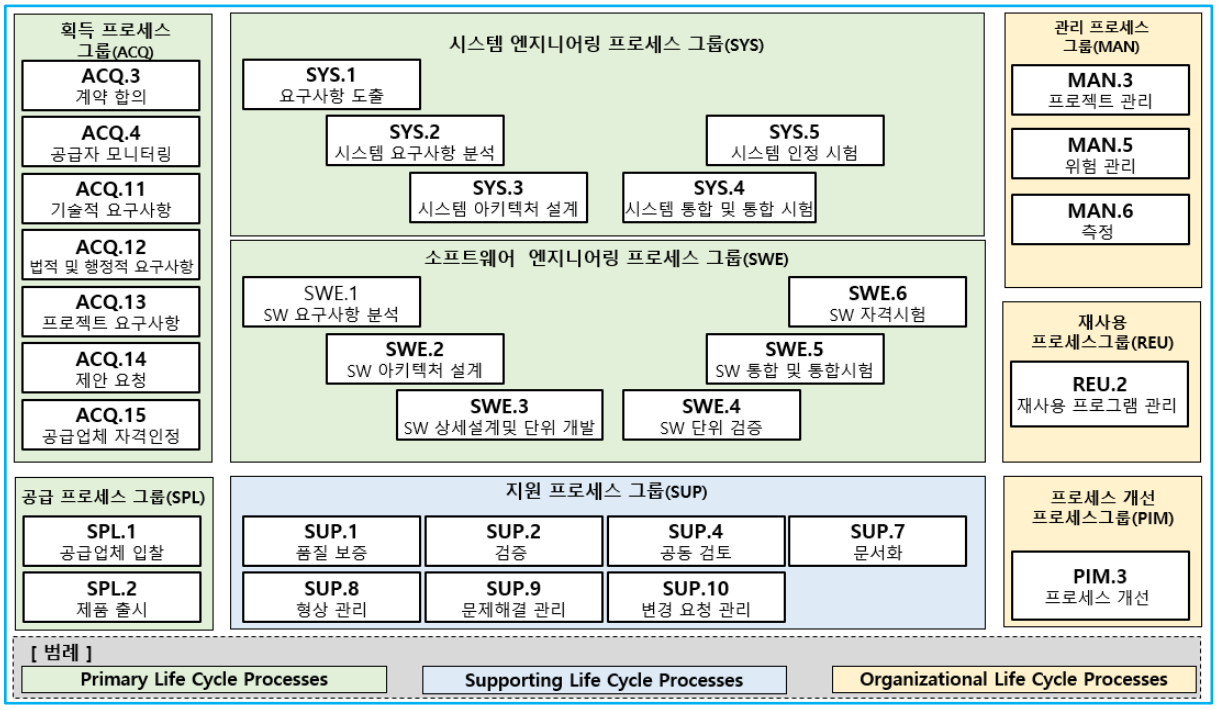
A-SPICE 는 소프트웨어 개발에서 수행해야 하는 프로세스에 대한 참조 모델(PRM: Process Reference Model)과 이를 평가할 수 있는 측정 모델(PAM: Process Assessment Model)을 제시하고 있다. 프로세스 참조 모델은 그림 9 와 같이 3 개의 라이프 사이클 카테고리, 7 개의 프로세스 군, 32 의 프로세스로 구성된다. 평가 측정 모델은 프로세스를 수행하는 능력의 정도를 측정하는 지표로 각 능력은 다음과 같은 레벨로 나누어 평가된다.

- 레벨 0(불완전한 상태): 프로세스 결과가 존재하지 않거나 성과를 내지 못하는 단계
- 레벨 1(실시됨): 프로세스가 수행되어 목적을 달성하고 있는 단계
- 레벨 2(관리됨): 프로세스와 작업산출물이 관리되고 있는 단계
- 레벨 3(확립됨): 조직을 위한 구체적인 표준 프로세스가 존재하며 조직의 표준 프로세스를 테일러링하여 활용되고 있는 단계

• 레벨 4(예측 가능함): 프로세스 수행을 정량적으로 측정하고, 과거 데이터를 분석하여 객관적인 판단을 내릴 수 있고, 사전에 정해진 범위 내에서 운영되고 있는 단계

• 레벨 5(최적화됨): 현재 및 향후 비즈니스 목적을 달성하도록 지속적으로 개선되고 있는 단계

그림 9. A-SPICE 프로세스 참조 모델 (Source: Automotive SPICE 프로세스 평가/참조모델 3.1)



많은 완성차 회사들, 특히 독일 자동차 협회를 중심으로 한 회사들은 Tier-1 부품사에게 필수적으로 A-SPICE 레벨 2-3 을 유지하도록 강하게 요구하고 있다. Tier-1 부품사가 이 레벨을 유지하기 위해서는 협력 관계에 있는 Tier-2 부품사나 Tier-3 부품사 역시 A-SPICE 의 동일한 레벨을 유지해야만 한다. 다시 말해, Tier-1 에서 Tier-3 에 이르기까지 모든 부품 업체들이 A-SPICE 를 도입해야 하는 상황이다.

국내에서도 현대 모비스, LG 전자, 나비스오토모티브시스템즈 등 많은 업체들이 A-SPICE 레벨 2 를 획득하였으며, 현대기아자동차 역시 신형 엔진 개발에 A-SPICE 를 적용하는 것으로 알려져 있다. 빠르게 A-SPICE 레벨 획득이 이루어지고 있는 만큼 우려의 목소리도 나오고 있다. 소프트웨어의 개발 과정이 지속적이고 효율적으로 이루어지도록 하기 위한 준비보다 획득 자체에 목적을 두어, 지속적으로 레벨을 유지하는 것이 어려울 수 있기 때문이다. 소프트웨어 개발 프로세스 개선을 위해서는 무엇보다 경영진의 적극적인 관심과 투자가 필요하며 구성원들의 인식 변화가 수반되어야 한다는 것을 다시 한 번 상기할 필요가 있다.

5. 자동차 SW 법제도와 규제 분석

5.1 자동차관리법

「자동차관리법」의 목적은 자동차의 등록, 안전기준, 자기인증, 제작결함 시정, 점검, 정비, 검사 및 자동차관리사업 등에 관한 사항을 정하여 자동차를 효율적으로 관리하고 자동차의 성능 및 안전을 확보하는 것이다(「자동차관리법」 제 1 조). 「자동차관리법」 제 29 조 제 1 항에 의거하여 해당 자동차가 **자동차안전기준**에 적합하지 못하면 운행하지 못하게 되어 있으며 제 29 조 제 2 항에 의거하여 자동차부품 또한 **부품안전기준**에 적합해야 한다. 관련 법령은 아래와 같다.

> **제 29 조(자동차의 구조 및 장치 등) ① 자동차는 대통령령으로 정하는 구조 및 장치가 안전운행에 필요한 성능과 기준(이하 "자동차안전기준"이라 한다)에 적합하지 아니하면 운행하지 못한다. ② 자동차에 장착되거나 사용되는 부품·장치 또는 보호장구(保護裝具)로서 대통령령으로 정하는 부품·장치 또는 보호장구(이하 "자동차부품"이라 한다)는 안전운행에 필요한 성능과 기준(이하 "부품안전기준"이라 한다)에 적합하여야 한다.**

자동차안전기준과 부품안전기준은 같은 법 제 29 조 제 4 항에 의거하여 국토교통부령인 「자동차 및 자동차부품의 성능과 기준에 관한 규칙」에 따른다. 이 중 대다수는 전자제어 장치와 무관하나 대표적으로 아래와 같은 안전 관련 전자제어 장치의 탑재를 강제하고 있다. 하지만 이러한 안전 기능을 탑재할 것을 강제할 뿐 이를 구현하는 소프트웨어에 대한 규정은 포함하고 있지 않기 때문에 이는 소프트웨어의 기능안전과는 무관하다.

- 제 12 조의 2(타이어공기압경고장치) ➔ Tire Pressure Management System
- 제 14 조의 2(차로이탈경고장치) ➔ Lane Departure Warning System
- 제 15 조의 2(자동차안정성제어장치) ➔ Electronic Stability Control
- 제 15 조의 3(비상자동제동장치) ➔ Autonomous Emergency Braking

자동차와 자동차부품에 대한 안전, 환경 규제는 국가에 따라 형식승인 제도나 자기인증 제도를 택하고 있는데 우리나라는 안전 분야의 경우 2003 년 형식승인 제도에서 자기인증 방식으로 변경하였다. 이에 대한 자세한 사항은 「자동차관리법」 제 30 조(자동차의 자기인증 등)와 제 20 조의 2(자동차부품의자기인증등)를 참고할 수 있다. 반대 환경 분야는 계속 형식승인 절차를 따르고 있다. 안전 분야 자기인증의 부족한 점을 보충하기 위하여 「자동차 관리법」 제 33 조의 2 는 같은 법 시행규칙에 의해 판매중인 자동차의 안전도를 평가하여 공표하도록 되어있다. 관련 법령은 다음과 같다.

제 33 조의 2(자동차의 안전도 평가) ① 국토교통부장관은 소비자에게 자동차의 안전도에 대한 정보를 제공하고 안전도가 높은 자동차를 제작하도록 유도하기 위하여 국토교통부령으로 정하는 바에 따라 자동차제작자등이 판매한 자동차의 안전도를 평가하여 그 결과를 공표(公表)하여야 한다. 국토 교통부령으로 정하는 바에 따라 자동차 제작자 등이 판매한 자동차의 안전도를 평가하여 그 결과를 공표하여야 한다.

이에 따라 자동차안전연구원에서 같은 법 시행규칙 제 53 조(자동차안전도평가 연간계획의 수립)와 제 54 조(자동차안전도평가의 시행 등)에 의거하여 자동차의 안전도 평가를 진행 중이다. 충돌안전성, 보행자안전성, 사고예방안전성에 대해서 각각 평가를 하며 그 중 사고예방안전성 테스트에 전방 충돌 경고 장치, 차로 이탈 경고 장치, 사각지대 감시 장치 테스트 등이 포함되어 있다. 하지만 이는 안전기능의 탑재 여부만 따지며 내부에 탑재된 소프트웨어의 개발 절차에 대해서는 평가하지 않는다. 자세한 안전도 평가결과는 자동차안전도평가 홈페이지(https://kncap.org)에서 확인할 수 있다. 다만 모든 판매 차량에 대해서 평가를 하지는 않고 있으며 신규 출시된 판매량이 많은 자동차 위주로 평가를 수행한다.

5.2 제조물책임법

ISO 26262 와 관계가 깊은 「제조물책임법」은 2000 년에 제정되었으며 2002 년부터 시행되었다. 2017 년에는 제조물의 결함을 방치함으로써 생명 또는 신체에 중대한 손해를 입힌 경우 그 손해의 최대 3 배의 징벌적 손해배상책임을 지우도록 개정되었다. 「제조물책임법」의 제조물은 "제조되거나 가공된 동산"으로 정의된다(「제조물책임법」 제 2 조 제 1 호). 이 때문에 소프트웨어를 「제조물책임법」의 적용대상으로 볼 수 있는가 여부에 논쟁의 여지가 있다. 하지만 자동차 부품에 내장된 임베디드 소프트웨어는 제조물과 일체화되어 있기 때문에 소프트웨어의 결함은 곧 제조물 자체의 결함으로 인정되는 것으로 볼 수도 있다.[8]

「제조물책임법」에 따르면 소비자가 피해를 입었을 때 제조업자에게 손해배상책임을 부여할 수 있고, 제조업자가 손해배상책임을 면할 수 있는 사유도 있으며 그 사항은 다음과 같다.

[8] 김윤명, 오병철, 강일신, 장준영, 박규홍, 이하정, 김민주, SW 제조물책임 관련 법제 현황 조사연구

제 4 조(면책사유) ① 제 3 조에 따라 손해배상책임을 지는 자가 다음 각 호의 어느 하나에 해당하는 사실을 입증한 경우에는 이 법에 따른 손해배상책임을 면(免)한다.

1. 제조업자가 해당 제조물을 공급하지 아니하였다는 사실

2. 제조업자가 해당 제조물을 공급한 당시의 과학·기술 수준으로는 결함의 존재를 발견할 수 없었다는 사실

3. 제조물의 결함이 제조업자가 해당 제조물을 공급한 당시의 법령에서 정하는 기준을 준수함으로써 발생하였다는 사실

4. 원재료나 부품의 경우에는 그 원재료나 부품을 사용한 제조물 제조업자의 설계 또는 제작에 관한 지시로 인하여 결함이 발생하였다는 사실

② 제 3 조에 따라 손해배상책임을 지는 자가 제조물을 공급한 후에 그 제조물에 결함이 존재한다는 사실을 알거나 알 수 있었음에도 그 결함으로 인한 손해의 발생을 방지하기 위한 적절한 조치를 하지 아니한 경우에는 제 1 항제 2 호부터 제 4 호까지의 규정에 따른 면책을 주장할 수 없다.

자동차 전자제어 시스템의 경우 「제조물책임법」 제 4 조 1 항 2 호의 "해당 제조물을 공급한 당시의 과학·기술 수준"을 ISO 26262 로 보는 것이 타당하다. 그렇기 때문에 자동차 제조사 입장에서는 ISO 26262 준수 여부가 향후 「제조물책임법」의 면책사유가 될 수 있기 때문에 매우 중요하며 특히 징벌적 손해배상의 도입 이후 그 중요성이 더 높아지고 있다.

5.3 소프트웨어산업진흥법

과학기술정보통신부는 2018. 3. 22. '소프트웨어산업진흥법 전부개정안'을 입법예고한 바 있다. 위 입법예고는 실제 법률개정으로 이어진 것은 아니나, 향후 소프트웨어의 안전에 관한 법적 통제가 이루어질 수 있음을 시사한다. 참고로 위 전부개정안의 내용은 아래와 같다.

제 38 조(소프트웨어안전 기준) ① 소프트웨어사업자는 소프트웨어안전을 확보하도록 노력하여야 한다.

② 과학기술정보통신부장관은 다음 각 호의 사항을 포함하는 소프트웨어안전에 관한 일반기준(이하 '일반안전기준' 이라 한다)을 정하여 고시할 수 있다.

1. 소프트웨어안전 관련 위험분석
2. 소프트웨어안전 보장 설계 및 구현방법
3. 소프트웨어안전 검증 방법
4. 운영 단계의 소프트웨어안전 확보방안
5. 그 밖에 소프트웨어안전 확보에 필요하다고 인정되는 사항

③ 중앙행정기관의 장은 제 2 항에 따른 일반안전기준을 참조하여 소관 분야별로 소프트웨어안전의 세부기준(이하 '세부안전기준' 이라 한다)을 정하여 고시할 수 있다.

④ 다만, 제 2 항 및 제 3 항의 기준은 국가정보화기본법 및 정보통신망 이용촉진 및 정보보호 등에 관한 법, 정보보호 산업의 진흥에 관한 법 등 정보보호 관련 법령에 별도의 기준 및 조치 등이 있는 경우 이를 따라야 한다.

위 전부개정안은 조문을 47 개에서 93 개조로 확대하였다. 이를 통해 소프트웨어안전 개념이 최초로 법적으로 정의되었다. 이에 의하면 "소프트웨어안전"이란 소프트웨어로 인한 사람의 생명이나 신체에 대한 위험의 발생을 방지하거나 이에 대한 충분한 대비가 되어 있는 상태를 말한다 (제 1 장 제 3 조). 또한 제 38 조부터 제 40 조에서는 소프트웨어 안전기준에 대한 고시의 법적 근거를 최초로 마련하고 소프트웨어사업자의 안전기준 준수 의무 및 소프트웨어안전 전문기관 지정 근거를 마련하고 있다. 제 38 조에서 소프트웨어안전에 관한 안전 기준을 정하며 그 내용은 위험분석, 안전 보장 설계 및 구현 방법, 검증 방법 등이 있다. 제 39 조에서는 분야별 위험의 발생 빈도와 피해의 심각도 등을 고려하여 소프트웨어안전을 확보하라고 명시되어 있다.[9]

[9] 과학기술정보통신부. 2018. 3. 22.자 소프트웨어산업 진흥법 전부개정법률(안) 입법 예고. 공고번호 제 2018-142 호. http://www.moleg.go.kr

제 39 조(소프트웨어안전 관리) ① **중앙행정기관의 장은 소관 분야별로 위험의 발생빈도와 피해의 심각도 등을 고려하여 소프트웨어안전의 확보가 필요한 소프트웨어를 안전관리 대상으로 지정할 수 있다.**

② **제 1 항에 따라 지정된 소프트웨어(이하 "안전관리 대상 소프트웨어" 라 한다)를 개발, 제작 또는 생산하는 자는 제 38 조제 3 항에 따른 소프트웨어 개발에 대한 세부안전기준을 준수하여야 한다.**

③ **안전관리 대상 소프트웨어를 운영하고 있는 국가기관 등은 제 38 조제 3 항에 따른 세부안전기준에 운영기준이 포함된 경우 안전관리 대상 소프트웨어의 운영계획을 수립하고 이행하여야 한다.**

5.4 소프트웨어 안전 기본법

2018 년 2 월 22 일 박정의원이 대표발의한 「소프트웨어 안전 기본법」은 소프트웨어에 대한 침해, 사고, 안전에 대한 개념을 정의하고 과학기술정보통신부 장관으로 하여금 3 년마다 "소프트웨어안전관리 기본계획"을 수립하고 시행하도록 하고있다. 중앙행정기관의 장과 지방자치단체의 장은 이에 따라 "소프트웨어안전관리 시행계획"을 수립하고 시행해야 한다. 국무총리를 위원장으로 "소프트웨어안전관리위원회"를 소프트웨어안전관리를 위한 제반 업무를 수행하도록 한다. 이 외에도 "소프트웨어의 성능과 안전에 관한 기준"을 마련하고 이에 따른 안전 평가, 소프트웨어안전책임관 임명, 소프트웨어안전관리원 설립 등을 추진하도록 법제화 노력을 기울이고 있다.

5.5 국외 법제도와 규제 분석

유럽연합은 1985 년 7 월 25 일 결함 있는 제조물에 관한 회원국의 법률, 규칙 및 행정규정의 조정을 위한 유럽 공동체 이사회의 지침으로 「EC 입법지침」을 공표하였다. 「EC 입법지침」이 공표되면서 유럽연합에 속해 있는 나라들은 이 법을 토대로 제조물책임법을 제정하였다. 해당 지침의 제조물규정에서는 "제품이란 제 1 차 농산물과 수렵물을 제외한 모든 동산을 의미하며 그 동산이 다른 동산 또는 부동산에 편입되어 있는 경우도 포함한다"고 규정하고 있으며 소프트웨어 등의 무체물이 '제품'에 포함되는지 여부는 명확하지 않다. 「EC 입법지침」의 항목들을 소프트웨어에 적용시킬 수 있는지와 소프트웨어가 제조물에 해당되는지 여부가 논의되고 있다. 하지만 'SW 제조물책임관련 법제 현황 조사연구'는 "소프트웨어가 다른 동산에 결합되어 문제를 발생한 경우에는 제조물책임에 관한 법리가 적용된다"고 보고 있으며, EU 의 소비자위원회는 Opinion of Consumers Committee on Year 2000 Related Problem 에서 "물리적인 특성을 가진 물질적인 제품과 그 소프트웨어가 불가분으로 결합되어 있을 때에는 소프트웨어를 제조물로 볼 수 있다"는 견해를 표명하고 있다.

영국에서는 1987 년에 「소비자보호법」을 제정하였다. 소프트웨어에 대해 제조물책임을 적용하지는 않지만 소프트웨어가 다른 제품과 일체로 판단될 때에는 제조물책임을 물을 수 있다. 제조물이 통상 당연히 기대되어야 할 안전성을 갖추지 않은 경우 결함이 존재하는 것으로 규정하고 있다. 제 4 조 1 항에 국내 「제조물책임법」과 비슷하게 당해 결함이 관련시점에 있어서 제조물에 존재하지 않았다면 면책사유임을 명시하고 있다.

독일에서는 「EC 입법지침」을 수용한 「제조물책임법」을 제정하였으며, 여기에서 "다른 동산이나 부동산의 일부를 구성하는 경우를 포함한 모든 동산 및 전기를 말한다"고 규정하고 있다. 온라인상으로 유통되는 소프트웨어에 대해 제조물책임법을 적용해야 하는지에 대해 논란이 있다. 이를 긍정하는 사람들은 제조물책임법의 적용 대상 확대를 근거로 두지만 반대하는 입장은 제조물의 정의에 맞게 소프트웨어가 동산이 아니기 때문에 제조물이 될 수 없다고 주장한다.

일본에서는 1995 년에 제조물책임법 시행 이후 2002 년에 개정되었다. 소프트웨어 자체는 제조물에 해당하지 않지만, 소프트웨어가 유체물에 체화 되어있으면 제조물로 보았다.

미국의 제조물책임을 규정하고 있는 「리스테이트먼트」는 "사용 내지 소비를 위하여 상업적으로 공급된 유형의 동산"을 그 대상으로 하고 있다. "유형의 동산"을 대상으로 하고 있기 때문에 무체재산의 경우 제조물책임이 적용될 수 있는지에 관하여 논의가 계속되고 있다. 일반적인 판결의 경향은 책의 내용과 같은 정보는 제조물에 해당되지 않는다고 보고 있는데, 최근 제 9 순회법원이 "소프트웨어와는 달리 책의 내용은 제조물에 해당하지 않는다"고 판단하여 소프트웨어의 제조물성을 인정하는 단초를 제시하기도 하였고 [10], Schafer v. State Farm Fire and Cas. Co. 사건에서는 명시적으로 소프트웨어를 제조물로 판단하기도 하는 등 소프트웨어에 대한 제조물책임을 인정하는 판결들이 등장하고 있다.

10 Winter v. G. P. Ptnam's Son, 938 F.2d, 1991, page 1033, 1035

6. 자동차 SW 국내 기업 현황

차량 내의 소프트웨어 비중이 높아지고, 이에 따라 소프트웨어의 결함이 증가하게 되자, 안전한 소프트웨어를 효율적으로 개발해야 한다는 목소리가 커지고 있다. 물론, 이 변화에는 도요타 사건에서 발생한 징벌적 배상 금액의 규모와 원인 규명 과정에서 드러난 위험한 소프트웨어의 실체가 큰 영향을 미쳤다. 이 사건의 긍정적 파급 효과는 ISO 26262 를 필수로 준수할 수 있게 한 것과 더불어, 개발자들 스스로 문제의 심각성을 공감하게 하고 인식의 변화를 이끌어냈다는 것이다. 이러한 가운데, 자동차 산업은 많은 고민들을 안고 있다. 설문 조사(총 28 개 업체)와 인터뷰(총 6 개 업체)를 통해 자동차 SW 안전에 대한 기업들의 현황과 현실적 우려를 살펴보았다.

6.1 안전한 소프트웨어 개발에 대한 인식

자동차 산업에서는 '안전한 소프트웨어 개발'은 'ISO 26262 를 준수하여 소프트웨어를 개발하는 것'과 동일한 의미로 사용된다. 그러나 ISO 26262 는 소프트웨어의 안전성을 보장하기 위해 어떻게 해야 하는지에 대한 표준이며, 위험을 측정하고 이를 방지할 수 있는 제품(시스템, 하드웨어, 소프트웨어)을 만드는 것을 목표로 하고 있다. 즉, ISO 26262 는 소프트웨어를 지속하여 안전하게 만들 수 있는 것에 대한 고려가 되어 있는 것이 아니라는 것이다.

소프트웨어는 비가시성, 불명확성, 주요 자원이 사람인 것 등의 특성을 가지고 있어 지속하여 유사한 생산성 및 결과물의 품질을 보증하는 것에 어려움이 있다. 독일 완성차 회사들은 이러한 소프트웨어의 특성을 고려하여 소프트웨어 개발 공정을 모니터링 해야 하는 필요성을 인지하고 SPICE 를 자동차 산업에 맞게 수정하여 자동차 소프트웨어 개발 프로세스를 평가하는 A-SPICE 를 만들었다. 그리고 협력 업체의 계약 기본 사항으로 A-SPICE 레벨을 요구하고 있다.

꼭 완성차 회사들의 요구가 아니라고 하더라도, 전문가들은 안전한 소프트웨어를 개발하기 위해서는 결국 ISO 26262 와 A-SPICE 기준을 모두 만족해야 한다고 말한다. 그러나 설문 조사 결과, ISO 26262 를 알고 있고(93%), 적용하거나 적용 계획이 있다고(62%) 응답한 것에 비해 A-SPICE 를 알고 있고(79%) 레벨을 획득하거나 획득 계획이 있다고(32%) 응답한 비율이 낮게 나타났다.

이러한 수치의 원인은 다양하게 분석될 수 있지만, 가장 큰 요인은 시스템, 그 중에서도 기계 중심인 기존의 자동차 산업 구조에 있다. 이로 인해, 전체 연구원 중 소프트웨어 인력(약 10~30%)의 비중이 낮고, 전반적으로 소프트웨어에 대한 인식이 아직은 낮은 편이다. ISO 26262 는 기존의 시스템, 하드웨어에서도 필수적으로 적용해야 하는 표준으로 받아들여지고 있는 반면, A-SPICE 는 소프트웨어에 대해서만 필수적으로 적용 받기 때문에 상대적 중요성이나 인식

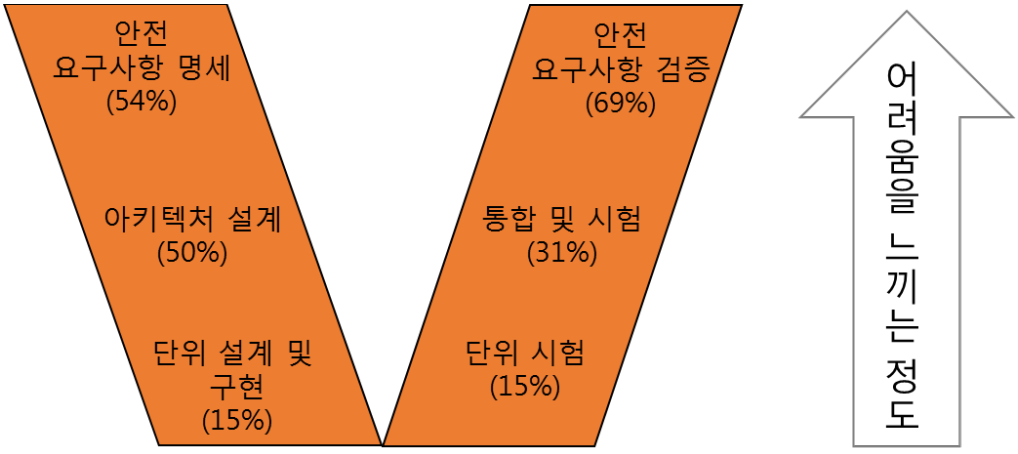
제고가 느리게 진행되고 있는 것으로 해석할 수 있다. 또한, ISO 26262 와 A-SPICE 를 각각 수행해야 하는 별개의 것으로 인식하고 있어 두 가지 중 하나를 우선 달성하고자 할 때, A-SPICE 보다는 ISO 26262 를 선택하는 것으로 보인다.

ISO 26262 및 A-SPICE 적용 시, 어려운 부분으로는 (1) 전담하는 조직 및 인원의 부재(71%), (2) ISO 26262 와 A-SPICE 에 대한 이해 부족(42%), (3) 사내 구성원의 인식 부족(40%)이라고 응답하였다. (1)과 (3)은 앞서 설명한 것과 같이, 기계 중심의 산업 및 조직의 구조에 기인한 것으로 이를 해결하기 위해서는 경영진의 변화 의지와 사내 조직 문화의 변화가 수반되어야 하며, (2)는 ISO 26262 와 A-SPICE 에 대한 가이드라인 및 교육으로 개선해 나갈 수 있다.

6.2 협업 체계에서의 소프트웨어 개발 성숙도

자동차 산업은 복잡한 협업 체계로 이루어져 있음에도 불구하고, 1) 국내의 협력 업체 간의 협의가 산출물 기반으로 이루어지지 않고, 2) 각 상위 업체의 요구사항이 명확하지 않으며, 3) 담당 업체나 담당자의 개별 능력에 따라 산출물 수준이 달라지는 문제가 있다. 이는 글로벌 경쟁에서 위험요소로도 작용을 할 수 있다. 설문 조사 결과에 따르면, 현업 종사자들은 그림 10 과 같이 안전 요구사항 명세 및 검증에 가장 큰 어려움을 느끼고 있었다.

그림 10. 안전한 소프트웨어 개발 단계별 어려움을 느끼는 정도



이는 협력 체계에서 안전 요구사항 추출/분석/명세에 대한 역할 및 책임이 불명확한 것이 큰 요인으로 꼽힌다. 완성차 회사와 부품사가 서로 각자의 역할에 대해 가지는 기대가 상이한 것으로 조사되었는데, 기능 안전 요구사항을 추출하는 역할의 주체가 완성차 회사인지, 부품사(Tier-1)인지에 대해 서로 다른 견해를 가지고 있었다. 그리고 대부분의 업체들이 ASIL 등급에 대한 능동적 고민을 하지 않고, 완성차 회사의 결정(69%)에 따르는 만큼, 완성차

회사에서부터 누락되거나 모호한 안전 요구사항이 나왔을 때 이에 대한 미온적 대처를 취하게 된다.

또한, 시스템에 비해 규모가 상대적으로 작은 소프트웨어는 요구사항 단계에서부터 아키텍처 및 설계, 구현에 대한 역할과 책임이 분리되어 있지 않은 경우가 많다. 이로 인해 V 모델의 좌측에 위치한 각 단계마다 작업자 간의 검토 과정을 거쳐야 함에도 동일 작업자에 의해 질적인 검증이 진행되기 어려운 것이 현실이다. V 모델의 우측에 위치한 각 테스트 단계는 좌측 프로세스 단계마다 산출된 결과물을 토대로 계획을 수립하고 테스트를 진행하도록 권고하고 있다. 그러나, 실제로는 한 단계씩 뒤쳐져서 그 다음 단계의 결과물을 토대로 테스트 계획을 수립하고 있다. 예를 들어, 단위 테스트의 경우 구현된 코드를 기반으로 테스트 케이스를 작성하여 코드 기준으로 커버리지는 우수하나, 유의미한 테스트가 진행되기가 어렵다. 결과적으로, 소프트웨어 개발 전반적으로 소프트웨어 공학적 접근이 부진했음을 알 수 있다.

6.3 소프트웨어 개발 프로세스의 급진적 변화 요구

소프트웨어 개발 프로세스를 잘 적용하고 있는지에 대한 조사 결과, 프로세스를 팀이나 프로젝트에 적합하게 테일러링해서 사용하는 경우는 26%, 전사 프로세스를 가지고 있는 경우는 48%였고, 프로세스 없이 팀이나 개인의 역량에 의존하는 경우도 26%에 달하고 있었다. 또한, 각 단계를 잘 이해하고 수행하는지에 대한 질문에 23%만이 긍정적인 답변을 하였다.

소프트웨어 개발에 대한 전사 프로세스가 존재하지만, 이를 팀 혹은 프로젝트별로 테일러링하여 사용할 수 없는 것에 대한 원인은 (1) ISO 26262 에 대한 부담, (2) 조직의 소프트웨어 프로세스 역량 부족으로 파악되었다. ISO 26262 를 준수하여 제조물을 개발하였다면 안전하게 개발하려고 충분히 노력했다는 것으로 보아 징벌적 과징금을 면책 받을 수 있다. 그러나 아직 자동차 결함으로 인한 법정 소송에서 면책을 받은 사례가 없으므로, 어느 수준까지 만족해야 하는지에 대한 기준을 판단하기 어렵다. 특히, 소프트웨어 프로세스 역량과 전문 인력이 부족한 상태에서 팀이나 프로젝트 구성원들에게 테일러링 역할을 할당하는 상황까지 더해져 전사 프로세스와 팀/프로젝트 프로세스는 동일할 수밖에 없다.

이러한 상황에도 불구하고, 부품사들은 독일의 완성차 회사들로부터 A-SPICE 레벨 획득에 대한 요구를 받고 있다. A-SPICE 인증은 소프트웨어 공학적 마인드 변화와 반복적 수행에 의한 내재화가 필수적이므로 점차적으로 발전시켜 제대로 인증을 받아야 함에도 불구하고 계약을 위해 특정 기간까지 인증을 받아야만 하는 처지에 놓여있다. 이는 과거의 CMMI 나 SPICE 의 실패 사례들에서 발생한 잠재적 문제(이벤트성 산출물 작성에 의한 소프트웨어 공학적 접근의 부정적 인식 증가 등)의 발생 가능성을 내포하고 있다.

6.4 도메인 특수성에 의한 한정된 정보 및 인력 구조

자동차 산업은 도메인의 특수성으로 인하여 제한적인 인력 시장을 유지해왔다. 이로 인해 자동차 소프트웨어에 대한 정보 역시 해당 도메인에서만 공유 가능한 정보로 유지되었다. 현재 자동차 소프트웨어 분야에서 일어나는 많은 문제점(소스 코드 규모의 증가, 기능 및 복잡도 증가, 비기능 요구사항의 증가 등)은 타 소프트웨어 분야에서는 오래 전부터 발생했고 이를 해결하기 위한 노력들이 꾸준히 이어져왔다.

그러나, 한정적인 정보 공유 및 인력 구조는 기존의 소프트웨어 산업의 해결책들을 자동차 소프트웨어에 적용하는 것을 어렵게 하고 있다. 도메인 내의 정보와 인력이 외부로의 확산이 어려운 경우, 도메인 내의 전문가들이 외부 인력의 도움을 받아 내부에 적용 가능한 프로세스 및 산출물을 만들어 내야 한다. 설문 조사 결과에서 가장 많은 도움을 받는 형태가 외부 세미나 및 교육(65%)과 컨설팅(55%)으로 조사되었다. 그러나 실제 얼마나 도움이 되는지에 대한 인터뷰 질문에서는 자동차 소프트웨어에 특화된 베스트 프랙티스를 참고할 수 없어서 적용이 어렵다는 의견이 우세했다.

이로 인해, 가장 우선시 되어야 하는 지원에서 특화된 교육(54%)과 전문가 채용(54%)이 필요하다고 답하였다. 사내 교육을 실시하는 비율(25%)도 낮았지만, 이 역시도, 전문 강사의 부족으로 자동차 소프트웨어에 특화된 교육을 진행하기 어려워서 일반 소프트웨어 혹은 임베디드 소프트웨어 관련 기술들을 진행하는데 그치고 있다. 전문가 채용에 있어서는 기존의 기계 중심의 자동차에서의 소프트웨어가 아닌 소프트웨어 중심의 자동차에서의 소프트웨어 개발임을 인지하고, 이를 개발 및 관리할 수 있는 인력을 충분히 확보해야 하지만 여전히 시스템 및 하드웨어 중심의 조직에서는 경영진의 인식 변화가 수반되어야 하는 부분이다.

6.5 마무리

안전한 자동차 소프트웨어를 효율적으로 개발하는 것에 있어 국내 업체가 당면해 있는 문제들을 살펴보았는데, 이를 해결하기 위해 가장 시급한 것은 (1) 소프트웨어 중심의 자동차로의 인식 변화와 (2) ISO 26262 와 A-SPICE 를 통한 안전한 소프트웨어 개발 프로세스 내재화이다.

안전한 소프트웨어 개발 프로세스 내재화를 위해서 이에 대한 실무자의 이해도 향상이 우선되어야 한다. 전반적인 이해도 향상을 위해서는 쉽게 접근 가능하고 이해하기 쉬운 가이드라인 제공이 효율적인 방법 중 하나이다.

본 가이드라인에서는 이를 위해, (1) ISO 26262 표준을 적용하여 개발하는 방법과 (2) ISO 26262 와 연관된 A-SPICE 단계들에 대해 소개하고 (3) 이를 심화 학습할 수 있도록 템플릿 예제와 해당 템플릿을 사용한 자동차 소프트웨어 사례를 제공한다.

6.6 참여기업

다음은 현황 조사에 참여한 기업에 대한 정보이다.

표 2. 자동차 소프트웨어 관련 설문 업체

규모	유형	참여업체 수
대기업	부품사(Tier-1)	3
	부품사(Tier-2)	1
	자동차 소프트웨어 개발사	1
	완성차 제조사	2
스타트업	자동차 소프트웨어 개발사	3
중견기업	부품사(Tier-1)	5
	부품사(Tier-2)	2
	소프트웨어 개발도구 공급사	2
중소기업	부품사(Tier-2)	4
	부품사(Tier-3)	1
	소프트웨어 개발 도구 공급사	2
	자동차 소프트웨어 개발사	2

표 3. 자동차 소프트웨어 관련 인터뷰 업체

규모	유형	참여업체 수
대기업	부품사(Tier-1)	2
	완성차 제조사	1
스타트업	자동차 소프트웨어 개발사	1
중견기업	소프트웨어 개발도구 공급사	2

PART 3

가이드 개요

1 기능안전 소프트웨어 개발 단계

2 ASIL 에 따른 소프트웨어 개발 방법

3 소프트웨어 개발 수명주기 단계별 가이드

4 지원 프로세스 가이드

5 ISO 26262 와 A-SPICE 공통 부분

6 적용 예제

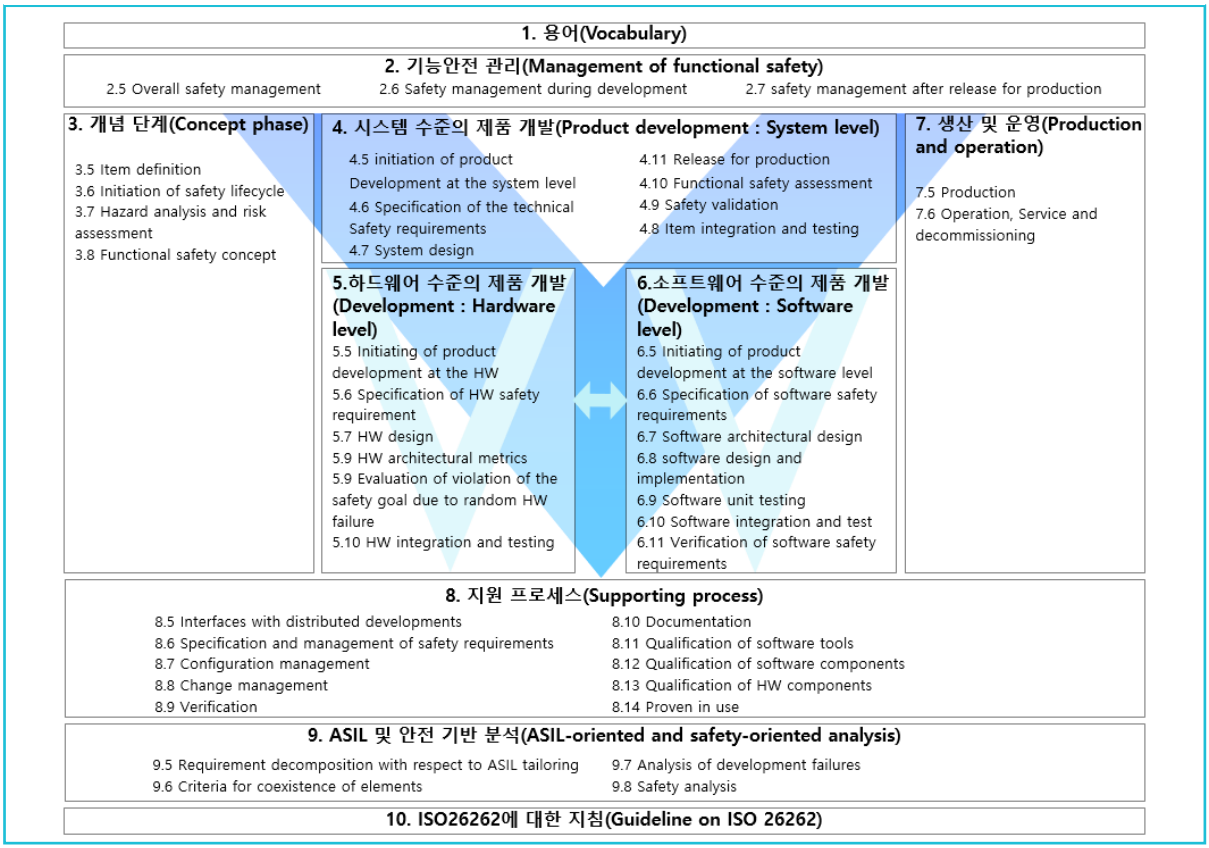
III. 자동차 소프트웨어 안전 가이드

1. 기능안전 소프트웨어 개발 단계

1.1 ISO 26262 전체 구조

ISO 26262 는 다음과 같이 전체 10 개 파트로 구성된다. 10 개 파트는 ISO 26262 를 적용하는 프로젝트에서 수행하는 프로젝트 수준의 개발 및 관리 활동과 조직 수준에서 수행하는 활동들을 포함하고 있다. 개발 활동은 개발 단계별 검증 활동을 수행할 수 있도록 V 수명주기 모델을 따르고 있다. 또한 시스템, 하드웨어, 소프트웨어 개발이 독립적으로 병행될 수 있는 구조로 되어 있어서 OEM 과 각 협력 업체의 분업 환경에 적합하게 업무를 분할하여 적용할 수 있도록 구성되어 있다.

그림 11. ISO 26262 전체 구조



각 파트 별로 주요 내용은 다음 표와 같다. 파트 1 은 ISO 26262 전체에서 사용되는 용어를 포함한다. 파트 2 는 기능안전 개발을 수행하는 조직 차원에서 수행하는 안전 문화 확립, 안전

관리자 역량관리를 비롯한 전반적인 안전 관리 요구사항을 포함한다. 파트 3 은 개발 아이템 정의를 기반으로 위험원 분석 및 리스크 평가를 수행하여 ASIL 등급 결정, 안전 목표, 기능 안전 요구사항을 도출하는 활동을 포함한다. 파트 4 부터 파트 6 까지는 시스템 수준, 하드웨어 수준, 소프트웨어 수준에서 수행해야 하는 개발 활동을 포함한다. 파트 7 은 개발 완료된 아이템에 대해서 생산 및 운영에 대한 요구사항을 포함한다. 파트 8 은 개발 활동에 공통적으로 적용할 수 있는 형상 관리, 문서화 같은 활동을 포함한다. 파트 9 와 파트 10 은 ASIL 및 안전 분석과 ISO 26262 지침을 포함한다.

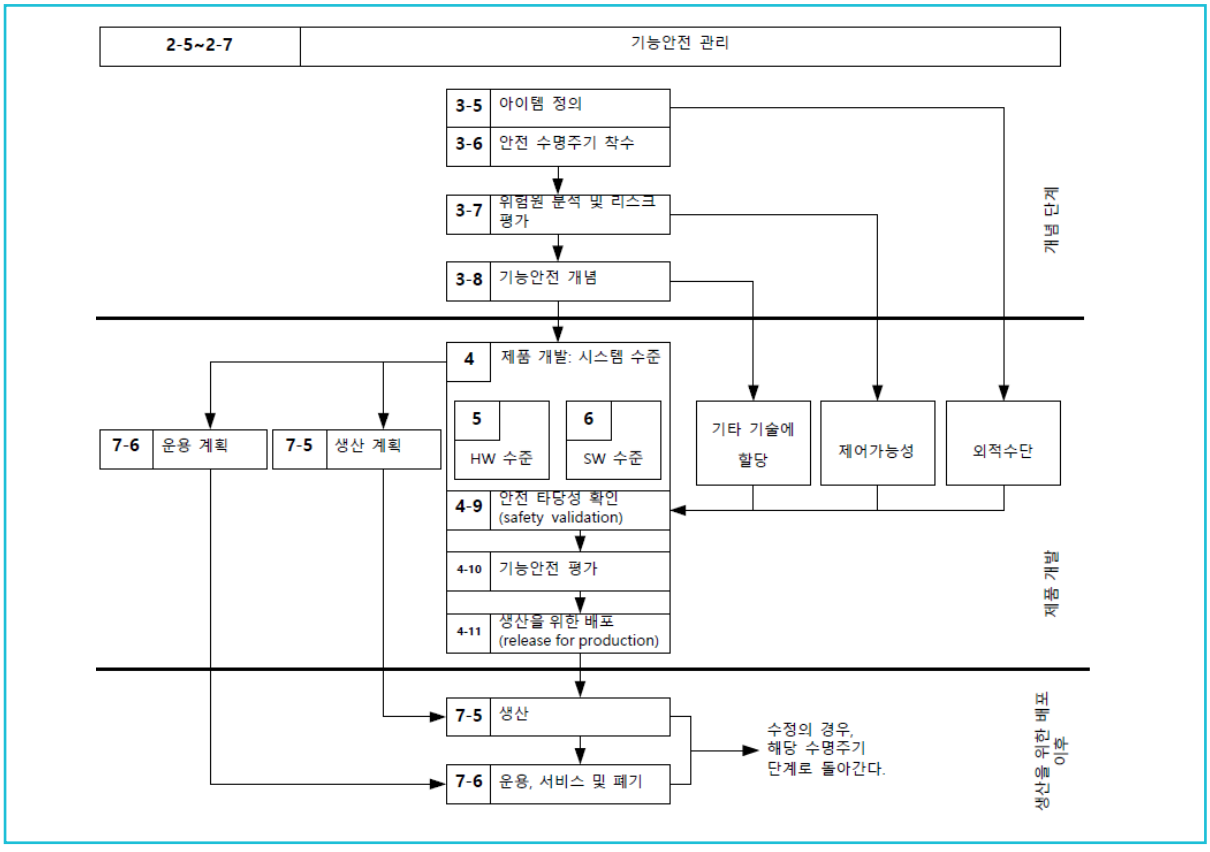
표 4. ISO 26262 주요 내용

파트	제목	주요 내용
1	용어	ISO 26262 관련 용어 정의
2	기능 안전 관리	안전문화와 같이 조직 차원에서 가져야할 아이템 개발, 생산 이후에 걸친 전반적 안전 관리 활동
3	개념 단계	개발 아이템 정의를 기반으로 위험원 분석 및 리스크 평가를 통한 ASIL 결정하고 안전목표 수립
4	시스템 수준의 제품 개발	V 수명주기에 따른 시스템 수준의 제품 개발 활동
5	하드웨어 수준의 제품 개발	시스템 설계 명세를 기반으로 하드웨어 수준의 개발, 통합, 검증 활동
6	소프트웨어 수준의 제품 개발	V 수명주기에 따른 소프트웨어 수준의 개발, 통합, 검증 활동
7	생산 및 운영	생산 계획, 양산, 서비스에 관한 활동
8	지원 프로세스	안전 요구사항 명세 및 관리, 형상관리, 변경관리, 검증, 소프트웨어 툴, 하드웨어 컴포넌트 자격 검증
9	ASIL 및 안전 기반 분석	ASIL 분해 방법, 위험 분석 방법
10	ISO 26262 에 대한 지침	ISO 26262 주요 개념, ASIL 분해 등 고려 사항

1.2 ISO 26262 안전 수명주기

ISO 26262 는 개념 단계, 제품 개발, 생산, 운영, 서비스 폐기의 안전 수명주기 활동을 수행하도록 되어 있다. 그림 12 의 안전 수명주기 흐름은 ISO 26262 전체 파트에서 수행하는 각 단계별 활동 흐름을 보여준다. 안전 수명주기는 안전 개발 활동을 수행하는 조직 차원에서 수행하는 기능안전 관리와 아이템을 개발하는 프로젝트에서 수행하는 개념 단계, 제품 개발, 생산을 위한 배포 이후 단계로 구성된다. 각 활동에 명시된 번호는 ISO 26262 해당 파트와 절을 의미한다. 예를 들면 3-5 아이템 정의는 ISO 26262 파트 3, 5 절 요구사항을 말한다.

그림 12. 안전 수명주기 흐름



1) 기능안전 관리

기능안전 관리는 ISO 26262 파트 2, 5 절부터 7 절까지 해당하는 활동으로 기능안전 아이템을 개발하는 조직에서 해야 하는 조직 차원의 활동을 정의하고 있다. 주요 내용은 조직의 안전 문화, 안전 활동 수행 인원의 역량관리, 기능안전 관리자(Safety Manager)의 역할 및 책임, 안전 계획(Safety Plan), 안전 케이스(Safety Case), 확인 수단(Confirmation Measure) 활동이다.

ISO 26262 전체 파트에서 요구하는 기능안전 개발 활동이 잘 수행되려면 조직 내에 안전 문화가 정착되어야 한다. 예를 들면, 안전을 최우선으로 여기고 기능 안전 개발에 필요한 자원이 할당되어야 원활하게 기능안전 개발 활동이 수행될 수 있다. ISO 26262 파트 2 에서 조직의 안전 문화가 어느 정도 수준인지 평가할 수 있는 안전 문화 평가 항목들이 예시로 포함되어 있다. ISO 26262 를 적용하는 조직에서는 이 평가 항목들을 활용하여 먼저 안전 문화가 조직 내에 정착되도록 해야 한다.

안전 관리자는 안전 수명주기 개발 단계에서 기능 안전 활동 계획과 조정을 책임진다. 안전 계획에 따라서 기능 안전 활동이 적합하게 수행되고 있는지 모니터링하는 것이 주요 업무이다. 안전 관리자는 안전 계획을 수립해야 하며 안전 수명주기 전체 각 개발 단계별 수행 목적, 절차, 담당자, 필요한 자원, 수행 일정, 관련된 산출물들을 포함한다.

안전 케이스(safety case)는 아이템의 안전 요구사항이 완전하다는 것을 증명하는 개발 활동의 최종 보고서로 볼 수 있다. 개발 기간 동안 ISO 26262 의 안전 활동에 따라서 개발된 작업 산출물을 근거해서 요구사항이 완전히 충족된 것을 증명할 수 있다. 안전 케이스는 개발 과정 진행됨에 따라서 점진적으로 작성되며 예비 안전 케이스, 중간 안전 케이스, 최종 안전 케이스 형태로 작성될 수 있다.

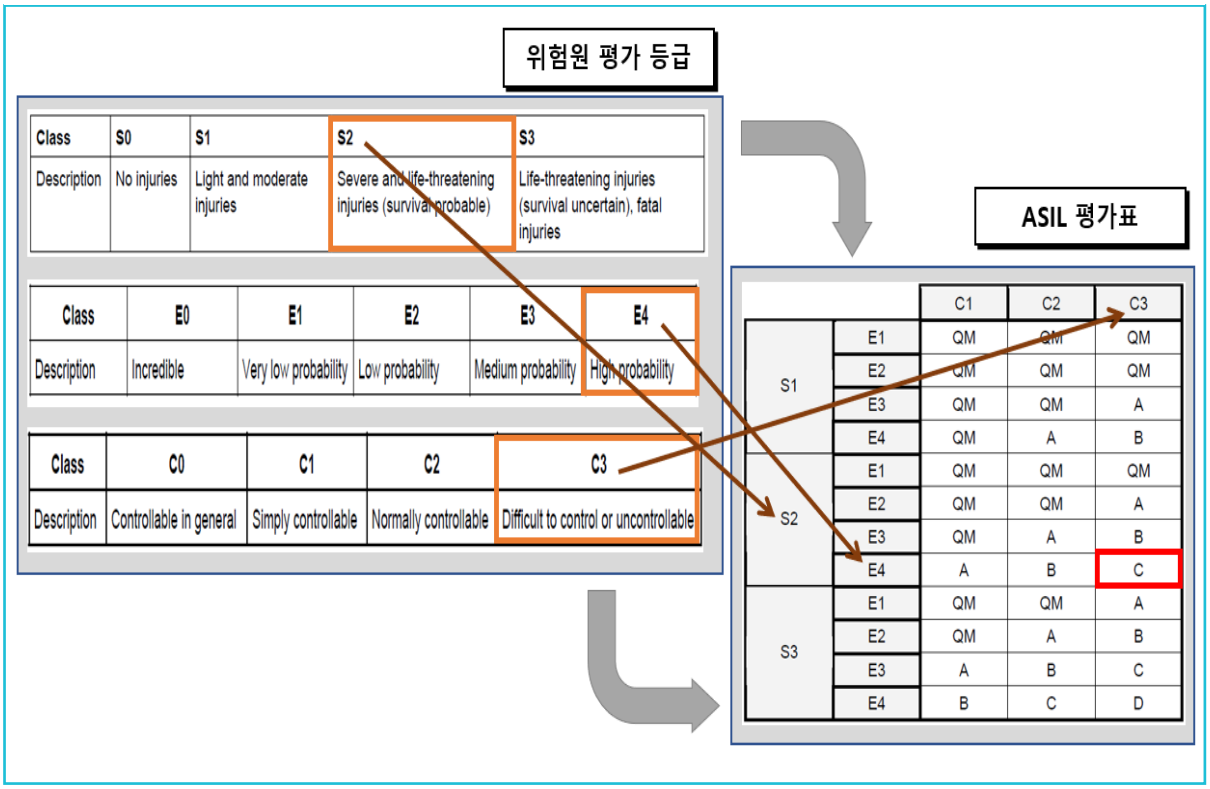
확인 수단(confirmation measure)은 안전 수명 주기 활동이 ISO 26262 에서 요구하는 활동들이 적합하게 수행되었는지 확인하는 것으로 확인 검토(confirmation review), 기능 안전 심사(Functional safety audit), 기능 안전 평가(Functional safety assessment)로 구분된다. 각 확인 수단은 ASIL 레벨에 따라서 프로젝트 수행 인원과 독립성을 가진 인원 또는 조직에서 수행해야 한다. 어떤 조직에서 ASIL 레벨 B 달성과 같은 기사를 접할 때가 있는데 이 때는 해당 조직에서 독립성을 갖는 전문 기관에서 기능 안전 평가를 받는 경우이다. ISO 26262 에서는 CMMI, A-SPICE 와 같이 공식적인 인증을 관리하는 기관이 없으며 전문 기관에서 자체적인 평가 체계와 기능안전 평가 요구사항을 기반으로 인증 평가를 수행하고 있다.

