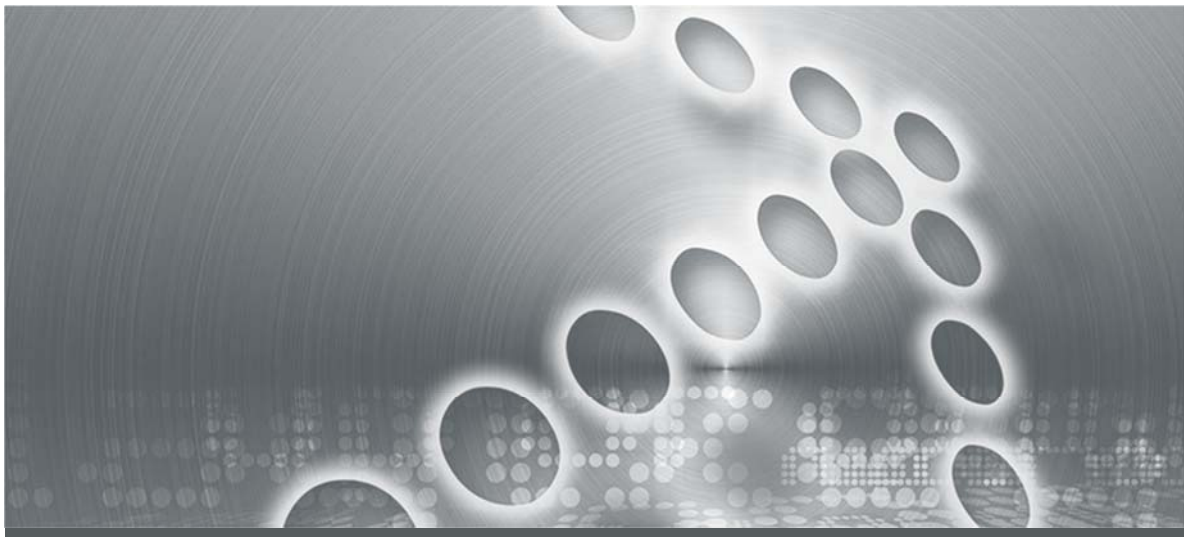


# Static Code Analysis
# Tools, Evaluation, Practical Usage

Christof Ebert and Belia Romero

Workshop Software Measurement, Kaiserslautern 2017

V1.0 | 2017-12-06



## Vector Consulting Services

- ...Experts for product development, product strategy and IT
- ...Interim management
- ...Global presence
- ...Training on Agile, require-ments, security, safety etc.
- ...Part of Vector Group with 2000 employees and well over 400 Mio. € sales
- ...Growing and thus continuously looking for talent

**www.vector.com/consulting**

VECTOR

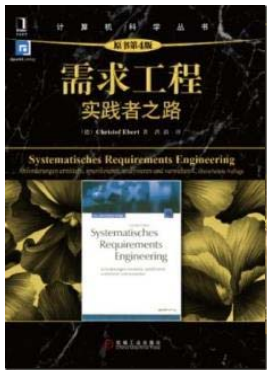
## Author: Christof EBERT

Christof Ebert is managing director at Vector Consulting Services.

He supports clients to improve product strategy and product development and to manage organizational changes.

Prior to that, he held senior management positions for ten years at Alcatel, most recently global director for software technologies.

A trusted advisor for companies around the world, member of industry boards, and professor at the University of Stuttgart and Sorbonne in Paris, Dr. Ebert authored several books.




