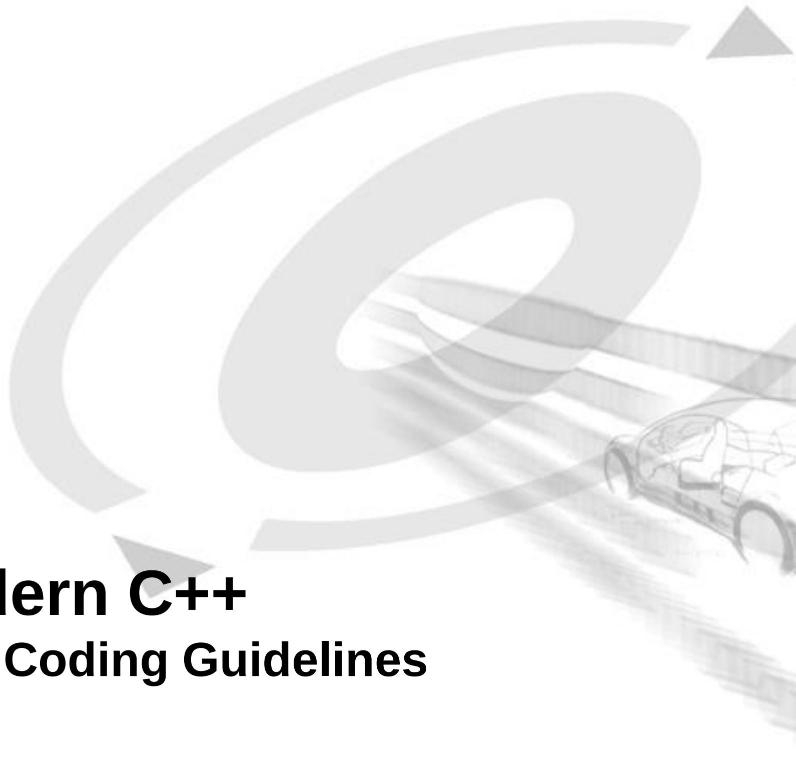
AUTOSAR



Driving Into the Future With Modern C++

A Look at Adaptive Autosar and the C++14 Coding Guidelines

Jan Babst

CppCon 2017

Sep 27 2017, Bellevue, WA

BMW Group MINI BOSCH Continental DAIMLER

Ford GM PSA GROUPE TOYOTA VOLKSWAGEN AKTIENGESELLSCHAFT

Overview

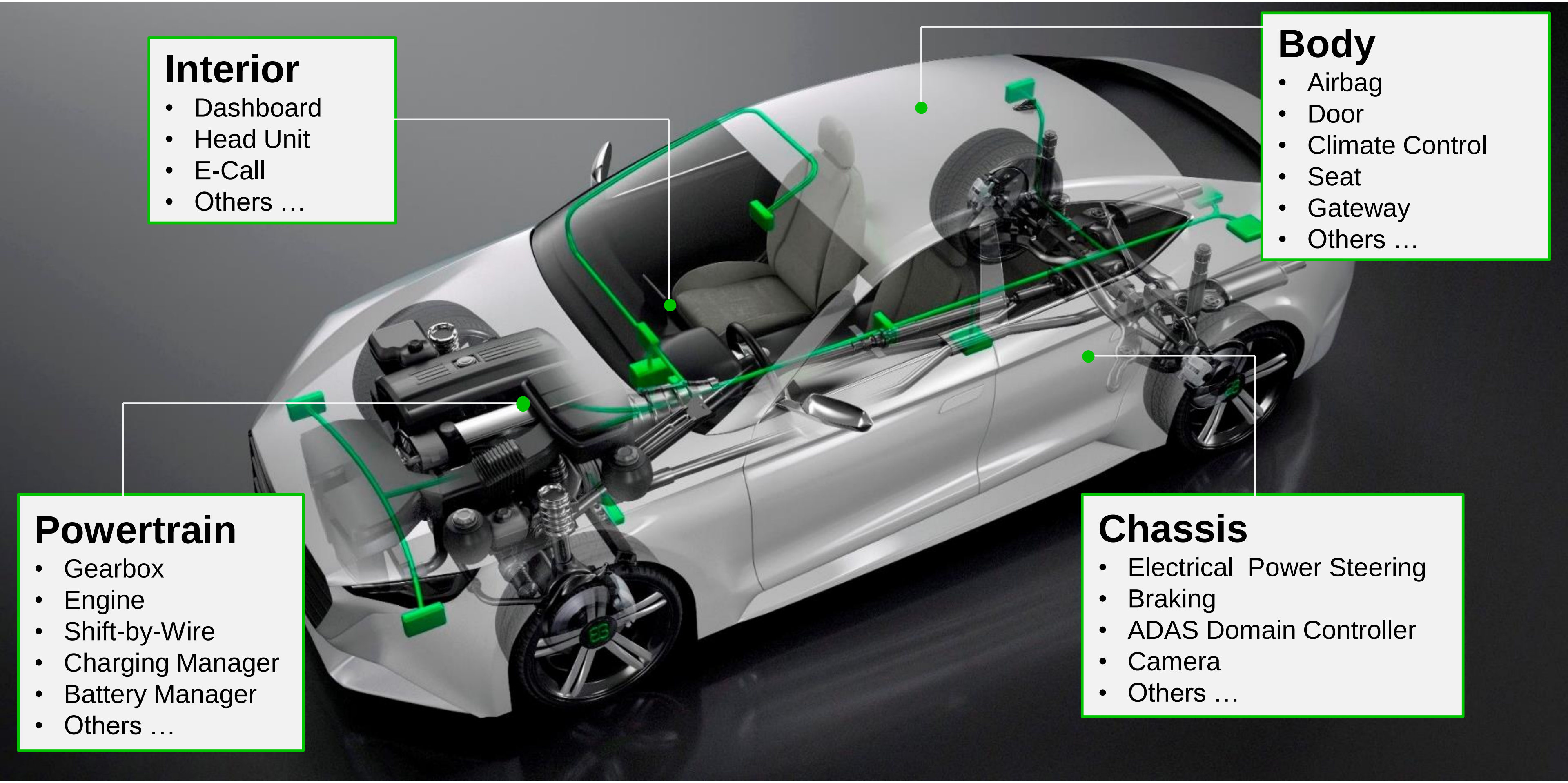
- What is Adaptive AUTOSAR?
- AUTOSAR C++14 guidelines
- Summary and Outlook

Overview

What is Adaptive AUTOSAR?

- AUTOSAR C++14 guidelines
- Summary and Outlook

Classic AUTOSAR



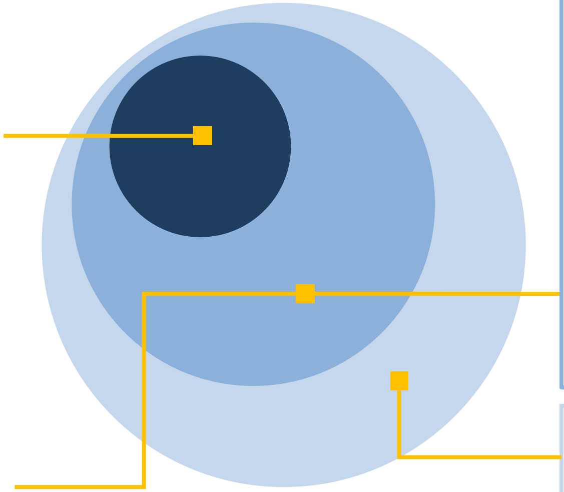
Interior
- Dashboard
- Head Unit
- E-Call
- Others …

Body
- Airbag
- Door
- Climate Control
- Seat
- Gateway
- Others …

Powertrain
- Gearbox
- Engine
- Shift-by-Wire
- Charging Manager
- Battery Manager
- Others …

Chassis
- Electrical Power Steering
- Braking
- ADAS Domain Controller
- Camera
- Others …

Partners (Status February 2016)



9 Core Partners

BMW Group
BOSCH
Continental
DAIMLER
Ford
PSA PEUGEOT CITROËN
TOYOTA
GM
VOLKSWAGEN AG

28 Development Partners

avelabs
BASICWORX
E.S.R.LABS
fka
INCHRON
intecs
easycore
Timing Architects
itemis
automotive engineering tool alliance
KEREVAL
OFFIS
AVIN SYSTEMS
osb
C&S
SGS
GLIWA
ikv++ technologies ag
seneos
TTTech
OPENSYNERGY
pure-systems
MIRABILIS
Testing Technologies
VALIDAS
RUETZ SYSTEM SOLUTIONS
SYMTA VISION

