



Safe Maritime ICT



Presentation at “Maritim Innovasjon”
Høvik 28. November 2007

DNV Research/SKR/
November 2007

ICT in essential maritime systems

MANAGING RISK



Safe Maritime ICT research project

MANAGING RISK



- Software has become an integral part of any maritime installation and the amount of software is increasing
- A joint industry research project August 2006 – May 2008
 - Budget: ca. 7,7 mill NOK. Partly funded by the Norwegian Research Council
 - 8 project partners plus Norwegian Research Council
- We search for the solutions to future software challenges in maritime software intensive systems
- Three research topics
 - WP1: Increased use of COTS/SOUP in maritime systems: How to deal with this challenge?
 - WP2: Independence / redundancy: how to deal with this requirement with respect to software?
 - WP3: Emergency handling: How to operate the ship if the automation systems fail?

Project partners:



Software affects safety I



What	Number
Incidents reported by the SMICT partners	39
Potentially critical incidents	20
The incidents were caused by:	
- Caused by poor programming	7
- Caused by poor design/specification	10
- Caused by incorrect parameterization	3
- Caused by poor integration	6
- SW did not take into consideration HW failure	7
- Unknown failure source (Complex)	6

- Outside the SMICT consortium we have gathered information on 8 other incidents caused by software failure.

Software affects safety II



The photo is for illustration only and has no connection to the particular incident described on this foil

- When: 2005
- Initial source of incident: Faulty driver in the software of the ESD system.
- The incident: A voltage pulse caused noise on a communication link, and a driver in the ESD system interpreted this as faulty information from the IOs and terminated the communication with the IOs without sending information about this further up in the system.
- Consequence: Minor, limited to some extra work and 8 hour delay of testing and demonstration for class approval

Software affects safety III



The picture is for illustration only and has no connection to the particular incident described on this foil

- When: 2003
- Initial source of incident: Two redundant GPS systems failed simultaneously
- The incident: The two GPS systems simultaneously changed “correct” position by 70 m. The software did not detect this failure, and consequently the DP system changed the rig position by 70 m.
- Consequences: Considerable economical losses. The drilling string was cut, and the rig was out of operation for several days.

Incident 1: PSV partly loss of propulsion

MANAGING RISK 



- When: 2005
- Initial source of incident: Control software "hang-up".

Incident 2: Erroneous stop of engine

MANAGING RISK 



- When: 2002 ± 1
- Initial source of incident: Loss of a continuous "running" signal from engine to IAS (Integrated Automation System).

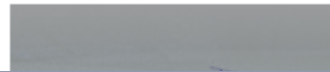
Incident 3: Handling of HW failure

- Initial source of incident: Hardware failure.
- The incident: Inadequate handling of available events.



Incident 5: Input out of range

MANAGING RISK 



- When: Summer 2006
- Initial source of incident: Input value

Incident 6: Process Station (PS) shut down

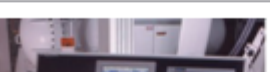
MANAGING RISK 



- Initial source of incident: A software component failed in the setting in which it was used

Incident 7: Operator Station (OS) failure

- Initial source of incident: Software failure.
- The incident: Inadequate handling of specific incidents.



Incident 8: Faulty embedded software

MANAGING RISK 



- When: 2004
- Initial source of incident: Faulty communication card between PS (Process Station) and IO card

Incident 9: HW fault tolerance

MANAGING RISK 



- Initial source of incident: HW failure in an IO card
- The incident: This failure is related to a

Incident 10: Unidentified fault

- Initial source of incident: Unidentified fault.
- The incident: Current situation.



Incident 11: Emergency Shut Down (ESD) system failure

MANAGING RISK 



- When: 2005
- Initial source of incident: Faulty driver in the software of the ESD system.