christof.ebert@vector.com
www.vector.com/consulting
@ChristofEbert

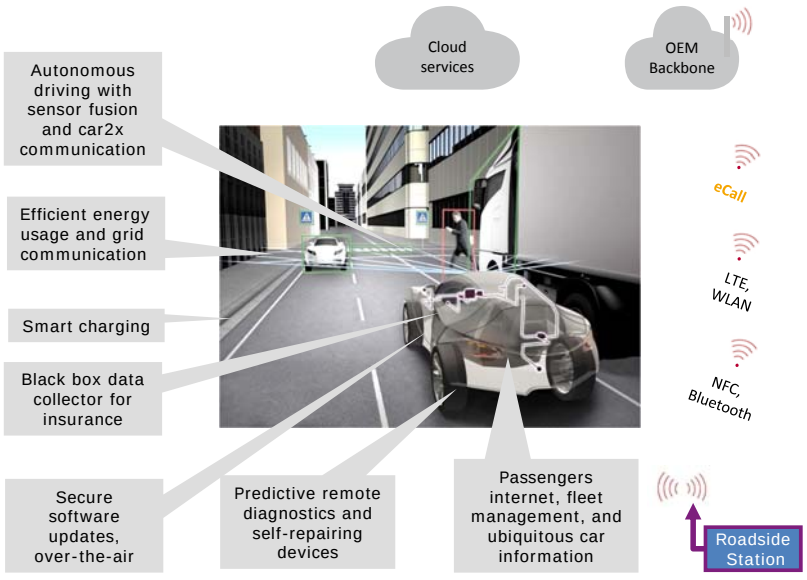
 



VECTOR

## Challenge: Critical Systems

- Convergence of IT and embedded systems: **Competence lack**
- Uncontrolled software complexity growth: **Security meets safety**
- DevOps and frequent updates: **Agile feature activation**
- Failures: **Liability risks**




**Strong need for continuously sustaining high level of code and design quality**

VECTOR

# Common Wisdom: Compiled Code is NOT Good Enough

```
…
int main ( ) {
    int yourNum;
    cout << "Please enter an integer value: ”;
    cin >> yourNum;
    cout << "The value you entered is << yourNum;
    cout << " and its double is ” << yourNum*2;
…
```

✓ Works
✓ Compiles
✗ Vulnerable
✗ Overflow

```
…
#define CHAR_BUFFER_SIZE 10
int main ( ) {
        int yourNum;
        cout << "Please enter an integer value: ";
        cin.width(CHAR_BUFFER_SIZE);
        cin >> yourNumber;
        cout << "The value you entered is " << yourNum;
       if (data < (CHAR_BUFFER_SIZE /2)) {
            int doub= yourNum*2;
            cout << " and its double is " << doub << ".\n";
        …
```

✓ Works
✓ Compiles
✓ Satisfies security criteria
✓ Standard conform



VECTOR

# What is Static Analysis?

## Method

- Static analysis is a verification method
- For efficiency and effectiveness primarily performed by tools

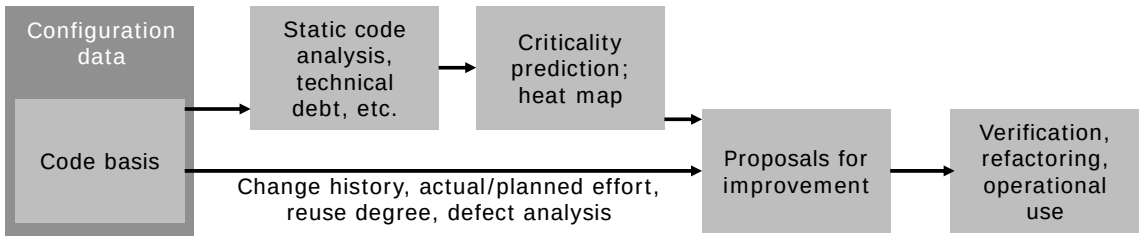
## Application

- Coding standards such as „only 1 return statement per function“
- Code complexity
- Typical defects, e.g. Type castings, initializations of variables
- Defined rule sets, e.g. MISRA-C for safety

