

STATIC AND DYNAMIC CODE ANALYSIS IN A CONTINUOUS TESTING PROCESS

BEG-VS/EEA4 | MATTHIAS SCHMIDT

Agenda

Static and Dynamic Code Analysis
in a Continuous Testing Process

1. Introduction
2. Test in General
3. Code Analysis
4. Analysis Tools
5. Processes
6. Improvements
7. Real Projects
8. Test Effort Calculation



1. INTRODUCTION

Static and Dynamic Code Analysis
in a Continuous Testing Process

Introduction

About me

Static and Dynamic Code Analysis
in a Continuous Testing Process

< Before Bosch



**University of Applied Sciences Karlsruhe:
Power and Automation
Technologies**

<http://www.hs-karlsruhe.de>



**Testing: ECU/EPMU of TP400
(Airbus, A400M Airlifter)**

<http://www.mtu.de>



**Test Management: ECMU of MTR390
(Eurocopter, Tiger
anti-tank and escort/
support helicopter)**

<http://www.mtu.de>

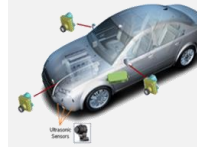


**Dipl.-Ing. (FH)
Matthias Schmidt**

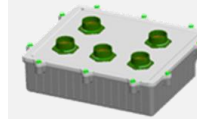
Project Leader,
Software Product
Manager Cantata

Test Management,
Verification and
Validation
(BEG-VS/EEA4)

> At Bosch



**Test Management: Surround-View Camera
System and Rear-View
Camera System**



**Test Management: Electronic Engine
Control Unit for
Airborne Systems**



- **Conductor:**
Musikkapelle
HN-Biberach e.V.
- **Alto-Saxophone:**
Bosch Big Band