2) 개념 단계

개념 단계는 아이템의 정의, 위험원 분석 및 리스크 평가, 기능 안전 개념 도출을 수행한다. 위험원 분석 및 리스크 평가는 아이템 정보를 기반으로 해당 아이템의 리스크를 평가하여 ASIL 레벨을 산정하는 활동으로 이후 기능 안전 개발 단계에서의 출발점이 된다. 위험원 분석 및

리스크 평가는 아이템의 기능 결함(malfunction)으로 인해 발생할 수 있는 차량 수준의 위험을 시나리오를 구성하여 평가한다. 시나리오를 통해 사고의 노출도, 인체에 미치는 심각도, 사고를 통제할 수 있는 통제도의 3 가지 요소를 분류하여 그림 13 의 평가표를 기준으로 ASIL 레벨이 결정된다. 예를 들면, 아이템의 고장으로 인해 발생할 수 있는 시나리오의 심각도가 S2(생명의 위협 심각), 노출도 E4(발생 확률 매우 높음), 통제도 C3(운전자가 제어 불능)인 경우에는 ASIL C 로 평가된다.

그림 13. ASIL 평가표 예제



ASIL 등급은 D 가 가장 높은 등급이고, ASIL A 부터 ISO 26262 에서 요구하는 기능 안전 활동을 적용해야 한다. QM 등급으로 판정 시에는 ISO 26262 적용 대상은 아니지만 CMMI, A-SPICE, IATF 16949 와 같은 품질 시스템을 개발 활동에 적용해야 한다.

일반적인 자동차 아이템의 ASIL 등급은 아래 그림 14 과 같이 분류할 수 있으나 실제 ASIL 등급은 위험 분석 및 리스크 평가에 따라서 결정되어야 한다.

그림 14. 일반적인 ASIL 등급 분류

위험원 분석 및 리스크 평가는 차량의 운전 시나리오 및 해당 아이템과 관련되는 전장 부품의 상호 작용에 따라서 달라진다. 일반적으로 OEM 에서 차종과 모델, 스펙 요구사항을 정의하고 협력업체에 개발을 의뢰하게 되므로 위험원 분석 및 리스크 평가는 OEM 에서 수행하는 것이 효과적이다. ASIL 등급과 함께 안전 목표(Safety Goal), 기능안전 요구사항(Functional Safety Requirement)을 도출되면 시스템 및 하드웨어, 소프트웨어 개발 협력 업체는 해당 ASIL 등급을 기준으로 안전 개발 활동이 수행된다.

그러나 선행 개발 또는 특정 타겟을 정의하지 않는 아이템을 개발해야 할 경우가 있다. 개발 아이템이 최종적으로 어떤 전장부품이나 시스템에 탑재될지 예측할 수 없거나 시스템 요구사항을 모르는 상태에서 개발하는 경우가 있다. 아이템이 여러 용도로 다르게 적용될 수 있고 여러 고객을 위해 일반적인 형태로 개발될 수도 있다. 이런 경우에 ISO 26262 파트 10 에서는 SEooC(Safety Element out of Context)를 적용하도록 하고 있다. Out of Context 는 특정한 Context 의 범위 밖에서 개발하는 경우, 즉 특정 시스템에 종속되지 않도록 최종 시스템에 적절한 가정을 하여 개발하는 것이다. 하나의 SEooC 는 시스템, 시스템 집합, 하부 시스템, 소프트웨어 컴포넌트, 하드웨어 컴포넌트나 부품이 될 수 있다. SEooC 는 상위 설계 단계에서 할당된 안전 요구사항을 포함하여 요구사항, 설계, 외부 설계에 대해서 가정을 세우고, 이에 따라 안전 개발

활동을 수행한다. ASIL 등급도 가정한 안전 요구사항에 따라서 정해지며, ASIL 등급에 따라서 ISO 26262 요구사항이 적용된다.

3) 제품 개발

제품 개발에서는 시스템 수준 개발, 하드웨어 수준 개발, 소프트웨어 수준 개발에 따라서 아이템이 개발된다. 개발된 아이템은 안전 타당성 확인과 기능안전 평가를 통해서 양산에 적용되도록 배포된다. 시스템 수준 개발은 개념 단계에서 도출된 기능안전 개념을 기준으로 안전 매커니즘을 설계한다. 시스템 수준 개발은 기술 안전 요구사항 명세, 시스템 설계, 구현, 시스템 통합, 타당성 확인, 검증, 기능안전 평가를 수행한다. 시스템 설계 과정에서 하드웨어 및 소프트웨어에서 수행해야 할 기능이 할당된다. 하드웨어 및 소프트웨어에 할당된 요구사항을 기준으로 하드웨어 개발과 소프트웨어 개발이 수행된다. 하드웨어 수준 개발에서는 시스템 설계 명세서를 기반으로 하드웨어 안전 요구사항 명세, 하드웨어 설계, 구현과 하드웨어 통합 및 시험이 수행된다. 소프트웨어 개발에서는 시스템 설계 명세서를 기반으로 소프트웨어 안전 요구사항 명세, 소프트웨어 아키텍처 설계, 구현, 소프트웨어 통합 및 시험, 소프트웨어 안전 요구사항이 수행된다.

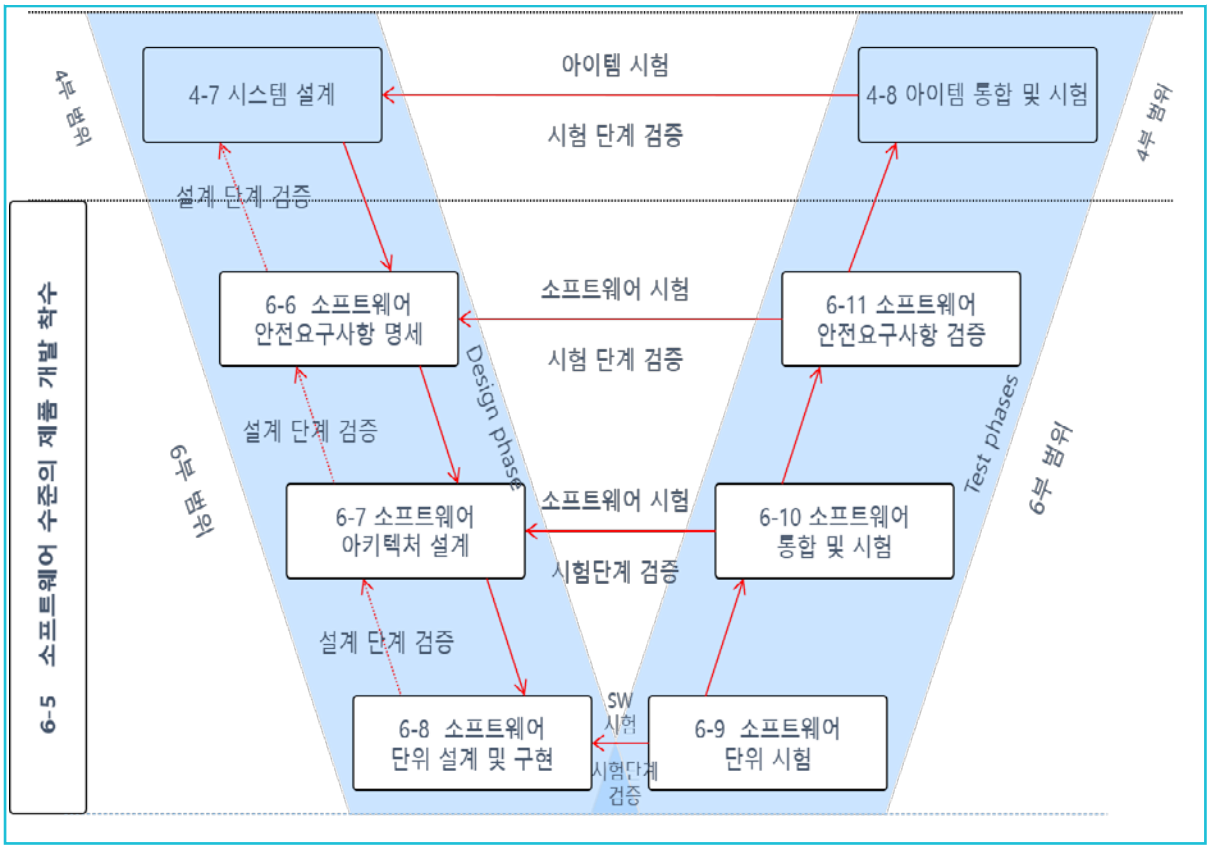
4) 생산을 위한 배포 이후

생산을 위한 배포 이후 단계는 아이템 개발이 완료된 후에 생산과 운영, 서비스 및 폐기 활동을 수행한다. 여기에는 안전 관련 특별 특성 및 유지보수, 수리, 폐기를 위한 지침 개발과 관리 활동이 수행된다.

1.3 기능안전 소프트웨어 개발 단계

ISO 26262 파트 6 에서 소프트웨어 개발 단계는 그림 15 와 같이 V 모델을 기반으로 수행된다. V 모델은 소프트웨어 개발 및 테스트 활동이 체계적으로 수행되도록 서로 짝을 이루어 연결되어 진행되도록 구성된다. 예를 들면, 소프트웨어 안전 요구사항 명세는 소프트웨어 안전 요구사항 검증 단계를 통해 검증해야 한다. V 모델 기반으로 소프트웨어 개발의 신뢰성을 높일 수 있다.

그림 15. 소프트웨어 개발 기준 단계 모델



ISO 26262 각 파트는 단계별로 수행하는 활동의 목적, 입력물, 출력물을 정의하고 있다. 입력물은 수행하는 활동에 필요한 사전 조건으로 소프트웨어 안전 요구사항 명세의 경우 시스템 수준 개발에서 기술 안전 개념과 시스템 설계 명세서가 있어야 한다. 출력물은 해당 활동을 통해 작성되는 산출물이며 소프트웨어 안전 요구사항 명세의 경우 소프트웨어 안전 요구사항 명세서가 작성되어야 한다. ISO 26262 파트 6 의 소프트웨어 개발 단계별 수행 활동 및 입력, 출력물의 전체 항목은 표 5 와 같다.

표 5. 소프트웨어 개발 단계 요약

단계	목적	입력물	출력물
소프트웨어 수준의 제품 개발 착수	소프트웨어 개발 활동을 위한 기능안전 활동 계획을 수립한다.	• 프로젝트 계획서 • 안전 계획서 • 기술안전 개념 • 시스템 설계 명세서 • 아이템 통합 및 시험 계획	• 안전 계획 • 소프트웨어 검증 계획 • 모델링 및 프로그래밍 언어의 설계 및 코딩 지침 • 도구 적용 지침
소프트웨어 안전 요구사항 명세	소프트웨어 안전 요구사항을 도출하고 검증한다.	• 기술안전 개념 • 시스템 설계 명세서 • 하드웨어 소프트웨어 인터페이스 명세서 • 안전 계획(갱신) • 소프트웨어 검증 계획	• 소프트웨어 안전 요구사항 명세서 • 하드웨어 소프트웨어 인터페이스 명세서 • 소프트웨어 검증 계획서 (갱신) • 소프트웨어 검증 보고서
소프트웨어 설계	소프트웨어 안전 요구사항을 구현하는 소프트웨어 아키텍처 설계를 개발한다.	• 안전 계획 • 모델링과 프로그래밍 언어를 위한 설계 및 코딩 지침 • 하드웨어 소프트웨어 인터페이스 명세서 • 소프트웨어 안전 요구사항 명세서 • 소프트웨어 검증 계획(갱신) • 소프트웨어 검증 보고서(갱신)	• 소프트웨어 아키텍처 설계 명세서 • 안전 계획(갱신) • 소프트웨어 안전 요구사항 명세서(갱신) • 안전 분석 보고서 • 종속 고장 분석 보고서 • 소프트웨어 검증 보고서 (갱신)
소프트웨어 단위 설계 및 구현	소프트웨어 아키텍처 설계와 관련된 소프트웨어 안전 요구사항에 따라 소프트웨어 단위를 설명한다. 명시한대로 소프트웨어 단위를	• 모델링과 프로그래밍 언어를 위한 설계 및 코딩 지침 • 소프트웨어 검증 계획(갱신) • 소프트웨어 아키텍처 설계 명세서 • 안전 계획(갱신) • 소프트웨어 안전	• 소프트웨어 단위 설계 명세서 • 소프트웨어 단위 실행 • 소프트웨어 검증 보고서 (갱신)

	구현한다. 소프트웨어 단위 설계 및 구현의 정적 검증	요구사항 명세서(갱신) • 소프트웨어 검증 보고서(갱신)	
소프트웨어 단위 시험	소프트웨어 단위가 소프트웨어 단위 설계 명세서를 충족하며 필요하지 않은 기능은 포함하지 않는다는 것을 증명한다.	• 하드웨어 소프트웨어 인터페이스 명세서 • 소프트웨어 검증 계획(갱신) • 안전 계획(갱신) • 소프트웨어 단위 설계 명세서 • 소프트웨어 단위 구현 • 소프트웨어 검증 보고서(갱신)	• 소프트웨어 검증 계획 (갱신) • 소프트웨어 검증 명세서 • 소프트웨어 검증 보고서 (갱신)
소프트웨어 통합 및 시험	소프트웨어 엘리먼트를 통합한다. 소프트웨어 아키텍처 설계가 임베디드 소프트웨어에 의해 구현된다는 것을 증명한다.	• 하드웨어 소프트웨어 인터페이스 명세서 • 소프트웨어 아키텍처 설계 명세서 • 안전 계획(갱신) • 소프트웨어 단위 구현 • 소프트웨어 검증 계획(갱신) • 소프트웨어 검증 명세서(갱신) • 소프트웨어 검증 보고서(갱신)	• 소프트웨어 검증 계획 (갱신) • 소프트웨어 검증 명세서(갱신) • 임베디드 소프트웨어 • 소프트웨어 검증 보고서(갱신)
소프트웨어 안전 요구사항검증	임베디드 소프트웨어가 소프트웨어 안전 요구사항을 충족하는 것을 증명한다.	• 소프트웨어 아키텍처 설계 명세서 • 안전 계획(갱신) • 소프트웨어 안전 요구사항 명세서(갱신) • 소프트웨어 검증 계획(갱신) • 소프트웨어 검증	• 소프트웨어 검증 계획 (갱신) • 소프트웨어 검증 명세서 (갱신) • 소프트웨어 검증 보고서 (갱신)

		명세서(갱신) • 소프트웨어 검증 보고서(갱신) • 통합 시험 보고서	

소프트웨어 개발 단계에서 신규, 갱신되는 산출물은 표 6 의 14 개로 요약될 수 있다. 신규는 해당 산출물이 해당 단계에서 처음 만들어지는 것을 의미하고 갱신은 기존 산출물이 해당 단계를 거치면서 보강되는 것을 의미한다.

표 6. 산출물별 수명주기 단계에 따른 작업

번호	산출물	착수	요구사항 명세	아키텍처 설계	단위 설계 구현	단위 시험	통합 시험	요구사항 검증
1	안전 계획	갱신		갱신				
2	소프트웨어 검증 계획	신규	갱신			갱신	갱신	갱신
3	모델링과 프로그래밍 언어를 위한 설계 및 코딩 지침	신규						
4	도구 적용 지침	신규						
5	소프트웨어 안전 요구사항 명세서		신규	갱신				
6	하드웨어 소프트웨어간 인터페이스 명세서		갱신					
7	소프트웨어 검증 보고서		신규	갱신	갱신	갱신	갱신	갱신
8	소프트웨어 아키텍처 설계 명세서			신규				
9	안전 분석 보고서			신규				
10	종속 고장 분석 보고서			신규				
11	소프트웨어 단위 설계 명세서				신규			
12	소프트웨어 단위 구현				신규			
13	소프트웨어 검증 명세서					신규	갱신	갱신
14	임베디드 소프트웨어						신규	

2. ASIL 에 따른 소프트웨어 개발 방법

ISO 26262 에서는 ASIL 에 따라서 소프트웨어 개발 활동에서 적용해야 하는 방법을 각 수행 단계별로 표 형태로 제시한다. 표에서 연속 항목의 경우 모든 방법이 ASIL 에 따라 권장하는 대로 적용되어야 한다. 표에 포함된 방법 이외의 다른 방법이 개발에 적용되는 경우 해당 요구사항을 충족한다는 근거가 제시되어야 한다. ASIL 권장수준이 다르게 표기된 방법들에 대해서는, 상위의 권장사항이 표기된 방법을 우선적으로 선택한다. 각 방법의 경우 해당 방법을 사용하기 위한 권장사항은 ASIL 에 따라 다르며 다음과 같이 매우 권장, 권장, 권장사항 없음으로 분류된다.

- ++"는 해당 방법이 식별된 ASIL 에 대해 매우 권장된다는 것을 나타낸다.
- "+"는 해당 방법이 식별된 ASIL 에 대해 권장된다는 것을 나타낸다.
- "o"는 해당 방법이 식별된 ASIL 에 대해 권장사항이 없다는 것을 나타낸다.

본 가이드는 ASIL B 등급까지 적용되는 방법들에 대한 설명을 포함한다. 본 가이드의 설명 대상은 아래 표 7 에서 회색으로 표시하였다. 본 가이드에 설명된 방법들의 용어는 ISO 26262 KS 표준을 참고하였다.

표 7. ISO 26262 개발 방법 중 본 가이드의 설명 범위

ID	항목	방법/주제		ASIL			
				A	B	C	D
06_001	모델링과 코딩지침에 포함되는 주제	1a	낮은 복잡도 강제 a	++	++	++	++
06_002		1b	언어의 서브셋 사용 b	++	++	++	++
06_003		1c	강한 타입 체크 강제 c	++	++	++	++
06_004		1d	방어적 구현 기법 사용	o	+	++	++
06_005		1e	입증된 설계 원리 사용	+	+	+	++
06_006		1f	모호하지 않은 그래픽 표현 사용	+	++	++	++
06_007		1g	스타일 지침 사용	+	++	++	++
06_008		1h	명명 규칙 사용	++	++	++	++

06_009	소프트웨어 아키텍처 설계를 위한 표기법	1a	비정형 표기법	++	++	+	++
06_010		1b	준정형 표기법	+	++	++	++
06_011		1c	정형 표기법	+	+	+	+
06_012	소프트웨어 아키텍처 설계 원리	1a	소프트웨어 컴포넌트의 계층적 구조	++	++	++	++
06_013		1b	소프트웨어 컴포넌트의 제한된 크기	++	++	++	++
06_014		1c	인터페이스의 제한된 크기	+	+	+	+
06_015		1d	각 소프트웨어 컴포넌트 내 높은 응집도(cohesion)	+	++	++	++
06_016		1e	소프트웨어 컴포넌트 사이 제한된 결합도(coupling)	+	++	++	++
06_017		1f	적합한 스케줄링 속성	++	++	++	++
06_018		1g	인터럽트의 제한된 사용	+	+	+	++
06_019	소프트웨어 아키텍처 수준의 오류 검출을 위한 메커니즘	1a	입출력 데이터 범위 확인	++	++	++	++
06_020		1b	타당성 확인(plausibility check)	+	+	+	++
06_021		1c	데이터 오류 검출	+	+	+	+
06_022		1d	외부 모니터링 기능	o	+	+	++
06_023		1e	제어 흐름 모니터링	o	+	++	++
06_024		1f	다양한 소프트웨어 설계	o	o	+	++
06_025	소프트웨어 아키텍처 수준의 오류 처리를 위한 메커니즘	1a	정적 복구 메커니즘	+	+	+	+
06_026		1b	적절한 데그러데이션 (graceful degradation)	+	+	++	++
06_027		1c	독립적 병렬 중복	o	o	+	++
06_028		1d	데이터 정정 코드	+	+	+	+
06_029	소프트웨어 아키텍처 설계 검증 방법	1a	설계에 대한 워크쓰루(walk-through)	++	+	o	o
06_030		1b	설계에 대한 인스펙션(inspection)	+	++	++	++
06_031		1c	동적 부품의 설계에 대한 시뮬레이션	+	+	+	++
06_032		1d	시제품 생성	o	o	+	++
06_033		1e	정형 검증	o	o	+	+
06_034		1f	제어 흐름 분석	+	+	++	++

06_035		1g	데이터 흐름 분석	+	+	++	++
06_036	소프트웨어 단위 설계를 위한 표기법	1a	자연 언어	++	++	++	++
06_037		1b	비정형 표기법	++	++	+	+
06_038		1c	준정형 표기법	+	++	++	++
06_039		1d	정형 표기법	+	+	+	+
06_040	소프트웨어 단위 설계 및 구현을 위한 설계 원리	1a	서브프로그램과 함수에서 하나의 진입점과 하나의 종료점	++	++	++	++
06_041		1b	동적 개체 또는 변수를 사용하지 않음, 혹은 그렇지 않으면 동적 변수 생성시 온라인 시험	+	++	++	++
06_042		1c	변수 초기화	++	++	++	++
06_043		1d	변수 이름을 다목적으로 사용하지 않음	+	++	++	++
06_044		1e	전역 변수를 사용하지 않음, 만약 사용해야 한다면 그 사용에 대해 정당화함	+	+	++	++
06_045		1f	포인터의 제한된 사용	o	+	+	++
06_046		1g	암시적 형 변환 없음	+	++	++	++
06_047		1h	숨겨진 데이터 흐름이나 제어 흐름 없음	+	++	++	++
06_048		1i	무조건적 점프 없음	++	++	++	++
06_049		1j	재귀 없음	+	+	++	++
06_050	소프트웨어 단위 설계 및 구현 검증 방법	1a	워크쓰루	++	+	o	o
06_051		1b	인스펙션	+	++	++	++
06_052		1c	준정형 검증	+	+	++	++
06_053		1d	정형 검증	o	o	+	+
06_054		1e	제어 흐름 분석	+	+	++	++
06_055		1f	데이터 흐름 분석	+	+	++	++
06_056		1g	정적 코드 분석	+	++	++	++
06_057		1h	의미적 코드 분석 (semantic code analysis)	+	+	+	+
06_058	소프트웨어	1a	요구사항 기반 시험	++	++	++	++

06_059	단위 시험 방법	1b	인터페이스 시험	++	++	++	++
06_060		1c	결함 주입 시험	+	+	+	++
06_061		1d	자원 사용 시험	+	+	+	++
06_062		1e	모델과 코드 간 비교 시험 (back-to-back test), 해당되는 경우	+	+	++	++
06_063	소프트웨어 단위 시험을 위한 시험 케이스 생성 방법	1a	요구사항 분석	++	++	++	++
06_064		1b	동치 클래스(equivalence class)의 생성 및 분석	+	++	++	++
06_065		1c	경계 값(boundary value)의 분석	+	++	++	++
06_066		1d	오류 추측	+	+	+	+
06_067	소프트웨어 단위 수준의 구조적 커버리지 지표	1a	구문 커버리지	++	++	+	+
06_068		1b	분기 커버리지	+	++	++	++
06_069		1c	MC/DC(수정된 조건/결정 커버리지)	+	+	+	++
06_070	소프트웨어 통합 시험 방법	1a	요구사항 기반 시험	++	++	++	++
06_071		1b	인터페이스 시험	++	++	++	++
06_072		1c	결함 주입 시험	+	+	++	++
06_073		1d	자원 사용 시험	+	+	+	++
06_074		1e	해당되는 경우 모델과 코드 간 비교 시험(back-to-back test)	+	+	++	++
06_075	소프트웨어 통합 시험을 위한 시험 케이스 생성 방법	1a	요구사항 분석	++	++	++	++
06_076		1b	동치 클래스(equivalence class)의 생성 및 분석	+	++	++	++
06_077		1c	경계 값(boundary value)의 분석	+	++	++	++
06_078		1d	오류 추측	+	+	+	+
06_079	소프트웨어 아키텍처 수준의 구조 커버리지 지표	1a	함수(function) 커버리지	+	+	++	++
06_080		1b	호출(call) 커버리지	+	+	++	++

06_081	소프트웨어 안전 요구사항 검증 실시를 위한 시험 환경	1a	Hardware-In-the-Loop	+	+	++	++
06_082		1b	ECU 네트워크 환경	++	++	++	++
06_083		1c	차량	++	++	++	++

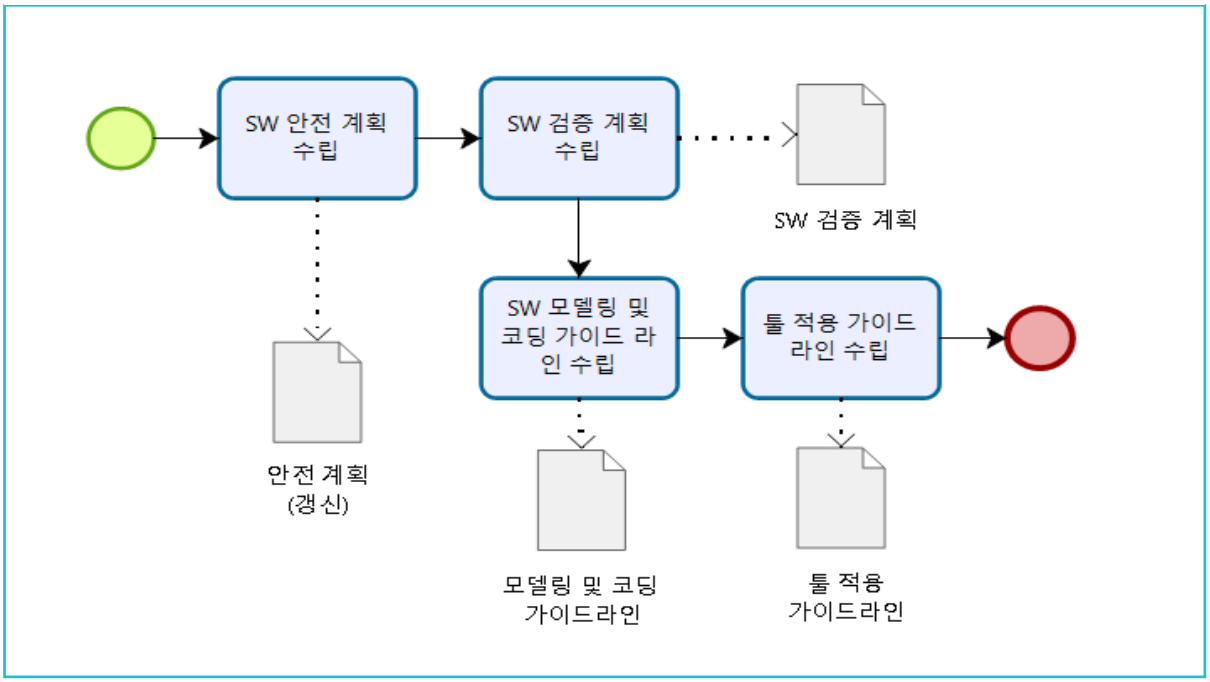
3. 소프트웨어 개발 수명주기 단계별 가이드

3.1. 소프트웨어 개발 착수

가. 수행 활동

소프트웨어 개발 착수 단계는 소프트웨어 개발 첫 번째 단계로 소프트웨어 전체 개발 단계에서 수행하는 개발 활동 및 검증 활동에 대해서 계획을 수립한다. 그림 16 과 같이 소프트웨어 개발 계획, 소프트웨어 검증 계획, 모델링 및 코딩 가이드라인, 툴 적용 가이드인 작성 활동으로 구성된다.

그림 16. 활동 흐름 – 소프트웨어 개발 착수



소프트웨어 개발 계획은 ISO 26262 파트 6 에 정의된 기능안전 소프트웨어 개발 단계를 기준으로 수립하지만 아이템의 크기, 복잡도, 재사용 컴포넌트의 사용 여부에 따라서 수명주기 단계를 생략, 조정, 추가하여 적용할 수 있다. 개발 수명주기를 조정하는 경우에는 ISO 26262 파트 2, 6.4.5 안전활동 조정에 따라서 조정이 적합하고 충분한지 근거를 포함해야 한다. SW 개발 활동 계획은 안전 계획서에 포함하여 작성하거나 별도 문서로 분리하여 작성할 수 있다.

소프트웨어 검증 계획은 각 개발 활동에서 작성되는 산출물들 중에서 검증할 산출물의 내용, 검증에 사용되는 방법, 검증 통과 기준 등을 포함한다. 검증 계획 항목은 ISO 26262 Part 8, 9 절 검증에서 요구하는 방법을 따르며, 본 가이드의 4.4 검증을 참조한다.

착수 단계에서는 소프트웨어 개발에 사용하는 도구(Tool)도 선정하여 요구사항 분석부터 설계, 시험에 이르는 전체 개발 활동에 사용하는 도구들에 대한 적용 가이드라인을 수립한다. 도구는 상용 또는 오픈 소스, 자체 개발한 도구들을 사용할 수 있으며 도구 선정 시에는 ISO 26262 파트 8 소프트웨어 도구 사용의 신뢰 확보에 따라서 ISO 26262 적용 적합성을 판단하여 선정한다. 도구의 적합성 판단은 4.6 소프트웨어 도구 사용의 신뢰 확보를 참조한다.

나. 수행 활동 구성

구 분	설 명
목 적	● 소프트웨어 개발 단계 동안 수행해야 할 기능 안전 활동을 계획한다.
시작 기준	● 프로젝트 계획이 작성되고 승인되었다. ● 안전 계획이 작성되고 승인되었다. ● 기술안전 개념이 도출되고 시스템 설계 명세서에 반영되었다. ● 시스템 설계 명세서가 작성되고 승인되었다.
입력물	● 프로젝트 계획(갱신) ● 안전 계획(갱신) ● 기술안전 개념 ● 시스템 설계 명세서 ● 아이템 통합 및 시험 계획
수행 활동	● 안전 계획 수립 ● SW 검증 계획 수립 ● SW 모델링 및 코딩 가이드라인 수립 ● 툴 적용 가이드라인 수립
출력물	● 안전 계획(갱신) ● 소프트웨어 검증 계획 ● 모델링과 프로그래밍 언어를 위한 설계 및 코딩 지침 ● 툴 적용 가이드라인
완료 기준	● 안전 계획에 소프트웨어 개발 단계 활동이 포함하여 갱신됨 ● 소프트웨어 검증 계획이 작성되고 승인됨 ● 모델링과 프로그래밍 언어를 위한 설계 및 코딩 지침이 작성되고 승인됨 ● 툴 적용 가이드라인이 작성되고 승인됨

다. 역할 및 책임

표 8. 역할 및 책임 – SW 개발 착수

역할 / 활동	안전 관리자	개발자	품질 담당자	프로젝트 관리자
안전 계획 수립	R	I	C	A
SW 검증 계획 수립	I	C	R	A
SW 모델링 및 코딩 가이드라인 수립	I	R	C	A
툴 적용 가이드라인 수립	I	R	C	A

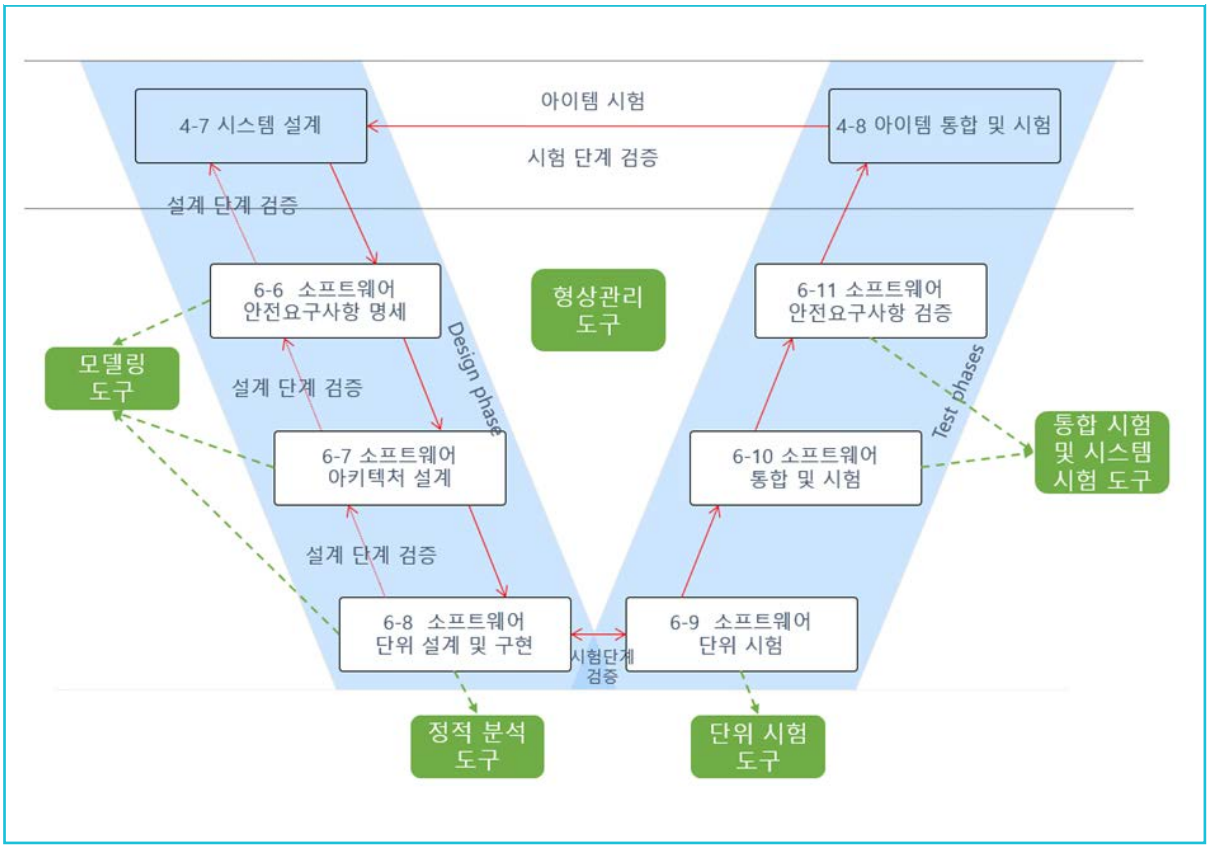
* R: Responsible, 담당자, 해당 업무를 실행하는 주체 및 책임자
* A: Accountable, 의사 결정권자, 해당 업무 최종 승인자
* C: Consulted, 조언자, 업무 협의 또는 의견 제공자
* I: Informed, 공유 대상자, 해당 업무 참조자

라. 산출물

- 안전 계획(갱신) : 소프트웨어 개발 착수 활동에서 수립된 소프트웨어 개발에 필요한 방법, 수명주기 등의 소프트웨어 안전 계획을 포함하여 안전 계획을 갱신한다.
- 소프트웨어 검증 계획 : 소프트웨어 개발 활동에서 수행하는 소프트웨어 검증 계획을 작성한다. 소프트웨어 검증 계획은 ISO 26262 파트 8 의 9 검증에 따라서 작성하며, 본 가이드의 4.4 검증을 참고한다. 검증 계획에는 검증할 작업 산출물과 검증 방법, 검증에 대한 합격 및 불합격 기준, 검증 환경을 포함한다.
- 모델링과 프로그래밍 언어를 위한 설계 및 코딩 지침 : 소프트웨어 개발 활동에서 수행하는 모델링과 설계 및 코딩 지침을 작성한다. 모델링과 설계 및 코딩 지침은 모델링 및 코딩 지침 항목의 주제를 포함한다.

- 도구 적용 지침 : 소프트웨어 개발 전체 활동에서 사용되는 도구에 대한 적용 지침을 수립한다. 소프트웨어 개발에서 사용할 수 있는 도구 목록 및 도구 사용 가이드라인을 포함한다. 도구는 소프트웨어 수준 개발의 V 모델에서 그림 17 과 같이 분류할 수 있다.

그림 17. 소프트웨어 개발 도구 분류



도구 분류별로 표 9 에 사용 가능한 도구들을 예시로 포함하였다. 도구들은 시장에서 구매할 수 있는 모든 도구들을 나열한 것은 아니며 많이 사용하거나 언급되는 도구들을 표시한 것이다. 도구 적용 시에는 ISO 26262 파트 8 소프트웨어 도구 사용의 신뢰성 확보 요구사항에 따라서 도구의 중요도 및 기능안전에 미치는 영향을 평가하여 도구를 사용해야 한다. 도구 사용의 신뢰성 확보 방법은 4.6 소프트웨어 도구 사용의 신뢰성 확보를 참고한다.

표 9. 소프트웨어 개발 도구 예시

구 분	도구 예시
모델링 도구	● DOORS, Rhapsody, Papyrus, MATLAB Simulink
정적 분석 도구	● SonarQube, CodeSonar, CodeScroll, QAC, Kiuwan
단위 시험 도구	● CppUnit, Junit
통합 시험 및 시스템 시험 도구	● VectorCast, HIL Simulator
형상관리 도구	● Git, SubVersion, ClearCase

마. 적용 기법

1) 모델링 및 코딩 지침 항목

표 10. 모델링 및 코딩 지침 포함 항목

주제		ASIL			
		A	B	C	D
1a	**낮은 복잡도 강제**	++	++	++	++
1b	**언어의 서브셋 사용**	++	++	++	++
1c	**강한 타입 체크 강제**	++	++	++	++
1d	방어적 구현 기법 사용	o	+	++	++
1e	입증된 설계 원리 사용	+	+	+	++
1f	**모호하지 않은 그래픽 표현 사용**	+	++	++	++
1g	**스타일 지침 사용**	+	++	++	++
1h	**명명 규칙 사용**	++	++	++	++
++: 매우 권장, +: 권장, o: 권장사항 없음					

- 낮은 복잡도 강제

소프트웨어 복잡도는 일반적으로 순환 복잡도(Cyclomatic complexity) 지표를 사용하며 Thomas J. McCabe[11]가 고안한 지표이다. 순환 복잡도의 계산은 다음 두 가지 계산식을 사용한다. 예를 들면, 함수에서 분기문이 2 개 존재한다면 복잡도는 계산식 2 에 따라서 3 이 된다.

계산

- 계산식 1: 복잡도 V(G) = 선(edge)의 수 – 마디(node)의 수 + 2
- 계산식 2: 복잡도 V(G) = 분기문의 수 + 1

소프트웨어 복잡도는 함수 내부의 제어 흐름에서 분기 및 조건문을 얼마나 포함하고 있는지에 따라 결정된다. 소프트웨어 복잡도가 높을수록 결함 발생 가능성이 증가하여 소프트웨어 안전 및 신뢰성에 영향을 미치기 때문에 일정 수준으로 낮추어야 한다.

ISO 26262 에는 소프트웨어의 복잡도를 일정 수준 이하로 낮추도록 권고하고 있으며 어느 정도가 낮은 수준인지는 언급하지 않지만, HIS Source Code Metrics[12]에서는 복잡도를 평가하는 기준으로 순환 복잡도(Cyclomatic Complexity)를 10 이하로 낮추도록 권고한다. Microsoft 의 코드 품질 규칙에서는 [13] 복잡도를 25 이하로 유지하도록 하고 있다. 개발하는 아이템의 특성에 따라서 적절하게 관리하는 것이 필요하다

- 언어의 서브셋

언어의 서브셋이란 C, C++ 등의 언어 문법이 개발자, 코드 생성자나 컴파일러에 의해 다르게 해석될 수 있는 모호하게 정의된 언어 구성 요소를 배제하기 위해 사용하는 언어 규칙을 정하여

---

[11] T.J. McCabe (1976). A Complexity Measure, IEEE Transactions on Software Engineering (Volume: SE-2, Issue: 4, Dec. 1976)

[12] Kuder, Helmar. (2008), HIS Source Code Metrics Version: 1.3.1

[13] https://msdn.microsoft.com/ko-kr/library/ms182212.aspx

제한하는 것이다. 언어의 서브셋을 사용하면 개발 과정에서 실수할 수 있는 조건문에서의 할당이나 로컬 및 글로벌 변수 사용, 런타임 오류를 발생할 수 있는 문제점들을 미리 방지할 수 있다.

ISO 26262 에서는 C, C++ 언어의 경우 MISRA (Motor Industry Software Reliability Association)에서 개발한 프로그래밍 가이드라인을 준수할 것을 요구하고 있다. MISRA 는 1990 년대 초 영국 정부 프로젝트인 "SafeIT"를 시작으로 안전성이 중요한 소프트웨어 개발을 위한 프로그래밍 표준을 발표하고 있다. 시작할 때 자동차 분야 기업들이 주축이 되어 시작했기 때문에 자동차 소프트웨어 분야에서는 전세계 표준으로 사용되고 있으며 항공, 국방 등 안전성이 중요한 다른 산업 분야에서도 많이 사용되고 있다. C, C++ 언어와 모델링 언어에 대한 가이드라인을 발표하고 있으며 이중에서 C 언어에 대한 가이드라인인 MISRA C 가이드가 가장 많이 사용되고 있다. MISRA C 가이드는 1998 년, 2004 년, 2012 년에 각각 새로운 버전이 발표되었고 이를 각각 MISRA C:1998, MISRA C:2004, MISRA C:2012 로 칭한다. MISRA C 표준들은 MISRA 홈페이지에서 PDF 파일 형태로 구매할 수 있다. 2018 년 기준으로 가장 많이 사용되는 표준은 MISRA C:2004 이며 MISRA C:2012 년 이전 버전에 비해 아래 내용이 추가되었으나 전체적인 내용은 비슷하다.

- C90 에 더해 C99 표준 지원
- 기존 142 개 규칙에 17 개 규칙이 추가되어 총 159 개 규칙으로 변경
- 규칙 회피가 불가능한 Mandatory 규칙 카테고리 추가
- 예제와 설명 방식 개선

앞으로 더 많이 사용될 최신 MISRA C:2012 를 기준으로 살펴보면 (이하 MISRA C), MISRA C 는 총 159 개의 가이드로 구성된다. 159 개 가이드는 크게 16 개의 지침(Directive)과 143 개의 규칙(Rule)로 구분된다. 여기서 지침은 개별 항목의 준수 여부를 소스코드만 가지고 객관적으로 판단하기 어려운 반면에 규칙은 판단 기준이 소스코드 수준에서 명확한 가이드를 의미한다. 예를 들어 "모든 어셈블리 언어 사용은 문서화되어야 한다"는 MISRA C 지침 중 하나로 코드만 가지고 준수 여부를 판단할 수 없고 추가적인 산출물에 대한 검토가 필요하다. 반면에 "루프 카운터 변수는 실수형이어서는 안된다"는 규칙은 소스코드만 검토하면 준수 여부를 판단할 수 있다. 아래 코드는 MISRA C 의 Rule 15.5 "함수는 맨 끝에 단 하나의 출구만 가져야 한다"를 위배한다. 입력

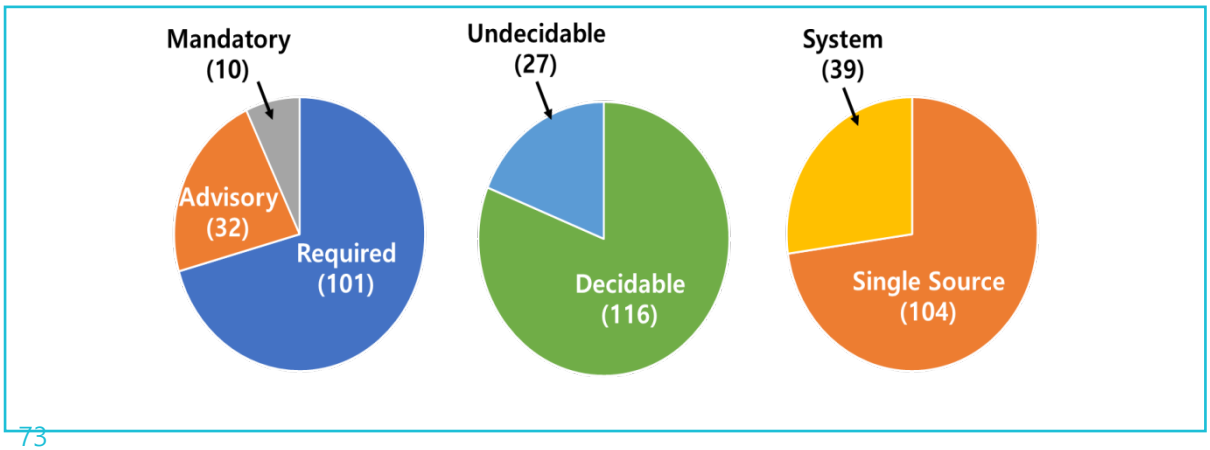
값에 따라 4 번째 줄 혹은 6 번째 줄에서 함수 끝에 이르기 전에 return 문이 실행될 수 있기 때문이다.

```
1: bool_t function(uint16_t n, char *p)
2: {
3:     if (n > MAX)
4:         return false;
5:     if (p == NULL)
6:         return false;
```

각 규칙은 Mandatory, Required, Advisory 3 개의 카테고리로 분류된다. Mandatory 규칙의 경우 무조건 지켜야 한다. Required 의 경우 정당한 사유가 있고 합당하게 문서화를 하면 이해관계자들의 합의에 의해 회피(deviation)가 가능하다. Advisory 의 경우 Required 의 경우보다 쉽게 회의가 가능하다. 143 개의 규칙은 Mandatory (10), Required (101), Advisory (32)개로 구성된다. 이 외에도 규칙 준수가 기계적으로 판단 가능하면 Decidable, 그렇지 않으면 Undecidable 규칙이라 한다. 또한 단일 소스파일만 분석하여 규칙 준수 여부가 판단 가능하면 Single Source 규칙, 아닐 경우 System 규칙이라 칭한다. 아래 그림은 MISRA C 규칙의 세가지 기준의 분류와 규칙들의 분포를 보여준다. 많은 규칙들이 Required, Decidable, Single Source 카테고리에 속하는 것을 알 수 있다.

Decidable 규칙의 경우 정적 분석(Static Analysis) 도구를 이용하여 자동으로 규칙 준수 여부를 체크하는 것이 일반적이다. Undecidable 규칙의 경우 기계적인 판단이 불가능하기 때문에 정적 도구로 완벽하게 체크하는 건 불가능하다.

그림 18. MISRA C 규칙 분류



MISRA C 가이드를 적용할 때 가장 중요한 것은 프로젝트 초기 단계부터 MISRA C 규칙을 적용하고 이를 정적 분석 도구를 이용하여 꾸준하게 체크하는 것이다. 하지만 현실적으로 바쁘게 진행되는 프로젝트 일정상 프로젝트 막바지에 이르러서야 정적 분석을 하고 문제를 해결하려고 하는 경우가 비일비재하다. 이 경우 수만 개 이상의 규칙 위반이 발생하고 이를 개별적으로 처리하는데 엄청난 노력이 들 뿐만 아니라 코드 변경 과정에서 과거에 검증된 로직에 대한 재검증이 필요하고 오히려 코드에 다른 버그가 생기는 원인이 된다. 또한 정적 분석 도구들도 완벽하지 않을 수 있기 때문에 문제가 없는데도 문제가 있다고 지나치게 보수적으로 보고하는 False Alarm 이 다수 발생하고 이것들을 하나하나 검토하여 문제가 없음을 건건이 보고하는데 큰 노력이 소모된다.

- 강한 타입 체크 강제

강한 타입 체크 강제는 다음과 같이 언어에서 변수 선언 시에 변수에 사용할 수 있는 정수, 실수와 같은 타입을 미리 선언하고, 컴파일 시에 변수 할당, 반환 값, 함수 호출에 사용되는 값을 검사하여 적합한지 여부를 확인하도록 한다.

```
int iTest = 10;
float fTest = 1.1;
char cTest = ‘c’;
```

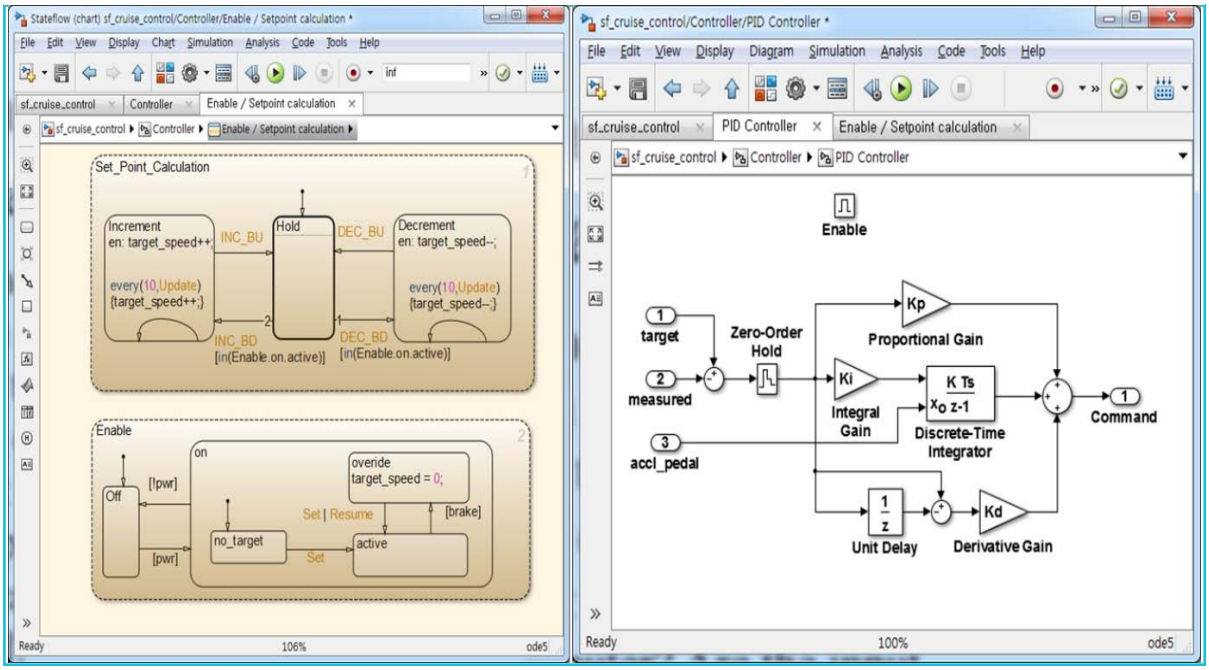
강한 타입 강제는 개발 코드가 의도하지 않게 런 타임 시에 형 변환되어 오류가 발생하지 않기 위한 것으로 C, C++, Java 언어 같은 경우가 해당한다. 반면에 약한 타입 언어는 한 유형의 값을 다른 유형의 값처럼 쉽게 사용할 수 있으므로 강한 타입 체크 강제보다 변수 사용의 제약이 적다. 그러나 변수에 사용되는 값을 예측하기 어려워 소프트웨어 실행 시에 오류 발생 가능성이 높기 때문에 신뢰성이 요구되는 소프트웨어의 개발 언어로는 적합하지 않다.

