“A methodology to ensure safety (certification) of complex software in safety critical automotive systems"

Francesco.Brancati@ResilTech.com
Scope of the Talk

This talk presents an approach to

Safety Analysis
Dependent Failure Analysis

according to the automotive standard ISO26262 for complex software with focus on the following features

- Embedded SW
- Library/component based
  - Suitable for SEooC (Safety Element out of Context) integration
- Multi-criticality software systems
1. Short Company Introduction
2. SW Safety Analysis and DFA in Automotive
3. ResilTech Methodology
4. Feedback from application and future directions
1. Short Company Introduction

2. SW Safety Analysis and DFA in Automotive

3. ResilTech Methodology

4. Feedback from application and future directions
ResilTech s.r.l.

Company founded in late 2007 by
- **specialists in the industrial field** of Verification and Validation (V&V) of critical systems and
- **university researchers** expert in Resilient Computing
- 2012-2016 **Spin-OFF** of the University of Florence

**Mission**

To provide engineering consulting and design services to companies and public bodies mainly for, but not limited to, the field of resilient systems and infrastructures

**ISO 9001:2015 certified**
Core services

Resilient Systems Design
Architecting and Implementation of Dependable Systems

Verification & Validation & Safety
Full V&V&S Cycles activities according to latest standards of SW intensive system

Support to Certification Bodies
Cooperation with National & International Certification Agencies

Cyber security
Security solution design and assessment

Advanced Training
On Safety Standards, system modeling, Life Cycle Cost Analysis, Verification and Validation
Creating Innovation

Main research topics

- Cost Effective V&V Methodologies and tools
- Integration of AI components in Safety Critical System
- Online Failure/Intrusion prediction and Detection
- Monitoring and Analysis (ML& AI)
- Continuous Transparent Biometric authentication
- Safety Platforms SW/system for Embedded System
- Intelligent and smart monitoring of SoCs
- Methods for Resilient time distribution

Ongoing Projects:
- SISTER - POR Toscana 2014
- STORM - H2020-DRS11-2015
- PROTECT ID – PON – MISE 2016
- Net2DG – H2020-LCE-2017
- YACHT4.0 – POR Toscana 2017
- Good4you – Innonetwork (Puglia)

Starting Projects:
- MAIA – PON-MIUR-2018
- ADVANCE – H2020-RISE-2018

Patents:
- METHOD AND APPARATUS FOR A RESILIENT SIGNALING OF TIME
  Italian Office N. 102015000072477

75° Meeting of IFIP Working Group 10.4 Chamonix, Switzerland.
24-28 January 2019 – Company Confidential
ISO TC22/ SC32/WG8
for ISO26262 (“Road vehicles - Functional safety“)
for ISO21448 SOTIF (“Safety of the Intended Functionality“)

OpenGL SC 2.0 is a safety critical subset of the Open Graphics Library for safety critical markets

- streamlined APIs can significantly reduce certification costs
- includes avionics and automotive displays
  - https://www.khronos.org/registry/OpenGL/specs/sc/sc_spec_2.0.pdf
1. Short Company Introduction

2. SW Safety Analysis and DFA in Automotive

3. ResilTech Methodology

4. Feedback from application and future directions
Automotive market trend

- “migration” of technology (and SW) from non safety relevant application.
- increasing need of having components with some degree of built-in error-detection capabilities
- To ease the integration and acceptance of SW non developed with full compliance to safety lifecycle (e.g. library porting from consumer application).
ISO26262 supports such industrial need asking to enhance the safety architecture of the SW even at component level (SEooC concept) when this applies.

**Safety Analysis (SA)**

**Dependent Failure Analysis (DFA)**

**Main goals:**
1. to **support the safety concept verification** when is based on the independence/diversity of software functions/components
2. to **verify the coexistence criteria** among the software components
3. to **support the specification of safety mechanism** at software architecture level, in order to mitigate SW failure identified in the analysis

**Additional info on part 9, sec 7,8 and annex C, but not SW specific**

**Main techniques:** **SW-FMEA** (3) and **DFA** (1-2)

Such requirements were already present in Edition 1 (2011), but lack of experience in application push the committee to provide a **full informative annex (E) to guide industry** in the second edition (2018)
Main aim within the lifecycle is:

to support the **specification of safety mechanism at software architecture level**.