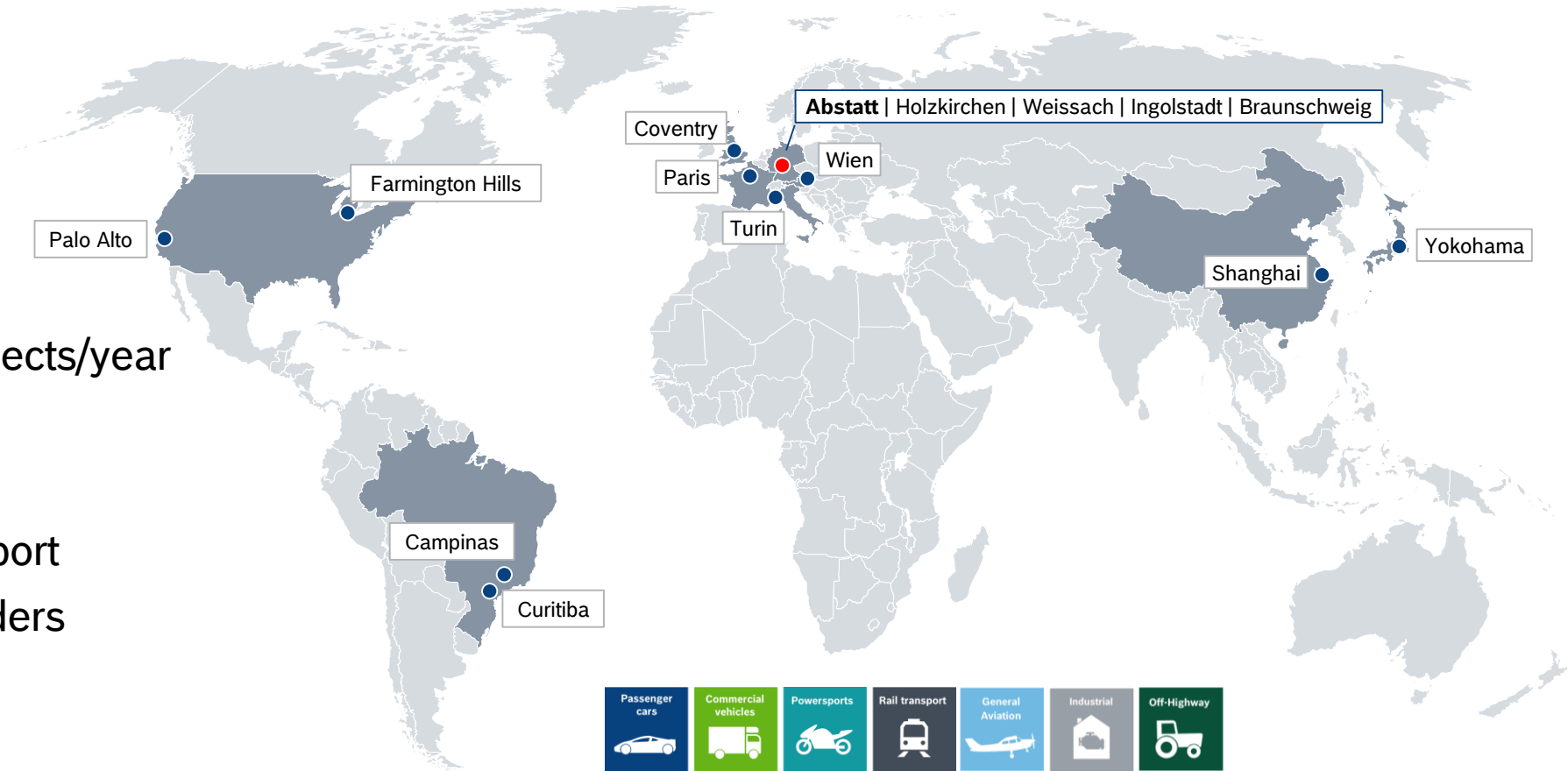


Introduction

Bosch Engineering GmbH

Static and Dynamic Code Analysis
in a Continuous Testing Process

100% Bosch
800 customer projects/year
9 countries
15 locations
115 years Motorsport
from **0** to **24** cylinders
of **1** piece upward



Introduction

Services of Department BEG-VS/EEA4

Static and Dynamic Code Analysis
in a Continuous Testing Process



Test Management



Test Strategy



E/E Integration and E/E System Test



End of Line Test



Software Test



Test System



Test Automation



Test Process Consulting



Test Training



2. TEST IN GENERAL

Static and Dynamic Code Analysis
in a Continuous Testing Process

Test in General

Importance of Testing



A system that does not work properly causes problems:

- ▶ Loss of image
- ▶ Damages (Personal injury or dead)
- ▶ Loss of money
- ▶ Loss of time



Common error causes:

- ▶ Human failure
- ▶ Complexity of code
- ▶ Complexity of infrastructure
- ▶ Modification of the technology
- ▶ System interactions

Usually a system should also meet contractual and legal requirements or standards.



Decrease of Risk + Quality Intensification = Confidence



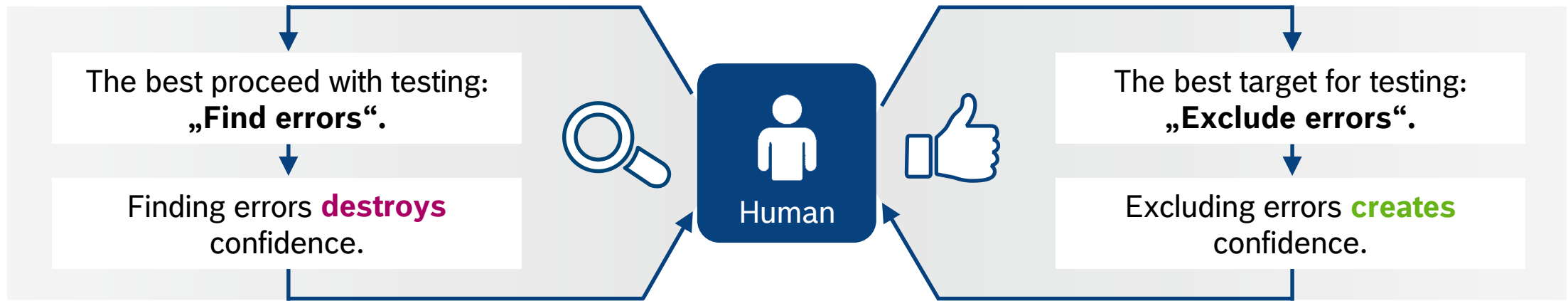
*„Due to the fault of a product someone killed, his body or health injured or something damaged, so **the product manufacturer is obliged to compensate the injured party for the resulting loss. (...)**“*

§ 1 Sec. 1 ProdhaftG, 1990, Liability

Test in General

Test Paradox

„The best way to **create** confidence is to **destroy** confidence!“



To achieve confidence, it is important that ...

- ▶ the developer gets an early and direct feedback concerning his development status.
- ▶ the developer can perform static and dynamic code analysis on his own machine.

