Code-Analysis and Metrics for SystemC

Tim Kraus, Ingo Feldner

Robert Bosch GmbH

Corporate Research and Advance Engineering

accellera SYSTEMS INITIATIVE

SYSTEMC™ EVOLUTION DAY

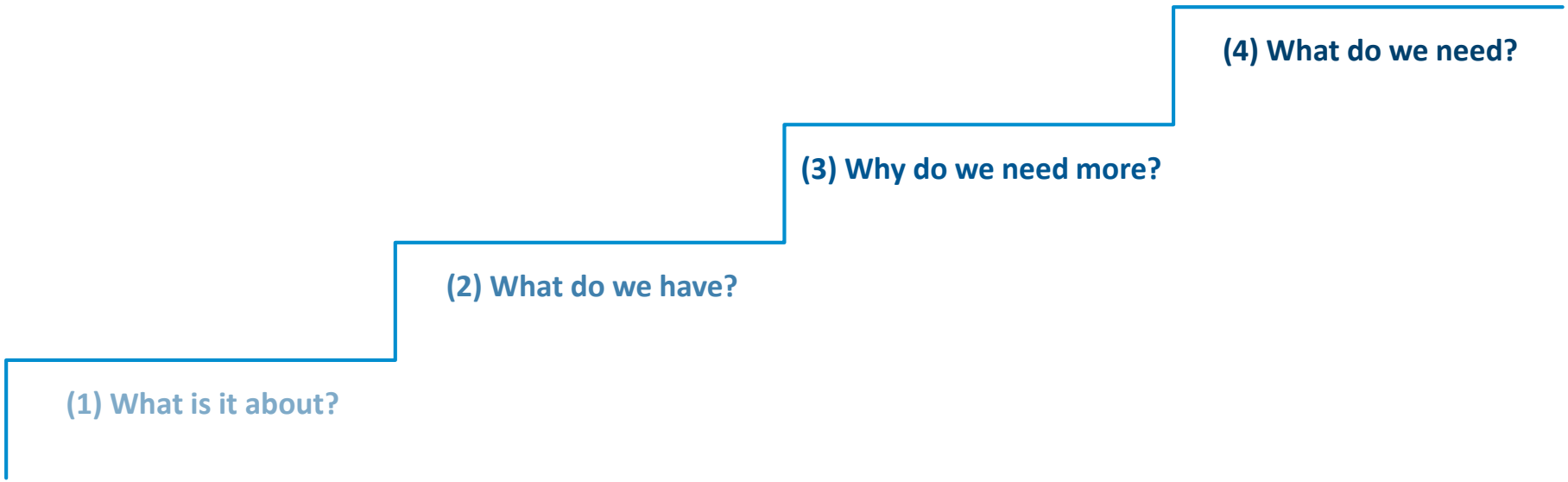
OCT 31, 2019 | MUNICH | GERMANY

Presentation Copyright Permission

– A non-exclusive, irrevocable, royalty-free copyright permission is granted by Robert Bosch GmbH to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC** standard, and in derivative works based on this standard.

Code-Analysis for SystemC

Overview



Code-Analysis for SystemC

(1) What is it about?

- The request for virtual prototypes in different applications is getting bigger

- So more and more models are around - developed in various contexts
  - Model developers are not necessarily C++ software-experts
  - New challenges come up in terms of maintaining and sharing models!

- State of the art in software development is: **Static Code Analysis**
  - We must make use of this technology for SystemC!

- For functional and performance evaluation: **Dynamic Analysis** (“Simulation Based”) → Not in focus here

