

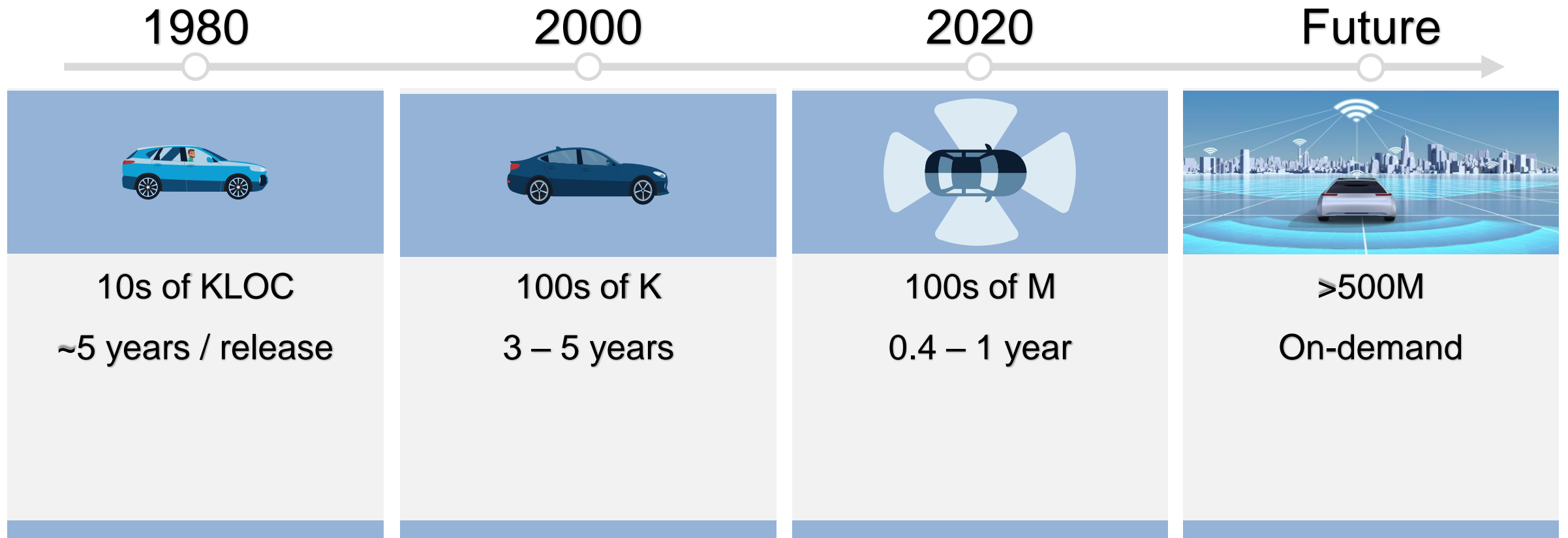
MathWorks
**AUTOMOTIVE
CONFERENCE 2022**
North America

