MathWorks AUTOMOTIVE CONFERENCE 2018

# 정형 기법을 활용한 AUTOSAR SWC의 구현 확인 및 정적 분석

*Develop high quality embedded software*

이 영준
Principal Application Engineer

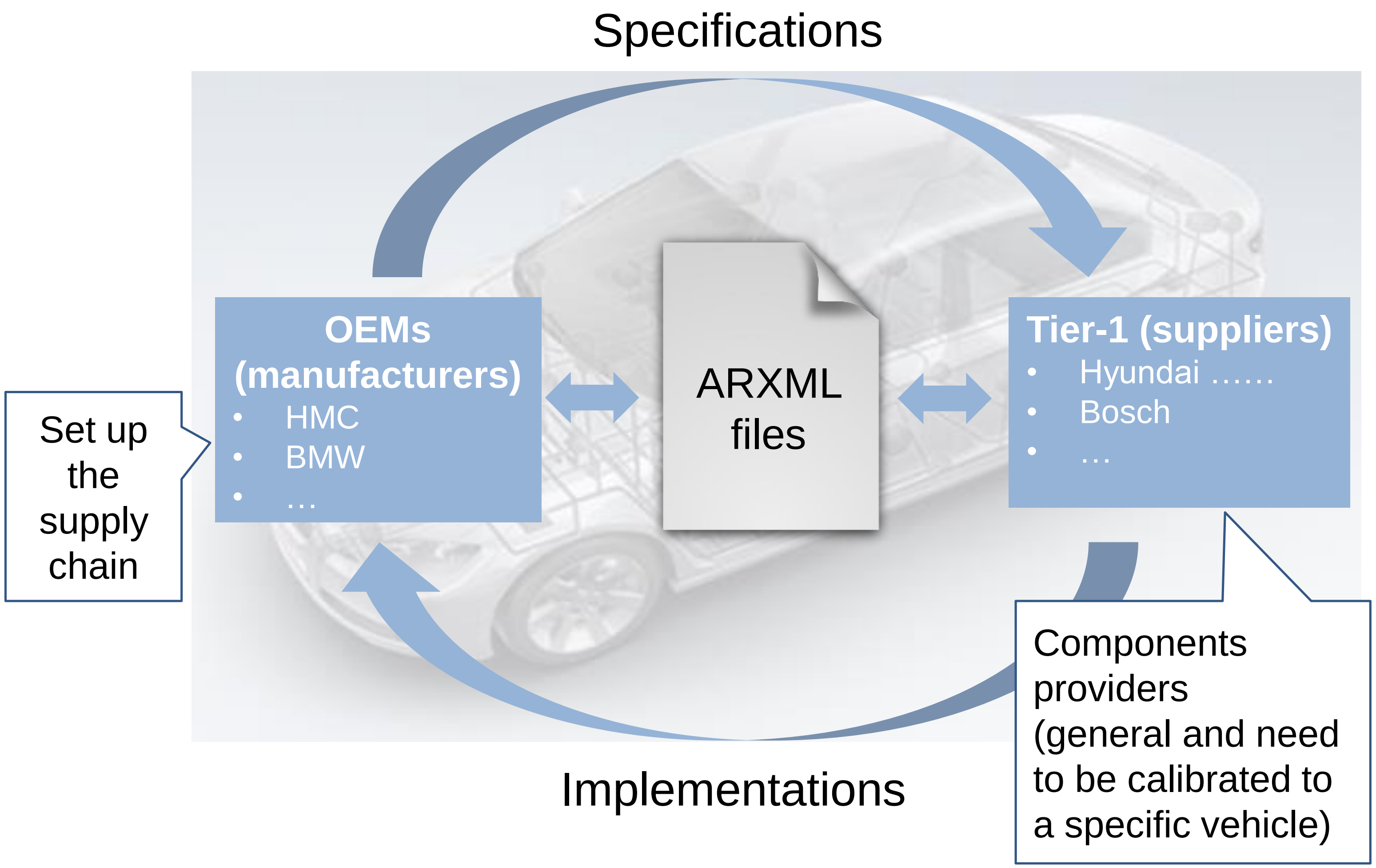
# Agendas

- **Unit-proving of AUTOSAR Component and Runtime error**
- **Secure Coding Standard and Polyspace**
  - **MISRA-C:2012 Amendment 1**
  - **ISO 17961**
  - **CERT-C/C++**
  - **CWE**