3. CODE ANALYSIS

Static and Dynamic Code Analysis
in a Continuous Testing Process

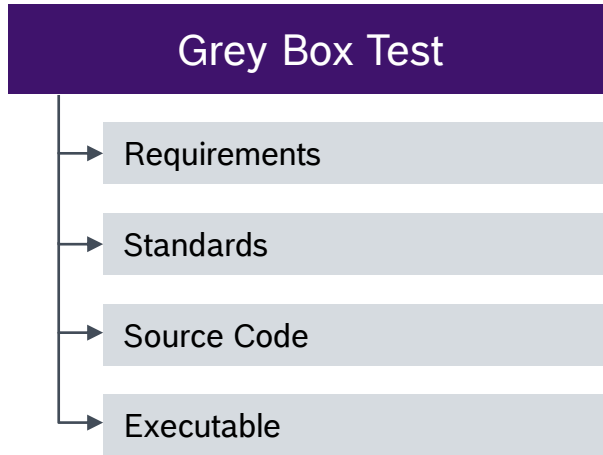
Code Analysis Static

White Box Test		Verification of	Review	Compiler	Analysis Tool	Review combined with Compiler and Analysis Tool
→	Requirements	Functional Requirements	■ yes	■ no	■ no	■ yes
	Coding Rules and Standards	Coding Rules and Standards	■ yes	■ no	■ yes	■ yes
		Lexically	■ yes	■ yes	■ yes	■ yes
→	Source Code	Syntax	■ yes	■ yes	■ yes	■ yes
		Semantic	■ yes	■ yes	■ yes	■ yes
		Complexity	■ yes	■ no	■ yes	■ yes
		Effort	■ high	■ mid	■ low	■ mid
		Benefit	■ mid	■ mid	■ mid	■ high

- ▶ The compiler-frontend includes lexical analysis (based on tokens), syntax analysis (checks grammar of programming language) and various kinds of semantic analysis (checks types, declarations, etc.).
- ▶ Analysis tools today can cover 80%-90% of coding rules and ~100% of a standard (e.g. MISRA).
- ▶ With the combination of compiler, analysis tools and review, the reviewer can focus on the functionality.

Code Analysis

Dynamic



Validation of	Test Tool (Black Box Test)	Analysis Tool (White Box Test)	Test Tool with included Analysis Tool (Grey Box Test)
Functional Requirements	■ yes	■ no	■ yes
Interfaces	■ yes	■ yes	■ yes
Code Coverage	■ no	■ yes	■ yes
Effort	■ mid	■ low	■ mid
Benefit	■ mid	■ low	■ high

- ▶ The test tool focus is to simulating and stimulating the function under test.
- ▶ The analysis tool is used to measure the code coverage according to the safety requirements from a standard (e.g. ASIL).
- ▶ Only in combination with both methods it is possible to show, whether there are some test cases missing or if there is really unreachable/dead code.

4. ANALYSIS TOOLS

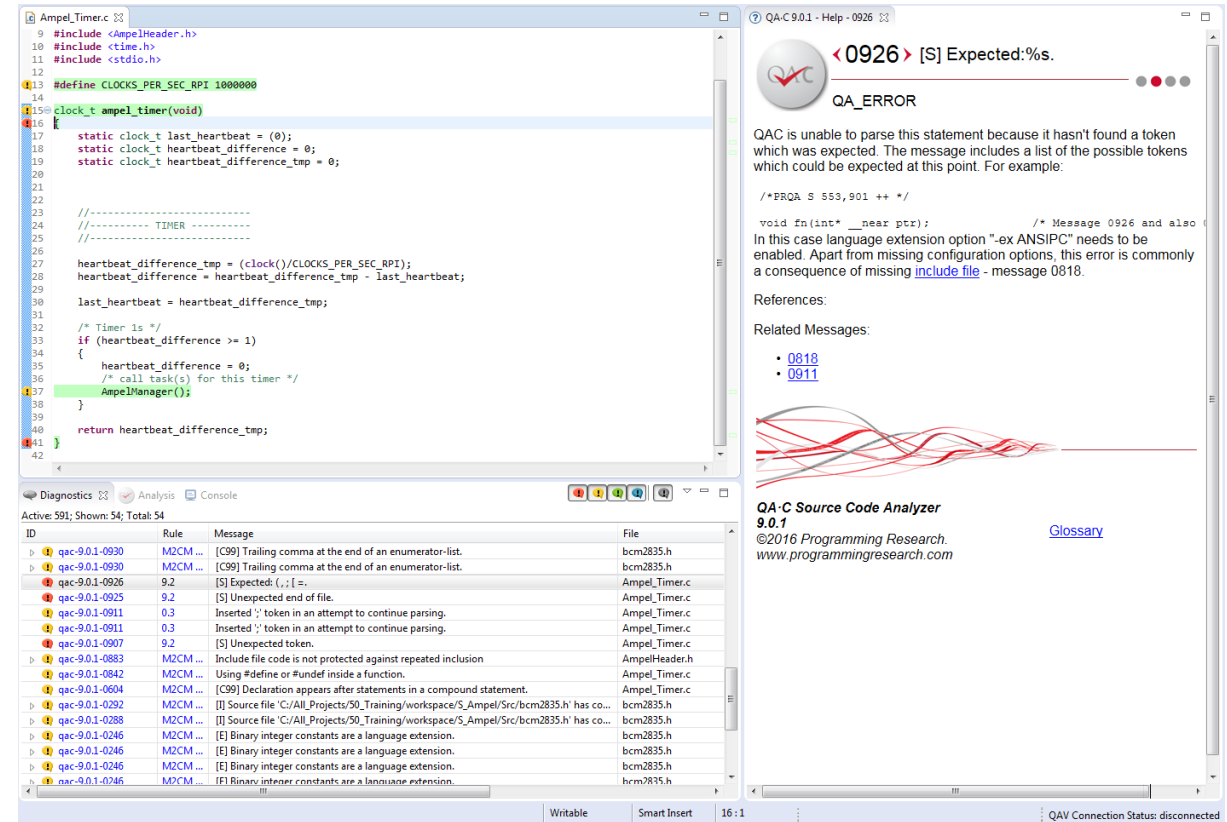
Static and Dynamic Code Analysis
in a Continuous Testing Process