Incident 12: Loss of propulsion

MANAGING RISK 



- When: 2000
- Initial source of incident: Fault in software which should interpret data

Incident 13: Partly loss of propulsion

- Initial source of incident: Software failure.
- The incident: Inadequate handling of values in



Incident 14: Blocking of valve operation

MANAGING RISK 



- When: 1994, but fault discovered 8 years later
- Initial source of incident: Software did not handle a leaky valve
- The incident: The software should block

Incident 15: "Deadlock" because of line failure

MANAGING RISK 



- This is a general problem and not related to one specific incident. The problem is that signals can be locked as a consequence of a line failure, and this again might lead to a shut down which it is difficult to recover from without either

Incident 16: Interface problems

- Very little but the problem produced some speed of



Incident 17: Loss of manoeuvrability

MANAGING RISK 



- When: Approx. 1992
- Initial source of incident: Poor software specification

Incident 18: Network jamming

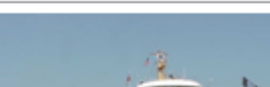
MANAGING RISK 



- The interviewee had little information about this incident. All he knew was that a hub had jammed the communication network

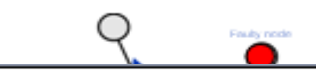
Incident 19: Intersystem communication

- Initial source of incident: Communication failure.
- The incident: Inadequate handling of communication



Incident 20: Network storm

MANAGING RISK 



- When: November 2005
- Initial source of incident: Network code not made according to standard specification

Incident 21: No possibility for reverse of engine

MANAGING RISK 



- Initial source of incident: A fault in the software caused the incident, no

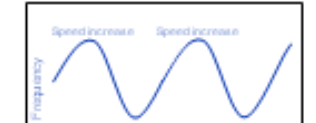
Incident 22: Corruption of speed reference

- Initial source of incident: Software fault in the



Incident 23: Integration problem

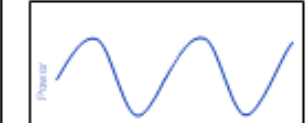
MANAGING RISK 



- Initial source of incident: Two different systems were not correctly integrated.
- The incident: The software from the automation system supplier tried to increase the speed, resulting in a

Incident 24: Power fluctuation

MANAGING RISK 



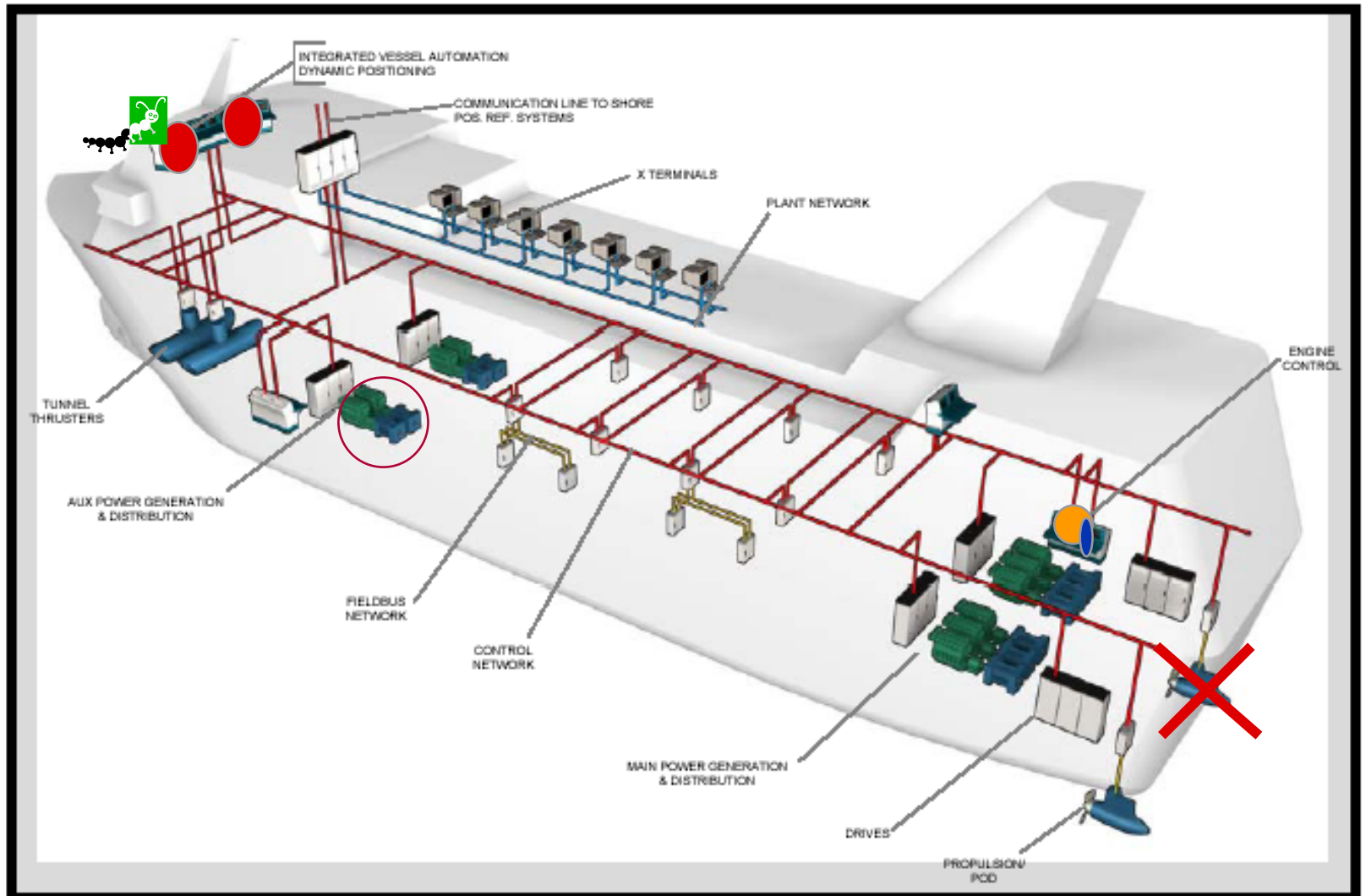
- Initial source of incident: Wrong parameterization of engine load limiter software
- The incident: The protection against engine overload (a limiter function) worked too "aggressively". This led to

