Software FMECA: A proposed approach applied to BOP control systems

Patrick Rossi
Approval Centre East Asia
DNV GL
Busan, Korea
patrick.rossi@dnvgl.com

Abstract– **Provisions within offshore rules and standards address practically all stages of hardware component life cycles of complex control systems, but relatively little towards software reliability. More guidance is needed on how to approach software alongside hardware in Failure Modes and Criticality Analysis of software dependent control systems for the offshore oil & gas industry. Recently DNV has performed software FMECAs on BOP control systems using techniques that enable software components to be assessed alongside hardware components using a common risk criticality calculation matrix. These FMECAs have shown to be useful in identifying hidden failure modes that would otherwise have been missed in the usual industry practice which often limits the consideration of software by targeting the hardware failure modes of components hosting the software (e.g. CPU) rather than the software components themselves (e.g. Function Block, library, critical software routines within the code, etc.). DNV's approach lead to software design decisions and new test cases being added to the testing program. These additions reduced and mitigated risks of software failure modes thus increasing the robustness of the BOP control systems. This approach involves reverse engineering of topology information, understanding the mechanisms of software failure modes and identifying unforeseen software consequences of hardware failure modes. Although the primary target of this paper are designers of well control systems, it is likely that most other offshore control system engineers will find the approach presented and examples of software failure modes to be useful in their future efforts for improved software dependent system reliability.**

Key Words– Software Reliability, Software Failure Mode and Effects Analysis

Contents

1. INTRODUCTION

As oil resources are increasingly difficult to access, oil companies are driven towards deeper waters, harsher weather down to moving production equipment on the seabed for deep-water unmanned and fully automated oil production facilities. These solutions are increasingly dependent on the extensive use of software.

Safety analysis studies are performed in the early phases of newbuild projects in order to increase the reliability of these critical systems and techniques have been standardized to help the industry maintain risks of system failure to a minimum.

However when it comes to the offshore standards, there is much more focus on hardware than software reliability. Provisions within offshore rules and standards address practically all stages of hardware component life cycles ranging from single component design to the integration and commissioning of complex control systems onboard drilling vessels.

The different focus level on software reliability versus hardware reliability may be in part due to the abstract and intangible nature of software making it more difficult to instinctively identify software as being components that merit just as much attention and rigor as do the hardware components of the offshore industry.

Lately Hyundai Heavy Industry shipbuilding division has also recognized that up until recently software has been somewhat neglected in their usual Newbuild project lifecycle [1].

Another reason for the difference in focus could be attributed to the lack of guidance on software reliability in this industry. This in turn leads to a lack of consideration of software during system safety and reliability analysis studies. Failure Mode and Criticality Analysis (FMECA) techniques have been standardized in IEC 60812 and referenced in offshore standards, yet little guidance can be found on how to treat software components during the analysis. Since hardware failure modes (e.g. bursting of gas pipe) are often mitigated by software functions (e.g. Emergency System Disconnect (ESD)), and software failure modes (e.g. fail to close valve command) are often mitigated by hardware components (e.g. redundant controller, or local manual operation), one could expect that both hardware and software system component failures would be assessed during FMECAs, however this is not a common practice in this industry.

Recently Det Norske Veritas (DNV) has performed software FMECAs on Blow Out Preventer (BOP) control systems using techniques that enable software components to be assessed alongside hardware components using a common risk calculation matrix.

These FMECAs have shown to be useful in identifying hidden failure modes that would otherwise have been missed in the usual industry practice. DNV's approach lead to software design decisions and extra system test cases that were added to the testing program. These additions reduced and mitigated risks of software failure modes thus increasing the robustness of the BOP control systems.

This paper provides guidance on how to approach software alongside hardware in Failure Modes and Criticality Analysis of software dependent control systems for the offshore oil & gas industry by reverse engineering software topology information, understanding the mechanisms of software failure modes and identifying unforeseen software consequences of hardware failure modes. Although the primary target of this paper are designers of well control systems, it is likely that most other offshore control system engineers will find the approach presented and examples of software failure modes to be useful in their future efforts for improved software dependent system reliability.

