Alessandra Melani – Samuele Mazzoleni

Advanced R&D Department
Safety System Development & Process Team

An efficient approach to consider SW quality metrics during unit design phase

AGENDA

- SW quality metrics introduction
- Standard approach for metrics compliance
- Proposed approach to improve efficiency
- Metrics estimated during design phase
- Practical example on a real use case
- Conclusions

AGENDA

- SW quality metrics introduction
- Standard approach for metrics compliance
- Proposed approach to improve efficiency
- Metrics estimated during design phase
- Practical example on a real use case
- Conclusions

Why SW quality metrics?

Software quality metrics are widely used in the automotive industry:

- To set **software quality goals** and measure them
- To maximize the chance to release high-quality software in a fast-paced development environment
- To ensure **maintainability**, **readability**, **testability** of the code

SW quality metrics reference standards

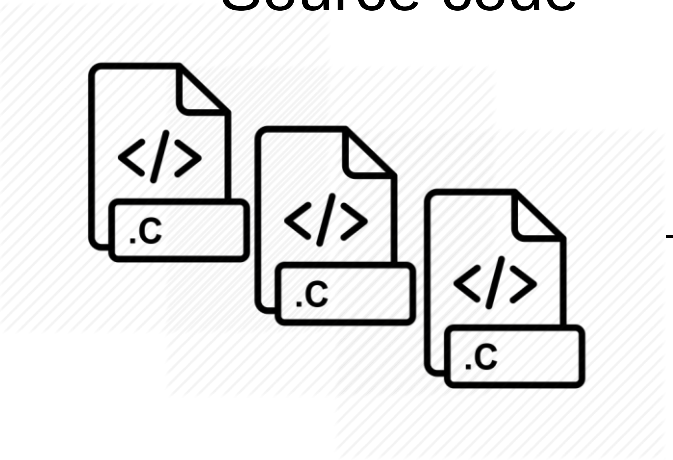
- Automotive SPICE (SWE.4 “Software Unit Verification” process)
  - **SWE.4.BP3: Perform static verification of software units.** Verify software units for correctness using the defined criteria for verification. Record the results of the static verification.

- ISO 26262:2018 (Part 6, Clause 9 “Software Unit Verification”)
  - Static code analysis is a method for software unit verification that is highly recommended (++) for all ASIL levels

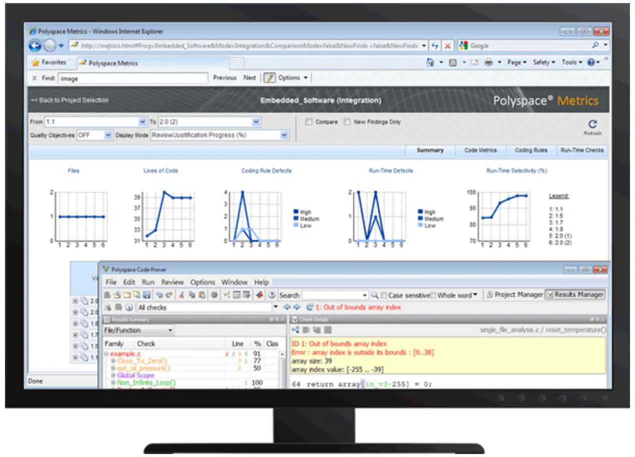
- HIS (Hersteller Initiative Software) Source Code Metrics
  - A set of metrics to be used in the evaluation of software is specified, as well as their acceptable range

Tool used to verify the compliance

Source code



Static analysis tool



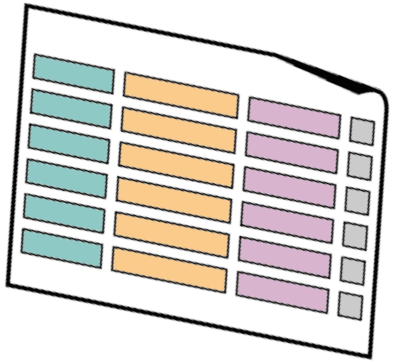
1: Out of bounds array index

single_file_analysis.c / reset_temperature()

ID 1: Out of bounds array index
Error : array index is outside its bounds : [0..38]
array size: 39
array index value: [-255 .. -39]

```
64  return array[in_v3-255] = 0;
```

- Mathworks Polyspace Bug Finder
- QA Systems QA-C
- Synopsys Coverity
- GrammaTech CodeSonar
- Etc.