- 모호하지 않은 그래픽 표현 사용

자동차 분야에서는 Model-Based Design (MBD)라는 개발 방법이 많이 사용되어 왔다. MBD 는 복잡한 문제를 해결하기 위하여 수학적 모델과 가시적인 설계 도구를 이용하는 것을 의미한다. 자동차 분야에서는 제어 로직을 개발할 때 Simulink 나 ASCET 과 같은 GUI 기반의 설계 툴을 많이 이용하는데 이렇게 차트나 다이어그램 형태로 복잡한 로직을 가시적으로 설계한 결과물을

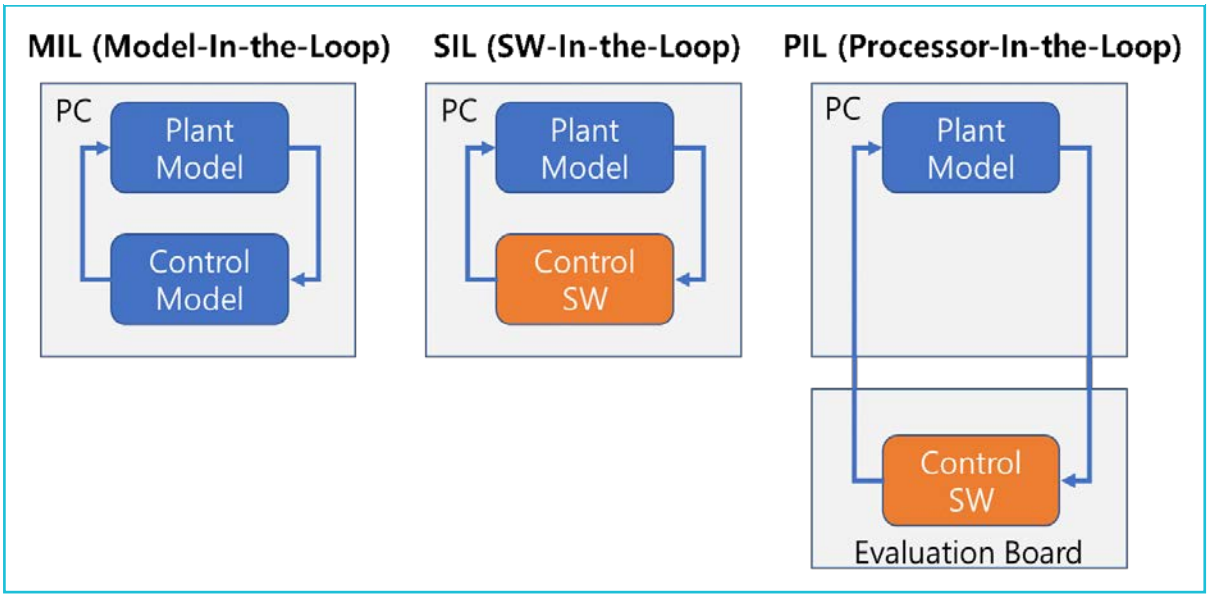
모델이라 한다. 다음 그림은 Simulink 라는 MBD 모두를 이용하여 개발한 크루즈 컨트롤 알고리즘 모델이다. 좌측은 크루즈 컨트롤의 ON/OFF 전환과 목표 속도 계산을 위한 로직이 차트 형태로 가시적으로 표현되어 있다. 여기서 결정된 목표 속도는 우측의 PID 제어 알고리즘에 전달되어 측정 속도의 차이 값을 계산하여 엔진 제어기로 전달할 명령 값을 결정하게 된다.

그림 19. MBD 방식으로 설계된 크루즈 제어 알고리즘 예제 (Source: Mathworks)



위와 같은 가시적인 그래픽 표현으로 알고리즘을 표현하면 자연어로 알고리즘을 기술할 때 생길 수 있는 모호함을 제거할 수 있다. 그렇기 때문에 알고리즘의 스펙을 정확히 기술하는 용도로 사용된다. 여기에 더해 MBD 도구들은 위의 모델을 PC 에서 시뮬레이션 할 수 있는 기능을 제공하는데 이를 Model-In-the-Loop (MIL) 시뮬레이션이라 한다. MILS 를 이용하면 아무런 하드웨어나 코딩 단계 없이 알고리즘에 대한 검증이 가능하다. 시뮬레이션의 정확도를 높이기 위해서 제어 모델을 C 코드로 변환하여 PC 에서 컴파일하여 실행해보는 Software-In-the-Loop (SIL) 시뮬레이션을 수행하는 것도 가능하다. 같은 C 코드라도 데이터 타입별 메모리 크기 등 프로세서 특성에 따라 결과값이 상이할 수 있기 때문에 C 코드를 크로스 컴파일하여 마이크로프로세서에 다운로드한 후 테스트하는 Processor-In-the-Loop (PIL) 시뮬레이션 방법도 많이 사용된다. 위 그림은 MIL, SIL, PIL 시뮬레이션 방법의 개념적인 차이를 보여준다. MIL 방법은 제어 대상과 제어기가 모두 PC 에서 개발된 그래픽 모델을 이용하여 시뮬레이션 된다. SIL 방법은 제어 대상은 MIL 방법과 동일하나 제어기 모델은 코드가 생성되어 PC 에서 컴파일된 후 실행된다. PIL 방법은 제어 코드가 임베디드 보드에서 실행된다. MIL, SIL, PIL 로 갈수록 시뮬레이션 결과가 실제 제어기 환경에 가까워진다.

그림 20. MIL, SIL, PIL 시뮬레이션 방법



위와 같이 개발되고 검증된 제어 모델은 단순히 초기 검증 용도와 문서화 용도로만 사용될 수도 있지만 최근에는 MBD 도구의 자동 코드 생성 기능을 이용하여 생성된 양산용 C 코드를 만들고 이를 기반으로 ECU 구현을 하는 방법이 일반적이다. 이를 위해 MBD 도구는 ISO 26262 와 MISRA C 표준에 맞춘 안전성이 보장되는 양산이 가능한 C 코드를 자동으로 생성하는 기능을 제공한다. 이와 같은 방법은 제어 알고리즘 개발의 스펙 결정 단계부터 설계, 구현에 이르는 일련의 단계들이 하나의 모델에서 출발해서 일관성 있게 통합되어 진행될 수 있다는 장점을 가지고 있다. 이는 ISO 26262 의 안전성 보장과도 깊은 관계가 있으며 ISO 26262-6 Annex B 를 보면 이와 같은 MBD 와 ISO 26262 와의 관계가 설명되어 있다.

- 스타일 지침 사용

스타일 지침은 개발자가 코딩 시에 결함을 줄이기 위해 해야 할 것과 하지 말아야 규칙을 정의한다. 개발자마다 각자의 코딩 스타일을 사용하는 것보다 공통적으로 사용하는 스타일 지침을 사용하여 코드의 가독성, 유지보수성과 호환성을 높일 수 있다. 스타일 지침은 개발 조직에서 자체적으로 정의하여 적용할 수 있지만 산업계에서 사용하는 표준을 적용하는 것이 좋다. 자동차 소프트웨어 개발에서는 MISRA 에서 정의한 MISRA C/C++ 가이드라인과 Barr

그룹에서 정의한 Embedded C Coding Standard[14] 를 적용할 수 있다. Embedded C Coding Standard 의 전체 규칙은 Barr 그룹 홈페이지에서 확인할 수 있다. 일반, 주석, 공백, 모듈, 데이터형, 프로시저, 변수, 문장으로 구분하여 결함을 줄일 수 있는 다양한 규칙들이 포함되어 있다.

스타일 지침의 예로는 코드 리뷰를 효율적으로 수행하기 위해 모든 코드 라인의 최대 글자 수를 80 자로 제한하는 것, 다음과 같이 C 의 연산자 순서 규칙에 의존하지 않고 연산 순서를 명확히 하기 위해 괄호를 사용하는 것들이 있다. C 언어 구문에는 많은 연산자가 사용되고, 복잡해지는 경우에 어떤 순서대로 연산이 되는지 명확하지 않기 때문에 컴파일러가 명확하게 순서를 파악할 수 있도록 괄호를 사용해야 한다. 아래 코드는 복잡한 연산자들 사이에 괄호 사용을 의무화한 스타일 지침이 적용되어 있다.

```
if ((iTest > 0) && (iTest < MAX_DEPTH))
{
    iTestResult = convert_test(iTest);
}
```

- 명명 규칙 사용

명명 규칙은 변수 명 및 함수 명, 모듈 명, 데이터 형에 대해서 개발팀에서 사용할 수 있는 공통의 규칙을 정의해서 사용한다. 자동차 소프트웨어 개발에 사용할 수 있는 명명 규칙으로 BARR 그룹에서 정의한 Embedded C Coding Standard 에서는 다음과 같은 규칙들이 있다.

- 변수명은 "_"로 시작하지 않는다.
- 변수명의 글자수는 31 자 이하로 해야 한다.
- 글로벌 변수명은 'g'로 시작해야 한다. (예: g_offset)
- 퍼블릭 함수명은 모듈명과 "_"로 시작해야 한다. (예: sensor_read())

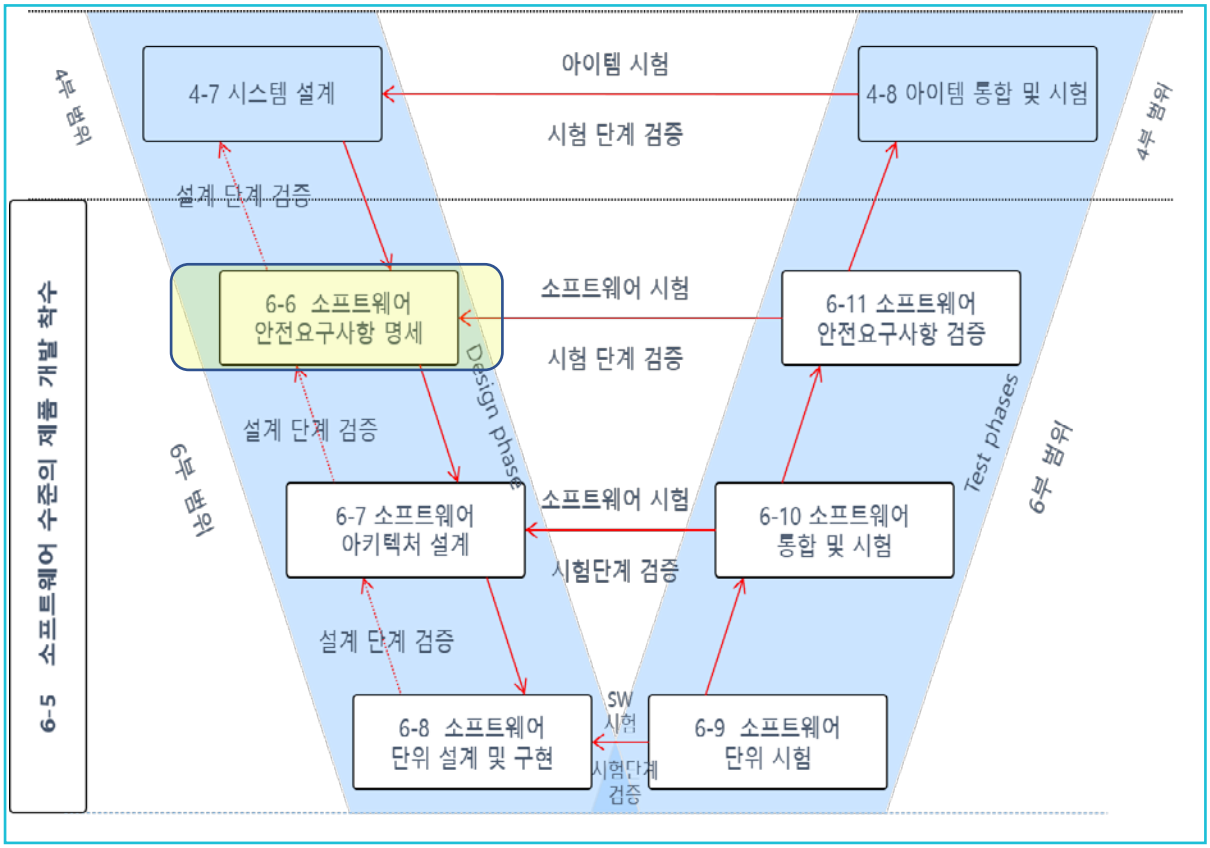
전체 명명 규칙은 Barr 그룹의 홈페이지에 공개된 규칙을 참고한다.

---

[14] Embedded C Coding Standard : https://barrgroup.com/Embedded-Systems/Books/Embedded-C-Coding-Standard

3.2. 소프트웨어 안전 요구사항 명세

그림 21. 소프트웨어 개발 수명 주기 – 소프트웨어 안전요구사항 명세



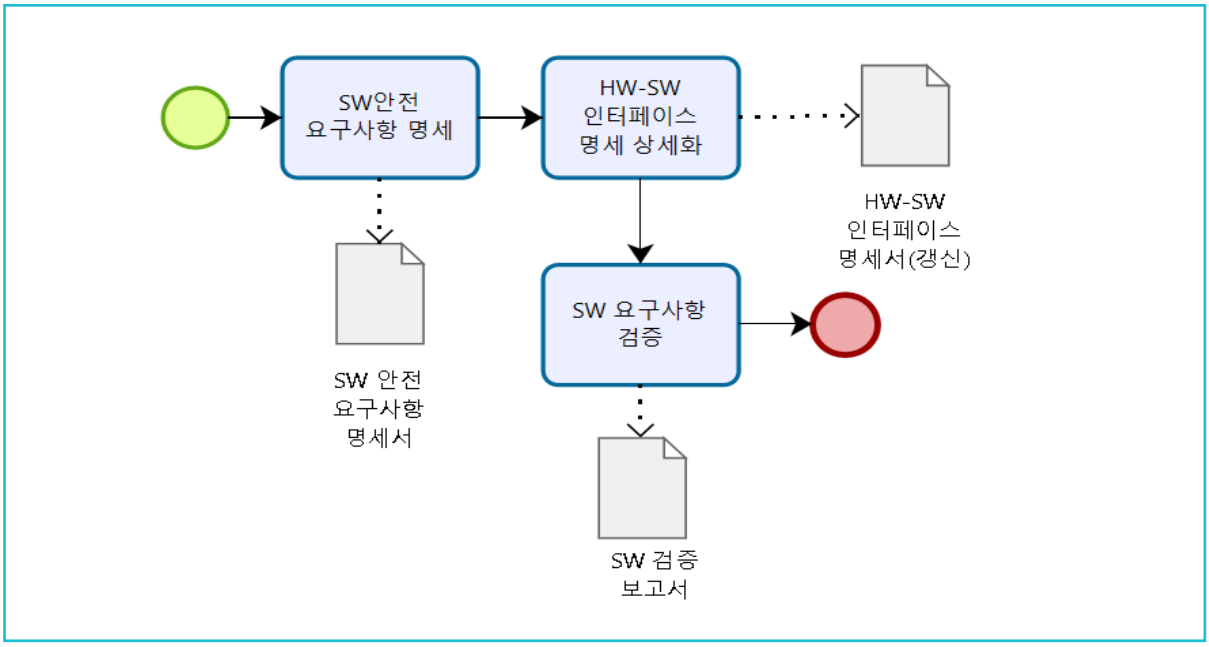
가. 수행 활동

소프트웨어 안전 요구사항 명세 단계는 소프트웨어 개발 수명주기에서 보는 것처럼 시스템 설계 이후에 시스템 설계 항목과 연결되어 수행된다. 시스템 설계 단계를 수행하면서 하드웨어에서 개발해야 할 항목과 소프트웨어에서 개발해야 할 항목이 결정된다. 시스템 설계에서 소프트웨어로 개발해야 할 부분은 기술 안전 개념과 시스템 설계 명세서에 작성되는데 이 설계를 기준으로 하여 소프트웨어 안전 요구사항을 도출한다. 또한, 시스템 설계 단계에서 작성되는 하드웨어와 소프트웨어 인터페이스 요구사항을 구체화한다.

소프트웨어 안전 요구사항 명세는 안전 요구사항 명세서 작성, HW-SW 인터페이스 명세서 상세화, SW 요구사항 검증 활동을 수행한다. 안전 요구사항 명세에는 소프트웨어 고장으로 인해

기술 안전 요구사항을 달성할 수 없는 사항들을 포함한다. 시스템의 안전한 상태를 달성하거나 유지하는 기능, 안전 관련된 하드웨어 엘리먼트의 결함을 검출 및 처리하는 기능, 소프트웨어 자체에서 발생할 수 있는 결함의 검출 및 처리하는 기능, 성능 및 시간 제약 관련된 기능들이 포함된다.

그림 22. 활동 흐름 – 소프트웨어 안전 요구사항 명세



소프트웨어 안전 요구사항 작성 시에는 ISO 26262 파트 8, 6 절의 안전 요구사항 명세 및 관리에 따라서 시스템 설계와 추적이 가능해야 하며, 시스템 설계에서 소프트웨어에 해당하는 기능을 모두 포함해야 한다. 또한, 각 안전 요구사항은 모호하지 않고, 이해가능해야 하며, 검증 가능해야 한다. 본 가이드의 4.1 안전 요구사항 명세 및 관리를 참고한다.

안전 요구사항 명세는 작성된 후에 기술 안전 요구사항과의 부합 및 일치 여부, 시스템 설계와 부합 여부, 하드웨어 소프트웨어 인터페이스의 일관성에 대해서 검증해야 한다. 검증 방법은 ISO 26262 파트 8, 6 절, 9 절에 따르며 수행 방법은 본 가이드의 4.4 검증을 참조한다. 검증 시에는 시스템 및 하드웨어, 소프트웨어 개발 담당자가 함께 참여하여 시스템 및 하드웨어 측면에서 소프트웨어의 요구사항이 적합한지 검증해야 한다.

나. 수행 활동 구성

구 분	설 명
목 적	● 소프트웨어 안전 요구사항을 도출하고 검증한다.
시작 기준	● 프로젝트 계획이 작성되고 승인되었다. ● 안전 계획이 작성되고 승인되었다. ● 기술안전 개념이 도출되고 시스템 설계 명세서에 반영되었다. ● 시스템 설계 명세서가 작성되고 승인되었다. ● 하드웨어와 소프트웨어 인터페이스 명세서가 작성되고 승인되었다.
입력물	● 기술안전 개념 ● 시스템 설계 명세서 ● 하드웨어와 소프트웨어 인터페이스 명세서 ● 안전 계획(갱신) ● 소프트웨어 검증 계획.
수행 활동	● 소프트웨어 안전 요구사항 명세 ● 하드웨어 소프트웨어 인터페이스 명세 상세화 ● 소프트웨어 안전 요구사항 검증
출력물	● 소프트웨어 안전 요구사항 명세서 ● 하드웨어 소프트웨어 인터페이스 명세서(갱신) ● 소프트웨어 검증 계획서(갱신) ● 소프트웨어 검증 보고서
완료 기준	● 소프트웨어 안전 요구사항이 작성되고 승인됨 ● 하드웨어 소프트웨어 인터페이스 명세서가 갱신되고 승인됨 ● 소프트웨어 검증 계획서가 갱신되고 승인됨 ● 소프트웨어 검증 보고서가 작성되고 승인됨

다. 역할 및 책임

표 11. 역할 및 책임 – 소프트웨어 안전 요구사항 명세

역할 / 활동	안전 관리자	개발자	검증 담당자	품질 담당자	프로젝트 관리자
소프트웨어 안전 요구사항 명세	I	R	I	C	A

하드웨어 소프트웨어 인터페이스 명세 상세화	I	R	I	C	A
소프트웨어 안전 요구사항 검증	I	I	R	C	A

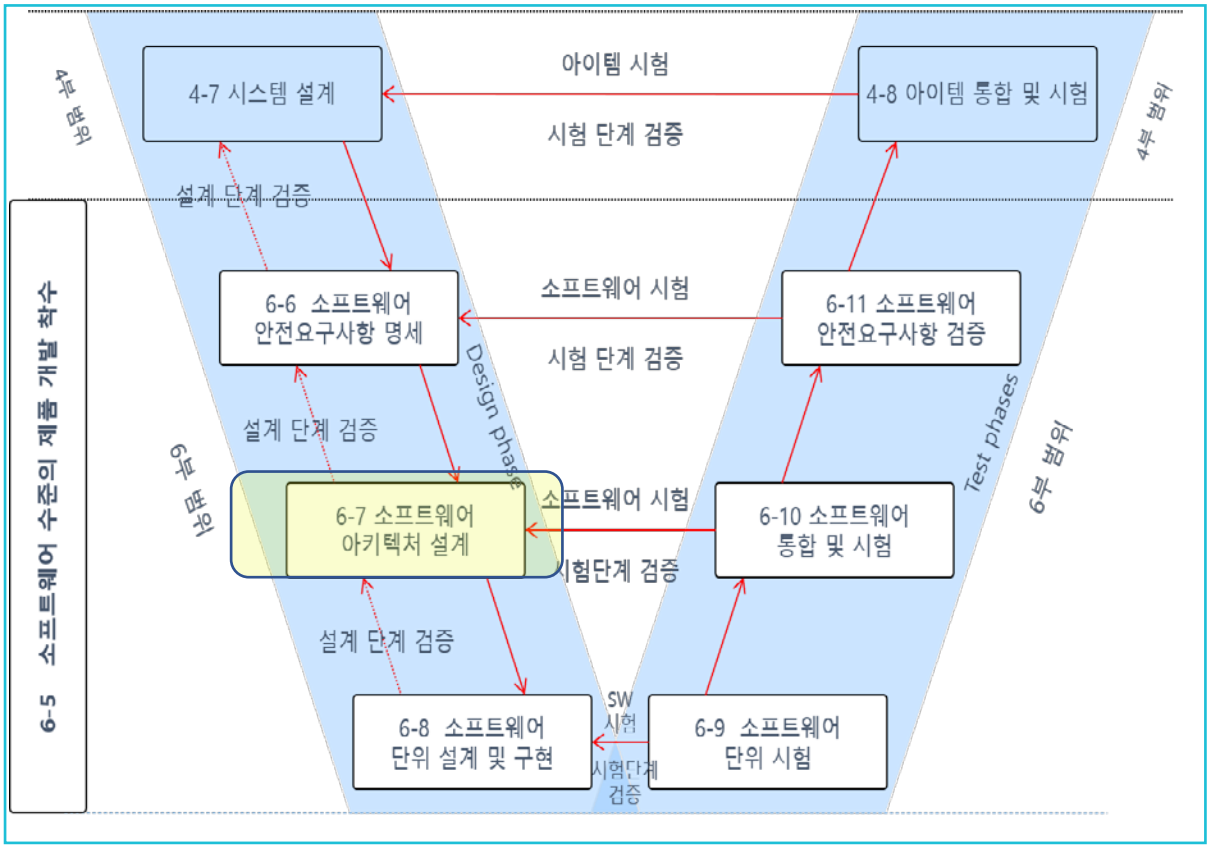
* R: Responsible, 담당자, 해당 업무를 실행하는 주체 및 책임자
* A: Accountable, 의사 결정권자, 해당 업무 최종 승인자
* C: Consulted, 조언자, 업무 협의 또는 의견 제공자
* I: Informed, 공유 대상자, 해당 업무 참조자

라. 산출물

- 소프트웨어 안전 요구사항 명세서 : 기술 안전 개념과 시스템 설계 명세서를 기반으로 도출된 소프트웨어 안전 요구사항을 포함한다. 각 요구사항은 시스템 설계의 ASIL 과 일치하도록 작성해야 한다.
- 하드웨어 소프트웨어 인터페이스 명세서(갱신) : 시스템 개발 단계에서 작성한 하드웨어 소프트웨어 인터페이스를 소프트웨어 안전 요구사항을 도출하며서 해당 기능과 하드웨어의 제어에 맞도록 구체화하여 갱신한다.
- 소프트웨어 검증 계획서(갱신) : 소프트웨어 개발 단계에서 작성된 소프트웨어 검증 계획을 소프트웨어 안전 요구사항 명세서에 대해서 수행하도록 수행 담당자와 일정, 방법, 환경 등을 포함하여 갱신한다.
- 소프트웨어 검증 보고서 : 소프트웨어 안전 명세서에 대한 검증 수행 결과에 대한 보고서를 작성한다. 소프트웨어 검증은 기술 안전 요구사항과의 부합 및 일치 여부, 시스템 설계와 부합 여부, 하드웨어 소프트웨어 인터페이스의 일관성을 비롯해서 요구사항의 명확성, 이해 가능성, 실행 가능성 등을 검증한 결과를 포함한다.

3.3. 소프트웨어 아키텍처 설계

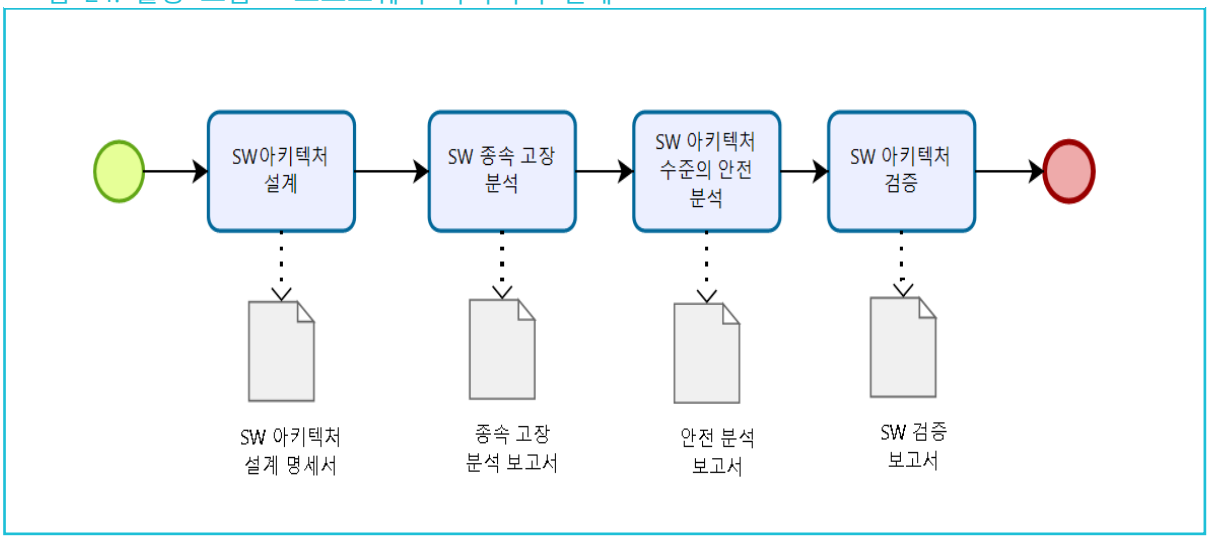
그림 23. 소프트웨어 개발 수명 주기 – 소프트웨어 아키텍처 설계



가. 수행 활동

소프트웨어 아키텍처는 소프트웨어를 컴포넌트로 분할하여 소프트웨어 요구사항을 어떤 컴포넌트에서 수행할지 해당 기능을 할당하고, 소프트웨어 내부 구조를 정의한 것이다. 소프트웨어 안전 요구사항 명세서에 도출된 기능 및 여러 가지 제한 요건을 반영하여 아키텍처를 구성한다. 소프트웨어 아키텍처는 다음 단계인 소프트웨어 구현의 도면 역할을 한다. 소프트웨어 아키텍처 설계는 SW 아키텍처 설계, SW 종속고장 분석, SW 아키텍처 수준의 안전 분석, SW 아키텍처 검증 활동 순서로 수행한다.

그림 24. 활동 흐름 – 소프트웨어 아키텍처 설계



소프트웨어 아키텍처 설계는 소프트웨어 컴포넌트 상호 작용은 계층 구조로 표현하며, 컴포넌트 상호 작용에서 인터페이스와 데이터 경로와 같은 정적인 측면과 흐름 순서 및 타이밍과 같은 동적인 측면을 포함해야 한다. 설계 표현 방법은 ASIL 레벨에 따라서 비정형, 준정형, 정형 표기로 구분하는데 ASIL B 에서는 비정형 표기법과 준정형 표기법이 주로 사용된다.

소프트웨어 아키텍처를 설계하는 동안 다음 사항을 고려해야 한다. (ISO 26262-6, 7.4.2)

- 소프트웨어 아키텍처 설계의 검증 가능성
- 설정 가능한 소프트웨어의 적합성
- 소프트웨어 단위의 설계 및 구현의 타당성
- 소프트웨어 통합 시험 동안의 소프트웨어 아키텍처에 대한 시험 가능성
- 소프트웨어 아키텍처 설계의 유지 보수성

소프트웨어 아키텍처 설계는 다음 사항을 포함해야 한다. (ISO 26262-6, 7.4.5)

- 소프트웨어 컴포넌트의 정적 설계 측면
  - 계층적 수준을 포함한 소프트웨어 구조
  - 데이터 처리의 논리적 순서

- 데이터 타입과 그 특성
- 소프트웨어 컴포넌트의 외부 인터페이스
- 소프트웨어의 외부 인터페이스
- 아키텍처와 외부 종속성의 적용범위를 포함한 제약

- 소프트웨어 컴포넌트의 동적 설계 측면
  - 기능 및 행동
  - 프로세스의 제어 흐름 및 동시성
  - 소프트웨어 컴포넌트 간 데이터 흐름
  - 외부 인터페이스에서의 데이터 흐름
  - 시간 제약
- 동적 행동을 결정하기 위한(예를 들어 태스크, 타임 슬라이스, 인터럽트) 동작 모드(예를 들어 전원 켜기, 종료(shut-down), 정상 동작, 보정, 진단)

소프트웨어 아키텍처 설계 단계에서 소프트웨어 안전 요구사항의 구현이 소프트웨어 컴포넌트 간의 간섭 부재나 독립성에 기반하여 구현하는 경우에는 ISO 26262 파트 9, 7 절에 따라서 종속 고장 분석을 수행해야 한다. 또한, 소프트웨어 안전 관련 컴포넌트 식별 및 안전 메커니즘의 효율성을 검증하기 위해 ISO 26262 파트 9, 8 절에 따라서 소프트웨어 아키텍처 수준의 안전 분석을 수행한다. 안전 분석의 결과를 기반으로 소프트웨어 아키텍처 수준에 필요한 소프트웨어 안전 메커니즘을 명시하기 위해 아래 나열된 오류 검출을 위한 메커니즘이 적용되어야 한다. (ISO 26262-6, 7.4.15)

- 소프트웨어에 할당된 기술안전 요구사항을 직접적으로 필요로 하지 않는다면 시스템 행동에 미치는 잠재적 영향을 분석하기 위해 시스템 수준에서 소프트웨어 안전 메커니즘의 사용을 검토한다.
- 소프트웨어 안전 메커니즘이 소프트웨어에 할당된 기술안전 요구사항에 의해 직접적으로 필요로 하지 않으면 소프트웨어 안전 메커니즘이 시스템의 행위에 미치는 잠재적 영향을 분석하기 위해 시스템 수준에서 소프트웨어 메커니즘의 사용을 검토한다.

소프트웨어 아키텍처 설계에 의해 도입된 새로운 위험원이 기존의 안전 목표에 포함되어 있지 않으면 ISO 26262-8, 8 절의 변경관리 프로세스에 따라 위험원 분석 및 리스크 평가에 포함되어 평가되어야 한다. (ISO 26262-6, 7.4.16)

나. 수행 활동 구성

구 분	설 명
목 적	● 소프트웨어 안전 요구사항을 구현하는 소프트웨어 아키텍처 설계를 개발하고 소프트웨어 아키텍처 설계를 검증한다.
시작 기준	● 프로젝트 계획이 작성되고 승인되었다. ● 안전 계획이 작성되고 승인되었다. ● 기술안전 개념이 도출되고 시스템 설계 명세서에 반영되었다. ● 시스템 설계 명세서가 작성되고 승인되었다. ● 하드웨어와 소프트웨어 인터페이스 명세서가 작성되고 승인되었다. ● 모델링과 프로그래밍 언어를 위한 설계 및 코딩 지침이 작성되고 승인되었다. ● 소프트웨어 안전 요구사항 명세서가 작성되고 승인되었다. ● 소프트웨어 검증 계획이 수립되고 승인되었다.
입력물	● 안전 계획(갱신) ● 모델링과 프로그래밍 언어를 위한 설계 및 코딩 지침 ● 하드웨어와 소프트웨어간 인터페이스 명세서 ● 소프트웨어 안전 요구사항 명세서 ● 소프트웨어 검증 계획(갱신)
수행 활동	● 소프트웨어 아키텍처 설계 ● 소프트웨어 종속 고장 분석 ● 소프트웨어 아키텍처 수준의 안전 분석 ● 소프트웨어 아키텍처 검증
출력물	● 소프트웨어 아키텍처 설계 명세서 ● 안전분석 보고서 ● 종속고장 분석 보고서 ● 소프트웨어 검증 보고서
완료 기준	● 소프트웨어 아키텍처 설계 명세서가 작성되고 승인됨 ● 안전 분석 보고서가 작성되고 승인됨 ● 종속고장 분석 보고서가 작성되고 승인됨 ● 소프트웨어 검증 보고서가 작성되고 승인됨

다. 역할 및 책임

표 12. 역할 및 책임 – 소프트웨어 아키텍처 설계

역할 / 활동	안전 관리자	개발자	검증 담당자	품질 담당자	프로젝트 관리자
소프트웨어 아키텍처 설계	I	R	I	C	A
소프트웨어 종속 고장 분석	I	I	R	C	A
소프트웨어 아키텍처 수준의 안전 분석	C	R	I	I	A
소프트웨어 아키텍처 검증	I	I	R	C	A

* R: Responsible, 담당자, 해당 업무를 실행하는 주체 및 책임자
* A: Accountable, 의사 결정권자, 해당 업무 최종 승인자
* C: Consulted, 조언자, 업무 협의 또는 의견 제공자
* I: Informed, 공유 대상자, 해당 업무 참조자

라. 산출물

- 소프트웨어 아키텍처 설계 명세서 : 소프트웨어를 구성하는 컴포넌트를 계층적으로 구조화한다. 안전 요구사항이 각 기능들이 어떤 컴포넌트에서 동작하는지 설계 명세서에 표현한다.
- 안전분석 보고서 : 소프트웨어 아키텍처 설계를 기준으로 안전 매커니즘의 효율성 검증을 위해 FMEA 또는 FTA 안전 분석을 수행한 결과를 포함한다. FMEA, FTA 의 수행 방법은 SW 안전가이드 공통 분야의 FMEA, FTA 항목을 참고한다.
- 종속고장 분석 보고서 : 종속족장 분석은 하나의 소프트웨어 컴포넌트의 고장이 다른 컴포넌트의 고장을 발생시키거나 하나의 원인으로 인해 여러 컴포넌트의 고장이 발생시킬 수 있는 부분에 대해서 식별하는 것이다. ISO 26262 파트 9, 7 절에 따라서 소프트웨어 아키텍처 수준에서 종속 고장 분석을 FTA 또는 FMEA 안전 분석을 수행한 결과를 포함한다.

- 소프트웨어 검증 보고서 : 소프트웨어 아키텍처 설계에 대해서 소프트웨어 안전 요구사항과의 부합, 대상 하드웨어와 호환성, 설계 지침과 부합되는지 검증하여 결과를 보고서에 포함한다.

마. 적용 기법

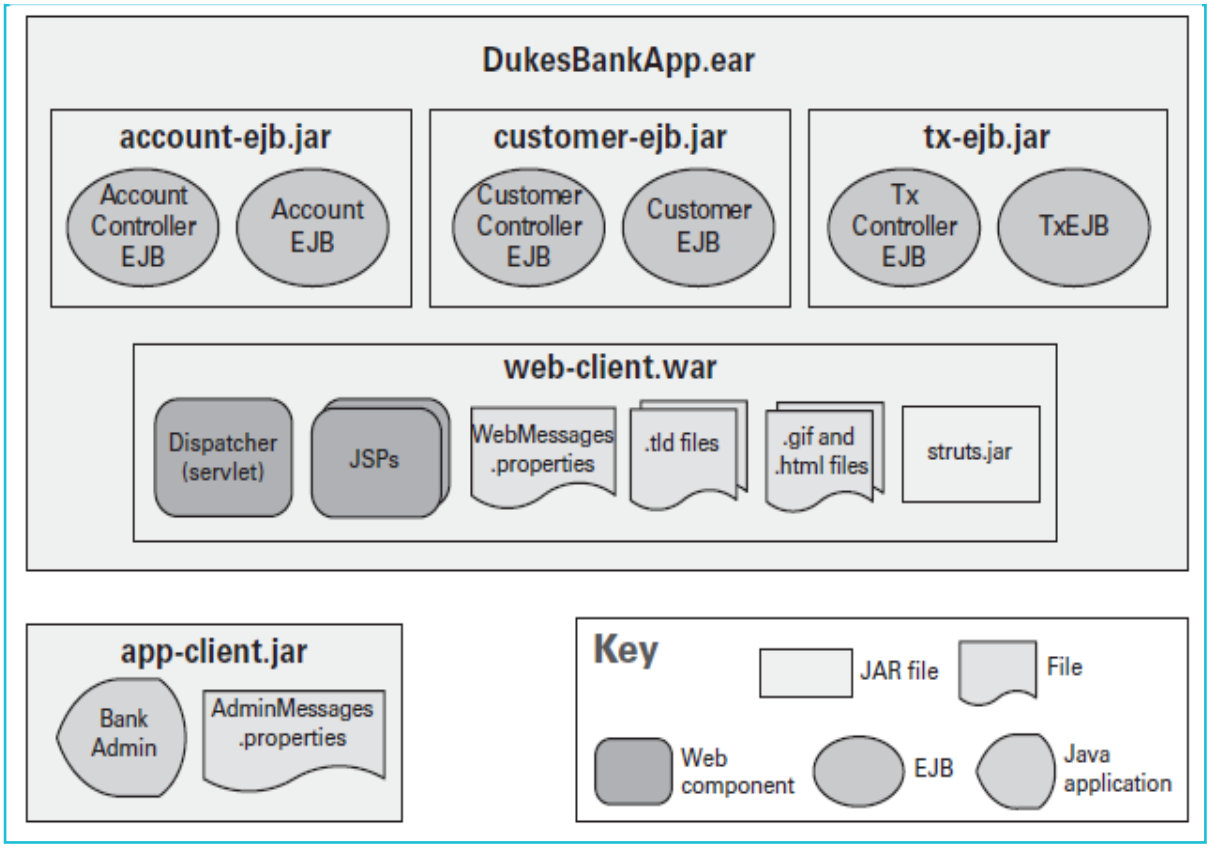
1) 소프트웨어 아키텍처 설계 표기법

방법		ASIL			
		A	B	C	D
1a	**비정형 표기법**	++	++	+	+
1b	**준정형 표기법**	+	++	++	++
1c	강한 타입 체크 강제(Enforcement of strong typing)	+	+	+	+
++: 매우 권장, +: 권장, o: 권장사항 없음					

- 비정형 표기법

비정형 표기법은 표기하는 방식이 완전하게 정의된 구문의 형태를 가지지는 않으나 해석이 가능한 표기법으로 해당 개발 활동 내에서 사용할 목적으로 자체적으로 정의한 표기법을 말한다. 다음 그림과 같이 다이어그램과 해당 다이어그램에 대한 범례를 표시하는 경우이다.

그림 25. 비정형 표기법

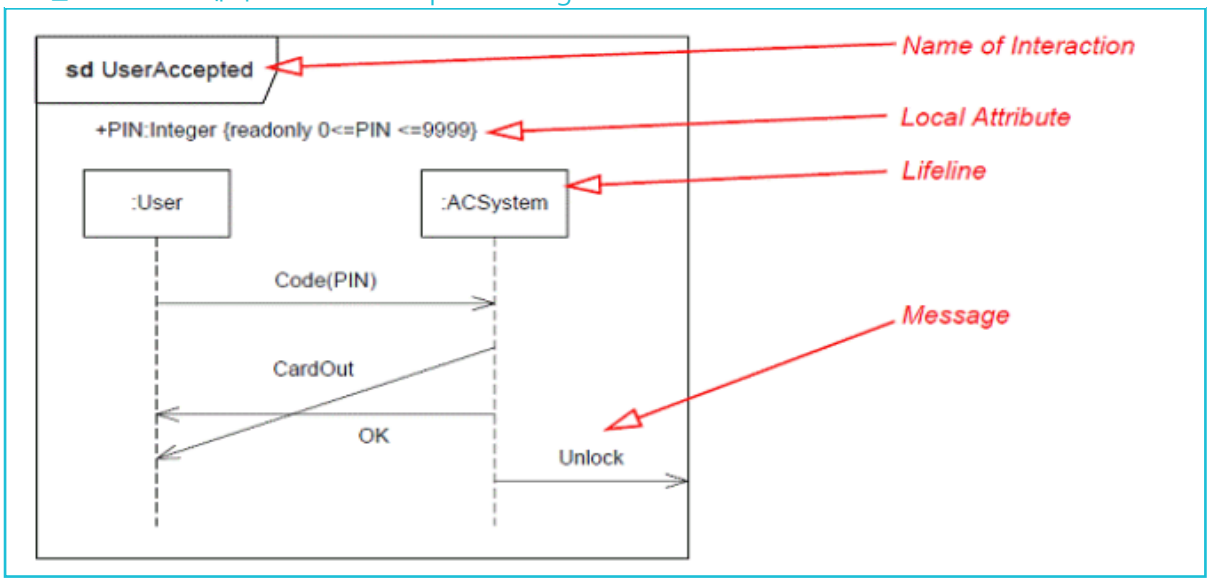


- 준정형 표기법 [15]

준정형 표기법은 아래 그림과 같이 UML, SysML 등 일반적으로 알려진 그래픽 모델링 언어가 해당된다. 각 모델링 요소의 사용법과 의미가 표준으로 정해져 있어서 이해하기 쉽지만, 일부 모호한 부분이 있어서 읽는 사람에 따라서 해석이 달라질 수 있다. 소프트웨어 아키텍처 설계 시에 비정형 표기법보다 준정형 표기법을 사용하는 것이 설계 항목에 대해서 관련자들과 이해하기에 용이하다. UML 은 소프트웨어 모델링에 모든 산업계에서 공통적으로 사용하고 있으며 현재는 버전 2.5.1 까지 공개되어 있다. UML 모델링 구성 요소 및 의미는 UML 웹 사이트에 공개된 문서를 참고한다.

[15] http://www.uml.org/

그림 26. UML 예시 : UML 2.5 Sequence Diagram



2) 소프트웨어 아키텍처 설계 원칙

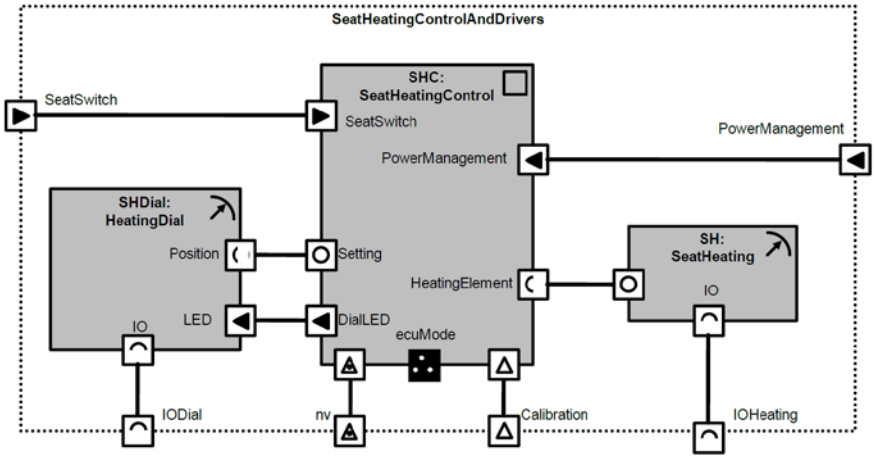
방법		ASIL			
		A	B	C	D
1a	**소프트웨어 컴포넌트의 계층적 구조**	++	++	++	++
1b	**소프트웨어 컴포넌트의 제한된 크기**	++	++	++	++
1c	인터페이스의 제한된 크기	+	+	+	+
1d	**각 소프트웨어 컴포넌트 내 높은 응집도(cohesion)**	+	++	++	++
1e	**소프트웨어 컴포넌트 사이 제한된 결합도(coupling)**	+	++	++	++
1f	**적합한 스케줄링 속성**	++	++	++	++
1g	인터럽트의 제한된 사용	+	+	+	++
++: 매우 권장, +: 권장, o: 권장사항 없음					

- 소프트웨어 컴포넌트의 계층적 구조

추상화 개념을 적용하여 이해하기 쉬운 소프트웨어 구조를 만들기 위해 소프트웨어 컴포넌트를 계층적 구조를 가지도록 설계한다. 이에 따른 대표적 설계 방식으로 AUTOSAR 표준의 콤포짓(Composite) 컴포넌트를 들 수 있다. 콤포짓 컴포넌트는 그 자체가 하나의 소프트웨어

컴포넌트이면서 그 안에 다른 소프트웨어 컴포넌트들을 포함하는 구조를 가지고 있다. 밖에서 보기에는 하나의 독립된 소프트웨어 컴포넌트이지만 그 구현은 몇 개의 다른 소프트웨어 컴포넌트들을 조합하여 만들어져 있다. 예를 들어 그림 27 은 6 개의 포트를 가지는 SeatHeatingControlAndDrivers 컴포넌트의 내부 구조이다. 이 내부에는 3 개의 소프트웨어 컴포넌트들이 연결되어 구성되는데 각각 SHDial:HeatingDial, SHC:SeatHeatingControl, SH:SeatHeating 이다. 이 내부 소프트웨어 컴포넌트들은 자기들끼리 포트로 연결되어 동작하며 최종적으로 콤포짓 컴포넌트의 포트에 연결되어 하나처럼 동작한다. 이러한 계층 구조를 통해서 복잡한 소프트웨어 컴포넌트의 구현을 단순한 복수의 소프트웨어 컴포넌트 개발의 합으로 만들 수 있고 이를 통해 복잡도 문제를 해결하는 것을 목표로 한다.

그림 27. 콤포짓 소프트웨어 컴포넌트 개념 (Source: AUTOSAR Consortium)



- 소프트웨어 컴포넌트의 제한된 크기

소프트웨어 컴포넌트의 크기는 커질수록 컴포넌트 내부의 연결 관계가 복잡해지고 오류 가능성이 증가하고, 유지보수가 어렵다. 소프트웨어 컴포넌트는 가능하면 작은 부분으로 분해해야 한다. SW 안전 공통 가이드의 3.3.3 SW 안전 기록 작성, IEC61508-3 B.9.1 소프트웨어 모듈 크기 제한 (Software module size limit) 항목을 참고한다.

- 각 소프트웨어 컴포넌트 내 높은 응집도(cohesion)

소프트웨어 컴포넌트 응집도를 높이는 것은 컴포넌트에 구현된 기능들이 하나의 작업만 수행하도록 설계하는 것이다. 하나의 컴포넌트 내에서 여러 가지 작업을 수행하도록 설계하면 컴포넌트 내에 관련성이 적은 데이터나 제어 로직이 포함되어 유지보수성이 떨어지고, 결함 발생 가능성이 높아지게 된다. 응집도가 낮으면 소프트웨어 컴포넌트에 중복되는 내용이 들어갈 수

있고 코드 변경 시에 관련된 내용을 수정하지 않음으로 오류가 생길 수 있다. 신뢰성 높은 소프트웨어를 개발하기 위해서는 컴포넌트 내의 연관성이 많은 처리 요소들을 하나의 컴포넌트내에 적절히 배치하여 높은 응집도를 갖도록 해야 한다.

소프트웨어 컴포넌트의 응집도는 응집도의 강도에 따라서 기능적 응집도, 순차적 응집도, 통신적 응집도, 절차적 응집도, 시간적 응집도, 논리적 응집도, 우연적 응집도로 구분한다. 기능적 응집도는 잘 정의된 하나의 기능이 하나의 컴포넌트를 구성하는 경우이며 응집도가 가장 높다. 우연적 응집도는 컴포넌트 내부의 각 구성 요소들이 서로 관련 없는 것들로 구성된 경우이다. 아래와 같은 코드는 논리적으로 유사한 기능을 수행하지만 서로 관계가 밀접하지 않은 논리적 응집도의 예이다.

```
void read_write(int type, int *buf, int size)
{
   switch(type)
   {
      1: line_read(buf, size); break;
      2: line_write(buf, size); break;
   }
```

- 소프트웨어 컴포넌트 사이 제한된 결합도(coupling)

결함도는 컴포넌트 간의 상호 의존하는 정도 또는 두 모듈 사이의 연관 관계를 말한다. 결합도는 컴포넌트의 독립성을 정하는 척도의 하나이고, 컴포넌트 간의 불필요한 인터페이스를 줄이도록 설계하는 것이 필요하다. 결합도가 높으면 컴포넌트 내의 관련 없는 부분 변경으로 인해 소프트웨어 오류가 발생할 수 있는 위험이 증가한다. 기능적으로 독립된 컴포넌트는 다른 모듈과 간단한 인터페이스만을 가지므로 개발이 쉽고 컴포넌트 간의 결함으로 오류가 생길 가능성이 줄어든다. 결합도가 낮을수록 컴포넌트를 수정해도 다른 컴포넌트에 영향을 거의 미치지 않기 때문에 오류가 발생해도 쉽게 발견하여 해결할 수 있다.

소프트웨어 컴포넌트의 결함도는 내용 결합도, 공통 결합도, 제어 결합도, 스템프 결합도, 자료 결합도로 분류할 수 있으며 내용 결함도는 가장 결합도가 높으며 자료 결합도는 가장 낮다.

아래와 같은 코드는 복합 자료 구조를 매개 변수로 전달하여 참조하는 스탬프 결합도를 가지는 함수 모듈의 예이다.

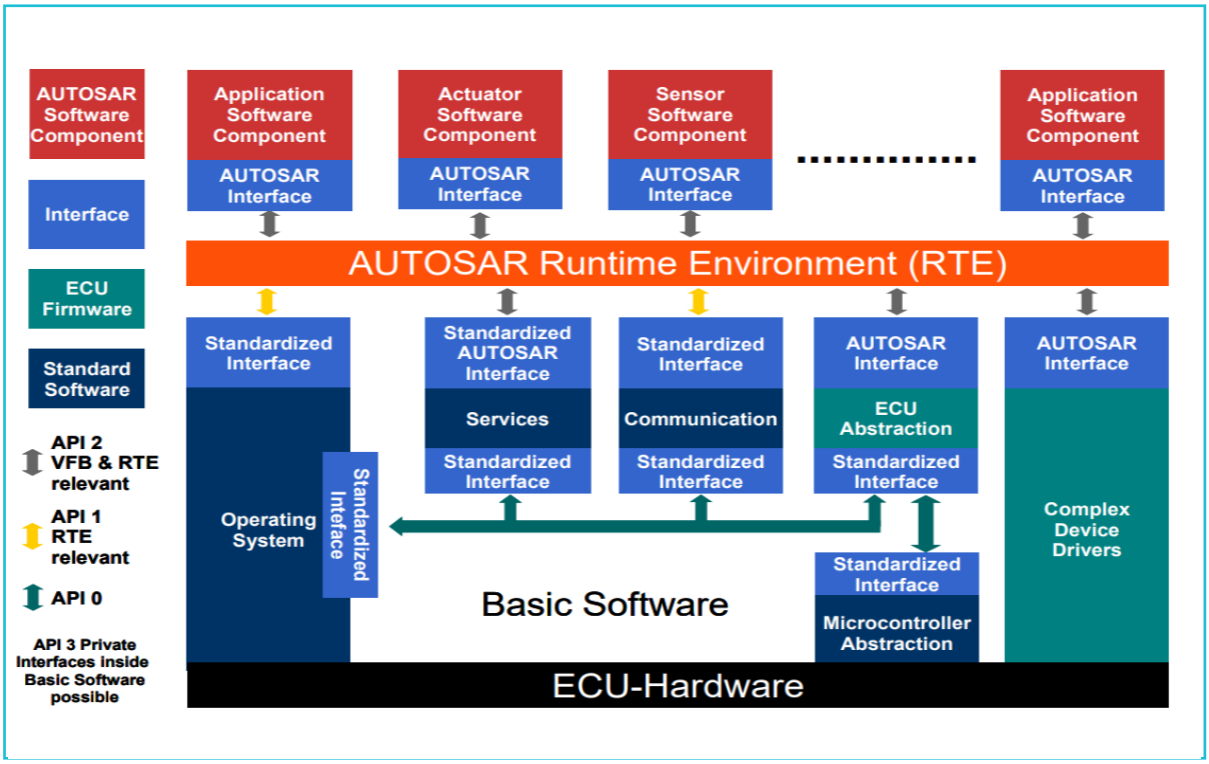
```
struct {
   int x, y;
} X

int add_func(struct X p)
{
   return p.x + p.y;
}
```

- 적합한 스케줄링 속성

소프트웨어 컴포넌트들 간의 스케줄링이 적합하게 설계되었는지를 의미한다. ISO 26262 에서 구체적으로 어떤 스케줄링 방법을 사용해야 한다고 명시하고 있지는 않지만 대부분의 경우 OEM 에서 발주할 때 어떤 버전의 AUTOSAR 운영체제와 개발도구를 사용해야 하는지를 한정하기 때문에 OEM 과 협의하여 이를 준수하여 표준에 맞춘 스케줄링 속성을 결정하면 된다.

AUTOSAR 운영체제는 AUTOSAR 표준의 일부로 기존의 OSEK 실시간 운영체제를 확장하여 개발되었다. AUTOSAR 운영체제라 하면 특정 운영체제 제품을 의미하지 않고 운영체제의 기능을 나열한 스펙을 의미하며 이를 준수하면 AUTOSAR 운영체제라 할 수 있다. 그림 28 은 AUTOSAR 실행환경에서 AUTOSAR 운영체제가 어떻게 다른 요소들과 상호작용하는지 보여준다. AUTOSAR OS 외에도 인포테인먼트 시스템의 경우에는 Linux OS 가 사용되기도 하지만 일반적으로 이 경우는 ISO 26262 절차를 따르지 않는 QM 등급인 경우가 대부분이다. 그렇기 때문에 최근에는 ISO 26262 기반으로 소프트웨어 개발을 한다면 AUTOSAR OS 를 사용한다고 생각해도 무방하다. Linux OS 를 이용하여 ISO 26262 준수를 할 수 있는 방법에 대한 연구는 현재 진행중이며 이 가이드의 범위를 벗어난다.