VECTOR

# Static Analysis Tools

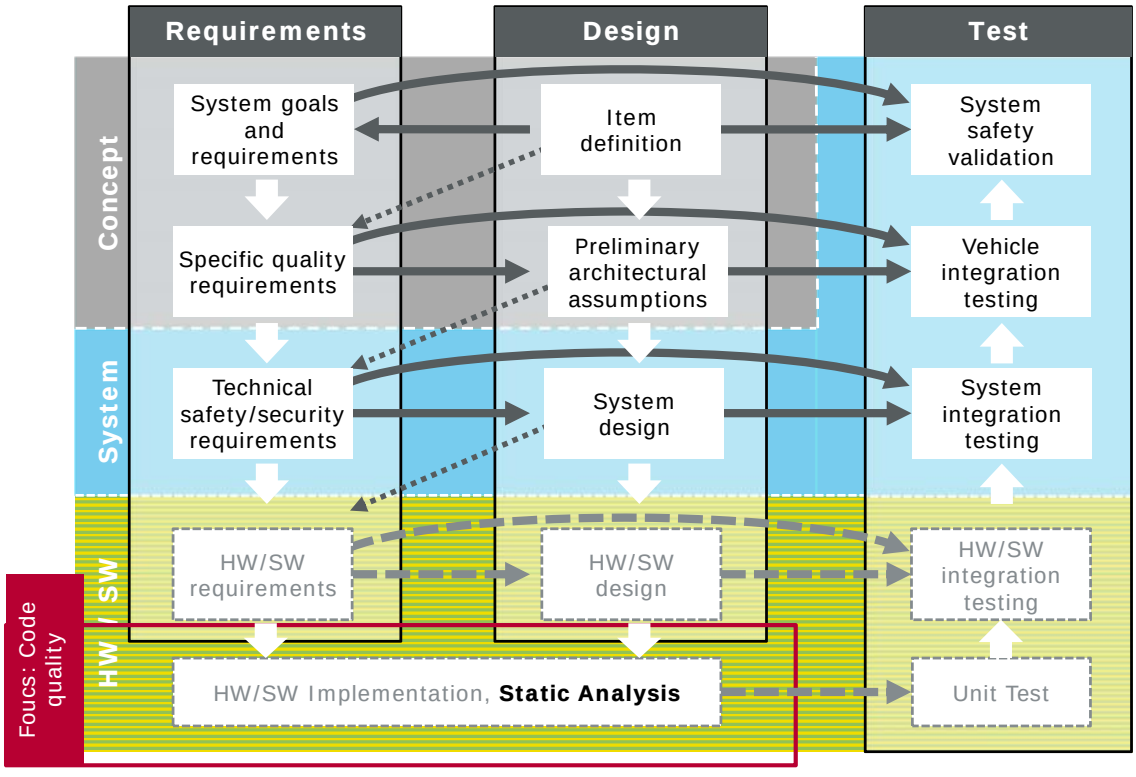





VECTOR

# Static Analysis in the Life-Cycle



VECTOR

## Typical Static Analysis Tools Checks – and their Findings

- Style Checking

```
//FIX: Function fails when value equals 0
String a = „Hallo Welt„;
```

- Type Checking

```
int theNumber = 3.53;
```

- Program understanding

```
if (false) {
  //DEADCODE
}
```

- Bug Finding

```
//Double free
if(1<5){
  free(myVar);
}
free(myVar);
```

- Security Reviews

```
password = „MYPASSWORD“;
privateKey= „MYKEY“;
```

**Many defects result from insufficient oversight and disturbances – which are later not anymore found with compile and test**






VECTOR

## Challenges with Static Analysis Tools

**Usability**

- High amount of false positives and warnings
- No guidance what and how to improve

**Analysis**

- No consideration of different states of the program (i.e. no execution, simulation)
- True positives are concealed between all the findings
- False positives rate is greater than the true positives rate
- Fault indications and impacts are difficult to understand
- Lack of remedial actions
- Defect priority is not configurable

**Software engineers are reluctant to the use of Static Analysis Tools**

VECTOR

## Our Motivation

- Assess and improve the overall quality of software (finished from suppliers, or in progress internally)
- Visualize code quality
  - Critical sections of the code
  - Predefined acceptance criteria
- Use results to identify critical code (segments) in early stages of the development process
  - Reduce rework costs
  - Improve quality
- Facilitate productive usage of static code analysis tools
- Evaluate performance of static analysis tools, and guide the selection of a tool


**Code quality assessment**

**With predefined acceptance / quality criteria**

**Hypothesis: Aggregation of results from different static analyzers delivers a more reliable and meaningful output**






VECTOR

## Static Analysis Tools Selection

- Selection of appropriate tools after a market research of the available COTS static analysis tools.
- Analyzed properties:
  - Monitored Process: Compilation, Compilation+Build
  - Languages
  - Standards Compliance: MISRA, CERT,...