Output of the Activity:

- **modified architecture** to accommodate error detection and error recovery mechanisms (and proper reactions of the SW in line with original safety concept).

And/or...

- Evidence that existing **architecture is completely or partially fine as it is**.
- Additional **Assumptions of Use** for system level
Challenges and Opportunities

Challenges:
1. The inclusion of mechanism as deadline or control flow monitoring in SW architectures is not new in the safety industry, but this is mostly done based on experience without a complete formal modelling of the architecture and of the SW faults.
2. This inclusion is often done when dealing with the entire system architecture while it may be beneficial also if applied to parts of it (e.g. OS+middleware or complex libraries).
3. The new annex in ISO26262 provides some guidance (example-based) but still delegates the definition of a clear methodology in line with the aim of an informative text.

Opportunities:
1. Proposal of a clear methodology to perform such activities.
2. This is important particularly for SW as fault modelling and FMEA approaches are more understood and applied in the industry at HW and system level rather than SW.
3. In addition an important aspect, generally not fully considered when defining SW Safety Mechanisms, is to consider how the effectiveness of V&V activities affect the “likelihood” of some SW faults.
   - Here the point is trade-off architectural changes versus fault-removal techniques.
1. Short Company Introduction

2. SW Safety Analysis and DFA in Automotive

3. ResilTech Methodology

4. Feedback from application and future directions
ResilTech Methodology – Overall View

**Input**

- ISO 26262
- Functional Safety Concept
- Technical Safety Concept
- SW Architecture Design

**Process**

1. SW FMEA
   - safety concept includes requirements on independence, or SW modules in the architecture have different ASILs assigned?
   - Yes
     - Different ASIL?
       - No
         - No
         - Yes
           - SW component coexistence analysis
           - Common Cause Failure Analysis
           - Cascading Failure Analysis
   - No

**Output**

- new or updated safety mechanisms
- software architecture changes
- new or updated ASIL levels for software modules
ResilTech Methodology – Input 1/4

Data from ISO26262-6 Annex D-Freedom from interference between software elements

Constitute the reference set of guidewords to define failure modes in Safety Analysis and DFA.

Timing and execution
- blocking of execution
- deadlocks
- Livelocks
- incorrect allocation of execution time
- incorrect synchronization between software elements.

Memory
- corruption of content
- inconsistent data (e.g. due to update during data fetch)
- stack overflow or underflow
- read or write access to memory allocated to another software element

Exchange of information
- repetition of information
- loss of information
- delay of information
- insertion of information
- masquerade or incorrect addressing of information
- incorrect sequence of information
- corruption of information
- asymmetric information sent from a sender to multiple receivers
- information from a sender received by only a subset of the receivers
- blocking access to a communication channel

ISO 26262
Functional Safety Concept
Technical Safety Concept
SW Architecture Design
Safety Mechanisms from ISO26262-6 «Table 4 — Mechanisms for error detection at the software architectural level

Constitute the reference set to select the “intended safety mechanisms” in SW FMEA and DFA. It can be refined, depending on the characteristics of the project.

<table>
<thead>
<tr>
<th>Mechanisms</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a Range checks of input and output data</td>
<td>++</td>
</tr>
<tr>
<td>1b Plausibility check</td>
<td>+</td>
</tr>
<tr>
<td>1c Detection of data errors</td>
<td>+</td>
</tr>
<tr>
<td>1d Monitoring of program execution</td>
<td>o</td>
</tr>
<tr>
<td>1e Temporal monitoring of program execution</td>
<td>o</td>
</tr>
<tr>
<td>1f Diverse redundancy in the design</td>
<td>o</td>
</tr>
<tr>
<td>1g Access permission control mechanisms</td>
<td>+</td>
</tr>
</tbody>
</table>
ResilTech Methodology – Input

Error Handling from ISO26262-6 “Table 5 — Mechanisms for error handling at the software architectural level”

Constitute the reference set to select the “intended safety mechanisms and related error handling” in SW FMEA and DFA. It can be refined, depending on the characteristics of the project.