그림 28. AUTOSAR 실행환경 구조 (Source: www.autosar.org)



소프트웨어 컴포넌트들은 표준 인터페이스 혹은 API (Application Programming Interface)를 통해 운영체제의 기능을 이용하는데 AUTOSAR 운영체제는 다음과 같은 주요 스케줄링 기능들을 제공한다.

- 멀티태스킹을 위한 고정 우선 순위(Fixed-Priority) 스케줄링
- 태스크 단위 선점형(Preemptive), 비선점형(Non-Preemptive) 스케줄링 선택
- 태스크의 실행 주기를 설정할 수 있는 Schedule Table 기능
- 태스크가 정해진 시간 이상을 사용하면 이를 체크하는 보호기능
- 태스크가 정해준 주기보다 더 자주 실행되면 이를 체크하는 보호기능
- 태스크 사이의 공유 자원 보호 기능과 이때 발생할 수 있는 태스크간 우선 순위 역전과 데드락을 방지하는 PCP (Priority Ceiling Protocol) 기능 이외에도 멀티코어 지원 등 AUTOSAR

OS 의 스케줄링 기능에 더한 더 자세한 설명은 AUTOSAR 표준 문서 [16]와 OSEK OS 표준 문서 [17]를 참고한다. 모두 인터넷에 PDF 파일 형태로 공개되어 있다.

OEM 은 사용할 AUTOSAR OS 를 지정하거나 제공하는 역할을 하고 Tier-1 공급사는 소프트웨어 컴포넌트의 통합을 담당한다. 그렇기 때문에 스케줄링에 대한 결정은 일반적으로 Tier-1 공급사의 소프트웨어 통합 담당자가 결정한다. 다수의 제어 알고리즘 개발자는 스케줄링에 대해서 자세히 알기는 힘들지만 아래의 내용을 통해 소프트웨어 통합 과정에 대한 이해를 돕고자 한다. 개별 알고리즘을 담당하는 모듈은 작은 함수로 개발되는데 이를 Runnable 이라 한다. 이 Runnable 은 대부분의 경우 일정한 주기를 가지고 실행되는데 이를 Runnable 주기라 한다. Runnable 코드와 이의 주기가 결정되면 소프트웨어 통합 담당자는 Runnable 들을 같은 주기끼리 묶어서 태스크로 만들고 태스크는 내부에서 자신이 담당하는 Runnable 함수들을 차례대로 호출하게 된다. 이때 Runnable 함수가 태스크 안에서 배치되는 순서를 Runnable Sequence 라 하는데 이 순서가 Runnable 사이의 데이터 흐름 순서와 일치하면 한 주기 내에서 데이터가 Runnable 사이를 자연스럽게 흐르게 되지만 순서가 불일치할 경우 한 주기에서 만들어진 데이터가 다음 주기가 되야 수신측 Runnable 에서 읽히게 된다.

태스크별 우선 순위를 설정할 때는 특별한 사유가 없는 경우 산업 표준인 RM (Rate Monotonic) 방식을 사용한다. RM 방식은 태스크의 수행 주기가 짧을수록 높은 우선순위를 할당하는 방법으로 우선순위가 고정이라고 할 때 최적의 스케줄링 방식임이 증명되어 있다. [18] RM 방식으로 스케줄링을 할 경우 전체 시스템 사용량(Utilization)가 69.3% 미만인 경우 태스크들이 각자의 실행 주기를 지킬 수 있다는 사실이 증면되어 있다. 시스템에 N 개의 태스크가 있고 각 태스크의 주기는 p_i 이고 최악 실행시간은 e_i 라 하면 시스템 사용량 U 는 아래와 같다.

$$U = \sum_{1 \le i \le N} \frac{e_i}{p_i}$$

[16] AUTOSAR Specification of Operating System

[17] OSEK/VDX Operating System Specification

[18] Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1), 46-61.

3) 오류 검출 메커니즘

방법		ASIL			
		A	B	C	D
1a	**입력과 출력 데이터 범위 확인**	++	++	++	++
1b	유효성 확인(plausibility check)	+	+	+	++
1c	데이터 오류 검출	+	+	+	+
1d	외부 모니터링 기능	o	+	+	++
1e	제어 흐름 모니터링	o	+	++	++
1f	다양한 소프트웨어 설계	o	o	+	++
++: 매우 권장, +: 권장, o: 권장사항 없음					

- 입력과 출력 데이터 범위 확인

입력과 출력 데이터 범위 확인은 오류 검출 메커니즘의 하나로 소프트웨어 컴포넌트 또는 소프트웨어 단위에서 다른 컴포넌트 또는 단위를 호출하는 경우에 설계 명세에 정의된 범위에 있는지 확인하는 방법이다. 소프트웨어 오류는 기능을 수행하기 전의 선행 조건인 입력 값과 기능을 수행하고 난 후에 후행 조건인 출력 값 또는 반환 값의 확인을 통해서 많은 부분을 줄일 수 있다. 입력과 출력 데이터 값만 확인하는 것만으로 오류가 발생하는 부분을 쉽게 찾을 수 있다. 입력 값이 잘못되었다면 값을 설정한 곳에서 오류가 있는 것이고, 출력 데이터 값이 잘못되었다면 기능 내부에 오류가 발생하는 경우이다. 예를 들면 값이 null 이 아니어야 하는데 null 인 경우, 수학 함수에서 전달한 값이 함수 안에서 다루는 범위 안에 있는지 확인할 수 있다. 또한, 변수를 초기화하지 않고 사용한다면 정상적인 범위를 벗어나 이상한 값이 들어갈 수 있다. 출력 데이터는 기능 수행의 결과가 적합하고 예상 범위 안에 있는지 확인하거나 코드로 인해 데이터가 손상되거나 이상한 값으로 설정되어 있지 않는지 확인한다. 알고리즘에서 사용하는 파일 핸들과 같은 자원을 정상적으로 반환했는지도 확인할 수 있다.

4) 설계 검증 기법

방법		ASIL			
		A	B	C	D
1a	**설계에 대한 워크쓰루(walk-through)[a]**	++	+	o	o
1b	**설계에 대한 인스펙션(inspection)[a]**	+	++	++	++
1c	설계의 동적 부분에 대한 시뮬레이션	+	+	+	++
1d	시제품 생성 [b]	o	o	+	++
1e	정형 검증	o	o	+	+
1f	제어 흐름 분석 [c]	+	+	++	++
1g	데이터 흐름 분석 [c]	+	+	++	++
++: 매우 권장, +: 권장, o: 권장사항 없음					

- 설계에 대한 워크쓰루(walkthrough)

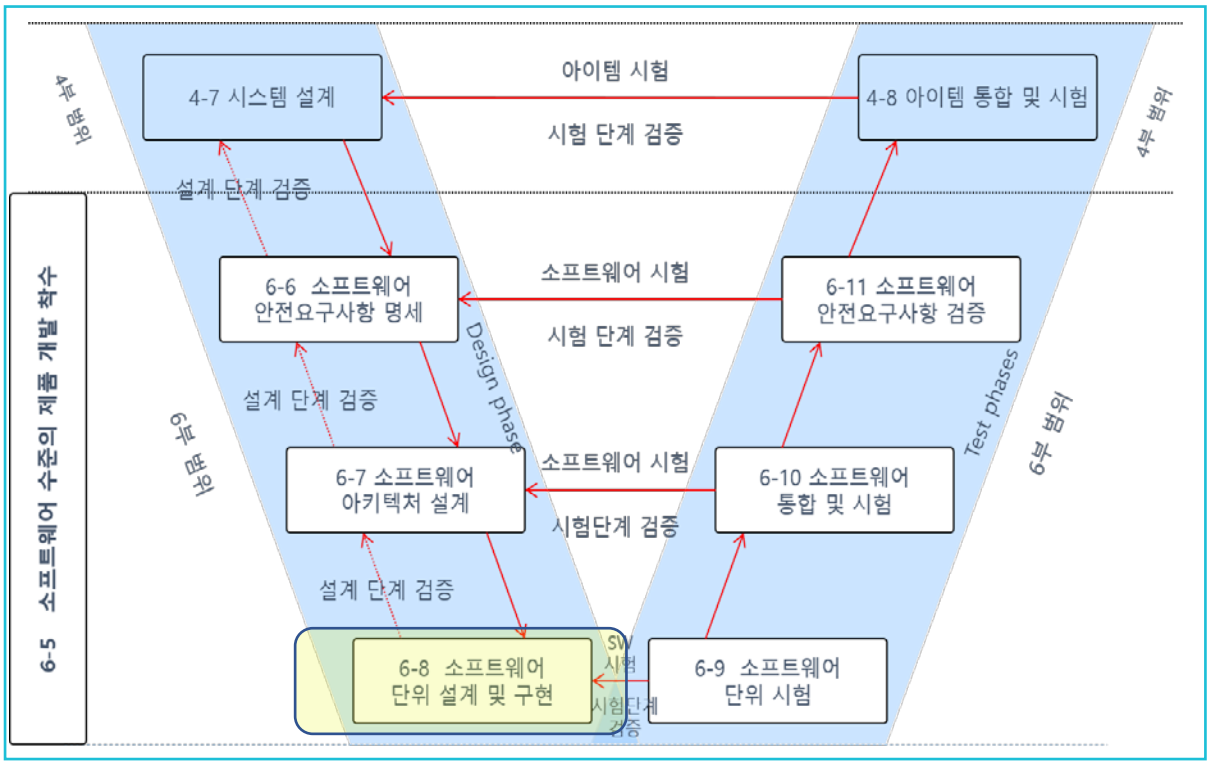
워크쓰루는 설계 명세서의 오류를 발견하는 활동으로 수행 방법은 SW 안전 공통 가이드 5.3.2 SW 안전 기록 작성의 IEC61508-3 B.8.6b 를 참고한다.

- 설계에 대한 인스펙션

인스펙션은 설계 항목의 결함을 발견하는 활동이며 수행 방법은 SW 안전 공통 가이드 5.3.2 SW 안전 기록 작성의 IEC61508-3 B.8.6a 를 참고한다. 인스펙션은 일반적으로 계획, 사전교육, 준비, 인스펙션 회의, 수정, 후속 조치로 나눠진다.

3.4. 소프트웨어 단위 설계 및 구현

그림 29. 소프트웨어 개발 수명 주기 – 소프트웨어 단위 설계 및 구현



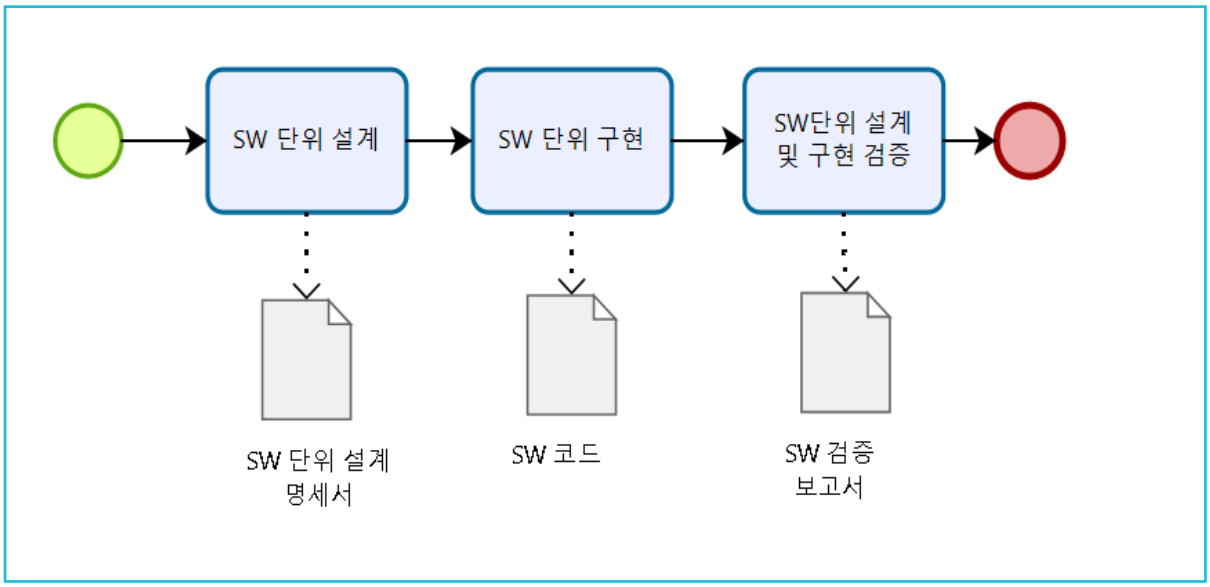
가. 수행 활동

소프트웨어 단위는 단독적으로 시험 대상이 될 수 있는 소프트웨어 아키텍처에서 가장 작은 수준의 소프트웨어 컴포넌트를 말한다. 소프트웨어 아키텍처 설계 단계에서 소프트웨어를 소프트웨어 컴포넌트 또는 단위 수준으로 분할하여 구조화가 된 후에 각 단위별로 내부 설계와 코드 구현을 수행한다. 내부 설계의 상세화 수준은 컴포넌트의 특징을 고려하여 코드로 구현할 수 있을 정도로 작성한다. 단위 내부의 정적인 구조 및 기능적 동작에 대한 동적 설계를 포함하여 설계한다.

소프트웨어 단위 설계 및 구현은 소프트웨어 단위 설계, 소프트웨어 단위 구현, 소프트웨어 단위 설계 및 구현 검증 활동으로 구성된다. 소프트웨어 단위 설계에서는 소프트웨어 아키텍처에서 식별된 각 단위별로 상세 설계를 수행한다. 소프트웨어 단위 구현에서는 단위 상세 설계를

기반으로 임베디드 소스 코드를 작성한다. 소프트웨어 단위 설계 및 구현 검증에서는 작성된 임베디드 소스 코드가 상세 설계를 충족하는지 여부를 검증한다.

그림 30. 활동 흐름 – 소프트웨어 단위 설계 및 구현



소프트웨어 단위 설계 및 구현은 다음 사항을 고려해야 한다. (ISO 26262-6, 8.4.4)

- 소프트웨어 아키텍처 설계에 기반한 소프트웨어 단위 내 서브프로그램과 기능의 올바른 실행 순서
- 소프트웨어 단위 사이 인터페이스의 일관성
- 소프트웨어 단위 사이 및 단위 내에서의 데이터 흐름과 제어 흐름의 정확성
- 단순성
- 가독성 및 이해 가능성
- 강건성
- 소프트웨어 수정의 적합성
- 테스트 가능성

단위 설계 표현은 ASIL 레벨에 따라서 자연 언어, 비정형 표기법, 준정형 표기법, 정형 표기법을 사용하며 ASIL B 레벨에서는 자연언어를 비롯해 비정형 표기법과 준정형 표기법을 사용한다. 각 표기법의 설명은 아래 단위 설계를 위한 표기법 항목을 참고한다.

소프트웨어 단위 설계 및 구현 검증 시에는 다음 검증 사항을 고려해야 한다. (ISO 26262-6, 8.4.5)

- 하드웨어와 소프트웨어간 인터페이스 명세와의 부합
- 추적성을 통해 소프트웨어 단위에 할당된 소프트웨어 안전 요구사항의 충족
- 설계 명세서와 소스 코드와의 부합
- 코딩 지침에 대한 소스 코드의 부합
- 타겟 하드웨어와 소프트웨어 단위 구현의 호환성

나. 수행 활동 구성

구 분	설 명
목 적	● 소프트웨어 단위 설계를 하고 소프트웨어 단위 구현을 하고 검증한다.
시작 기준	● 프로젝트 계획이 작성되고 승인되었다. ● 안전 계획이 작성되고 승인되었다. ● 기술안전 개념이 도출되고 시스템 설계 명세서에 반영되었다. ● 시스템 설계 명세서가 작성되고 승인되었다. ● 하드웨어와 소프트웨어 인터페이스 명세서가 작성되고 승인되었다. ● 모델링과 프로그래밍 언어를 위한 설계 및 코딩 지침이 작성되고 승인되었다. ● 소프트웨어 안전 요구사항 명세서가 작성되고 승인되었다. ● 소프트웨어 검증 계획이 수립되고 승인되었다.
입력물	● 모델링과 프로그래밍 언어를 위한 설계 및 코딩 지침 ● 소프트웨어 아키텍처 설계 명세서 ● 하드웨어와 소프트웨어간 인터페이스 명세서 ● 소프트웨어 안전 요구사항 명세서(갱신) ● 안전 계획(갱신)

	● 소프트웨어 검증 계획(갱신) ● 소프트웨어 검증 보고서(갱신)
수행 활동	● 소프트웨어 단위 설계 ● 소프트웨어 단위 구현 ● 소프트웨어 단위 검증
출력물	● 소프트웨어 단위 설계 명세서 ● 소프트웨어 코드 ● 소프트웨어 검증 보고서
완료 기준	● 소프트웨어 단위 설계 명세서가 작성되고 승인됨 ● 소프트웨어 단위 검증 보고서가 작성되고 승인됨

다. 역할 및 책임

표 13. 역할 및 책임 – 소프트웨어 단위 설계 및 구현

역할 / 활동	안전 관리자	개발자	검증 담당자	품질 담당자	프로젝트 관리자
소프트웨어 단위 설계	I	R	I	C	A
소프트웨어 단위 구현	I	R	I	C	A
소프트웨어 단위 검증	I	I	R	C	A

* R: Responsible, 담당자, 해당 업무를 실행하는 주체 및 책임자
* A: Accountable, 의사 결정권자, 해당 업무 최종 승인자
* C: Consulted, 조언자, 업무 협의 또는 의견 제공자
* I: Informed, 공유 대상자, 해당 업무 참조자

라. 산출물

- 소프트웨어 단위 설계 명세서 : 소프트웨어 아키텍처 설계에서 식별된 컴포넌트별로 정적 및 동적 설계를 포함하여 작성한다.
- 소프트웨어 코드 : 소프트웨어 단위 설계 명세를 기준으로 각 단위별로 소프트웨어 개발 착수 활동에서 작성한 스타일 지침, 명명 규칙을 준수하여 코드를 작성한다.

- 소프트웨어 검증 보고서 : 소프트웨어 단위 설계서에 대해서 하드웨어와 소프트웨어 간 인터페이스 명세와 부합, 소프트웨어 안전 요구사항의 충족 여부에 대해서 검증하여 결과를 보고서에 포함한다. 소프트웨어 코드의 검증은 코딩 지침과 부합하는지 여부를 검증하며 일반적으로 정적 분석 도구를 사용하여 검증한다.

마. 적용 기법

1) 설계 표기법

방법		ASIL			
		A	B	C	D
1a	**자연 언어**	++	++	+	+
1b	**비정형 표기법**	+	++	++	++
1c	**준정형 표기법**	+	++	++	++
1d	정형 표기법	+	+	+	+
++: 매우 권장, +: 권장, o: 권장사항 없음					

- 자연 언어

자연 언어는 프로그래밍 언어들, 즉 FORTRAN, COBOL, PASCAL, C, C++, Ada, LISP 및 PROLOG 등과 같은 인공 언어와는 다르게, 어법이 정해진 규칙만을 따르지 않고 일상적으로 사용되는 언어의 구조적인 체계를 말한다. 요약하면 자연어는 어떤 정돈된 완벽한 문법이나 형식적인 의미가 없는 언어를 말한다.

인간과 인간이 통신을 하고자 할 때에는 문어 (written language) 및 구어 (spoken language) 에 의한 수단으로 할 수 있다. 문어는 구어에 비해 문장의 애매모호함의 정도가 작은데, 그 이유는 어느 정도 정돈된 문법을 가지기 때문이다. 반면에 구어는 어떤 정돈된 완벽한 문법이나 형식적인 의미에 구애받지 않고 사용되므로 구어를 이해하기 위해서는 모든 잡음과 가청신호의 애매함을 처리할 수 있는 충분한 지식이 있어야 하므로 구어를 이해하는 것은 문어를 이해하는 것보다 훨씬 어렵다.

- 비정형 표기법

SW 아키텍처 설계의 비정형 표기법을 참고한다.

- 준정형 표기법

SW 아키텍처 설계의 준정형 표기법을 참고한다.

2) 설계 원리

방법		ASIL			
		A	B	C	D
1a	**서브프로그램과 함수에서 하나의 진입점과 하나의 종료점**	++	++	++	++
1b	**동적 개체 또는 변수를 사용하지 않음, 혹은 그렇지 않으면 동적 변수 생성시 온라인 시험**	+	++	++	++
1c	**변수 초기화**	++	++	+	+
1d	**변수 이름을 다목적으로 사용하지 않음**	+	++	++	++
1e	전역 변수를 사용하지 않음, 만약 사용해야 한다면 그 사용에 대해 정당화한다.	+	+	++	++
1f	포인터의 제한된 사용	o	+	+	++
1g	**암시적 형 변환 없음**	+	++	++	++
1h	**숨겨진 데이터 흐름이나 제어 흐름 없음**	+	++	++	++
1i	**무조건적 점프 없음**	++	++	++	++
1j	재귀 없음	+	+	++	++
++: 매우 권장, +: 권장, o: 권장사항 없음					

위의 방법들은 코딩 수준의 상세 규칙이다. 자동차 소프트웨어를 개발할 때는 MISRA C 표준에 의해서 코딩 작업이 이루어지기 때문에 MISRA C 규칙을 준수함으로써 위의 규칙도 지킬 수 있다. 아래 표는 위의 각 방법과 거기에 대응되는 MISRA C 규칙 사이의 연관 관계이다. 표현의 명확성을 위해 원문의 영어 표현을 그대로 사용하였다.

번호	ISO 26262 요구사항	MISRA C: 2004 규칙번호	MISRA C: 2012 규칙번호
1a	One entry and one exit point in sub programs and functions	14.7 (Required)	15.5 (Advisory)

1b	No dynamic objects or variables, or else online test during their creation	20.4 (Required)	18.7 (Required) 21.3 (Required)
1c	Initialization of variables	9.1 (Required)	9.1 (Mandatory)
1d	No multiple use of variable names	5.2 (Required) 8.7 – 8.10 (Required)	5.1 – 5.3 (Required) 5.8 (Required) 5.9 (Advisory)
1e	Avoid global variables or else justify their usage	5.2 (Required)	5.3 (Required)
1f	Limited use of pointers	11.1 – 11.4 (Required) 16.7 (Advisory) 17.1 – 17.6 (Required)	8.13 (Advisory) 11.1–11.9 (Required)
1g	No implicit type conversions	6.1 – 6.2 (Required) 10.1 – 10.4 (Required) 11.1 – 11.2 (Required)	11.1–11.9 (Required) Appendix C.1.1
1h	No hidden data flow or control flow	N/A	N/A
1i	No unconditional jumps	14.4 (Required) 20.7 (Required)	15.1 (Advisory) 15.2–15.4 (Required) 21.4 (Required)
1j	No recursions	16.2 (Required)	17.2 (Required) 21.9 (Required)

- 서브프로그램과 함수에서 하나의 진입점과 하나의 종료점

C 함수로 설명하면 함수에 진입할 수 있는 포인트와 함수가 종료되는 포인트는 각 1 개만 존재해야 한다는 의미이다. C 함수의 경우 함수에 진입하면 무조건 함수 안의 첫번째 행부터 시작하기 때문에 기본적으로 첫번째 요건은 만족한다. 다만 함수가 종료되는 포인트는 return 문으로 지정할 수 있기 때문에 하나의 함수 안에 여러 개의 return 문을 사용하면 여러 개의 종료점을 가질 수 있다. MISRA C (2004)의 규칙 14.7 과 MISRA C (2012)의 규칙 15.5 는 종료점이 하나여야 한다고 명시하고 있다. 아래 함수의 경우 3 개의 종료점이 있기 때문에 이 요구사항을 만족시키지 못한다.

```
bool_t check_limit(uint16_t n)
{
   if (n > LIMIT) {
        return false; // 종료점
   } else {
      return true; // 종료점
   }
   return true;  //종료점
}
```

- 동적 개체 또는 변수를 사용하지 않음, 혹은 그렇지 않으면 동적 변수 생성시 온라인 시험

Heap 메모리를 사용하는 calloc, malloc, realloc, free 함수는 메모리 누수, 사용 해제된 메모리에 대한 잘못된 접근 등의 문제를 일으킬 수 있기 때문에 사용하지 않는 것을 기본으로 한다. MISRA C (2004)의 규칙 20.4 와 MISRA C (2012)의 규칙 21.3 은 이를 명시하고 있다. MISRA C (2012)의 지침 4.12 는 C 표준 라이브러리뿐만 아니라 서드파티 패키지의 동적 메모리 할당도 명시적으로 금지한다. 이에 더해 최근 컴파일러들은 C99 의 Flexible Array Member 를 지원하는데 이를 이용하면 구조체에 크기가 동적으로 결정되는 Array Member 를 만들 수 있다. 이 기능은 동적 메모리 할당과 연결되어 사용되기 때문에 MISRA C (2012)의 규칙 21.3 에 의거하여 사용하면 안된다. 다만 꼭 필요한 경우에는 아래 코드와 같이 메모리 동적 할당이 성공했는지 반드시 온라인 시험을 수행해야 한다.

```
    char *ptr;

    ptr = (char *)malloc(BUFFER_SIZE);
    if (ptr == NULL) {  // 온라인 시험
        return FAIL;
    }
```

- 변수 초기화

변수값을 초기화하지 않고 사용할 경우 우연히 그 변수에 들어있던 원하지 않는 값을 사용하여 로직이 오동작을 할 수 있다. 특히 C 언어의 경우 전역변수는 자동으로 0 으로 초기화되지만 함수의 지역변수와 같은 자동변수의 경우 메모리 할당만 이루어지고 해당 메모리에 대한 초기화는 이루어지지 않는다. 그렇기 때문에 특히 자동변수의 경우 무조건 변수를 초기화한 후 사용해야 한다. MISRA C (2004)의 규칙 9.1 과 MISRA C (2012)의 규칙 9.1 은 자동변수의 경우 초기화를 강제하고 있다. MISRA C 는 자동변수만을 대상으로 하고 있지만 모든 변수에 대해서 초기화하는 것을 강제하는 것이 안전하다. 아래 코드의 경우 지역 변수 max_speed 를 실수로 초기화하지 않고 사용하는 경우이다. max_speed 변수에 어떤 값이 들어있을지 예측이 불가능하기 때문에 큰 문제를 일으킬 수 있다.

```
bool_t check_speed(uint16_t speed)
{
    uint16_t max_speed;
    if (speed > max_speed) {  // 초기화 없이 사용
        result = false;
    } else {
        result = true;
    }
    return result;
}
```

- 변수 이름을 다목적으로 사용하지 않음

C 언어의 경우 동일한 이름을 가진 전역변수와 지역변수가 동시에 존재할 수 있다. 이 경우 전역변수는 전체 코드에서 접근되지만 지역변수는 해당 변수가 선언된 함수 안에서만 접근할 수 있다. 같은 이름으로 변수를 접근해도 해당 함수 안에서는 지역변수의 메모리가 읽히고 다른 함수에서는 전역 변수의 메모리가 읽히게 된다. 이와 같은 변수명 작명법은 혼란을 야기할 수 있기 때문에 사용이 금지된다. MISRA C (2014) 규칙 5.2 와 MISRA C (2012) 규칙 5.8 - 5.9 는 이를 명시적으로 금지한다. 변수명을 길게 작명할 경우 컴파일러에 따라 인지할 수 있는 변수명의 길이에 한계가 있는 경우 코드상에서 사람의 눈으로는 다른 변수명으로 구분되지만 컴파일러는

같은 변수명으로 인식할 수 있다는 점도 고려해야 한다. 이 점은 MISRA C (2012) 규칙 5.1-5.2 를 참고한다. 아래 코드에는 count 라는 같은 이름을 가지는 변수가 두개 존재한다.

```
uint32_t count;
static void function(void)
{
    int16_t count = 0; // 전역 변수와 변수 이름이 중복되는 지역 변수
}
```

- 암시적 형 변환 없음

C 언어에서 다른 변수형을 가지는 변수들이나 상수들 사이에 연산이 일어나거나 할당이 되면 형 변환이 일어난다. C 언어에서 형변환이 일어날 때는 변수형 사이의 표현 범위와 정밀도의 차이 때문에 정보의 손실이 발생하는데 만약 암시적 형 변환이 있다면 이 정보 손실이 개발자가 의도한 것인지 실수에 의한 것인지 알 수 없다.

```
int function(void)
{
    int a=3;
    float b= 4.5;
    a = a + b; // float 에서 int 로의 암시적 형 변환이 일어나는 부분
    a = a + (int)b // 명시적 형 변환
}
```

- 숨겨진 데이터 흐름이나 제어 흐름 없음

코드를 읽을 때 가장 중요한 것은 데이터와 제어의 흐름이다. 각 행이 실행되면서 변수 값이 어떻게 바뀌는지 그리고 다음에 어떤 행이 실행되는지가 모두 코드 안에 표현되어야 한다. 만약 그렇지 않으면 본인이 작성한 코드인 경우에는 코드를 잘못 이해하여 문제를 일으킬 수 있다. 예를 들어 C 언어의 매크로를 잘못 사용하면 이와 같은 문제를 야기할 수 있다. 아래 코드는 반복적으로 사용하는 함수의 정상종료 부분을 짧은 매크로로 줄여서 사용하는 코드를 보여준다. 이와 같이 하면 약간의 타이핑 시간은 줄일 수 있으나 데이터의 흐름과 제어 흐름을 모두 매크로

안에 감추게 된다. 이 코드에서는 ret 라는 변수값이 바뀌는 데이터 흐름과 함수가 종료되는 제어 흐름이 OK 매크로 안에 감춰진다.

```
#define OK (ret=1; return ret;)


int function(void)
{
    int ret = 0;
   ...
   OK;  //데이터의 흐름이 보이지 않는 부분
}
```

- 무조건적 점프 없음

C 언어의 무조건적 점프를 수행하는 goto 는 사용이 금지된다. 프로그램의 제어 흐름을 복잡하게 하고 분석하기 어렵게 만들기 때문이다. MISRA C (2004) 규칙 14.4 와 MISRA C (2012) 규칙 15.1 은 goto 의 사용을 금지한다. goto 의 사용은 무조건 금지해야 맞으나 MISRA C (2012)는 만약 어쩔 수 없이 goto 가 사용된다면 주의해야 할 점을 MISRA C (2012) 규칙 15.2, 15.3, 15.4 에 명시하고 있다. 이에 더해 서로 다른 함수간의 무조건적 점프를 지원하는 setjmp, longjmp 의 사용도 금지된다. 이는 MISRA C (2004) 20.7 과 MISRA C (2012) 21.4 에 명시되어 있다.

3) 검증 방법

방법		ASIL			
		A	B	C	D
1a	**워크쓰루**	++	+	o	o
1b	**인스펙션**	+	++	++	++

1c	준정형 검증	+	+	++	++
1d	정형 검증	o	o	+	+
1e	제어 흐름 분석	+	+	++	++
1f	데이터 흐름 분석	+	+	++	++
1g	**정적 코드 분석**	+	++	++	++
1h	의미적 코드 분석(semantic code analysis)	+	+	+	+
++: 매우 권장, +: 권장, o: 권장사항 없음					

- 워크쓰루

워크쓰루는 명세와 구현 사이의 불일치를 발견하는 활동으로 수행 방법은 SW 안전 공통 가이드 5.3.2 SW 안전 기록 작성의 IEC61508-3 B.8.6b 를 참고한다.

- 인스펙션

인스펙션은 소프트웨어 요소의 결함을 발견하는 활동이며 수행 방법은 SW 안전 공통 가이드 5.3.2 SW 안전 기록 작성의 IEC61508-3 B.8.6a 를 참고한다. 인스펙션은 일반적으로 계획, 사전교육, 준비, 인스펙션 회의, 수정, 후속 조치로 나눠진다.

- 정적 코드 분석

정적 코드 분석은 소프트웨어를 코드 수준에서 분석하여 결함을 찾아내는 것이다. 동적 분석이 컴파일된 프로그램을 실행하여 분석하는 반면에 정적 코드 분석은 프로그램을 실행하지 않고 순수하게 소스코드에 대한 분석만을 수행한다. 수행 방법은 SW 안전 공통 가이드 IEC61508-3 A.9.3 을 참고한다.

일반적으로 정적 코드 분석을 한다고 할 때는 사람에 의한 검토가 아니라 자동화 도구를 이용한 분석을 의미한다. 이와 같은 도구를 정적 코드 분석 도구(Static Code Analysis Tool)라고 한다. 자동차 제어 분야에서는 MISRA C 가 표준 코딩 가이드로 사용되기 때문에 MISRA C 규칙에 대한 준수 여부를 정적 코드 분석 도구를 이용해 검증하는 것이 일반적이다. MISRA C 정적 코드 분석 도구는 현재까지는 모두 고가의 상용 제품으로만 판매되고 일반적으로 커스터마이징을 위한 개발 비용과 도구 적용을 위한 비용 및 컨설팅과 교육을 포함하여 제공된다. 중소 소프트웨어 개발

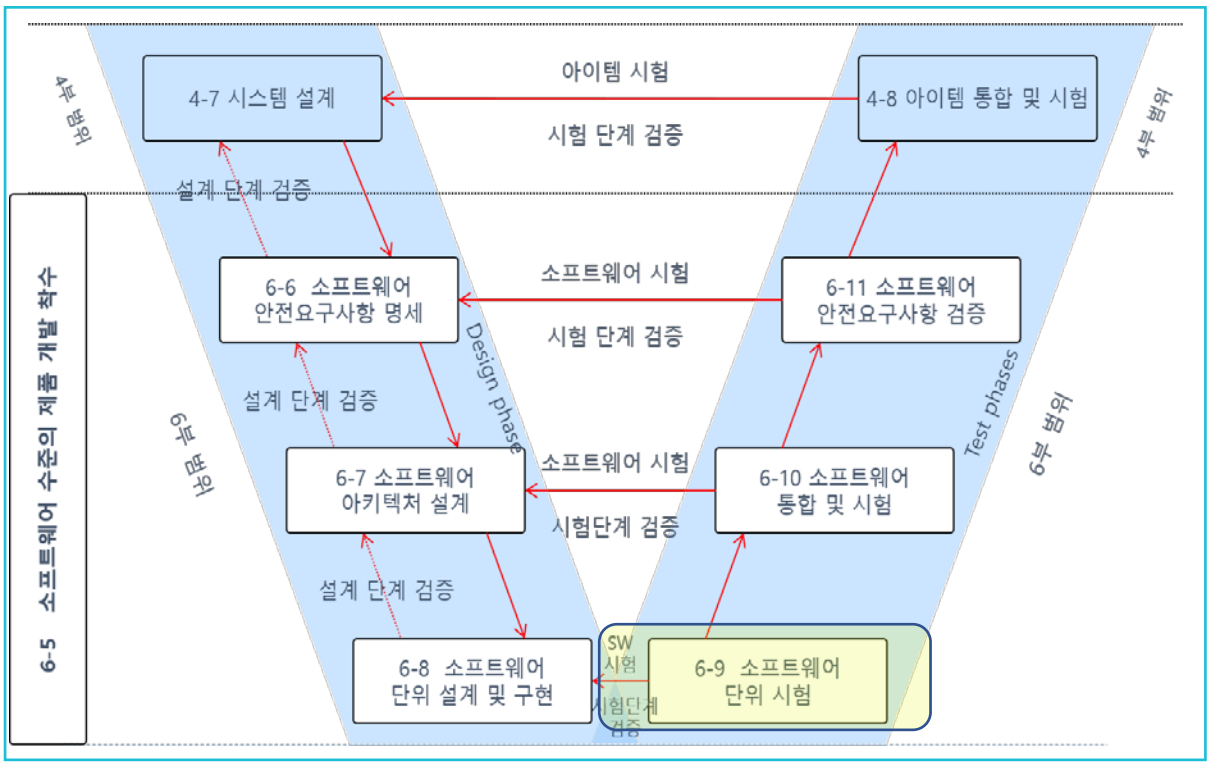
기업의 경우 오픈 소스 정적 분석 도구를 사용하길 원하는 경우가 많은데 MISRA C 의 경우 각 규칙 문구 자체가 저작권에 걸려있기 때문에 도구에서 MISRA C 규칙 문구를 쓰기 위해서는 도구 개발사자 MISRA 측에 비용을 지불해야 한다.[19] 그렇기 때문에 오픈 소스 도구 개발이 되어있지 않으며, 따라서 중소 소프트웨어 기업의 경우 발주처에서 MISRA C 에 대한 정적 분석 결과를 요구하면 이를 위한 도구 제공을 계약 단계에서 협의해야 한다. 또한 MISRA C 의 규칙들 중에서 프로젝트 사정상 준수가 어려운 규칙들은 deviation permit[20]을 미리 합의하는 것이 중요하다.

[19] MISRA C FAQ

[20] MISRA Compliance:2016 Achieving compliance with MISRA Coding Guidelines

3.5. 소프트웨어 단위 시험

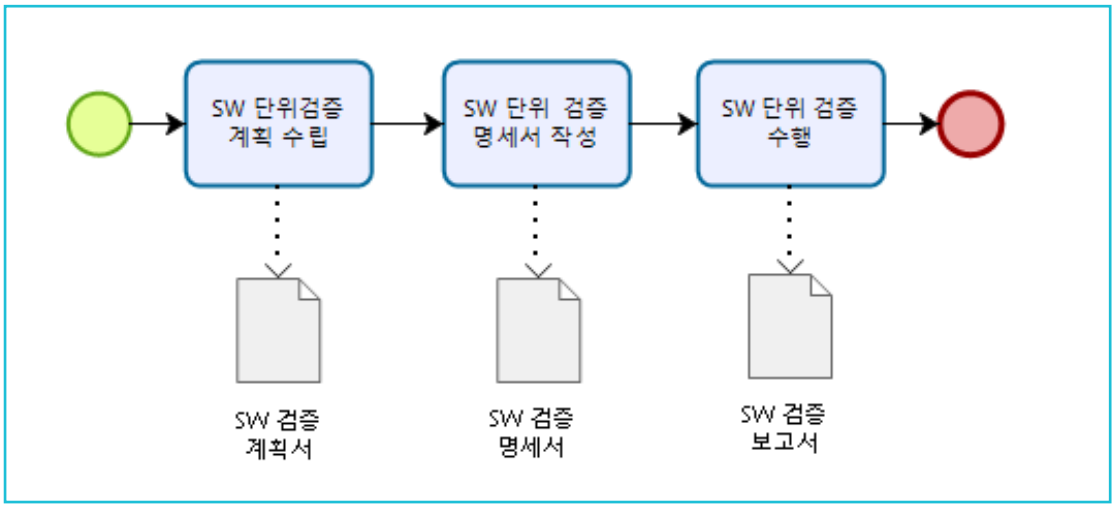
그림 31. 소프트웨어 개발 수명 주기 – 소프트웨어 단위 시험



가. 수행 활동

소프트웨어 단위 시험에서는 소프트웨어 아키텍처 설계에서 식별된 각 단위에 대해서 시험을 계획하고 수행하여 설계 명세서를 만족하는지 확인하는 활동이다. 단위 시험을 통해서 단위 설계 명세서대로 해당 단위가 구현되었는지 여부, 하드웨어와 소프트웨어간 인터페이스에는 오류가 없는지 여부와 오류 처리 메커니즘의 효과성 등에 대해서 검증할 수 있다. 단위 시험 방법은 ASIL 레벨별로 요구사항 기반 시험, 인터페이스 시험, 결함 주입 시험, 자원 사용 시험과 모델과 코드 간의 비교 시험을 수행할 수 있다. ASIL B 레벨에서는 요구사항 기반 시험과 인터페이스 시험을 해야 한다.

그림 32. 활동 흐름 – 소프트웨어 단위 시험



단위 테스트 계획 수립 시에 다음 사항을 포함하여 계획을 수립 한다.(ISO 26262-8, 9.4.1)

- 검증할 작업 산출물의 내용
- 검증에 사용되는 방법
- 검증에 대한 합격 및 불합격 기준
- 해당되는 경우, 검증 환경
- 해당되는 경우, 검증에 사용되는 도구
- 이상점이 탐지될 때, 취해야 하는 조치
- 회귀전략

단위 테스트 케이스의 명세는 다음 사항을 포함하여야 한다. (ISO 26262-8, 9.4.2.2)

- 고유 식별자
- 검증되어야 하는 관련된 작업 산출물의 버전에 대한 참고 사항
- 사전 조건 및 설정
- 해당되는 경우, 환경 조건
- 입력 데이터, 입력 데이터의 시간적인 순서와 값

- 출력 데이터, 출력 값의 허용 범위, 시간 동작, 허용 오차 동작 등을 포함하는 예상되는 행위

테스트 케이스에 시험 방법에 대해서는 다음 사항을 명시해야 한다. (ISO 26262-9.4.2.3)

- 시험 환경
- 논리 및 시간적인 의존성
- 자원

소프트웨어 단위 시험은 아래와 같은 환경에서 수행될 수 있다.

- 루프 내의 모델(model-in-the-loop)시험
- 루프 내의 소프트웨어(software-in-the-loop)시험
- 루프 내의 프로세서(processor-in-the-loop)시험
- 루프 내의 하드웨어(hardware-in-the-loop)시험

검증 결과의 평가는 다음과 같은 정보를 포함해야 한다. (ISO 26262-8, 9.4.3.2)

- 검증된 작업 산출물의 고유식별
- 검증 계획 및 검증 명세서에 관한 참조자료
- 검증 환경과 사용된 검증 도구의 설정 및 평가하는 동안 사용되는 보정 데이터
- 예상된 결과에 대한 검증 결과의 부합화 수준
- 검증에 대한 합격과 불합격에 대한 명확한 판단 기준(검증 결과가 불합격이면 불합격에 대한 근거 및 검증된 작업 산출물의 변경을 위한 제안)
- 어떤 검증 단계가 실행되지 않은 이유

나. 수행 활동 구성

구 분	설 명
목 적	● 소프트웨어 단위 설계 명세서에 맞게 단위 구현이 되었는지 검증한다.
시작 기준	● 프로젝트 계획이 작성되고 승인되었다. ● 안전 계획이 작성되고 승인되었다. ● 기술안전 개념이 도출되고 시스템 설계 명세서에 반영되었다. ● 시스템 설계 명세서가 작성되고 승인되었다. ● 하드웨어와 소프트웨어 인터페이스 명세서가 작성되고 승인되었다. ● 모델링과 프로그래밍 언어를 위한 설계 및 코딩 지침이 작성되고 승인되었다. ● 소프트웨어 안전 요구사항 명세서가 작성되고 승인되었다. ● 소프트웨어 검증 계획이 수립되고 승인되었다.
입력물	● 하드웨어와 소프트웨어간 인터페이스 명세서(갱신) ● 소프트웨어 검증 계획(갱신) ● 안전 계획(갱신) ● 소프트웨어 단위 설계 명세서 ● 소프트웨어 단위 구현 ● 소프트웨어 검증 보고서(갱신) ● 도구 적용 지침
수행 활동	● 소프트웨어 단위 검증 계획 ● 소프트웨어 단위 검증 명세 ● 소프트웨어 단위 검증 수행
출력물	● 소프트웨어 검증 계획(갱신) ● 소프트웨어 검증 명세서 ● 소프트웨어 검증 보고서(갱신)
완료 기준	● 소프트웨어 단위 검증 계획이 작성되고 승인됨 ● 소프트웨어 단위 검증 명세서가 작성되고 승인됨 ● 소프트웨어 단위 검증 보고서가 작성되고 승인됨

다. 역할 및 책임

표 14. 역할 및 책임 – 소프트웨어 단위 시험

활동 \ 역할	안전 관리자	개발자	검증 담당자	품질 담당자	프로젝트 관리자
소프트웨어 단위 설계	I	R	I	C	A
소프트웨어 단위 구현	I	R	I	C	A
소프트웨어 단위 검증	I	I	R	C	A

* R: Responsible, 담당자, 해당 업무를 실행하는 주체 및 책임자
* A: Accountable, 의사 결정권자, 해당 업무 최종 승인자
* C: Consulted, 조언자, 업무 협의 또는 의견 제공자
* I: Informed, 공유 대상자, 해당 업무 참조자

라. 산출물

- 소프트웨어 검증 계획서: 소프트웨어 단위 시험 계획을 말하여 단위를 시험하여 검증할 계획 수립한다. 검증 대상, 검증 방법, 검증 담당자, 검증 환경, 검증 통과 기준을 포함한다.
- 소프트웨어 검증 명세서: 소프트웨어 단위를 시험하기 위한 단위 테스트 케이스를 설계하여 테스트 케이스 내역을 포함한다. 테스트 케이스는 테스트 사전 조건, 수행 절차, 테스트 통과 기준을 포함한다.
- 소프트웨어 검증 보고서 : 소프트웨어 단위 시험에 대한 수행 결과를 포함한다. 테스트 수행 일정, 테스트 케이스별 수행 성공 및 실패 결과, 전체 테스트 케이스의 성공 비율 등을 포함하여 검증 결과에 대해서 판단할 수 있는 정보를 제공한다.

마. 적용 기법

1) 소프트웨어 단위 시험 방법

방법		ASIL			
		A	B	C	D
1a	**요구사항 기반 시험**	++	++	++	+
1b	**인터페이스 시험**	++	++	++	++
1c	결함 주입 시험	+	+	+	++
1d	자원 사용 시험	+	+	+	++
1e	모델과 코드 간 비교 시험(back-to-back test), 해당되는 경우	+	+	++	++
++: 매우 권장, +: 권장, o: 권장사항 없음					

- 요구사항 기반 시험

요구사항기반 시험은 테스트 대상이 요구사항을 만족하는지 판단하는 것으로 시험 대상 단위가 요구사항을 처리 및 커버하는지 확인하는 것이다. 시험에서 발견한 결함 정보 등을 제공해 개발자 및 품질 담당자를 비롯한 관련자에게 필요한 정보를 제공한다. 요구사항 기반 시험은 모든 ASIL 등급에서 필수적으로 적용되는 방법이다. 요구사항 기반 시험은 요구사항을 분석, 테스트 케이스 설계, 테스트 수행, 테스트 결함 조치 활동 순서로 수행한다. 요구사항 분석에서 요구사항 기반으로 테스트가 필요한 시험 항목을 도출하고, 모든 요구사항이 누락되지 않도록 확인해야 한다. 테스트 케이스 설계는 동등 분할, 경계 값 분석 등의 테스트 설계 기법을 활용하여 해당 요구사항이 컴포넌트 또는 단위에서 적합하게 구현되었는지 검증한다. 테스트 수행에서 발견된 결함은 요구사항과의 관련성에 따라서 결함 중요도, 결함 분류를 하여 관련자와 공유하고 수정하도록 한다. 테스트 수행 내역이 모든 요구사항에 대해서 수행되었는지 확인하는 것이 필요한데 테스트 케이스 설계 시에 관련 요구사항과 추적할 수 있도록 요구사항 식별자를 포함하도록 한다.

- 인터페이스 시험

인터페이스 시험은 모든 ASIL 등급에서 안전 관련 기능이 구현된 단위에서 필수적으로 수행하는 항목이다. 인터페이스는 소프트웨어 관련 단위 사이의 인터페이스, 해당 소프트웨어 단위와 하드웨어의 인터페이스 검증을 포함한다. 인터페이스에서 입력 유효 값 처리 여부, 모든 비유효

입력값 거부 여부들을 포함해 인터페이스의 오류가 없는지 확인한다. SW 안전 공통 가이드 5.3.2 SW 안전 기록 작성의 IEC61508-3 A.5.7 : 인터페이스 테스트(Interface testing)를 참조한다.

2) 소프트웨어 단위 시험 케이스 생성 방법

방법		ASIL			
		A	B	C	D
1a	**요구사항 분석**	++	++	++	+
1b	**동치 클래스(equivalence class)의 생성 및 분석**	+	++	++	++
1c	**경계 값(boundary value)의 분석**	+	++	++	++
1d	오류 추측	+	+	+	+
++: 매우 권장, +: 권장, o: 권장사항 없음					

- 요구사항 분석

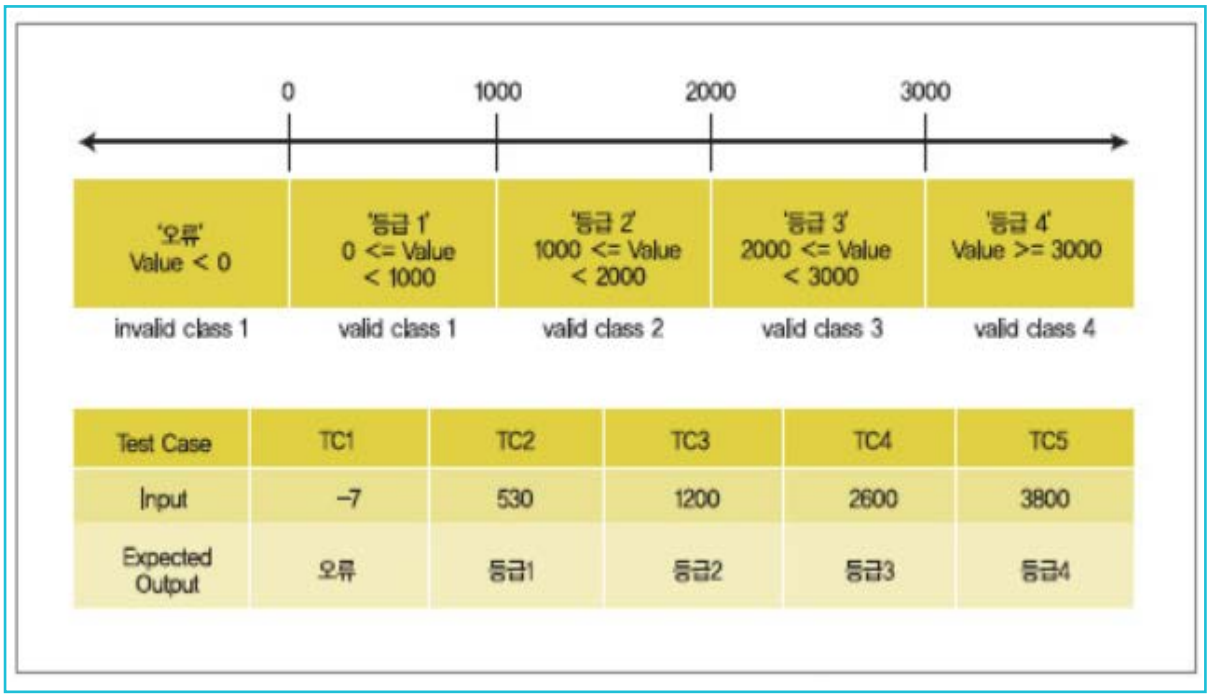
요구사항 분석은 테스트 케이스 설계 시에 문서화된 요구사항 명세서를 분석하여 테스트 항목을 구성하고 테스트 선행 조건, 테스트 절차, 테스트 완료 조건을 정의하는 것이다. 테스트 케이스 설계를 위해 요구사항 분석이 적절하게 수행되기 위해서는 요구사항의 오류가 없도록 요구사항 명세 단계에서 적합한 검증이 선행되어야 한다. 요구사항이 모호한지, 요구사항들이 서로 일관성은 없는지, 요구사항은 검증 가능한지 여부에 따라서 테스트 설계의 정확성이 결정된다. 요구사항 분석 기반으로 테스트 케이스가 적합하게 설계되었는지 확인하기 위해 요구사항 대비 테스트케이스 구현과 실행 비율로 판단하는 요구사항 커버리지와 요구사항과 테스트케이스의 추적 관계를 확인할 수 있다. 요구사항 명세서에 명시된 요구사항을 기능적 요구사항, 비기능적 요구사항인지 분석하여 해당 요구사항에 적합한 테스트 절차를 정의한다. 요구사항 정의서에 명시된 기능, 비기능에 대해 테스트 케이스에 작성하여 구현된 소프트웨어가 요구사항과 일치하는지 확인한다.

- 동치 클래스의 생성 및 분석

동치 클래스의 생성 및 분석은 소프트웨어 또는 시스템의 입력과 출력 값의 영역을 독립적인 분할 영역(클래스)로 구분하여 각 영역에서 대표 값을 선택하여 테스트 케이스를 설계한다. 같은 영역에 있는 값은 내부에서 같은 방식으로 처리하기 때문에 어떤 값을 선택해도 결과값이 같다는

원리를 이용한다. 동치 클래스는 입력값과 출력 값, 내부 값, 시간 관련 값, 인터페이스 매개변수에 적용할 수 있다. 입력 값 분할 영역은 유효 영역, 입력되면 안되는 비유효 영역으로 나누고 결과 값 분할 영역도 값 분할 영역에 따른 유효 영역, 비유효 영역으로 나눈다. 동치 분할 영역을 구성하기 위해서는 요구사항 명세서를 분석하여 입력과 출력 영역을 유사한 클래스로 분할하고, 각 분할된 클래스에서 대표하는 테스트 케이스를 도출한다. 예를 들어 아래 그림과 같이 0 부터 3000 이상의 입력 값에 대해서 유효하지 않은 클래스 1 개, 유효한 클래스 4 개를 생성하여 각 클래스를 대표하는 값을 하나씩 선정하는 경우 전체 5 개의 테스트 케이스를 설계할 수 있다.

