UPCOMING IMPACT OF THE SECOND EDITION OF THE ISO 26262

MGIGroup, 11.07.2017

SOFTWARE QUALITY.
MADE IN GERMANY.

SOLUTIONS FOR INTEGRATED QUALITY ASSURANCE
OF EMBEDDED SOFTWARE

MOTIVATION

- Release ISO 26262:2011
  - Major milestone for safeguarding safety-related systems
  - ISO 26262 applies to model-based SW development, too
- Second edition of ISO 26262
  - Distributed for review end of 2016
  - Final publication scheduled for 2018
  - Captures experiences from the last few years
- Analysis of the impact on model-based software development
  - Main questions for gap analysis
  - Major differences in general requirements
  - Changes to model-based software development
- Gain: prepare mandatory changes in advance of second edition

OUTLINE

- Use cases of model-based development
- Generic process for model-based software development
- General findings
- Changes in individual phases
  - From “Specification of software safety requirements” to “Verification of the software”
- Summary and conclusions

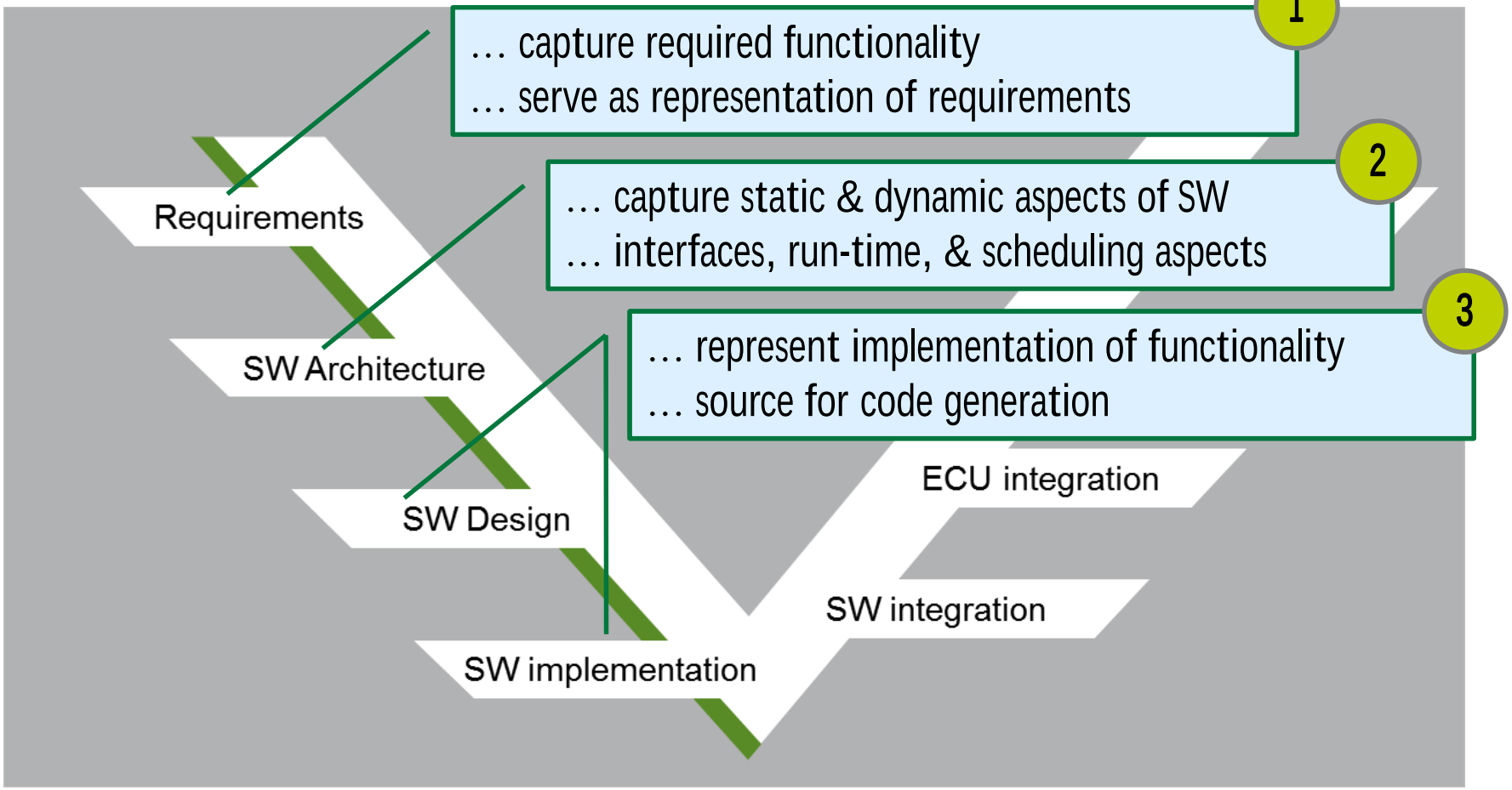
USE CASES OF MODEL-BASED DEVELOPMENT

- Significant change in Annex B (informative) on model-based development
- Five use cases for model-based software development
  - Specification of software safety requirements
  - Representation of software architectural design
  - Design and implementation of software units
  - Integration of software components
  - Verification of the software

- Benefit
  - Unified reference for communication
  - Clarification: e.g. models for SW safety requirements might not be used for integration of software components