116 Associate Partners
16 Attendees

43 Premium Partners

FIAT
CHRYSLER
CNH INDUSTRIAL
HONDA The Power of Dreams
HYUNDAI
Great Wall
mazda
RENAULT
VOLVO
VOLVO

General OEM

Autoliv
DELPHI
HELLA
DENSO
JTEKT
Johnson Controls
LEAR CORPORATION
MAGNETI MARELLI
Panasonic
TRW
ThyssenKrupp
Valeo
ZF

Generic Tier 1

Mentor Graphics
EB
KPIT Cummins Infosystems Limited
vector

Standard Software

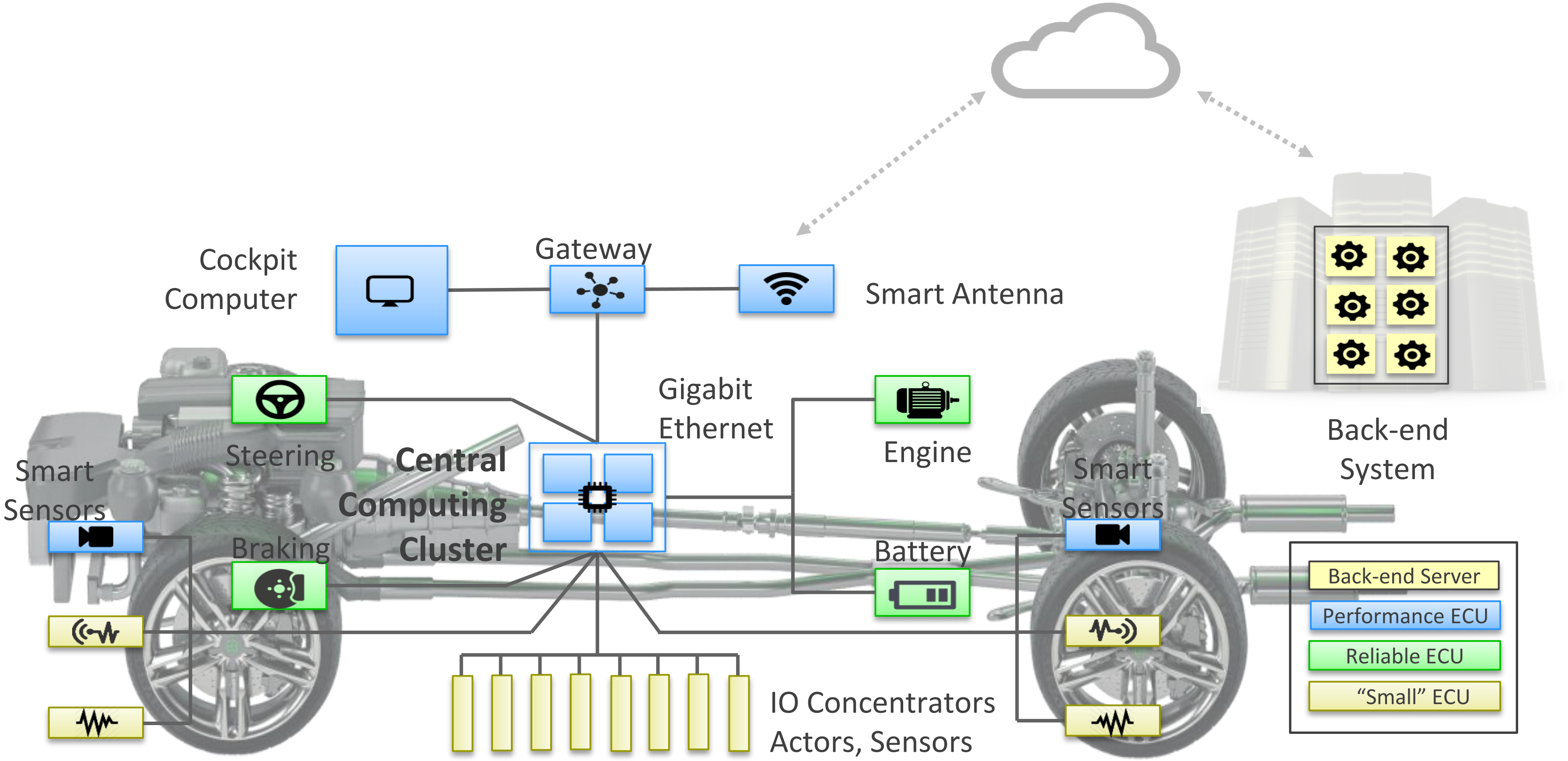
cea
altran
dSPACE
ETRI
DASSAULT SYSTEMES
Fraunhofer
ETAS
automotive engineering iav
MBtech
TATA CONSULTANCY SERVICES
The MathWorks
iGATE
itk ENGINEERING
ixia
sodius
L&T Technology Services
TATA ELXSI LIMITED