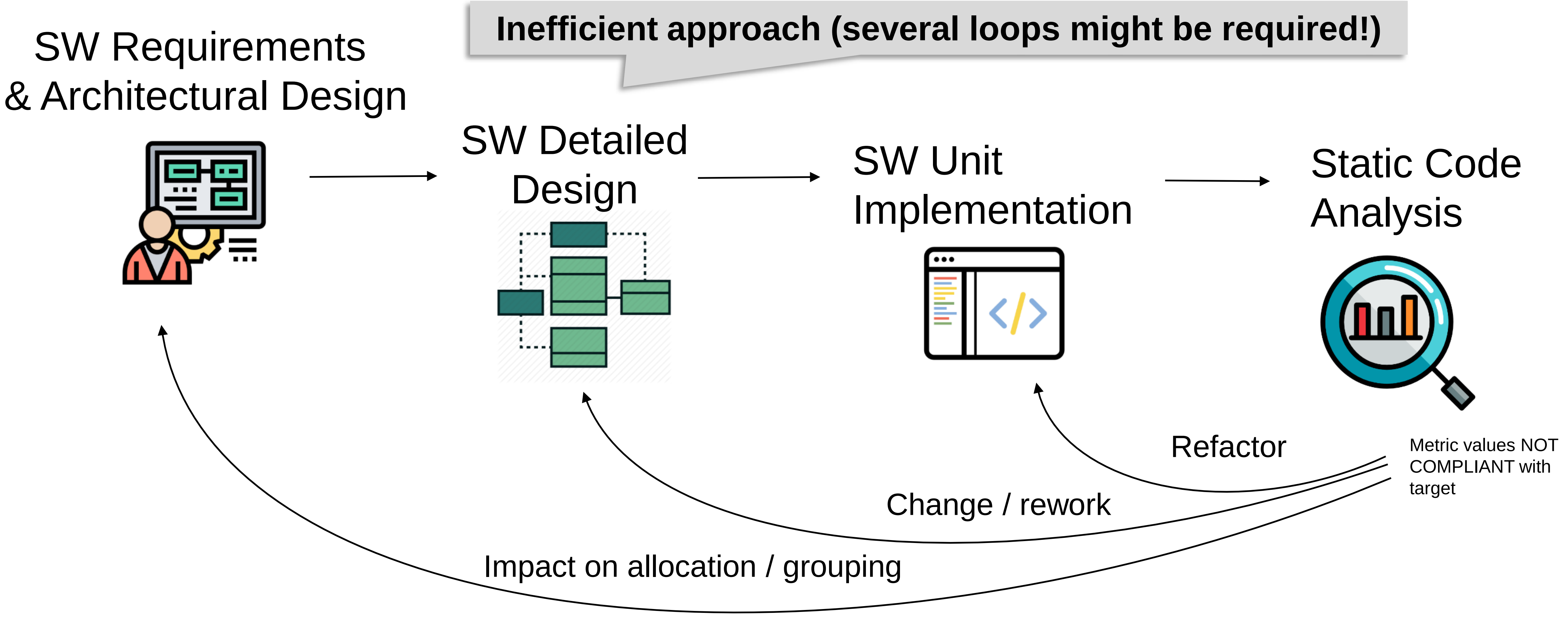


Code metrics
database

AGENDA

- SW quality metrics introduction
- Standard approach for metrics compliance
- Proposed approach to improve efficiency
- Metrics estimated during design phase
- Practical example on a real use case
- Conclusions

Standard approach for metrics compliance



Ineffiicient approach (several loops might be required!)