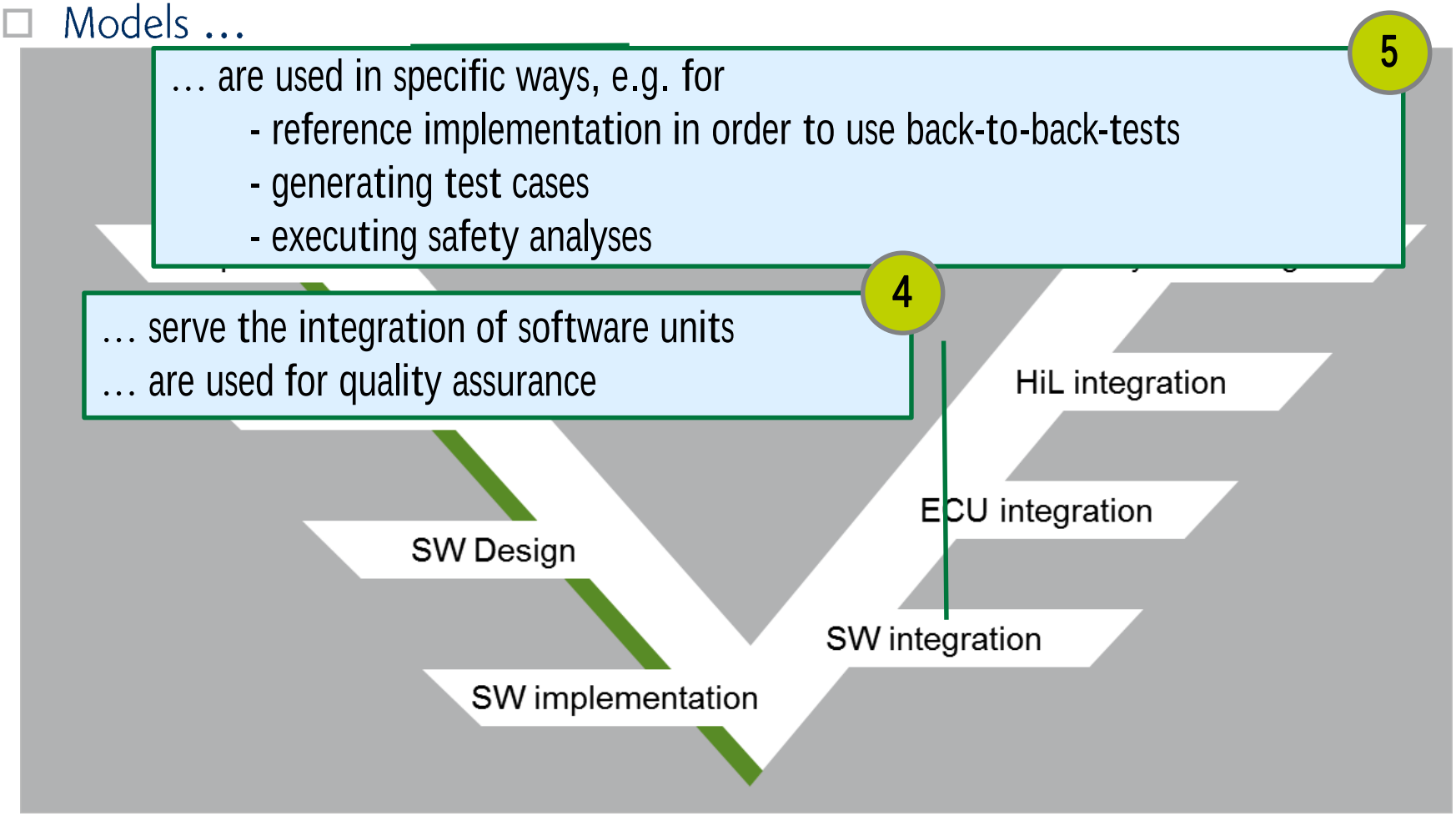
USE CASES OF MODEL-BASED DEVELOPMENT

- Models …



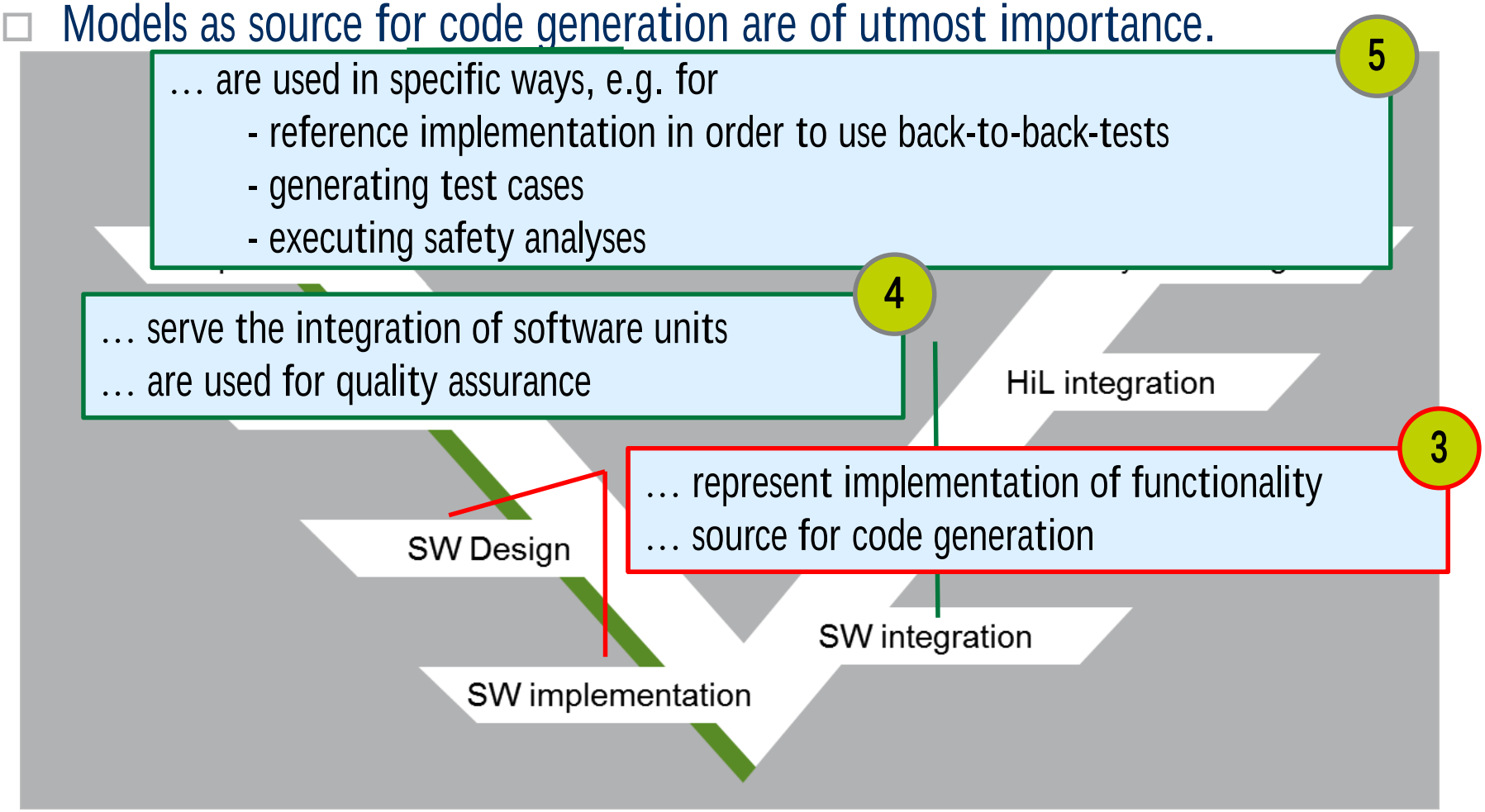
USE CASES OF MODEL-BASED DEVELOPMENT

- Models …



USE CASES OF MODEL-BASED DEVELOPMENT

- Models as source for code generation are of utmost importance.



GENERAL FINDINGS: SOFTWARE DEVELOPMENT FOR MULTI-CORE HW PLATFORMS

- New methods are listed that address typical issues arising from concurrent software execution on multi-core systems

- Extensions by methods:
  - Table 1 "Modelling and Coding guidelines" asks for guidelines on the "Representation of concurrency aspects"
  - Table 4 "Mechanisms for error detection" generalizes "Control flow monitoring" to "Temporal monitoring of program execution"
  - Table 4 asks for active access permission control mechanisms
  - Table 6 "Methods for the verification of the software architectural design" asks for scheduling analysis

GENERAL FINDINGS: EVOLUTION OF TECHNOLOGY AND BEST PRACTICES (1 / 2)

- Table 5 "Mechanisms for error handling": more detailed mechanisms for redundancy
  - "Independent parallel redundancy" split into recommendations on a) homogeneous and b) diverse redundancy in the design
- Table 9 "Methods for software unit verification": added verification techniques
  - Pair-programming
  - Static analyses based on abstract interpretation
- Table 12 "Methods for verification of software integration" added verification techniques
  - No longer just applicable to software units but also to software integration
  - E.g. analyses on control or data flow, static code analysis, as well as static analyses based on abstract interpretation