<table>
<thead>
<tr>
<th>Mechanisms</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a Static recovery mechanism</td>
<td>+</td>
</tr>
<tr>
<td>1b Graceful degradation</td>
<td>+</td>
</tr>
<tr>
<td>1c Homogenous redundancy in the design</td>
<td>+</td>
</tr>
<tr>
<td>1d Diverse redundancy in the design</td>
<td>o</td>
</tr>
<tr>
<td>1e Correcting codes for data</td>
<td>+</td>
</tr>
<tr>
<td>1f Inhibit access permission violations</td>
<td>+</td>
</tr>
</tbody>
</table>
ResilTech Methodology – Input 4/4

**Input**

**Functional safety concept (ISO26262-2):** specification of the *functional safety requirements*, with associated information, their allocation to architectural *elements*, and their interaction necessary to achieve the *safety goals*.

**Technical safety concept (ISO26262-2):** specification of the *technical safety requirements* and their allocation to *system elements* for implementation by the *system design*.

**Software Architecture Design:**

- **(software) Architecture (ISO26262-2):** representation of the structure of the *item or systems or elements* that allows identification of building blocks, their boundaries and interfaces, and includes the allocation of requirements to hardware and software *elements*.
- **Design (FP7 AMADEOS):** The process of defining an architecture, components, modules and interfaces of a system to satisfy specified requirement.
SW FMEA: Steps

Process

- SW FMEA
  - Granularity
  - Failure Modes
  - RPN
    - Likelyhood
    - Severity
    - Detectability
  - Status Classification

Company Confidential
SW FMEA: Granularity

**Process**

- It depends on the target software, as general rule:
  - Mandatory:
    - system APIs and APIs of components at the high level design.
  - Recommended:
    - subcomponent (module) levels. The level of details reached for the SW FMEA analysis depends on the complexity of the component and the design principles.
    - At least:
      - internal resource usage
      - internal IPC
      - timing
      - local scheduling and priority
## SW FMEA: Failure Modes

### Process

- Guidewords from ISO26262-6 Annex D
- Component failure modes, which can (should) be refined depending on the project:
  - Fails to execute or halts
  - Executes incompletely or concludes abnormally
  - Output incorrect, missing or late (includes possibility of returning no error or wrong error codes)
  - Incorrect timing – too early, too late, slow, etc..
  - Incorrect internal state change
  - Incorrect internal IPC
  - Incorrect local scheduling and priority
  - Incorrect internal resource usage (virtual/physical resources, computational power)
  - Erroneous data management and data corruption
  - Wrong calibration data
  - Others...

### Failure Modes

- **Granularity**
- **Failure Modes**
  - **RPN**
    - **Likelihood**
    - **Severity**
    - **Detectability**
- **Status Classification**
SW FMEA: Likelyhood

Process

- In case the target system is an evolution of an existing one:
  - History from bug reports
  - Code metrics as cyclomatic number, code smells detection (if applicable)

- Otherwise (no SW reuse), we can use design metrics as:
  - design complexity
  - configuration complexity (if applicable)
  - hardware, OS, libraries dependencies
  - Number of global values, of function/system status, dimension of data structures, shared resources

- These values could contribute to the definition of a concept of classes of likelihood of failures
## SW FMEA: Likelihood (example)

### Process

<table>
<thead>
<tr>
<th>Failure mode description for particular SW Component complexity parameters</th>
<th>Likelihood pre-V&amp;V</th>
<th>V&amp;V activity</th>
<th>V&amp;V efficiency</th>
<th>Likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time library exceeding time slot</td>
<td>Cyclomatic number $\geq Y$ &amp; LOC $\geq X$</td>
<td>High</td>
<td>WCET estimation</td>
<td>Low</td>
</tr>
<tr>
<td>Run time library exceeding time slot</td>
<td>Cyclomatic number $&lt; Y$ &amp; LOC $&lt; X$</td>
<td>Low</td>
<td>WCET estimation</td>
<td>Low</td>
</tr>
<tr>
<td>Run time library exceeding time slot</td>
<td>Cyclomatic number $\geq Y$ &amp; LOC $\geq X$</td>
<td>High</td>
<td>Complete set of performance / timing test</td>
<td>High</td>
</tr>
</tbody>
</table>
SW FMEA: Severity

- Severity of failures effects should be evaluated with respect to the design specification, the safety concept and the safety goal.
- It is difficult to provide objective ways to measure severity.
- We just distinguish in two classes: YES or NO
  - if the failure leads to the violation of a safety goal or a safety requirement it is classified: Severity= YES
  - otherwise it is classified: Severity= NO
SW FMEA: Detectability

The Failures detectability of mitigation items is the estimated ability of the selected countermeasures to detect and tolerate a given component failure.

