2011년 09월 29일 11시 16분 갱신

자동차 분야의 기능 안전 규격 “ISO26262” 라는 것은 무엇인가? (2):

ISO26262 Part.6 소프트웨어 개발

자동차 분야 전용의 기능 안전 규격 “ISO26262”. 이번 회는, 국내에서도 실시되고 있는, 비교적 프로세스 개선에 착수하기 쉬운 ISO26262의 “소프트웨어 개발”에 대해서 자세하게 해설한다.

글: 河野喜一(NEC 컨설팅사업부) | 출처: MONOist

번역: 이채원 • 카이젠컨설팅

전 회 “다시 〈ISO26262〉의 전체상을 파악해두자”의 예고대로, 이번 회는 “ISO26262의 소프트웨어 개발”에 대해서 소개합니다.

자동차 기능 안전 규격 “ISO26262”의 소프트웨어 개발 개요

소프트웨어의 안전이란 무엇인가?

──도대체, “소프트웨어의 안전” 이란 무엇일까요?

개발자에게 있어서는, “소프트웨어의 안전이라고 하면, ‘품질’이다!!” 라고 오해하시는 분도 있습니다만, 자동차 기능 안전 규격에서의 소프트웨어 안전이란, 품질이라기 보다는, “오작동 (입력)에 따른 오동작 (출력)을 시키지 않는 fail-safe나, 제어 중에 고장이 나더라도 서서히 제어를 정지한다 라는 fail-soft의 개념”으로 이해할 수 있습니다.

기능 안전 규격에서는, 안전 레벨에 따른 fail-safe나 fail-soft의 구체적인 기법을 적용하는 것이 요구됩니다.

ISO26262의 소프트웨어 개발 개요

“ISO26262”의 소프트웨어 개발에서는, 시스템 설계서의 내용을 소프트웨어 요구사항으로 분리하고, 소프트웨어 아키텍처를 검토한 후 소프트웨어 단위 설계&코딩을 실시합니다.

개발 후에는 단위 Test, 통합 Test를 거쳐 소프트웨어 안전 요구 검증을 실시하는 것으로, 소프트웨어에 결함이 포함되지 않은 것을 보증합니다.

각 공정에서는, 안전 레벨에 따라서 적용해야 하는 기법이 정의되어 있어, 안전 레벨이 높으면 높을수록 여러 가지 종류의 fail-safe나 fail-soft의 구축이 요구되어, 형식 기법/형식 검증 (주 1) 이라는 최첨단의 기술도 요구됩니다.

※주1 : 본 연재에서는, “Formal Notation”을 형식 기법이라고 부르고, “Formal Verification”을 형식 검증이라고 번역하고 있습니다.

하지만, ISO26262에서 정의되고 있는 내용은 최소한의 필요 조건이므로, 실제 개발에서는 품질 특성이나 새로운 안전성을 고려한 설계가 추천됩니다.