Incident 25: Incorrect calculation of m

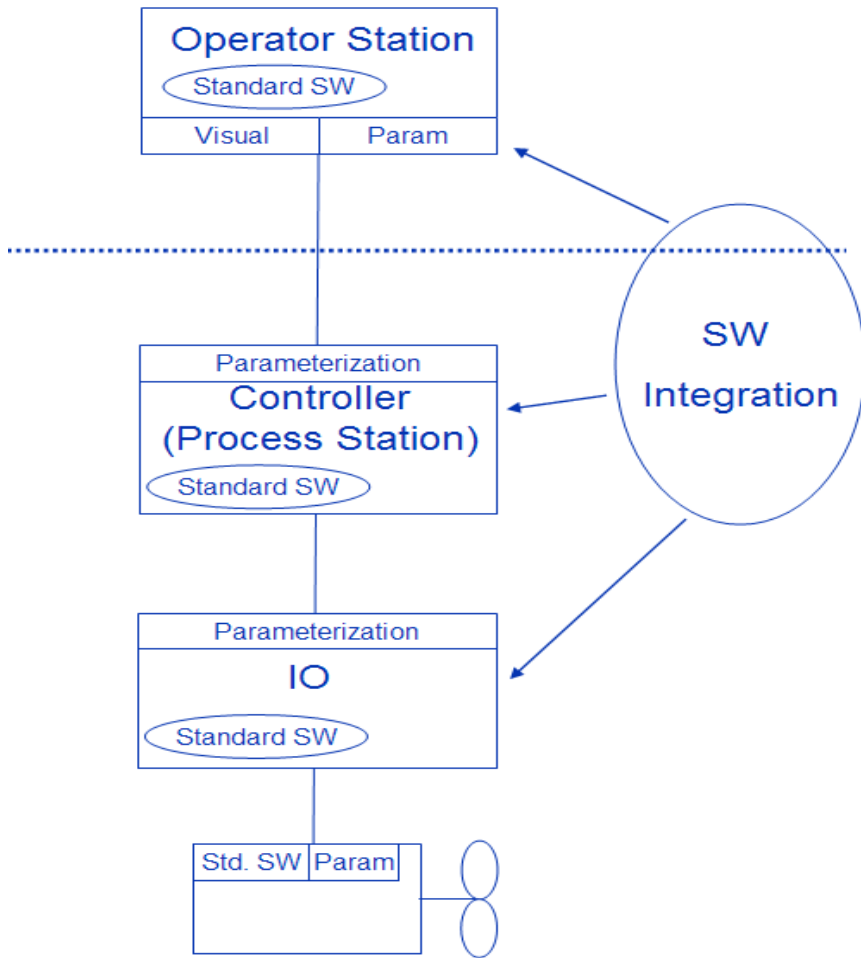
- Initial source of incident: Software fault in the
- The incident: The change take into account propulsion