VECTOR software
coverity™
MathWorks®
axivion
stopping software erosion
AbsInt
sonarqube
FORTIFY
PC-LINT
C/C++
SWAMP
SQUORING Technologies

VECTOR

# Static Analysis Tools Overview

| Characteristics | Astrée | Coverity | Klocwork | Polyspace | PCLint |
|---|---|---|---|---|---|
| **Coding Standard** | MISRA 2012. Partially: SEI CERT C, CWE | MISRA 2012 | CWE MISRA 2012 | MISRA 2012 | Lint Analysis for MISRA support |
| **Supported Languages** | C | C, C++, Java, C#, JavaScript, Objective-C | C, C++, C# | C, C++ | C, C++ |
| **Free Demo** | Yes | Yes | Yes | Yes | Yes |
| **Installation** | Local | Cloud (demo) | Local | Local | Local |
| **Extra Plugins** | No | Yes | No | No | Yes |
| **Own Rules** | Yes | No | Yes | Yes | No |
| **Monitors** | Compilation | Compilation + Build | Compilation + Build | Compilation + Build | Compilation |
| **Usability** | 10 | 6 | 10 | 8 | 6 |
| **Documentation** | Excellent | Good | Excellent | Excellent | Good |
| **HW requirements** | Low | High | Medium | High | Low |

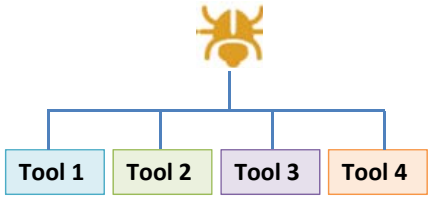


VECTOR

# Static Analysis Tools Performance Evaluation (1/2)

- Empirical study of mainstream COTS static analysis tools
- Standardized Juliet library of code with known defects
  - 114 CWE Test scenarios, each with different variations and complexities
  - Defect types: Overflow, Dead Code, Division by Zero, etc.
  - Approximately 64.000 Test Cases available

```
void CWE570_Expression_Always_False__global_01_bad()
{
    /* FLAW: This expression is always false */
    if (globalFalse)
    {
        printLine("Never prints");
    }
}
```




Juliet: Standardized test scenarios with defective code
CWE: Common Weakness Enumeration
NIST: National Institute of Standards and Technology
SAMATE: Software Assurance Metrics and Tools Evaluation

VECTOR

# Static Analysis Tools Performance Evaluation (2/2)

## Confusion Matrix

Tool Analysis:

| Defect exists | Defect does not exist |
|---|---|
| TP | FN |
| FP | TN |

Real Classification:

← Defect exists

← Defect does not exist

TP: True Positives
TN: True Negatives
FP: False Positives
FN: False Negatives

## Sensitivity

How good a tool can identify a defect (also "Recall)

$$\frac{TP}{TP+FN}$$

## Precision

How good a tool avoids false positives

$$\frac{TP}{TP+FP}$$

**Our focus is first on effect: Sensitivity and then on efficiency and usability: Precision**



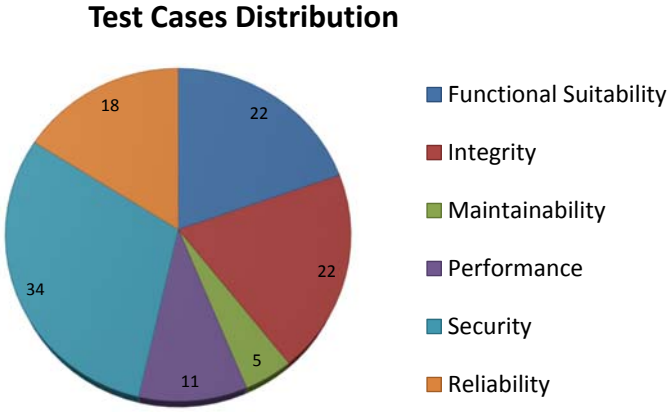
VECTOR

# Defects Classification

6 different error categories and 13 subcategories were defined that comprise the most common programming defects.