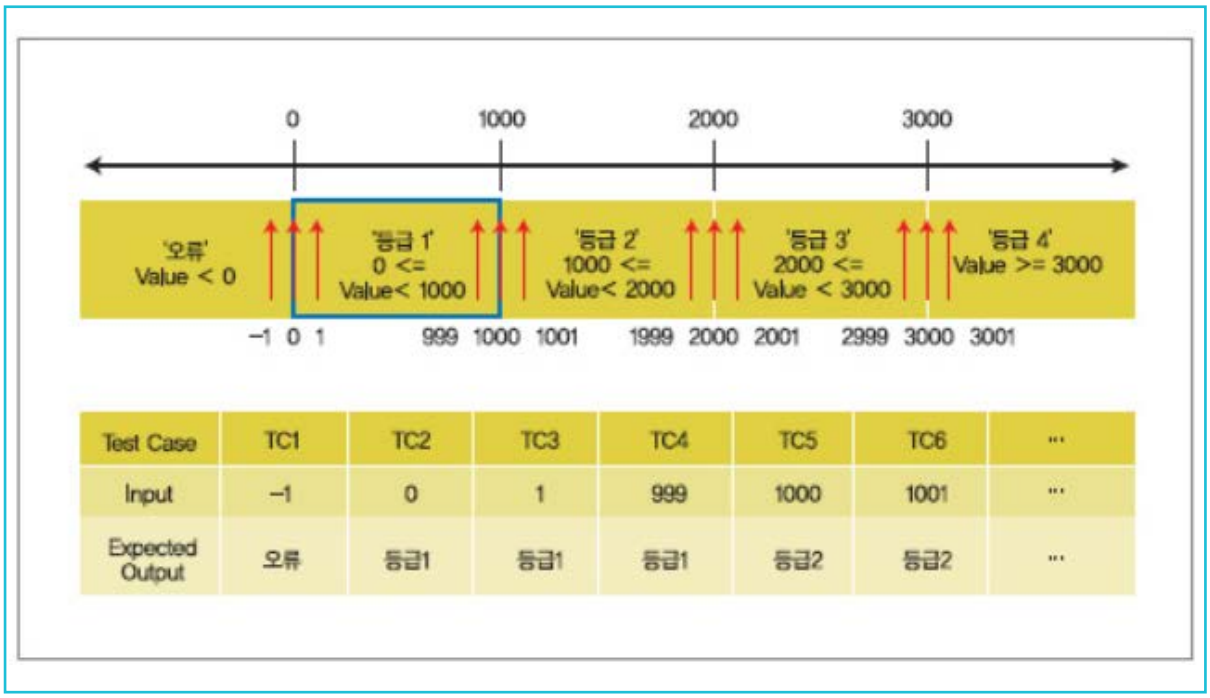
그림 33. 동치 클래스 생성 및 분석 예시



- 경계 값의 분석

경험적으로 입력 값의 경계 값 근처에서 결함이 발생하는 확률이 높기 때문에 이를 방지하기 위한 테스트 케이스 설계 방법으로 입력 영역의 경계 값에 초점을 맞추어 테스트케이스를 도출한다. 경계 값을 도출하기 전에 동치 분할을 먼저 수행하고, 각 동치 분할의 경계값을 분석한다. 입력 조건이 값의 범위라면 범위의 끝부분의 유효한 값과 그 보다 위와 아래의 유효하지 않은 값으로 테스트케이스를 생성하고, 입력 조건이 테이블이나 일렬리스트와 같은 집합이라면 집합 중 첫째 원소와 마지막 원소에 근거해서 테스트케이스를 작성한다. 아래 그림과 같이 경우에 동치 클래스의 경계 값을 근거로 해 테스트 케이스를 도출할 수 있다.

그림 34. 경계 값 분석 예시



Test Case	TC1	TC2	TC3	TC4	TC5	TC6	···
Input	-1	0	1	999	1000	1001	···
Expected Output	오류	등급1	등급1	등급1	등급2	등급2	···

3) 단위 수준의 구조적 커버리지 지표

방법		ASIL			
		A	B	C	D
1a	**구문 커버리지**	++	++	+	+
1b	**분기 커버리지**	+	++	++	++
1c	MC/DC(수정된 조건/결정 커버리지)	+	+	+	++
++: 매우 권장, +: 권장, o: 권장사항 없음					

- 구문 커버리지

구조적 커버리지는 단위 테스트 케이스가 단위 수준의 요구사항 커버리지를 충족하는지 측정하여 달성된 커버리지가 불충분하면 테스트 케이스를 추가한다. 구조적 커버리지를 통해서 요구사항의 적합성, 사용되지 않는 데드 코드(Dead code), 비활성화 또는 의도하지 않는 기능을 찾아서 제거할 수 있다.

구문 커버리지는 소스코드 내의 모든 문장을 최소 한번 이상 실행하도록 테스트케이스를 설계하는 방법이다. 소스 코드 라인 수 기준으로 수행된 라인 수 비율로 측정한다. 테스팅이 얼마나 충분함이 얼마인지를 측정하는 것이다. 충분함을 백분율(%)로 표현한다. 구문 커버리지 목표가 100%인데 현재 구문 커버리지가 80%라면 누락된 항목(20%)을 테스트하기 위해 해당 테스트 케이스를 추가한다. 예를 들면 아래 함수는 5 개의 코드로 되어 있으며, 테스트 수행 시에 5 개의 코드가 모두 실행되면 구문 커버리지는 100%가 된다.

```
int add_func()
{
  int i = 0;
  int n = 0;

  i = i+1;
  j = j+1;
```

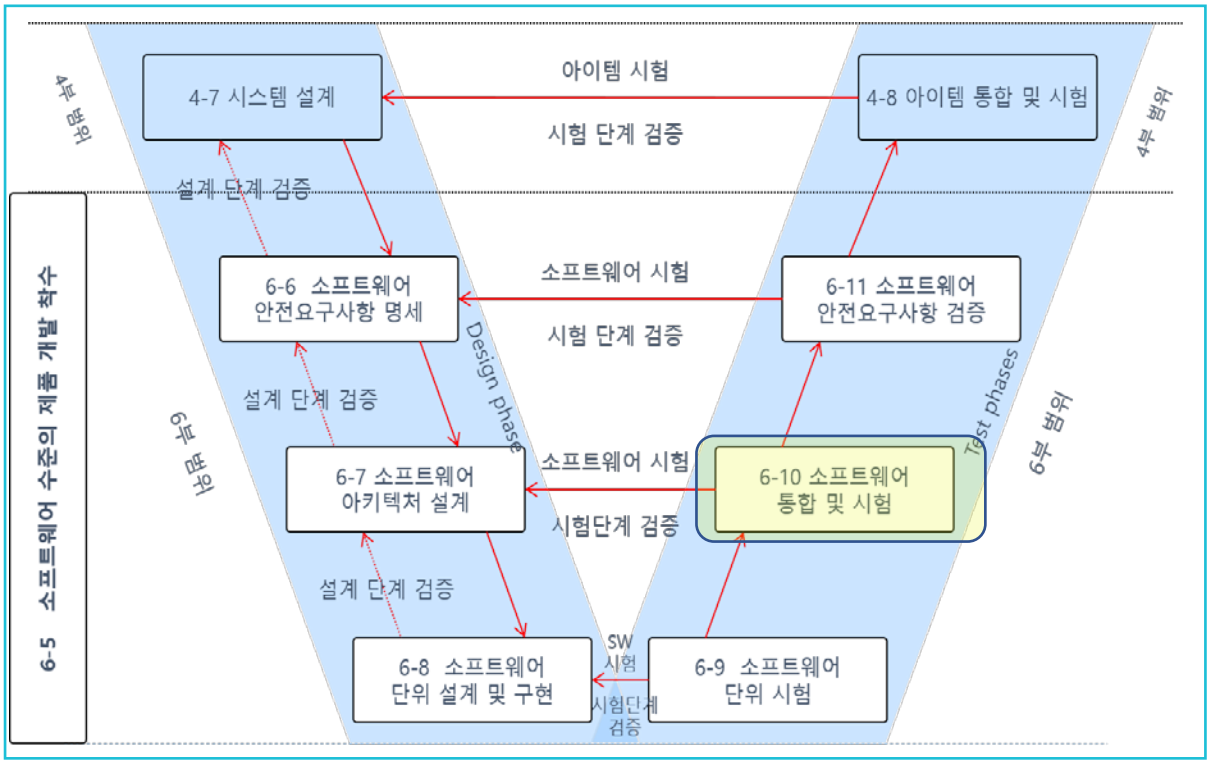
- 분기 커버리지

분기 커버리지는 코드에 있는 소스 코드 내의 분기점에서 결과가 참(true), 거짓(false)를 적어도 한번 이상 수행하도록 테스트 케이스를 설계하는 것이다. 모든 선택 분기점을 한 번 이상씩 테스트하여 분기에서 제어 흐름을 놓치지 않도록 한다. 소스 코드 내에 있는 전체 분기와 테스트 수행된 분기의 수에 대한 비율을 백분율(%)로 측정한다. 아래와 같은 코드의 경우 분기 커버리지를 만족하기 위해서는 `if (a > 0 || b > 0)` 분기가 `true, false` 한 번 이상 결정되어 수행되어야 한다. 이를 위해서는 최소한 `a=1, b=1` 과 `a=0, b=0` 값을 가지는 두 개 의 테스트케이스를 설계해야 한다.

```
int add_func(int a, int b)
{
    if (a > 0 || b > 0) {
        pritf("%d",a+b);
    } else {
        printf("low value");
    }
}
```

3.6. 소프트웨어 통합 및 시험

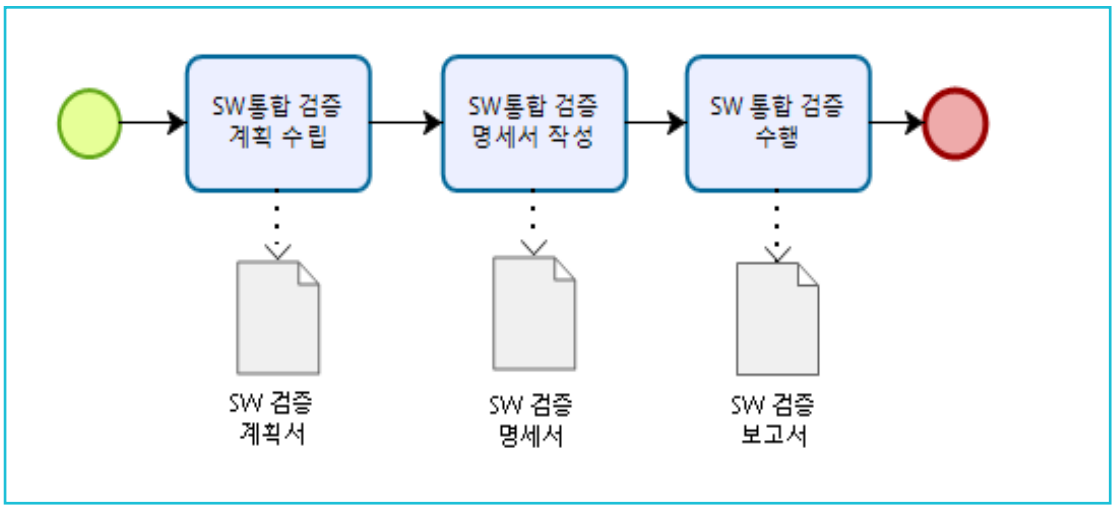
그림 35. 소프트웨어 개발 수명 주기 – 소프트웨어 통합 및 시험



가. 수행 활동

소프트웨어 통합 및 시험 단계에서는 소프트웨어 단위 시험을 통과한 컴포넌트를 통합하고, 소프트웨어 아키텍처 설계와 부합되는지 시험하여 검증한다. 소프트웨어 통합 시에는 통합되는 컴포넌트의 기능적 종속성과 하드웨어와 소프트웨어 간의 종속성을 고려해야 한다. 통합 시험의 대상은 소프트웨어 컴포넌트이며 소프트웨어 아키텍처 설계에 대한 부합과 함께 하드웨어와 소프트웨어의 인터페이스에 따라서 정상적으로 통합되어 수행되는지 확인한다.

그림 36. 활동 흐름 – 소프트웨어 통합 및 시험



통합 시험 계획은 다음 사항을 포함하여야 한다. (ISO-26262-8, 9.4.1)

- 검증할 작업 산출물의 내용
- 검증에 사용되는 방법
- 검증에 대한 합격 및 불합격 기준
- 해당되는 경우, 검증 환경
- 해당되는 경우, 검증에 사용되는 도구
- 이상점이 탐지될 때, 취해야 하는 조치
- 회귀전략

소프트웨어 통합 시험에는 요구사항 기반 시험, 인터페이스 시험, 결합 주입 시험 등을 할 수 있으며, ASIL 레벨 B 에서는 요구사항 기반 시험과 인터페이스 시험이 필수이다. 통합 시험을 수행하기 위해서는 통합 테스트 케이스를 설계해야 하며 요구사항 분석과 동치 클래스 분석,

경계 값 분석을 사용할 수 있다. 테스트 케이스의 명세는 다음 사항을 포함하여야 한다. (ISO 26262-8, 9.4.2.2)

- 고유식별
- 검증되어야 하는 관련된 작업 산출물의 버전에 대한 참고 사항
- 사전 조건 및 설정
- 해당되는 경우, 환경 조건
- 입력 데이터, 입력 데이터의 시간적인 순서와 값
- 출력 데이터, 출력 값의 허용 범위, 시간 동작, 허용 오차 동작 등을 포함하는 예상되는 행위

테스트 케이스에 시험 방법에 대해 다음 사항을 명시해야 한다. (ISO 26262-8, 9.4.2.3)

- 시험 환경
- 논리 및 시간적인 의존성
- 자원

그리고 테스트 케이스가 적절하게 설계되어 요구사항을 충분히 커버하는지 파악하기 위해서 함수 커버리지와 호출 커버리지를 측정할 수 있다. 함수 커버리지는 소프트웨어 함수가 실행된 비율이며, 호출 커버리지는 소프트웨어 함수에 대한 호출의 실행 비율이다. 각 지표는 ASIL C 이상에서는 필수적으로 측정하여 통합 시험이 충분히 수행되었는지 평가해야 한다.

소프트웨어 통합 시험은 아래와 같은 환경에서 수행될 수 있다.

- 루프 내의 모델(model-in-the-loop)시험
- 루프 내의 소프트웨어(software-in-the-loop)시험
- 루프 내의 프로세서(processor-in-the-loop)시험
- 루프 내의 하드웨어(hardware-in-the-loop)시험

나. 수행 활동 구성

구 분	설 명
목 적	● 소프트웨어 엘리먼트 사이에 특별한 통합 수준 및 인터페이스를 시험한다.
시작 기준	● 프로젝트 계획이 작성되고 승인되었다. ● 안전 계획이 작성되고 승인되었다. ● 기술안전 개념이 도출되고 시스템 설계 명세서에 반영되었다. ● 시스템 설계 명세서가 작성되고 승인되었다. ● 하드웨어와 소프트웨어 인터페이스 명세서가 작성되고 승인되었다. ● 모델링과 프로그래밍 언어를 위한 설계 및 코딩 지침이 작성되고 승인되었다. ● 소프트웨어 안전 요구사항 명세서가 작성되고 승인되었다. ● 소프트웨어 검증 계획이 수립되고 승인되었다.
입력물	● 하드웨어와 소프트웨어간 인터페이스 명세서 (갱신) ● 소프트웨어 아키텍처 설계 명세서 ● 안전 계획(갱신) ● 구현된 소프트웨어 단위 ● 소프트웨어 검증 계획(갱신) ● 소프트웨어 검증 명세서 ● 소프트웨어 검증 보고서(갱신)
수행 활동	● 소프트웨어 통합 검증 계획 수립 ● 소프트웨어 통합 검증 ● 소프트웨어 통합 검증 보고
출력물	● 소프트웨어 검증 계획(갱신) ● 소프트웨어 검증 명세서(갱신) ● 소프트웨어 검증 보고서(갱신)
완료 기준	● 소프트웨어 통합 검증 계획서가 작성되고 승인됨 ● 소프트웨어 검증 명세서가 작성되고 승인됨 ● 소프트웨어 검증 보고서가 작성되고 승인됨

다. 역할 및 책임

표 15. 역할 및 책임 – 소프트웨어 통합 및 시험

역할 / 활동	안전 관리자	개발자	검증 담당자	품질 담당자	프로젝트 관리자
소프트웨어 통합 검증 계획	I	I	R	C	A
소프트웨어 통합 검증 명세	I	I	R	C	A
소프트웨어 통합 검증 보고	I	I	R	C	A

* R: Responsible, 담당자, 해당 업무를 실행하는 주체 및 책임자
* A: Accountable, 의사 결정권자, 해당 업무 최종 승인자
* C: Consulted, 조언자, 업무 협의 또는 의견 제공자
* I: Informed, 공유 대상자, 해당 업무 참조자

라. 산출물

- 소프트웨어 검증 계획서: 소프트웨어 통합 시험 계획을 말하며 통합 시험하여 검증할 계획 수립한다. 검증 대상, 검증 방법, 검증 담당자, 검증 환경, 검증 통과 기준을 포함한다.
- 소프트웨어 검증 명세서: 소프트웨어 통합 시험을 수행하기 위한 테스트 케이스를 설계하여 테스트 케이스 내역을 포함한다. 테스트 케이스는 테스트 사전 조건, 수행 절차, 테스트 통과 기준을 포함한다.
- 소프트웨어 검증 보고서 : 소프트웨어 통합 시험에 대한 수행 결과를 포함한다. 테스트 수행 일정, 테스트 케이스별 수행 성공 및 실패 결과, 전체 테스트 케이스의 성공 비율 등을 포함하여 통합 시험 결과에 대해서 판단할 수 있는 정보를 포함한다.

마. 적용 기법

1) 소프트웨어 통합 시험 방법

방법		ASIL			
		A	B	C	D
1a	**요구사항 기반 시험**	++	++	++	++
1b	**인터페이스 시험**	++	++	++	++
1c	결함 주입 시험	+	+	+	++
1d	자원 사용 시험	+	+	+	++
1e	해당되는 경우 모델과 코드 간 비교 시험(back-to-back test)	+	+	++	++
++: 매우 권장, +: 권장, o: 권장사항 없음					

- 요구사항 기반 시험

아키텍처 수준의 소프트웨어 요구사항을 기반으로 시험하는 방법을 말한다. 단위 시험의 요구사항 기반 시험과 동일한 방법으로 단위 시험의 요구사항 기반 시험을 참조한다.

- 인터페이스 시험

단위 시험의 인터페이스 시험과 동일한 방법으로 단위 시험의 인터페이스 시험을 참조한다.

2) 통합 시험 테스트 케이스 생성 방법

방법		ASIL			
		A	B	C	D
1a	**요구사항 분석**	++	++	++	++
1b	**동치 클래스(equivalence class)의 생성 및 분석**	+	++	++	++
1c	**경계 값(boundary value)의 분석**	+	++	++	++
1d	오류 추측	+	+	+	+
++: 매우 권장, +: 권장, o: 권장사항 없음					

- 요구사항 분석

단위 시험의 요구사항 분석과 동일한 방법으로 단위 시험의 요구사항 분석을 참조한다.

- 동치 클래스 생성 및 분석

단위 시험의 동치 클래스 생성 및 분석과 동일한 방법으로 단위 시험의 인터페이스 시험을 참조한다.

- 경계 값 분석

단위 시험의 경계 값 분석과 동일한 방법으로 단위 시험의 인터페이스 시험을 참조한다

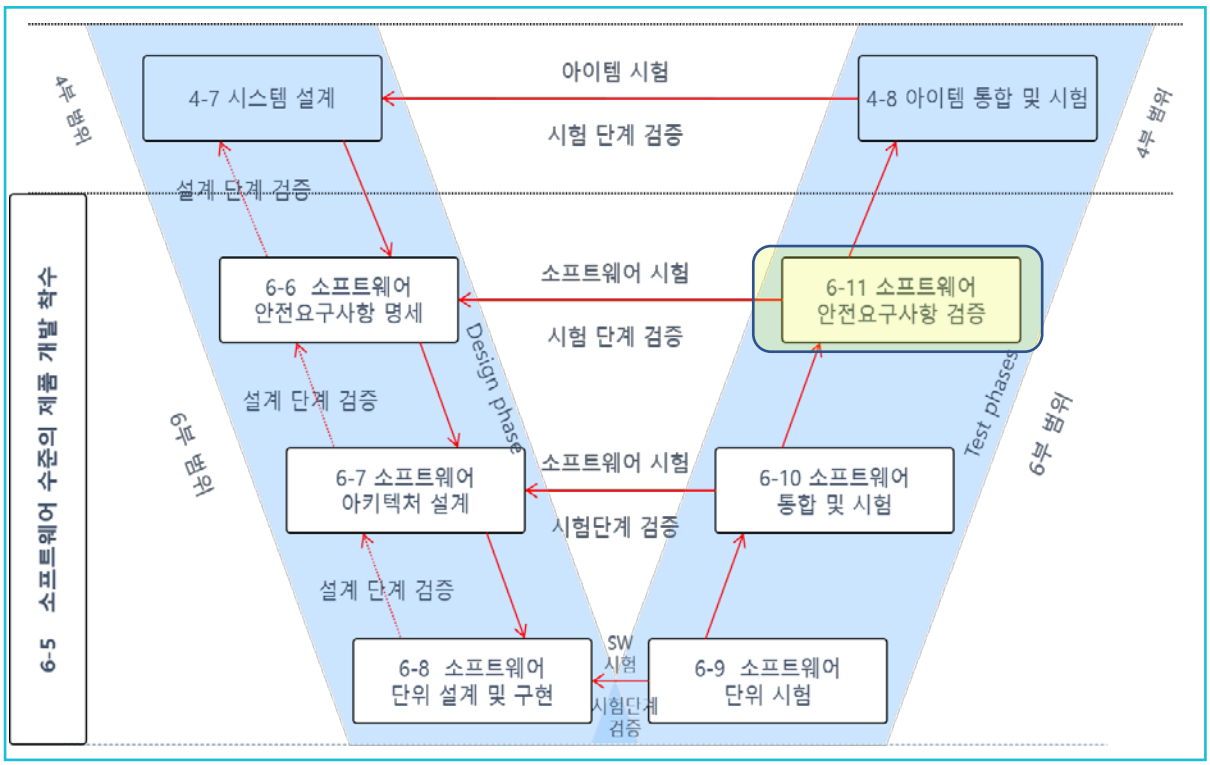
3) 구조 커버리지 지표

방법		ASIL			
		A	B	C	D
1a	함수(function)의 커버리지	+	+	++	++
1b	호출(call) 커버리지	+	+	++	++
++: 매우 권장, +: 권장, o: 권장사항 없음					

구조 커버리지 지표는 통합 시험 설계가 요구사항을 충분히 커버하는 지표이며, 함수 및 호출 커버리지로 측정한다. ASIL C 이상은 필수로 구조 커버리지를 측정하여 통합 시험 케이스가 충분히 도출되어 시험되었는지 확인한다.

3.7. 소프트웨어 안전 요구사항 검증

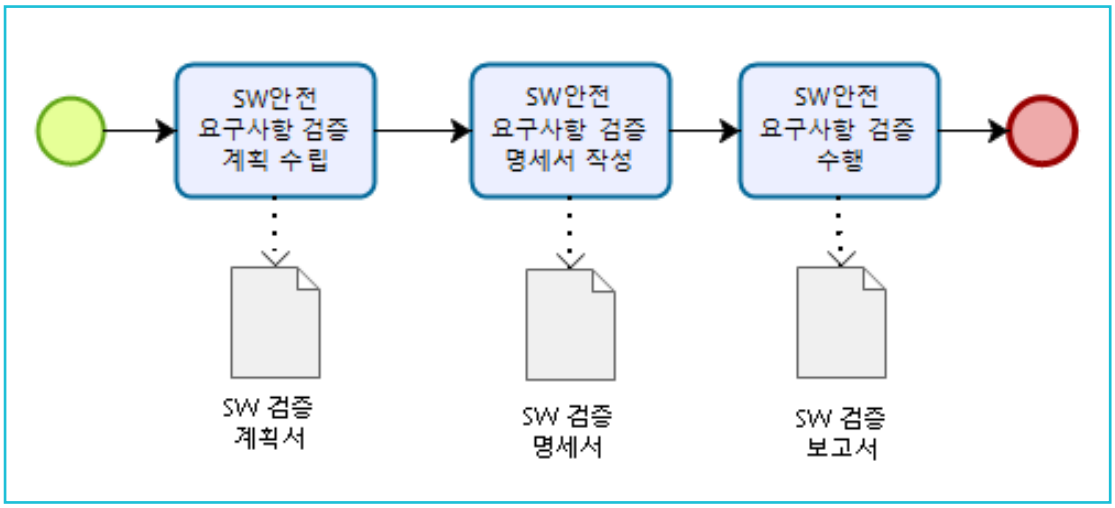
그림 37. 소프트웨어 개발 수명 주기 – 소프트웨어 안전 요구사항 검증



가. 수행 활동

소프트웨어 안전 요구사항 검증 단계에서는 통합하여 검증된 임베디드 소프트웨어가 소프트웨어 안전 요구사항과 부합하여 수행되는지 검증한다. 안전 요구사항 시험은 대상 하드웨어에서 수행하며, 개발된 소프트웨어가 실제 차량 및 운영 환경에서 동작하는지 확인하기 위하여 Hardware-In-The-loop, ECU 네트워크 환경과 차량에서 시험 환경을 구성해 확인해야 한다. 소프트웨어 안전 요구사항의 검증을 통해 소프트웨어 안전 요구사항의 커버리지가 충분한지 여부를 확인하고, 시스템 레벨에서 하드웨어와 소프트웨어 통합이 가능한 수준인지 평가한다. 소프트웨어 안전 요구사항의 검증이 통과되면 하드웨어와 소프트웨어를 통합하여 아이템 수준의 검증을 수행하는 단계로 진행할 수 있다.

그림 38. 활동 흐름 – 소프트웨어 안전 요구사항의 검증



나. 수행 활동 구성

구 분	설 명
목 적	● 임베디드 소프트웨어가 소프트웨어 안전 요구사항을 충족하는지 증명한다.
시작 기준	● 프로젝트 계획이 작성되고 승인되었다. ● 안전 계획이 작성되고 승인되었다. ● 기술안전 개념이 도출되고 시스템 설계 명세서에 반영되었다. ● 시스템 설계 명세서가 작성되고 승인되었다. ● 하드웨어와 소프트웨어 인터페이스 명세서가 작성되고 승인되었다. ● 모델링과 프로그래밍 언어를 위한 설계 및 코딩 지침이 작성되고 승인되었다. ● 소프트웨어 안전 요구사항 명세서가 작성되고 승인되었다. ● 소프트웨어 검증 계획이 수립되고 승인되었다.
입력물	● 소프트웨어 아키텍처 설계 명세서 ● 안전 계획(갱신) ● 소프트웨어 안전 요구사항 명세서(갱신) ● 소프트웨어 검증 계획(갱신) ● 소프트웨어 검증 명세서(갱신) ● 소프트웨어 검증 보고서(갱신)

	● 통합 시험 보고서
수행 활동	● 소프트웨어 안전 요구사항 검증 계획 수립 ● 소프트웨어 안전 요구사항 검증 수행 ● 소프트웨어 안전 요구사항 검증 보고
출력물	● 소프트웨어 검증 계획(갱신) ● 소프트웨어 검증 명세서(갱신) ● 소프트웨어 검증 보고서(갱신)
완료 기준	● 소프트웨어 검증 계획서가 작성되고 승인됨 ● 소프트웨어 검증 명세서가 작성되고 승인됨 ● 소프트웨어 검증 보고서가 작성되고 승인됨

다. 역할 및 책임

표 16. 역할 및 책임 – 소프트웨어 안전 요구사항 검증

역할 / 활동	안전 관리자	개발자	검증 담당자	품질 담당자	프로젝트 관리자
소프트웨어 안전 요구사항 검증 계획	I	I	R	C	A
소프트웨어 안전 요구사항 검증 명세	I	I	R	C	A
소프트웨어 안전 요구사항 검증 보고	I	I	R	C	A

* R: Responsible, 담당자, 해당 업무를 실행하는 주체 및 책임자
* A: Accountable, 의사 결정권자, 해당 업무 최종 승인자
* C: Consulted, 조언자, 업무 협의 또는 의견 제공자
* I: Informed, 공유 대상자, 해당 업무 참조자

라. 산출물

- 소프트웨어 검증 계획서: 소프트웨어 안전 검증 시험 계획을 말하며 시험하여 검증할 계획 수립한다. 검증 대상, 검증 방법, 검증 담당자, 검증 환경, 검증 통과 기준을 포함한다.

- 소프트웨어 검증 명세서: 소프트웨어 안전 검증 시험을 수행하기 위한 테스트 케이스를 설계하여 테스트 케이스 내역을 포함한다. 테스트 케이스는 테스트 사전 조건, 수행 절차, 테스트 통과 기준을 포함한다.
- 소프트웨어 검증 보고서 : 소프트웨어 안전 검증 시험에 대한 수행 결과를 포함한다. 테스트 수행 일정, 테스트 케이스별 수행 성공 및 실패 결과, 전체 테스트 케이스의 성공 비율 등을 포함하여 통합 시험 결과에 대해서 판단할 수 있는 정보를 포함한다.

마. 적용 기법

1) SW 안전 요구사항 검증 실시를 위한 시험 환경

방법		ASIL			
		A	B	C	D
1a	Hardware-In-the-Loop	+	+	++	++
1b	**ECU 네트워크 환경** [a]	++	++	++	++
1c	**차량**	++	++	++	++
++: 매우 권장, +: 권장, o: 권장사항 없음					

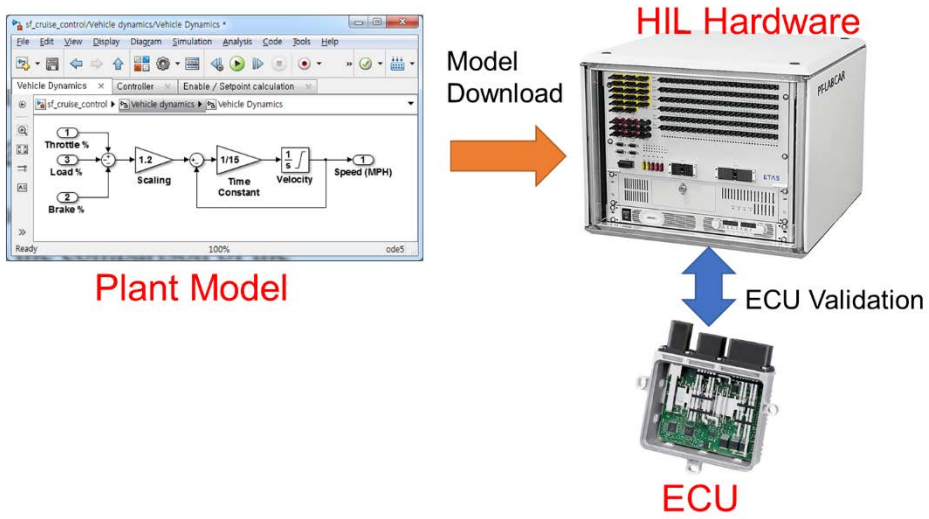
- Hardware-In-the-Loop

ECU 를 검증할 때 가장 정확한 방법은 ECU 를 완성된 차량에 직접 연결하여 테스트하는 것이다. 하지만 검증 초기 단계에 실차를 이용할 경우 (1) 실차가 준비되기 전에 테스트 불가능하고, (2) ECU 오동작이 실험 인원에 위험을 초래할 수 있고, (3) 테스트 케이스별로 실차 테스트 환경을 만드는데 많은 비용과 노력이 들어가는 등의 어려움이 있다. Hardware-In-the-Loop (HIL) 테스트는 위의 문제들을 해결하고 효율적인 ECU 초기 검증을 수행하기 위해 개발되었다. HIL 테스트 환경은 검증 대상 ECU 와 HIL 시뮬레이터로 구성된다. HIL 시뮬레이터는 물리 시스템 즉 실차를 모사하는 가상 환경을 제공한다. 여기에 연결된 ECU 는 마치 실차에 연결된 것과 같은 입출력 시그널을 CAN 과 같은 네트워크 혹은 디지털/아날로그 입출력 케이블을 통해 HIL 시뮬레이터와 주고받게 된다. HIL 시뮬레이터는 모사하고자 하는 물리시스템의 실시간 동작 행태를 모사하는 Plant 모델을 실시간 운영체제 위에 소프트웨어로 실행하여 ECU 와 입출력을 교환하면서 ECU 가 가상의 물리시스템과 상호작용하도록 한다. 예를 들어 엔진 ECU 를 테스트할 때는 연료 분사, 폭발, 배기와 같은 엔진의 물리적인 움직임을 소프트웨어로 모사하는 엔진 HIL 시뮬레이터가 사용되고 조향 제어 ECU 를 테스트할 때는 핸들, 조향 부품, 지면과 바퀴를 가상

환경에서 모사하는 조향 HIL 시뮬레이터가 사용된다. 이 외에도 최근 ADAS ECU 들은 가상의 도로 환경에서 차량의 종횡방향 거동, 주변 차량의 움직임, 환경 인지 센서값 등을 가상으로 모사하는 발전된 HIL 환경인 가상 드라이빙 시뮬레이터를 사용하여 검증한다.

아래 그림 39 는 HIL 테스팅의 개념을 그림으로 보여준다. 좌측의 차량의 동역학 모델(Plant 모델)이 HIL 하드웨어에 탑재되면 HIL 하드웨어는 마치 자신이 실제 차량인 것처럼 시뮬레이션을 수행하며 HIL 하드웨어와 연결된 ECU 와 상호작용을 하게 된다. ECU 는 직접 차량에 물려서 동작하는것과 같은 착각을 하게 된다. 시뮬레이션 환경이기 때문에 실제로는 만들기 힘든 위험하거나 드물게 발생하는 경우의 수를 만들어서 테스트할 수 있다. 충분한 검증을 위해서 최근에 많이 사용되고 있으며 ISO 26262 2 판에서는 모든 ASIL 에 대해서 Highly Recommended 로 변경되었다.

그림 39. Hardware-In-the-Loop 개념

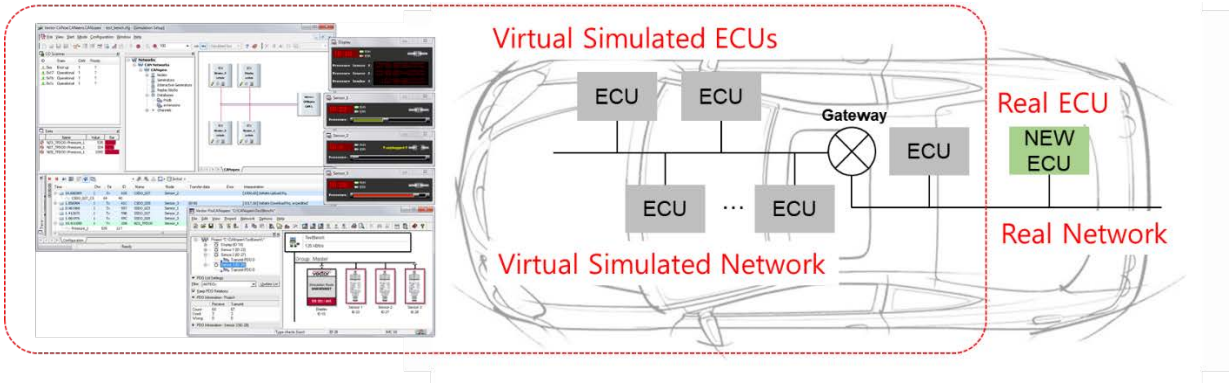


- ECU 네트워크 환경

ECU 는 독자적으로 동작하는 경우가 드물고 In-Vehicle Network 으로 연결된 다른 ECU 들과 협조하면서 동작하는 경우가 많다. 어떤 경우에는 다른 ECU 들의 도움 없이는 테스트케이스 실행이 불가능한 경우도 있다. 예를 들어 트랜스미션 제어 ECU 의 경우 변속시 엔진 제어 ECU 에 토크를 떨어뜨리기를 요청하고 이에 대응하여 변속 시점이 결정된다. 이와 같은 테스트케이스를 실행하기 위해서는 트랜스미션 제어 ECU 단독으로는 불가능하고 엔진 제어 ECU 와 연결된 상태에서 테스트를 실행해야 한다. 이와 같은 실험 환경을 만들기 위해서는 테스트용 차량을 만들거나 실험실 안에 ECU 들이 연결된 네트워크 환경을 만들어야 한다. 이때 실제 양산에 사용할 ECU 들을 네트워크에 연결하여 테스트할 수도 있지만 그 전에 그림 40 과 같은 "Rest of

the Bus" 시뮬레이션을 하는 것도 가능하다. "Rest of the Bus" 시뮬레이션은 개발한 ECU 를 PC 에 CAN 과 같은 네트워크로 연결하고 PC 에서 수행되는 시뮬레이션 소프트웨어가 CAN 버스의 나머지 부분 전체 (Rest of the Bus)를 시뮬레이션하는 것을 의미한다. Vector 사의 CANoe 소프트웨어가 대표적으로 이와 같은 기능을 제공한다. 이와 같은 시뮬레이션 소프트웨어는 시뮬레이션하는 ECU 의 세부 동작을 기술하기 위해 프로그래밍 언어를 이용해 간단한 로직을 기술할 수 있다.

그림 40. Rest of the Bus 시뮬레이션



- 차량

실제 차량에서 ECU 검증을 실행하는 것을 의미한다. 최근에는 시뮬레이션 기술이 발전하여 실제 차량보다 HIL 테스트를 많이 사용하기 때문에 ISO 26262 2 판에서는 ASIL A, B 에 대해서 Highly Recommended 에서 Recommended 로 중요도가 내려갔다. 하지만 이는 소프트웨어 검증 단계에서 실차 테스트의 중요성이 약해진 것이지 소프트웨어와 하드웨어가 결합된 시스템 테스트 단계에서는 실차 테스트가 가장 중요하다.

4. 지원 프로세스 가이드

4.1. 안전 요구사항 명세 및 관리

ISO 26262 파트 8, 6 절에서는 안전 요구사항 작성 및 관리하는 방법에 대해서 언급하고 있다. 안전 요구사항 명세 및 관리 대상이 되는 안전 요구사항은 파트 3 에서 작성되는 기능안전 요구사항, 파트 4 의 시스템 안전 요구사항, 파트 5 의 하드웨어 안전 요구사항, 파트 6 의 소프트웨어 안전 요구사항 모두 해당한다. 각 파트에서 안전 요구사항은 작성할 때 파트 8, 6 절의 안전 요구사항 명세 및 관리를 따르도록 하고 있다.

안전 요구사항의 표현 방법은 비정형 표기법, 준정형 표기법, 정형 표기법이 있으며 ASIL B 레벨은 비정형 표기법을 사용하도록 권고하고 있다. 비정형 표기법 및 준정형 표기법은 소프트웨어 요구사항, 소프트웨어 아키텍처 설계, 상세 설계에 공통적으로 사용하며 상세 내용은 3.3 소프트웨어 아키텍처 설계의 비정형 표기법, 준정형 표기법을 참고한다.

안전 요구사항을 작성할 때 각 안전 요구사항은 안전 목표로부터 동일한 ASIL 레벨을 할당 받는다. 기능안전 목표로부터 기능안전 요구사항, 시스템 요구사항, 하드웨어 요구사항, 소프트웨어 요구사항으로 각 ASIL 은 일관성이 있어야 하며 안전 요구사항에 해당 ASIL 을 표시해야 한다.

1) 안전 요구사항 명세

안전 요구사항을 작성할 때는 다음과 같은 특징 및 특성을 준수해야 한다. 각 안전 요구사항은 다음과 같은 특징을 갖도록 작성해야 한다. (ISO 26262-8, 6.4.2.4)

- 안전 요구사항은 모호하지 않고 이해하기 쉬어야 한다.
- 안전 요구사항은 더 이상 분해할 수 없는 수준이어야 한다. 두 개 이상의 요구사항으로 구분할 수 있다면 분리하여 작성한다.
- 안전 요구사항은 다른 요구사항들과 내부적으로 일관성이 있어야 한다. 특정 안전 요구사항을 구현하려면 다른 요구사항을 구현할 수 없는 경우는 일관성이 없는 것이다. 전체 요구사항이 모순되지 않도록 작성해야 한다.
- 안전 요구사항은 아이템의 제약 조건을 고려하여 구현 가능해야 한다.

- 안전 요구사항은 검증 단계에서 검증할 수 있어야 한다.

안전 요구사항은 다음과 같은 특성이 있어야 한다. (ISO 26262-8, 6.4.2.5) 안전 요구사항의 특성은 안전 요구사항 작성에 필수적으로 포함해야 하는 항목이다.

- 안전 수명주기 동안 변경되지 않고 지속되는 고유 식별자
- 안전 요구사항 상태: 제안, 승인, 허용, 검토가 될 수 있다.
- 안전 요구사항에 할당된 ASIL 레벨

2) 안전 요구사항 관리

안전 요구사항은 관리 측면에서 다음과 같은 속성이 있어야 한다. (ISO 26262-8, 6.4.3.1)

- 요구사항 계층 구조: 계층 구조는 안전 요구사항이 여러 가지 연속 수준으로 구성된다는 것을 의미한다. 상위의 안전 요구사항과 하위의 안전 요구사항이 서로 부합되어야 한다.
- 요구사항 그룹화: 안전 요구사항은 아키텍처에 따라서 그룹화 및 분류하여 관리한다. 일반적으로 아키텍처에서 식별된 컴포넌트 단위로 그룹화할 수 있다.
- 완전성: 완전성은 하위의 요구사항이 상위의 요구사항을 모두 구현한다는 것을 의미한다. 예를 들면, 소프트웨어 안전 요구사항은 상위의 시스템 요구사항에서 식별된 소프트웨어 해당 부분을 모두 포함해야 한다.
- 외부 일관성: 소프트웨어의 전체 요구사항 수준만 아니라 소프트웨어 요구사항과 하드웨어 요구사항, 시스템 요구사항과 같은 외부 요구사항과 일관성이 있어야 한다.
- 외부적인 측면의 일관성: 외부 일관성은 다수의 안전 요구사항이 서로 모순되지 않는다는 것을 의미한다.
- 계층 구조에서 중복 없음: 안전 요구사항은 상위, 하위로 연결되는 계층 구조를 갖는데 소프트웨어 수준의 안전 요구사항(수평 측면)과 시스템 수준(수직 측면)에서 모두 중복되는 요구사항이 없는 것을 의미한다.
- 계층 구조의 어떤 수준 내에서도 중복되는 정보가 없음. 중복되는 정보가 없다는 것은 안전 요구사항의 내용이 계층 구조를 가지는 단일 수준의 안전 요구사항에 대하여

반복되지 않으며(수평적인 측면), 그 내용이 모든 계층 수준에서 참(true)인 것(수직적인 측면)을 의미한다.

안전 요구사항은 개발 단계에 요구사항이 반영되고 있는지 추적해야 한다. 요구사항 추적을 위해서 다음 항목이 반영되어 한다. (ISO 26262-8, 6.4.3.2) 안전 요구사항과 설계, 검증 및 시험 각 단계에서 요구사항이 추적되어야 한다. 또한, 안전 요구사항의 변경이 발생하는 경우에 수행하는 영향 분석과 기능안전 평가 활동도 연계되어 추적이 가능해야 한다.

- 상위 계층 수준에서 안전 요구사항 각각의 출처
- 하위 계층 수준 또는 설계 단계에서 구현에 관하여 도출된 개별적인 안전 요구사항
- 검증 명세서

4.2. 형상관리

형상관리는 안전수명주기 동안에 작성되는 요구사항 명세서, 설계서 등의 산출물에 대한 작성 규칙을 수립한다. 형상관리를 위한 계획을 수립하고 소프트웨어 개발 단계를 비롯한 시스템 개발 단계, 하드웨어 개발 단계 등 전체 안전 수명주기 동안 수행되어야 한다. 형상관리를 통해서 산출물이 임의로 변경되지 않고 개발 전체 활동 동안에 무결성을 갖도록 한다. 산출물의 변경 이력을 통해서 이전 버전과 현재 버전의 변경 내역에 대해서 추적할 수 있다. ISO 26262 에서는 구체적인 방법을 언급하고 있지 않지만 IATF 16949, ISO 10007, ISO/IEC 12207 에 따라 수행할 수 있다. SW 안전 공통가이드의 5.3.2 SW 안전 기록 작성 IEC61508-3 A.8.5: 소프트웨어 형상 관리(Software configuration management)를 참고한다.

4.3. 변경관리

변경관리는 안전 수명주기 동안에 변경이 발생하는 경우에 산출물의 변경사항을 분석하고 관리하는 활동이다. 변경 요청이 발생되면 바로 변경하는 것이 아니라 기능 안전에 미치는 영향을 평가하여 변경 여부에 대해서 의사결정 하는 절차가 필요하다. 소프트웨어 컴포넌트의 결함 발생, 기능 변경, 기능 추가 등으로 인해 변경이 발생할 수 있다.

변경 절차는 변경 요청, 변경 요청 분석, 변경 요청 결정의 근거 기록, 변경 적용 및 구현, 변경 결과 문서화 순서로 수행한다. 변경 요청 시에는 요청 ID, 요청 날짜, 변경 이유, 변경 내용, 변경해야 하는 형상항목 및 산출물을 포함해야 한다. 변경 요청 분석 시에는 변경의 유형(예: 에러 해결, 개선, 예방)과 변경에 영향을 미치는 관련자, 기능 안전에 미치는 영향, 구현 및 검증 일정을 포함해야 한다. 영향 분석의 결과를 바탕으로 변경 요청을 평가하여 관련자들의 변경 수락, 거절, 연기에 대한 결정을 해야 한다. 변경은 적용 이후에는 적절하게 변경되었는지, 변경 과정에 오류가 발생하지 않았는지 파악하기 위해 검증을 수행해야 한다. 검증 이후에는 변경된 산출물 목록, 변경 세부 사항, 변경 일정 등을 기록하여 변경에 대한 이력을 관리한다

4.4. 검증

검증은 ISO 26262 파트 8 에서는 요구하는 활동으로 개념 단계, 시스템 수준 개발 단계, 하드웨어 수준 개발 단계, 소프트웨어 수준 개발 단계에서 작성되는 산출물에 공통적으로 적용된다. 각 단계에서 개발되는 산출물들이 산출물이 정확하게 작성되었는지, 일관성은 있는지 등에 대해서 확인한다. 검증의 방법으로는 리뷰, 워크쓰루, 인스펙션, 모델 체킹, 시뮬레이션, 공학적 분석, 시험이 있다.

검증을 수행하기 위해서는 검증 대상이 되는 산출물과 검증에 사용하는 방법, 검증 환경을 포함하여 검증 계획을 수립해야 한다. 검증 계획에는 합격 및 불합격 기준을 포함하여 검증 종료 시에 검증 통과 여부를 판별할 수 있게 한다. 검증 환경은 시험이나 시뮬레이션에 필요한 도구, 장비 등이 포함된다. 시험의 경우에는 회귀 전략이 포함되는데 아이템이나 엘리먼트의 변경이 발생하는 경우에 재검증을 어떻게 할 것인지 방법을 포함한다. 회귀 전략에는 전체적으로 다시 재검증할 것인지, 일부만 재검증할 것인지를 선택하는 기준을 포함해야 한다. 변경이 발생하는 부분에 대해서 영향 분석과 해당 변경이 발생하는 컴포넌트와 의존성을 고려하여 재검증 범위를 선정할 수 있다.

검증 수행 시에 검증 절차는 검증 명세서로 작성되는데 시험인 경우에는 테스트케이스가 된다. 소프트웨어 단위 시험에서는 단위 테스트 케이스, 소프트웨어 통합 시험에서는 통합 시험 테스트케이스가 검증 명세서로 작성된다. 검증을 시뮬레이션으로 수행하면 시뮬레이션 시나리오가 검증 명세서가 된다. 테스트 케이스는 작성 시에 각 테스트 케이스 항목별로 고유한 식별자가 있어야 하며, 테스트 사전 조건, 테스트 환경 조건, 입력 데이터, 출력 예상 값을 포함한다. 테스트 케이스 관리를 위해서 시험 방법에 따라서 그룹화하여 관리한다.

검증을 수행한 후에는 검증 결과를 문서화해야 하는데 시험의 경우는 단위 시험 결과, 통합 시험 결과가 해당 된다. 검증 결과에는 다음 정보를 포함해야 한다. (ISO 26262-8, 9.4.3.2)

- 검증된 작업 산출물의 고유식별
- 검증 계획 및 검증 명세서에 관한 참조자료
- 검증 환경과 사용된 검증 도구의 설정 및 해당되는 경우, 평가하는 동안 사용되는 보정 데이터
- 예상된 결과에 대한 검증 결과의 부합화 수준

검증에 대한 합격 또는 불합격에 대한 명확한 판단 기준(검증 결과가 불합격이면, 불합격에 대한 근거 및 검증된 작업 산출물들의 변경을 위한 제안사항 포함)

4.5. 문서화

문서화는 ISO 26262 파트 8 에서 요구하는 활동으로 전체 기능안전 수명주기에서 작성되는 모든 문서들이 적용 대상이다. 기능안전 수명주기에서 작성되는 문서들은 반드시 종이로 작성될 필요는 없으며 문서 자동화 도구와 같은 온라인 문서도 사용될 수 있다.

소프트웨어 개발 활동을 비롯해 전체 기능안전 수명주기에서 작성되는 문서는 기능안전 평가 자료로 활용되기 때문에 기본적으로 문서의 구조나 형식이 일정 요건을 갖고 있어야 한다. 문서 간에 서로 중복을 피하고 문서의 가독성을 유지해야 한다.

ISO 26262 문서 작성 시에 다음을 고려해야 한다. (ISO 26262-8, 10.4.3)

- 정확하고 간결하다.
- 깔끔한 방식으로 구성된다.
- 의도된 사용자가 이해하기 쉽다.
- 유지관리 될 수 있다.

ISO 26262 문서에는 다음 항목들을 포함해야 한다. (ISO 26262-8, 10.4.5)

- 제목, 내용의 범위 참조
- 작성자와 승인자
- 여러 문서 개정(버전)의 고유식별: 변경 이력은 변경이 발생할 때마다 작성자명, 날짜, 개요에 관한 정보를 포함한다. 상태는 초안(Draft), 발행(released)과 같이 작성한다.

4.6. 소프트웨어 도구 사용의 신뢰성 확보

기능 안전 소프트웨어 개발 활동에는 문서 자동화 도구, 기능안전 관리 도구, 시험 도구 등 여러 가지 도구가 사용된다. 각 도구에는 상용 도구, 오픈 소스 도구, 사내에서 개발한 도구를 사용할 수 있다. 이러한 도구들은 도구의 중요도 및 영향에 따라서 개발되는 소프트웨어의 신뢰성에 영향을 미칠 수 있다. 예들 들면, 정적 분석 도구에서 소스 코드의 오류를 적절하게 탐지하지 못하거나 단위 시험 커버리지 측정 도구에서 잘못된 커버리지 값을 측정한다면 개발되는 소프트웨어에 오류가 생길 수 있다. 이러한 소프트웨어 도구의 오류로 인한 기능안전 소프트웨어의 오류 영향과 리스크를 줄이기 위해서 ISO 26262 파트 8 에서는 소프트웨어 도구의 신뢰성을 확보하는 활동을 요구한다.

소프트웨어 도구 사용의 신뢰성 확보는 소프트웨어 도구를 사용하는 입장에서는 상용으로 구매할 수 있는 도구인 경우에 도구의 신뢰성 분류에 따라서 개발사로부터 ISO 26262 에 적용하기 위한 가이드 또는 ISO 26262 기능안전 개발에 적합하다는 제 3 자 인증서를 통해 신뢰성을 확인할 수 있다. 오픈 소스 및 프로젝트 팀에서 자체적으로 개발하여 사용하는 도구는 신뢰성 분류에 따라서 사용 사례, 개발 프로세스, 타당성 시험을 통해 평가해야 한다. 소프트웨어 도구 신뢰성 평가가 수행되면 ISO 26262 파트 2 에서 언급된 확인 검토(Confirmation Review)를 통해서 소프트웨어 도구 기준 평가 보고서와 소프트웨어 도구 인정 보고서에 대해서 평가해야 한다. 소프트웨어 도구의 평가 기준에는 도구의 사용 현황, 사용 환경, 기능적 제약 사항, 일반 작동 조건 등을 고려해야 한다.