We propose three classes on such coverage:

- **High**
  - For example, a checksum coverage that is adequate for ASIL B.

- **Medium**
  - For example, a range check, which is not able to detect approximation errors.

- **Low**
  - LOW shall be selected whenever evidence for High or Medium coverage cannot be provided.
SW FMEA: Risk Probability Number

Process

- Granularity
- Failure Modes
- RPN
  - Likelihood
  - Severity
  - Detectability
- Status Classification

**SW FMEA**

- Safe as is
- Acceptable
- Additional mitigation requested

**RPN**

- Low
- Medium
- High

**Severity (S)**

- Yes
- No

**Likelihood (L)**

- High

**Detectability (D)**

75° Meeting of IFIP Working Group 10.4 Champery, Switzerland.
24-28 January 2019 – Company Confidential
SW FMEA: Status Classification

Process

Open: the failure is not yet managed.
Mitigated: a safety mechanism was previously implemented in order to mitigate the failure.
Ignored: the impact of the failure is, to an acceptable extent, a minor one. It is not necessary to mitigate it. (severity=NO).
Closed: a new solution is introduced to mitigate the failure
Transferred: the failure is not mitigated; its propagation will be mitigated at a later phase. This generally means that new assumptions of use, modification of architectural design, or additional V&V activities are introduced and matched to mitigate this failure.
<table>
<thead>
<tr>
<th>ID</th>
<th>SW Component ID</th>
<th>Component Failure Mode</th>
<th>Component Failure Description</th>
<th>Likelihood</th>
<th>Effect description</th>
<th>Severity class</th>
<th>Existing mitigation and impact</th>
<th>Detectability</th>
<th>RPN</th>
<th>Intended Mitigations</th>
<th>Status (with RPN post mitigation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPP-2</td>
<td>C/C++ Runtime library</td>
<td>Crash</td>
<td>No computations performed/application crash. It happens when the result has NULL input iterators.</td>
<td>High</td>
<td>Application crashes</td>
<td>Yes</td>
<td>Application fails to contact the safety monitor. Safety monitor reports to MCU</td>
<td>High</td>
<td>Acceptable</td>
<td>AoU - Applications must report their state to the Health Monitor</td>
<td>Transferred</td>
</tr>
<tr>
<td>CPP-4</td>
<td>C/C++ Runtime library</td>
<td>Error in implementation of exception handling</td>
<td>Possible issues: * Stack not correctly unwound * Exception not thrown, wrong exception thrown * Memory not available for exception handling</td>
<td>Low</td>
<td>Wrong execution flow, memory leaks</td>
<td>Yes</td>
<td>Full validation and code developed according to ISO 26262 part 6.</td>
<td>High</td>
<td>Acceptable</td>
<td></td>
<td>Mitigated</td>
</tr>
<tr>
<td>CPP-5</td>
<td>C/C++ Runtime library</td>
<td>Input not accepted</td>
<td>Dynamic memory allocation fails. This can happen e.g. if dynamic memory fails within the RT</td>
<td>High</td>
<td>Computation not performed and error code returned</td>
<td>Yes</td>
<td>Error code returned to the application that can take corrective measures.</td>
<td>High</td>
<td>Acceptable</td>
<td>AoU - Applications must handle error status</td>
<td>Transferred</td>
</tr>
<tr>
<td>CPP-8</td>
<td>C/C++ Runtime library</td>
<td>IEEE exception</td>
<td>Executes incompletely and returns error code</td>
<td>High</td>
<td>Computation not performed and error code returned</td>
<td>Yes</td>
<td>Error code returned to the application that can take corrective measures.</td>
<td>High</td>
<td>Acceptable</td>
<td></td>
<td>Mitigated</td>
</tr>
</tbody>
</table>
ResilTech Methodology – Overall View

Input

ISO 26262
Functional Safety Concept
Technical Safety Concept
SW Architecture Design

Process

SW FMEA

safety concept includes requirements on independence, or SW modules in the architecture have different ASILs assigned?

Yes

No

Dependent Failure Analysis

Cascading Failure Analysis

Common Cause Failure Analysis

Different ASIL?