Tools and Services

freescale semiconductor
Infineon
RENESAS
ST life.augmented

Semi-conductors

Adaptive AUTOSAR



Further information …

www.autosar.org

Why use C++14?

Why not use C++03?

Why not use C++03?

- ISO safety standards require to use “state-of-the-art”
- Need to attract developers
- C++14 provides better alternatives than C++03
  - to avoid unwanted implicit conversions
    (**`auto`**, **`explicit`**, uniform initialization)
  - to safely loop over a container
    (range-**`for`**)
  - to create type-safe functions with variable number/type of arguments
    (variadic templates)
  - …
- Concurrency, parallelism
  - at least some basic support

ara::com API

```
auto fut = remoteService.DoSomething();

fut.get(); // event-driven, blocking

fut.then([](auto f){
    process_result(f.get());    // event-driven, non-blocking continuation
    …
});

while (!stop) {    // real-time, polling
    if (fut.is_ready()) {
        process_result(fut.get());
        …
    }
    else {
        // do something else
    }
}
```

Not supported by C++17 future!

Why not use C++17?

Overview

- What is Adaptive AUTOSAR?

AUTOSAR C++14 guidelines

- Summary and Outlook

AUTOSAR C++14 Guidelines

- Why and how?
- Single return
- Exceptions
- Dynamic memory
- Miscellaneous

Why and how?

- Single return
- Exceptions
- Dynamic memory
- Miscellaneous

Why do we need guidelines?

- Start a project without them …
- Process and safety standards say:
  - Have guidelines
  - Check them continuously with automatic tool

Why do we need guidelines?

“C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off.” — Bjarne Stroustrup

“C++14 makes it even harder, but you can still blow your whole leg off.” — my two cents

C Programmer? …

Where to look?

- **Safety, but not C++14**
  - Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program, 2005
  - MISRA C++:2008 Guidelines for the use of the C++ language in critical systems, The Motor Industry Software Reliability Association, 2008

- **C++11/14, but not safety**
  - High Integrity C++ Coding Standard Version 4.0, Programming Research Ltd, 2013
  - Software Engineering Institute CERT C++ Coding Standard, Software Engineering Institute Division at Carnegie Mellon University, 2016
  - C++ Core Guidelines, http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines, 2017

Guidelines for the use of the C++14 language in critical and safety-related systems

- Written as an update to MISRA C++ 2008
- Traceability to MISRA, JSF, HIC++, SEI CERT C++, C++ Core Guidelines
- Version 17/03 available publicly: https://www.autosar.org/fileadmin/files/standards/adaptive/17-03/general/specs/AUTOSAR_RS_CPP14Guidelines.pdf

Credits

AUTOSAR C++14 Guidelines

- Why and how?
- Single return
- Exceptions
- Dynamic memory
- Miscellaneous

MISRA C++ 2008 says:

Rule 6-6-5 (Required)

A function shall have a single point of exit at the end of the function.

Exception

…

Throwing an exception that is not caught within the function is not considered a point of exit for this rule.

No “single exit” rule anymore!

- AUTOSAR C++ Guidelines drop the “single exit” rule completely

1. We have to deal with multiple exits anyway

```
void foo()
{
    std::vector<int> v(10); // allocates memory
    // …
    bar(); // may throw exception
    // …
} // deallocates memory
```

Use **R**esource **A**quistion **I**s **I**nitialization!