Example of passenger ship subsystems



Software dominates in all parts of an IAS



- Three main “types” of software:
 - Standard (Base) software
 - Integration software
 - Parameterization
- Standard software is used unchanged in several different system deliveries
- Much of the integration software and parameterization is unique for each system delivery.
- Often produced under non-ideal working conditions

Overall philosophy: Proposed Approach

- Quality assessment of software in an Integrated Automation System (IAS) should:
 - **Product** focus on the standard components
 - **Process** focus on the overall engineering project
 - **Competency** focus for personnel doing commissioning and parametrization

- The quality assessment process should be flexible and efficient

- We recommend use of Safety Case
 - Safety Case allows the supplier of the system to have a degree of freedom in how the quality is documented

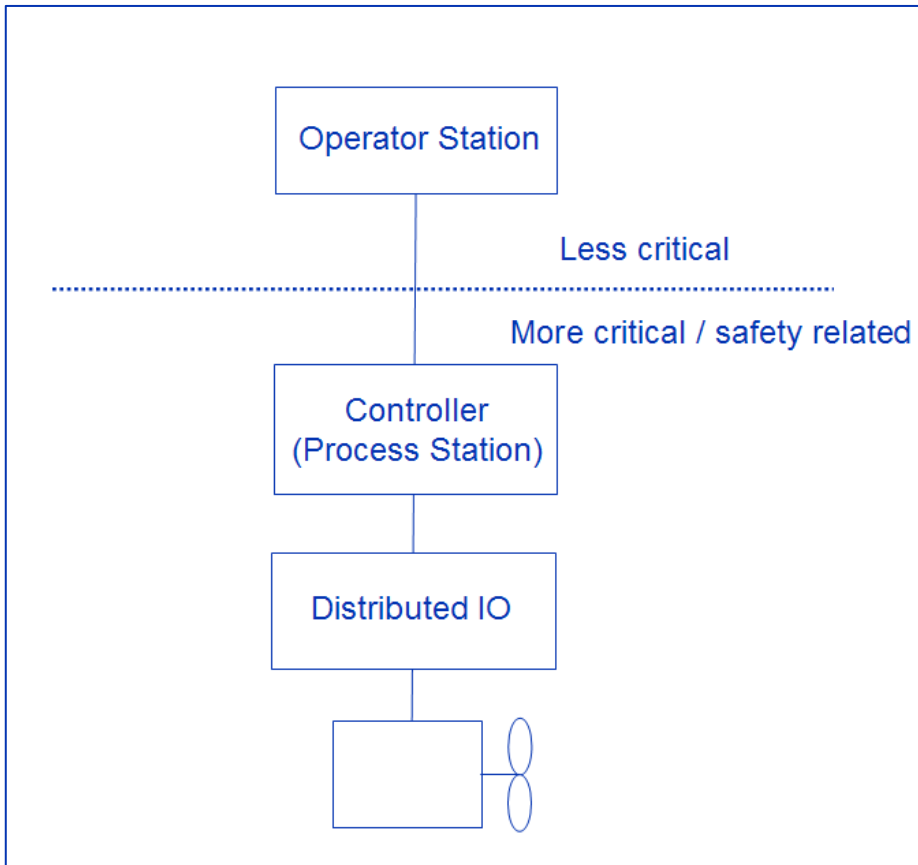
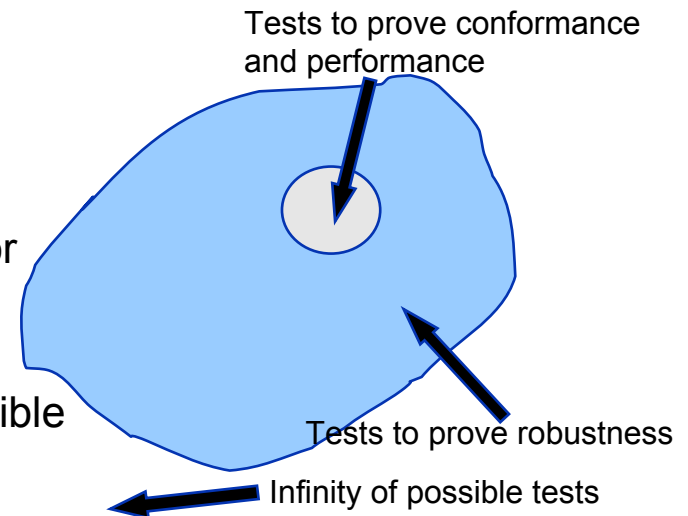