**Faster Software Delivery with Polyspace
in Your Software Factory**

Patrick Munier, Software Development Director, MathWorks

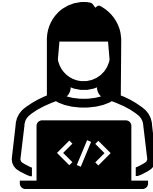


Software Lines of Code and Delivery Frequency are increasing

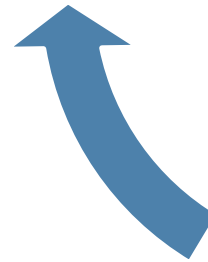


Software Development in Embedded systems is facing major challenges

Size and complexity
C++, AUTOSAR
Staffing SW developers



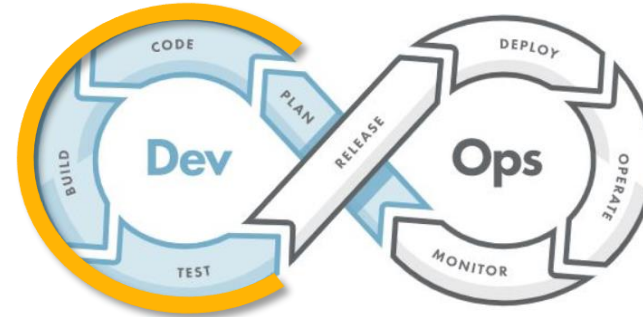
Faster delivery



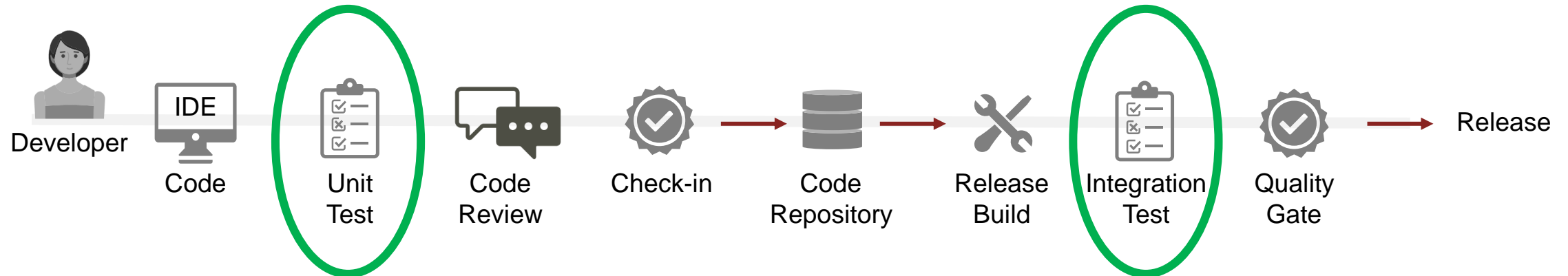
Safety and Cybersecurity
ISO 26262, ISO/SAE 21434



Software Factory help solve these challenges



DevOps is a set of practices that combines software development (Dev) and IT operations (Ops)