SW Requirements & Architectural Design

SW Detailed Design

SW Unit Implementation

Static Code Analysis

Metric values NOT COMPLIANT with target

Refactor

Change / rework

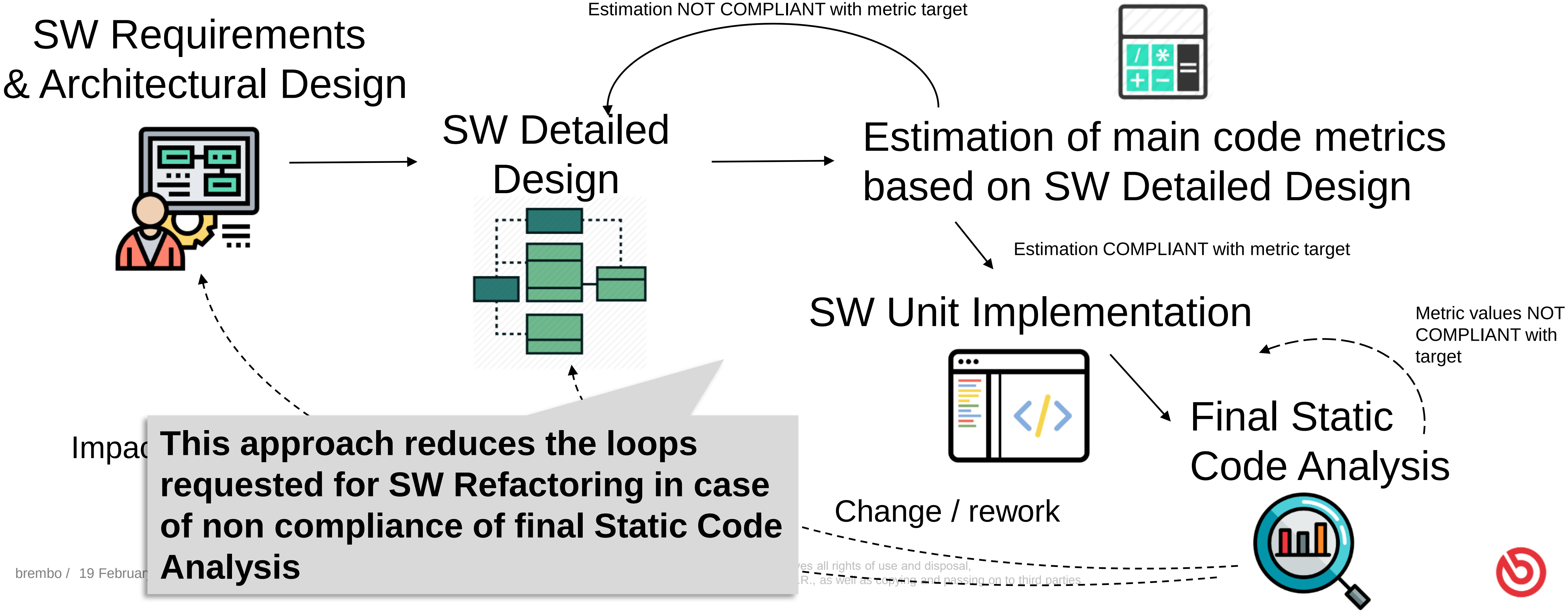
Impact on allocation / grouping

AGENDA

- SW quality metrics introduction
- Standard approach for metrics compliance
- Proposed approach to improve efficiency
- Metrics estimated during design phase
- Practical example on a real use case
- Conclusions

Proposed approach to improve efficiency

Before starting SW Unit Implementation, add an estimation on code metrics compliance based on SW Detailed Design



Estimation NOT COMPLIANT with metric target

SW Requirements & Architectural Design

SW Detailed Design

Estimation of main code metrics based on SW Detailed Design

Estimation COMPLIANT with metric target

SW Unit Implementation

Metric values NOT COMPLIANT with target

Final Static Code Analysis

Change / rework

Impac

This approach reduces the loops requested for SW Refactoring in case of non compliance of final Static Code Analysis