2. Objective

While compliance to regulatory requirement to perform a System Failure Mode and Effects analysis (FMEA) are often the main driver behind the investments for conducting safety analysis studies, the methodology presented in this paper aims to help improve the reliability of the software dependent system and reduce costs by discovering design faults early in the life cycle phases. The proposed approach for Software FMECA (SFMECA) will ultimately drive both product and process improvements. Product improvements because design changes are part of the possible mitigation actions, and process improvements because a SFMECA is a great tool for enhancing the test programs.

3. Approach

The key elements of the FMECA analysis generally follows the following steps (IEC 60300-3-1:2003 - A.1.7.3):

- identification of how the component of system should perform;
- identification of potential failure modes, effects and causes;
- identification of risk related to failure modes and its effects;
- identification of recommended actions to eliminate or reduce the risk;
- follow-up actions to close out the recommended actions.

In this paper the standard FMECA approach is not presented in its entirety (see IEC 60812). This paper will instead focus on the extension of the usual practice of applying FMECA process to hardware by adding Software Potential Failure Modes (SPFMs) to the analysis. We will start by first addressing key aspects of the preparation work followed by the suggested steps of a SFMECA workshop. Lastly we will touch upon some important considerations to keep in mind during the process and present some examples of the types of failure modes that have been identified applying this approach on BOP control systems.

3.1. Mapping the software to the system scope

In this paper we will assume that the system boundaries have already been determined so the next step would be to prepare the FMECA worksheet. In order to prepopulate the FMECA table with software components and functions, the system design information needs to be assessed:

- What are the functions, especially critical functions?
- Where will the software be executed and data transmitted (Programmable Logic Controllers (PLC), Central Processing Units (CPU), Switches, Input/Output (I/O) modules)?
- What are the different software packages to be deployed?
- How do the software packages communicate with the outside world and with each other (interfaces, protocols)?

These questions lead to the identification of the necessary information that will be needed as inputs to the SFMECA such as functional design specifications, electrical and communication drawings, software architecture and/or topology.

A common problem in the offshore industry is that software design information is not always available or completed during the usual design phase of the newbuild project lifecycle; either the system is novel or the available software developed by the suppliers of control systems lacks traceability to software design information such as a software architecture or topology. The topology should be pre-existing; however as a backup this can easily be reverse engineered by using the existing knowledge of the types of software to be deployed and any available electrical and communications drawings. For example let's imagine that the diagram in Figure 1 is actually an electrical/communication drawing representing the scope of the distributed PLC & Computer based BOP control system (sensors, grounding, power, pushbuttons and x-interfaces, etc. have been left-out for simplicity).

(Step 1) The first thing we would need to do is identify all CPUs, PLCs, Switches & I/O modules on which the software will be running (Windows, Siemens Step-7, PLC application software & Firmware; etc.). The idea here is to identify all the CPUs and microprocessors within the electrical and communications drawings. If there is a CPU on these drawings, chances are that there will be software running on it. Figure 1 shows an example of tagged CPUs, PLCs, I/O Modules & Switches on a given drawing:

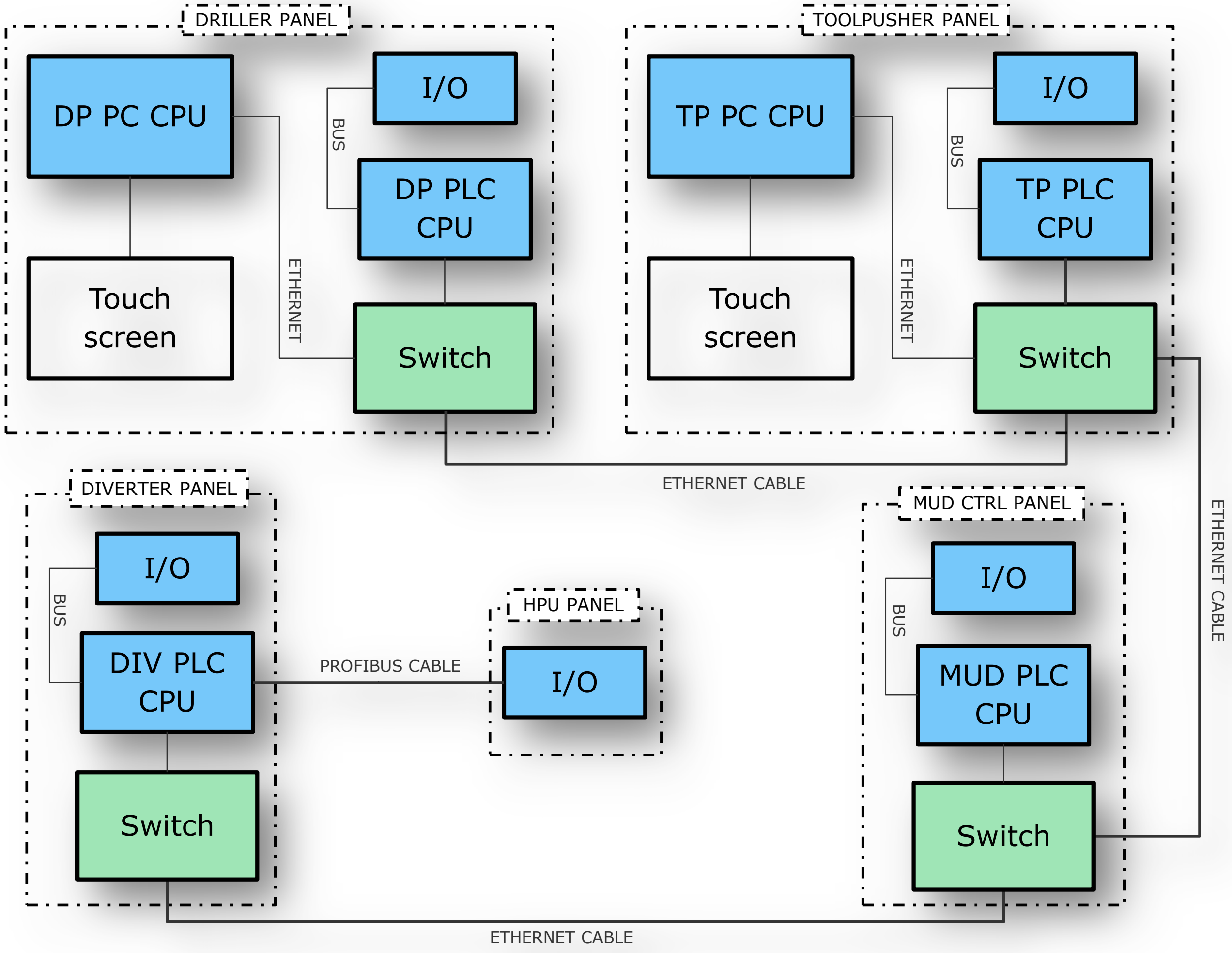


Figure 1: Example of drawing on which CPUs, I/O modules and Switches are identified

(Step 2) Once we have identified all the hardware on which software will be running and/or transmitting, we will need to list all software packages, firmware and operating systems to be deployed on this hardware. For example we might be aiming to deploy Windows, Siemens Step-7, Original Equipment Manufacturer (OEM) firmware, the PLC application software, (e.g. BOP stack control, HPU control, etc.). (Step 3) The next step is to retrace the communication links and protocols and build the software topology by mapping the identified software in (Step 2) to the hardware identified in (Step 1) and sequentially numbering the Software Components (SC) in the resulting topology (sc.#..). (Step 4) Lastly we shall ensure that we have accounted for all the known software functions by allocating them to the software topology created in (step 3) such as the example shown in Figure 2 :