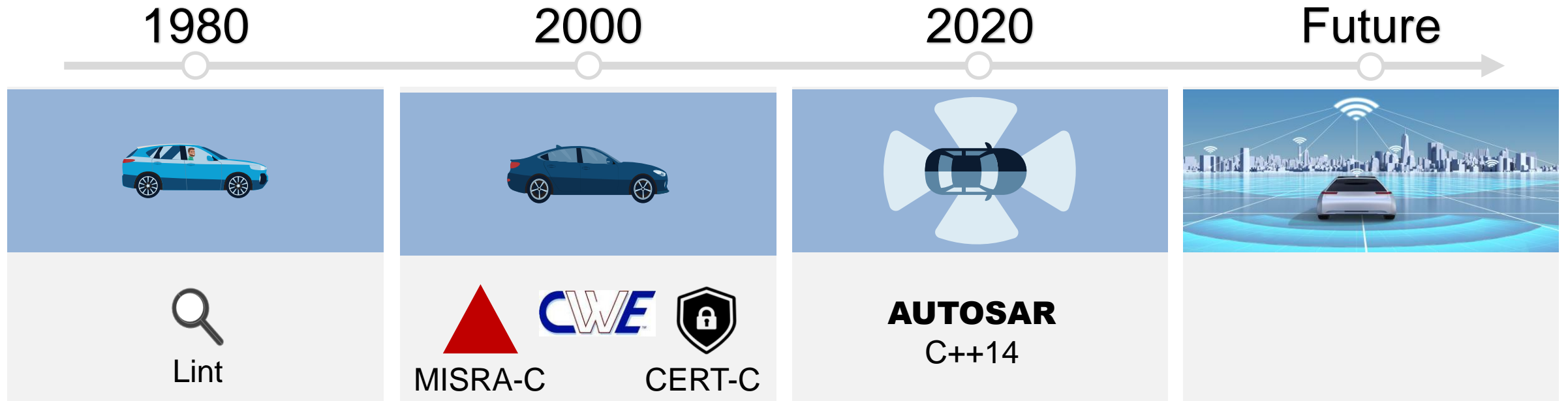


1. Repeatability

2. Faster delivery

3. Higher quality

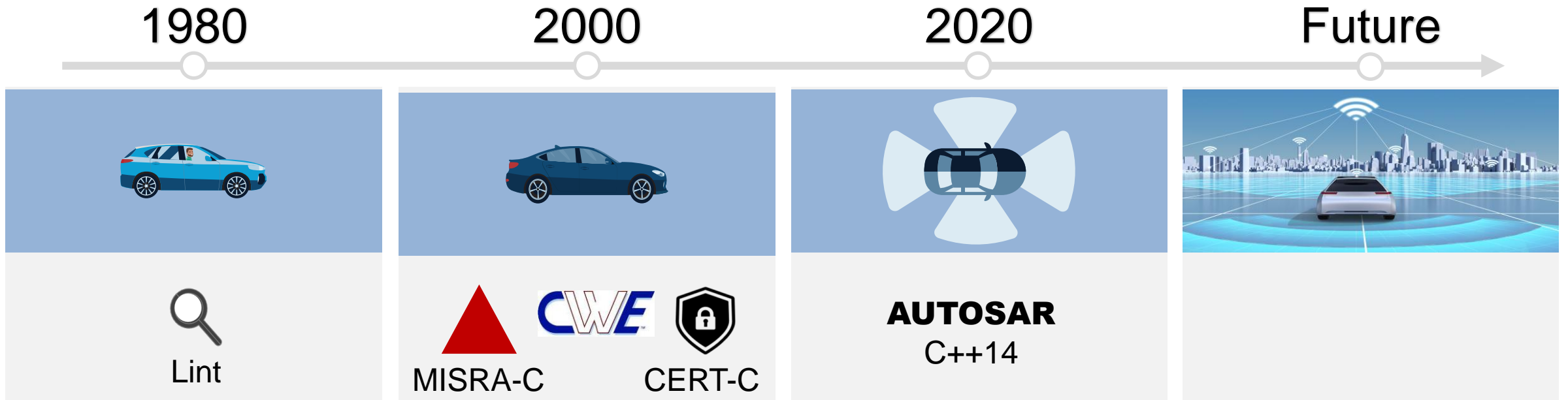
Static Analysis analyzes code without the need of tests to execute



Static Analysis is the most effective way to check compliance with Coding Rules

5.3 Compliance

In order to ensure that code complies with all of the MISRA C guidelines, a compliance matrix should be produced. This matrix lists each guideline and indicates how it is to be checked. For most guidelines, the easiest, most reliable and most cost-effective means of checking will be to use a static analysis tool or tools, the compiler, or a combination of these. Where a guideline cannot be completely checked by a tool, then a manual review will be required.



Static Analysis finds various kinds of defects



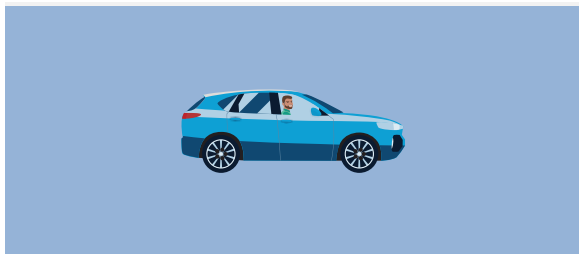
1980

2000

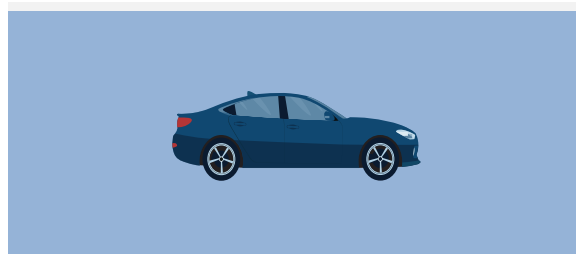
Bug Finder Analysis

Find defects custom

- Defects
 - Numerical
 - Static memory
 - Dynamic memory
 - Data flow
 - Resource management
 - Programming
 - Concurrency
 - Data race (Impact: High)
 - Data race through standard library function call (Impact: High)
 - Data race on adjacent bit fields (Impact: High)
 - Deadlock (Impact: High)



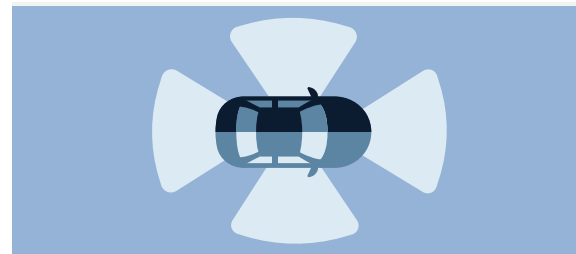
Lint



MISRA-C



CERT-C



AUTOSAR

C++14

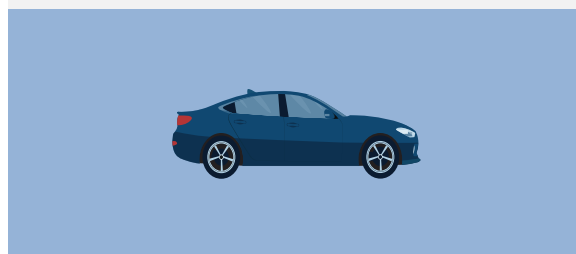
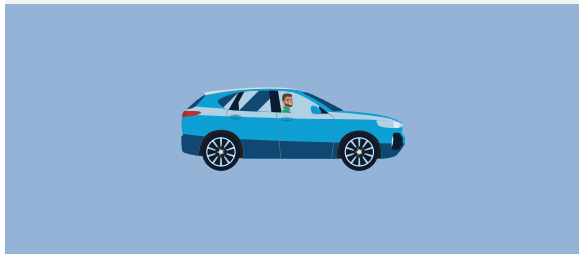