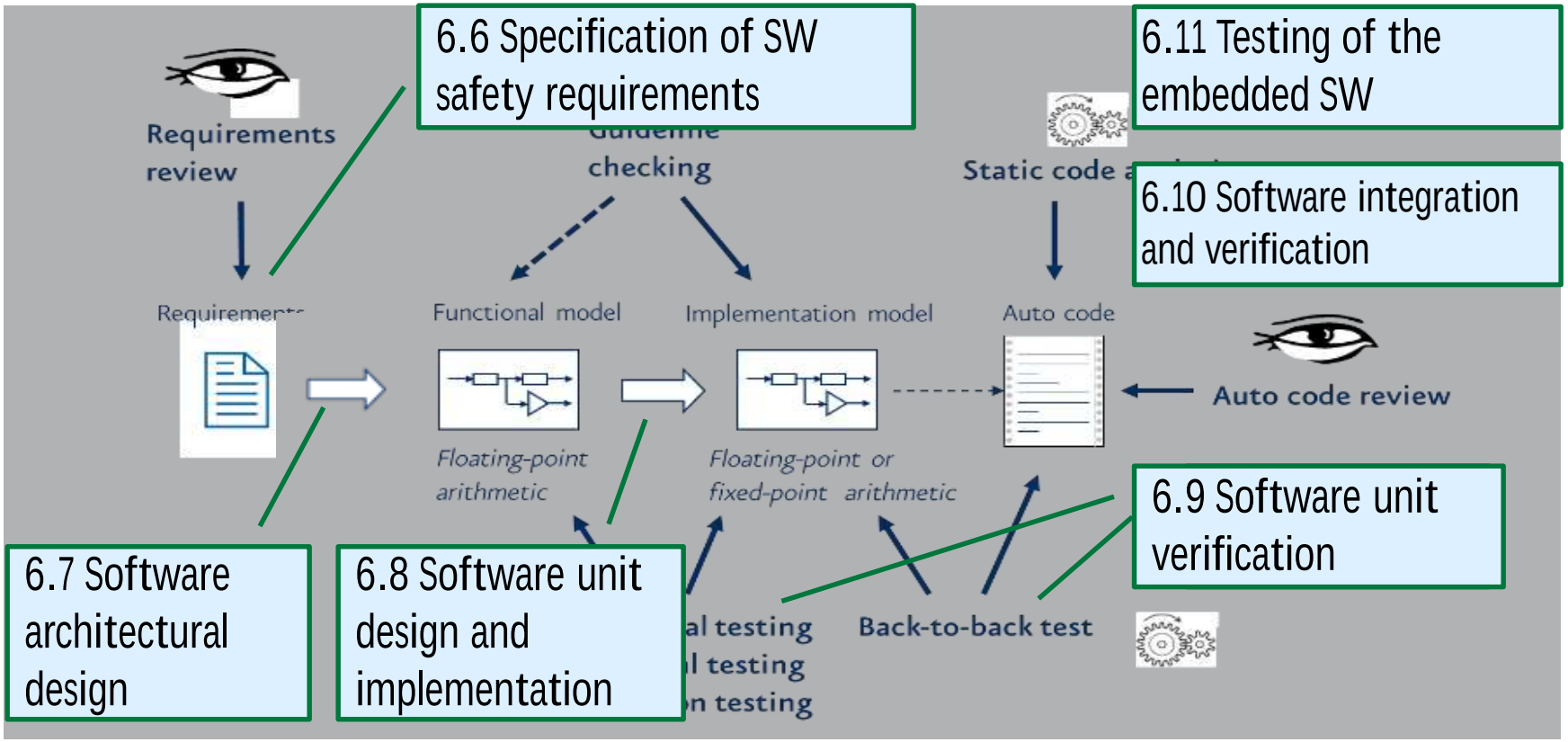
GENERAL FINDINGS: EVOLUTION OF TECHNOLOGY AND BEST PRACTICES (2 / 2)

- New table 16 "Methods for deriving test cases" for testing embedded software
  - Recommended methods partially drawn from verification of software integration
  - Added analysis of functional dependencies and operational use cases
- The method recommendation level has been adopted in various tables
  - Reflects the evolved state-of-the-art nature
  - Several techniques have been proven to be more powerful
    - ➔ Promotion from a simple "recommendation", i.e. "+", to "highly recommended"

		2011	2018
7 Notations for software unit design	1c) Semi-formal notations	ASIL B: ++	ASIL B: +
8 Design principles for software unit design	1d) No multiple use of variable names	ASIL A: +	ASIL A: ++
	1e) Avoid global variables or else justify their usage	ASIL A, B: +	ASIL A, B: ++

GENERIC PROCESS FOR MODEL-BASED SW DEVELOPMENT

- A generic process for model-based software development has been defined
  - Compliant with ISO 26262 Part 6

