Avoiding Top Mistakes in Safety Critical Software Development

Almost Famous Quotes

"Safety-Critical may seem subjective … but your software cannot be."

--Vance Hilderman, CEO AFuzion Incorporated, 2016

Agenda

- *Intro Quick Requirements Quiz*
- *Intro to Safety-Critical Requirements*
- *Safety-Critical Requirements Overview*
- *Requirements Examples: Weak versus Stronger*
- *Safety-Critical Requirements Best Practices*
- *Requirements Management Case Study*
- *Answers to Quiz*
- *Question & Answer Session*

About Today's Presenters: Vance Hilderman from AFuzion Inc.

- ***AFuzion Inc.:***
  - ✓ Current staff: 250+ safety-critical projects onsite in 35 countries
  - ✓ Safety-Critical Consulting Services, Training, Gap Analysis, Mentoring

A Safety-Critical Requirements Quiz: True or False?

1. T/F: On most safety-critical projects, most requirements pertain to Safety.
2. T/F: Safety-critical requirement development must follow a Waterfall.
3. T/F: The best way to assess safety-critical requirements is via test execution.
4. T/F: Safety-Critical standards (IEC, DO-XXX, ISO, etc) provide clear guidance for developing and assessing requirements.
5. T/F: Most safety-critical software defects are due to bugs and manufacturing defects.

(Answers will be provided at the end of this Session)

AFUZION

Safety-Critical– Definitions

- **Software Requirement:**
  - ✓ ***"Measurable software processing that is necessary."***

- **Safety-Critical Software Requirement:**
  - ✓ ***"A necessary aspect within a system whose anomalous behavior could negatively impact safety."***

Safety-Critical Systems - Basic Facts

- “Safety-critical” encompasses many domains: Industrial, Automotive, Aerospace, Nuclear, Medical, etc.
- Software size and complexity are rapidly increasing
- Yesterday’s non-safety-critical is becoming tomorrow’s safety-critical due to IoT, integration/connectivity, etc.
- Requirements are needed to guide development, identify hazard detection/mitigation, and assess implementation.

Safety-Critical Requirements – Background

- Experts say majority of safety-critical failures stem from requirements.
- Safety-critical requirements include Safety aspects, but not exclusively: also address Functional, Performance, etc.
- Most safety-critical requirements specifications are incomplete: lack complete hazard prevention/mitigation.
- Need requirement identification, specification, verification, and management.

Safety Goals Versus Requirements

- Safety Goals: conceptual desires regarding safety:
  - ✓ "The system must be safe."
  - ✓ "Serious accidents cannot occur."
  - ✓ "Personnel can never be killed or injured."
- Goals are not requirements: they cannot be guaranteed.

➢ **A major problem is specifying safety goals as if they were verifiable requirements.**

Safety Requirements – Generalized Examples

- "The XYZ System shall yield less than than X Type 7 Fault Incidents per year where "yield" is defined by ABC."
- "The system shall react to accidents of type X by performing Y."
- "The ABC system shall yield fewer than Y Type 4 Safety Incidents per 200,000 hours of operation."
- "The system shall have the ability to detect incidents of type ABC and report such incidents via the XYZ mechanism."