# Unit-Proving of AUTOSAR Component And Runtime Error

# What is AUTOSAR?

- The Automotive industry and its challenges



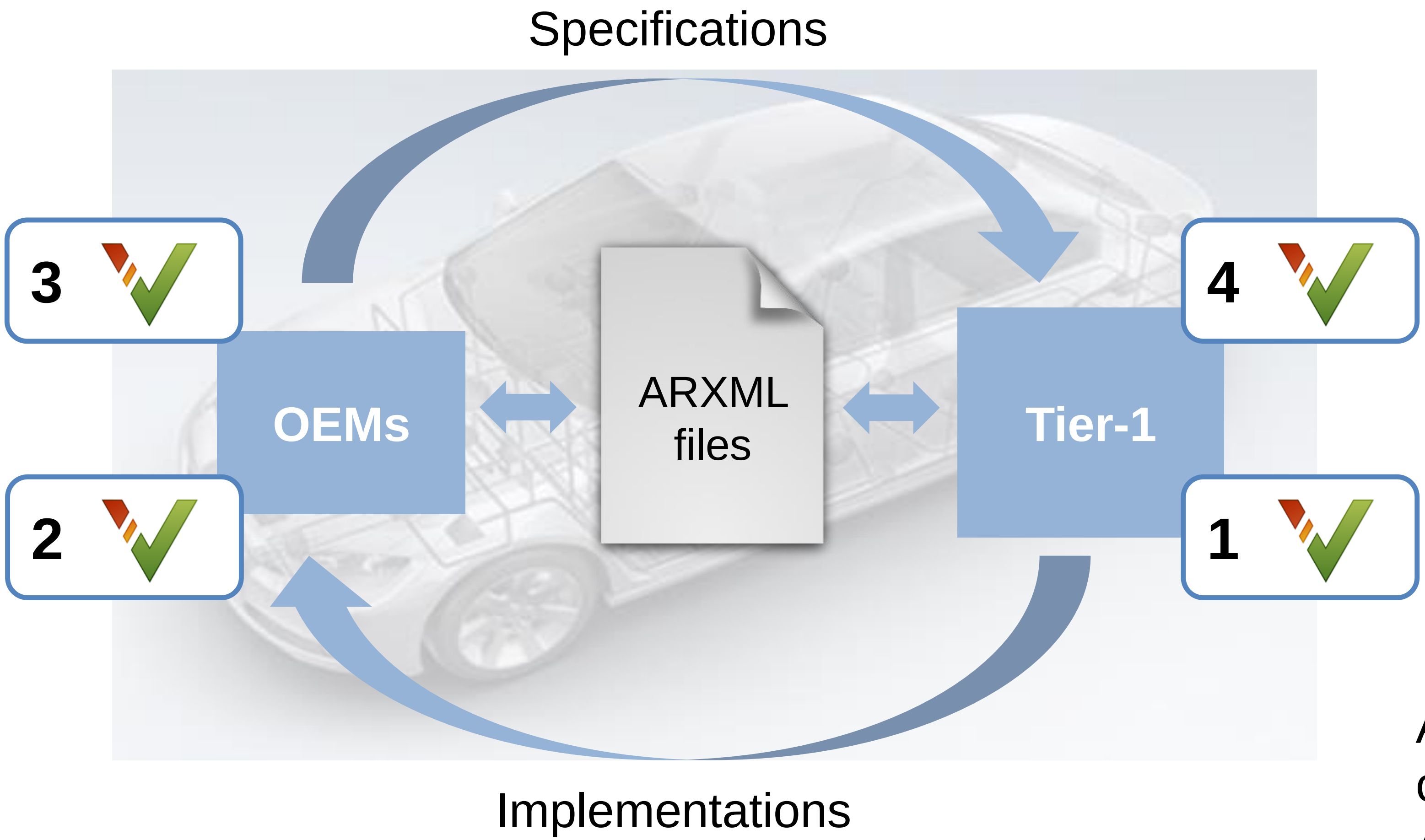

OEMs objectives:
- Integration from different suppliers
- Need confidence in the supplier's code

Supplier's challenge:
- Time-to-market
- Code size
- Pressure from OEMs

AUTOSAR solves by providing a software architecture and common specifications (ARXML files)

Need for validation of AUTOSAR components among actors

# How Polyspace for AUTOSAR can help?




1) check for run-time errors and mismatch in the ARXML specifications

2) Check if implementation follow specifications

3) assess impact of changes in the specifications

4) check implementation against specifications updates

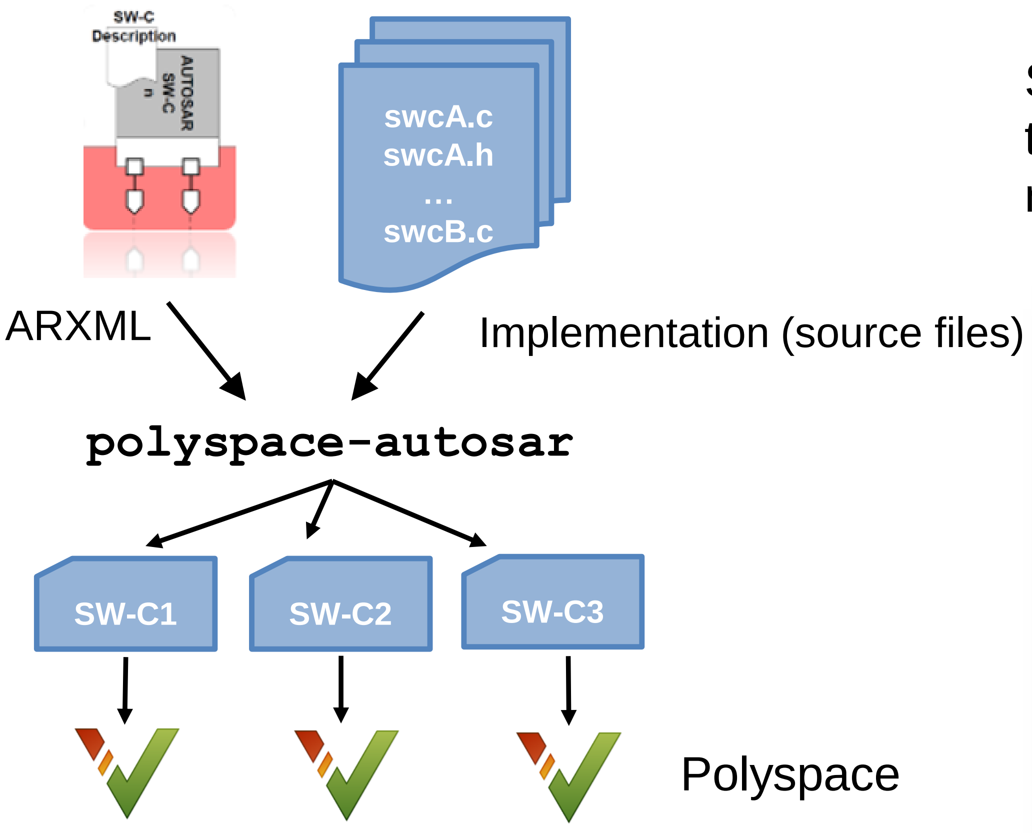
ARXML files are used to communicate, Polyspace for AUTOSAR is used to prove robustness and compliance