Code-Analysis for SystemC

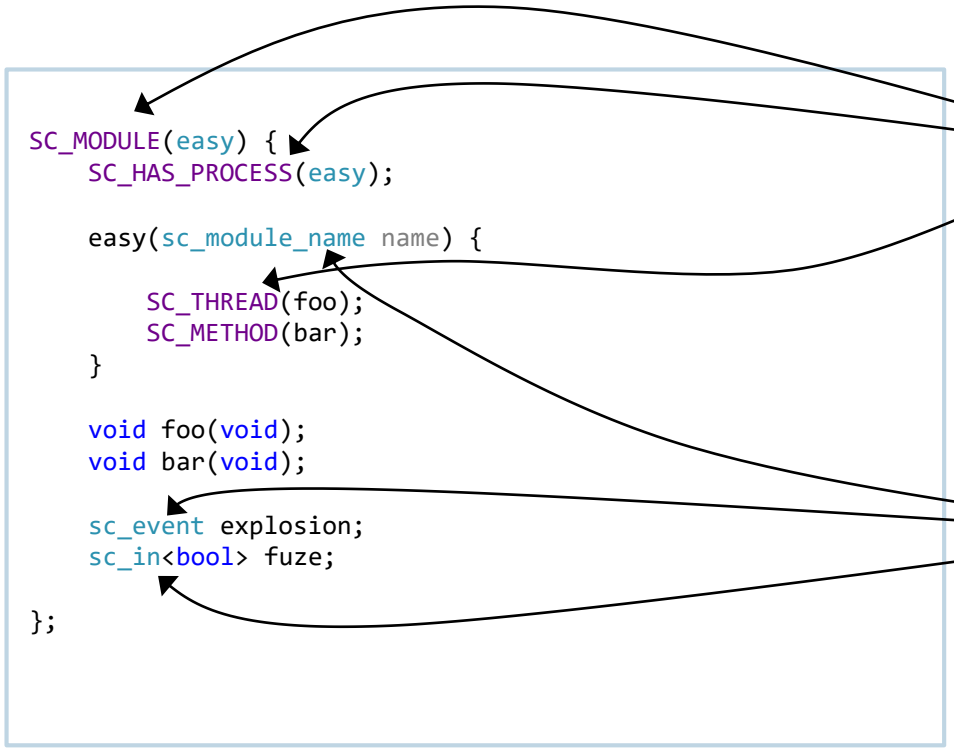
(2) What do we have?

- Many proven (commercial and free) tools for static code-analysis for C and C++
  - Check for stability, runtime errors
  - Check of quality metrics like complexity, maintainability, style
  - Find common bugs in safety-critical embedded-systems according to a rulesets
    (like MISRA-C++ or CERT)

- Common Problem: How can a big library be handled? (like QT, OpenCV, Boost, **SystemC**)
  - Scanning the lib is not really an option!?
    - Significantly increased scan times
    - Additional information needed about compiler and host system (conditional compiles)
    - Findings in the lib are not in the responsibility of the users!

Code-Analysis for SystemC

(2) Example “easy”: Requirements



```
SC_MODULE(easy) {
    SC_HAS_PROCESS(easy);

    easy(sc_module_name name) {

        SC_THREAD(foo);
        SC_METHOD(bar);
    }

    void foo(void);
    void bar(void);

    sc_event explosion;
    sc_in<bool> fuze;

};
```

Scanner needs to know these macros!

Scanner needs to know about the types!

Code-Analysis for SystemC

(3) Why do we need more?

- No well supported way to analyze SystemC specific constructs available!
- Increase quality of SystemC models and create a common understanding
- Establish metrics: objective and quantitative measure for model quality
  - Common base for development
- Benefit for **Model Developers:**
  - Increased maintainability reduces total cost of ownership
  - Prove the quality of own work
- Benefit for **VP Users / System Builder:**
  - Established way to define requirements for the purchased model
- Benefit for **Tool Suppliers:**
  - Integrate support for code-scanners as a new feature to increase usability of the tool

Code-Analysis for SystemC

(4) What do we need?

- **Input** from experienced SystemC programmers about possible issues
  - Problems that are not detected by the compiler but occur at runtime (“port not bound”)
  - Code-constructs that are known as bad to maintain
  - Code-constructs that are known to be bad for simulation performance

- Classification of the issues
  - By detection mechanism
  - By severity (bugs, warnings, code-smells, performance)

- Best for everyone: Common understanding of high-quality SystemC Code in the community
  - Make the ruleset open and free to use
  - Make the rules and metrics usable for a large number of development environments