Yes

SW component coexistence analysis

No

Output

new or updated safety mechanisms
software architecture changes
new or updated ASIL levels for software modules

75° Meeting of IFIP Working Group 10.4 Champery, Switzerland.
24-28 January 2019 – Company Confidential
The goal is
• to identify and analyze the possible common cause and cascading failures between supposedly independent software elements,
• to assess their risk of violating a safety goal (or derived safety requirements)
• to define new safety measures to mitigate such risk if necessary.

Steps:
– Software component independence analysis
  • cascading failures analysis
  • common cause failures analysis
– In case of sub-elements with different ASILs
  • Software component coexistence analysis
SW DFA: Cascading Failure Analysis

Process

It refers **exclusively to software** and it is organized in the following steps:

**Step 1.** A checklist to define failures that may propagate through a failure chain is identified. Each element is numbered with an ID.

**Step 2.** Identify couples of SW components to be checked for independence based on the requirements of independence (e.g., parallel elaboration with diverse algorithm). Each set is numbered with an ID.

**Step 3.** A guideword-based analysis is applied to each set, to understand the impact of such failures from a system-level point of view.
SW DFA: Cascading Failure Analysis: checklist

**Process**

**Step 1: checklist (to be refined and tailored for each project)**

– Some Examples:

<table>
<thead>
<tr>
<th>ID</th>
<th>Category</th>
<th>Element</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing_1</td>
<td>Timing and execution</td>
<td>Block of Execution</td>
<td>Does a block of execution of the SW component impact the destination SW components?</td>
</tr>
<tr>
<td>Timing_3</td>
<td>Timing and execution</td>
<td>Deadlocks</td>
<td>Are there potential situations where the SW component experiments deadlocks? (e.g. locking mutexes, waiting for the return value of a function, etc.)</td>
</tr>
<tr>
<td>Timing_5</td>
<td>Timing and execution</td>
<td>Execution Time</td>
<td>Is the SW component taking too much time to execute? Or is it too fast? Or starts at a wrong instant?</td>
</tr>
<tr>
<td>Memory_1</td>
<td>Memory</td>
<td>Corruption of content</td>
<td>Check the possible propagation of corrupted data from the source to the destination SW components.</td>
</tr>
<tr>
<td>Memory_3</td>
<td>Memory</td>
<td>stack overflow/underflow</td>
<td>Check for possible stack overflow/underflow during memory usage</td>
</tr>
<tr>
<td>Information_1</td>
<td>Exchange of Information</td>
<td>Information repetition</td>
<td>Does a potential information repetition create a cascading failure in the destination SW components?</td>
</tr>
<tr>
<td>Information_2</td>
<td>Exchange of Information</td>
<td>Loss of data</td>
<td>Does a potential loss of data create a cascading failure in the destination SW components?</td>
</tr>
<tr>
<td>Information_6</td>
<td>Exchange of Information</td>
<td>Incorrect sequence</td>
<td>Does a potential incorrect sequence of information create a cascading failure in the destination SW components?</td>
</tr>
<tr>
<td>Information_7</td>
<td>Exchange of Information</td>
<td>Corruption</td>
<td>Does a potential corruption of information create a cascading failure in the destination SW components?</td>
</tr>
</tbody>
</table>
## SW DFA: Cascading Failure Analysis: resulting table (example)

<table>
<thead>
<tr>
<th>ID</th>
<th>ID of SW components under analysis</th>
<th>Cascading failure checklist</th>
<th>Failure Description</th>
<th>Likelihood</th>
<th>System Failure Mode (effect)</th>
<th>Severity Class</th>
<th>Existing mitigation and impact</th>
<th>Detectability</th>
<th>RPN</th>
<th>Intended Mitigations</th>
<th>Status (with RPN post mitigation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCF-5</td>
<td>(1 application using C/C++ Runtime, with certain requirements on independence)</td>
<td>Timing</td>
<td>A completion or failure indication is received too late - A job hasn’t completed on time</td>
<td>HIGH</td>
<td>Both applications cannot proceed</td>
<td>YES</td>
<td>watchdog timer expired to indicate to the applications that the job hasn’t completed</td>
<td>HIGH</td>
<td>Acceptable</td>
<td></td>
<td>Closed</td>
</tr>
</tbody>
</table>

75° Meeting of IFIP Working Group 10.4 Champery, Switzerland. 24-28 January 2019 – Company Confidential
SW DFA: Common Cause Failure Analysis

Process

It refers **exclusively to software** and it is organized in the following steps:

**Step 1.** A set of guidewords for events or root causes that may be cause of common failures of software elements is identified. Each keyword is numbered with an ID.