# Polyspace for AUTOSAR features

## Automatic split by component

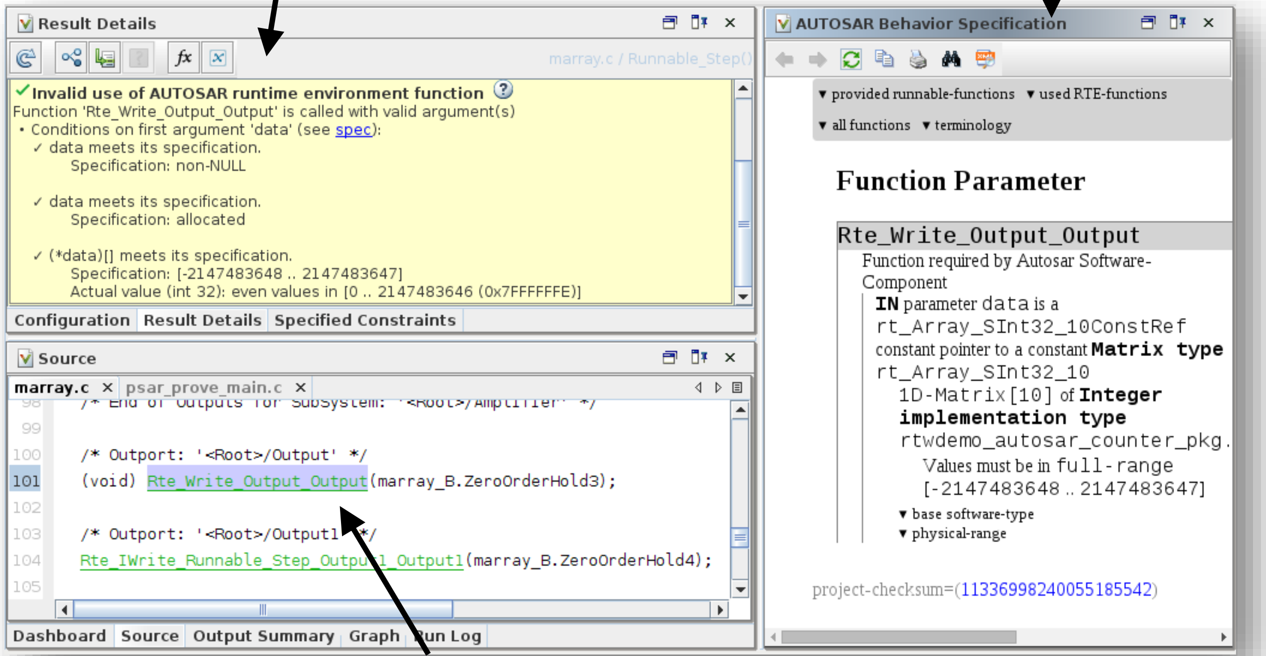
## Sound analysis

## Back to specifications




Sound analysis plus checks to prove that the code matches the specification

New view to detail the AUTOSAR specification



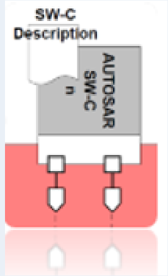

## Automatic launching on each component

## Prove specs matching

# Polyspace for AUTOSAR workflow

## Specifications

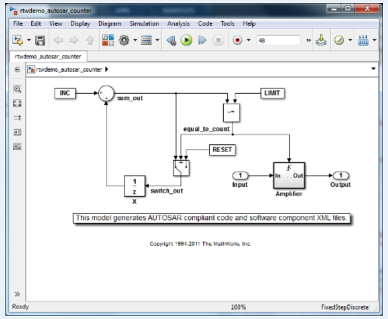
ARXML files




## Implementation

swcA.c
swcA.h
...
swcB.c

Simulink model



Split the code in components as defined in ARXML

**Polyspace for AUTOSAR**

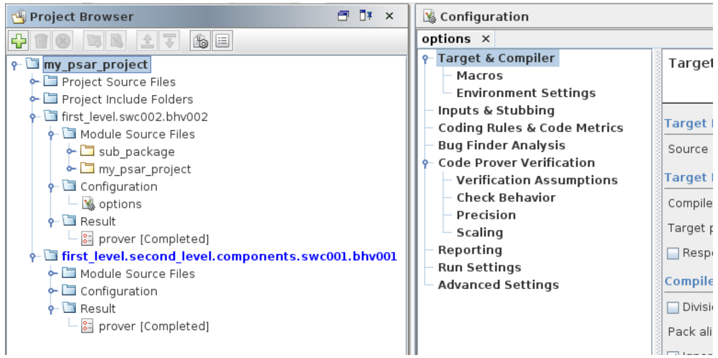
SW-C1

SW-C2

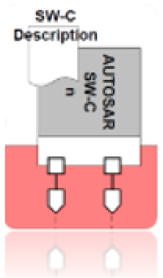
SW-C3

Perform a separate unit analysis of each component with Polyspace

- ✓ Free of run-time errors
- ✓ Checks that code of runnable respects its output specification
- ✓ Checks that code of runnable calls Rte functions in respect of their specification