Mistake #1 - 4: Forgetting Requirements Best Practices

✓ **Requirements are the Foundation to Safe Safety-Critical Software**

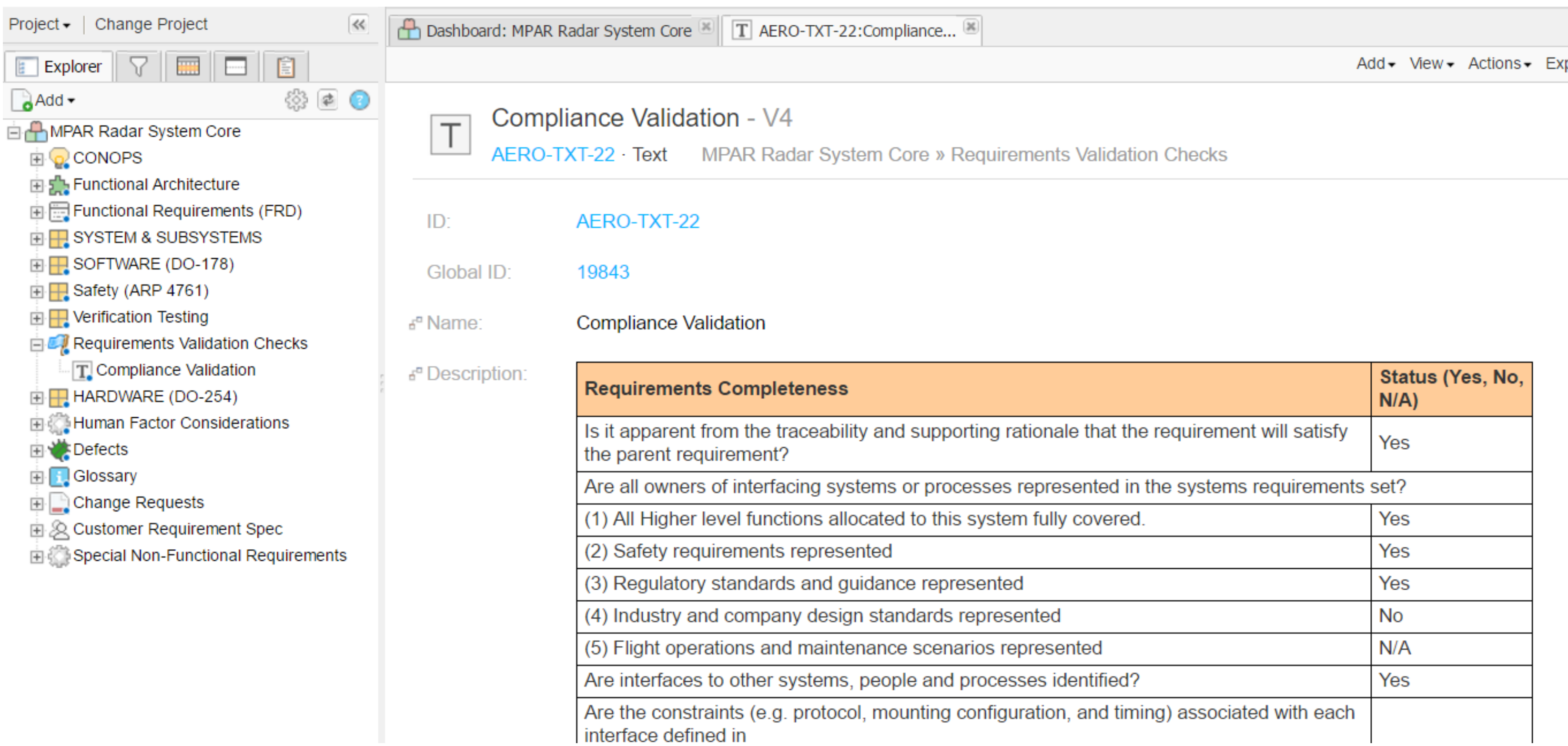
Mistake #1: Forgetting to Conduct Requirements Reviews Often

Review Requirements Early and Often – Make use of Lean and Agile

- **Conduct more frequent iterative reviews**
- **Review requirements in smaller batches**
- **Increase quality by having more thorough reviews & defined criteria**
- **Use collaborative techniques during review**

Mistake #2: Forgetting to Use Safety-Critical Requirements Checklists

Make use of checklists when performing requirements reviews



Requirements Completeness	Status (Yes, No, N/A)
Is it apparent from the traceability and supporting rationale that the requirement will satisfy the parent requirement?	Yes
Are all owners of interfacing systems or processes represented in the systems requirements set?	
(1) All Higher level functions allocated to this system fully covered.	Yes
(2) Safety requirements represented	Yes
(3) Regulatory standards and guidance represented	Yes
(4) Industry and company design standards represented	No
(5) Flight operations and maintenance scenarios represented	N/A
Are interfaces to other systems, people and processes identified?	Yes
Are the constraints (e.g. protocol, mounting configuration, and timing) associated with each interface defined in	

Mistake #3 Not Reviewing Software Requirements in Context

Review Requirements with their traces in context.

1 Set: TCS Low Level Software Requirements - V1

Review | Feedback | Stats | Tools

12 Items

Actions

Checklist

LLR

HLR

Test

5.2.3 TCS Low Level Software Requirements

5.2.3.1 Interfaces Checklist

ID	Topic	Criteria
6	Interfaces	Are the interfaces between the CE and the system defined?
7	Interfaces	Are communication protocols specified?
8	Interfaces	Is the memory map specified?
9	Interfaces	Is the pin list defined, including power and test pins?
10	Interfaces	Are internal communication protocols defined?
11	Interfaces	Is the data size and bit order defined?