Static Analysis proves the absence of critical runtime errors



1980

2000



Lint



MISRA-C



CERT-C

Green: reliable
safe pointer access

Red: faulty
out of bounds error

Gray: dead
unreachable code

Orange: unproven
may be unsafe for some conditions

Purple: violation
MISRA-C/C++ or JSF++
code rules

Range data
tool tip

```
static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;

    for (i = 0; i < 100; i++) {
        *p = 0;
        p++;
    }

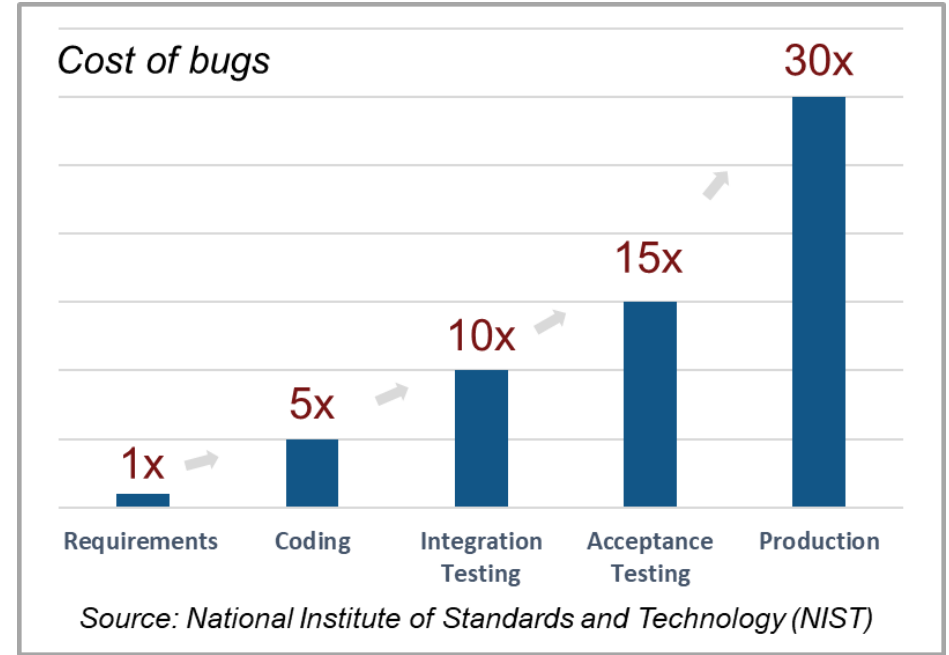
    if (get_bus_status() > 0) {
        if (get_oil_pressure() > 0) {
            *p = 5;
        } else {
            i++;
        }
    }

    i = get_bus_status();

    if (i >= 0) {
        *(p - i) = 10;
    }
}
```

variable 'i' (int32): [0 .. 99]
assignment of 'i' (int32): [1 .. 100]

Static Analysis should be used early in the development process

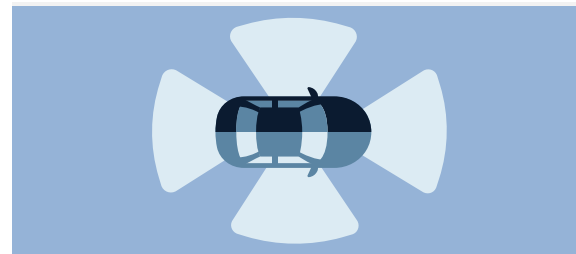
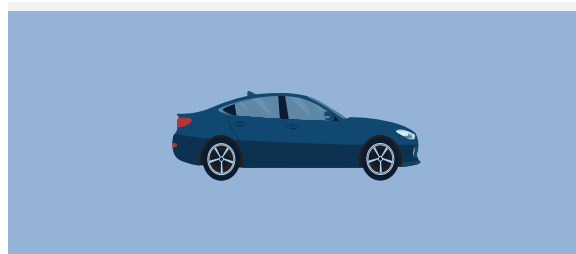
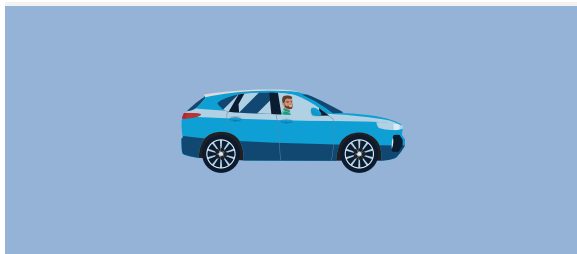