# Polyspace and AUTOSAR

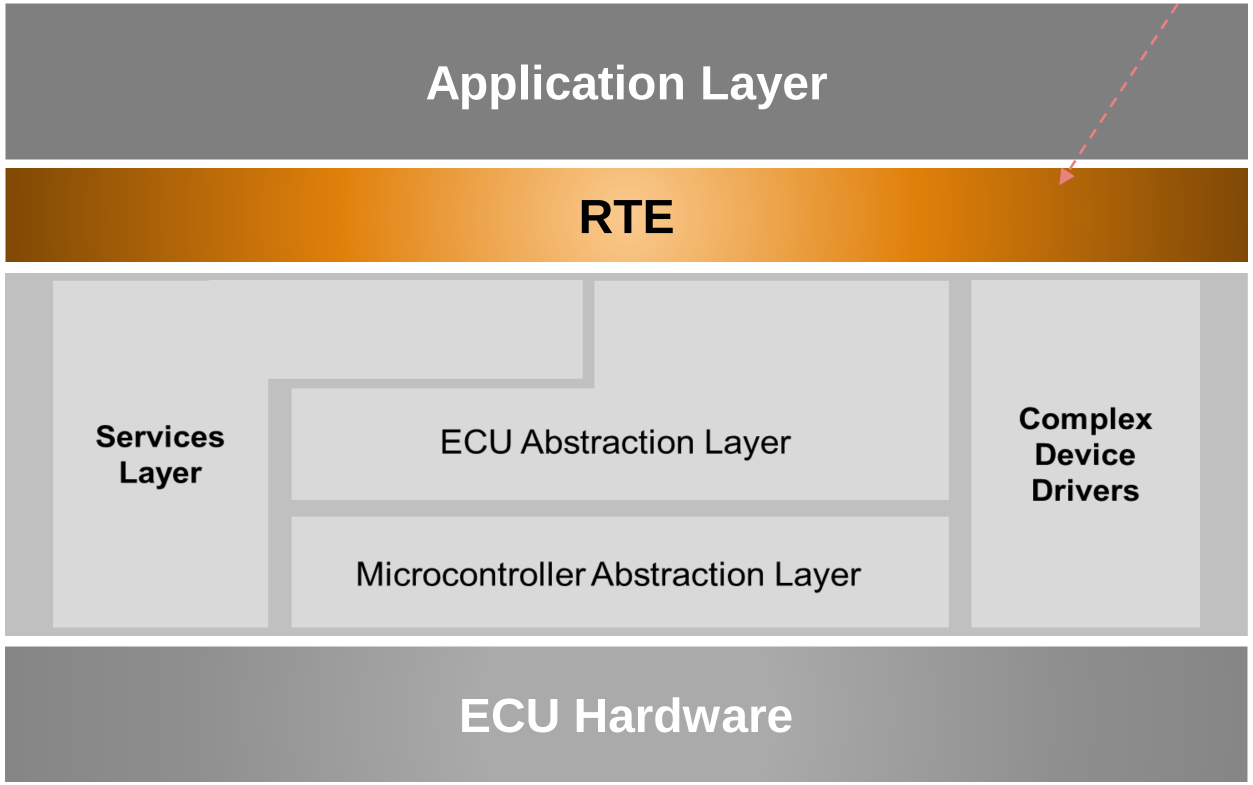



ARXML

*ARXML provides specification of Application Layer and link with RTE*

*Polyspace verifies the match between code and ARXML*

AUTOSAR architecture

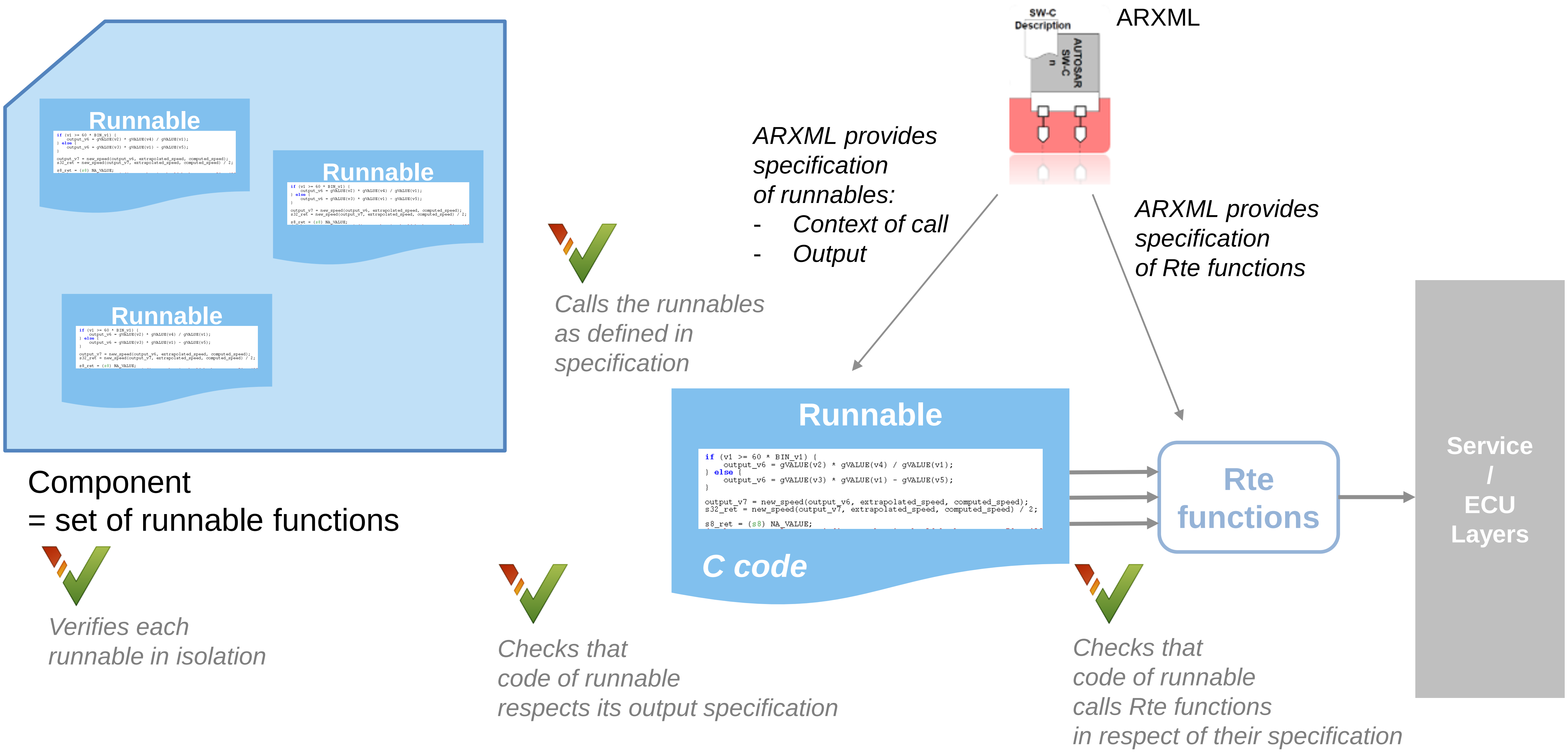



*Polyspace verifies the Application Layer*

*Polyspace stubs the RTE Layer*
*RTE Layer not verified by “Polyspace for AUTOSAR”*
*Polyspace can verify RTE*

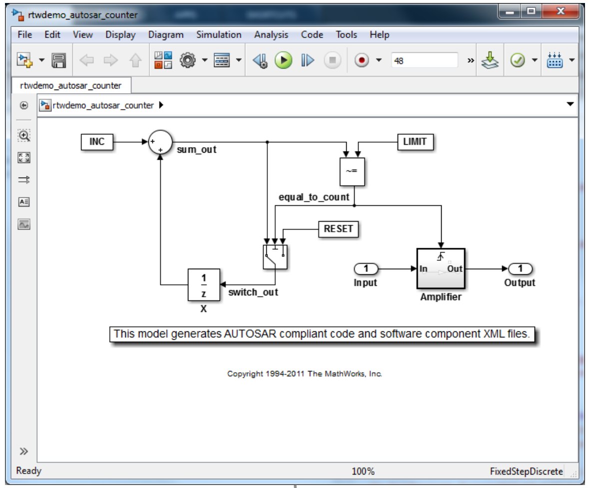
*Not verified by “Polyspace for AUTOSAR”*
*Polyspace may verify these*