Analysis Tools

QAC/QAC++

Static and Dynamic Code Analysis
in a Continuous Testing Process

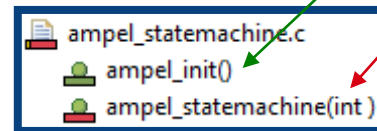
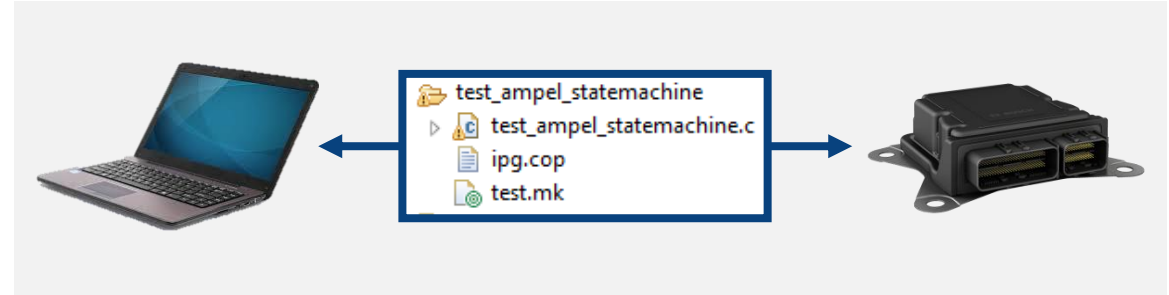
- ▶ Applicable for C and C++ Code.
- ▶ Working with multiple compilers.
- ▶ Applicable for Windows and Linux systems.
- ▶ PRQA-Framework plugin for eclipse integration.
- ▶ Compliance module add-ons for (e.g. MISRA, CERT).
- ▶ Command line Interface is applicable.
- ▶ Certified tools.



Analysis Tools Cantata

Static and Dynamic Code Analysis
in a Continuous Testing Process

- ▶ Works with C and C++ code.
- ▶ Tests can be performed on host and target.
- ▶ Provided as eclipse plugin for Windows and Linux systems.
- ▶ Includes code coverage analysis.
- ▶ Test scripts are based on the same programming language as the developed source code.
- ▶ Command line interface is applicable.
- ▶ Requirement gateway for bidirectional traceability.
- ▶ RTTR test script converter.
- ▶ Certified tool.



```
void ampel_init(){
    init_leds();
    set_led(A1_LED_RED, ON);
    set_led(A1_LED_YELLOW, OFF);
    set_led(A1_LED_GREEN, OFF);
    set_led(A2_LED_RED, ON);
    set_led(A2_LED_YELLOW, OFF);
    set_led(A2_LED_GREEN, OFF);
}

int ampel_statemachine(int state){
    int returnvalue = -1;
    // ...
}
```

Summary by	EXECUTED	Overall	Statistics				
Coverage type	INFEASIBLES	(wavg)	avg /	min /	max /	dev /	num
entry point	0	50.0%	50.0%	0.0%	100.0%	70.7%	2
statement	0	8.5%	50.0%	0.0%	100.0%	70.7%	2
decision	0	0.0%	50.0%	0.0%	100.0%	70.7%	2
boolean eff.	0	100.0%	100.0%	100.0%	100.0%	0.0%	2

EXPORT_COVERAGE: coverage data written to "test_ampel_statemachine.cov"