Code-Analysis for SystemC

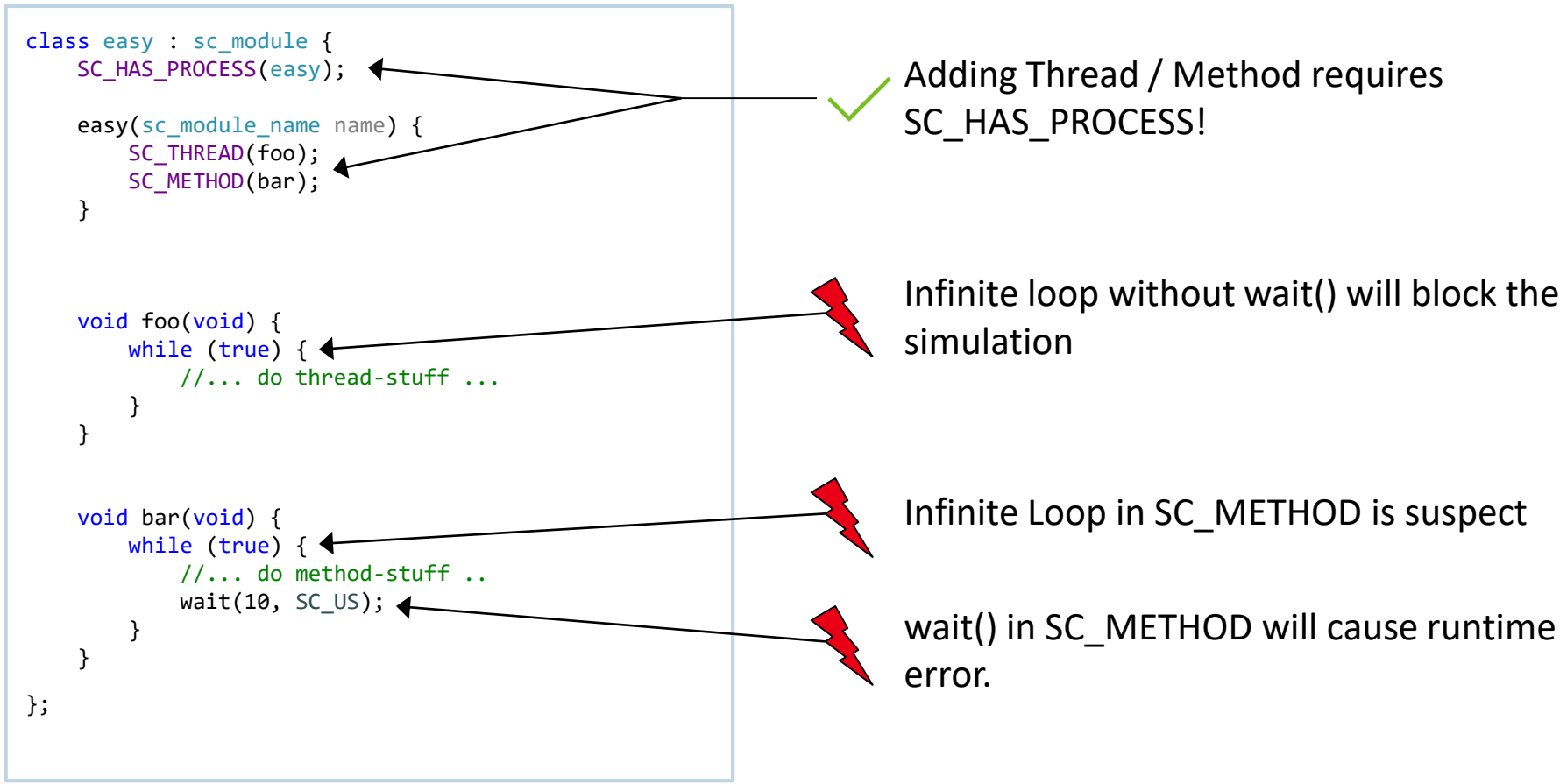
(4) What do we need?

```
class easy : sc_module {
    SC_HAS_PROCESS(easy);

    easy(sc_module_name name) {
        SC_THREAD(foo);
        SC_METHOD(bar);
    }


    void foo(void) {
        while (true) {
            //... do thread-stuff ...
        }
    }


    void bar(void) {
        while (true) {
            //... do method-stuff ..
            wait(10, SC_US);
        }
    }
};
```