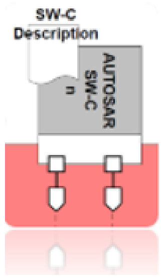
# Unit verification of an AUTOSAR software component



# Unit verification of an AUTOSAR software component

Simulink model




ARXML

*ARXML provides specification of runnables:*
- *Context of call*
- *Output*

*ARXML provides specification of Rte functions*

*Calls the runnables as defined in specification*

**Runnable**

```c
if (v1 >= 60 * BIN_v1) {
    output_v6 = gVALUE(v2) * gVALUE(v4) / gVALUE(v1);
} else {
    output_v6 = gVALUE(v3) * gVALUE(v1) - gVALUE(v5);
}

output_v7 = new_speed(output_v6, extrapolated_speed, computed_speed);
s32_ret = new_speed(output_v7, extrapolated_speed, computed_speed) / 2;

s8_ret = (s8) NA_VALUE;
```

***C code***

**Rte functions**

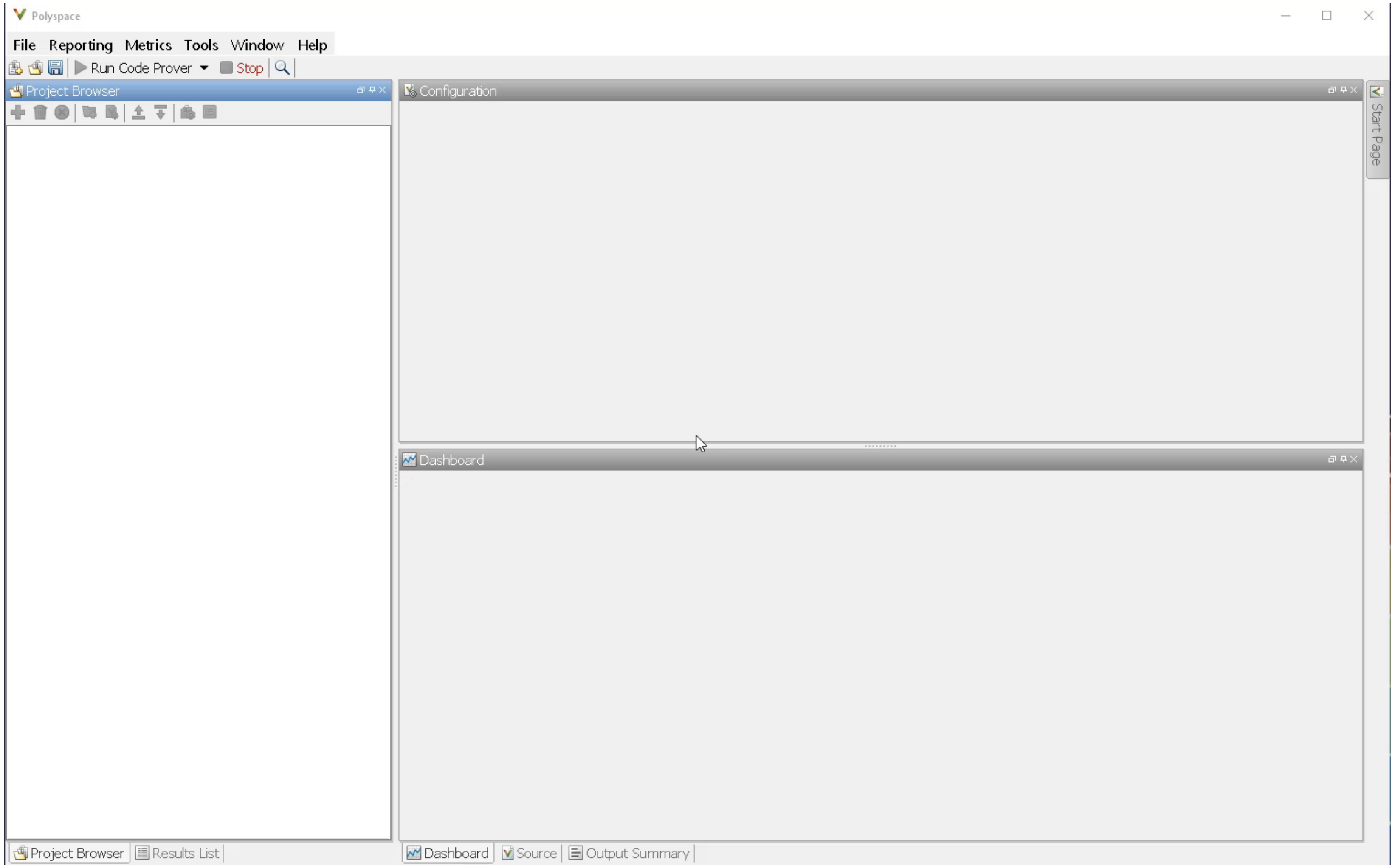
**Service / ECU Layers**

*Checks that code of runnable respects its output specification*

*Checks that code of runnable calls Rte functions in respect of their specification*

