Safe and certified software for autonomous mobility

How Apex.AI Certified ROS 2 According to ISO 26262 ASIL-D
Dejan Panceric (dejan@apex.ai)
Relevance to Autoware Community

1. Autoware.Auto uses ROS2. If AWF members want to certify whole or parts of Autoware.Auto and then sell that as part of the fully certified complete SW AD stack - they need a certified framework.

2. ROS2 is a C++14 product. This talk is largely about how to certify any C++ code base (independent of ROS2, Autoware.Auto, ...)

3. The process of certification improved the quality of Apex.AI fork of ROS2. Consequently this talk is also about the code quality.

4. ISO 26262 is at the center of standardization in the automotive industry. There is no more important standard and one can not avoid it if you build production systems for automotive. Consequently this talk is about sharing an experience of going through the process of ISO 26262 certification.
But the entire AD Stack is Huge

<table>
<thead>
<tr>
<th>Focus of this talk</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Application</td>
</tr>
<tr>
<td>Algorithmic Software</td>
</tr>
<tr>
<td>On-Board Software</td>
</tr>
<tr>
<td>Framework Software</td>
</tr>
<tr>
<td>Operating System</td>
</tr>
<tr>
<td>Functional Safety</td>
</tr>
<tr>
<td>Testing</td>
</tr>
<tr>
<td>Design</td>
</tr>
<tr>
<td>Development Tools</td>
</tr>
<tr>
<td>Off-Board Software and Data</td>
</tr>
<tr>
<td>Data Processing</td>
</tr>
<tr>
<td>Operations</td>
</tr>
<tr>
<td>Mapping</td>
</tr>
<tr>
<td>Maps</td>
</tr>
<tr>
<td>Sensors</td>
</tr>
<tr>
<td>Components</td>
</tr>
<tr>
<td>Vehicle</td>
</tr>
<tr>
<td>System Integration</td>
</tr>
<tr>
<td>2D/3D Perception, Tracking, Class.</td>
</tr>
<tr>
<td>Abstraction of hardware, middleware, OS, interfaces</td>
</tr>
<tr>
<td>Operating System</td>
</tr>
<tr>
<td>Functional Safety</td>
</tr>
<tr>
<td>Testing</td>
</tr>
<tr>
<td>Design</td>
</tr>
<tr>
<td>Development Tools</td>
</tr>
<tr>
<td>Data Processing</td>
</tr>
<tr>
<td>Operations</td>
</tr>
<tr>
<td>Mapping</td>
</tr>
<tr>
<td>Maps</td>
</tr>
<tr>
<td>Sensors</td>
</tr>
<tr>
<td>Components</td>
</tr>
<tr>
<td>Vehicle</td>
</tr>
</tbody>
</table>
We have a Base Functional AD System - What Next?

1. **System Safety:**
   a. ISO 26262 Certification
      i. Code
      ii. HW
   b. System specification and operating environment (ODD)
   c. HARA
   d. Design for redundancy
   e. Validation plan
      i. System validation (ISO 15288)
   f. SOTIF
      i. Scenario-based testing with statistical sampling in simulation (NCAP, NHTSA scenarios)
   g. Closed course testing
   h. Public road testing
      i. Simulation
      i. SIL and HIL Testing

2. **AV Technology:**
   a. Object and event detection and response
   b. Fallback systems

3. **AV Operation:**
   a. ODD
   b. AV Operators
   c. Incident response and management

4. **Interfaces:**
   a. Passenger and road user interface
   b. Cybersecurity
   c. Data management

Source: Motional, [VSSA, 2021](https://www.vssacar.org)
We have a Base Functional System - What Next?

1. System Safety:
   a. ISO 26262 Certification
      i. Code
      ii. HW
   b. System specification and operating environment (ODD)
   c. HARA
   d. Design for redundancy
   e. Validation plan
      i. System validation (ISO 15288)
   f. SOTIF
      i. Scenario-based testing with statistical sampling in simulation (NCAP, NHTSA scenarios)
   g. Closed course testing
   h. Public road testing
      i. Simulation
      i. SIL and HIL Testing

2. AV Technology:
   a. Object and event detection and response
   b. Fallback systems

3. AV Operation:
   a. ODD
   b. AV Operators
   c. Incident response and management

4. Interfaces:
   a. Passenger and road user interface
   b. Cybersecurity
   c. Data management

Source: Motional, VSSA, 2021

Focus of the rest of the talk
**Apex.OS Development Lifecycle**

**ISO 26262/SEooC/part3,part6…. processes**

### Automotive Stakeholder Requirements (ASR)
- Elicitation, Safety Concept, SW Safety Requirements
- UML (unified modeling language), FMEA