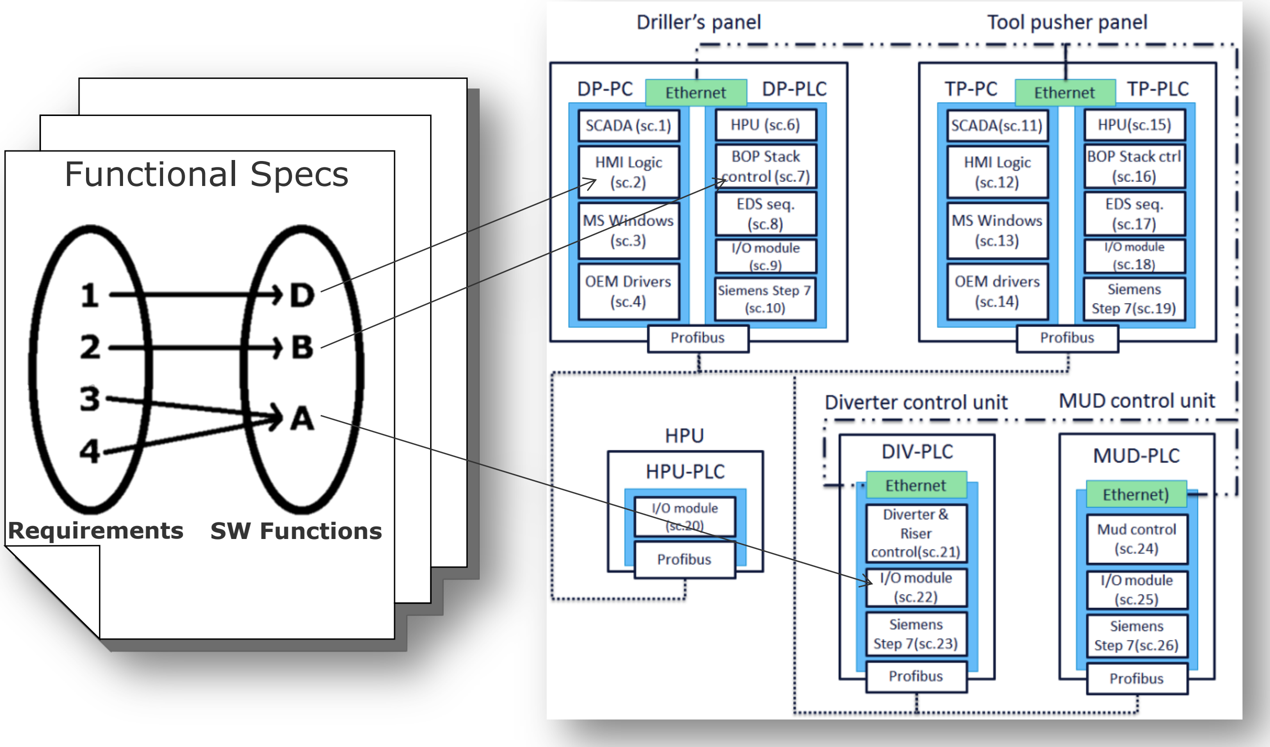
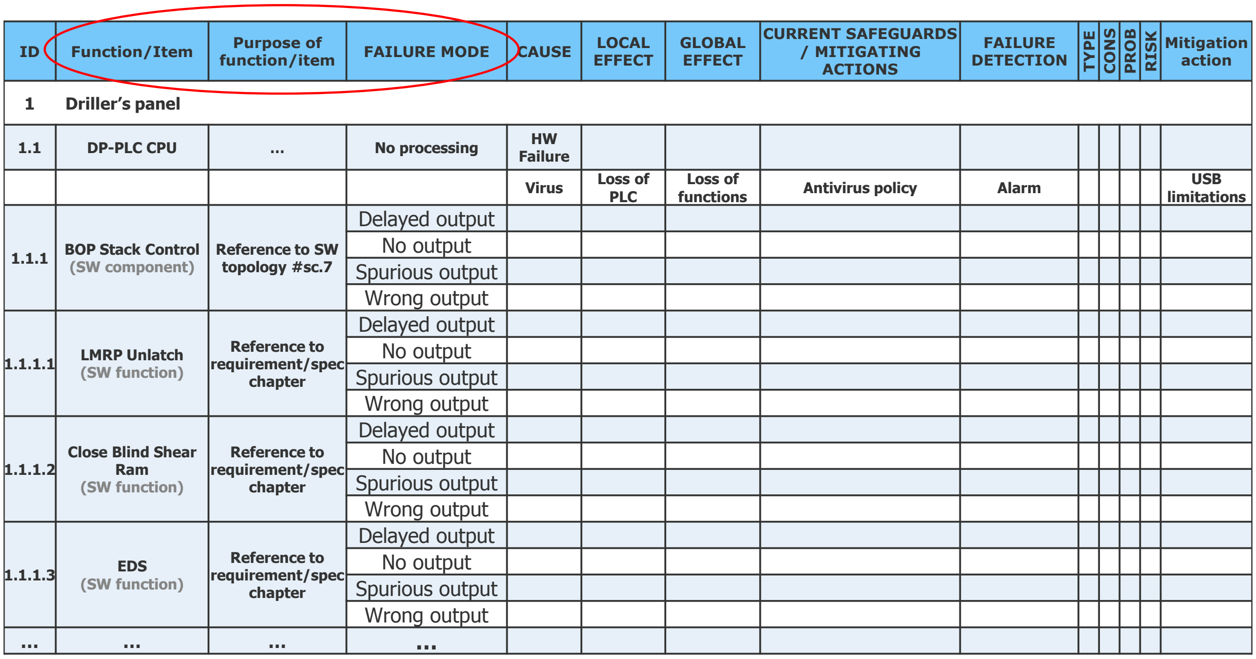