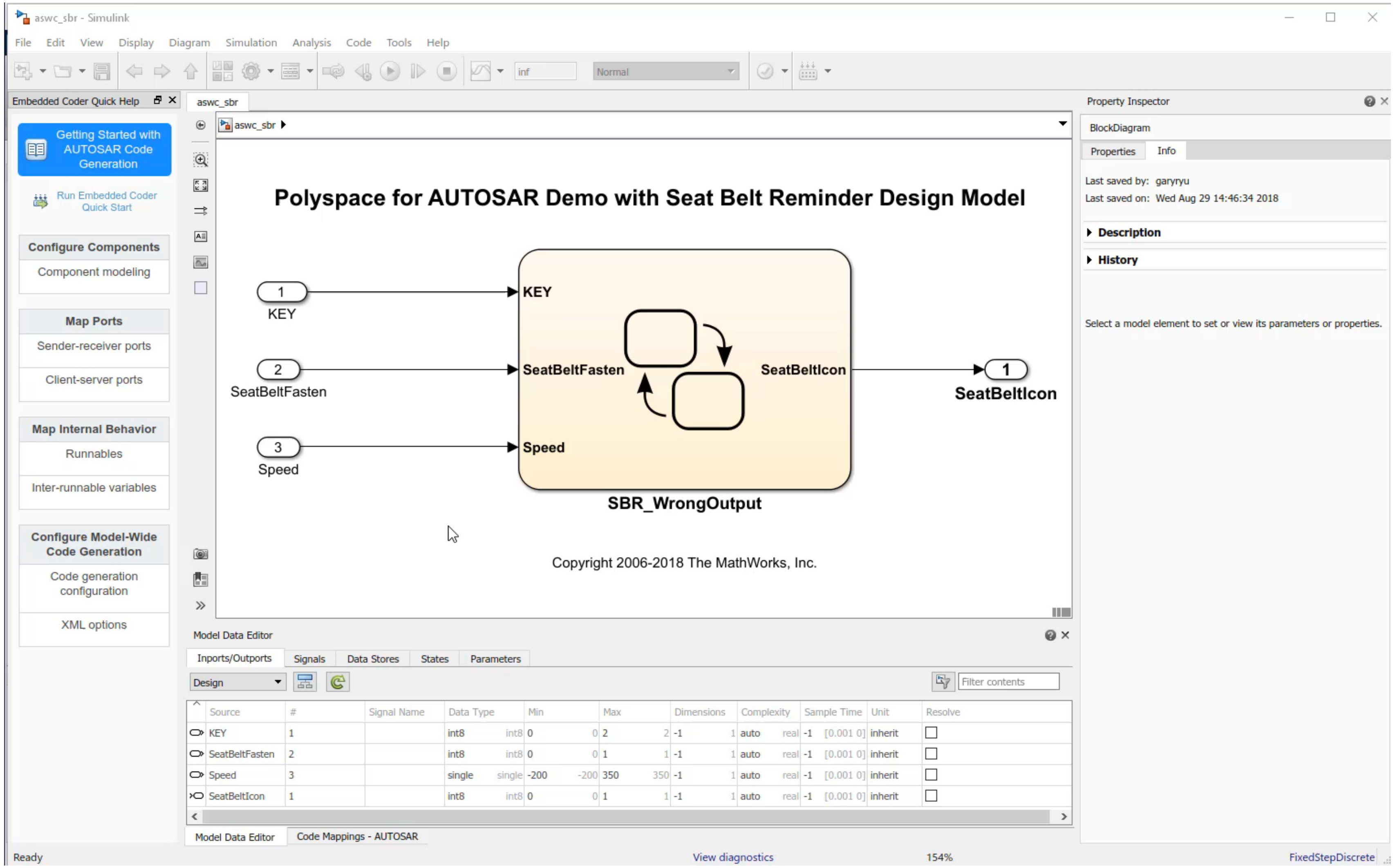
# Hand-Written Code based on ARXML

## *Polyspace for AUTOSAR SWC*



# Generated Code From Simulink Model based on ARXML

## *Polyspace for AUTOSAR SWC*



# Workflow Benefits

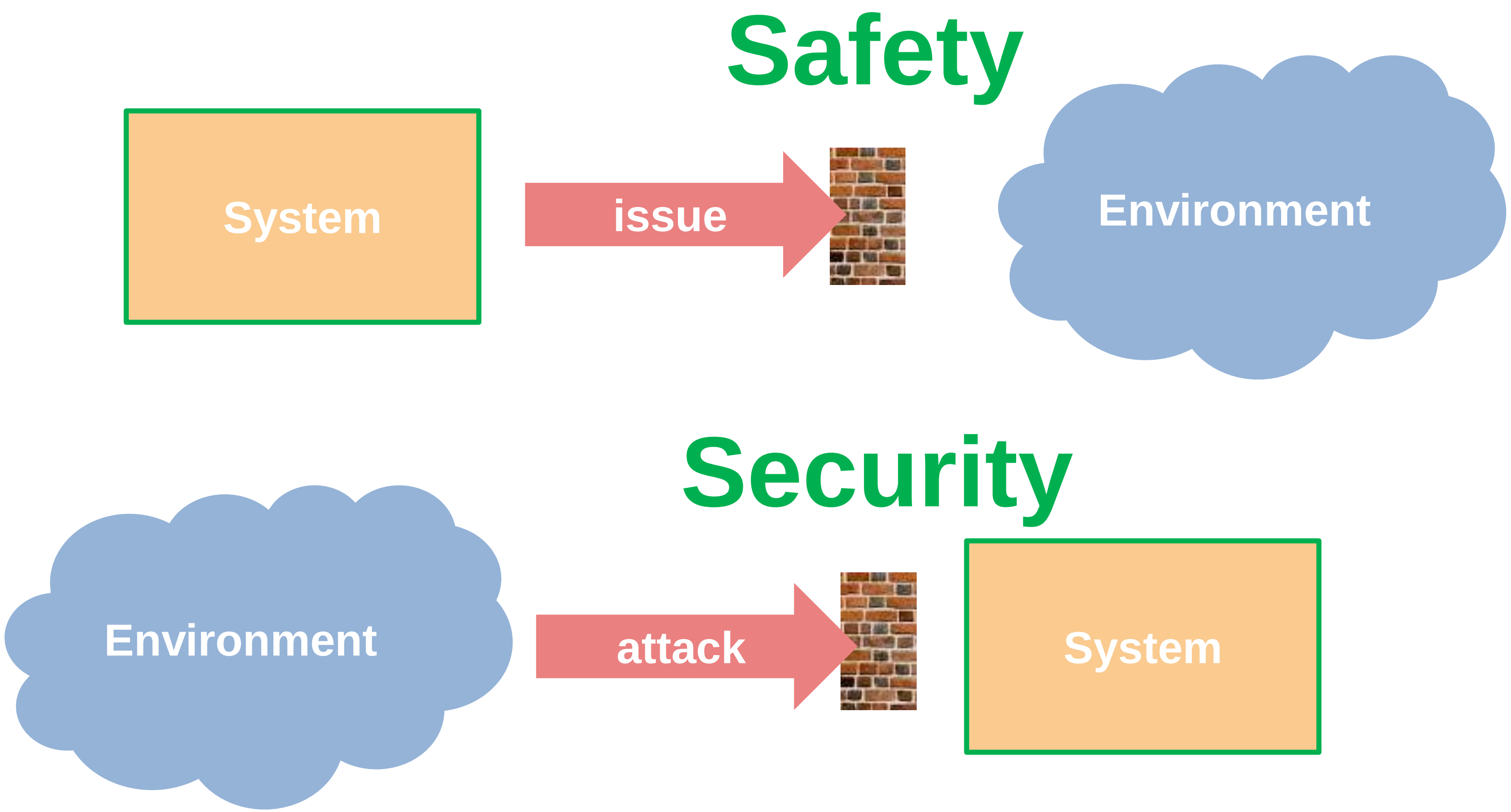
- Provide automatically the best configuration for Polyspace