**Step 2.** Identify **couples of SW components**
- Typically, these are elements that:
  - Realize safety-critical functionalities through software diversity.
  - Are replicated software, running on the same hardware.
  - Implement redundant functionalities:
    - This item includes the redundancy of a safety mechanisms with respect to a target element.

**Step 3.** A guidewords-based analysis is applied to each set, to understand the impact of such failures from a system-level point of view.
**SW DFA: Cascading Failure Analysis**

**Step 1: checklist (to be refined and tailored for each project)**

**– Some Examples:**

<table>
<thead>
<tr>
<th>ID</th>
<th>Domain</th>
<th>Event or Root Cause</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>Spatial</td>
<td>Misbehaviour of a shared resource or service</td>
<td>Any software element can act as a shared resource or service (remember that these are resource or service external to the elements sets that will be identified in Step 2). However, from experience, we recommend attention to: software libraries, drivers, services, files, algorithms, virtual communication channels, IPC mechanisms, signals, calibration data, data.</td>
</tr>
<tr>
<td>E2</td>
<td>Spatial</td>
<td>Unavailability of a shared resource or service</td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td>Temporal</td>
<td>Slow shared resource or service</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>ID of couple of components under analysis</td>
<td>Common cause failure checklist</td>
<td>Failure Description</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------</td>
<td>------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CCF-5</td>
<td>(2 applications using C/C++ Runtime, with certain requirements on independence)</td>
<td>Temporal</td>
<td>A completion or failure indication is received too late - A job hasn't completed on time</td>
</tr>
</tbody>
</table>
SW DFA: SW Component Coexistence Analysis

Process

- The goal is to check that lower ASIL component failures do not impact on higher ASIL components
  - It should be expected that the pair of components have been already investigated by the cascading failures analysis
  - In this case, the checklist for cascading failure is re-used
- Output may be a new ASIL level for the SW component. In fact, status may have values:
  - **No impact**: failure of the lower-ASIL or QM component have no effect on the higher ASIL element
  - **New ASIL**: failure of the lower-ASIL or QM component propagates to the higher ASIL element, and a new evaluation of assigned ASIL is required
  - **Architecture review**: assigned ASILs are not changed but architecture is reviewed.
### SW DFA: Common Cause Analysis: resulting table (example)

<table>
<thead>
<tr>
<th>ID</th>
<th>ID of SW components under analysis</th>
<th>Cascading failure checklist</th>
<th>Component failure description</th>
<th>Likelihood</th>
<th>System Failure Mode (Effects)</th>
<th>Severity class</th>
<th>Detectability</th>
<th>RPN</th>
<th>New ASIL</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>COA-1</td>
<td>Compute runtime (ASIL B), Application (ASIL D)</td>
<td>Timing</td>
<td>A completion or failure indication is received too late - A job hasn't completed on time</td>
<td>HIGH</td>
<td>Application cannot proceed</td>
<td>YES</td>
<td>External watchdog timer expired to indicate to the applications that the job hasn't completed</td>
<td>HIGH</td>
<td>ASIL D</td>
<td>New ASIL</td>
</tr>
</tbody>
</table>
1. Short Company Introduction
2. SW Safety Analysis and DFA in Automotive
3. ResilTech Methodology
4. Feedback from application and future directions
Feedback from application 1/2

• Positive
  – Having a clear method to follow
    • *To verify the completeness of the Safety Requirements and Mechanisms and also Assumption of Use in particular in case of SEOOC*
    • *Standardize requirements for supplier: most important for long supply-chain as in automotive*
  – Having a guideline on selective application of Safety Mechanisms (run-time)
  – Good acceptance from Quality Departments
Feedback from application 2/2

• Negative
  – Effectiveness of analysis highly depend on detailed SW architecture design
    • Typically not available when it should
  – Once a potential safety impact is found it is not always straightforward to motivate usage of on-line error detection and mitigation techniques versus process oriented solutions (e.g. "improve" SW testing)
Future directions

- Developing low complexity modelling facilities
- Running model execution to evaluate “severity” prior to SW development
- Connection with Fault Injection campaign to validate “detectability” post development
- Formalized / semiformalized SW architecture would allow to be input for semi-automatic analysis
  - Despite a number of tools and methodologies available in last decades adoption from industry is still far from becoming a common practice