소프트웨어 도구를 사용하기 위해서는 다음과 같은 사항을 포함하여 계획을 수립한다. (ISO 26262-8, 11.4.4.1

- 소프트웨어 도구의 식별 및 버전 번호

- 소프트웨어 도구의 설정 정보: 소프트웨어 도구의 유즈 케이스는 소프트웨어 도구 혹은 적용되는 소프트웨어 도구의 기능이 적용되는 부분 집합에 관한 사용자의 상호 작용을 기술할 수 있다. 유즈 케이스는 소프트웨어 도구가 실행되는 설정 및 환경에 대한 요구사항을 포함할 수 있다.
- 소프트웨어 도구가 실행되는 환경
- 소프트웨어 도구 오류로 인한 위반될 수 있는 아이템이나 엘리먼트에 할당된 안전 요구사항의 최대 ASIL
- 필요 시에 소프트웨어 도구를 검증하는 방법

소프트웨어 도구에 대한 적절한 평가를 위해서 다음 정보가 있어야 한다. (ISO 26262-8, 11.4.4.2)

- 소프트웨어 도구의 특징, 기능, 기술적인 속성에 대한 설명
- 해당되는 경우 사용자 매뉴얼 및 기타 사용 지침
- 동작에 요구되는 환경
- 해당되는 경우, 이상 작동 조건에서 소프트웨어 도구의 예상 동작
- 알려진 소프트웨어 도구의 오작동, 적합한 보호장치, 해당되는 경우, 방지 또는 기타 해결 수단에 대한 설명

소프트웨어 도구를 분석에 의해서 평가할 때 다음의 정보를 포함하여야 한다. (ISO 26262-8, 11.4.5)

- 의도한 목적: 기능의 시뮬레이션, 소스코드 생성 또는 임베디드 소프트웨어의 시험, 안전 수명주기의 조정 또는 ISO 26262 에서 요구하는 활동 및 업무의 단순화나 자동화
- 입력사항과 예상되는 출력: 이후의 개발 활동, 소스코드, 시뮬레이션 결과, 시험 결과, 기타 ISO 26262 의 작업 산출물에 대하여 필요한 입력 데이터
- 해당되는 경우, 환경 및 기능 제약

소프트웨어 도구의 평가는 다음과 같이 수행한다.(ISO 26262-8, 11.4.5.2)

- 개발중인 안전 관련 아이템이나 엘리먼트에서 특정 소프트웨어 도구가 오류를 일으키거나 오류 탐지에 실패할 오류의 가능성: 오류 확률이 없다는 근거가 있으면 TI(Tool Impact)1 으로 선택한다. 그 외의 모든 경우에는 TI2 가 선택된다.
- 소프트웨어 도구가 오작동하고 해당 오류출력 생성을 방지하는 수단 또는 소프트웨어 도구가 오작동하고 오류출력을 검출하기 위하여 사용되는 수단의 신뢰 정도: 오작동과 해당 오류출력이 방지되거나 검출되는 신뢰수준이 높으면 TD(Tool error Detection)1 으로 선택하고, 오작동과 해당 오류출력이 방지되거나 검출되는 신뢰수준이 중간이면 TD2 가 된다. 그 외의 모든 경우에는 TD3 가 된다. 개발 프로세스에서 가능한 시스템적 수단이 없으면 TD3 가 기본적으로 적용된다. 소프트웨어 도구가 다른 소프트웨어 도구로부터의 출력을 검증하는데 사용되는 경우 후속단계의 소프트웨어 도구의 평가를 통하여 소프트웨어 도구간의 상호 의존성이 고려되고, 이후에 사용되는 도구에 관하여 적절한 TD 가 선택된다. 생성된 소스 코드가 ISO 26262 에 따라 검증된 경우 코드 생성기에서 TD1 을 선택할 수 있다.

TI 와 TD 의 등급이 결정되면 다음 도구의 신뢰 수준 결정 표에 따라서 TCL 을 결정한다

표 17. 도구의 신뢰수준(TCL)의 결정

구분		도구 오류 검출		
		TD1	TD2	TD3
도구 영향	TI1	TCL1	TCL1	TCL1
	TI2	TCL1	TCL2	TCL3

도구의 신뢰 수준(TCL)에 따라서 TCL2 와 TCL3 가 소프트웨어 도구 인정을 수행해야 하고, TCL1 인 경우는 해당되지 않는다. TCL3 는 다음 표와 같이 ASIL 레벨에 따라서 TCL3, TCL2 에 적합한 소프트웨어 도구 인정 방법을 적용한다. ASIL 레벨 B 에서는 사용으로 인하여 증가된 신뢰, 도구 개발 프로세스의 평가 방법이 적용된다.

표 18. TCL3 로 분류된 소프트웨어 도구의 인정

방법		ASIL			
		A	B	C	D
1a	**사용으로 인하여 증가된 신뢰**	++	++	+	+
1b	**도구 개발 프로세스의 평가**	++	++	+	+
1c	소프트웨어 도구의 타당성 확인	+	+	++	++
1d	안전 표준에 따라 개발	+	+	++	++
++: 매우 권장, +: 권장, o: 권장사항 없음					

표 19. TCL2 로 분류된 소프트웨어 도구의 인정

방법		ASIL			
		A	B	C	D
1a	**사용으로 인하여 증가된 신뢰**	++	++	+	+
1b	**도구 개발 프로세스의 평가**	++	++	+	+
1c	소프트웨어 도구의 타당성 확인	+	+	++	++
1d	안전 표준에 따라 개발	+	+	++	++
++: 매우 권장, +: 권장, o: 권장사항 없음					

사용으로 인하여 증가된 신뢰는 다음 정보들이 있는 경우에 사용으로부터 신뢰가 증가한 것으로 본다.

- 과거에 동일한 목적으로 소프트웨어 도구가 상당한 양의 유즈 케이스, 결정된 작동 환경, 유사한 기능 제약 상황에서 사용되었다
- 사용을 바탕으로 하는 증가된 신뢰에 관한 정당화는 충분하고 적합한 데이터를 기반으로 한다. 데이터는 축적된 사용량을 통해 획득된다.
- 소프트웨어 도구의 명세서는 변경되지 않는다.
- 이전에 개발하는 동안 습득된 소프트웨어 도구에 관한 오작동의 발생과 해당 오류출력은 시스템적인 방식으로 축적된다.

도구 개발 프로세스의 평가는 소프트웨어 도구 개발을 위해 적용되는 개발 프로세스가 적합한 표준을 따랐는지 평가하는 것이다. 예를 들면, A-SPICE, CMMI, ISO 15504 등의 표준으로 따라서 개발되었는지 평가하고, 오픈 소스의 경우는 커뮤니티가 사용하는 표준도 적용 가능하다.

소프트웨어 도구의 타당성 확인은 도구의 기능 및 비기능 품질을 평가하기 위해 시험을 수행한다. 도구 시험을 통해서 소프트웨어 도구가 명시된 요구사항에 부합하는지 증명하고, 도구의 이상 작동 조건에 대해서 파악해야 한다.

소프트웨어 도구에 관한 인정은 다음 사항을 포함하여 문서화되어야 한다. (ISO 26262-8, 11.4.6.2)

- 소프트웨어 도구의 고유식별 및 버전 번호
- 소프트웨어 도구의 평가 분석에 대한 참고사항을 이용하여 분류된 소프트웨어에 관한 최대 도구 신뢰수준(TCL)
- 소프트웨어 도구가 오작동하거나 해당 오류출력을 생산할 때, 위반될 수 있는 안전 요구사항 중 사전 결정된 최대 ASIL 또는 특정 ASIL
- 소프트웨어 도구에 관한 인정을 위한 설정 및 환경
- 안정을 수행하는 개인이나 조직
- 인정에 적용되는 방법
- 소프트웨어 도구의 인정에 적용되는 방법
- 소프트웨어 도구의 인정에 적용되는 수단의 결과
- 해당되는 경우, 인정 수행 중에 식별된 사용 제약 및 오작동

4.7. 소프트웨어 컴포넌트 인정

소프트웨어 컴포넌트 인정은 소프트웨어 개발 단계에서 신규로 개발하지 않고 기존에 개발된 컴포넌트가 ISO 26262 기능안전 수명주기를 준수하지 않고 개발된 된 것으로 재사용하는 경우에 적용한다. 재사용하는 컴포넌트가 기능안전 소프트웨어 개발에 적합한지, 해당 재사용 컴포넌트가 소프트웨어에 오류를 발생시키지 않는지 영향을 평가하는 활동이다.

소프트웨어 컴포넌트를 재사용하기 위해서는 다음 정보가 확인 가능해야 한다. (ISO 26262-8, 12.4.1)

- 소프트웨어 컴포넌트의 명세
- 소프트웨어 컴포넌트가 요구사항을 준수한다는 증거 자료
- 소프트웨어 컴포넌트가 의도된 사용에 적합하다는 증거자료
- 컴포넌트를 위한 소프트웨어 개발 프로세스가 적합한 국내 표준이나 국제 표준을 기반으로 한다는 증거 자료

소프트웨어 컴포넌트의 인정 계획에는 다음 사항을 포함 한다. (ISO 26262-8, 12.4.2.1)

- 소프트웨어 컴포넌트의 고유 식별자
- 소프트웨어 컴포넌트가 부정확하게 수행되는 경우 위반할 수 있는 안전 요구사항의 최대 목표 ASIL
- 소프트웨어 컴포넌트를 인정하기 위하여 수행되는 활동

소프트웨어 컴포넌트의 명세는 다음 사항을 포함하여야 한다. (ISO 26262-8, 12.4.3.1)

- 소프트웨어 컴포넌트의 요구사항 : 기능 요구사항, 고장인 경우의 행위, 반응 시간, 자원 사용, 런타임 환경에 대한 요구사항, 과부하 상태의 행위(강건성) 등
- 소프트웨어 컴포넌트 설정에 대한 설명
- 인터페이스에 대한 설명
- 필요 시, 적용 매뉴얼
- 소프트웨어 컴포넌트 통합에 대한 설명
- 이상 작동 조건 시 기능의 반응 : 재진입이 불가한(non-re-entrant) 소프트웨어 컴포넌트 기능의 재진입 요청
- 다른 소프트웨어 컴포넌트와의 종속성
- 관련된 기타 해결수단과 알려진 이상점에 대한 설명

소프트웨어 컴포넌트의 검증에서는 다음 항목을 수행한다. (ISO 26262-8, 12.4.3.2)

- ISO 26262-6, 9 절에 따라 요구사항 커버리지를 보여주어야 한다. : 이 검증은 주로 요구사항 기반 시험을 바탕으로 한다. 개발하는 동안 또는 개발 이전 통합 시험을 실시하는 동안 실행된 소프트웨어 컴포넌트의 요구사항에 기반한 시험 결과가 사용될 수 있다.
- 고장의 경우, 정상 작동 조건 및 행위 모두를 포함하여야 한다.
- 안전 요구사항의 위반을 야기하는 알려진 오류가 없음을 보여주어야 한다.

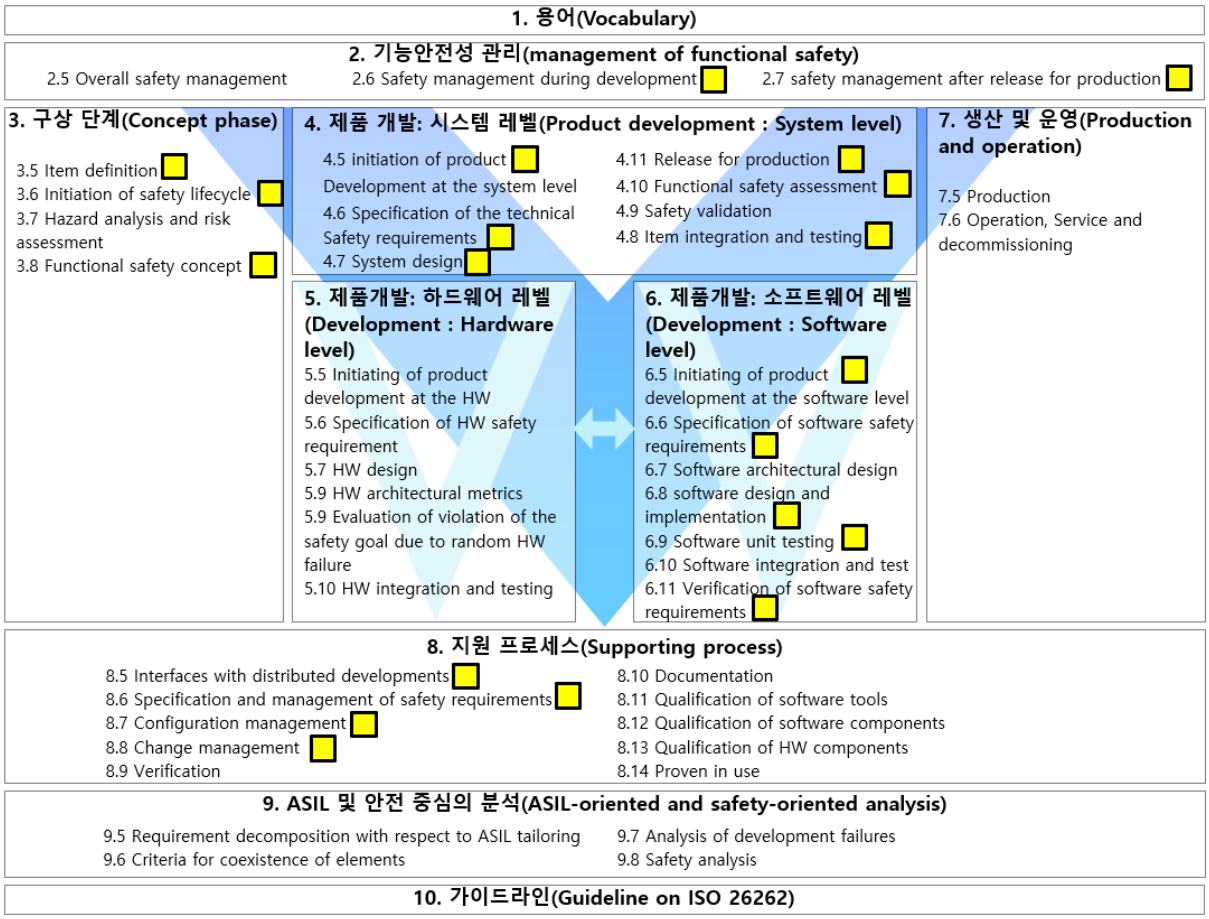
소프트웨어 컴포넌트의 인정은 다음과 같은 정보를 포함하여 문서화되어야 한다. (ISO 26262-8, 12.4.3.5)

- 소프트웨어 컴포넌트의 고유 식별
- 소프트웨어 컴포넌트의 고유 설정
- 인정을 실행하는 개인이나 조직
- 인정에 사용되는 환경
- 소프트웨어 컴포넌트에 대한 인정을 실시하는데 적용된 검증 수단의 결과
- 소프트웨어 컴포넌트가 잘못 실행되는 경우, 위반될 수 있는 안전 요구사항의 최대 목표 ASIL

5. ISO 26262 와 A-SPICE 공통 부분

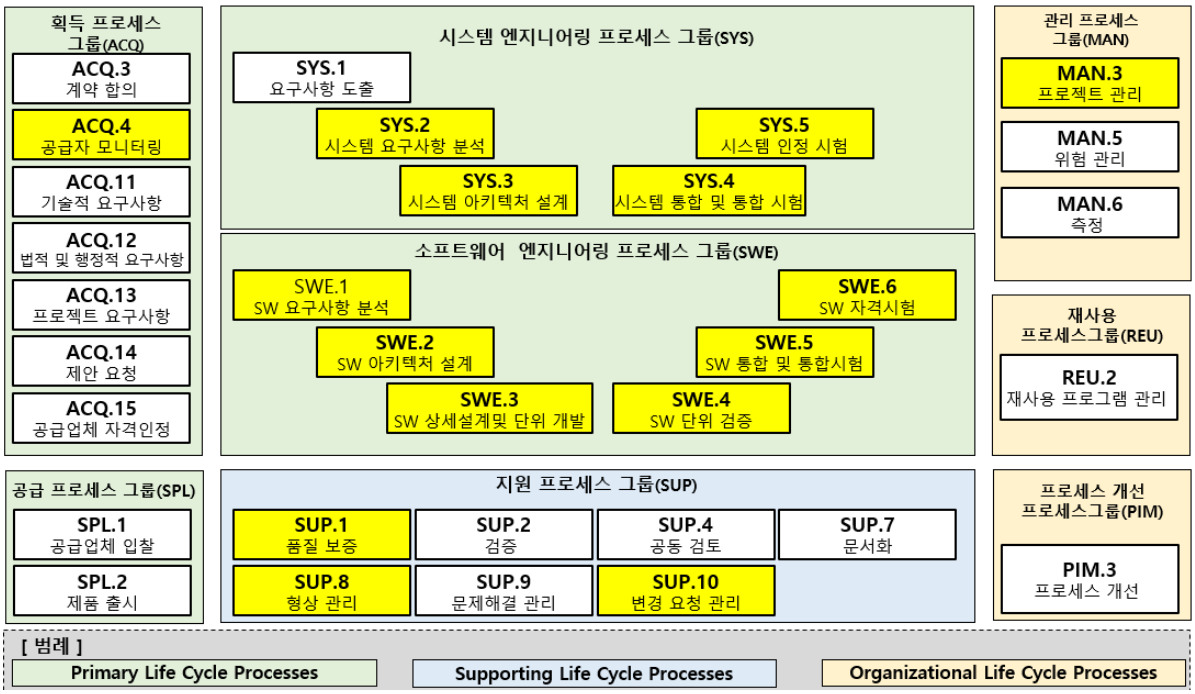
ISO 26262 전체 구조에서 독일자동차협회(VDA) Scope 부분을 표시하면 그림 41 과 같다. ISO 26262 는 기능 안전 관점에서 제품을 개발하고, A-SPICE 는 일반적인 제품을 개발하는 활동이므로 공통 부분의 의미는 활동이 유사하다는 것으로 세부적인 모든 활동이 일치하다는 것은 아니다. 다만, 공통 부분은 ISO 26262 와 A-SPICE 를 동시에 적용해야 하는 경우에 유사한 부분으로 참고할 수 있다.

그림 41. ISO 26262 와 A-SPICE 공통 부분



유럽 OEM 의 주요 평가 범위인 VDA Scope 범위 측면에서 그림 42 와 같이 노란색 부분으로 공통 부분을 표시하였다. 시스템 엔지니어링 프로세스 그룹은 시스템 요구사항을 도출하고 시스템 아키텍처 설계 및 통합, 시험하는 것으로 ISO 26262 파트 4 시스템 수준 개발 활동과 유사하다. 소프트웨어 엔지니어링 프로세스 그룹은 소프트웨어 요구사항 도출하고 소프트웨어 아키텍처 설계 및 단위 개발, 단위 검증, 시험하는 것으로 ISO 26262 파트 6 소프트웨어 수준 개발 활동과 유사하다. 지원 프로세스 그룹은 SUP.8 형상관리가 ISO 26262 파트 8 지원 프로세스의 형상관리와 유사하고, SUP.10 변경 요청관리는 ISO 26262 지원 프로세스의 변경관리와 유사하다. 공급자 모니터링은 협력업체의 개발 활동을 모니터링 하는 것으로 ISO 26262 파트 8 지원 프로세스의 분산 개발 환경의 요구사항 일부와 유사하다.

그림 42. A-SPICE VDA Scope 와 ISO 26262 공통 부분



A-SPICE 와 ISO 26262 의 공통 수준은 아래 표와 같다.

표 20. A-SPICE VDA Scope 와 ISO 26262 공통 수준

A-SPICE	ISO 26262	공통 수준
SYS.1 요구사항 도출	아이템 정의	○
SYS.2 시스템 요구사항 분석	기능 안전 컨셉	◇
	기술 안전 요구사항 명세	◇
	기능 안전 요구사항 명세 및 관리	△
SYS.3 시스템 아키텍처 설계	시스템 설계	△
SWE.1 소프트웨어 요구사항 분석	소프트웨어 안전 요구사항 명세	△
SWE.2 소프트웨어 아키텍처 설계	소프트웨어 아키텍처 설계	△
SWE.3 소프트웨어 상세 설계 및 단위 개발	소프트웨어 단위 설계 및 구현	△
SWE.4 소프트웨어 단위 검증	소프트웨어 단위 시험	△
SWE.5 소프트웨어 통합 및 통합 시험	소프트웨어 통합 및 시험	△
SWE.6 소프트웨어 인정 시험	소프트웨어 안전 요구사항 검증	△
SYS.4 시스템 통합 및 통합 시험	아이템 통합 및 시험	△
SYS.5 시스템 인정 시험	해당 없음	
ACQ.4 공급업체 모니터링	분산 개발 인터페이스	△
MAN.3 프로젝트 관리	개념 단계 및 제품 개발 동안의 안전 관리	△
	안전 수명주기 착수	△
	시스템 수준 개발 착수	△
	소프트웨어 수준 개발 착수	△
SUP.1 품질보증	개념 단계 및 제품 개발 동안의 안전 관리	△
	기능 안전 평가	△
SUP.8 형상관리	형상관리	△
SUP.9 문제 해결 관리	해당 없음	
SUP.10 변경 요청 관리	변경관리	△

○: 높음 △: 중간 ◇: 낮음

ISO 26262 와 A-SPICE 는 적용 대상이 다르기 때문에 ISO 26262 를 따른다고 해서 A-SPICE 를 충족한다고 볼 수는 없지만 ASIL 레벨별 적용 기법 측면에서 A-SPICE 적용 시에 활용할 수 있는 공통 부분들이 존재한다. 본 가이드에서 설명한 ISO 26262 소프트웨어 수준의 개발 측면에서 A-SPICE 적용 시에 공통적으로 활용할 수 있는 부분은 다음과 같다.

1.1 SWE.1 소프트웨어 요구사항 분석

소프트웨어 요구사항 분석에서는 시스템 요구사항 중에서 소프트웨어 관련 있는 부분을 도출하고 분석 및 검증하는 것으로 다음과 같은 활동을 수행해야 한다.

- SWE.1.BP1: 소프트웨어 요구사항을 명세한다.
- SWE.1.BP2: 소프트웨어 요구사항을 구조화한다.
- SWE.1.BP3: 소프트웨어 요구사항을 분석한다.
- SWE.1.BP4: 운영 환경에 미치는 영향을 분석한다.
- SWE.1.BP5: 검증 기준을 개발한다.
- SWE.1.BP6: 양방향 추적성을 수립한다.
- SWE.1.BP7: 일관성을 보장한다.
- SWE.1.BP8: 합의된 소프트웨어 요구사항을 의사소통한다.

. 이 중에서 BP1~BP3 의 요구사항 명세 및 구조화와 BP6~BP7 의 양방향 추적성 및 일관성 보장이 ISO 26262 파트 8 안전 요구사항의 명세 및 관리와 유사하다. 안전 요구사항의 명세, 검증 기준, 추적성 수립 항목을 참고하여 A-SPICE 에 적용되는 개발 제품의 요구사항 분석 및 추적성 관리 활동에 적용할 수 있다.

1.2 SWE.2 소프트웨어 아키텍처 설계

소프트웨어 아키텍처 설계는 소프트웨어 설계를 개발하여 요구사항을 어떤 엘리먼트에서 구현할 것인지 할당하고 설계를 평가하는 것으로 다음과 같은 활동을 수행한다.

- SWE.2.BP1: 소프트웨어 아키텍처 설계를 개발한다.
- SWE.2.BP2: 소프트웨어 요구사항을 할당한다.
- SWE.2.BP3: 소프트웨어 앨리먼트의 인터페이스를 정의한다.
- SWE.2.BP4: 동적 행태를 서술한다.
- SWE.2.BP5: 자원 소모 목표를 정의한다.
- SWE.2.BP6: 대안의 소프트웨어 아키텍처를 평가한다.
- SWE.2.BP7: 양방향 추적성을 수립한다.
- SWE.2.BP8: 일관성을 보장한다.
- SWE.2.BP9: 합의된 소프트웨어 아키텍처 설계를 의사소통한다.

이 중에서 BP1~BP4 의 소프트웨어 아키텍처 설계 및 요구사항 할당 시에 ISO 26262 에서 요구하는 소프트웨어 아키텍처 설계 표기법의 비정형 표기법, 준정형 표기법을 적용하여 아키텍처를 명세할 수 있다. 또한, 소프트웨어 아키텍처 설계 원칙인 소프트웨어 컴포넌트의 계층적 구조, 컴포넌트의 제한된 크기, 높은 응집도, 제한된 결합도를 적용하면 소프트웨어 아키텍처의 신뢰성을 높일 수 있으며 BP6 소프트웨어 아키텍처 평가의 평가 기준으로 활용할 수 있다. 평가 방법으로는 설계 검증 방법인 워크쓰루, 인스펙션 방법을 활용할 수 있다.

1.3 SWE.3 소프트웨어 상세 설계 및 단위 개발

소프트웨어 상세 설계 및 단위 개발은 소프트웨어 아키텍처를 구체화하여 소프트웨어 단위를 명세하고 개발하는 것으로 다음과 같은 활동을 수행한다.

- SWE.3.BP1: 소프트웨어 상세 설계를 개발한다.
- SWE.3.BP2: 소프트웨어 단위의 인터페이스를 정의한다.
- SWE.3.BP3: 동적 행태를 서술한다.

- SWE.3.BP4: 소프트웨어 상세 설계를 평가한다.
- SWE.3.BP5: 양방향 추적성을 수립한다.
- SWE.3.BP6: 일관성을 보장한다.
- SWE.3.BP7: 합의된 소프트웨어 상세 설계를 의사소통한다.
- SWE.3.BP8: 소프트웨어 단위를 개발한다

이 중에서 BP1~BP4 의 소프트웨어 상세 설계, 단위 인터페이스 정의, 동적 형태 서술 시에 ISO 26262 에서 요구하는 소프트웨어 단위 설계 및 구현의 설계 표기법인 비정형 표기법, 준정형 표기법을 적용할 수 있다. 또한, 서브 프로그램과 함수에서 하나의 진입점과 하나의 종료점을 가져야 하는 것처럼 단위 설계 원리를 적용하여 신뢰성 높은 설계를 개발하고, BP4 의 상세 설계 평가의 기준으로 활용할 수 있다. BP4 의 평가 방법으로 단위 설계 검증 방법인 워크쓰루, 인스펙션 방법을 적용할 수 있다.

1.4 SWE.4 소프트웨어 단위 검증

소프트웨어 단위 검증은 소프트웨어 단위가 상세 설계와 비기능 요구사항을 준수 여부를 검증하는 것으로 다음과 같은 활동을 수행한다.

- SWE.4.BP1: 회귀 전략을 포함한 소프트웨어 단위 검증 전략을 개발한다.
- SWE.4.BP2: 단위 검증을 위한 기준을 개발한다.
- SWE.4.BP3: 소프트웨어 단위의 정적 검증을 수행한다.
- SWE.4.BP4: 소프트웨어 단위를 시험한다.
- SWE.4.BP5: 양방향 추적성을 수립한다.
- SWE.4.BP6: 일관성을 보장한다.
- SWE.4.BP7: 결과를 요약하고 의사소통한다.

BP1 부터 BP4 에 해당하는 단위 검증 방법으로는 정적 및 동적 분석, 코드 리뷰, 단위 시험 등을 포함한다. 단위 검증 전략 및 검증 기준으로 ISO 26262 소프트웨어 개발 착수 활동의 모델링 및

코딩 지침 항목과 소프트웨어 단위 시험의 단위 시험 방법, 단위 시험 케이스 생성 방법, 단위 수준의 구조적 커버리지 방법을 적용하여 단위 검증에 공통적으로 적용할 수 있다.

1.5 SWE.5 소프트웨어 통합 및 통합 시험

소프트웨어 통합 및 통합 시험은 소프트웨어 단위를 소프트웨어 아키텍처 설계와 일치하는 아이템으로 통합하고 설계 검증 및 시험하는 것으로 다음과 같은 활동을 수행한다.

- SWE.5.BP1: 소프트웨어 통합 전략을 개발한다.
- SWE.5.BP2: 회귀 시험 전략을 포함한 소프트웨어 통합 시험 전략을 개발한다.
- SWE.5.BP3: 소프트웨어 통합 시험을 위한 명세서를 개발한다.
- SWE.5.BP4: 소프트웨어 단위와 소프트웨어 아이템을 통합한다.
- SWE.5.BP5: 시험 케이스를 선택한다.
- SWE.5.BP6: 소프트웨어 통합 시험을 수행한다.
- SWE.5.BP7: 양방향 추적성을 수립한다.
- SWE.5.BP8: 일관성을 보장한다.
- SWE.5.BP9: 결과를 요약하고 의사소통한다.

소프트웨어 통합 및 통합 시험은 ISO 26262 의 소프트웨어 통합 및 시험 활동과 유사하다. B2~B6 의 소프트웨어 통합 시험 전략 및 통합 시험에 ISO 26262 의 요구사항 기반 시험, 인터페이스 시험과 같은 소프트웨어 통합 시험 방법과 요구사항 분석, 동치 클래스 생성 및 분석과 같은 통합 시험 테스트 케이스 생성 방법을 공통적으로 적용할 수 있다.

1.6 SWE.6 소프트웨어 인정 시험

소프트웨어 인정 시험은 통합된 소프트웨어가 소프트웨어 요구사항을 준수하는지 확인하는 것으로 다음과 같은 활동을 수행한다.

- SWE.6.BP1: 회귀 시험 전략을 포함한 소프트웨어 인정 시험 전략을 개발한다.
- SWE.6.BP2: 소프트웨어 인정 시험을 위한 명세서를 개발한다.

- SWE.6.BP3: 시험 케이스를 선택한다.
- SWE.6.BP4: 통합된 소프트웨어를 시험한다.
- SWE.6.BP5: 양방향 추적성을 수립한다.
- SWE.6.BP6: 일관성을 보장한다.
- SWE.6.BP7: 결과를 요약하고 의사소통한다.

소프트웨어 인정 시험은 ISO 26262 소프트웨어 안전 요구사항 검증과 유사하다. BP1~B4 의 안전 요구사항 검증 실시를 위한 시험 환경으로 Hardware-In-the-Loop, ECU 네트워크 환경, 차량 환경을 공통적으로 적용할 수 있다.

1.7 SUP.8 형상 관리

형상관리는 작업 산출물의 무결성을 수립하고 유지하는 것으로 다음과 같은 활동을 수행한다.

- SUP.8.BP1: 형상 관리 전략을 개발한다.
- SUP.8.BP2: 형상 항목을 식별한다.
- SUP.8.BP3: 형상 관리 시스템을 수립한다.
- SUP.8.BP4: 브랜치 관리를 수립한다.
- SUP.8.BP5: 수정과 출시를 통제한다.
- SUP.8.BP6: 베이스라인을 수립한다.
- SUP.8.BP7: 형상 상태를 보고한다.
- SUP.8.BP8: 형상화된 항목에 대한 정보를 검증한다.
- SUP.8.BP9: 형상 항목과 베이스라인의 저장을 관리한다.

BP1 ~ BP9 의 수행 활동은 ISO 26262 의 형상관리를 구체화한 것으로 볼 수 있다. 이름이 서로 동일한 것에서 볼 수 있는 것처럼 ISO 26262 와 A-SPICE 의 형상관리 활동을 매우 유사하며, ISO 26262 보다 A-SPICE 형상관리 활동이 더 구체적으로 명시하고 있다. 동일한 형상관리 프로세스를 구축하여 기능안전 수명주기와 제품 개발 활동에 공통적으로 적용할 수 있다.

1.8 SUP.10 변경 요청 관리

변경 요청 관리는 변경 요청을 관리하고 추적하는 것으로 다음 활동을 수행한다.

- SUP.10.BP1: 변경 요청 관리 전략을 개발한다.
- SUP.10.BP2: 변경 요청을 식별하고 기록한다.
- SUP.10.BP3: 변경 요청의 상태를 기록한다.
- SUP.10.BP4: 변경 요청을 분석하고 평가한다.
- SUP.10.BP5: 이행 전에 변경 요청을 승인한다.
- SUP.10.BP6: 변경 요청의 이행을 검토한다.
- SUP.10.BP7: 종료까지 변경 요청을 추적한다.
- SUP.10.BP8: 양방향 추적성을 수립한다.

A-SPICE 의 변경 요청 관리는 ISO 26262 변경 요청 관리와 매우 유사하다. 변경 요청 관리를 효과적으로 적용하기 위해 상호 보완하여 적용하는 관계라고 볼 수 있다. A-SPICE 변경 요청 관리 적용 시에 ISO 26262 변경 요청 관리의 요구사항을 공통적으로 적용할 수 있다.

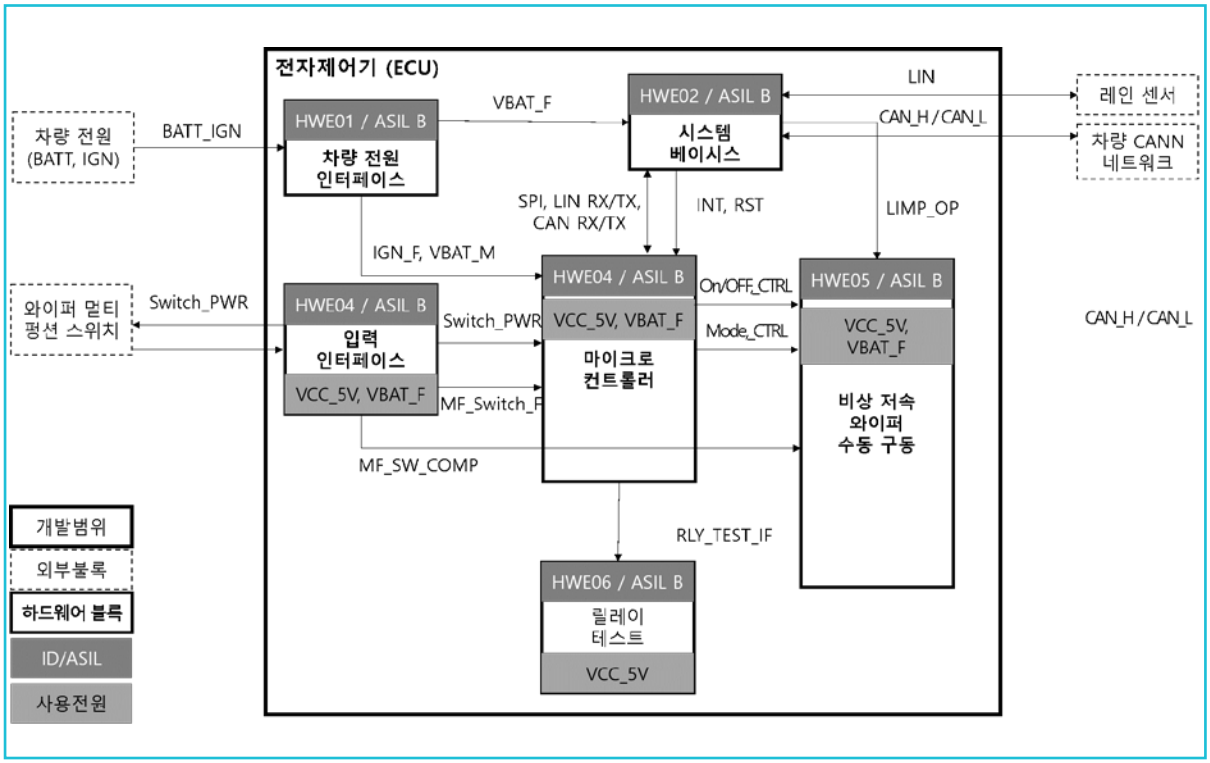
6. 적용 예제

6.1. 개발 대상 시스템

ISO 26262 파트 6 에서 소프트웨어 개발 단계는 V 모델을 기반으로 수행된다. 각 개발 단계별 활동에 대한 이해를 위해 단계별 활동을 예제를 통해서 알아보고자 한다. 개발 단계별 활동 예제 대상은 차량 전원 인터페이스이다.

차량전원 인터페이스 개발 시스템 구성은 아래 그림과 같다.

그림 43. 차량 전원 인터페이스 시스템



차량 전원 인터페이스 시스템 개발 범위에서의 차량 전원 인터페이스에 대한 속성을 정의하면 다음과 같다.

표 21. 차량 전원 인터페이스 속성

엘리먼트	ASIL	설명	입력	출력
차량 전원 인터페이스 회로	B	• HW 로 구성 • 차량 상시 전원 (Battery, BATT)에 대한 전압 안정화 • 차량 신호 (IGN)에 대한 필터링 수행 • 역극성에 의한 ECU 파손 방지 기능	• BATT (차량전원) • IGN (차량신호)	• BATT_Filtered (안정화된 BATT 전원) • IGN_Filtered (노이즈가 제거된 IGN 신호)

위의 식별된 차량 전원 인터페이스 속성에 해당하는 안전 메커니즘(safety mechanism)은 다음과 같다.

표 22. 차량 전원 인터페이스 안전 메커니즘

안전 메커니즘	검출 대상	검출 방법	결함 확정	결함 대응/경고
차량전원 모니터링 [SM01]	• BATT 과전압 • BATT 저전압	• BATT 전압 모니터링 (허용 범위 이외 검출)	• 크랭킹 이후에 BATT 과전압 또는 BATT 저전압이 250ms 이상 지속(debounce 5 회) • 진단 간격: 50ms	• CAN 경고신호 전송 (to cluster) 결함 확인 직후 CAN message 전송 완료까지의 시간 간격: 100ms

차량 전원 인터페이스 안전 메커니즘(safety mechanism) 차량전원모니터링[SM01]의 기술안전요구사항(Technical safety requirements, TSR)은 다음과 같다.

표 23. 기술안전요구사항(TSR)

분류	TSR ID	ASIL	TSR	할당

차량전원 모니터링 [SM01]	TSR_01	B	전압 모니터링을 통해 BATT 과전압 및 BATT 저전압을 검출하고 검출 시 CAN 을 통해 운전자에게 경고를 수행한다. • 고장 감지 시간 간격: 250ms(연속적) • 고장 검출 시작점: IGN on 이후, 2000ms 경과 시점 • 고장 반응 시간 간격: 100ms	마이크로 컨트롤러 (Microcontroller, MCU) • ADC 측정 • 입력 모니터링 • CAN 공통(comm). 시스템 기준 차량 전원 인터페이스 회로

SW 안전요구사항(Software Safety Requirement, SSR)을 개발하기 위해서는 기술안전개념(Technical Safety Concept, TSC) 분석을 통해 기술안전요구사항(TSR)의 구현을 위해 필요한 SW 기능의 식별이 요구된다.

6.2. SW 안전 요구사항

표 24. SW 안전 요구사항 템플릿

1. 개요

- 문서의 전반적인 소개와 내용을 기술한다.

1.1. 목적

- 문서의 목적에 대해 기술한다.

1.2. 용어 및 약어 정의

- 문서에 사용되는 용어 및 약어에 대해 기술한다.

1.3. 참조 문서

- 참조되는 문서에 대해 기술한다.

1.3.1. 참조 표준

- ISO 26262, 소프트웨어 코딩 규약(예: MISRA C:2004 등)
- 고객사의 소프트웨어 컴포넌트(및 라이브러리) 또는 기타 표준 규약에 따르는 소프트웨어 컴포넌트와의 통합을 위한 인터페이스 표준

문서 제목	문서번호	발행일	발행처

1.4. 외부 인터페이스

- 시스템 설계단계에서 정의된 system interface, user interface 및 hardware interface 를 구현하기 위한 소프트웨어 요구사항을 작성한다.
- 다음과 같은 인터페이스를 고려한다.
  : In-board communication interface (MCU ↔ system elements)
  : ECU connector ↔ Software input variables(세부사항은 HW-SW 인터페이스 명세에 의해 구체화됨)
- 각 인터페이스별로 다음과 같은 항목을 고려해서 작성한다.
  Interface signal 의 range(min, max), 단위(unit), scaling, Interface function(API), Hardware interface type(예: SPI, UART 등)

1.5. SW 설계 제약

- SW 개발에 제약조건으로 작용할 수 있는 항목을 기술한다.
- 다음과 같은 항목이 제약조건이 될 수 있다.
  : SW 개발 환경 및 개발 방법(예: 모델기반 개발, SW 도구 적용 기준 등)
  : HW 의 기술적 특징에 의존하는 소프트웨어 개발(예: 입력신호 A 와 B 는 동일 ADC 에 기반해서 Mux 를 통해 입력되어야 함)
  : Cots 또는 Legacy SW 의 사용으로 인한 Application SW 개발에 대한 제약

: SW interrupt service routine 에 대한 제약 사항(예: nesting 의 금지 등)
: 메모리 관련 제약 사항(예: heap stack 의 최소 마진 등)

2. SW 기능 요구사항

2.1. SW 구조 개요

- SW 요구사항에 대한 충분한 이해를 돕기 위한 SW 의 구조적 특징
- 다음과 같은 사항이 기술될 수 있다.
  : SW 의 간략 구조(정적 구조 또는 기능 구조)
  : SW 의 내부 element 간의 inter-connection
  : SW platform 의 적용 컨셉 등

2.2. SW 기능 상세(안전 측면)

- System requirements specification 에 정의된 시스템 수준의 기능을 구현하기 위해 SW 가 제공해야하는 각 기능에 대한 요구사항을 기술한다.
- 다음 항목이 기술되어야 한다.
  : 각 SW 기능에서 요구되는 기능(functionality)
  : 각 SW 기능에 요구되는 성능(예: signal resolution, algorithm execution speed)
  : 각 SW 수행을 위해 필요로 하는 입력 데이터에 대한 요구사항(예: data source, input data name, data handling sequence, input processing timing)
  : 각 SW 기능에서 산출해야 하는 출력 데이터에 대한 요구사항
  : 각 SW 기능 수행과 관련된 시간 제약 사항(예: detection interval, execution time, reaction time)
  : 기타 요구사항(예: data/code security, data duplication, resource consumption)

분류	TSR ID	ASIL	TSR	할당

2.2.1. SW 기능

- SW 각각의 기능에 대한 상세 요구사항을 작성한다.

Requirement ID	
ASIL	

Upper Layer Reference	**Technical Safety Requirements ID / ASIL**	
	System Level Design Reference	
Requirement		
Allocation Target		
Verification Methods		

2.3. Task 간 인터페이스 요구사항(프로세스)

- SW 가 task 또는 process 로 구성되는 경우, 각 task 또는 process 간의 상호작용에 대한 요구사항을 기술한다(필요시 작성)

Requirement ID		
Upper Layer Reference	**System Level Requirements ID**	
	System Level Design Reference	
Requirement		
Requirement Status		
Allocation Target		
Verification Methods		

3. SW 안전 요구사항

- 기술안전요구사항(TSR)으로부터 도출된 SW 안전 요구사항(SSR)을 기술한다. (ASLI, safety state, FTTI 에 대한 일치성의 유지 필요)

Requirement ID		
ASIL		
Upper Layer Reference	**Technical Safety Requirements ID / ASIL**	
	System Level Design Reference	
Requirement		
Allocation Target		
Verification Methods		

4. SW 구성/보정 요구사항

- SW 구성/보정과 관련된 요구사항을 기술한다.
  : Calibration parameter name / parameter address / data range / 각 parameter 및 value 에 대한 설명
  : Calibration parameter 적용 방법(예: flashing method)에 대한 요구사항

Requirement ID		
Upper Layer Reference	**System Level Requirements ID**	
	System Level Design Reference	
Requirement		
Requirement Status		
Allocation Target		
Verification Methods		

5. 시험 요구사항

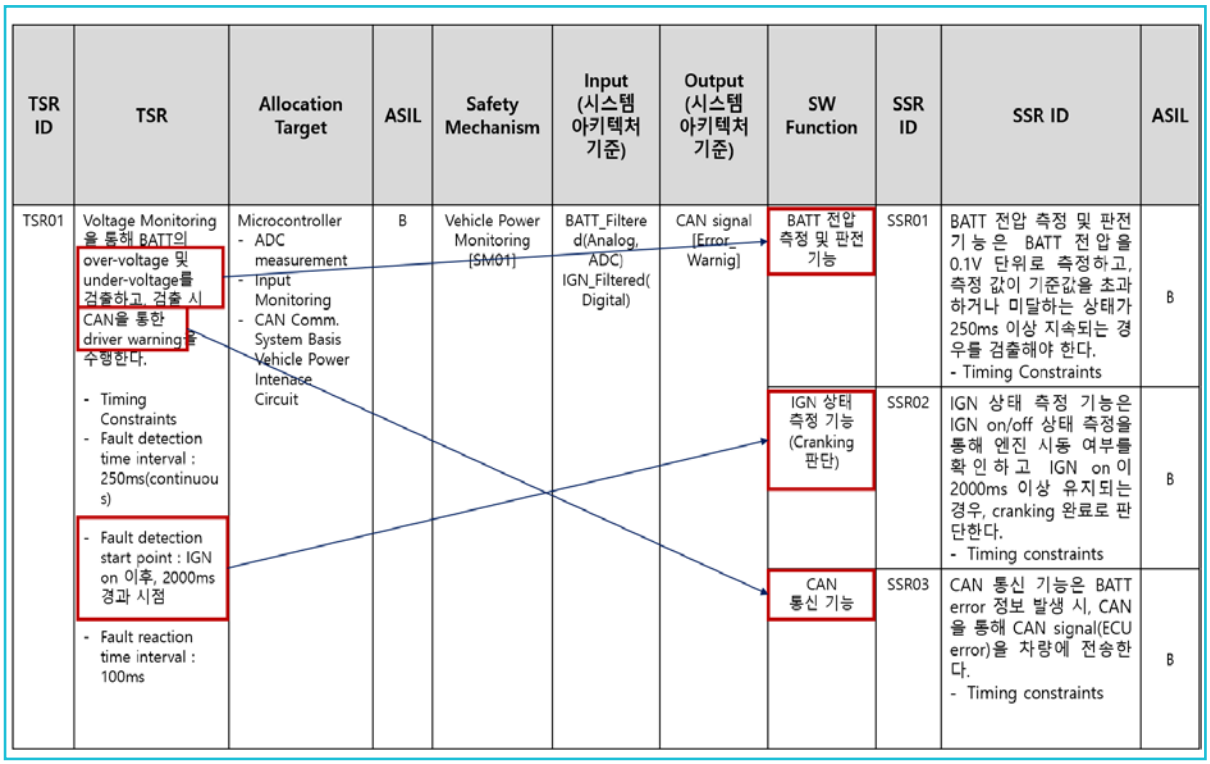
- SW 시험에 대한 요구사항을 기술한다.
- 다음의 항목을 고려 할 수 있다.
  : Static analysis criteria(예: coding rule, complexity metrics, dangerous data analysis 등)
  : SW code 에 대한 목표 test coverage(예: branch coverage, MC/DC 등)
  : SW 시험 환경에 대한 요구사항(예: target environments 시험, host 환경 시험 등)

Requirement ID		
Upper Layer Reference	**System Level Requirements ID**	
	System Level Design Reference	
Requirement		
Requirement Status		
Allocation Target		
Verification Methods		

가. SW 기능 도출

아래 그림은 상위 요구사항인 차량전원 인터페이스에서 설계된 차량전원 모니터링 기능에 대한 시스템 수준의 요구사항인 기술안전요구사항 TSR_01 의 분석을 통해 소프트웨어 수준에서 필요한 기능을 분석하고 SW 안전요구사항(SSR)을 도출하는 과정이다.

그림 44. 소프트웨어 개발 수명 주기 – 소프트웨어 안전 요구사항 검증



TSR ID	TSR	Allocation Target	ASIL	Safety Mechanism	Input (시스템 아키텍처 기준)	Output (시스템 아키텍처 기준)	SW Function	SSR ID	SSR ID	ASIL
TSR01	Voltage Monitoring 을 통해 BATT의 over-voltage 및 under-voltage를 검출하고, 검출 시 CAN을 통한 driver warning을 수행한다. - Timing Constraints - Fault detection time interval : 250ms(continuous) - Fault detection start point : IGN on 이후, 2000ms 경과 시점 - Fault reaction time interval : 100ms	Microcontroller - ADC measurement - Input Monitoring - CAN Comm. System Basis Vehicle Power Intenase Circuit	B	Vehicle Power Monitoring [SM01]	BATT_Filtered(Analog, ADC) IGN_Filtered(Digital)	CAN signal [Error_Warnig]	BATT 전압 측정 및 판전 기능	SSR01	BATT 전압 측정 및 판전 기능은 BATT 전압을 0.1V 단위로 측정하고, 측정 값이 기준값을 초과하거나 미달하는 상태가 250ms 이상 지속되는 경우를 검출해야 한다. - Timing Constraints	B
							IGN 상태 측정 기능 (Cranking 판단)	SSR02	IGN 상태 측정 기능은 IGN on/off 상태 측정을 통해 엔진 시동 여부를 확인하고 IGN on이 2000ms 이상 유지되는 경우, cranking 완료로 판단한다. - Timing constraints	B
							CAN 통신 기능	SSR03	CAN 통신 기능은 BATT error 정보 발생 시, CAN을 통해 CAN signal(ECU error)을 차량에 전송한다. - Timing constraints	B

TSR_01 중에서 SW 기능으로 구현해야 하는 기능은 다음과 같다.

- 과전압 및 저전압을 검출
  - BATT 전압측정 및 판정 기능
- CAN 을 통한 운전자에게 경고
  - IGN 상태측정 기능
- 고장 검출 시작점
  - IGN on 이후 2000ms 경과 시점에 CAN 통신 기능

도출된 SW 기능에 대한 SW 안전요구사항(SSR)은 다음과 같이 정의되었다.

표 25. 소프트웨어 기능의 식별 및 안전요구사항(SSR) 도출 결과

번호	TSR 기반의 SW 기능 도출 근거	도출된 SW 기능	설 명
1	과전압 및 저전압검출	BATT 전압 측정 및 판정 기능	BATT 전압 측정 및 판정 기능은 BATT 전압을 0.1V 단위로 측정하고, 측정 값이 기준 값 (16.0V, 7.0V)을 초과하거나 미달하는 상태가 250ms 이상 지속되는 경우를 검출해야 한다. - 시간제약: 진단 주기: 50ms 이하
2	CAN 을 통한 운전자에게 경고	CAN 통신 기능	CAN 통신 기능은 BATT error 정보 발생 시 CAN 을 통해 CAN 신호(ECU error)를 차량에 전송한다. - 시간제약: 100ms 이내 CAN 메시지 전송
3	고장 검출 시작점: IGN on 이후, 2000ms 경과 시점	IGN 상태 측정 기능	IGN 상태 측정 기능은 IGN on/off 상태 측정을 통해 엔진 시동 여부를 확인하고, IGN on 이 2000ms 이상 유지되는 경우 크래킹 완료로 판단한다. - 시간제약: IGN on 최초 발생 시점부터, 50ms 간격으로 IGN on/off 검출 수행

나. SW 안전요구사항(SSR) 명세

SW 안전요구사항(SSR)의 명세시에는 기술안전개념(TSC)과 안전요구사항과 SW 컴포넌트 간의 명시적인 인과관계가 표현되도록 작성한다. 아래 표는 작성방법을 설명하고 있다.

표 26. SSR_01

Requirement ID		예: SSR_01 • Software Safety Requirement 에 01 ID 를 가지는 기능이라는 뜻이다. 예와 같이 작성하거나 요구사항 식별이 가능한 ID 를 작성한다.
ASIL		예: ASIL B • 해당 요구사항이 ASIL 에 대해 기술한다.
Upper Layer Reference	**Technical Safety Requirements ID / ASIL**	TSR_01 / ASIL B • TSR 로부터 도출된 SSR 이기 때문에 도출된 TSR 의 ID 를 기술한다.
	System Level Design Reference	예: Vehicle Power Monitoring [SM01] • 시스템 레벨에서 설계시 참조되는 대상에 대해 기술한다.
Requirement		예: BATT 전압 측정 및 판정 기능은 BATT 전압을 0.1V 단위로 측정하고, 측정 값이 기준 값 (16.0V, 7.0V)을 초과하거나 미달하는 상태가 250ms 이상 지속되는 경우를 검출해야 한다. - Timing constraints -- 진단 주기: 50ms 이하 • 위와 같이 TSR 에서 도출된 SSR 의 내용을 기술한다.
Allocation Target		예: INP_Process_Logic, BATT_Monitoring • 적용 대상에 대해 식별하여 기술한다.
Verification Methods		예: Testing by fault injection • 검증 방법에 대해 기술한다.

SW 안전요구사항 명세 예제는 아래와 같다.