- Detect inconsistencies between AUTOSAR specifications and code implementation

- Unit verification of AUTOSAR software components with Polyspace
  - ✓ Sound analysis: proves that code respects the specification
  - ✓ Static analysis: considers all potential cases

# Secure Coding and Polyspace

# Safety vs. Security




**Note:** Security issues may cause safety issues

SAE J3061

# Cybersecurity – Industry Activities & Standards

**SAE – Vehicle Cybersecurity Systems Engineering Committee**

- SAE J3061 - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems
- SAE J3101 - Requirements for Hardware-Protected Security for Ground Vehicle Applications (WIP)
- SAE "Cybersecurity Assurance Testing Task Force" (TEVEES18A1)

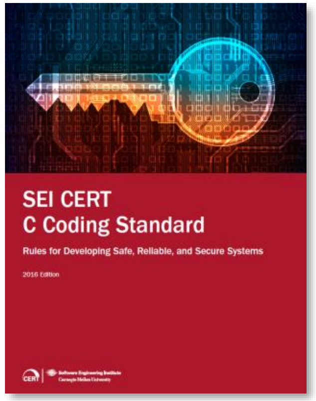
**Coding standards & practices that we observe at automotive customers**

- MISRA-C:2012 Amendment 1
- ISO/IEC TS 17961 – C Secure Coding Rules
- CERT-C / CERT-C++
- CWE – Common Weakness Enumeration

# ISO/IEC TS 17961 Compared with Other Standards

| Coding Standard | C Standard | Security Standard | Safety Standard | International Standard | Whole Language |
|---|---|---|---|---|---|
| CWE | None/all | Yes | No | No | N/A |
| MISRA C:2004 | C89 | No | Yes | No | No |
| MISRA C:2012 | C99 | No | Yes | No | No |
| CERT C99 | C99 | Yes | No | No | Yes |
| CERT C11 | C11 | Yes | Yes | No | Yes |
| ISO/IEC TS 17961 | C11 | Yes | No | Yes | Yes |

Table is based on the book:



# SEI CERT C Coding Standard

- This coding standard consists of ***rules*** and ***recommendations***, collectively referred to as *guidelines*.

- **Rules** are meant to provide normative requirements for code, whereas

- **Recommendations** are meant to provide guidance that, when followed, should improve the safety, reliability, and security of software systems.

# CERT-C Coverage with Polyspace

- You can map Polyspace results to CERT C rules and recommendations
- Using Polyspace results, you can address **103** CERT C rules (**90%**) and **95** CERT-C recommendations (50%)
  - The CERT C website, under continuous development, lists 118 rules and 188 recommendations (Count based on The CERT C++ Coding Standard document, 2016 Edition)

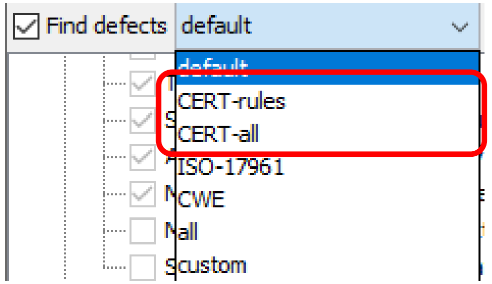
SEI CERT C Coding Standard

# CERT-C++ coverage with Polyspace

- You can map Polyspace results to CERT C++ rules
- Using the Polyspace results, you can address **34** CERT C++ rules (40%) and **79** CERT C rules that also apply to C++ (**99%**)

  – The CERT C++ website, under continuous development, lists 163 rules including 80 CERT C rules that also apply to C++ (based on count in April 2018 in CERT-C++ web site)

  ✓ Two new arguments for option `-checkers` in C++ mode (-lang CPP): `CERT-rules` (only CERT-C++ rules) and `CERT-all` (it includes also CERT-C rules that apply)



# Completeness And Soundness

*From ISO 17961*

- False Negatives
  - Failure to report a real flaw in the code is usually regarded as the most serious analysis error, as it may leave the user with a false sense of security.
- False Positives
  - The tool reports a flaw when one does not exist.

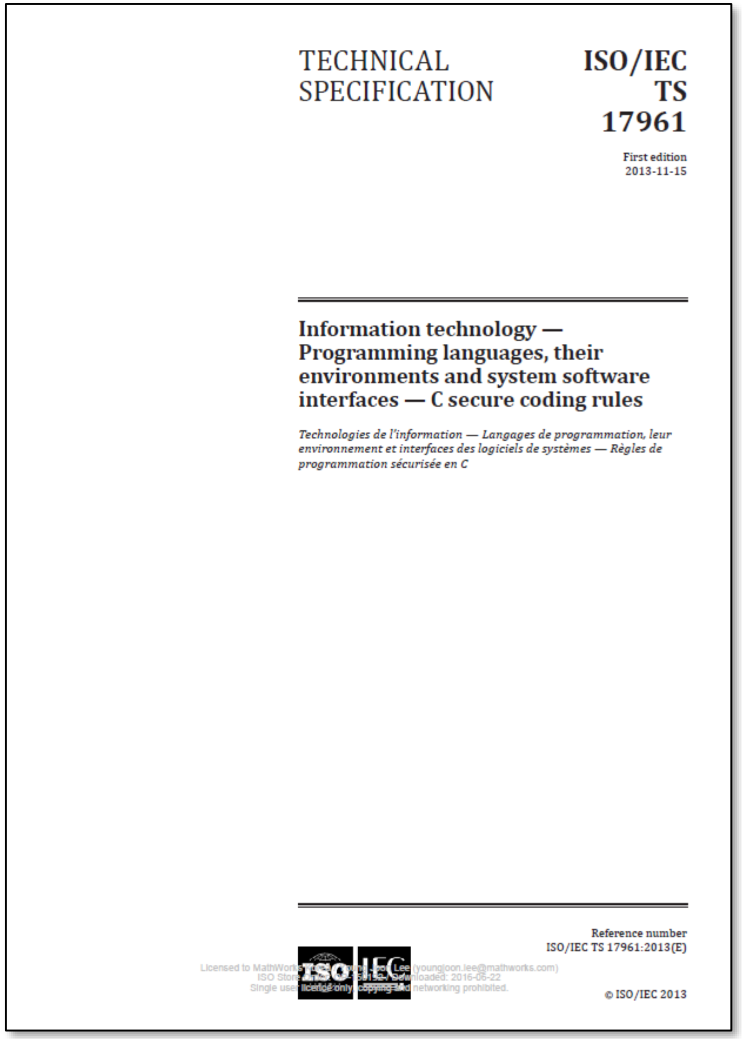
**Table 1 — Completeness and soundness**