=====

= Test Finished: Thu Mar 26 14:43:30 2015 =

=====

= Test	Script	Checks	Checks	Checks	Call Seq	TEST
= Case	Errors	Failed	Passed	Warned	Failures	RESULT
= test_ampel_init	0	0	0	1	0	PASS
= Other	0	3	1	2	0	>> FAIL
= TOTALS	0	3	1	3	0	>> FAIL

=====

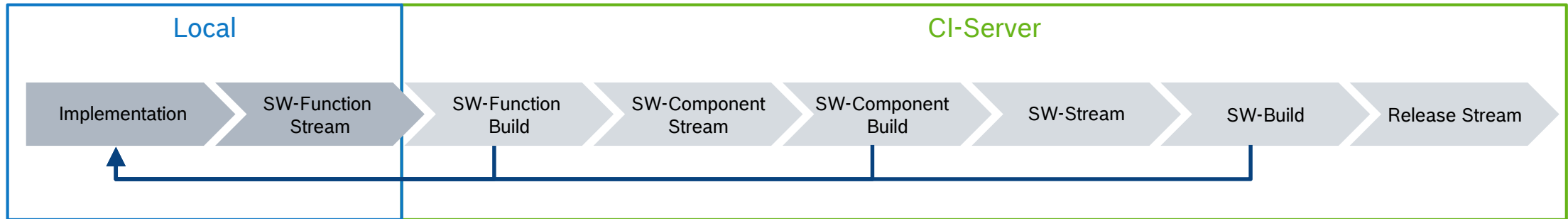
5. PROCESSES

Static and Dynamic Code Analysis
in a Continuous Testing Process

Processes

Continuous Integration (CI)

- ▶ CI is the process joining SW-Functions to a SW-Component and SW-Components to a complete SW.
- ▶ The target of CI is to detect integration problems in an early phase of the integration process.
- ▶ Therefore a fully automated build chain is required.



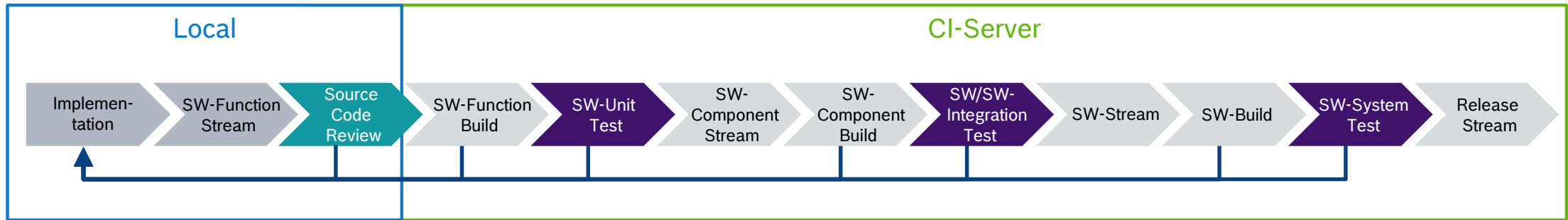
Benefits:

- ▶ Avoids last-minute integration chaos at release dates.
- ▶ Frequent code check-ins push the developers to create a modular code with less complexity.
- ▶ Continuous available builds for testing, demo or release purposes.

Processes

Continuous Testing (CT)

- ▶ CT is the process executing automated tests between the CI steps.
- ▶ The target is to detect rule deviations and interface & functional problems in an early phase.
- ▶ Therefore a fully automated test framework is required.



Benefits:

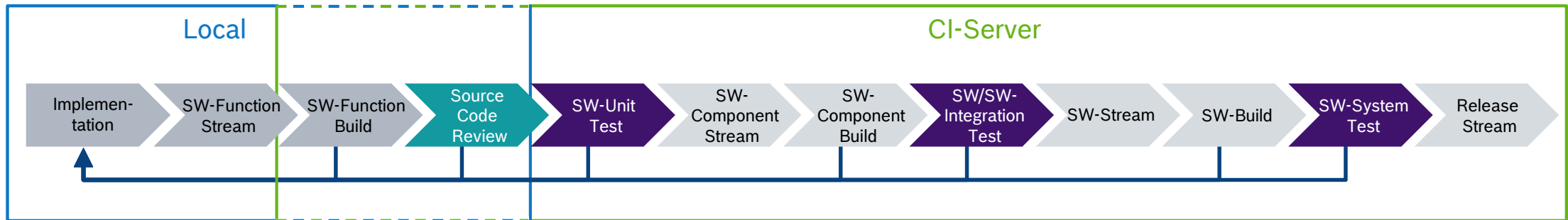
- ▶ Immediate feedback regarding system-wide impact of changes.
- ▶ Leads developer to write cleaner and testable code.
- ▶ Architecture and requirements engineering is test driven.