Defects were classified by their impact on the code, e.g.

- Dead Code > Maintainability
- Double Free > Functional Suitability
- Heap Inspection > Security



VECTOR

# Static Analysis Tools Performance Results

| | | Sensitivity (%) | | | |
|---|---|---|---|---|---|
| Test Cases | Total Cases | Tool 1 | Tool 2 | Tool 3 | Tool 4 |
| Functional Suitability: Crash | 15.534 | 43,54% | 56,09% | 0% | 18,40% |
| Maintainability | 546 | 23,26% | 19,78% | 44,14% | 84,25% |
| Security: Execute unauthorized code | 6.102 | 75,71% | 9,10% | 0% | 17,80% |
| Security: Assume Identity | 522 | 0% | 0% | 0% | 0,77% |
| Reliability: Fault Tolerance | 236 | 7,63% | 0% | 53,59% | 27,12% |
| ... | | | | | |

**One static analysis tool on its own cannot detect all relevant defect types**



VECTOR

# Case Study: Automotive OEM

## Challenge

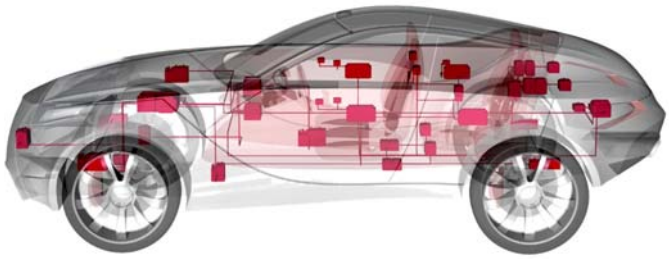
- Improvement of code compliance of a complex safety-critical automotive system development project (> 10 software suppliers)

## Vector contribution

- Evaluation of MISRA deviations in several safety-critical automotive SW projects
- Analysis and assessment of the deviations
- Establishing acceptance criteria for delivered code
- Support in improvement of coding standards and stepwise migration to MISRA-C compliant code

## Result

- Improvement of average MISRA-C compliance from 66 % of the rules to 85 % of the rules within 12 months



VECTOR

# Case Study: Automotive Tier-1

### Challenge

- MISRA-C compliance of Vector CANopen slave

### Vector contribution

- Assessment of MISRA-C rules, update of coding guidelines
- Removed several potentially safety-critical MISRA-C rule violations
- Integration of systematic, tool-based static analysis early in the development lifecycle (akin to unit testing)

### Result

- High MISRA-C compliance (136 rules out of 141)
- Assurance of “legitimate” deviations for future safety cases
- Approx. 40% less effort than in comparable project where static analysis was performed later in the lifecycle



VECTOR

# Summary and Guidance

- Static code analysis is a useful metric for source code quality and understandability – and as such a predictor for security, safety, reliability risks.
- Improvement of identified critical code based on rule compliance helps to detect and remove defects.
- Low usability of COTS analysis tools. Analysis tools will usually report far too many non-compliances when exercised on code that has not been developed according to underlying rules.
- Static code analysis usage and necessary code improvements require high effort in the range of person months per software component.

**Vector recommendations:**

- Use static analysis for early code quality improvement, and for supplier evaluation/acceptance
- Handle like unit testing: do it early and automated.
- Apply expert know-how to avoid unnecessary efforts.

Thank you for your attention.
Contact us – We are glad to support you.

**Passion. Partner. Value.**

## Vector Consulting Services

www.vector.com/consulting
consulting-info@vector.com
Phone: +49 711 80670-0