AGENDA

- SW quality metrics introduction
- Standard approach for metrics compliance
- Proposed approach to improve efficiency
- Metrics estimated during design phase
- Practical example on a real use case
- Conclusions

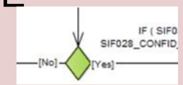
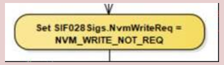
Project and file metrics estimation

- Project metrics: project-wise requirement, oriented to high-level SW coding rules
- File Metrics: single unit file (.c and .h) requirements

TYPE	CODE METRIC	DESCRIPTION	ESTIMATION POSSIBLE	HOW TO ESTIMATE
Project metrics	Number of recursions	Number of call graph recursions (number of call cycles over one or more functions). If one function is at the same time directly recursive (it calls itself) and indirectly recursive, the call cycle is counted only once	☺	At design phase and as SW development guideline
	Number of direct recursions	Number of call cycles of functions to themselves	☺	At design phase and as SW development guideline
	Number of GOTO statements	Number of GOTO statements within a function. break and continue are not counted as GOTO statements	☺	At design phase and as SW development guideline
File metrics	Comment density	Relationship of the number of comments (outside of and within functions) to the number of statements	☹	Not possible, strictly related to SW unit implementation phase

Function metrics estimation (1/2)

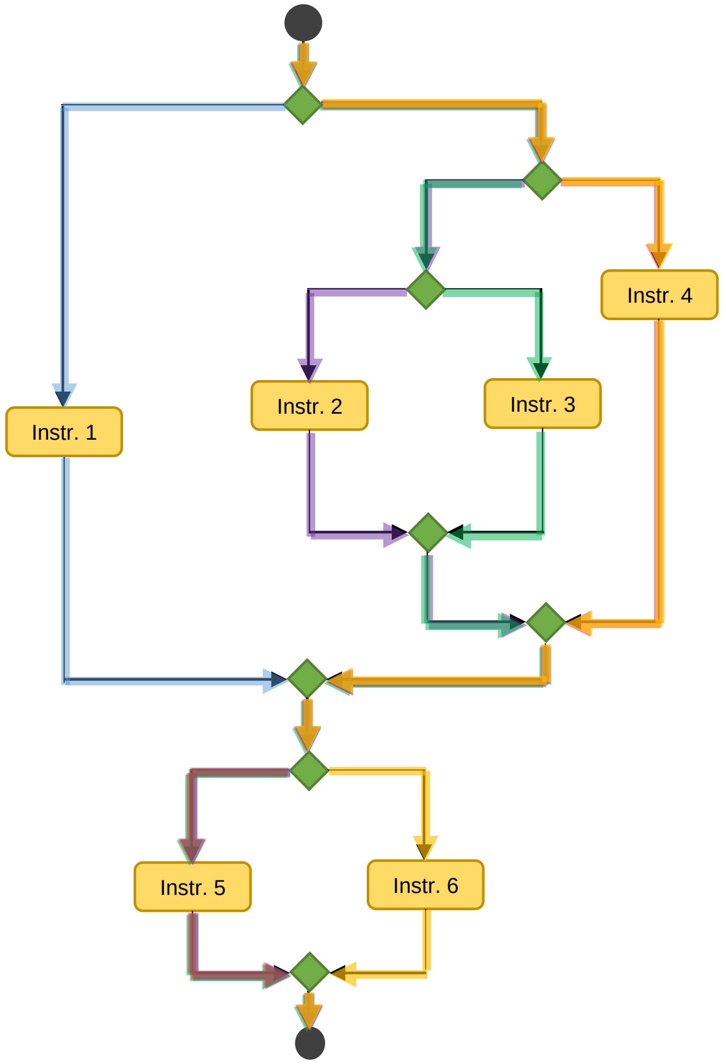
- **Function metrics:** function-wise requirement oriented to function complexity and maintainability