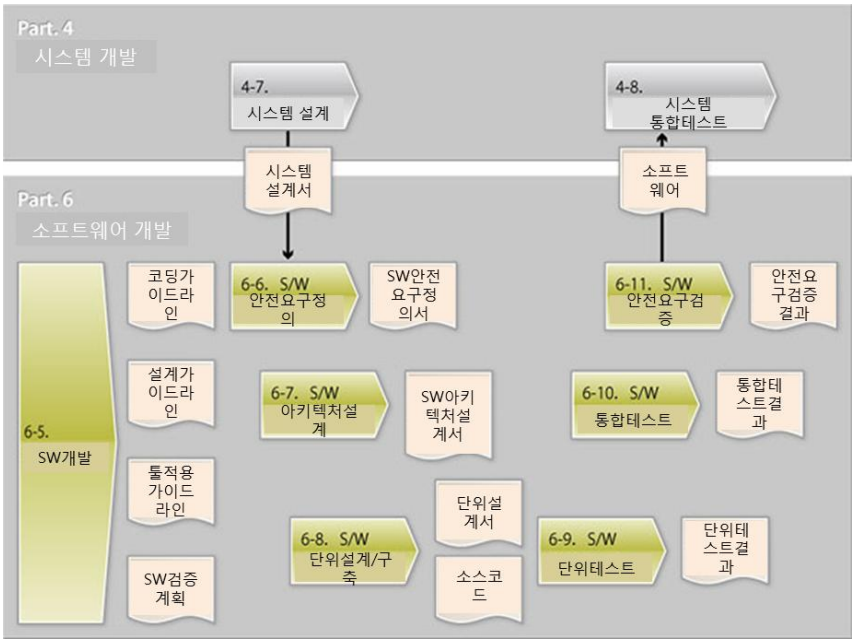

그림 1 소프트웨어 개발 프로세스

사전에 준비할 것

ISO26262의 소프트웨어 레벨에서는, 사전 준비로써 개발 언어의 평가 기준을 수립하고, 개발 언어를 선택합니다. 예를 들어, 코딩 표준으로 유명한 MISRA-C와 같이 애매성이나 오해를 일으키기 쉬운 C언어 보다는, Ada언어를 이용했던 것을 들 수 있습니다.

이와 같이 개발 툴을 선택&툴 인정을 실행하고, 다음의 5가지 문서의 개발 순서를 작성합니다.

- 프로세스 정의서
- 개발 기법 적용 가이드라인

- 설계 가이드라인
- 개발(코딩) 표준
- 개발 툴 사용자 매뉴얼

상기의 준비가 끝나고, 소프트웨어 개발을 시작한 후에는, 개발 규모나 개발 난이도에 따라 소프트웨어 개발 프로세스를 조정 (통합/분해)하고, 개발 계획을 수립합니다.

안전 요구사항 정의

일본의 자동차 업계에서는, 많은 기업들이 소프트웨어 요구사항 정의를 실시하지 않고 시스템 개발의 성과물인 시스템 설계서부터 직접 소프트웨어의 개발을 수행해 버리는 경향이 있습니다. 하지만, 시스템 설계서의 내용은 어디까지나 시스템 레벨이며 소프트웨어 수준의 요구사항은 기재되어 있지 않습니다.

이 때문에, 시스템 설계서를 바탕으로 소프트웨어의 안전 요구사항을 정의해야 합니다. 시스템 설계서의 동작 시나리오나 비기능 요구를, 연산 처리를 고속화 하기 위한 변수의 scaling, 컴퓨터의 반올림 오차 등을 고려해서 소프트웨어 시점에서 재정의 하거나 OS나 middleware 등의 요구사항을 정의합니다. ISO26262에서는, 결함을 검출해서 제어하고, 결과를 통보하는 활동이 매우 중요합니다.

또, 안전 요구사항 정의에서는 안전 레벨에 따른 표기법이 지정되어 있어, SysML/UML 이라는 준형식 기법이나 텍스트 베이스의 VDM-SL (ISO/IEC 13871-1), 모델 베이스의 SDL (ITU-T Z.100), Lustre 이란 형식 기법에 의한 기술이 필요합니다. 자동차 업계에서는, CAN/Flex Ray, OSEK 등의 표준 규격서가 모델 베이스의 형식 기법 SDL에서 기술되고 있습니다.