1980

2000

2020

Future



Lint



MISRA-C

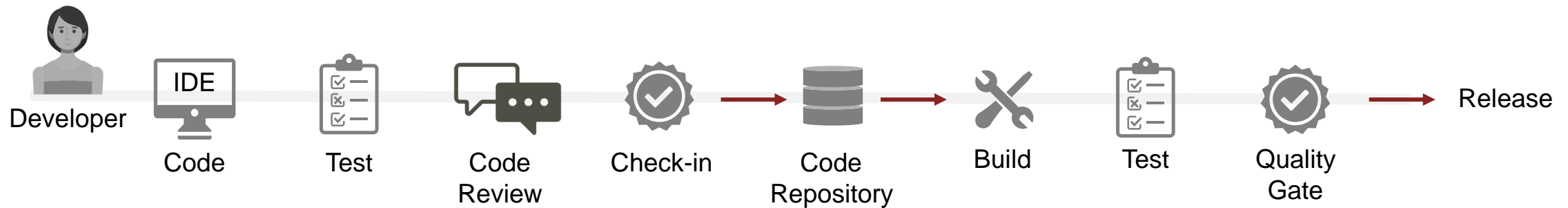


CERT-C

AUTOSAR

C++14

Static Analysis in Software Factories provides many benefits



1. Repeatability



Find errors
and verify
coding standards
consistently

2. Faster delivery



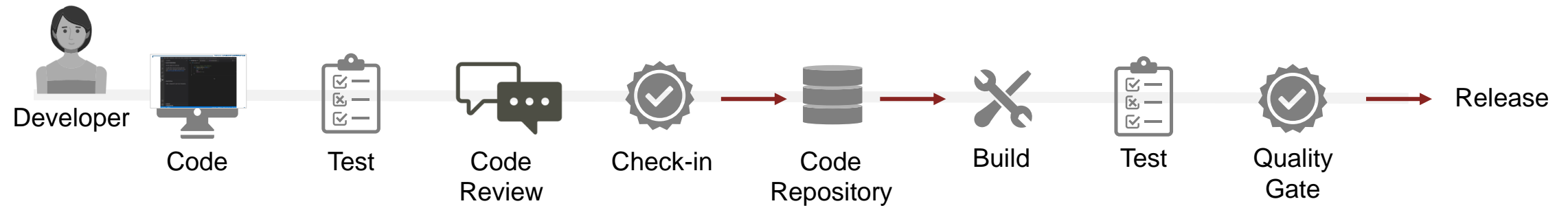
Find defects **early**
Reduce testing
Upskill Developers