Figure 2: Typical architecture of an Integrated Automation System

- The trend:
 - *Use of COTS is widespread and increasing.*
 - *COTS components are covering more and more of the functionality*
- Few of the suppliers have a policy for when to use/not to use COTS
- More COTS is used in the Less critical parts of the IAS than in the more critical parts.
- The quality of most COTS is uncertain and there are considerable risks connected to use of COTS in safety related systems

- Certification of COTS components
 - Not plausible for the “Generic, not made of maritime use” sub category
- Proven in use
 - Mentioned by all the interviewees
 - Potential problems:
 - There has to be a first time use, i.e. difficult to apply to customized software
 - Require good failure reporting routines
- Thorough testing
 - Difficult to avoid, applies to all the different sub categories
 - Difficult to design test cases with high coverage for complex systems
 - Testing is suited to prove conformance and performance, but it is much more difficult – if possible at all - to prove the robustness through testing.



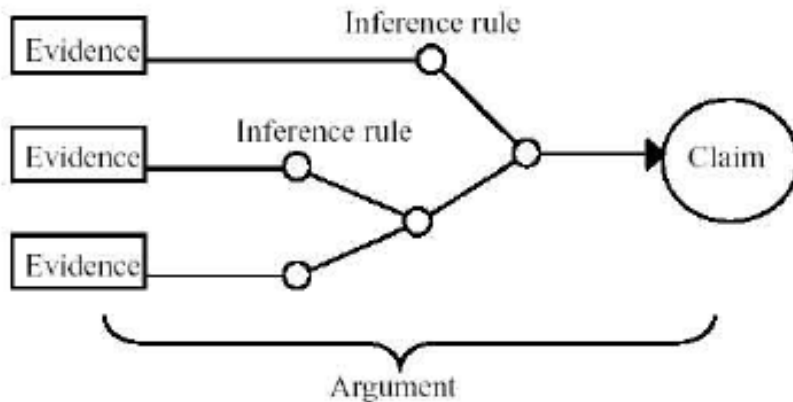
■ Control of input parameters

- Several interviewees pointed out incorrect input as one of the most common faults
- Useful mitigating action for all the COTS sub categories
- Can increase the effect of testing since constraints of input reduce the number of possible execution paths and thereby reduce the scope of the testing

■ Assessment of supplier's work processes

- Best suited for large and complex systems where it is difficult to assess the quality by testing and inspection of code
- Need insight into the suppliers' work processes
- Likely to be most applicable for sub contractors

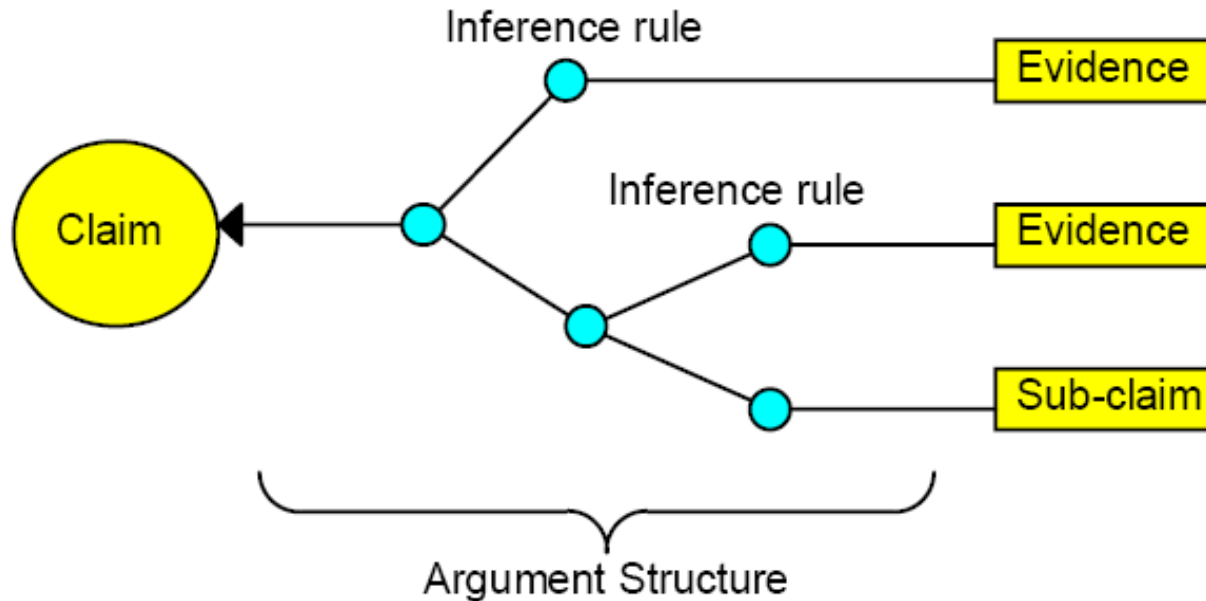
- COTS Policy: The supplier applying should have a written policy for when to use, and when to not use COTS
- Risk Analysis: Assessment of COTS: begin with a risk analysis to identify critical components
- Build Safety Case: Build safety case according to the information available for the COTS
 - process based assessment
 - product based assessment
 - testing
 - fault injection
 - proven-in-use
 - architectural measures
- Final decision: The strength of arguments, claims and evidence will be the basis for whether the COTS can be accepted used