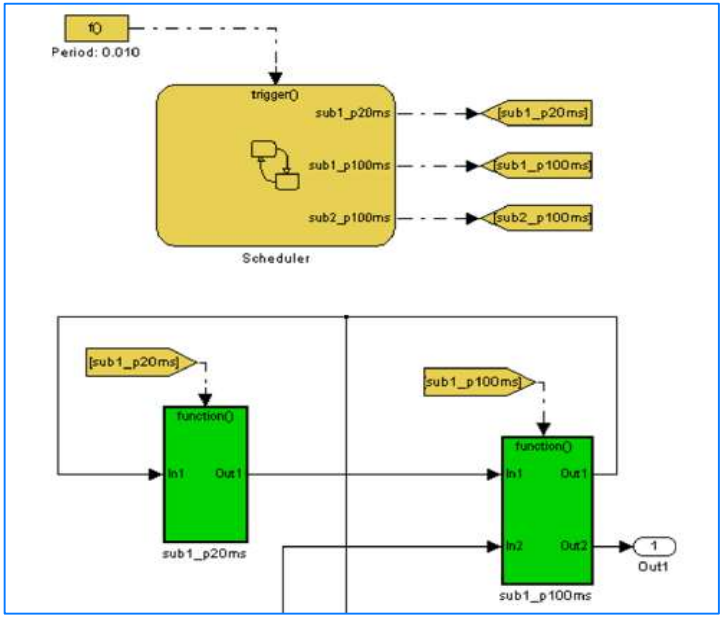


CHANGES TO PHASE: SPECIFICATION OF SOFTWARE SAFETY REQUIREMENTS

- Minor change
  - Safety plan and software verification plan: still input into the specification of software safety requirements anymore, as part of the overall safety management
- Note: Models for requirements specification
  - No dedicated recommendations for use case “Specification of software safety requirements” from Annex B
  - No requirements on how models shall be used for requirements specification
  - Particular refinement and interpretation of standard needed when using models for the specification of safety requirements

CHANGES TO PHASE: DEVELOPING THE SOFTWARE ARCHITECTURAL DESIGN

- Minor change: Table 3 “Principles for software architectural design”
  - Extended to cope with multi-core systems
  - Safety-related software components shall use only priority-based interrupts