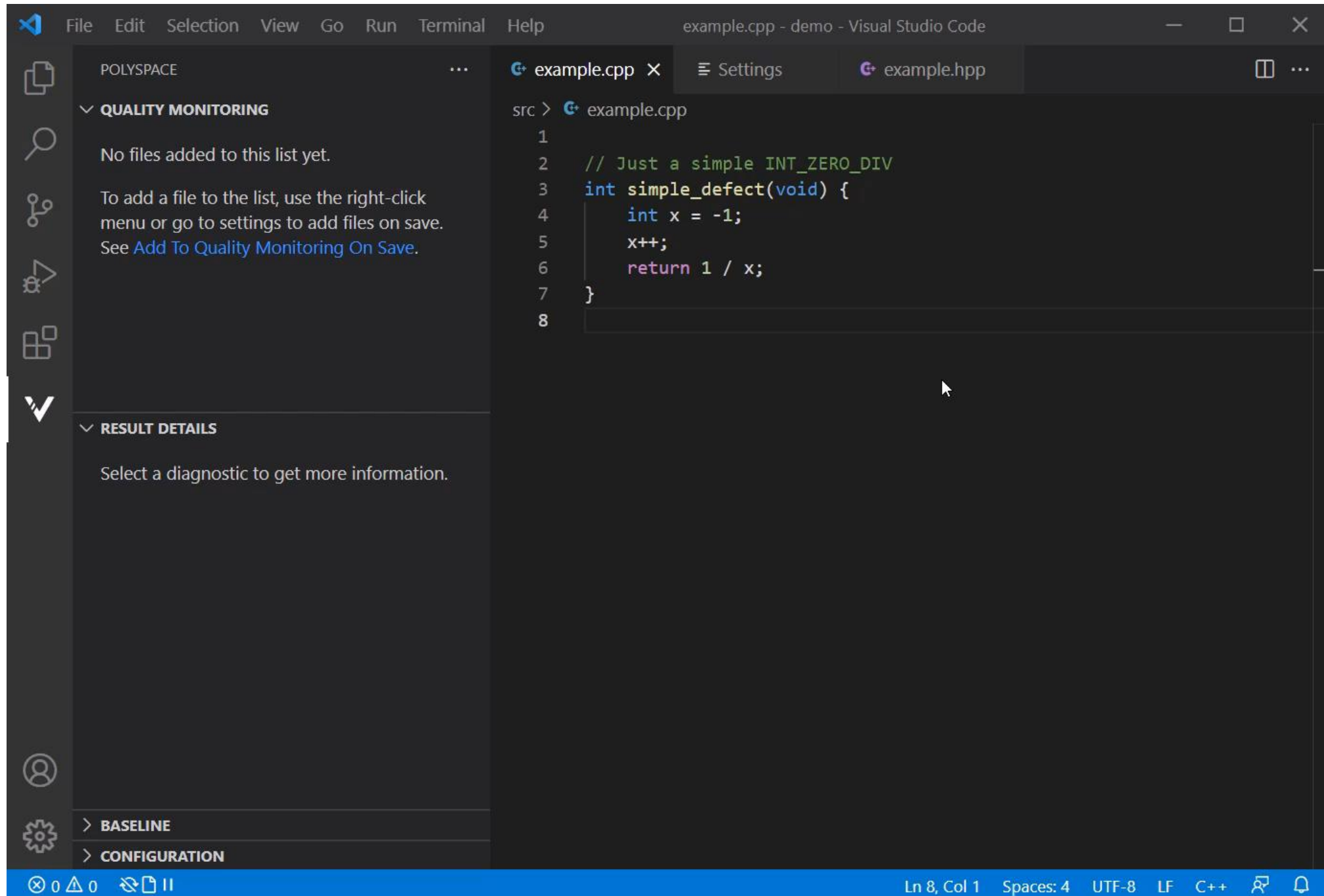
3. Higher Quality



Quality gates
Including the **absence** of
critical **runtime errors!**

Static Analysis can be integrated as early as in the IDE





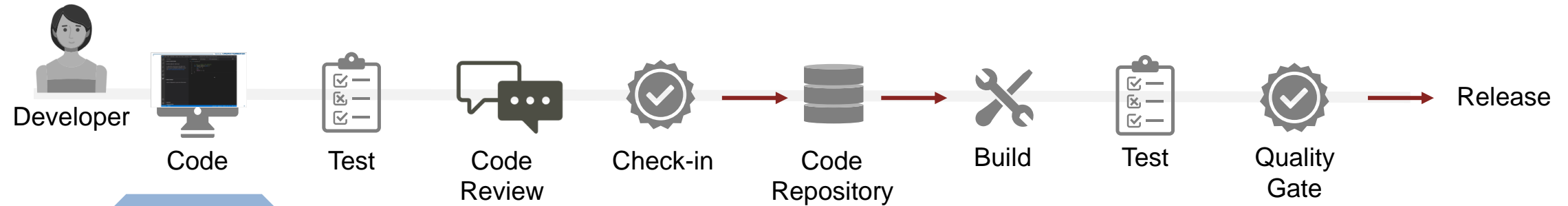
Visual Studio Code interface showing a C++ file named `example.cpp` with a simple defect. The editor is in dark theme. The left sidebar shows the **QUALITY MONITORING** and **RESULT DETAILS** panels. The main editor area shows the code for the `simple_defect` function.

```
src > example.cpp
1
2 // Just a simple INT_ZERO_DIV
3 int simple_defect(void) {
4     int x = -1;
5     x++;
6     return 1 / x;
7 }
8
```

The **QUALITY MONITORING** panel indicates that no files have been added to the list yet. The **RESULT DETAILS** panel prompts the user to select a diagnostic to get more information.