- The validity of a safety case is dependent on the belief of the assessor, and if the safety of a component could be proven just by examining the product, or proven in use - not the developmental process, - and documented in a safety case, this could be valuable for assessment of COTS components.

- A safety-case is a documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment.
- A safety case must contain enough information to give a clear impression of the system's safety properties and indicate where more details about the system can be found if this is needed
- Class may assess a safety case, rather than the product it refers to.

Safety Case Principle



Claim: a property of the system or some subsystem

Evidence: the basis of the safety argument. This can be either facts, assumptions, or sub-claims derived from a lower level sub-argument

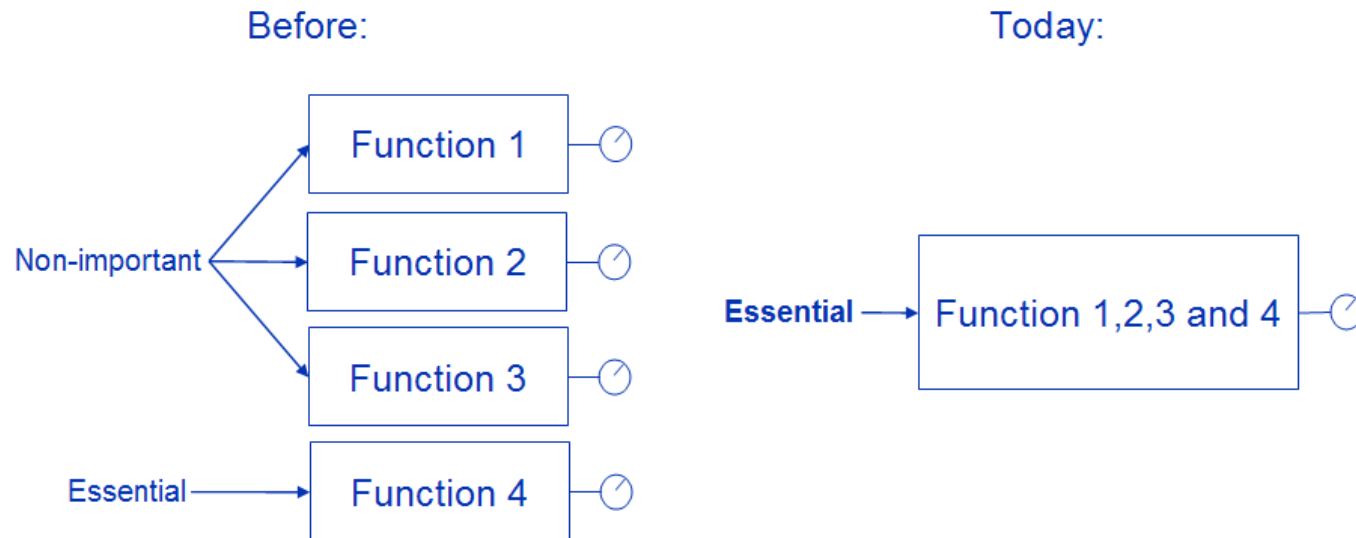
Argument: linking the evidence to the claim. Can be deterministic, probabilistic or qualitative