There are no related items.

5.2.3.2 The FCU RVDT Voltage Feedback Current Value in engineering unit (mV) [OSIN_FCU_RVDT_VFBK.k] shall be read from the Operating System analog input array IN_OS_ANLG[0].

Hide related items

Upstream Items (1)

UAV-HLR-1 The CL shall read the FCU RVDT Voltage Feedback [OSIN_FCU_RVDT_VFBK] in engineering unit (mV) from the Operating S...

Downstream Items (2)

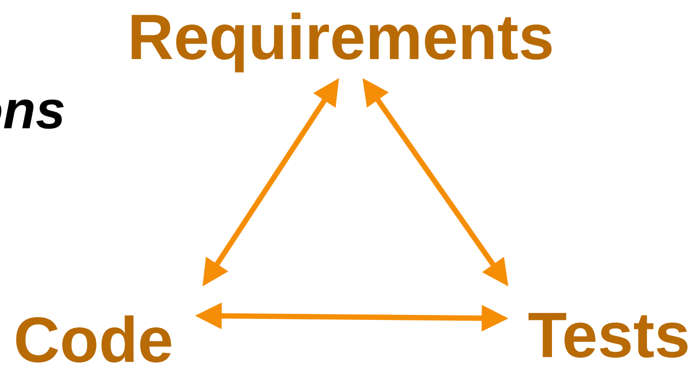
UAV-SW-2 n_VioltageFeedback

UAV-VER-27 Voltage Feedback Test

Mistake #4– Neglecting Traceability

When creating new requirements, tests cases, or code:

- ✓ **Establish the required traces immediately, not later when you need to demonstrate those traces to the auditor**
- ✓ **Make traceability updates mandatory for review of Requirements, Code, & Tests**
- ✓ **Two-way traceability for:**
  1. ***Requirements to Code Functions***
  2. ***Code Functions to Tests***
  3. ***Requirements to Tests***