TYPE	CODE METRIC	DESCRIPTION	ESTIMATION POSSIBLE	HOW TO ESTIMATE
Function metrics	Executable lines	Total number of lines with source code instructions within the function body that are not declarations (without static initializer), comments, braces, or preprocessing directives	☺	Count in the design the number of: • IF/FOR/WHILE statements  • Actions 
	Language scope	Indicator of the cost of maintaining or changing functions, computed as $(N_1 + N_2) / (n_1 + n_2)$, where N_1 is the total number of operators, N_2 is the total number of operands, n_1 is the number of different operators, and n_2 is the number of different operands	☹	Strictly related to SW reuse capability, libraries, etc.
	Cyclomatic complexity	Maximum number of independent paths through program source code. Independent path is defined as a path that has at least one edge which has not been traversed before in any other path	☺	1 + Number of IF/FOR/WHILE statements of the function design ***Note:*** *this estimation is accurate only if the Detailed Design structure assumptions explained later are satisfied*

Cyclomatic complexity estimation

- Binary decisions: N = 4
- Cyclomatic complexity: N + 1 = 5

- Required assumptions:
  - Only one entry and exit point for each function
  - Control structures including only binary decisions
  - Control flow that follows only one direction, from function start to end



Function metrics estimation (2/2)

- Function metrics: function-wise requirement oriented to function complexity and maintainability

TYPE	CODE METRIC	DESCRIPTION	ESTIMATION POSSIBLE	HOW TO ESTIMATE
Function metrics	Number of calling functions	Number of distinct callers of a function	☺	Count the function callers in the design
	Number of called functions	Number of distinct functions called by a function	☺	Count in the function design the number of distinct functions called
	Call levels	Depth of function nesting, i.e., maximum depth of control structures within a function body	☺	Evaluate the depth of control structures included in the function design
	Number of function parameters	Number of parameters per functions. It gives an indication of how complex is the function interface	☺	Count the number of parameters specifed in the design about function call
	Number of return statements	Number of explicit return statements within a function body	☺	At design phase and as SW development guideline

Metrics estimation recap

- **10 out of 12 code metrics can be estimated at design phase**

- In case of non-compliance, the Detailed Design can be changed **before SW coding**

- The updated design will «guide» the unit implementation in a direction that increases the probability of compliance of final Static Code Analysis

AGENDA

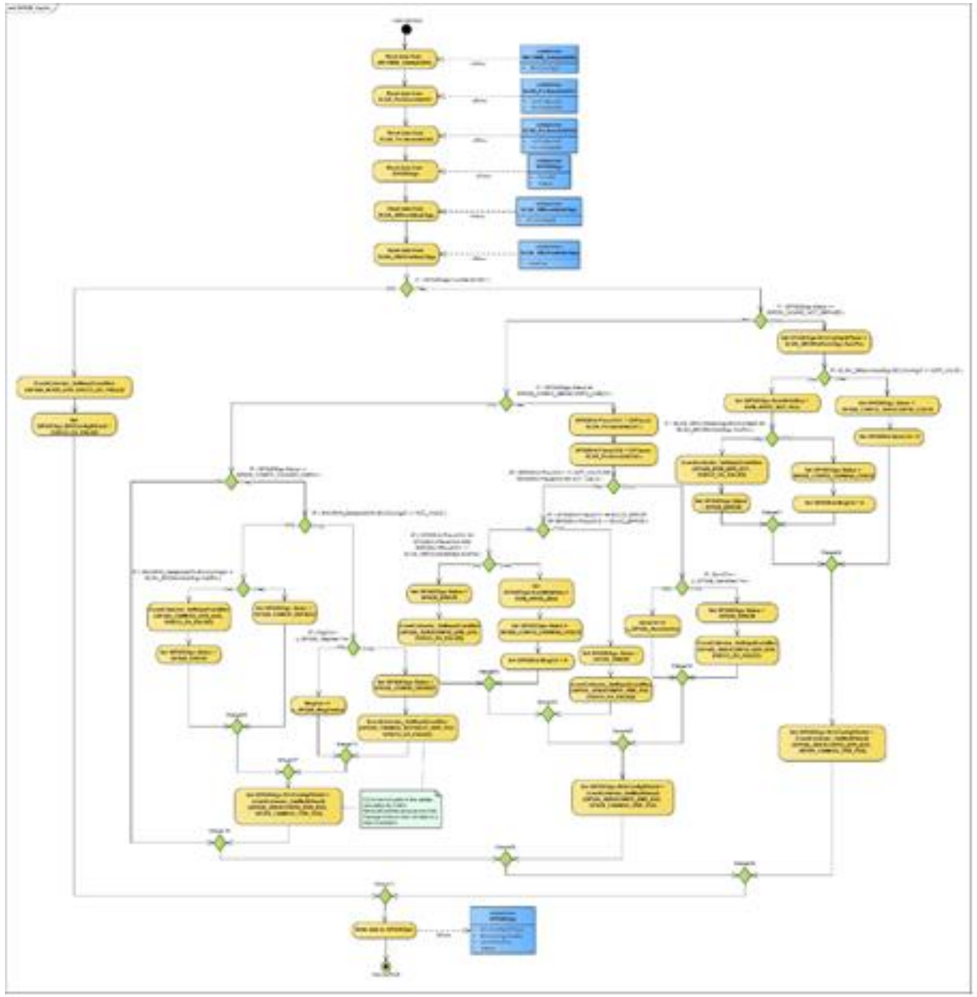
- SW quality metrics introduction
- Standard approach for metrics compliance
- Proposed approach to improve efficiency
- Metrics estimated during design phase
- Practical example on a real use case
- Conclusions