Inference rule: provides the logical basis for the steps in an argument

- Essential system: a system supporting services which need to be in continuous operation to maintain the crew's ability to manoeuvre the vessel (propulsion and steering)
- Redundancy: is the ability to maintain or restore a function when a failure has occurred
 - Guidance note: Redundancy is defined as two mutually independent systems that can maintain a function. The systems may be of a different type or have different functionality
- Diversity: realization of the same function by different means. For example
 - different processors, programming languages, algorithms, development teams
- Independence: Mutual dependence between components means that the function of the components and their power supply is not dependent on some common component or system
 - System B is independent of system A when any single system failure occurring in system A has no effect on the maintained operation of system B. A single system failure occurring in system B may have an effect on the maintained operation of system A
 - Two systems are mutually independent when a single system failure occurring in either of the systems has no consequences for the maintained operation of the other systems according to above

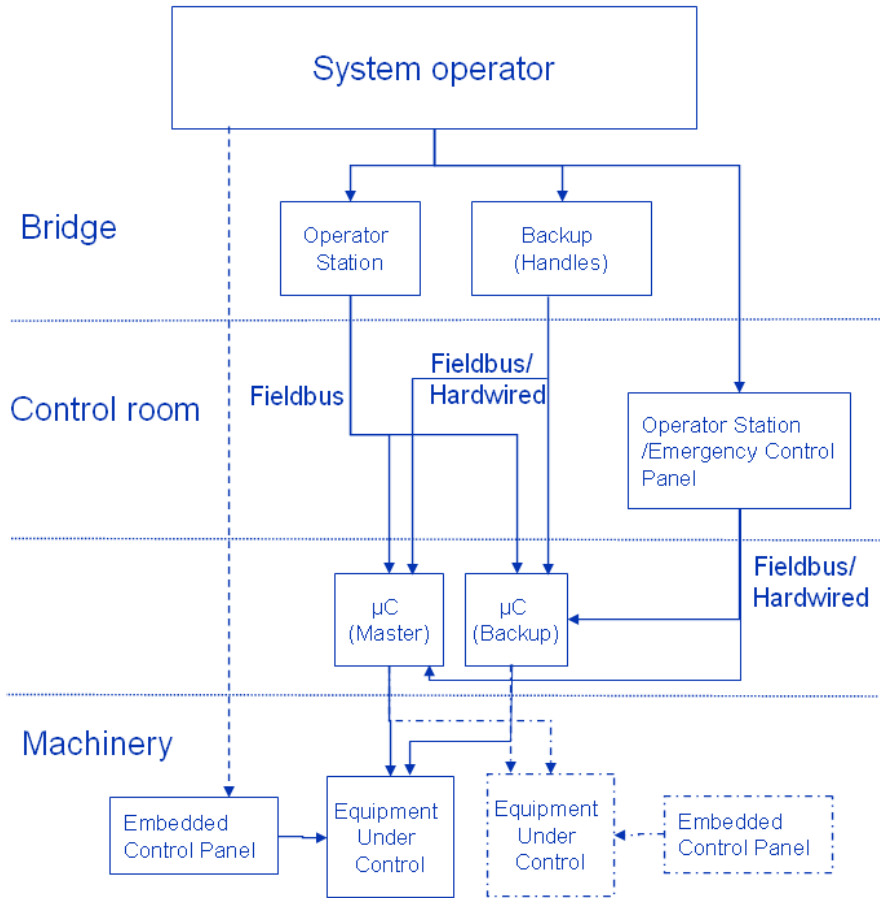
More integration: "More becomes essential"

- When a component supporting essential functions is integrated with other components supporting non-essential functions (important, non-important), the new component as a whole will have to be treated as essential if this integration may lead to former non-important functions affecting the performance of essential functions