표 27. SRR_02

Requirement ID		SSR_02
ASIL		ASIL B
Upper Layer Reference	**Technical Safety Requirements**	TSR_01, 05, 07, 10, 11, 12, 13, 14 / ASIL B

	ID / ASIL	
	System Level Design Reference	차량전원 모니터링 [SM01]
Requirement		IGN 상태 측정 기능은 IGN on/off 상태 측정을 통해 엔진 시동 여부를 확인하고, IGN on 이 2000ms 이상 유지되는 경우, cranking 완료로 판단한다. - Timing constraints : IGN on 최초 발생 시점부터, 50ms 간격으로 IGN on/off 검출 수행
Allocation Target		INP_Process_Logic
Verification Methods		결함 주입 시험

표 28. SSR_03

Requirement ID		SSR_03
ASIL		ASIL B
Upper Layer Reference	Technical Safety Requirements ID / ASIL	TSR_01
	System Level Design Reference	차량전원 모니터링 [SM01]
Requirement		CAN 통신 기능은 BATT error 정보 발생 시, CAN 을 통해 CAN signal (ECU error)을 차량에 전송한다. - 시간제약: 100ms 이내 CAN message 전송
Allocation Target		CAN_Manager, CAN_Driver
Verification Methods		결함 주입 시험

6.3. HW-SW 인터페이스 명세

가. HW-SW 인터페이스 명세 템플릿

표 29. HW-SW 인터페이스 명세 템플릿

1. HW 개요

- 문서의 전반적인 소개와 내용을 기술한다.

1.1. 목적

- 문서의 목적에 대해 기술한다.

1.2. 용어 및 약어 정의

- 문서에 사용되는 용어 및 약어에 대해 기술한다.

1.3. 참조 문서

- 참조되는 문서에 대해 기술한다.

1.4. 특징

- 다음 항목을 고려하여 하드웨어 장치 특징을 약술한다

  : MCU model

  : 지원되는 HW peripheral

1.5. HW 블록 다이어그램(HW Block Diagram)

- 구성도를 작성한다.

1.6. ECU 커넥터와 HW 내부 엘리먼트 간의 관계도

Connector			Interface Circuit	HW element					SW Variable Name
Pin No.	Pin Desc.	Signal Type		HW ID / Name	MCU Pin No.	Pin Name	Register Name	Direction	

2. MCU/IC 설정 정보

2.1. 부트(Boot)및 동작 모드(Operating Modes)

- 각 MCU 및 IC 의 부트 모드 및 동작 모드를 기술한다.

2.1.1. 부트 모드(Boot Mode)

Mode name	Mode description	Entrance Trigger	Exit Condition	Applicable (Y/N)

2.1.2. 동작 모드(Operating Mode)

Mode name	Purpose of mod	Mode configuration method (incl. REG, HW-jumper)	Applicable (Y/N)

2.2. 클럭 설정(Clock Configuration)

- 다음 항목을 기술한다.
  : 각 MCU 및 IC 의 동작 클럭을 선택하기 위한 HW 파라미터(예: HW 레지스터 또는 HW pin 설정, PLL 파라미터)
  : 클럭 생성 소스(clock generation source)의 정보(예: 오실레이터 클럭 속도)
  : 코어 블록 및 주변장치(peripherals)를 위한 각 MCU 또는 IC 의 이상적인 내부 클럭 속도

3. 메모리 설정 정보

Start Addr.	Size (KB)	Memory Area Name	Description / Constraints

4. 주변 장치(Peripheral) 설정 정보

4.1. MCU/IC 주변 장치 할당

MCU/IC Peripheral		Assignment	
Category	Channel	Assigned to	Purpose / Note

4.2. 주변 장치 세부사항

- 스타트-업 코드 또는 하드웨어 제어 프로그램(예: 디바이스 드라이버)의 개발을 지원하기 위해 주변 장치에 대한 설정을 기술하는 것이다.

No.	Peripheral Name	Register ID	Address	Init Value	Configuration Value	Description

5. 인터페이스 회로 설명

- 다음 항목을 기술한다.
  : HW 인터페이스 회로 식별
  : 회로의 목적(예: 신호 클램핑, ESP 보호, 노이즈 필터링, 등)
  : 소프트웨어 개발 측면에서 고려 되어야하는 모든 관련된 인터페이스 회로 특성

HW interface circuit ID (HW	Purpose of interface	Description

block ID)	circuit	

나. ECU 커넥터 인터페이스

ECU 커넥터와 HW 내부 엘리먼트 간의 관계도

Connector				HW element					
Pin no	Pin Desc	Signal Type	Interface circuit	HW ID / Name	MCU Pin No	Pin Name	Register Name	Direction	SW Variable Name
1	BATT	Analog (Power)	Yes	HWE04	23	VSUP	VSUP_READ	Input	RINP_BATT
2	IGN	Analog (Signal)	Yes	HWE04	25	GPIO3	GPIO3_REG	Input	RINP_IGN

- Pin No.: ECU 커넥터의 각 커넥터에 해당하는 식별자
- Pin description: Pin 이름
- Signal Type: Analog, PWM, CAN message, LIN message, 등.
- Interface circuit: ECU 커넥터와 MCU 또는 다른 IC 사이의 인터페이스 회로가 존재하는 경우, 인터페이스 회로의 HW 식별자를 기입한다.
- HW ID / Name: 각 HW 엘리먼트들의 식별자 및 이름을 기입한다.
- MCU Pin No.: MCU 또는 IC 의 각 pin 들의 식별자를 기입한다.
- Pin Name: MCU 또는 IC 의 각 pin name 을 기입한다.
- Register Name: MCU 또는 IC 의 pin 이 내부 HW 레지스터와 연결되는 경우, HW 레지스터의 이름을 기입하거나 해당이 없는 경우에는 "N/A"로 표시한다.
- Direction: MCU 또는 IC 의 관점에서 신호 및 메시지의 방향(In / Out / Bi-directional)을 표시한다.
- SW variable Name: SW 변수명을 사용하여 SW 진입점(entry point)을 기술한다.

• HW 설계 명세서와 SW 설계 명세서 사이의 일관성(consistency)을 유지한다.

다. MCU/IC 설정 정보

- 부트(Boot) 및 동작 모드(Operating Modes)

각 MCU 및 IC 의 부트 모드 및 동작 모드를 기술한다. 하드웨어 설계 명세서에 존재하는 경우, 해당 내용에 대한 참조를 표시한다.

- 부트 모드(Boot Mode)

Mode name	Mode description	Entrance Trigger	Exit Condition	Applicable (Y/N)
Single chip Mode	flash memory 로 부팅됨	FAB, ABS pin 설정을 따름	N/A	Y
Serial Boot	serial device 로 부팅됨	FAB, ABS pin 설정을 따름	N/A	N

• Mode name: 각 MCU 및 IC 의 부트 모드 식별자

• Mode description: 각 부트 모드에 대한 간략한 설명

• Entrance Trigger: 부트 모드를 개시하는 트리거 소스

• Exit condition: 각 부트 모드에 대한 종료 조건(exit conditions)을 기술한다.

• Applicable: 부트 모드가 HW 제품에 적용되는 경우 "Y" 또는 "N"를 표시한다(일반적으로 HW 제품은 MCU 또는 IC 의 모든 부트 모드를 지원하도록 설계되지 않는다)

• 자세한 정보 또는 MCU/IC 데이터시트에 설명된 스펙을 기술하는 것은 적절치 않다. 본 항목의 목적은 SW 엔지니어에게 포괄적인 이해를 제공하는 것이다. 필요한 경우, HW 데이터시트 문서 또는 HW 설계 명세서에 대한 참조를 표시한다.

- 동작모드(Operating mode)

Mode name	Purpose of mode	Mode configuration method (incl. REG, HW-jumper	Applicable (Y/N)
Default	Normal Operation	Power on 리셋 이후, 기본 동작 모드	Y
Sleep	차량 key-off 상태에서 ECU 대기 상태 유지	MCU sleep pin 에 active high 인가 및 SW 를 통한 Sleep_Activation_REG 설정 수행	Y

- Mode name: 각 MCU 및 IC 의 동작 모드 식별자
- Purpose of mode: 각 동작 모드의 목적에 대한 간략한 설명
- Mode configuration method: 각 동작 모드에 대한 트리거링(Triggering) 또는 진입 조건을 기술한다.
- Applicable: 동작 모드가 HW 제품에 적용되는 경우 "Y" 또는 "N"를 표시한다(일반적으로, HW 제품은 MCU 또는 IC 의 모든 동작 모드를 지원하도록 설계되지 않는다)
- 자세한 정보 또는 MCU/IC 데이터시트에 설명된 스펙을 기술하는 것은 적절치 않다. 본 항목의 목적은 SW 엔지니어에게 포괄적인 이해를 제공하는 것이다. 필요한 경우, HW 데이터시트 문서 또는 HW 설계 명세서에 대한 참조를 표시한다.

- 클럭 설정(Clock Configuration)

다음 항목을 기술한다.

- 각 MCU 및 IC 의 동작 클럭을 선택하기 위한 HW 파라미터
  예: HW 레지스터 또는 HW pin 설정, PLL 파라미터
- 클럭 생성 소스(clock generation source)의 정보
  예: 오실레이터 클럭 속도(Oscillator clock speed)
- 코어 블록 및 주변장치(peripherals)를 위한 각 MCU 또는 IC 의 이상적인 내부 클럭 속도
- 자세한 정보 또는 MCU/IC 데이터시트에 설명된 스펙을 기술하는 것은 적절치 않다. 본 항목의 목적은 SW 엔지니어에게 포괄적인 이해를 제공하는 것이다. 필요한 경우, HW 데이터시트 문서 또는 HW 설계 명세서에 대한 참조를 표시한다

라. 메모리 설정 정보

Start Addr.	Size (KB)	Memory Area Name	Description / Constraints
0x0000_0000	Xx	FLASH0	NOR FLASH
0x1000_0000	Xx	FLASH1	NAND FLASH
0x4000_0000	Xx	SRAM0	SRAM (66MHz)
0xFFFF_0000	Xx	Config. Register	

HW 설계 관점에서 각 메모리 장치의 가용 메모리 공간 및 특성을 기술한다. SW 메모리 영역에 대한 정의(예: 코드 영역, 데이터 영역 등) 기술하는 것은 적절치 않다. 본 정보의 목적은 SW 엔지니어에게 메모리 맵(memory map)을 제공하는 것이다. 필요한 경우, HW 데이터시트 문서 또는 HW 설계 명세서에 대한 참조를 표시한다.

마. 주변장치(Peripheral) 설정 정보

MCU/IC Peripheral		Assignment	
Category	Channel	Assigned to	Purpose / Note
Interrupt	0	Driver switch 1	ECO mode detection
	1		
	2		
	3	Reserved	Timer 1 과 공유
Timer	0	Reserved	
	1	Reserved	
	2		
	3	OS core time tick	OS clock tick generation
Watchdog Timer			
DMA	0		
	1		
FlexRAY	-		

CAN	0	P-CAN	P-CAN networking
	1	B-CAN	B-CAN networking
LIN	-		
ADC	0		
	1		
Sine Wave Generator	0		
SPI	0	Serial FLASH ROM	
	1	DC-DC converter	

- Category: 다음의 카테고리들을 고려하여 주변장치들을 분류한다: 인터럽트 / 타이머 카운터 / 외부 네트워크를 위한 주변 장치들 / in-board 네트워크를 위한 주변 장치들 / ADC, 등.
- Channel: 주변 장치는 몇 개의 채널로 구성될 수 있다.
- Assigned to: MCU 또는 IC 의 주변 장치는 내부 SW(예: HW 타이머 3 와 OS 코어 타임 틱(time tick)) 또는 외부 HW 엘리먼트(예: ADC0 와 Accel. position sensor)에 할당될 수 있다.
- Purpose: 주변 장치들의 목적
- MCU/IC 데이터시트에 기술된 정보 또는 스펙의 세부사항을 기술하는 것은 적절치 않다. 필요한 경우, HW 설계 명세서 또는 SW 설계 명세서에 대한 참조를 표시한다.
- HW 적으로 공유되는 Peripheral resource 인지 여부가 표기되어야 한다.

- 주변 장치 세부사항

본 항목의 목적은 스타트-업 코드 또는 하드웨어 제어 프로그램(예: 디바이스 드라이버)의 개발을 지원하기 위해 주변 장치에 대한 설정을 기술하는 것이다. 모든 주변 장치에 대한 모든 파라미터들을 기술할 필요는 없으나, 각 ECU 및 IC 의 올바른 동작을 보증하는 초기 값(init values)은 기술되어야 한다. 필요한 경우, HW 데이터시트 문서 또는 HW 설계 명세서에 대한 참조를 표시한다.

No.	Peripheral Name	Register ID	Address	Init Value	Configuration Value	Description (Meaning)

				(After Reset)		
1	Interrupt	ISR	B+0x010	0x0000	R only	-
		CEPCFR0	B+0x014	0x0000	-	-
		IMR	B+0x020	0x0000	0x0001	Mask - ECH interrupt must be disabled to assure execution of boot sequence

MCU 또는 IC 의 올바른 동작을 보증하기 위해 필요한(설정되어야 하는) HW 레지스터 설정 정보만 기술한다.

6.4. SW 아키텍처 설계

가. SW 아키텍처 설계 명세서 템플릿

표 30. SW 아키텍처 설계 명세서 템플릿

가. 개요

- 문서의 전반적인 소개와 내용을 기술한다.

1.1. 목적

- 문서의 목적에 대해 기술한다.

1.2. 용어 및 약어 정의

- 문서에 사용되는 용어 및 약어에 대해 기술한다.

1.3. 참조 문서

- 참조되는 문서에 대해 기술한다.

1.4. 구조

- SW component list 에 정의된 전체 software component 들이 표기되어야 함
- UML class diagram 의 사용을 권장함
- Middle ware, device driver 와 같은 hardware dependent software component 도 같이 표기돼야 함
- 기능안전 관련 software component 의 경우 ASIL 이 표기되어야 함

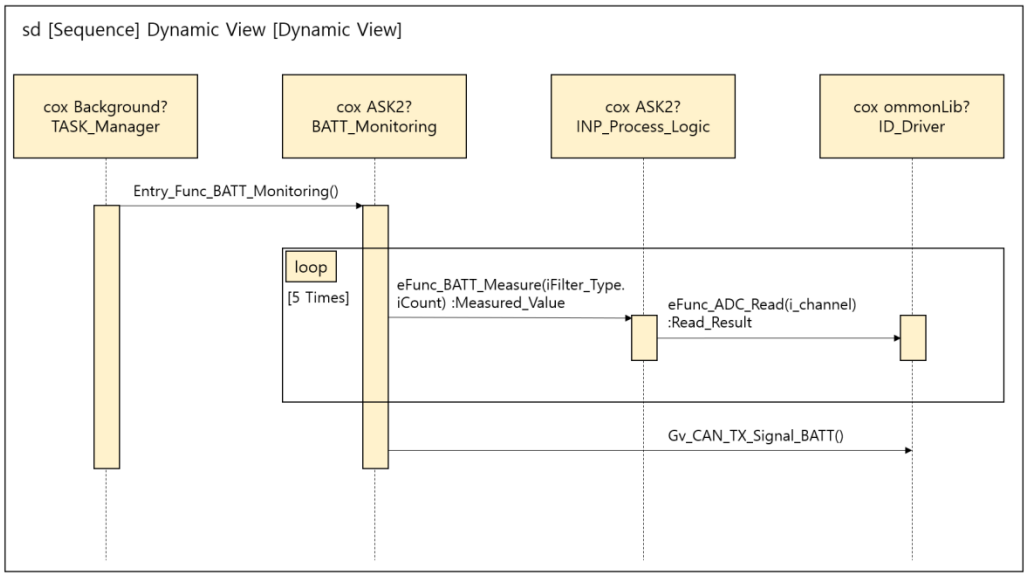
1.5. SW components list

Component ID	Component Name	Component Type	ASIL

1.6. 동적 설계

1.6.1. 시퀀스 다이어그램

SW Requirements IDs (SSR ID 포함)	
Note	



1.7. 리소스 사용률

- 각 SW component 별 요구되는 HW 자원을 표기

Component ID	**Component Name**	**Required Resources**	**SW REQ. ID (Incl. SSR/ASIL)**

나. SW Task Structure

- SW task(process)간의 interaction/dependency 를 정의
- 다음 항목을 정의한다.
  : 전체 SW task list
  : 각 task 에 대한 설계 사양 및 task 간의 communication 사양

2.1. Task List

Task ID	**Priority**	**Execution Time**	**Execution**	**Member of Task**

			Interval	

2.2. Task 명세

- Task switching 조건(예: allocated time slot expired, 특정 interrupt 발생 등)
- 스케쥴링 전략(예: Round Robin, First come first served 등)
- Inter-task communication 방법(예: flag, message box, semaphore 등)
- Task 간 자원 공유
- SW partitioning 특성(예: memory protection 전략 등)

SW Requirements IDs (Incl. SSR ID)	
Note	

다. SW Components 상세

- 각 SW component 의 인터페이스 및 기능을 정의
- 다음과 같은 항목을 정의할 수 있음
  : 해당 SW component 의 기능
  : Input/output data
  : 해당 SW component 를 구성하는 SW unit 항목
  : 해당 SW component 의 control/data dependency

3.1. SW Component 이름: [component 이름]

3.1.1. 기능

Component Name		Component ID	
ASIL		Related SW Requirements	
Purpose of the component			
Functionality			

3.1.2. Input Data

Input Data Name	Data Type	Value Range / Unit	Input Type	Description

3.1.3. Output Data

Output Data	Data Type	Value	Output Type	Description

Name		Range / Unit		

3.1.4. Internal Software Units List

No	Unit Name (Function Prototype)	Function description

라. 기타

- 상기 이외에 필요한 내용 명시(예: memory map 등)

나. SW 아키텍처 세부항목

표 31. SW 아키텍처 세부 항목

분류.	항목	설명
Static Design Aspect	Software structure including its hierarchical levels	SW 정적 아키텍처를 통해 표현됨
	The logical sequence of data processing	SW 정적 아키텍처 및 sequence diagram 에 의해 표현됨
	The data types and their characteristics	개별 SW 컴포넌트 사양 및 전역변수 사양을 통해 정의됨
	External interfaces of the software components	개별 SW 컴포넌트 사양을 통해 정의됨
	The external interfaces of the software	Hardware software interface 사양을 통해 정의됨
	The constraints including the scope of the architecture and external dependencies	-
Dynamic Design Aspect	Functionality and behavior	상위 수준 SW 기능 및 하위 수준 SW 기능에 의해 정의됨
	The control flow and concurrency of processes	Sequence diagram 및 Task 구조 사양을 통해 정의됨

	The data flow between the software components	SW 정적 아키텍처, sequence diagram 을 통해 표현됨
	The data flow at external interfaces	System design specification 수준에서 표현됨 (ISO 26262 – 6, 7.2 General 의 Note 참조)
	Temporal constraints	-

다. 소프트웨어 컴포넌트 리스트 작성

"하위 수준 SW 기능"을 기반으로 SW 컴포넌트 리스트를 정의한다. 이를 정의하는 목적은 하위 수준 SW 기능과 SW 컴포넌트는 서로 1:1 의 관계가 아닐 수 있기 때문이다. 따라서, 개별 SW 컴포넌트 관점에서 SW 컴포넌트 내부에 구현되어야 하는 기능 및 입/출력 요소를 정의함으로써 하위 수준 SW 기능과 SW 컴포넌트 간의 정합성을 달성할 수 있다.

그림 45. 하위 수준 SW 기능과 SW 컴포넌트 간의 관계

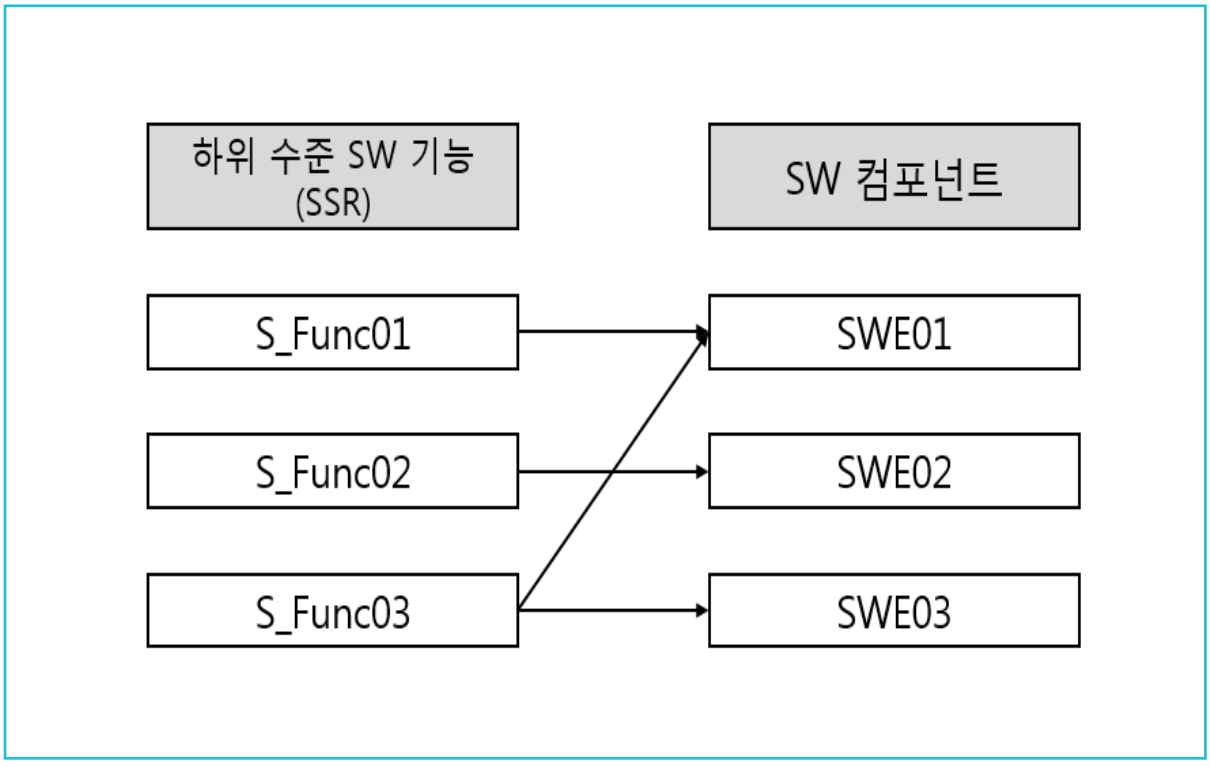


표 32. 소프트웨어 컴포넌트 리스트

Component ID	Component Name	Component Type	ASIL
SWE02	BATT_Monitoring	Application level component / New development	B
SWE04	CAN_Manager	Application level component / New development	B
SWE03	CAN_Driver	Application level component / New development	B

- Component ID/Component Name: SW 컴포넌트의 식별자 및 이름
- Component Type: New development, Legacy or COTs. Application level component or Basis level component(예: AUTOSR BSW, device driver, middle ware 등)
- ASIL: SW 컴포넌트의 ASIL 등급(해당 SW 컴포넌트에 할당된 software safety requirement 의 ASIL 등급 및 coexistence criteria 에 따라 결정된다.)

라. 정적 아키텍처 설계

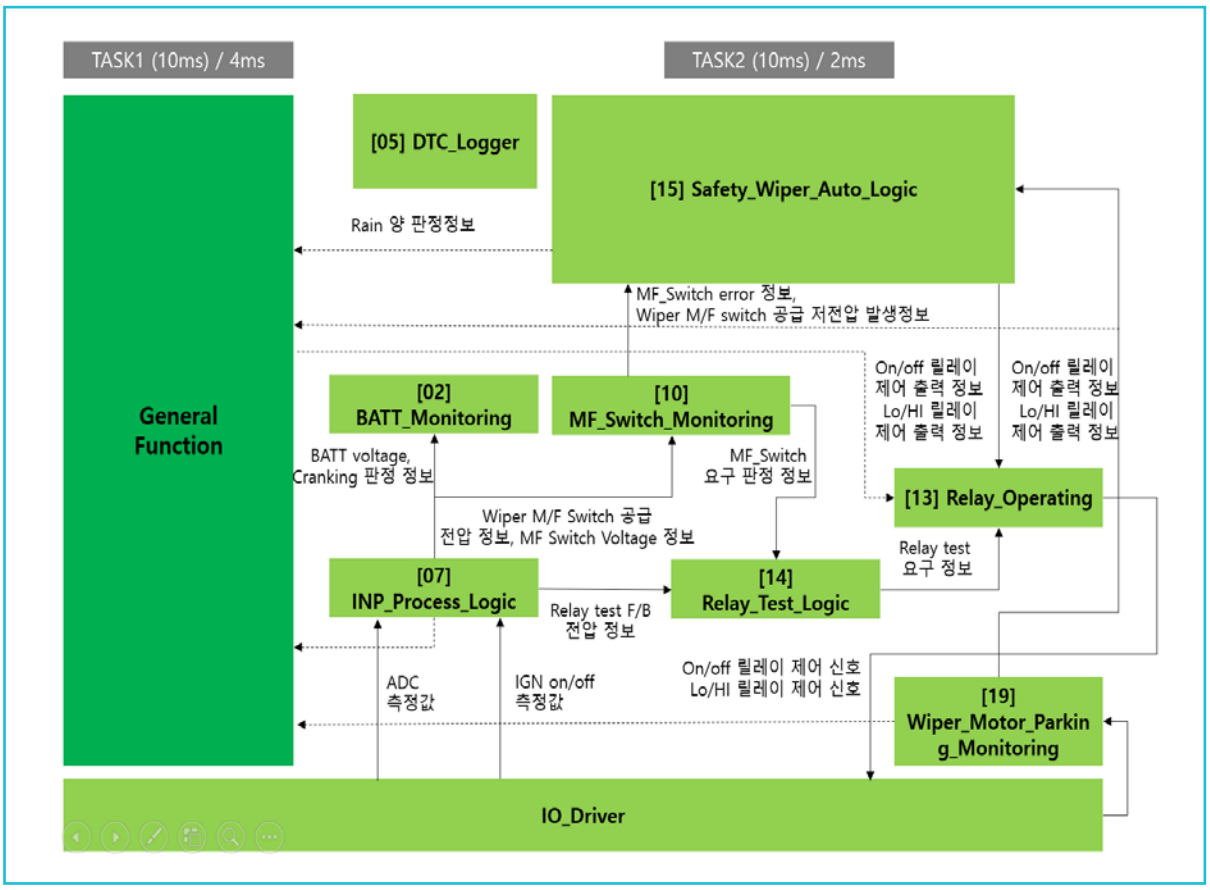
SW 정적 아키텍처는 SW 컴포넌트 리스트를 이용해 구성될 수 있다. SW 컴포넌트 리스트는 기본적으로 컴포넌트의 기능, 해당 기능을 수행하기 위한 입력 정보, 해당 기능 수행을 통한 출력 정보로 구성된다. 이 입/출력 관계를 연결함으로써 기본적인 SW 정적 아키텍처의 초안을 정의할 수 있다. 이후에 SW 기능 구현 관점의 구현 가능성, 효율성 및 기능안전에서 요구하는 아키텍처 준수 조건(예: independence, freedom form interference)및 시간 제약 사항(예: fault detection time interval, fault reaction time interval)에 대한 적합성 분석을 통해 SW 정적 아키텍처 수정 및 확정될 수 있다.

SW 정적 아키텍처는 기능안전 설계 과정에서 수행되는 안전 분석에 대한 중요한 입력 정보로 작용한다. SW 정적 아키텍처의 표현 요소는 다음과 같다.

표 33. SW 정적 아키텍처의 표현 요소

항목	설명	비고
SW 컴포넌트	SW 정적 아키텍처를 구성하는 SW 컴포넌트	단, 안전 메커니즘을 구성하는 SW 컴포넌트를 중심으로 표현되었으며, 안전관련 의도 기능 및 비 안전관련 의도 기능은 General Function 이라는 이름으로 간략화 되어 있음.
	Task 정보	각 SW 컴포넌트가 어떤 Task 의 구성요소인지 표현됨.
SW 컴포넌트 간의 상호 작용	Data flow	SW 변수 수준의 data flow 는 SW 세부 설계를 통해 사양화 될 수 있으나, 안전 분석 수행을 위해서는 정보 수준의 data flow 로 충분 할 수 있음.
	Control flow SW 컴포넌트 간의 호출 관계	SW 컴포넌트 내부 함수 간의 호출 관계는 표기되지 않음.
	Task 간 정보 교환	Task 간의 전역 정보 흐름을 표기
동작 방식	동작 정보	Sequence diagram 을 통해 표현될 수 있음.

그림 46. SW 정적 아키텍처 예시



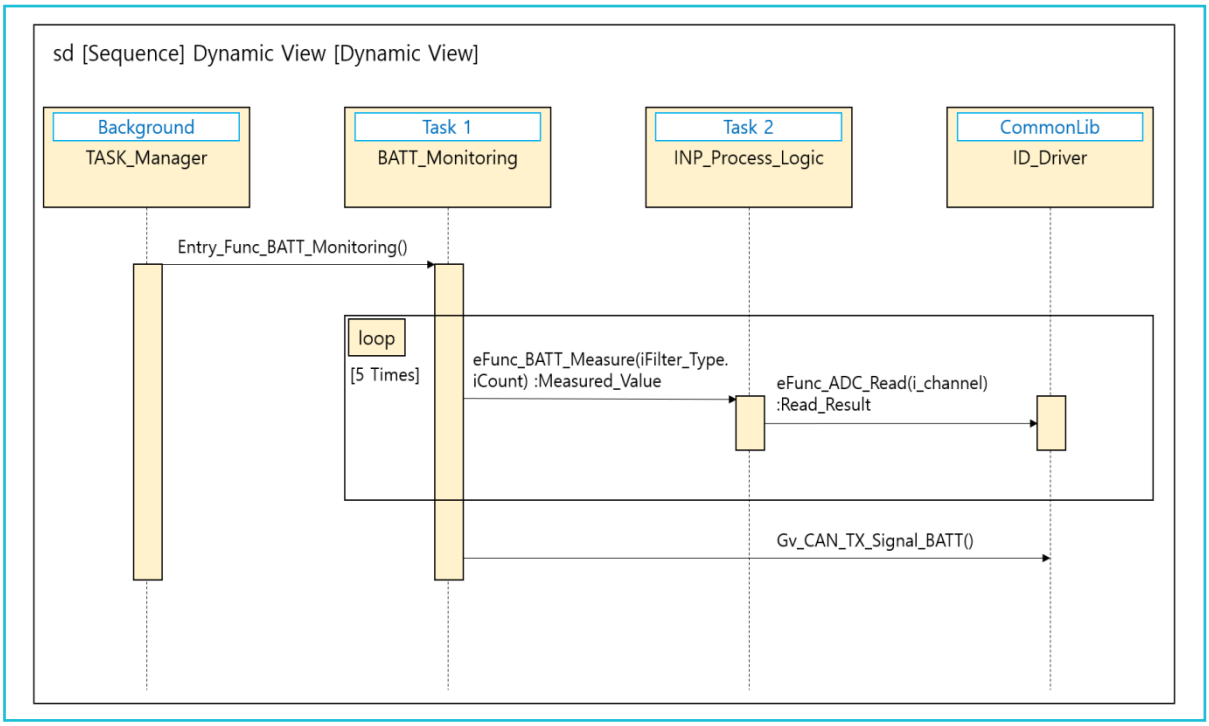
소프트웨어 정적 아키텍처 작성 시 주의사항은 다음과 같다.

- 소프트웨어 컴포넌트 리스트에 정의된 전체 소프트웨어 컴포넌트들이 표기되어야 한다.
- UML 클래스 다이어그램(class diagram)의 사용을 권장한다.
- Middle ware, device driver 와 같은 하드웨어 관련 컴포넌트들이 같이 표기되어야 한다.
- ASIL 이 할당되는 소프트웨어 컴포넌트의 경우 ASIL 이 표기되어야 한다.

마. 동적 설계

다음 예제는 UML 의 Sequence diagram 에 정의된 표기법 또는 동등 수준의 정보를 표현할 수 있는 표기법을 적용하였다.

그림 47. Sequence diagram 작성 예시



바. 소프트웨어 Task Structure 정의

Task 의 구조는 다음과 같은 정보를 포함해야 한다.

- Task 식별자
- 각 task 의 실행 간격(time interval)
- 각 task 의 수행 시간(execution time duration)
- Task 의 수행 순서 및 우선순위(특히, 선점형 스케줄링의 경우)
- 각 task 의 구성 멤버(SW 컴포넌트)
- 각 Task 내부 별 SW 컴포넌트의 수행 순서

Task ID	Priority	Execution Time	Execution Interval	Member of Task
Task1	3	4ms	20ms	General Function

Task2	1	4ms	20ms	Safety Task – BATT monitoring
Task3	2	2ms	10ms	Safety Task - Communication

- SW 구현 방법에 따라 interrupt service routine 도 일종의 task 로 간주될 수 있음
- OS 또는 scheduler 가 없거나 또는 전체 소프트웨어가 1 개의 task 로 구성되는 경우에도 작성이 필요함

Sequence diagram 은 다음 항목에 대해서 표기되어야 한다.

- 상위 수준 SW 기능 또는 하위 수준 SW 기능 별 SW 의 동작(SW 의 복잡도가 높은 경우, 하위 수준 SW 기능별로 표기하는 것이 바람직함)
- 기술안전개념(TSC)에 정의된 안전메커니즘 별 SW 의 동작

그림 48. Task 구조 작성 예시

Task ID	Time Interval	Max. Time Duration	Description	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Task1	20ms	4ms	General Function	Max. 4ms										Max. 4ms									
Task2	20ms	4ms	Safety Task - Wiper Control					Max. 2ms										Max. 2ms					
Task3	10ms	2ms	Safety Task - Communication							Max. 2ms										Max. 2ms			
Task4	40ms	2ms	Safety Task - Monitoring									Max. 2ms										Max. 2ms	

소프트웨어 정적 아키텍처 작성 시 주의사항은 다음과 같다.

- SW 구현 방법에 따라 interrupt service routine 도 일종의 task 로 간주될 수 있다.
- OS 또는 scheduler 가 없거나 또는 전체 소프트웨어가 1 개의 task 로 구성되는 경우에도 작성이 필요하다.

사. 전역변수(Global Variable)정의

기능안전 관점에서 전역 변수는 SW 컴포넌트 간의 interference 를 일으킬 수 있는 주요 요소에 해당한다. 각 전역변수에 값을 쓰는 SW 컴포턴트와 값을 읽는 SW 컴포넌트를 정의함으로써 freedom from interference 및 independence 를 저해하는 원인을 식별할 수 있다.

표 34. 전역 변수 사양 작성 양식

번호	전역 변수 이름	Data Type	분류	값	Read	Write	SR
-	전역 변수의 이름	전역 변수의 Data type	SW 정적 아키텍처와의 관계성	해당 전역 변수가 가질 수 있는 값의 유효 범위 및 개별 값의 의미	해당 전역 변수의 값을 읽는 SW 컴포넌트의 식별자	해당 전역 변수에 값을 쓰는 SW 컴포넌트의 식별자	안전 관련성 여부

모든 전역 변수는 SW 정적 구조에 표기된 data 정보 중 1 개 이상의 정보와 mapping 관계를 가져야한다. 그렇지 않은 경우 SW 정적 구조가 실제 구현 정보를 모두 표현하지 않았다는 의미가 될 수 있으며 안전분석 수행 시 systematic fault 의 유발 요소로 작용할 수 있다.

아. SW 컴포넌트 상세설계

SW 컴포넌트 상세설계시 작성 내용은 다음과 같다.

- 각 SW 컴포넌트의 인터페이스 및 기능을 정의
- 해당 SW 컴포넌트의 기능 및 input/output data 정의
- 해당 SW 컴포넌트를 구성하는 SW unit 항목 정의
- 해당 SW 컴포넌트의 control/data dependency 정의

SW 컴포넌트 상세설계 예시는 다음과 같다.

Software Component Name: [BATT_Monitoring]

- Functionality

Component Name	BATT_Monitoring	**Component ID**	SWE02
ASIL	ASIL B	**Related SW Requirements**	SSR_01
Purpose of the component	BATT 고전압, 저전압 판정		
Functionality	입력 정보 · ADC band gap 전압 측정 값 · IGN 상태 정보 (Global) 판정 로직 · BATT > 16.0 → Over-voltage · BATT < 7.0 → Under-voltage 출력 · BATT error 정보 (Global) · HW reset		

- Input Data

Input Data Name	Data Type	Value Range / Unit	Input Type	Description
BATT_VOLT_INP	U_INT_8	0x0 - 0x255 (0V - 48V)	Function Call Parameter (Entry function)	BATT voltage (필터링 된 값)
CRANK_STATUS	U_INT_8	0xFE: Off 0x7F: Cranking completed	Global variable	Cranking 완료 정보 (Global)

작성 시 주의사항은 다음과 같다.

- Functional call parameter, 관련된 global variable, 타 software component 및 task 가 송신하는 message, 특정 HW register 의 값 등이 Input data 로 간주될 수 있음.
- 필요 시, input data 의 address 를 표기할 수 있음 (특정 address 로 고정해야 하는 경우. 예: calibration data)

- Output Data

Output Data Name	Data Type	Value Range / Unit	Output Type	Description
BATT_STATUS	U_INT_8	0xF0: In range 0x0A: Over-voltage 0x0D: Under voltage	Global variable	BATT error 정보 (Global)
SET_REG_RST	U_INT_8	0xF1: HW reset	HW register	HW reset

작성 시 주의사항은 다음과 같다.

- Functional call return value, 해당 software component 가 호출하는 global variable, 타 software component 또는 task 로 전송하는 message 등은 output data 로 간주될 수 있음.
- 필요 시, input data 의 address 를 표기할 수 있음 (특정 address 로 고정해야 하는 경우)

- Internal Software Units List

No	Unit Name (Function Prototype)	Function description
1	UInt8 ADC_Config_BATT(Void)	BATT 측정을 위한 MCU register 값 설정 함수 Return value - 0xF1: configuration OK - 0x1C: configuration fail
2	UInt8 ADC_Read_BATT(Void)	MCU 의 ADC peripheral 를 통해 BATT 전압을 측정하고 측정 값에 대한 SW filtering 수행 후, 값을 반환하는 함수 Return value - 0x01 ~ 0xFE: 측정 값 (0V ~ 48V)

		- 0x00: ADC fault (short to GND) - 0xFF: ADC fault (internal failure)
3	UInt8 BATT_Volt_Check(U_INT_8 BATT_VOLT_INP)	측정된 ADC 값을 통해 over-voltage, under-voltage 를 판정하고, 판정 결과를 write 하는 함수 Return value - 0xF1: BATT ok (in-range) - 0x1C: BATT failure - over voltage - 0x1A: BATT failure - under voltage

6.5. 안전분석

가. SW 안전분석 템플릿

1. 개요

- 문서의 전반적인 소개와 내용을 기술한다.

1.1. 목적

- 문서의 목적에 대해 기술한다.

1.2. 용어 및 약어 정의

- 문서에 사용되는 용어 및 약어에 대해 기술한다.

1.3. 참조 문서

- 참조되는 문서에 대해 기술한다.

2. SW 안전분석

2.1. SW 아키텍처 수준의 안전분석

- 연역적 분석 또는 귀납적 분석 내용을 기술한다.

2.2. SW 안전분석 절차

- 소프트웨어 고장모드(failure mode) 식별
- 소프트웨어 고장모드의 잠재 원인 식별
- 소프트웨어 고장모드로 인한 잠재 영향 식별
- 안전 메커니즘의 효과성 평가

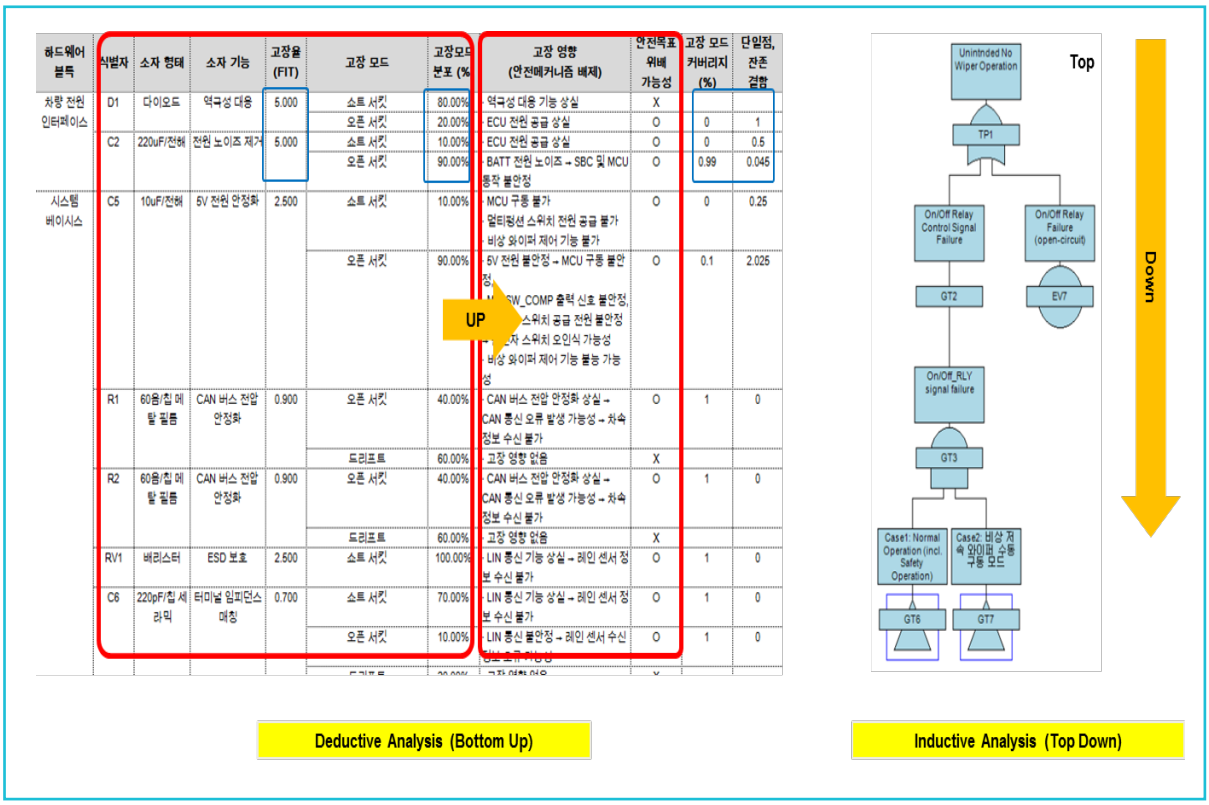
2.3. 고장모드 식별

Failure mode 식별 방법	**권장 수준**	**비고**

나. SW 아키텍처 수준의 안전분석

소프트웨어는 시스템적 고장(systematic failure)이 발생하므로 정성적 분석방법만이 적용 가능하다. 정성적 분석에는 정성적 FTA, HAZOP, 정성적 ETA, 정성적 FMEA 등이 있다. HAZOP, ETA, FMEA 등은 연역적 분석(deductive analysis) 및 bottom up 방식으로 분류되며 FTA 등은 귀납적 분석(inductive analysis) 및 top-down 방식으로 분류될 수 있다. 연역적 분석과 귀납적 분석 예시는 다음과 같다.

그림 49. 연역적 분석 및 귀납적 분석



다. SW 안전분석 수행절차

소프트웨어 수준의 안전분석의 목적 및 분석대상, 방법 등은 아래와 같다.

표 35. SW 안전분석 수행절차

목적	• Software 구성 요소 중 safety related parts 와 non-safety related parts 를 구별 • Systematic faults 및 random HW faults 에 대한 safety mechanism 의 효과성을 검증 • 기능안전 관점에서 무결성을 갖는 software design 수행을 지원한다.
분석 대상	• Software architectural design
분석방법	• 정성적인 분석 수준의 분석 수행

안전분석 수행절차는 다음과 같다.

- 소프트웨어 고장모드(failure mode) 식별
- 소프트웨어 고장모드의 잠재 원인 식별
- 소프트웨어 고장모드로 인한 잠재 영향 식별
- 안전 메커니즘의 효과성 평가

라. SW 고장모드(failure mode) 식별

SW 는 높은 수준의 복잡성을 가지며 구조 역시 다양성을 갖기 때문에 HW 와 같이 정형화된 방법을 이용하여 failure mode 를 도출하기 어려운 특성을 갖는다. 하지만 분석의 일관성 유지를 위해 체계적인(systematic manner) failure mode 도출 방법이 요구되기 때문에, 다음과 같은 방법의 적용을 권장한다.

표 36. SW 고장 모드 식별

Failure mode 식별 방법	권장 수준	비고
SHARD (Software Hazard Analysis)	Mandatory	HAZOP (Hazard and operability)의 변형으로 software 에 적용하는데 적합한 guideword 6 종을 제공한다.
SW failure modes database	Optional	산업현장에서 자주 발생하는 것으로 보고되는 SW failure mode 의 집합으로, failure mode 에 대한 추가적인 탐색이 필요한 경우 적용한다.

마. SHARD 방법 적용

다음은 SHARD guidewords 를 이용하여 특정 SW component 의 failure mode 를 식별하는 과정을 예시로 보여준다.

1) SW 컴포넌트에 대한 정의 예제

- Component Name: Actuator F/B acquisition
- Component Functionality

MCU 의 ADC 를 이용하여 Actuator 의 F/B 전압값을 측정 (측정주기 50ms)

F/B 전압 값이 3.3V 보다 큰 경우, "Mode_A"라고 판정하고, 3.3V 보다 작은 경우 "Mode_B"라고 판정

- Component 의 입력 요소
  : ADC F/B voltage
- Component 의 출력 요소
  : Actuator Mode 판정 값

2) Actuator F/B acquisition 컴포넌트에 SHARD guidewords 를 적용한 결과

표 37. Actuator F/B acquisition 컴포넌트에 SHARD guidewords 적용 결과

SW Component	SHARD Guidewords	Failure Modes
Actuator F/B acquisition	Omission	Actuator F/B 측정을 수행하지 않음 (측정값 판정 불가)
	Commission	해당사항 없음
	Early	해당사항 없음

	Late	해당사항 없음
	Coarse	F/B 전압 3.3V 이상 값을 Mode_B 라고 판정 또는 3.3V 미만 값을 Mode_A 라고 판정
	Subtle	해당사항 없음

바. SW 고장 모드 데이터베이스(SW failure mode database)

아래 표는 SW 에서 일반적으로 발생할 수 있는 고장모드(failure mode)에 대해 기술하고 있다.

표 38. SW 고장모드 데이터베이스

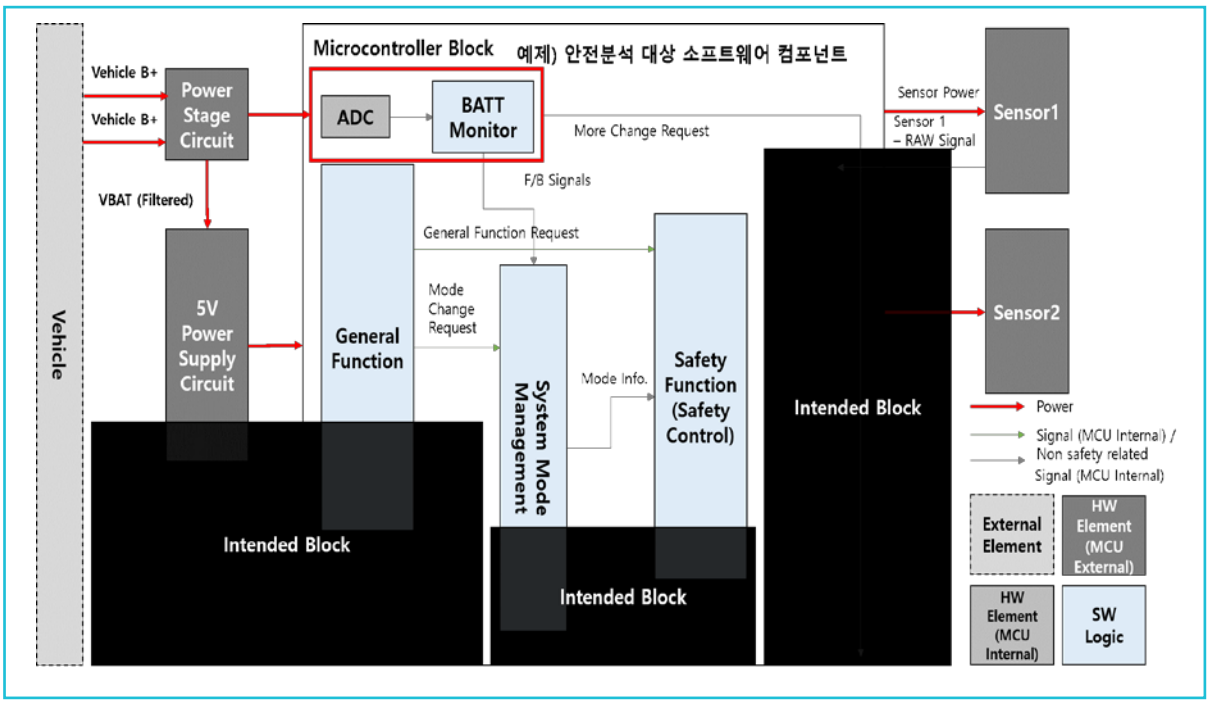
SW Component		Description
Functional failure modes	Faulty functionality	• SSR 이 complete 하지 않음 • SSR 에 un-written assumption 이 존재함 • SSR 간에 conflict 가 존재함 • SW 상에 의도하지 않은 extra functionality 가 존재함
	Faulty timing	• 요구되는 operation 이 too early 또는 too late 하게 수행됨 • Timer 의 값이 너무 크거나 작게 설정됨 • SW 설계상 의도하지 않은 race condition 이 존재
	Faulty sequence or order	• 요구되는 operation 이 정의된 순서 (required order)를 지키지 못함 • 요구되는 state transition 이 incorrect 함 • Dead state 가 존재함 • Orphan state 가 존재함 (orphan state 실행 상태에서 parent state 가 종료됨)

	Faulty data	• 특정 SW 모듈이 incorrect format 의 data 를 생성함 • Data format 은 correct 하지만, data 값이 incorrect 함 • Accuracy requirements 가 너무 tight 하거나 너무 tight 하지 않음
	Faulty error detection	• 자체 SW failure 또는 HW failure 에 대한 detection 실패 • 발생한 failure 로부터 recovery 실패
	False alarm	• False alarm 발생
	Faulty error handling	• Device error 무시
Interface failure modes	Faulty communication and processing	• Loss of network connection • Congested network • Processing 하기 위해 가용량보다 더 많은 resource 필요 • Incorrect command • No command • Too many command (processing capacity 초과)
	Faulty COTs interface	• COTs command 가 wrong verion 임 • Incorrect COTs command • Incorrect input to COTs command 등
	Faulty OS interface	• OS command fault
	Faulty database interface	• Database access/handling fault

사. SW 안전분석 예시

안전분석은 소프트웨어 아키텍처 설계를 대상으로 한다. Microcontroller Block 내부에 붉은색 박스로 표시된 ADC(하드웨어), BATT Monitor 가 분석대상이다. BATT Monitor 는 소프트웨어 컴포넌트이며, ADC 는 BATT Monitor 에 의해 모니터링 되는 하드웨어 컴포넌트이다.

그림 50. 소프트웨어 아키텍처(상위 수준)



분석된 안전분석결과는 다음과 같다.

그림 51. 소프트웨어 안전분석 결과 예제

System element of safety			Polential Failure Mode(NO/UA/Coarse/Subtle)	Polential Effects of Failure		SG Violation		Potential Casuse(s) of Failure	Current Safety Mechansim			Analysis Results	
Element Name	ASIL	Functions/ signals		Local-Module Level	Global-ECU Level	SG 01	SG 02		Detection / Failure Control(Process measure제외)	Latent Avoidance	SM	Verdict	Note/ Required Revision
BATT-ADC (HW Only)	B	12VPower(VS, VP)A/D converting	Converting fault(Nomal, limited operation range→out of range)	ECU deactivation (no operation, noLNcommunication)	SG01, SG02관련 영향성 없음								
			Converting fault(Out of range→normal, limited operation range)	ECU저전압, 고전압 상황 검출 실패(전원out of range 상태에서 ECU 구동 시도)	Effect를 특정할 수 없음 (보수적으로 SG01,02 모두 발생 가능하다고 가정)	O	O	Random HW failure - ADC 자체 failure	ADC health check(SM05) -Band-gap reference voltage check		SM 05	OK	Clock,ADC reference power가 rootcause가 될 수 있음 - Clock monitoring, VDDA에 대한 under voltage monitoring 수행함
BATT Monitor (Logic)	B	고전압, 저전압 검출	검출error(Normal, Limited operation range→out of range)	ECU deactivation(no operation, noLNcommunication)	SG01, SG02관련 영향성 없음								
			검출error(out of range→Normal, Limited operation range)	ECU 저전압,고전압 상황 검출 실패(전원 out to range 상태에서 ECU 구동 시도)	Effect를 특정할 수 없음 (보수적으로 SG01,02 모두 발생 가능하다고 가정)	O	O	Logic system aticfaiure	해당없음		SF F	OK	BATT 모니터링 자체가, SG01, SG02위반에 대응하기 위한 안전 기능으로, BATT 모니터링의 실패가 즉각적인 SG위반을 유도하지 않음
			No operation(over voltage, under voltage 검출 기능 수행 없음)	ECU 저전압,고전압 상황 검출 실패(전원 out to range 상태에서 ECU 구동 시도)	Effect를 특정할 수 없음 (보수적으로 SG01,02 모두 발생 가능하다고 가정)	O	O	Logic system aticfaiure	Program flow monitoing(SM10) Stack Monitoring(SM11)		SM 10, SM11	OK	해당 fault가 SW의 구동(execution 여부 및 execution timing)에 영향을 주는 경우, SM 10의 한 HW reset 발생
			RHF에 의한 기능 오동작(failure mode 특정 불가)	특정불가	특정불가(보수적으로 SG01,02 모두 발생 가능하다고 가정)	O	O	Random HW failure - Logic processing 관련	SM 12-SM 18 -RAM/ROM ECC, Register test, Clock monitoring, RAM/ROM test,		SM 12-18	OK	

위 표에 각 열이 의미하는 것은 다음과 같다.