### Feature set reduction
- Apply real-time and determinism constraints
  1. Memory static
  2. Remove blocking calls and recursions

### ROOS
- builtin_interfaces
- connext_micro_support
- allocator
- logging
- rclcpp
- threading

### V&V
- Req, arch, unit, integration, system, performance, fault injection tests
- Safety manual, Restrictions, Traceability

### Conf. Reviews
- Apex_ecu_monitor (native)
- Apex_utils (native)

### Apex.OS Cert
- builtin_interfaces_cert
- connext_micro_support_cert
- allocator_cert
- logging_cert
- rclcpp_cert
- threading_cert

© 2021 Apex.AI, Inc.
Apex.OS Certification Activities per package

1. Reduce the feature set of a package and extensions
2. Investigation to make APIs memory static & ensure no blocking calls
3. Static Code Analysis (SCA)
4. Structural Coverage (statement, branch and MC/DC)
5. Notations of Designs (modelling diagrams)
6. Principles of SW architecture and design
7. Control and data flow analysis
8. Integration and Specialized tests
9. Requirements and traceability
10. Safety Analysis (FMEA)
11. Generate Safety Artifacts (TUV submission)
12. Testing on Target platform/hardware
13. Tool Classification and Qualification

Total 24 pkgs selected for first release of Apex.OS

Some activities such as Tool classification and qualification, integration testing done at Apex.OS level.

Close to 100 safety artifacts had to be generated to provide evidence of ASIL D compliance to our certification agency.
ROS 2 exhibits the following gaps to enable **real-time performance**.

**Real-Time Gaps**

- **Non static mem operations**
  - Standard containers: Runtime mem allocation
  - Standard exceptions: Mem fragmentation
  - Exception throw causes mem allocation

- **non-real-time middleware**
  - Thread priorities, scheduling, pinning: Handler lookup on-deterministic due to inheritance
  - No control (std::thread)

- **Standard threading**
  - Blocking calls/deadlocks: Higher risk of dead locks since no tooling
  - Scheduling based on readiness of data (executor): Increased thrashing
Apex.OS addresses the following gaps to achieve real-time performance.

- **Real-Time**
  - static mem operations
  - Real-Time DDS
  - apex::threading
    - Blocking calls/deadlocks
      - Eliminated
    - Reliance on OS scheduler vs ROS executor
      - Greatly reduced thrashing
  - apex::containers
    - apex::string
    - apex::map/set
    - apex::vector
  - apex::exceptions
    - Process defined to make catching exceptions deterministic
  - Better control over thread priorities, scheduling and pinning
**Apex.OS Certification Activities per package**

1. Reduce the feature set of a package and extensions
2. Investigation to make APIs memory static & ensure no blocking calls
3. Static Code Analysis (SCA)
4. **Structural Coverage (statement, branch and MC/DC)**
5. Notations of Designs (modelling diagrams)
6. Principles of SW architecture and design
7. Control and data flow analysis
8. Integration and Specialized tests
9. Requirements and traceability
10. Safety Analysis (FMEA)
11. Generate Safety Artifacts (TUV submission)
12. Testing on Target platform/hardware
13. Tool Classification and Qualification

Total 24 pkgs selected for first release of Apex.OS

Some activities such as Tool classification and qualification, integration testing done at Apex.OS level.

Close to 100 safety artifacts had to be generated to provide evidence of ASIL D compliance to our certification agency.
Branch coverage vs. MC/DC coverage

**Branch coverage**

Each branch (True and False) should be tested at least once

**MC/DC coverage**

Every condition in a decision (True and False) should be tested independently

For example (A && B),

1. Create the truth table

<table>
<thead>
<tr>
<th>ROW</th>
<th>A</th>
<th>B</th>
<th>Res</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

2. Find pairs for which only one condition independently affect the outcome

a = {1,3}, b = {1,2} -> 1, 2, 3 condition should be tested.

3. For n conditions we only require n+1 tests
To get to the 100% of line (statement), branch and MC/DC (pairs) test coverage we had to add 3000 tests (on top of the 1500 existing tests).
What was tedious?

• Getting 100% MC/DC coverage for heavily templated modern C++ code is tedious.
  ◦ Commercial coverage tool has issues, while it parses modern C++ codes such as a lambda function and template code. (e.g. on the next slide)

• The code base has a lot of hard to reach defensive type coding.
  ◦ Required significant stubbing/mocking of C++ standard library, middleware, and external functions that are implemented in Apex.OS. (e.g. on the next slide)
Issues Parsing Certain Modern C++ Constructs

• A method with multiple lambda functions
  ◦ The commercial coverage tool could not parse a method that contained multiple lambda functions.