6. IMPROVEMENTS

Static and Dynamic Code Analysis
in a Continuous Testing Process

Improvements CT Process

- ▶ Do the source code review after the function build.
- ▶ Add check-questions for the compiler warnings and errors to the review check list.



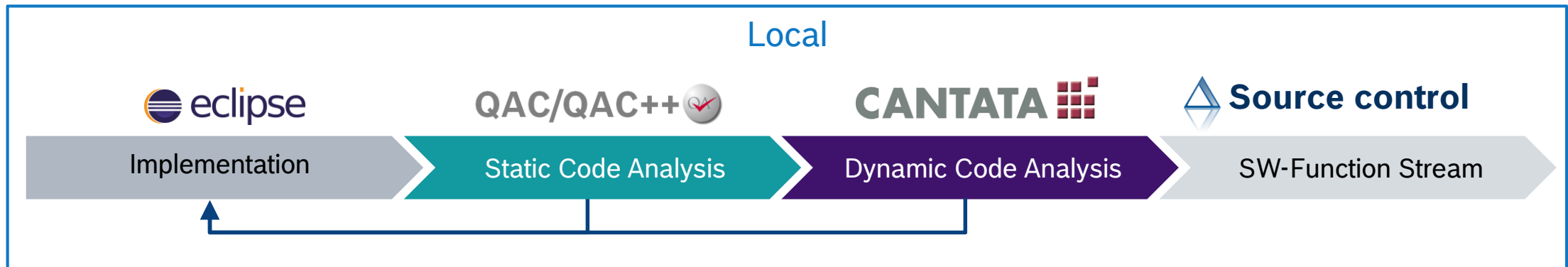
Benefits:

- ▶ Avoids unneeded review effort, when function build fails.
- ▶ Solving all compiler warnings and errors leads to a stable build.

Improvements

Local Environment Side (1)

- Eclipse environment, that includes QAC/QAC++ and Cantata as plug-in.

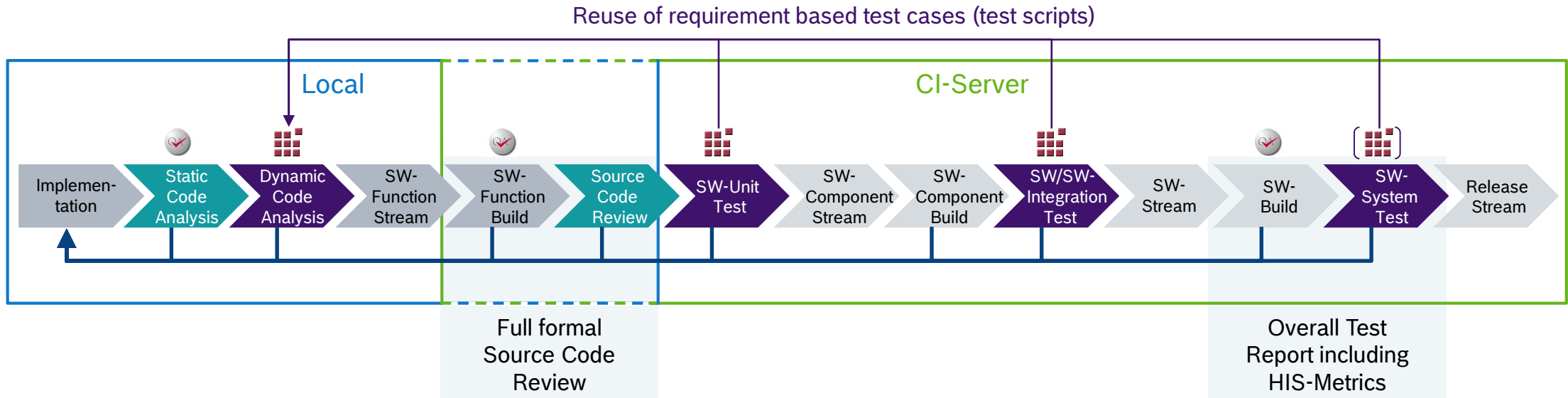


Benefits:

- Developer and tester work in the same environment.
- Developer can perform analysis and tests on his own machine to ensure a basic quality, before the SW check-in to the SW-Function Stream takes place.

Improvements Local Environment Side (2)

- Add Local environment change to CT environment.