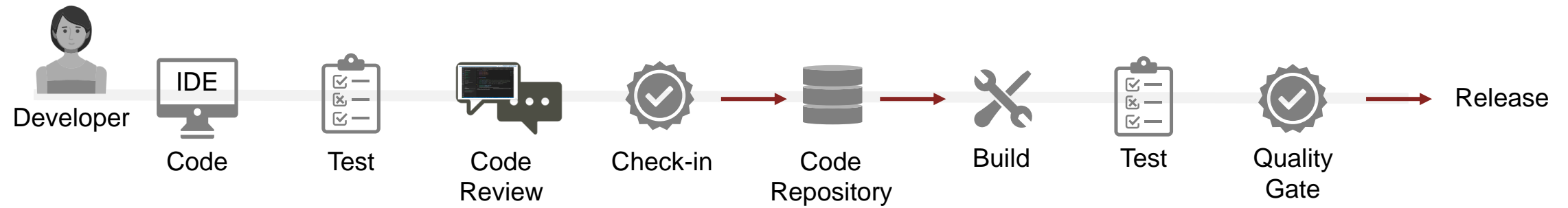
The status bar at the bottom shows: Ln 8, Col 1 | Spaces: 4 | UTF-8 | LF | C++

Polyspace as You Code finds defects early and helps train developers



- Find and fix defects as soon as code is written
- Instant feedback, enables continuous training

Static Analysis can be a verification bot in the Code Review



The screenshot shows the Visual Studio Code interface for a C project named 'pi_main.c'. The editor displays the following code:

```

1  #include "pi_global.h"
2  #include "pi_alg/pi_alg.h"
3  #include "pi_reset/pi_reset.h"
4  #include "pi_log/pi_log.h"
5  #include "pi_task/pi_task.h"
6  #include <pthread.h>
7
8  #define BUFFERLEN 10
9  #define NUM_THREADS 3
10
11 /* Global variable definitions */
12 float32 angle_norm_vals[11] = {0.0F, 0.1F, 0.2F, 0.3F, 0.4F, 0.5F, 0.6F, 0.7F, 0.8F, 0.9F, 1.0F};
13 float32 pos_norm_vals[11] = {1.0F, 0.9F, 0.8F, 0.7F, 0.6F, 0.5F, 0.4F, 0.3F, 0.2F, 0.1F, 0.0F};
14
15 map_data angle_norm_map = { .valueLo = 0.0F, .iHi = 10, .uSpacing = 3.14159F/20.0F};
16 map_data pos_norm_map = { .valueLo = 0.5F, .iHi = 10, .uSpacing = 0.4F};
17
18 pthread_mutex_t integral_state_mutex;
19 pthread_mutex_t inp_val_mutex;
20 pthread_mutex_t read_failure_mutex;

```

The left sidebar shows the 'QUALITY MONITORING' section with a list of files and their status:

- pi_alg.c pi_alg ✓
- pi_globals.c pi_app ✓
- pi_main.c pi_app ✓
- pi_global.h ✓
- pi_log.c pi_log ✓
- pi_log.h pi_log ✓
- pi_reset.c pi_reset ✓
- pi_task.c pi_task ✓
- pi_utils.c pi_task ✓

Below this is the 'RESULT DETAILS' section with the text: "Select a diagnostic to get more information."

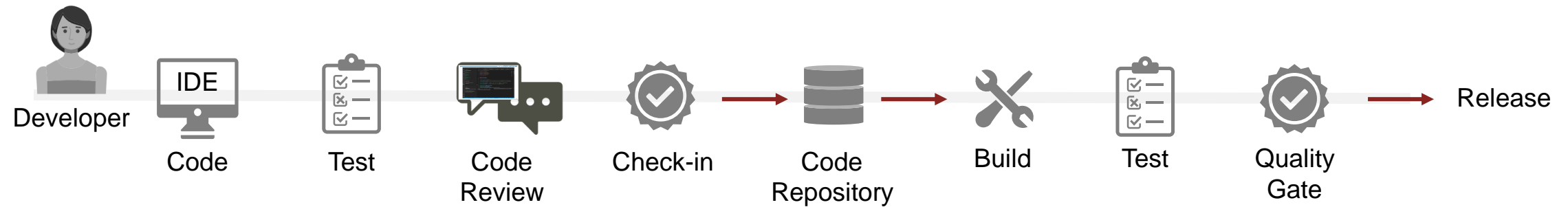
The 'CONFIGURATION' section shows:

- Build setting: Get from JSON Compilatio...
- Build options generated ✓
- CCDB file: compile_commands.json
- Checkers file: checkers.xml

The 'PROBLEMS' panel at the bottom shows: "No problems have been detected in the workspace."