  **Solution:** fixing the bug of the commercial coverage tool.

• Template code with locally defined class
  ◦ The commercial coverage tool cannot parse a class internally defined in a function or the class that is used for the parameter of the template class or function.

  **Solution:** defining the class with the global scope
Mocking GNU C Lib Functions: Example: `clock_gettime()`

```c
rcutils_system_time_now(rcutils_time_point_value_t * now)
{
    RCUTILS_CHECK_ARGUMENT_FOR_NULL(now,
    RCUTILS_RET_INVALID_ARGUMENT);
    struct timespec timespec_now;
    int32_t posix_error;
    posix_error = clock_gettime(CLOCK_REALTIME, &timespec_now);
    if (posix_error != 0) {
        RCUTILS_SET_ERROR_MSG("clock_gettime error");
        return RCUTILS_RET_ERROR;
    }
    if (RCUTILS_WOULD_BE_NEGATIVE(timespec_now.tv_sec,
        timespec_now.tv_nsec)) {
        RCUTILS_SET_ERROR_MSG("unexpected negative time");
        return RCUTILS_RET_ERROR;
    }
}
```

```c
int clock_gettime(clockid_t clk_id, struct timespec * tp) __THROW__
{
    int ret = 0;
    if (nullptr != timeUnixPtr) {
        ret = timeUnixPtr->clock_gettime(clk_id, tp);
    }
    return ret;
}
```

```c
TEST_F(time_gmock, rcutils_system_time_now) {
    rcutils_time_point_value_t now = 0;
    rcutils_ret_t ret;
    EXPECT_CALL(*timeUnixPtr, clock_gettime(_, _)).WillRepeatedly(Return(-1));
    ret = rcutils_system_time_now(&now);
    EXPECT_EQ(ret, RCUTILS_RET_ERROR);
    rcutils_reset_error();
}
```

Apex.OS source code

Apex.OS test code
Apex.OS Certification Activities per package

1. Reduce the feature set of a package and extensions
2. Investigation to make APIs memory static & ensure no blocking calls
3. Static Code Analysis (SCA)
4. Structural Coverage (statement, branch and MC/DC)
5. Notations of Designs (modelling diagrams)
6. Principles of SW architecture and design
7. Control and data flow analysis
8. Integration and Specialized tests
9. Requirements and traceability
10. Safety Analysis (FMEA)
11. Generate Safety Artifacts (TUV submission)
12. Testing on Target platform/hardware
13. Tool Classification and Qualification

Total 24 pkgs selected for first release of Apex.OS

Some activities such as Tool classification and qualification, integration testing done at Apex.OS level.

Close to 100 safety artifacts had to be generated to provide evidence of ASIL D compliance to our certification agency.
What was technically challenging?

- There are no good commercial tools for identifying runtime memory allocations and blocking calls. We created new internal tool (apex_tracing_check) that uses LTTng framework to flag infractions.

- Making exceptions handling memory static is complex (and still a research topic) We solved it by patching system malloc() and a special (exception handling) memory pool. (see next slide)
Elimination of Memory Allocation and Blocking Calls (MA/BC) - Approach

- We implemented `apex_tracing_check` tool which is based on LTTng
- The code is instrumented by adding the macro on top of the function
- It requires to be build with some extra compilation flags to enable the macro
- After this, test cases are executed to find infractions

Example on how `apex_tracing_check` will detect and report infraction/s

```cpp
bool

TRACING_CHECK_START()

std::unique_lock<apex::time_limit_mutex> lock(interrupt_mutex);

TRACING_CHECK_END()
```

Example of mutex infraction in `sleep_for()` function
Elimination of Memory Allocation and Blocking Calls (MA/BC) - Findings

- Using `apex_tracing_check` and having 100% MC/DC coverage, it’s possible to verify that there is no MA/BC in runtime.

```cpp
if (ret != RMW_RET_OK) {
    const auto msg = std::string("failed to compare gids: ");
    std::string(rmw_get_error_string().str);
    const apex::string256_t msg = apex::vargargs_to_string("failed to compare gids: ");
    rmw_get_error_string().str);
    rmw_reset_error();
    throw std::runtime_error(msg);
    throw apex::runtime_error(&msg);
}
```

Replacing `std::string` and `std::exception` to avoid memory allocations in runtime

```cpp
Context::sleep_for(const std::chrono::nanoseconds & nanoseconds)
{
    TRACING_CHECK_START()
    std::unique_lock<std::mutex> lock(interrupt_mutex);
    + std::unique_lock<apex::time_limit_mutex>> lock(interrupt_mutex);
```

Replacing `std::mutex` with `apex::time_limit_mutex` to avoid blocking system call in runtime
Allocation Handling during Exception (apex_malloc pkg)

Allocation request from an exception during runtime?