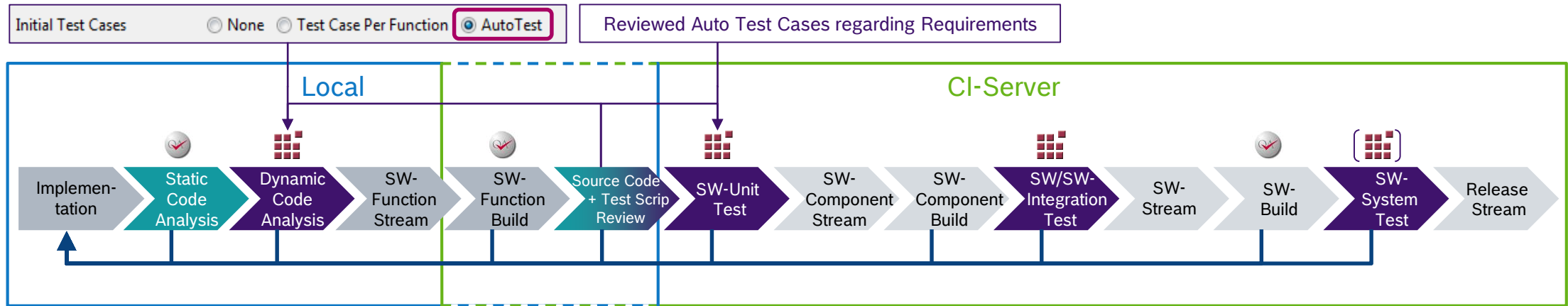


Benefit:

- Normally all findings from the static analysis are fixed before the formal source code review takes place.

Improvements Using Auto Test Case Generator

- ▶ Using auto test case generator from Cantata for spot check and low level test case development.



Benefits:

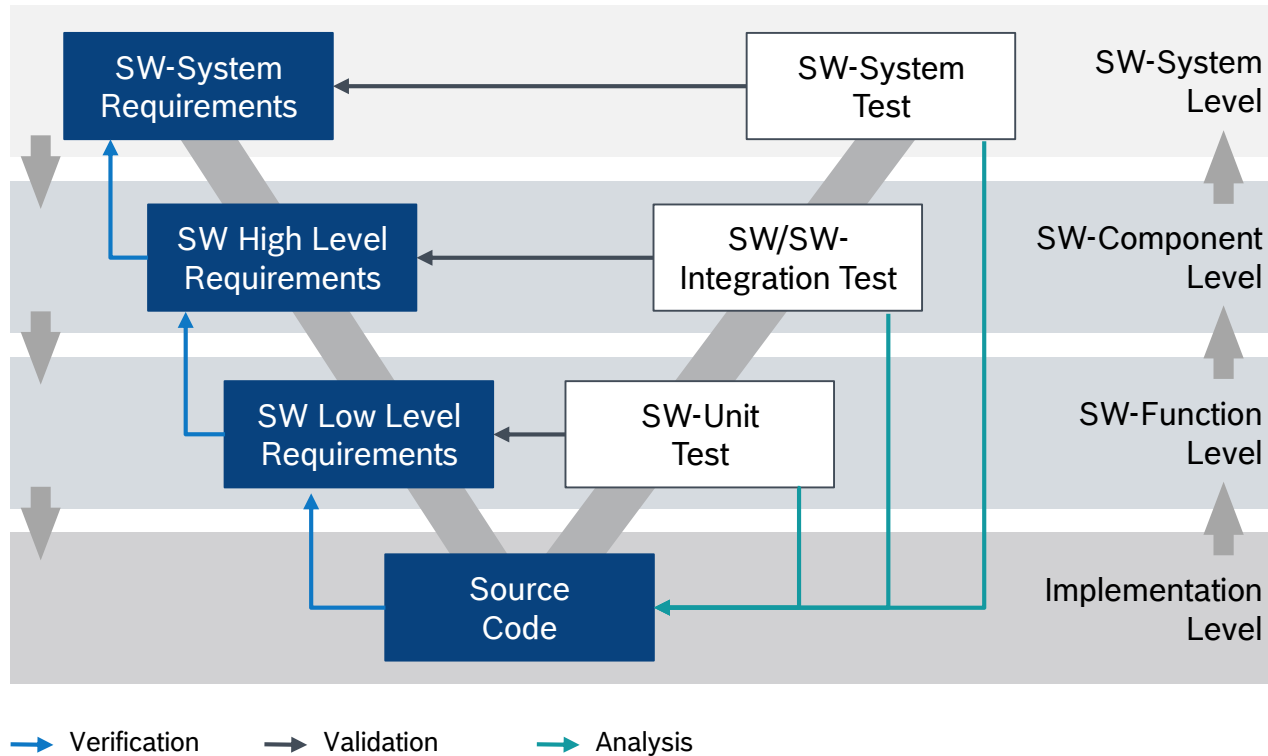
- ▶ Checks if each path can be executed.
- ▶ Provides test cases and coverage values.
- ▶ Reduces effort through functional test case development.