  - Rationale: interrupt structure that is easier to understand and analyze
  - For software components rated ASIL D this principle is even listed with “++”
- Software architectural design shall also represent concurrency aspects
  - As processes or tasks
  - Option in Simulink:
    use central state chart scheduling
    subsystem tasks by function calls
    or based on a fixed and periodic rate

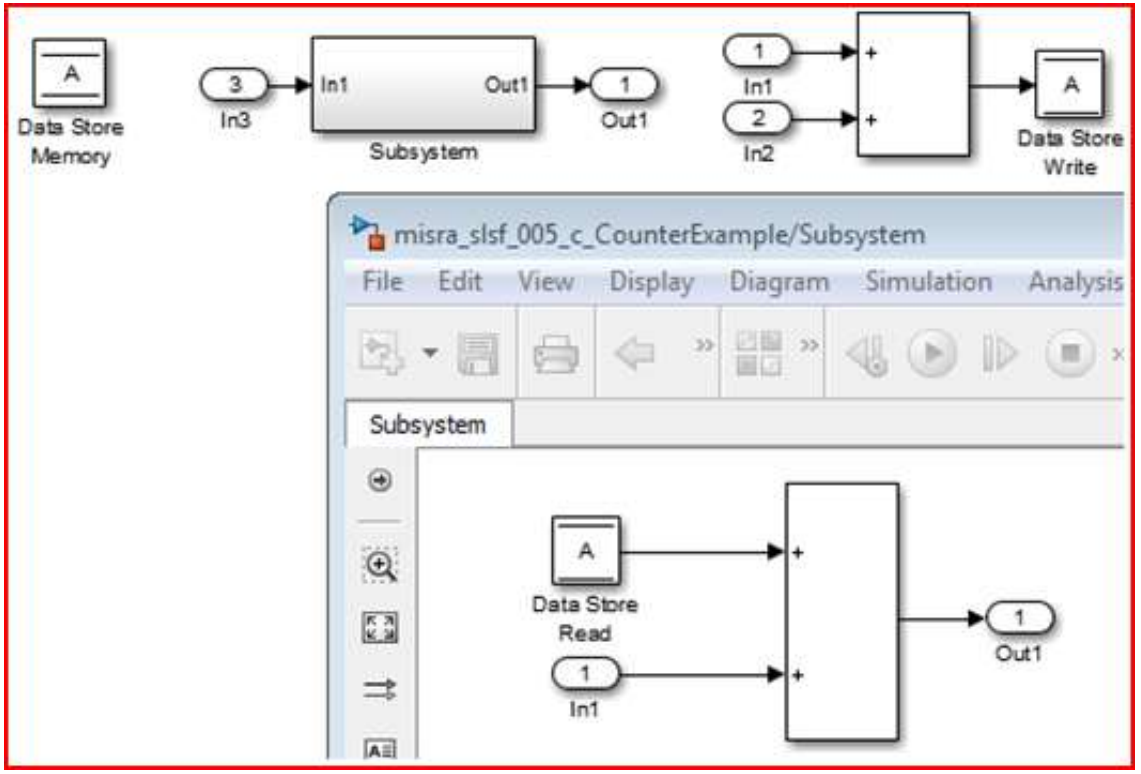


… as we have discussed in the last MGIG

CHANGES TO PHASE: DEVELOPING THE SOFTWARE ARCHITECTURAL DESIGN

- New principle for all ASILs required
  - Appropriate management of shared resources
  - Software architectural design shall consider access to shared resources
- Counter example:
  - Use of global data store memory blocks
  - Data store blocks shall not span over different levels of a model
  - See misra_slsf_0005_c

Further instances of snared resources?