- Element name
  분석하고자 하는 대상으로 소프트웨어(Logic) 또는 소프트웨어와 의존성을 가지는 하드웨어이다.
- ASIL
  해당 엘리먼트에 할당된 ASIL 수준
- Functions/Signal
  해당 엘리먼트가 수행하는 function 이나 출력되는 signal 정보

- Potential Failure Mode
  해당 엘리먼트가 가질 수 잇는 잠재적인 failure mode 로서 SHARD 를 적용하여 failure mode 를 식별

  - Potential Effects of Failure
    Local-Module Level: Potential Failure Mode 에 의해 해당 엘리먼트가 영향을 받는 결과
    Global-ECU Level: Local-Module Level 에 의해 ECU 전체 수준에서 영향받는 결과

- SG Violation
  Potential Effects of Failure 로 인해 safety goal(SG) 위배(violation)가 일어나는지에 대한 분석

- Potential Causes of Failure
  Potential Failure Mode 에 대한 잠정 원인

- Current Safety Mechanism
  Detection / Failure Control: Potential causes of failure 를 single point failure 관점에서 탐지하거나 고장발생을 방지할 수 있도록 현재 설계된 safety mechanism 의 식별
  Latent Avoidance: Latent failure 관점에서 탐지하거나 고장발생을 방지할 수 있도록 현재 설계된 safety mechanism 의 식별

- Analysis Results
  안전분석을 통해 SG Violation 을 일으킬 수 있는 Potential causes of Failure 에 대하여 현재 설계된 safety mechanism 이 충분한지 판단

- Note/Required Revision: 분석결과 필요한 내용을 기입하거나 변경되어야 하는 설계요구사항을 식별

6.6. 종속 고장분석

가. 종속 고장분석 템플릿

1. 개요 - 문서의 전반적인 소개와 내용을 기술한다. 1.1. 목적 - 문서의 목적에 대해 기술한다. 1.2. 용어 및 약어 정의

- 문서에 사용되는 용어 및 약어에 대해 기술한다.

1.3. 참조 문서

- 참조되는 문서에 대해 기술한다.

2. 종속고장분석

2.1. SW 컴포넌트 상세

SW 컴포넌트 ID/이름		ASIL	SSR ID	SW 컴포넌트 내부 기능 항목	SW 컴포넌트 입력 요소	SW 컴포넌트 출력 요소

2.2. SW 안전 분석과 종속 고장분석

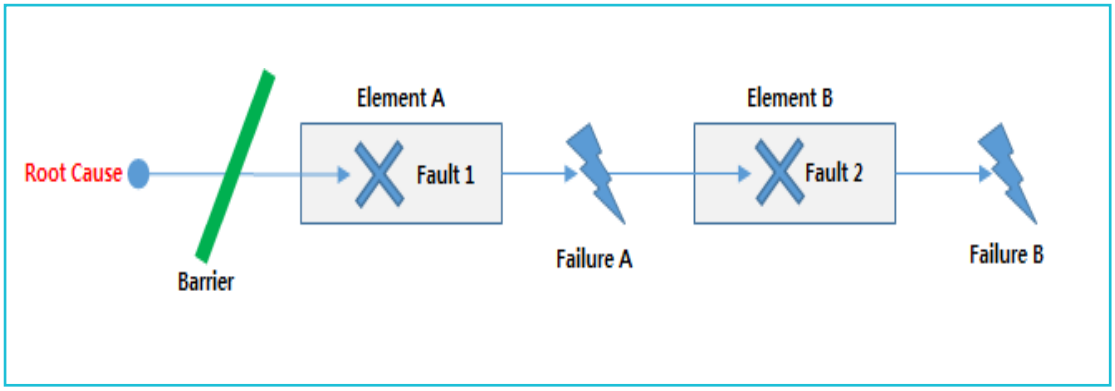
SW Component				Failure Mode Identification					Cause Analysis	Effect Analysis						Fault Control			Confirmation
Component Group	Component Name	ASIL	Type	Functionality /Property	Type	Usage	Guideword	Interpretation	Possible Causes	Effect	Violation of SG	CF		CCF		Counter Measure	Note	Effectiveness	
												CF	Independence Effect	Root cause of CCF	Independence Effect				

나. 공존성 기준(Co-existence criteria)

종속 고장은 공통원인 고장(common cause failure)와 연계 고장(cascading failure)로 분류될 수 있다. 해당 분류는 논리적인 분류로 특정 형태의 SW 결함이 항상 공통원인 고장 또는 연계 고장으로 작용하는 것은 아니다.

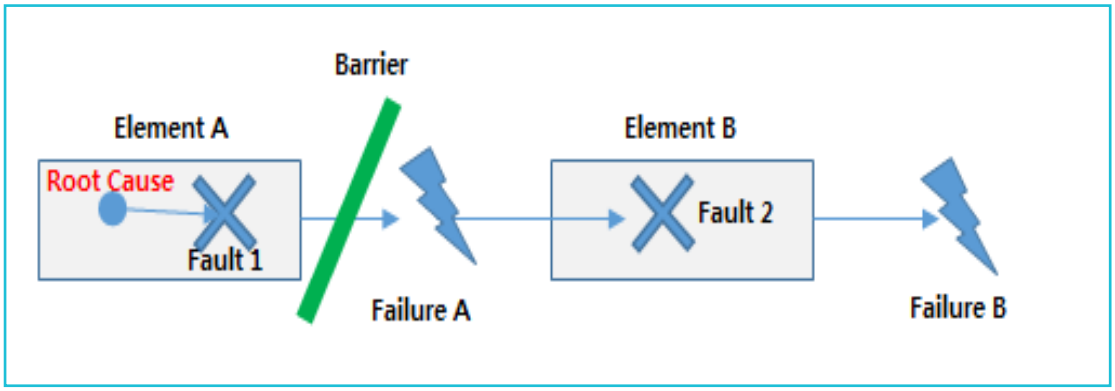
원인이 외부에 있을 경우에는 외부원인으로 인해 내부 엘리먼트(element) 결함(fault)가 유발되어 해당 결함으로 발생하는 고장(failure)가 타 엘리먼트에 결함을 연쇄적으로 유발하는 경우는 아래와 같다.

그림 52. 원인이 외부 엘리먼트에 존재할 경우



원인이 내부에 있을 경우에는 내부원인으로 인해 엘리먼트가 결함이 유발되어 해당 결함으로 고장이 발생하는 경우는 다음과 같다.

그림 53. 원인이 내부 엘리먼트에 존재할 경우



이와 같은 근본원인으로 연계 고장(cascading failure)을 방지하기 위해서는 근본원인의 발생이 사전에 차단되도록 설계하거나(prevention) 또는 근본원인 유발시에 이로 인한 1 차적인 고장(failure)을 검출하고 이에 대한 대응 동작(safety reaction)을 수행하여 2 차적인 고장의 발생을 방지하는 설계가 요구된다.

다. 간섭에 대한 자유성(Freedom from interference)

안전 요구사항을 위반할 수 있는 둘 이상의 엘리먼트간에 연계고장이 없는 상태를 간섭에 대한 자유성이라고 정의한다. 여기서 소프트웨어 파티셔닝(partitioning)이란 설계 시 기능 또는

엘리먼트를 분리하는 것으로 연계 고장(cascading failure)을 방지하기 위한 목적으로 결함을 격리시키기 위해 사용된다.

라. SW 컴포넌트 상세 예제

SW 안전 분석 및 종속 고장 분석을 설명하기 위해 SW 아키텍처 및 SW 컴포넌트 사양 예제를 기반으로 SW 안전 분석 및 종속 고장 분석 예제는 다음과 같다.

표 39. SW 컴포넌트 상세 예제

SW 컴포넌트 ID/이름		ASIL	SSR ID	SW 컴포넌트 내부 기능 항목	SW 컴포넌트 입력 요소	SW 컴포넌트 출력 요소
SWE01	ADC_Health_Check	B	SSR_35	• ADC 검증 기능	• ADC band gap 전압 측정 값 • IGN 상태 정보 (Global)	• HW reset • ADC failure 판정 정보 (Global) • ADC band gap 전압 측정 설정
SWE02	BATT_Monitoring	B	SSR_01	• BATT 고전압, 저전압 판정	• BATT voltage (필터링된 값) • Cranking 완료 정보 (Global)	• BATT error 정보 (Global) • HW reset

마. SW 안전 분석 & 종속 고장분석 예제

그림 54. SW 안전 분석 및 종속 고장분석 예제

SW Component		Failure Mode Identification				Cause Analysis	Effect Analysis						Fault Control		Confirmation
Component Name	ASIL	Functionality /Property	Type	Guide word	Interpretation	Possible Causes	Effect	Violation of SG	CF		CCF		Counter Measure	Effectiveness	
									CF	Independence Effect	Root cause of CCF	Independence Effect			
BATT Monitoring	B	Function (over voltage, under voltage detection)	Function	Omission	BATT_Monitoring 기능 수행 안함	Task scheduling error Deadlock /livelock	Output signal (BATT error 정보) update 되지 않음 CAN_Manager에 의한 Warning 신호 발생 불가	YES (SG01)	O	없음	X	없음	Program flow monitoring	High	OK
				Commission	규정된 진단 시점보다 이르거나 늦은 시점에서 진단 수행	Task scheduling error	FTTI 위배 가능성 존재	YES (SG01)	O	없음	X	없음	Program flow monitoring	High	OK
		Signal (BATT error 정보)	Output	Coarse	BATT 전압 OV/UV 발생 → 미 발생으로 출력 생성	Random HW failure (RAM) Variable write error	CAN_Manager에 의한 Warning 신호 발생 불가	YES (SG01)	O	없음	X	없음	RAM ECC	Medium	OK

6.7. 소프트웨어 단위 설계

가. 소프트웨어 단위 설계 템플릿

1. 개요

- 문서의 전반적인 소개와 내용을 기술한다.

1.1. 목적

- 문서의 목적에 대해 기술한다.

1.2. 용어 및 약어 정의

- 문서에 사용되는 용어 및 약어에 대해 기술한다.

1.3. 참조 문서

- 참조되는 문서에 대해 기술한다.

2. SW 단위 설계

2.1. SW 단위명

Unit ID/Name	단위 ID 와 명칭을 기술한다.			
Unit ASIL	단위의 ASIL 등급 작성한다.		SW REQ./ASIL	추적을위한 SSS ID 작성
Inputs	Name	Input Type	Data Type	Specification

Outputs	Name	Output Type	Data Type	Specification
	-			
Functionality	기능에 대해 기술한다.			
HW Resource Usage	활용하는 HW 를 기술한다.			
Target Execution Time Duration	실행되는 시간을 기술한다.			

2.2. SW 단위명

Unit ID/Name	단위 ID 와 명칭을 기술한다.			
Unit ASIL	단위의 ASIL 등급 작성한다.		SW REQ./ASIL	추적을 위한 SSS ID 작성
Inputs	Name	Input Type	Data Type	Specification
Outputs	Name	Output Type	Data Type	Specification
	-			
Functionality	Pseudo code 등을 기술한다.			
HW Resource Usage	활용하는 HW 를 기술한다.			
Target Execution Time Duration	실행되는 시간을 기술한다.			

단위 설계 명세서 항목

- 함수의 Prototype
- ASIL 등급 및 관련된 SSR 의 식별
- 함수의 입력 데이터(입력 argument, 전역변수)
- 함수의 출력 데이터(return value, 전역변수)
- 함수의 기능 (예: Pseudo code)
- 함수가 사용하는 HW 자원
- 함수 수행 시간

나. 소프트웨어 단위 설계 예

소프트웨어 단위 설계는 일반적으로 소프트웨어 아키텍처 단계에서 식별된 소프트웨어 컴포넌트 단위로 작성된다. 단위 설계 작성 예제는 다음과 같다.

표 40. 단위 설계 예제

Unit ID/Name	SW_Unit_01 / UINT8 ADC_Config_BATT(Void)			
Unit ASIL	ASIL B		**SW REQ./ASIL**	SSR_01
Inputs	**Name**	**Input Type**	**Data Type**	**Specification**
	None			
Outputs	**Name**	**Output Type**	**Data Type**	**Specification**
	-	Return Value	UINT8	0xF1: Configuration OK 0x1C: Configuration Fail

Functionality	1. ADC 채널 설정: 3 번 채널
	2. ADC 동작 모드 설정: Single Measurement
	#Pseudo code
	UINT8 ADC_Config_BATT(Void)
	{
	1. ADC_REG 03 → register clear
	2. 3 번 channel ADC → enable
	3. 3 번 channel ADC 동작 모드 → Single Measurement
	4. Read ADC_REG 03
	If (register == correct)
	Return 0xF1
	Else
	Return 0x1C
	}			

HW Resource	MCU ADC Channel 03			

Usage	
Target Execution Time Duration	Under 200us

다음의 항목을 기술하여 소프트웨어 단위의 사양을 정의한다.

- 각 소프트웨어 단위의 기능(pseudo code, flow chart 등을 사용)
- 각 소프트웨어 단위의 인터페이스(function prototype 등)
- 각 소프트웨어 단위의 execution trigger 조건
- 각 소프트웨어 단위의 ASIL(안전관련인 경우)
- 각 소프트웨어 단위가 사용하는 data 및 data 의 structure

6.8. 소프트웨어 검증

ISO 26262 는 소프트웨어 단위 시험, 소프트웨어 통합 및 시험, 소프트웨어 안전 요구사항 검증 단계에서 ISO 26262 파트 8, 9 검증에 따라서 검증 계획, 검증 명세서, 검증 보고서를 작성하도록 하고 있다. 검증 계획은 시험 계획, 검증 명세서는 시험 케이스, 검증 보고서는 시험 보고서로 볼 수 있으며 각 시험 단계별 작성 항목은 유사하므로 단일 템플릿을 활용할 수 있다. 본 가이드에서는 시험 계획, 시험 명세서, 시험 보고서 템플릿으로 제시한다.

가. 시험 계획 템플릿

1. 개요
    - 문서의 전반적인 소개와 내용을 기술한다.

1.1. 목적
    - 문서의 목적에 대해 기술한다.

1.2. 용어 및 약어 정의
    - 문서에 사용되는 용어 및 약어에 대해 기술한다.

1.3. 참조 문서
    - 참조되는 문서에 대해 기술한다.

2. 역할 및 책임

- 시험 수행 활동별로 담당자 및 역할을 기술한다.

3. 시험 일정
- 시험 활동별 시작 일정, 종료 일정을 기술한다.

4. 시험 대상 및 범위

4.1. 시험 대상
- 시험 대상을 기술하며 단위 시험은 소프트웨어 단위, 통합 시험은 소프트웨어 컴포넌트가 해당된다.

4.2. 시험 범위
- 시험 수행에 대한 범위를 기술한다.

4.3. 가정 및 제약 사항
- 시험 수행에 대한 가정 및 제약 사항을 기술한다.

5. 시험 전략

5.1. 시험 방법
- ASIL 등급별 요구사항 기반 시험, 인터페이스 시험 등의 시험 방법을 기술한다.

5.2. 시험 케이스 설계 방법
- ASIL 등급별 경계 값 분석, 동치 분할 등의 시험 케이스 설계 방법을 기술한다.

5.3. 회귀 시험
- 회귀 시험 조건, 시험 대상 선정, 절차를 기술한다.

5.4. 시험 중단 및 재개 기준
- 시험 수행의 중단 및 재개 기준을 기술한다.

6. 시험 산출물
- 시험 수행 활동의 결과 산출물을 기술한다.

7. 시험 종료 기준
- 시험에 대한 종료 기준을 기술한다.

8. 시험 환경 및 도구
- ASIL 등급별 시험 환경과 시험에 사용되는 장비, 소프트웨어 도구를 기술한다.
- 시험 환경 및 도구의 필요 기간, 담당자를 기술한다.

나. 시험 케이스 템플릿

1. 시험 ID
- 시험 항목을 고유하게 식별할 수 있는 ID 를 기술한다.

2. 시험 대상
- 시험 대상이 되는 항목에 대한 ID 를 기술한다.

3. 시험명
- 시험에 대한 명칭을 기술한다.

4. 테스트 절차
- 시험을 수행하기 위한 절차를 수행 순서에 맞춰 기술한다.

5. 시험 전제 조건
- 시험을 수행하기 위한 전제 조건을 기술한다.

6. 입력 값
- 시험 수행에 입력하는 값을 입력한다.

7. 기대 결과
- 시험 입력 값 및 시험 절차에 따라서 수행한 경우 정상적으로 출력되는 예상 값을 기술한다.

8. 수행 결과
- 시험 입력 값 및 시험 절차에 따라서 수행한 결과 값을 기술한다.

9. 시험 결과
- 수행 결과에 기대 값과 비교하여 합격 및 불합격을 기술한다.

다. 시험 보고서 템플릿

1. 개요
- 문서의 전반적인 소개와 내용을 기술한다.

1.1. 목적
- 문서의 목적에 대해 기술한다.

1.2. 용어 및 약어 정의
- 문서에 사용되는 용어 및 약어에 대해 기술한다.

1.3. 참조 문서
- 참조되는 문서에 대해 기술한다.

2. 역할 및 책임
- 시험 수행 활동별로 담당자 및 역할을 기술한다.

3. 시험 일정
- 시험 활동별 시작 일정, 종료 일정을 기술한다.

4. 시험 결과
- 시험 수행에 대한 측정 결과 및 합격/불합격 결과를 기술한다.
- 시험 결과와 함께 시험에 대한 분석 결과, 권고 사항을 기술한다.

5. 시험 산출물
- 시험 수행 결과 산출물을 기술한다.

6. 시험 결과 잔존 이슈

- 시험 수행 결과 해결되지 않은 결함, 이슈 등을 기술한다.

본 가이드에서는 소프트웨어 단위 시험, 통합 시험, 안전 요구사항 검증에서 공통적으로 수행하는 시험 방법과 시험 케이스 생성 방법에 대해서 설명한다.

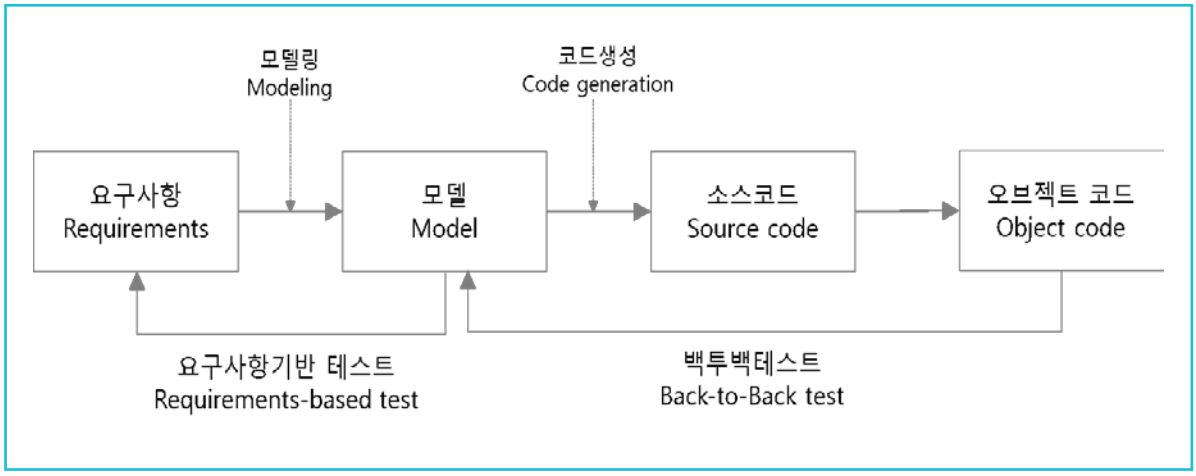
라. 시험 방법

1) 요구사항 기반 시험

요구사항기반 시험은 시험 아이템이 최종 사용자의 요구사항을 만족하는지를 확인하는 시험이다. 요구사항기반 시험은 정황(Context)에 크게 의존적이며 요구사항이 불완전하거나 일관성이 없으면 기대결과를 정확히 알 수 없으므로 시험에 많은 어려움을 겪는다. 요구사항에 우선 순위 정보가 있다면 시험 우선 순위를 선정하는데 도움이 된다.

백투백 테스트(back to back test) 관점에서는 모델이 요구사항을 모두 올바르게 구현했는지 확인한다. 아래 그림과 같이 요구사항은 실행 가능한 모델로 변환되므로 시뮬레이션 및 모델의 요구사항 기반 시험이 가능하다. 모델이 모든 요구사항을 처리함을 검증하기 위해 요구사항 추적표를 작성하여 요구사항을 빠짐없이 테스트 케이스로 만들고 실행했음을 보여줘야 한다. 모델과 코드간 비교 시험은 동일한 시험 케이스를 사용해 모델과 코드의 수행 결과를 비교해 모델과 코드 실행 결과가 같음을 확인한다. 요구사항을 기반으로 만든 시험 케이스는 이후 MIL, SIL, PIL 레벨에서 실행되고 그 결과를 비교한다.

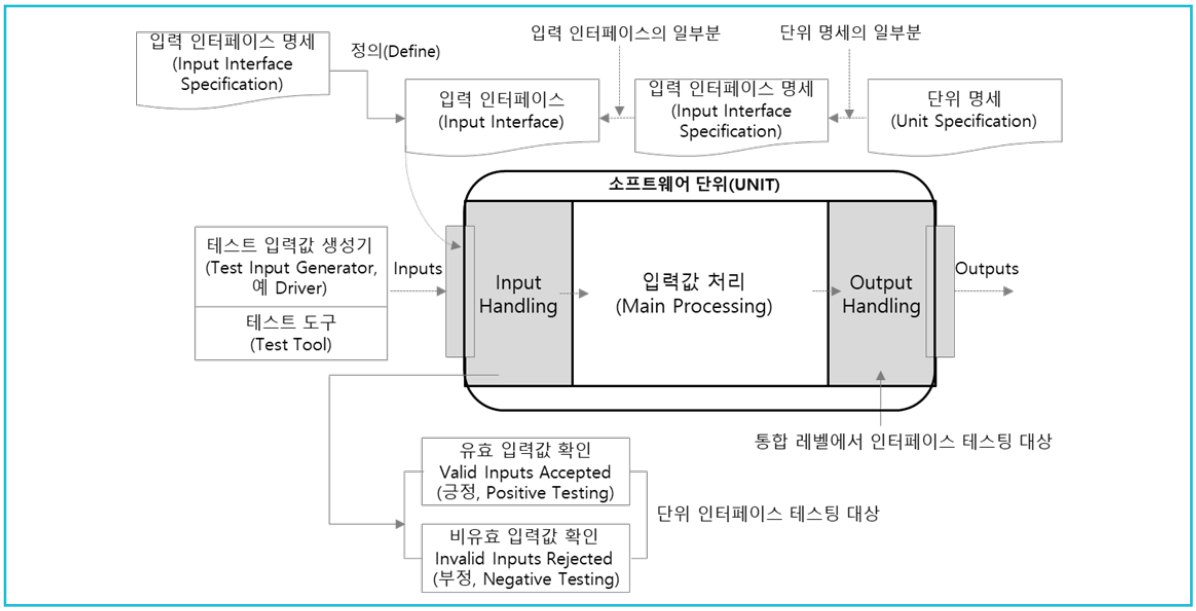
그림 55. 요구사항 기반 시험과 백투백 테스트



2) 인터페이스 시험

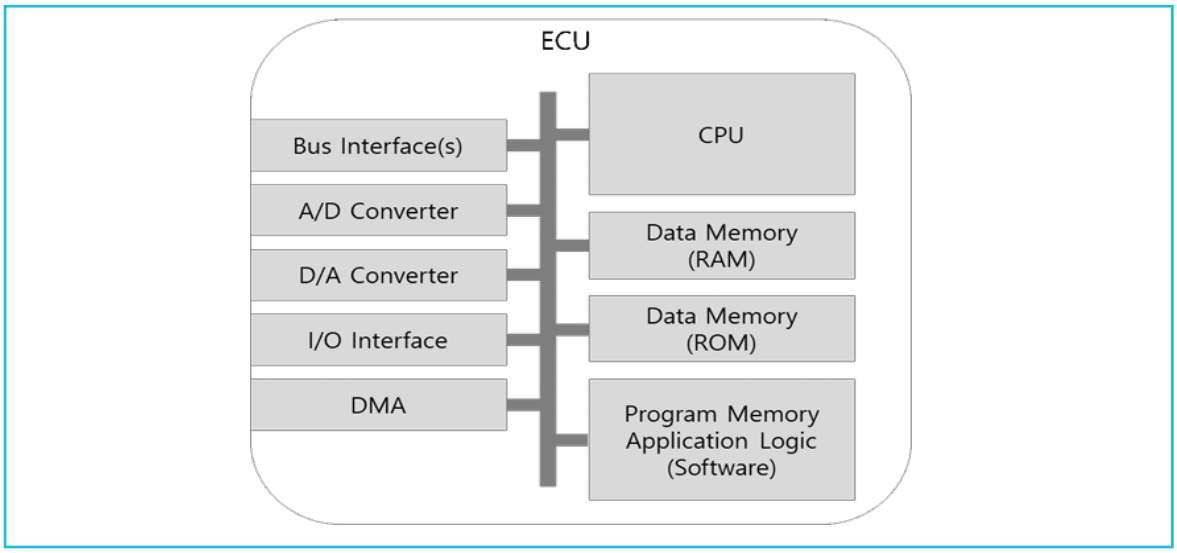
인터페이스 시험은 단위 시험에서는 소프트웨어 단위 내 입력 인터페이스 시험을 수행한다. 소프트웨어 통합 시험에서는 다른 단위와 입력 및 출력 인터페이스를 시험한다. 소프트웨어 단위인 함수 또는 모듈이 함수 또는 모듈 내에서 입력이 정상적으로 처리되는지, 다른 함수나 모듈의 인터페이스와 입력/출력 동작을 정상적으로 수행하는지 시험한다. 다음 그림과 같이 모듈 내의 입력 처리를 중심으로 시험한다.

그림 56. 단위 인터페이스 시험 구성



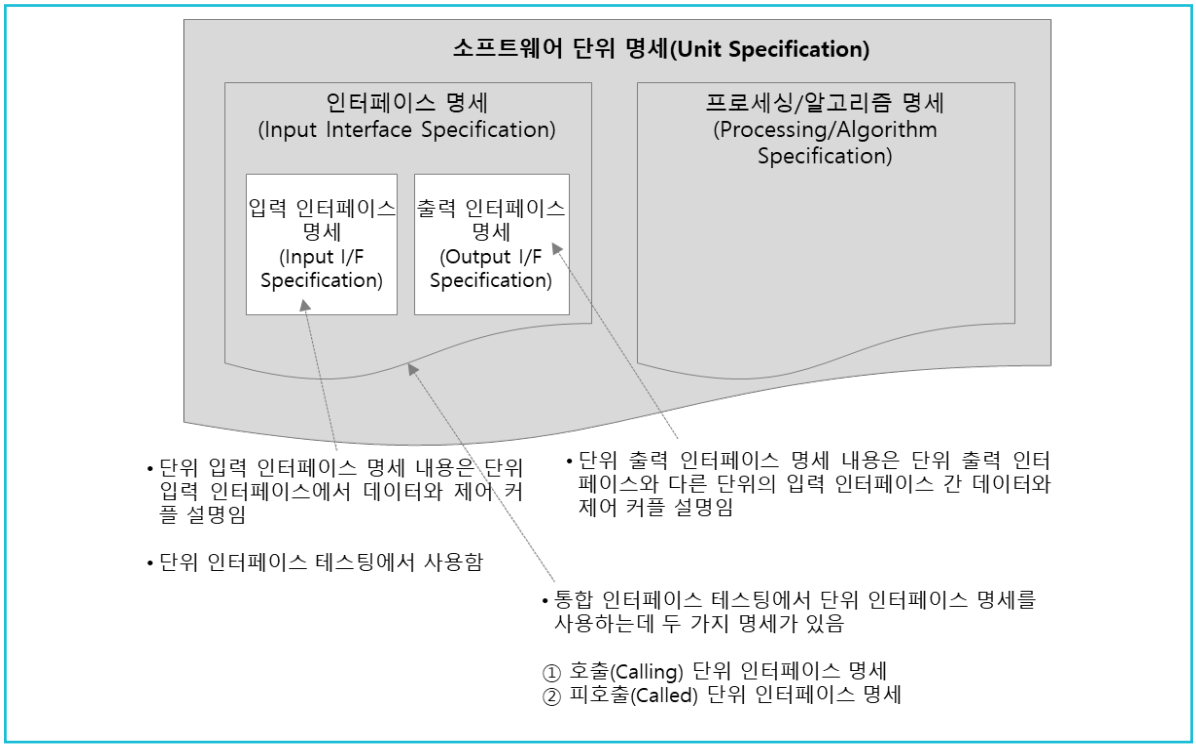
인터페이스 통합 시험은 인터페이스 단위 시험이 끝난 후 수행한다. 각 단위에서 입력 처리 동작 여부를 확인 후 인터페이스 통합 시험을 진행해야 어떤 단위에서 입력 처리 관련 결함이 발생했는지 확인이 가능하다. 인터페이스 통합 시험은 출력 처리 확인에 중점을 두며 여러 단위가 있을 경우 단위 간 통합 시 출력을 확인한다. 인터페이스 시험에서 사용할 수 있는 시험 케이스 생성 방법에는 동등 분할, 경계 값 분석이 가능하다. 이를 통해서 인터페이스에서 모든 입력 유효 값이 적합하게 처리되는지 확인하고, 비유효 값의 처리에 대해서 확인할 수 있다. 인터페이스 시험에서 발견되는 결함은 인터페이스 불일치, 입력 및 출력 명세 결함으로 분류할 수 있다. 인터페이스의 예로 다음 그림과 같이 ECU 내에 Bus Interface, A/D Converter, I/O Interface 등이 있다.

그림 57. 단위 인터페이스 예시



인터페이스 시험을 수행하기 위해서는 단위 인터페이스에 대한 정의가 필요하며 단위 인터페이스 명세는 다음과 같이 구성할 수 있다.

그림 58. 단위 인터페이스 명세



마. 시험 케이스 생성 방법

ASIL B 에 해당하는 시험 케이스 생성 방법에는 동치 클래스 생성 및 분석, 경계 값의 분석이 있으며 적용 예제에 다음과 같이 적용할 수 있다.

1) 동치 클래스의 생성 및 분석

차량전원은 유효 값과 비유효 값을 다음과 같이 구성하여 시험 조건을 구성할 수 있다.

표 41. 동치 클래스 시험 조건 예제

구 분	입력값	출력값
유효 값	• 7.0V ≤ 전압 ≤ 16.0V	• MCU 로 정상 값 전달
비유효 값	• 전압 > 16.0V • 전압 < 7.0V	• CAN signal 을 통해 이상 신호 전송

표 42. 동치 클래스 시험 케이스 예제

구 분	시험 케이스	
	사전조건 / 입력	기대결과
7.0V ≤ 전압 ≤ 16.0V	전압 = 13.0V	MCU 로 정상 값 전달
전압 > 16.0V	전압 = 18.0V	CAN signal 을 통해 이상 신호 전송
전압 < 7.0V	전압 = 5.0V	CAN signal 을 통해 이상 신호 전송

2) 경계값의 분석

경계값 분석은 동등분할 영역의 경계에서 결함을 발견할 확률이 높기 때문에 경계를 중심으로 시험 케이스를 생성한다. 경계값 적용 방법은 다음과 같다.

• 시험 대상을 분석해 동치 클래스를 식별한다.

- 각 경계에서 유효 경계값과 비유효 경계값을 식별한다.
- 유효 경계값은 해당 경계 조건을 만족하는 가장 인접한 값, 비유효 경계값은 해당 경계 조건을 벗어난 가장 인접한 값이다 7.0V ≤ 전압 ≤ 16.0V 에서 경계는 7.0V 와 16.0V 이다.

표 43. 경계 값 분석 시험 조건 예제

구 분	입력값	출력값
유효값	• 7.0V ≤ 전압 ≤ 16.0V	• MCU 로 정상 값 전달
비유효값	• 전압 > 16.0V • 전압 < 7.0V	• CAN signal 을 통해 이상 신호 전송

표 44. 경계 값 분석 시험 케이스 예제

동치 분할	경계값	시험 케이스	
		사전조건 / 입력 값	기대결과
7.0V ≤ 전압 ≤ 16.0V	전압 = 7.0V	전압 = 7.0V	MCU 로 정상 값 전달
	전압 = 16.0V	전압 = 16.0V	MCU 로 정상 값 전달
전압 < 7.0V	전압 = 6.9V	전압 = 6.9V	CAN signal 을 통해 이상 신호 전송
전압 > 16.0V	전압 = 16.1V	전압 = 16.1V	CAN signal 을 통해 이상 신호 전송

부록 IV 용어집

본 가이드에서 자주 사용한 용어를 국제 표준인 ISO 26262, ISO 29119 에서 선별하여 설명하였다.

용어	설명
아키텍처 (Architecture)	빌딩 블록(building block) 및 빌딩 블록의 경계와 인터페이스를 식별할 수 있게 해주고, 기능이 할당된 하드웨어와 소프트웨어 엘리먼트를 포함하는 아이템이나 기능 또는 시스템, 엘리먼트 구조를 표현한 것
ASIL 분해 (ASIL decomposition)	안전 요구사항을 충분히 독립적인 엘리먼트에 중복하여 분배함으로써, 해당 엘리먼트에 할당된 중복된 안전 요구사항의 ASIL 을 감소시키는 목적을 가짐
가용성 (Availability)	어떤 시간 또는 기간 동안, 필요한 외부자원을 이용할 수 있다고 가정했을 때, 제품이 주어진 환경에서 필요한 기능을 수행할 수 있는 상태에 있는 능력
베이스라인 (baseline)	형상관리 하에 있는 관리 프로세스를 통해 향후 개발을 위한 기준으로 사용되는 하나 이상의 작업 산출물, 아이템 또는 엘리먼트 버전
분기 커버리지 (branch coverage)	실행된 제어 흐름 분기의 백분율 100% 분기 커버리지는 100% 구문 커버리지를 의미하며 조건문(if-statement)은 항상 두개의 분기(branch) 참(true)과 거짓(false)을 가지며, 이것은 else 절(clause)의 유무와 상관없다
컴포넌트 (Component)	논리적 및 기술적으로 분리 가능하고 하나 이상의 하드웨어 소자나 하나 이상의 소프트웨어 단위로 구성된 시스템 수준은 아닌 엘리먼트
엘리먼트 (element)	컴포넌트, 하드웨어, 소프트웨어, 하드웨어 소자, 소프트웨어 단위를 포함한 시스템 또는 시스템의 일부
임베디드 소프트웨어 (embedded software)	프로세싱 엘리먼트에서 수행되는 완전히 통합된 소프트웨어. 프로세싱 엘리먼트는 일반적으로 마이크로 컨트롤러, FPGA(field programmable gate array)나 ASIC(application specific integrated circuit)이지만 보다 복잡한 컴포넌트나 하위시스템도 될 수 있다
오류 (error)	계산 값, 관찰 값 또는 측정된 값이나 조건에 대비하여 명시적인 참값 또는 이론적으로 바른 값 또는 조건과의 불일치. 오류는 예측 불가능한 작동상황 또는 시스템이나 하위시스템, 컴포넌트 내에 발생하는 결함으로 인해 일어날 수 있으며 결함은 엘리먼트내에서 그 자체가 오류로 나타날 수 있으며 오류는 결국

	고장의 원인이 된다
외부 수단 (external measure)	아이템에서 비롯된 리스크를 감소시키거나 완화시켜주는 수단으로 아이템과 분리 및 구분됨
고장 (failure)	요구된 대로 기능을 수행하는 엘리먼트의 능력이 종료되는 것 부정확한 명세서는 고장의 원인이다.
고장형태 (failure mode)	엘리먼트 또는 아이템에 고장이 발생하는 방식
고장율 (failure rate)	하드웨어 엘리먼트에 대한 고장을 생존 확률로 나눈 확률 밀도 고장율은 일정하다고 가정하며 일반적으로 "λ"로 표시된다
결함 (fault)	엘리먼트 또는 아이템에 고장을 일으킬 수 잇는 비정상적인 상태 영구, 불연속, 순간 결함(특히 소프트웨어 오류)이 고려되며 불연속 결함은 여러 번 반복해서 발생하다가 사라진다. 이러한 종류의 결함은 예를 들어, 컴포넌트가 파괴되기 직전이나 스위치의 작은 결함으로 인해 발생할 수 있다. 일부 시스템적 결함(예를 들어 타이밍 차이)이 불연속 결함을 야기할 수 있다
결함 모델 (fault model)	결함으로 인해 발생하는 고장모드를 표시한 것 결함 모델은 일반적으로 현장 경험이나 신뢰성 핸드북을 기반으로 한다
결함 반응시간 (fault reaction time)	결함을 감지하여 안전한 상태에 이르는데 소요되는 시간
결함 허용 시간간격 (fault tolerant time interval)	결함 한 개 또는 여러 결함들이 위험한 사건이 발생하기 전에 시스템에 존재할 수 있는 시간
필드 데이터 (field data)	총 작동시간, 발생한 모든 고장과 서비스 중 발견된 이상점과 같이 아이템이나 엘리먼트의 사용 시 획득된 데이터 필드 데이터는 일반적으로 고객이 사용함으로써 발생한다.
정형 표기법 (formal notation)	완벽하게 정의된 구문 및 의미체계가 있는 기술 기법 Z(Zed); NuSMV; PVS(Prototype Verification System); VDM(Vienna Development Method)
정형 검증 (formal verification)	요구되는 행위를 정형표기법으로 명세한 것에 대해 시스템의 정확성을 증명하기 위하여 사용되는 방법
기능 개념 (functional concept)	원하는 행위를 구현하는데 필요한 의도된 기능 및 상호작용에 대한 명세 기능 개념은 개념 단계 동안 개발된다
위해 (harm)	개인의 건강에 미치는 물리적인 상해나 손상
위험원	아이템의 오작동 행위로 인해 야기되는 위해에 대한 잠재 근원

(hazard)	이 정의는 ISO 26262 의 범위로 제한되고 있다. 보다 일반적인 정의는 위해의 잠재 근원이다
위험원 분석 및 리스크 평가 (hazard analysis and risk assessment)	아이템의 위험사건을 식별 및 분류하고 해당 위험을 방지 또는 완화시킴으로써 비합리적 리스크를 피하기 위하여 필요한 안전 목표와 ASIL 을 명시하는 방법
비정형 표기법 (informal notation)	완전하게 정의된 구문의 형태를 가지지 않은 기술 기법 그림이나 다이어그램과 같은 기술방법과 불완전한 구문 정의는 그 의미가 완전하게 정의되지 않았다는 것을 말한다
상속 (inheritance)	개발 프로세스 동안 다음 수준단계까지 요구사항의 특성을 변경하지 않고 전달하는 것
초기 ASIL (initial ASIL)	위험원 분석에서 도출된 ASIL 또는 ASIL 분해 이전의 ASIL 초기 ASIL 은 ASIL 분해 또는 향후 ASIL 분해를 위한 시작점이다.
인스펙션 (inspection)	이상을 감지하기 위해 공식 절차에 따라 작업 산출물을 조사 인스펙션은 하나의 검증 수단이며 인스펙션은 일반적으로 관련 아이템이나 엘리먼트의 작동상황은 포함하지 않는다는 점에서 시험과 다르다. 이상점이 검출되면, 일반적으로 재 작업을 하고 재 작업된 산출물을 재 인스펙션 한다. 공식 절차는 일반적으로 사전 정의된 절차, 체크리스트, 중재자, 결과 검토로 이루어진다.
의도된 기능성 (intended functionality)	안전 매커니즘을 제외한 아이템, 시스템, 엘리먼트에 대해 명시된 행위
아이템 (item)	ISO 26262 가 적용되는 자동차 수준에서 기능을 수행하는 시스템 또는 시스템 배열
아이템 개발 (item development)	아이템을 구현하는 완전한 형태의 프로세스
잠재 결함 (latent fault)	다중점 결함 검출 간격이내에 결함 발생 유무가 안전 매커니즘에 의해 감지되지 않고 운전자에 의해서도 인식되지 않는 다중점 결함
수명주기 (lifecycle)	아이템 개념부터 폐기에 걸친 전체 단계
오작동 행위 (malfunctioning behavior)	설계된 대로 작동하지 않는 아이템의 고장이나 의도되지 않은 행위
모델 기반 개발 (model-based development)	개발할 엘리먼트의 기능 행위를 기술하기 위해 모델을 이용하는 개발 모델에 사용되는 추상화 수준에 따라 모델은 시뮬레이션이나 코드 생성 또는 둘 다에 사용될 수 있다.
수정 (modification)	아이템의 허가된 개조(alteration) ISO 26262 에서 수정(modification)은 수명주기 조정(tailoring)을 위해

	재사용 하는 것이며 수정(modification)은 기존 아이템에서 새로운 아이템을 생성하기 위해 적용되는 반면 변경(change)은 아이템의 수명주기 동안 적용된다.
신규 개발 (new development)	이전에 명시하지 않은 기능이나 기존 기능에 새롭게 구현한 것 또한 이 두가지 모두에 해당하는 아이템 구현 프로세스
분할 (partitioning)	설계 시 기능 또는 엘리먼트를 분리하는 것 분할은 연계고장을 방지하기 위한 목적으로 결함을 격리시키기 위해 사용될 수 있다. 분할된 설계 엘리먼트간 간섭에 대한 자유성을 구현하기 위하여 추가적인 비기능적 요구사항이 도입될 수 있다.
실증 논거 (proven in use argument)	후보품 사용으로 발생한 필드 데이터 분석에 근거하여 해당 ASIL 의 요구사항에 맞게 사용하는 아이템의 안전 목표를 손상시킬 수 있는 후보품의 고장 확률에 관한 입증자료
실증 대체 (proven in use credit)	실증 논거에 의하여 제공된 수명주기 하위단계를 해당 작업 산출물로 대체하는 것
중복(구현) (redundancy)	엘리먼트가 요구되는 기능을 수행하거나 정보를 표시하는데 충분한 수단(means)에 부가적으로 추가한 수단 ISO 26262 에서 중복은 안전 목표 또는 명시된 안전 요구사항을 달성하거나 안전 관련 정보를 표시하기 위해 사용된다. 이중 기능 컴포넌트는 가용성 증가 또는 결함 감지를 위한 중복(구현)의 예이다. 안전관련 정보를 나타내는 데이터에 패리티 비트 추가를 추가하는 것은 결함 검출을 위해 중복구현을 실시한 것이다.
회귀전략 (regression strategy)	아이템이나 엘리먼트를 변경 하였을 때 해당 변경작업이 기존에 검증된 아이템이나 엘리먼트의 부품이나 그 특성에 영향을 끼치지 않았다는 것을 증명하기 위한 전략
잔존결함 (residual fault)	하드웨어 엘리먼트에서 발생하는 안전 목표의 위반을 야기하는 결함 부분으로 안전 매커니즘에 의해 보호되지 않음 이 것의 전제는 하드웨어 엘리먼트는 결함의 일부분에 대해서만 안전 매커니즘 커버리지를 갖는다는 것이다. 어는 한 고장형태 모드가 낮은 커버리지(60%)를 갖는다고 한다면 동일한 고장형태의 나머지 40%가 잔존 결함이다.
잔존 리스크 (residual risk)	안전 수단을 구현하더라도 남아있는 리스크
검토 (review)	검토 목적에 따라서 작업 산출물이 의도된 목적대로 달성되었는지 작업 산출물을 조사하는 것 검토는 체크리스트로 수행될 수 있다.
리스크 (risk)	위해 발생확률과 위해의 심각도 간의 결함

강건설계 (robust design)	유효하지 않은 입력 또는 스트레스 환경 조건에서도 올바르게 작동할 능력이 있는 설계 - 강건성은 다음과 같이 이해될 수 있다. 소프트웨어의 경우 강건성은 비정상적인 입력과 조건에 올바르게 작동되는 능력이다. - 하드웨어의 경우 강건성은 설계 한계 내에서 환경 스트레스를 견디고 서비스 수명 내내 안정적일 수 있는 능력이다. - ISO 26262 맥락에서 강건성이란, 경계영역에서도 안전한 행위를 제공하는 능력이다.
안전활동 (safety activity)	안전 수명주기상에 있는 하나 이상의 하위단계에서 수행된 활동
안전 아키텍처 (safety architecture)	안전 요구사항을 충족하기 위한 엘리먼트와 그것들 간의 상호작용의 집합
안전 목표 (safety goal)	위험원 분석 및 리스크 평가의 결과로 도출된 최상위 안전 요구사항 한 가지 안전 목표가 여러 위험원과 관련될 수 있으며 여러 가지 안전 목표가 단일 위험원과 관련될 수 있다.
안전 수단 (safety measure)	시스템적 고장을 방지하거나 제어하고 하드웨어 우발 고장을 감지 또는 제어하거나 관련된 위해에 대한 영향을 완화하기 위한 활동이 기술적인 해결책 안전 수단의 예로 FMEA 와 글로벌 변수를 사용하지 않은 소프트웨어를 들 수 있다. 안전 수단은 안전 매커니즘을 포함한다.
안전 매커니즘 (safety mechanism)	결함을 검출하거나 고장을 제어함으로써, 안전한 상태에 도달하거나 유지하기 위하여 전기/전자장치 기능이나 엘리먼트 또는 기타 기술로 구현된 기술적 해결책 안전 매커니즘은 단일고장을 야기하는 결함을 방지하거나 결함이 잠재화 되는 것을 방지하기 위하여 아이템 내에 구현된다. 안전 매커니즘은 기능안전 개념에서 정의한 바와 같이 다음 두 가지 중 하나이다. - 아이템을 안전 상태로 전이 또는 유지 시키거나 - 운전자가 고장의 영향을 제어할 수 있도록 운전자에게 경보를 제공하는 것
안전관련 엘리먼트 (safety-related element)	안전 목표를 위반할 수 있는 잠재성을 가진 엘리먼트 고장-안전(Fail-safe) 엘리먼트는 최소한 하나의 안전 목표에 기억할 수 있을 때 안전관련 엘리먼트로 간주한다.
안전관련 기능 (sfatey-related function)	안전 목표를 위반할 수 있는 잠재성을 가진 기능
안전관련 특별특성	아이템이나 엘리먼트의 특성 또는 통상적으로 예측 가능한 어떤

(safety-related special characteristic)	차이로 인해 기능안전에 영향을 미치거나 기억하거나 감소시킬 수 있는 아이템/엘리먼트의 생산 프로세스 특성이라는 용어는 ISO 16949 에서 정의하고 있다. 안전관련 특별특성은 아이템이나 엘리먼트의 개발 단계에서 발생한다. 온도 범위; 만료 날짜; 조입 토크(fastening torque); 제품 공차(production tolerance); 형상(configuration)
안전 타당성 확인 (safety validation)	안전 목표가 충분하고 달성되었는지 조사 및 시험을 기반으로 한 보증 ISO 26262-4 는 타당성 확인(validation)에 적합한 방법들을 제공하고 있다.
준정형 표기법 (semi-formal notation)	구문이 완벽하게 정의되었지만, 의미에 대한 정의는 불완전할 수 있는 기술 기법 SADT(System Analysis and Design Techniques); UML(Unified modeling Language)
준정형 검증 (semi-formal verification)	준정형 표기법에서 제공되는 기술에 기반한 검증 시스템 행위가 모델과 일치하는지 시험하기 위하여 준정형 모델로부터 시험 벡터를 생성하여 사용
서비스 노트 (service note)	아이템에 대해 유지보수 절차를 적용하여 수행할 때 고려되는 안전과 관련된 정보를 문서화 한 것 안전관련 특별특성; 필요한 안전 작동 조건
심각도 (severity)	발생 가능한 위험원 상황에서 한 명 이상에게 끼칠 수 있는 위해 정도를 추정한 것 위험원 분석 및 리스크 평가에서 파라미터 "S"는 위해의 잠재적인 심각도를 나타낸다.
단일점 고장 (single-point failure)	안전 목표를 직접적으로 위반할 수 있는 단일점 결함으로 인한 고장 단일점 결함은 하나의 엘리먼트에 대해 진단 커버리지가 0%인 잔존 고장과 같다. 최소한 한 개의 안전 메커니즘이 어떤 하드웨어 엘리먼트에 대해 정의되었다면(예를 들어 마이크로 컨트롤러에 대한 위치독(watchdog)), 해당 하드웨어 엘리먼트의 결함은 단일점 결함이 아니다.
단일점 결함 (single-point fault)	안전 메커니즘으로 보호되지 않으며 직접적으로 안전 목표의 위반을 야기하는 엘리먼트의 결함 단일점 고장도 참고
소프트웨어 컴포넌트 (software component)	하나 이상의 소프트웨어 단위
소프트웨어 도구 (software tool)	아이템이나 엘리먼트 개발에 사용되는 컴퓨터 프로그램

소프트웨어 단위 (software unit)	단독적으로 시험 대상이 될 수 있는 소프트웨어 아키텍처에서 가장 작은 수준의 소프트웨어 컴포넌트
구문 커버리지 (statement coverage)	소프트웨어 내에서 실행된 구문에 대한 백분율
시스템 (system)	센서, 컨트롤러, 엑츄에이터간에 서로 관련된 엘리먼트의 집합 서로 관련된 센서나 엑츄에이터는 시스템에 포함되거나 시스템 외부장치가 될 수 있다. 시스템 엘리먼트는 또 다른 시스템이 될 수 있다.
시스템적 고장 (system failure)	설계 변경, 제조 프로세스, 작동절차, 문서, 기타 관련인자들을 변경함으로써 제거될 수 있는 고장으로 어떤 원인에 대해 결정적 방식(deterministic way)으로 발현됨
시스템적 결함 (system fault)	프로세스나 설계 수단을 적용하여야만 방지될 수 있는 결함으로서 해당 결함에 대한 고장은 일정한 방식으로 발현됨
기술안전 개념 (technical safety concept)	기술안전 요구사항과 해당 요구사항들은 시스템 설계에 따라 구현하기 위해서 시스템이나 엘리먼트에 할당한 것
기술안전 요구사항 (technical safety requirement)	관련된 기능안전 요구사항의 구현을 위해 파생된 요구사항 파생된 요구사항은 완화(mitigation)을 위한 요구사항을 포함한다.
시험 (testing)	아이템이나 엘리먼트가 명시된 요구사항을 만족하는지 여부를 검증하고, 이상점을 감지하며 그것의 행위에 대해 신뢰할 수 있도록 계획하고 준비하며 작동 또는 시범 작동하는 프로세스
순간 결함 (transient fault)	한 번 생성되고 이후 사라지는 결함 간헐적 결함은 전자파 간섭으로 인해 나타날 수 있으며, 비트 플립을 유도할 수 있다. SEU(Single Event Upset)와 SET(Single Event Transient)와 같은 소프트 오류가 순간 결함이다.
검증 (verification)	단계나 하위단계에서 요구사항에 대해 완전성 및 정확한 명세여부 또는 구현여부를 판단하는 것
검증 검토 (verification review)	개발 활동에 대한 결과가 프로젝트 요구사항이나 기술 요구사항 또는 두 가지 모두를 충족한다는 것을 보장하기 위한 검증 활동 검증 검토의 개별 요구사항은 ISO 26262 의 각 부의 별도 절에 나타나 있다. 검증 검토의 목표는 유즈 케이스와 고장형태와 관련하여, 아이템이나 엘리먼트의 기술적 정확성과 완결성을 확인하는 것이다. 기술검토; 워크쓰루; 인스펙션
워크쓰루 (walk-through)	이상점을 감지하기 위한 작업 산출물의 체계적인 조사 활동 워크쓰루는 검증의 한 방법이다. 워크쓰루는 일반적으로 관련

	아이템이나 엘리먼트의 작동은 포함하지 않는다는 점에서 시험과 다르다. 이상점이 검출되면, 일반적으로 재작업을 하고 재 작업된 작업 산출물은 워크쓰루를 통해 검증한다. 워크쓰루 동안 개발자는 한 명 이상의 평가자에게 작업 산출물을 단계별로 설명한다. 이것을 하는 목적은 작업 산출물에 대해 공통적으로 이해하고, 작업 산출물 내에 있는 이상점을 식별하는 것이다. 인스펙션과 워크쓰루 모두 동료(peer)검토 타입으로 워크쓰루는 인스펙션 보다 덜 엄격한 동료 검토 형식이다.