__cxa_allocate_exception malloc()  
Apex::exception_allocator()  
Search for a free block in "pre-defined mem pool" and return ptr to this free block

Exception routine uses the ptr to service. Stack unwinding will release the block back to "pre-defined mem pool"

If search return null ptr (unlikely): c++ -> qnx emergency pool. If emergency pool return null then process terminates

Not from heap, no fragmentation, deterministic memory allocation

Back to runtime
**Apex.OS** Certification Activities per package

1. Reduce the feature set of a package and extensions
2. Investigation to make APIs memory static & ensure no blocking calls
3. Static Code Analysis (SCA)
4. Structural Coverage (statement, branch and MC/DC)
5. Notations of Designs (modelling diagrams)
6. Principles of SW architecture and design
7. Control and data flow analysis
8. Integration and Specialized tests
9. Requirements and traceability
10. Safety Analysis (FMEA)
11. Generate Safety Artifacts (TUV submission)
12. Testing on Target platform/hardware
13. Tool Classification and Qualification

Total 24 pkgs selected for first release of **Apex.OS**

Some activities such as Tool classification and qualification, integration testing done at **Apex.OS** level.

Close to 100 safety artifacts had to be generated to provide evidence of ASIL D compliance to our certification agency.
FMEA (Failure Mode and Effects) Analysis was performed on each public API in every Cert package.

New tests were added as a result of FMEA=> add example bug from cpputils

33 Software Safety Requirements were added as a result of FMEA Activity.

40 Restrictions and 25 Recommendations added to Safety Manual because of FMEA.

<table>
<thead>
<tr>
<th>Activity</th>
<th># Real Issues Found</th>
<th># Files Changed</th>
<th># Merge Requests</th>
<th>Git Commits</th>
<th># Changed code lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMEA Analysis</td>
<td>17</td>
<td>55</td>
<td>17</td>
<td>3</td>
<td>62</td>
</tr>
</tbody>
</table>
Examples of Safety Related Changes from ROS 2 to Apex.OS

Example of requirement tracing

“The function rclcpp::Context::sleep_for() shall timeout immediately when zero and negative values are given for the nanoseconds argument”

Solution: The function was not working as described. During the requirement tracing the function and its corresponding tests were fixed

“rclcpp shall provide functionality to assert the liveliness of a publisher”

Solution: There were no tests verifying these requirements. As a result of the analysis a test was added

Example of issue detected as a result from FMEA

“If rclcpp publisher takes more time than is expected to publish a message the application could malfunction”

Solution: If used along with the ApexNode, function calls that exceed the expected time may cause max_cycle_time to be exceeded, which will then notify the user of the failure.
Certification in Numbers

- First round of Apex.OSCert contained ~65K lines of code
- 14 person years of effort (1 full time for 2 years, 12 full time for a year)
- 24 ROS 2 + native Apex.OS packages certified
- > $5M cost in tool licenses, infrastructure, and engineering resources
- 100% statement, branch, and MC/DC coverage
- ~3000 new tests added to fulfill safety/certification compliance
- ~300 safety requirements generated from FMEA, TSC, and Tools C&Q
- ~100 artifacts submitted to third party auditor (TÜV NORD) for ISO 26262 ASIL D compliance assessment (~2000 A4 pages if printed)
  - Total of 5 iterations of audits were conducted by TÜV NORD
Summary of Safety Related Changes from ROS 2 to Apex.OS

ROS 2

• Not real-time/deterministic
• No formal requirements compliant to ISO 26262
• No safety analysis
• No Static Code Analysis (SCA) or code coverage

Apex.OS

• Several changes to improve real-time/determinism.
  Removed all runtime memory allocations and blocking calls.
• Formal requirements written and traced to design and test.
• SW FMEA carried on every package to derive additional requirements and/or restrictions.
• Full compliance to AUTOSAR cpp14 V3.19 coding guidelines.
• Full MC/DC coverage.

Getting full MC/DC coverage and removing runtime memory allocations was challenging and took most of the time!
Thanks

Apex.AI
Autoware.Auto
Autoware Foundation
Contact: dejan@apex.ai