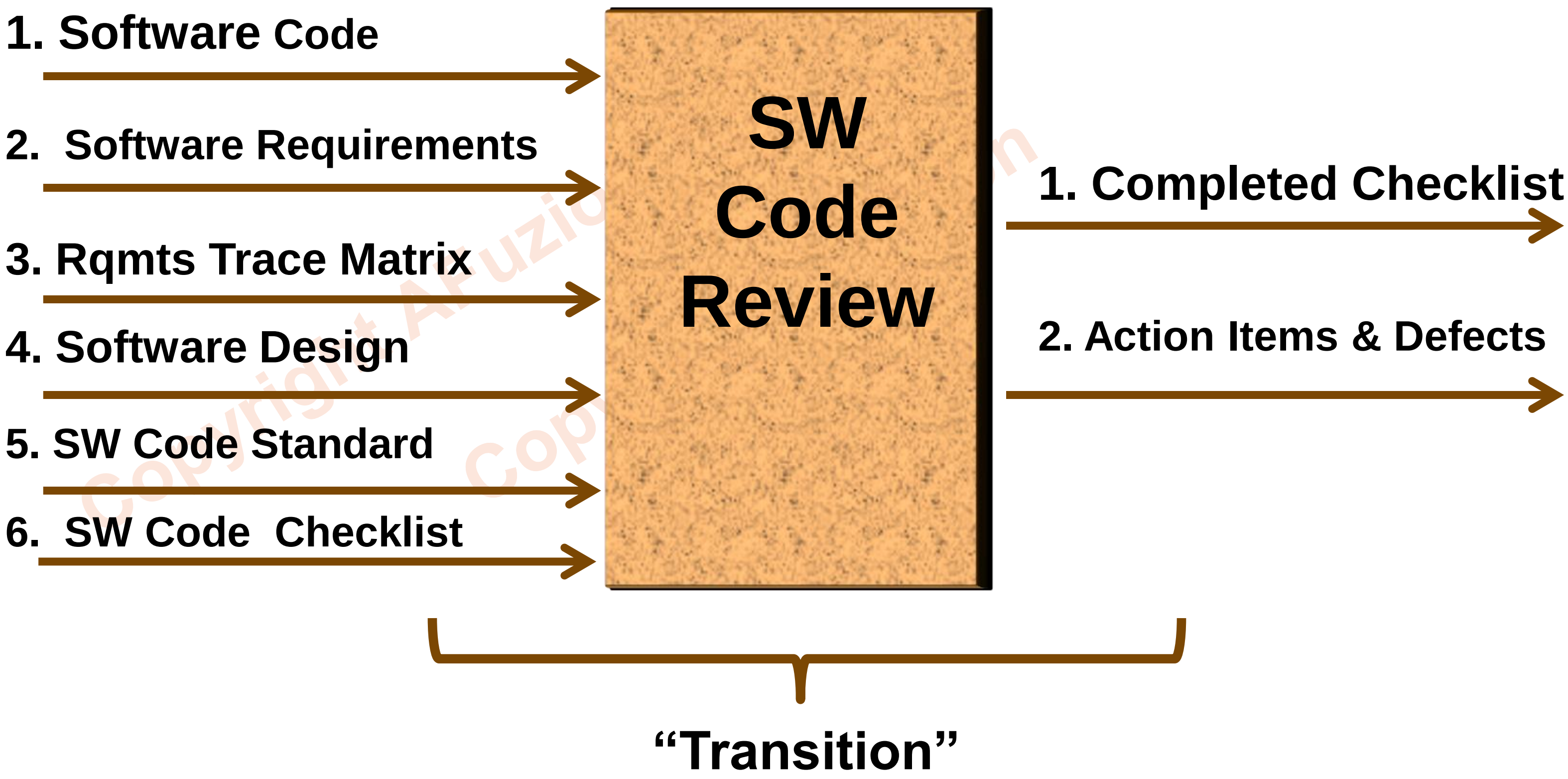


Mistake #5: Neglecting Transition Criteria

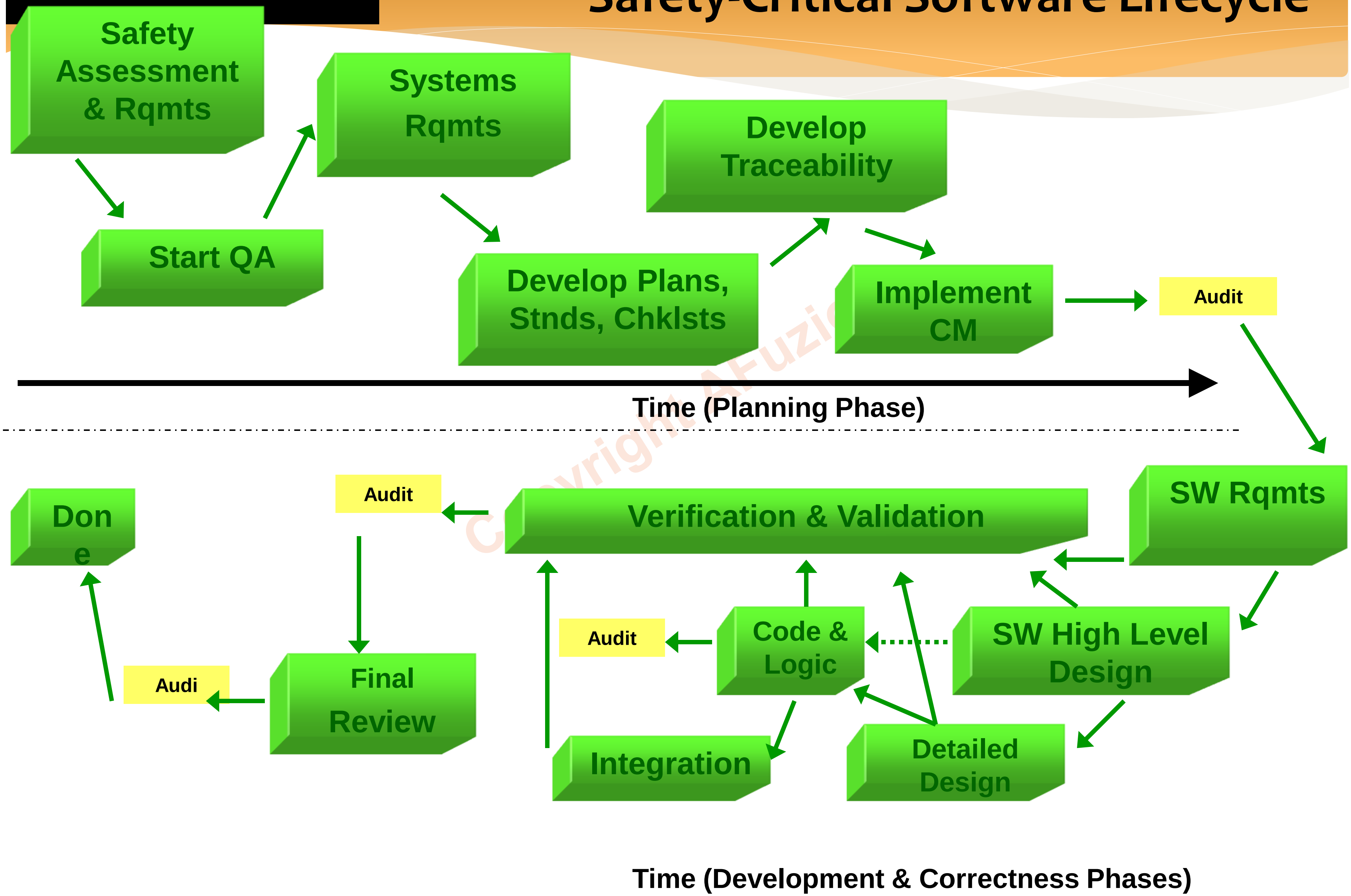
Question: What are Transition Criteria?!?