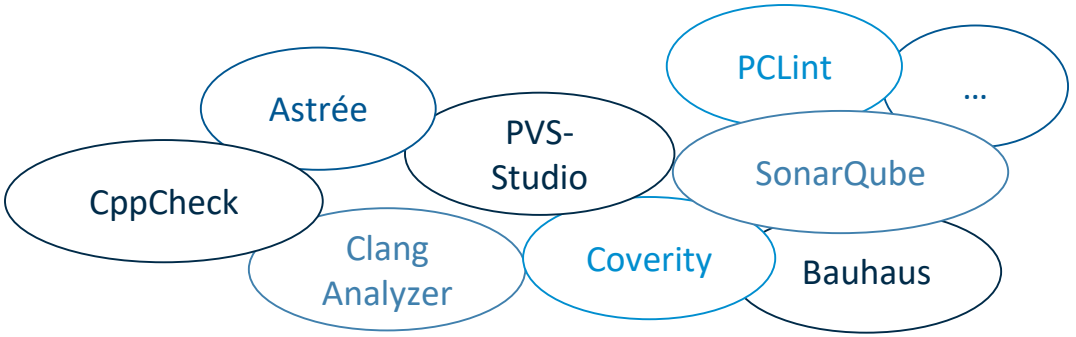


Code-Analysis for SystemC

(4) What do we need?

- Precondition: make existing C++ Analyzers work for SystemC (library issue)

- A tool to work with:



What is a well working system for you?

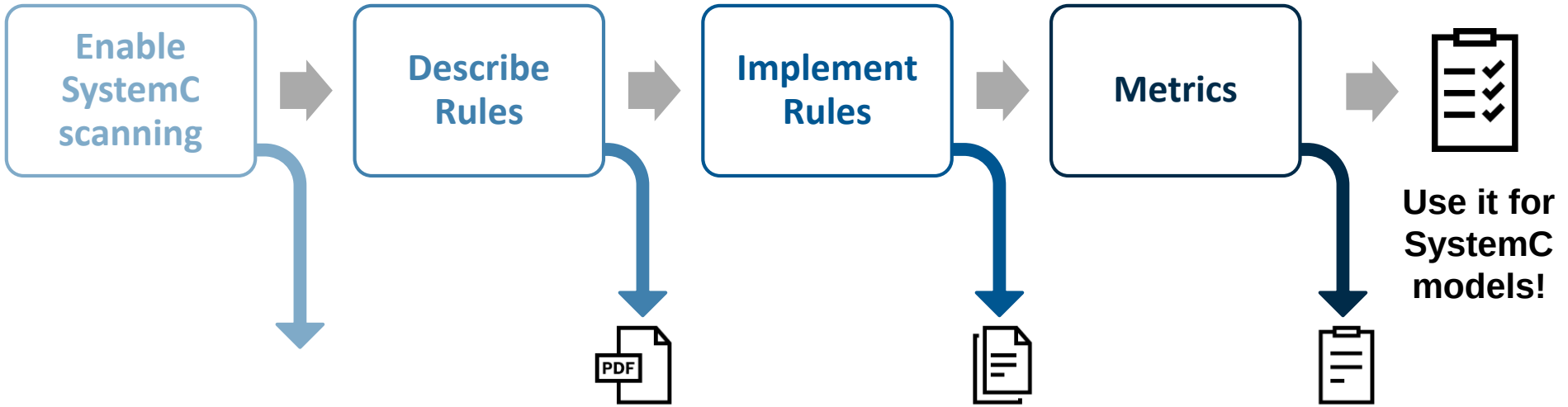
Contribute with your experiences!

Code-Analysis for SystemC

(4) What do we need?

Proposal: *Implement it on the existing SystemC code examples!*

Step by step...



Tool Parameters

- Best practices
- Tool dependent settings / config files

Publish Document (Whitepaper?)

- Application oriented
- Tool-independent
- Based on standard SystemC

Make rules usable in a scanner-tool

- Classified by severity
- Keep implementation as generic as possible

Broadly recognized acceptance criteria

- Can be used when contracting a model implementation

LET'S OPEN UP THE DISCUSSION!

If you are **interested to join** the activity:

→ See Accellera announcement https://forums.accellera.org/

→ Contact us directly per mail

... until end of 2019!

tim.kraus@de.bosch.com
ingo.feldner@de.bosch.com