7. REAL PROJECTS

Static and Dynamic Code Analysis
in a Continuous Testing Process

Real Projects

Reachable Code Coverage



Code Coverage is hard to reach

Reachable Code Coverage **50%–80%**

Reachable Code Coverage **>95%**

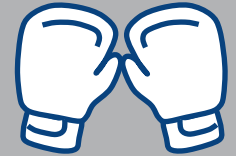
100% code coverage is not always possible/payable because of ...

- ▶ hardware related SW parts.
- ▶ used compiler defines to distinguish between different variants.
- ▶ selected coverage level (e.g. MCDC).

Real Project Testing

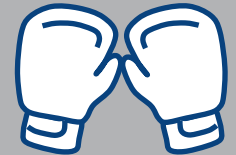
Challenge

- ▶ A lot of projects start with testing late.
- ▶ What happens then? → A fast and big ramp up of the test team.
- ▶ Why? → To get all testing artefacts for the customer.



Challenge

- ▶ Usually the requirement engineering is not done completely, so that in most cases low level requirements for unit tests do not exist.
- ▶ What happens then? → The required test cases will be derived from the implementation.
- ▶ Why? → The standards and customer require SW-Unit Tests.



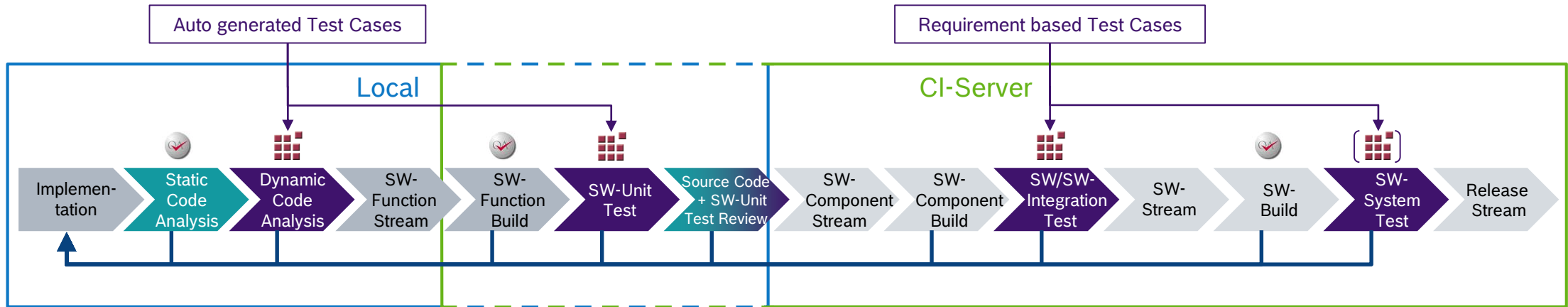
RESULT

- ▶ This causes effort with low benefit for the product quality only.
- ▶ Waste a lot of time and money that the project preferable can use for other important things.



Real Projects Idea

- Using auto test case generator for spot check and low level tests without test case review in between.



Benefits:

- Checks if each path can be executed.
- Provides test cases and coverage values.

8. TEST EFFORT CALCULATION

Test Effort Calculation Using Static Metrics

Static Code Analysis Metrics	Dynamic Code Analysis Effort
Cyclomatic Complexity (STCYC)	Indicates the minimum number of test cases to reach full branch coverage.
Number of called functions (STCAL)	Provides an estimation about the number of stubs that have to be generated.
Number of calling functions (STM29)	Reflects how many requirements could affected.
Number of function parameters (STPAR)	Number of function parameters is related to the complexity of test vectors.
Estimated static program paths (STPTH)	As higher the value, as more complex it is to be fully tested.
Number of Statements (STST3)	Can be used to estimate the required time for testing (experience value 60LOC/day and person).
Call Graph	Helps to identify Stubs/Wrappers.



By combining Static and Dynamic Code Analysis in a Continuous Testing Process, **effectively sustained quality improvement** can be achieved.

THANK YOU

Static and Dynamic Code Analysis
in a Continuous Testing Process

Matthias Schmidt

Test Management, Verification and Validation (BEG-VS/EEA4)

Bosch Engineering GmbH | Postfach 13 50 | 74003 Heilbronn | GERMANY | www.bosch-engineering.com

Tel. +49(7062)911-7481 | Mobil +49(172)5804670 | Fax +49(7062)911-6401 | Matthias.Schmidt6@de.bosch.com