Practical example on a real use case

- Safety Integrity Function **SIF028_Cyclic** to be implemented with hand-written coding

- Detailed Design developed starting from SW architectural design and SW unit requirements using a semi-formal notation (SysML activity diagram)

- Code metrics estimated with the proposed approach
  - E.g., cyclomatic complexity estimation is equal to 14 (= 13 IF statements + 1)



Tool metrics result comparison

TYPE	CODE METRIC	ESTIMATION FROM DETAILED DESIGN
Project metrics	Number of recursions	0
	Number of direct recursions	0
	Number of GOTO statements	0
Function metrics	Executable lines	~ 50
	Cyclomatic complexity	14 (13 IFs + 1)
	Language scope	-
	Number of calling functions	1
	Number of called functions	2
	Call levels	6
	Number of function parameters	0 (void function)
	Number of return statements	0

Estimated metrics compliant with target → No design change needed

SIF028_Cyclic implemented with hand-written coding

Tool metrics result comparison

TYPE	CODE METRIC	ESTIMATION FROM DETAILED DESIGN	ACTUAL VALUE FROM STATIC CODE ANALYSIS
Project metrics	Number of recursions	0	0
	Number of direct recursions	0	0
	Number of GOTO statements	0	0
Function metrics	Executable lines	~ 50	55
	Cyclomatic complexity	14 (13 IFs + 1)	14
	Language scope	-	6.9
	Number of calling functions	1	1
	Number of called functions	2	2
	Call levels	6	6
	Number of function parameters	0 (void function)	0
	Number of return statements	0	0

Final Static Code Analysis performed using Mathworks Polyspace, confirming the initial estimation

AGENDA

- SW quality metrics introduction
- Standard approach for metrics compliance
- Proposed approach to improve efficiency
- Metrics estimated during design phase
- Practical example on a real use case
- Conclusions