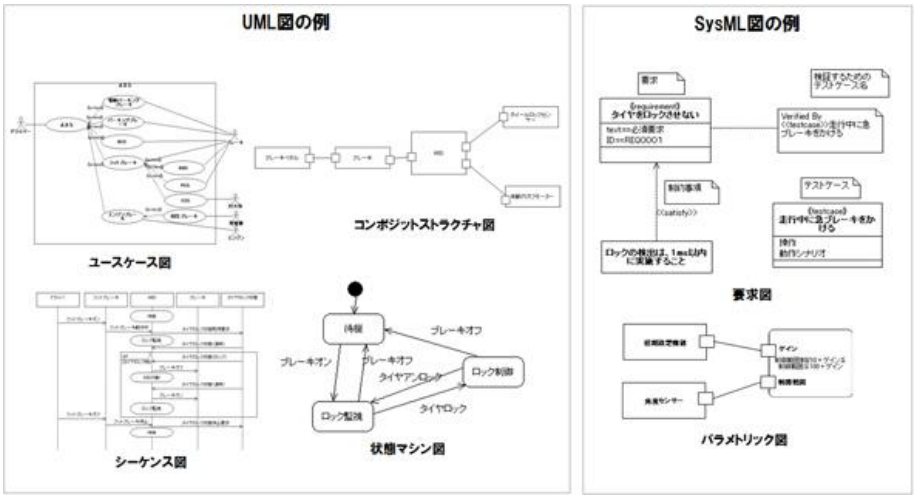

그림2 UML, SysML 그림의 예

안전 요구사항을 정의한 후에는 안전 레벨에 따른 리뷰나 시뮬레이션을 실시하여 요구사항의 결함이나 누락, 누출을 검출합니다. 이는 결함이 발생하지 않는 것을 공식적으로 증명하는 것입니다만, 안전 레벨에 따라서는 수학적으로 증명하는 기법인 형식 검증이 추가로 요구되는 경우도 있습니다.

아키텍처 설계

실제 개발은 기존의 아키텍처를 수정 또는 재사용을 하지, 그다지 아키텍처 설계를 실시하지 않는 일도 있습니다만 기능의 추가/수정이 쉬운 아키텍처를 검토해두지 않으면 다른 사람이 수정할 수 없는 블랙 박스가 되어버립니다.

ISO26262의 아키텍처 설계에서는, 모든 소프트웨어 요구사항을 컴포넌트라는 단위에 할당해 기능 통합이나 기능 분할을 실시해서 소프트웨어의 아키텍처를 설계하고 컴포넌트마다 외부 I/F, 동작 시나리오, 동작 상태를 정의합니다. 특히 ISO26262에서는 통신 상대로부터 응답이 없어졌을 경우나 이상 메시지가 보내져 왔을 경우의 처리도 검토할 필요가 있습니다.

컴포넌트 정의 후에는 컴포넌트를 신규 개발할 것인지, 기존의 컴포넌트를 재사용할 것인지, 시판의 컴포넌트 (OS, middleware 등)을 구입할 것인지 라는 검토도 행합니다.

아키텍처 설계에서도 안전 레벨에 따라 표기법이 지정되어 있어, SysML/UML/MATLAB/Simulink 라는 준형식 기법이나 VDM-SL, SDL, Lustre 라는 형식 기법에 의한 기술이 필요하게 되는 일도 있습니다.

이 아키텍처 설계 후에는, 아키텍처 내의 기능을 어떻게 CPU에 배치할까의 검토를 수행하고 복수 기능을 하나의 ECU에 통합해 가는 작업을 실시합니다.

일본에서는 먼저 ECU를 정해서 소프트웨어를 싣고 가는 것이 많습니다만, 유럽에서는 소프트웨어의 메모리나 처리 속도의 예측을 행한 후에 ECU에 배분합니다. 이 기법을 도입하게 되면, ECU에 소프트웨어를 구축한 후에 “처리 속도가 요구에 못 미친다”, “메모리가 부족하다” 등 이러한 문제가 발생하는 일을 방지할 수 있습니다.

아키텍처 설계를 실시한 후에는, 안전 레벨에 따른 리뷰나 시뮬레이션을 실시하고 기능간 I/F에 잘못이나 누락/누출이 없는 것도 확인합니다. 이는 결함이 발생하지 않는 것을 공식적으로 증명하는 것입니다만, 안전 레벨에 따라서는 수학적으로 증명하는 기법인 형식 검증이 추가로 요구되는 경우도 있습니다.

단위 설계

실제 개발에서는, 단위 설계를 하지 않는 일은 없다고 생각합니다만, 경우에 따라서는 단위 설계서가 없이 소스코드만 있거나 소스코드만 수정되어 있고 전체적으로 단위 설계서가 갱신되지 않는 것도 있습니다.

단위 설계에서는, 모든 기능의 상세 설계를 실행하고 외부 I/F, 동작 시나리오, 동작 상태, 처리를 기재합니다.

여기서는 안전 레벨에 따라 보수성을 고려, 코딩 표준, 표기 방법이 정의되고 있습니다. 표기 방법에서는, SysML/UML/MATLAB/Simulink 라는 준형식 기법이나 VDM-SL, SDL, Lustre 이란 형식 기법에 따른 기술이 필요하게 되는 경우도 있습니다.