Generic resource wrapper:
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0052r2.pdf

2. Reduces code complexity

```
int single_exit(int decision1, int decision2)
{
    int result = 0;
    if (decision1 == 0) {
        if (decision2 == 0) {
            // …
            result = …;
        }
        else {
            result = …;
        }
    else {
        result = …;
    }
    return result;
}
```

2. Reduces code complexity

```
int multiple_exits(int decision1, int decision2)
{
    if (decision1 != 0) {
        return …;
    }
    if (decision2 != 0) {
        return …;
    }
    // …
    return …;
}
```

3. Avoids dataflow anomalies (avoid temporaries)

```
int single_exit(int decision)
{
    int result;
    if (decision == 0) {
        result = -1;
    }
    else {
        result = 1;
    }
    return result;
}
```

3. Avoids dataflow anomalies (avoid temporaries)

```
int single_exit(int decision)
{
    int result; // potentially UR (undefined-referenced)
    if (decision == 0) {
        result = -1;
    }
    else {
        result = 1;
    }
    return result;
}
```

3. Avoids dataflow anomalies (avoid temporaries)

```
int single_exit(int decision)
{
    int result = 1;
    if (decision == 0) {
        result = -1; // DU (defined-unused)
    }
    else {
        result = 1; // DD (double-define)
    }
    return result;
}
```

Both MISRA C++ 2008 and AUTOSAR C++14 require avoiding dataflow anomalies!

3. Avoids dataflow anomalies (avoid temporaries)

AUTOSAR C++14 achieves it:

```
int multiple_exits(int decision)
{
    if (decision == 0) {
        return -1;
    }
    return 1;
}
```

AUTOSAR C++14 Guidelines

- Why and how?
- Single return
- Exceptions
- Dynamic memory
- Miscellaneous

What others say …

- MISRA C++ 2008

"… can provide an effective and clear means of handling error conditions …

However, ... can also lead to code that is difficult to understand."

- C++ Core Guidelines (2017)

"The preferred mechanism for reporting errors ... is exceptions rather than error codes. A number of core language facilities, including dynamic_cast, operator new(), ... [and] the C++ standard library make[s] ... use of exceptions.

Few programs manage to avoid some of these facilities."

What we say …

- You are not forced to use exceptions for your own error reporting
- Do not pretend that you can ignore exceptions
- Use them appropriately and correctly

- Above all: do not use for control flow!

Last ditch – catch all

In **`main()`** and every thread main function:

```
try {
    // program code …
}
catch (std::runtime_error& e) {
    // Handle runtime errors …
}
catch (std::logic_error& e) {
    // Handle logic errors …
}
catch (std::exception& e) {
    // Handle all expected exceptions …
}
catch (...) {
    // Handle all unexpected exceptions …
}
```

Some concerns about exceptions

- Hidden control flow
- Additional exit point from functions
- Exception safety / program state after exception is thrown
- Impact on runtime performance
- Impact on worst-case execution time

Hidden control flow

Rule A15-0-1

A function shall not exit with an exception if it is able to complete its task.

```
bool isMessageCrcCorrect(std::string const& message)
{
    std::uint8_t computedCrc = computeCrc(message);
    std::uint8_t receivedCrc = message.at(0);
    if (computedCrc != receivedCrc) {
        throw std::logic_error("Crc not correct");
        // Not compliant - could perform its task
    }
    return true;
}
```

Hidden control flow

Rule A15-0-1

A function shall not exit with an exception if it is able to complete its task.

```
bool isMessageCrcCorrect(std::string const& message)
{
    std::uint8_t computedCrc = computeCrc(message);
    std::uint8_t receivedCrc = message.at(0);
    if (computedCrc != receivedCrc) {
        return false;
        // Compliant - could perform its task
    }
    return true;
}
```

Hidden control flow

Rule A15-0-1

A function shall not exit with an exception if it is able to complete its task.

```
bool isMessageCrcCorrect(std::string const& message)
{
    std::uint8_t computedCrc = computeCrc(message);
    std::uint8_t receivedCrc = message.at(0);
    // Compliant - throws std::out_of_range if message
    // is empty, i.e. could not perform its task
    if (computedCrc != receivedCrc) {
        return false;
        // Compliant - could perform its task
    }
    return true;
}
```

Additional exit point from functions

- Same reasoning: Exceptions are not for control flow
- See previous discussion about multiple exit points

Exception safety / program state after exception is thrown

Rule A15-0-2

At least the basic guarantee for exception safety shall be provided for all operations. In addition, each function may offer either the strong guarantee or the nothrow guarantee.