| | | False positives | |
|---|---|---|---|
| | | Y | N |
| False negatives | N | Sound with false positives | Complete and sound |
| | Y | Unsound with false positives | Complete and unsound |

Polyspace Code Prover → Sound with false positives

Polyspace Bug Finder → Unsound with false positives

# ISO/IEC TS 17961
# C secure coding rules


TECHNICAL SPECIFICATION

ISO/IEC TS 17961

First edition
2013-11-15

**Information technology — Programming languages, their environments and system software interfaces — C secure coding rules**

*Technologies de l'information — Langages de programmation, leur environnement et interfaces des logiciels de systèmes — Règles de programmation sécurisée en C*

Reference number
ISO/IEC TS 17961:2013(E)

# ISO 17961 - C Secure Coding Rules

- The purpose of this Technical Specification is to specify analyzable secure coding rules that can be automatically enforced to detect security flaws in C-conforming applications.
- To be considered a security flaw, a software bug must be triggerable by the actions of a malicious user or attacker.

# ISO 17961 - C Secure Coding Rules



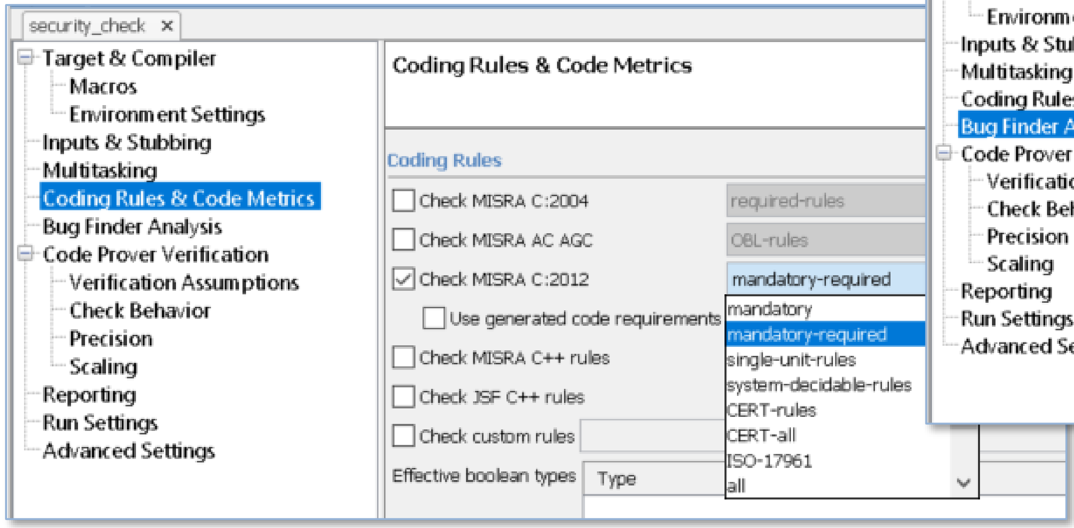
## 3.2 Coverage Summary

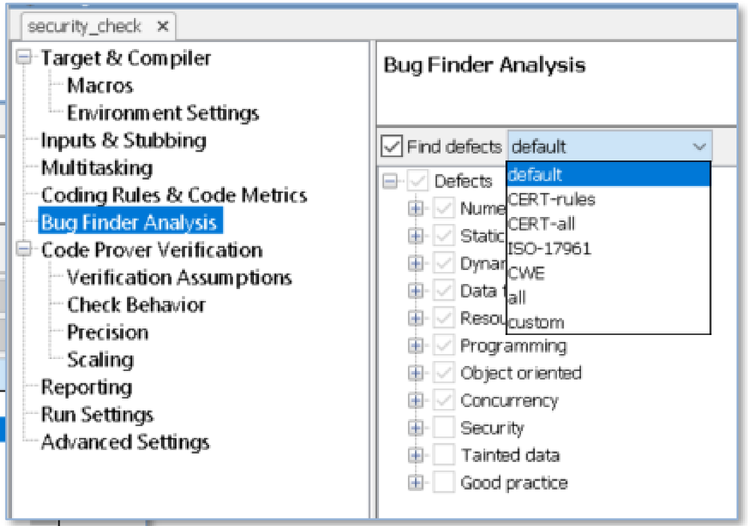
In summary, the coverage of MISRA C:2012 against C Secure is as follows:

| Classification | Strength | Number |
|---|---|---|
| Explicit | Strong | 20 |
| | Weak | 2 |
| Implicit | Strong | 1 |
| | Weak | 6 |
| Restrictive | Strong | 11 |
| | Weak | 0 |
| Partial/Restrictive | Strong/None | 2 |
| None | None | 4 |
| | Total | 46 |

# Polyspace Bug Finder And Security Standard

- Well-know defects for unreliable code like buffer overflows, dead code…
- Plus two categories: Security and Tainted data




- Security Standards
  - CERT-C
  - ISO-17961 (Full)
  - MISRA-C 2012 (Full)
  - CWE

- The mapping table between Polyspace Bug Finder and Security Standard
  - MATLAB_INSTALL\polyspace\resources\Polyspace Results R2018b.xlsx

# How does *Polyspace* help you with embedded software security?

- Detecting security vulnerabilities and underlying defects early

- Provides Exhaustive Documentation and recommendation for security fix

- Proving absence of certain critical vulnerabilities

- Complying with industry standards – MISRA-C, CWE, CERT C, ISO 17961

# Q & A