SDL같은 모델 베이스 개발의 경우, 개발 환경에 따라서는 모델링 한 후에 시뮬레이션이나 자동 코드 생성을 행하는 것도 가능합니다.

단위 설계를 한 후에는, 안전 레벨에 따라 리뷰, 시뮬레이션, 정적 해석을 실시하고 기능에 이상이 발생하지 않는 것, Target 환경에 적합한 것, 코딩 표준에 따르고 있는 것을 확인합니다. 이는 결함이 발생하지 않는 것을 공식적으로 증명하는 것입니다만, 안전 레벨에 따라서는 수학적으로 증명하는 기법인 형식 검증이 추가로 요구되는 경우도 있습니다.

단위 테스트

단위 테스트에서는, 맨 처음 테스트 계획을 수립합니다.

이것은, 개발이 지연되면 테스트를 생략해 버리는 경향이 있기 때문에, 확실히 테스트를 실시하는 기간을 확보하는 것이 목적입니다.

단위 테스트에서는 함수 단위에서 테스트를 실시합니다만, 요구대로 동작하는 것은 물론 결함을 발생시키더라도 이상 동작하지 않는 것도 확인하고 있습니다. 동시에 안전 레벨에 따라 함수 단위의 테스트 Coverage (C0, C1, MC/DC)를 측정하는 것으로 함수의 분기 Pass 전부를 테스트해서, 이상 동작이 일어나지 않는 것을 검증합니다. 이 테스트 Coverage 기준은, 항공업계의 DO-178B로부터 유용되고 있기 때문에, 항공 업계의 테스트 툴로 대응하고 있는 경우가 많아지고 있습니다.

또, 일본에서는 테스트 Coverage 결과를 남기지 않는 경우가 많습니다만, ISO26262에서는 테스트 Coverage 결과도 포함된 테스트 결과를 모두 남겨야 합니다.

단위 테스트~통합 테스트에서는, 가능한 한 대상 환경에 가까운 환경으로 테스트를 실시하고, 소프트웨어 인 더 루프 테스트 (SILS), 프로세서 인 더 루프 테스트 (PILS), 모델 인 더 루프 테스트(MILS), 하드웨어 인 더 루프 테스트(HILS) 라는 테스트 환경을 이용하여야 합니다.

환경		대응테스트	예
PC	시뮬레이션 / 수동 시뮬레이션	SILS	자동 생성 코드 / 수동 작성 코드에서의 시뮬레이션
		MILS	모델링 툴에서의 시뮬레이션

대상 환경	외부 테스트 환경(JTAG, TCP/IP)	PILS	JTAG, TCP/IP, USB 등에 의한 테스트
	H/W 테스트 환경(자동 테스트)	HILS	HILS 접속에 의한 테스트

그림 3 SILS/PILS/MILS/HILS의 대응

통합 테스트

통합 테스트도, 통합 테스트 계획을 수립 합니다만, 아키텍처 설계의 계층마다 차례로 함수를 통합해가고 있기 때문에, 아키텍처 설계의 계층 수에 의존한 계획을 수립하는 것에 주의가 필요합니다.

통합 테스트에서는, 복수의 함수를 통합해서 테스트하는 것을 반복 실시합니다.

통합 테스트에서는, 함수를 통합한 단위로 테스트를 실시합니다만, 요구사항대로 동작하는 것은 물론 결함을 발생시켜도 이상 동작하지 않는 것도 확인하고 있습니다. 동시에 안전 레벨에 따라 함수가 실행되었는지 아닌지의 테스트 Coverage (S0, S1)을 측정하는 것으로, 함수 Call의 Pass 모두를 테스트해서, 이상 동작이 일어나지 않는 것을 보증합니다.

소프트웨어 안전 요구사항 검증

소프트웨어 안전 요구사항 검증에서도, 맨 처음에 검증 계획서를 작성합니다.

여기서는, 여러 가지 환경에서 안전 요구사항을 충족시키는 것을 검증합니다. 이 때문에, 환경이나 요구사항 케이스 수에 따라, 계획을 수립할 필요가 있습니다.