Mistake #5:
Forgetting Code Review "Transition Criteria"

* What are the Inputs & Outputs for a Software code Review?



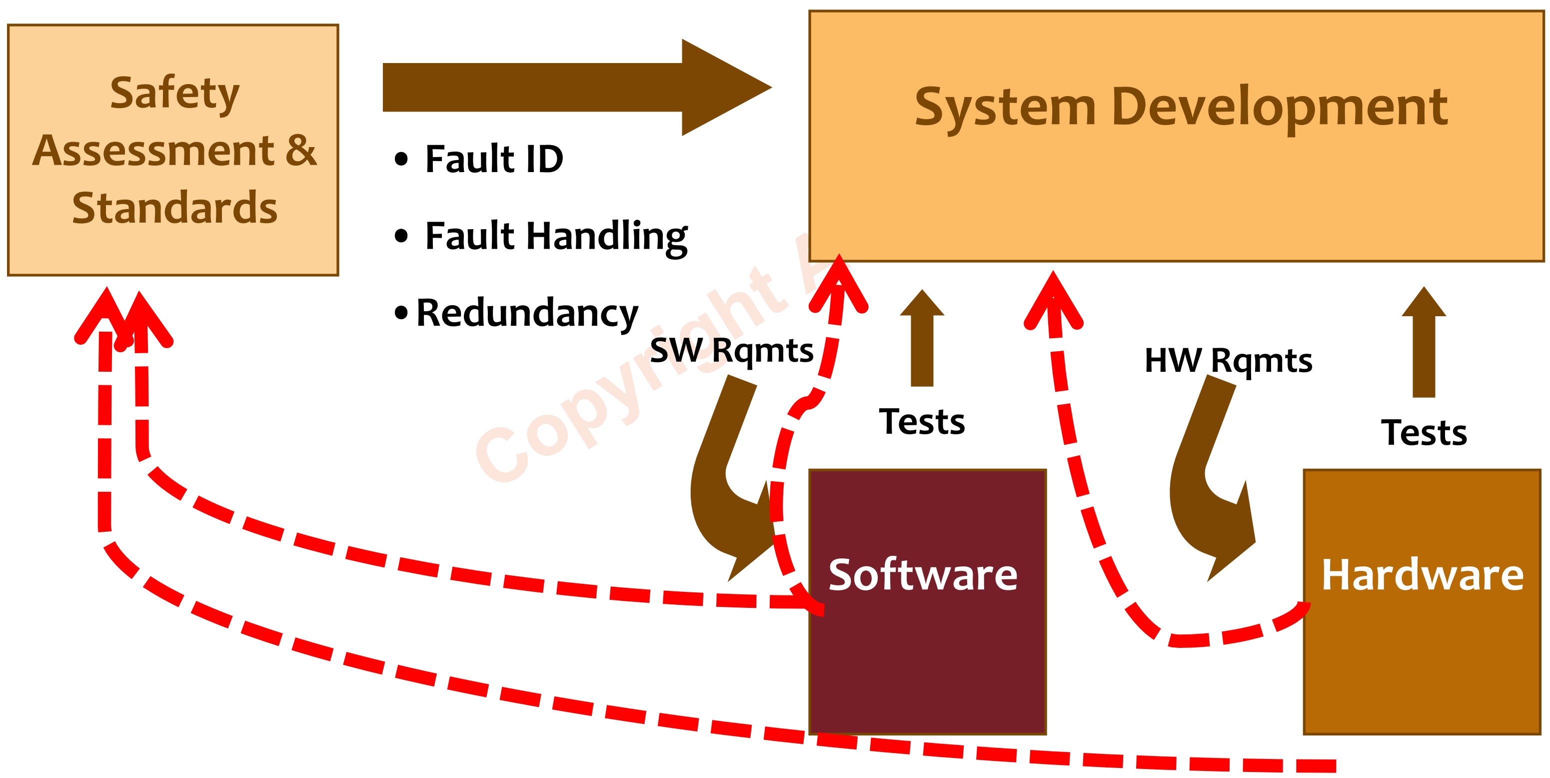
Mistake #6– Forgetting the AFuzion Safety-Critical Software Lifecycle