CHANGES TO PHASE: DESIGN AND IMPLEMENTATION OF SOFTWARE UNITS

- "Semi-formal" notation for SW unit design refined (see footnote of Table 7)
  - Pseudo code
  - UML or SysML
  - Simulink / Stateflow
  - Structured sentences or requirement patterns with controlled vocabulary
- Model-based development with subsequent code generation is the standard use case
  - Manual coding from models not recommended
- Strengthened recommendations for specific design principles
  - "no multiple use of variable names" (see misra_slsf_027_j)
  - "avoid global variables" (see misra_slsf_018_a)

CHANGES TO PHASE: SOFTWARE UNIT VERIFICATION

- Notion "verification" extends "test" through other measures like analyses and reviews
  - See also merge of tables 9 "Methods for the verification of software unit design and implementation" and 10 "Methods for software unit testing" into new table 9 "Methods for software unit verification"
- Static analyses explicitly include MISRA rule checks
- When evidence is given that model coverage represents code coverage properly, measurement of structural coverage at model level can replace the measurement at code level

CHANGES TO PHASE: SOFTWARE INTEGRATION AND VERIFICATION

- Again, notion "verification" extends "test" through other measures like analyses and reviews
  - Following the evolution of technology, static analyses known from unit verification shall be applied to integration level
    - Analyses of control or data flow
    - Static code analysis
    - Static analyses based on abstract interpretation
  - Static code analyses include modeling or coding rule checks (e.g. MISRA)
  - Additionally, software architecture testing can be performed at model level using analogue structural coverage metrics

Your best practices for integration test of applications:

- Integration at model level with model referencing or libraries?
- Integration at c-code level

CHANGES TO PHASE: TESTING OF THE EMBEDDED SOFTWARE

- Renamed “Verification of Software Safety Requirements”

- Change: Methods for deriving test cases recommended
  - Copied from methods for integration testing
  - Additional test methods: analysis of functional dependencies and operational use cases

SUMMARY

- Model-based development use cases have been refined
  - "Design and implementation of software units" explicitly mentioned
  - Four further use cases for models in SW development
- General findings
  - Recommendations extended to software development for multi-core HW platforms
  - Evolution of technology and best practices reflected by extension of methods and change of degree of recommendations
- Changes in individual phases
  - Model-based development with subsequent code generation is the standard use case; manual coding from models not recommended
  - Verification methods lifted from unit to integration level, and testing extended by analyses

RECENT POST ON MGI-GROUP

- Michael Burke: “The history junction block in Stateflow™ is a simple, powerful, and often misunderstood entity. In this blog post, I will try to explain the use of the history junction, through a, hopefully, humorous example.”
  - see https://www.linkedin.com/groups/8414520/8414520-6281161112935231492

MGIGROUP – NEXT ONLINE MEETING

- Tuesday, October 17, 2017
  3:00 pm CET (Berlin)
  9:00 am EST (Detroit)
  6:30 pm IST (Bangalore)
  9:00 pm CST (Beijing)
  10:00 pm JST (Tokyo)

- Link to Event:
  https://model-engineers-event-en.webex.com/model-engineers-event-en/onstage/g.php?MTID=e0fd2ed73fd321ca8396228357045aa0e

MES
MODEL ENGINEERING SOLUTIONS

MODEL ENGINEERING SOLUTIONS GMBH

Waldenserstraße 2 - 4
10551 Berlin
Germany

T: +49 30 2091 6463-0
F: +49 30 2091 6463-33
info@model-engineers.com
www.model-engineers.com