안전 레벨에 의해서 네트워크 시뮬레이터, HILS, 실제 차량에 의한 검증이 요구됩니다.

Automotive SPICE만으로는, ISO26262 소프트웨어 개발 요건을 준수할 수 없다.

Automotive SPICE의 인증을 취득하면, 충분히 ISO26262의 소프트웨어 개발의 인증도 취득할 수 있다고 오해하고 있는 분이 있습니다만 Automotive SPICE에서는, 프레임워크만이 정의되어 있어 구체적인 기법이나 안전에 관한 요구사항이 포함되어있지 않습니다.

이 때문에, Automotive SPICE의 인증은 ISO26262의 소프트웨어 개발의 기반은 됩니다만, 그대로는 ISO26262의 소프트웨어 개발의 인증을 취득할 수 없는 것을 주의해 주십시오.

2011년 09월 29일 11시 16분 갱신

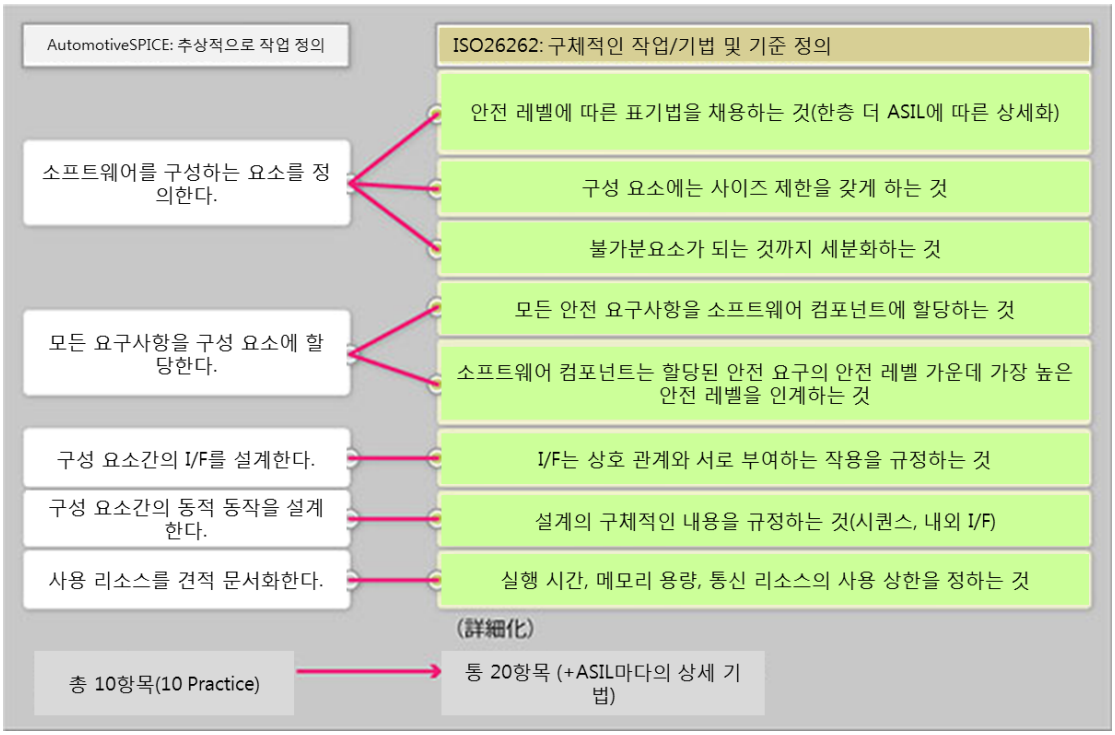

그림 4 Automotive SPICE ENG.5와 ISO26262 Part.6-7의 대응 예

다음 회는, 시스템, 하드웨어, 소프트웨어에 관계된 ISO26262 Part.8 지원(관리 프로세스)에 대해서 말씀드릴 예정입니다. 기대해 주세요! (다음 회에 계속)

원문 | http://monoist.atmarkit.co.jp/mn/articles/1109/29/news003.html#l_ay_iso2626202_fig02.jpg

http://monoist.atmarkit.co.jp/mn/articles/1109/29/news003_2.html