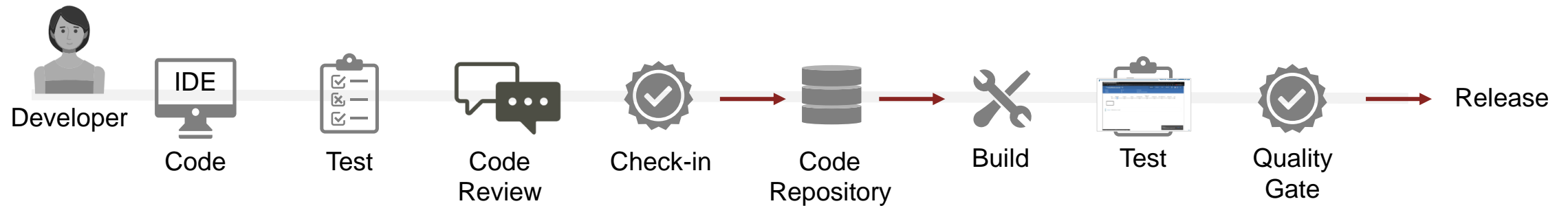
The status bar at the bottom indicates: "Ln 21, Col 34 Spaces: 4 UTF-8 CRLF C Win32".

Polyspace automates (a boring part of) the Code Reviews



- Automate tedious code review tasks
- Help developers focus on algorithms

Static Analysis should be integrated in a CI pipeline



jenkins / ScriptedPipeline_Example x +

Not secure | demo-polyspace-ci.gnb.mathworks.com:8080/blue/organizations/jenkins/ScriptedPipeline_Example/detail/ScriptedPipeline_Example/52/pipeli...

ScriptedPipeline Example < 52

Pipeline Changes Tests Artifacts

Branch: — < 1s No changes
Commit: — - Started by user anonymous

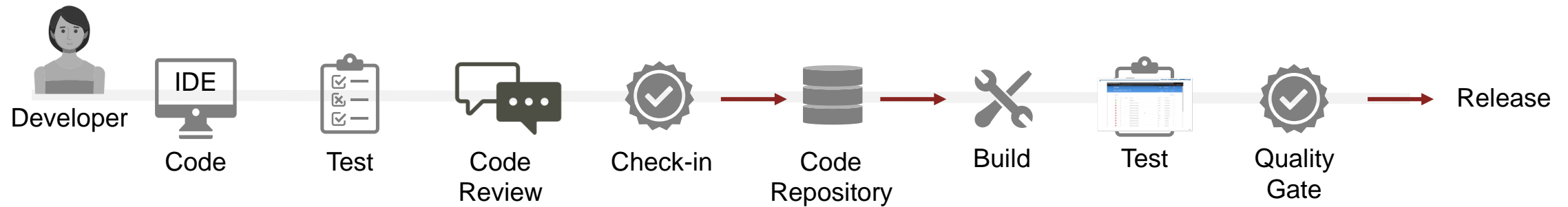
```
graph LR; Start((Start)) --> Checkout((Checkout)); Checkout --> Configure((Configure)); Configure --> Analyze((Analyze)); Analyze --> PublishResults((Publish Results)); PublishResults --> AssignFindings((Assign new findings)); AssignFindings --> QualityGate((Quality Gate)); QualityGate --> MakePDFReport((Make PDF Report)); MakePDFReport --> Notification((Notification)); Notification --> End((End));
```

Queued Waiting for run to start

Started "ScriptedPipeline_Example" #52 [OPEN](#) ×

demo-polyspace-ci.gnb.mathworks.com:8080/blue/organizations/jenkins/.../pipeline

Polyspace assesses Software Quality vs Quality Gates



- Define consistent Quality Gates
- Evaluate final quality of code
- Automatically generate reports

Conclusion – Future direction...

Safe and
Secure

Fully autonomous
Zero carbon

Downloadable
New features

Obviously developed with a Software Factory
... with heavy use of Static Analysis from IDE to CI!

5G - 6G
Telecommunication

Billions LOC
software inside

To help you apply Static Analysis...
We'll continue to evolve Polyspace!

Thank you