Impact on runtime performance

- Depends on compiler
- GCC and Clang offer “zero cost exception handling”
- “Zero cost” only as long as exception is not thrown

Impact on worst-case execution time

- **Rule A15-0-6** *Worst case execution time must be analyzed.*

- **Rule A15-0-7** *Exception handling mechanism shall guarantee a deterministic worst-case time execution time.*

AUTOSAR C++14 Guidelines

- Why and how?
- Single return
- Exceptions
- Dynamic memory
- Miscellaneous

Why do we need dynamic memory?

- Size of data only known at runtime
- Lifetime of data independent from object lifetimes
- Sharing/transmitting data across threads (**`promise`** – **`future`**)
- Type erasure, e.g. **`std::function`**
- Some language/library features use dynamic memory implicitly
  - Exception handling
  - Containers (can be customized)
  - **`std::function`** (customization deprecated in C++17)

Few programs manage to avoid some of these facilities

Dynamic memory issues

- Memory leaks
- Memory fragmentation
- Non-deterministic execution time
- Out of memory

Memory leaks

- Use RAII
- Do not call **`new`** and **`delete`** explicitly

Easy to achieve with

- **`std::vector`**, **`std::string`**, and other containers
- **`std::unique_ptr`**, **`std::make_unique`**
- **`std::shared_ptr`**, **`std::make_shared`**

Memory fragmentation

- Allocator must minimize fragmentation
- Usually OS **`malloc`**/**`free`** is pretty good at that
- Techniques/implementations for custom allocators are available

Non-deterministic execution time

- Allocators must guarantee deterministic WCET

Either

- OS **`malloc`**/**`free`** makes this guarantee

or

- Roll your own: **`malloc`**/**`free`**, **`new`**/**`delete`**, custom allocators

or

- Allocate/deallocate only during non-realtime phases of the program

Out of memory

- Define maximum memory needs, use pre-allocated storage

AUTOSAR C++14 Guidelines

- Why and how?
- Single return
- Exceptions
- Dynamic memory
- Miscellaneous

What about C++14 features?

Lambdas

use, but …

- No implicit capture

```
// Non-compliant
std::int32_t sum{0};
std::for_each(v.begin(), v.end(), [&](std::int32_t rhs) {
    sum += rhs;
});

// Compliant
sum = 0;
std::for_each(v.begin(), v.end(), [&sum](std::int32_t rhs) {
    sum += rhs;
});
```

Lambdas

use, but …

- Always write parameter list, even if empty

```
std::int32_t x{0};
std::generate(v.begin(), v.end(), [&x]() {
    return x++;
});
```

Lambdas

use, but …

- Don’t nest

No example ☺

Auto

May use, but only when

- initializing from a return value

```
auto const sz = vec.size();
```

- initializing a non-primitive, non-utterable type

```
auto const lambda
    = [](int32_t x, int32_t y) { return x * y; };
// C++14! Cannot use {} here.
// Otherwise, always use {}
int32_t i{42};  // not auto i = 42;
A a{};
```

- the language requires it
  - Generic lambdas
  - Return type deduction

Do we actually forbid something?

- Atomics – *in application code*
- Threads, sync primitives – *in application code*
- Explicit **`new/delete`**
- **`dynamic_cast`**
- **`reinterpret_cast`**
- **`C-style casts`**
- **`wchar_t`**
- **`<cstdarg>`**
- **`<cstdio>`**
- **`<clocale>`**
- **`<locale>`**
- …

Overview

- What is Adaptive AUTOSAR?
- AUTOSAR C++14 guidelines
- Summary and Outlook

Summary

- Adaptive AUTOSAR brings modern C++ to the automotive world on a large scale
- The AUTOSAR C++14 Guidelines are the first comprehensive C++14 guidelines for automotive / critical systems development
- Still work in progress, inkomplete

Outlook

- Guidelines
  - More analysis of existing rules, traceability
  - Rules on standard library usage
  - Rules on multithreading
  - Tool support
  - Handover to “proper” organization

- Adaptive AUTOSAR
  - Release 10/17 upcoming

Questions?