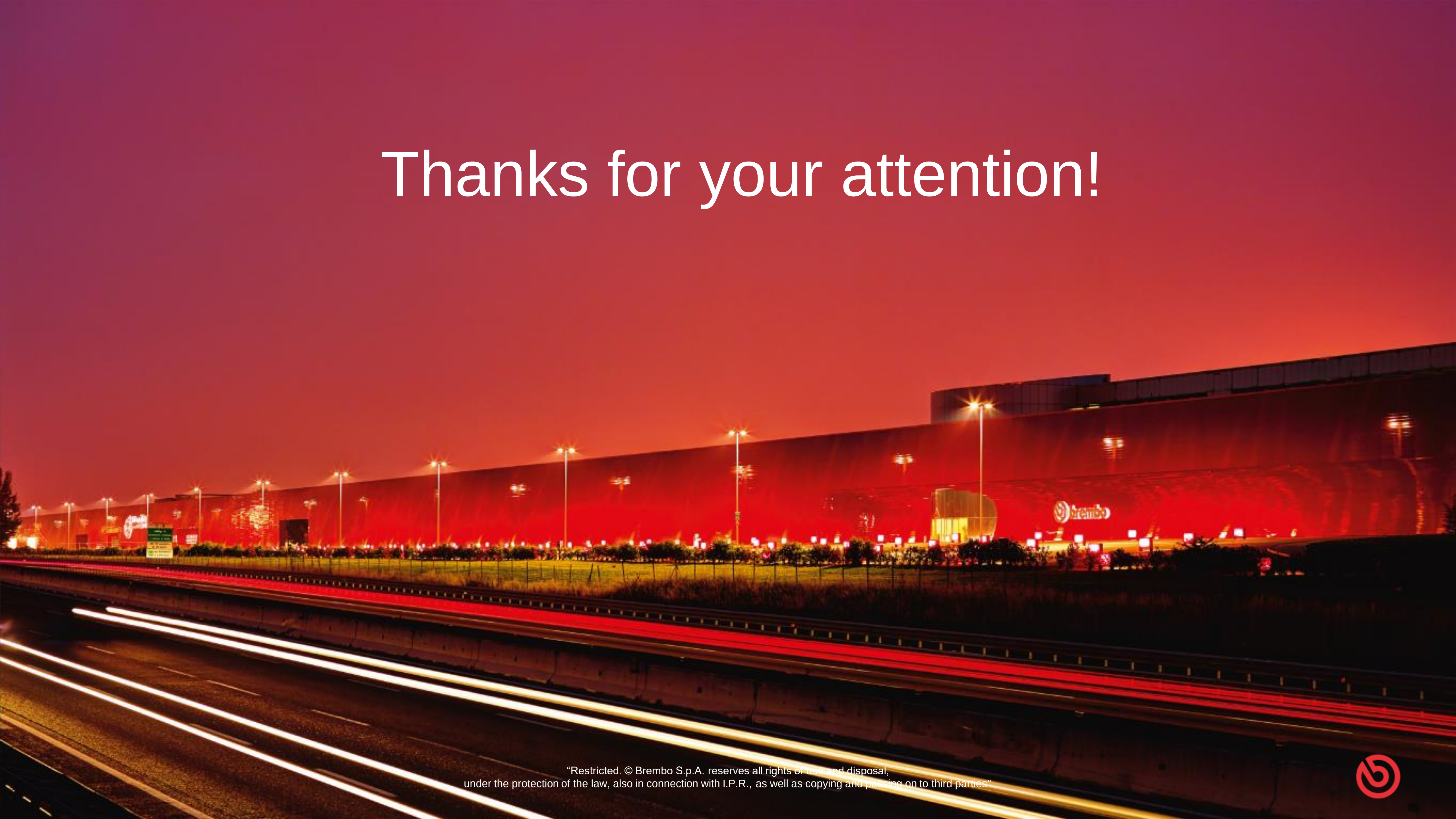
Conclusions

- Code metrics are a crucial indicator to evaluate the quality of a SW unit

- Evaluating code metrics only at the end of the development process could lead to **significant reworking effort**, possibly impacting all previous phases:
  - SW Requirements
  - SW Detailed Design
  - SW Coding

- The proposed approach reduces the loops requested for SW refactoring in case of non-compliance of final Static Code Analysis by providing an estimation of code metrics before SW unit implementation

- In this way, quality objectives are met, while reducing **time**, **effort** and **risks**



Thanks for your attention!