Mistake #7 – Safety "Before & During"

Mistake Neglecting to Apply Safety Standards BEFORE Software Development then During!

Remember:
Safety Regimen Feedback Loops



AFUZION

Mistake #8 – Neglecting Independence

#8: Neglecting "Independence"

- *Independence refers to the Correctness process (reviews and verification)*
- *Required Independence increases as criticality increases*
- *Not providing required Independence necessitates development rework*
- *If future criticality level changes and Independence lacking: rework.*
- *Recommendation: even when Independence is not required, do it: better review and insurance against future critical level increase*

Independence Question

* *What is the minimum number of persons required for developing an ASIL C Automotive project?*

(Hint: include QA person and Cert Authority, but what else?)

Mistake #9 – Inadequate Plans or not Following!

#9 Inadequate formal plans and not following them

- *Plans that are not compliant to standards*
- *Lack of checklists*
  - *E.g. Code reviews without requirements traceability to code*
- *Plans are not complete prior to starting associated developments*
- *Plans not QA approved*
- *Plans too detailed, requiring extreme customization for each project*

Mistake #10

8. Lack of Automated Testing = Expen$ive Regression Test
   - *Testing is a "life-of-product" continual activity*
   - *Testing costs exceed development costs, over product lifetime*
   - *Guilty-until-proven innocent regression: retest everything unless you can ensure no side effects*
   - *Best regression test strategy? Retest everything (only practical via automated testing)*

Mistake #11: When to "Verify"?

Mistake: Writing Test Case AFTER software developed

Best Practice:
Write Tests BEFORE Code

Traditional Software Engineering Sequence:

1. Write Software Requirements
2. Review Software Requirements
3. Develop Software Design
4. Review Software Design
5. Write Software Code
6. Review Software Code
7. Write Software Test Cases
8. Execute Software Test Cases
9. Review Software Tests & Results

AFUZION

Best Practice: Write Tests BEFORE Software

Recommended “Best Practice” Software Engineering Sequence:

1. Write Software Requirements
2. Review SoftwareRequirements, by writing Test Cases!
3. Develop Software Design
4. Review Software Design
5. Write Software Code
6. Review Software Code
2. Write Software Test Cases Based on SW Requirements
7. Execute Software Test Cases
8. Review Software Tests & Results

A Safety-Critical Requirements Quiz: Answers

1. T/F: On most safety-critical projects, most requirements pertain to Safety. **FALSE**

2. T/F: Safety-critical requirement development must follow a Waterfall. **FALSE**

3. T/F: The best way to assess safety-critical requirements is via test execution. **FALSE**

4. T/F: Safety-Critical standards (IEC, DO-XXX, ISO, etc) provide clear guidance for developing and assessing requirements. **FALSE**

5. T/F: Most safety-critical product defects are due to bugs and manufacturing defects. **FALSE**

DO-178C

DO-254

AFuzion: When Safety is Critical.™

DO-278A

ISO 26262

Conclusion Q & A

For free technical safety-critical whitepapers, see
www.afuzion.com