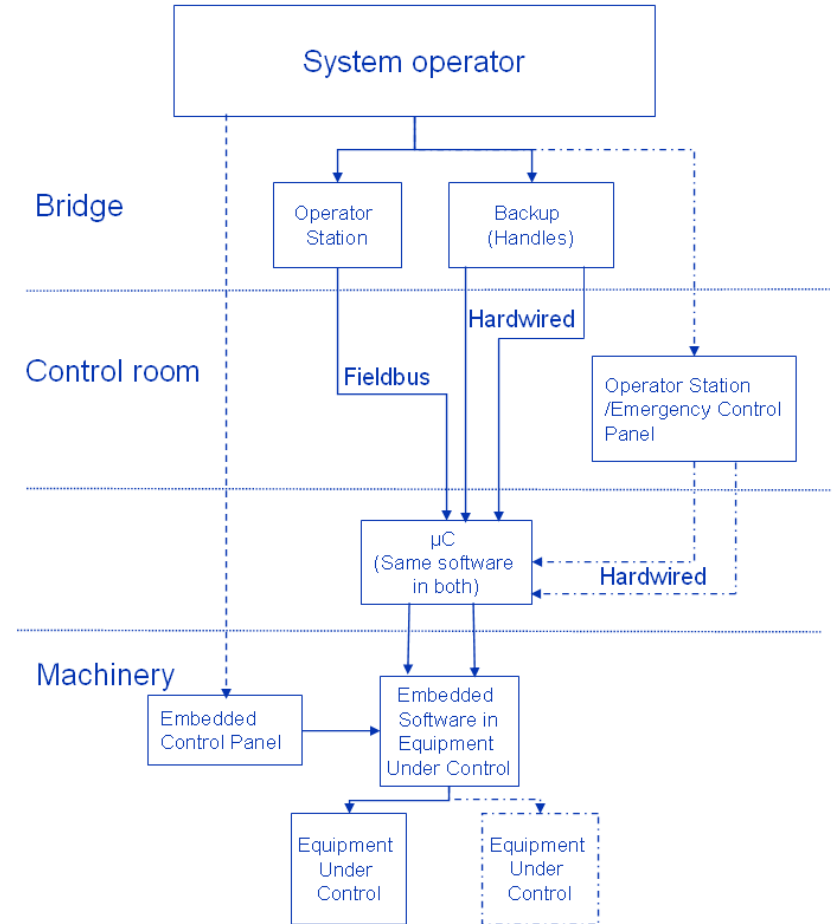


- **Redundancy**, e.g. manual operating facilities, shall be “built in” to the extent necessary for maintaining the safe operation of the vessel. Changeover to systems, designed with redundancy, shall be simple even in cases of failure to control and monitoring systems. Redundancy is defined as two mutually independent systems that can maintain a function
- **Alarm system, automatic control system** and protective **safety system** shall be designed as – at least two – mutually independent systems. The alarm and safety systems must not be designed as one combined system
- Where a computer based system is part of an essential function, a secondary means of operation shall be provided by either non-computer based system or by an independent computer based system of **appropriate diversity**

Reference architecture



Controlling machinery: Hardware redundancy



SW architecture model of the same system

- The yards and suppliers are aware of the Rules with respect to redundancy and independence
- Software redundancy this has not been considered in practice
- The redundant hardware components runs exactly the same software
 - Diverse software is considered too expensive
 - Many suppliers assert that introducing software redundancy does not increase the reliability of the overall system
- It is an ongoing trend to send more information on the same network
 - Information from different systems on the same fieldbus
 - Reduced cabling reduces cost and weight
- Networks are physically redundant, but run the same software, often COTS
- Networks are becoming increasingly important
- Software failure as a common cause failure should be considered in risk analysis

- Compile the source code with two different compilers
 - Obtaining a trust level for a compiler requires 10-15 years
 - Will only reduce the number of failures due to compiler error

- "Backup" software with only minimal functionality for manoeuvrability
 - This software could be made so simple that it could be extensively tested
 - Can be implemented with limited increase in cost

State diversity – is it real?

- "State diversity" between independently running software duplicates sufficient protection against software single failures?
 - More "hope" than reality?
 - Back-up software "hot standby": The state will be the same in both computers most of the time?
 - How much state diversity is necessary?
 - Can software state diversity be measured or proved?

- From the literature:
 - An article by Amman and Knight (IEEE Trans. On Comp., 1988): Protection against hardware failures also protected against a considerable number of software failures
 - Most of the field software faults not identified during testing are so-called "Heisenbugs". States that hardware redundancy give some software fault tolerance for Heisenbugs

- More research is needed

Emergency handling - EH (WP just started)

- Interview with all project partners (6 suppliers, 2 yards)
- Interview 8 ship owners
- Experiences from EH
- How is EH planned in early phases?
- Training?
- Simple mode of operation?
- Is it possible to operate the vessel manually?
- System for how to learn from incidents?
- Improvements proposal for Rules?

Questions?





www.dnv.com