Figure 2: Example of software topology (right) reversed engineered from electrical and communications drawings

An effective way to ensure that all of the software functions have been accounted for is to use a requirements traceability matrix that traces all the requirements through the design information and down to the test cases; adding a column referencing the software components identified on the software topology (sc.#___) helps to then sort the traceability matrix table in order to list all software functions within a given component. If no requirements traceability matrix is available then the facilitator should mark-off each section of the design documentation describing software functions and reference which software component hosts the given function within the design documentation.

3.2. Prepopulating the FMECA table

An another important step of the preparation process will be to address the appropriate level of granularity; that is the lowest functional level on which failure mode analysis will later be performed. Too high of a level will not produce much more added value than a pure hardware FMECA, and too low of a level of detail will swamp the team with tedious repetitive work that will have an adverse effect on the team's participation. That said the level of granularity will depend on the level of available information; as a rule of thumb we will use the lowest level of functions described in the functional design specifications such as valve commands, interlocks, Emergency Disconnect System (EDS) sequences, Human Machine Interface (HMI) functions, watchdogs, etc. Table 1 shows an example of a prepopulated SFMECA table and details the level of granularity up to the failure modes:

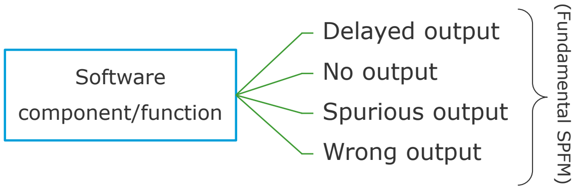


ID	Function/Item	Purpose of function/item	FAILURE MODE	CAUSE	LOCAL EFFECT	GLOBAL EFFECT	CURRENT SAFEGUARDS / MITIGATING ACTIONS	FAILURE DETECTION	TYPE	CONS	PROB	RISK	Mitigation action
1	Driller's panel												
1.1	DP-PLC CPU	...	No processing	HW Failure									
				Virus	Loss of PLC	Loss of functions	Antivirus policy	Alarm					USB limitations
1.1.1	BOP Stack Control (SW component)	Reference to SW topology #sc.7	Delayed output										
			No output										
			Spurious output										
			Wrong output										
1.1.1.1	LMRP Unlatch (SW function)	Reference to requirement/spec chapter	Delayed output										
			No output										
			Spurious output										
			Wrong output										
1.1.1.2	Close Blind Shear Ram (SW function)	Reference to requirement/spec chapter	Delayed output										
			No output										
			Spurious output										
			Wrong output										
1.1.1.3	EDS (SW function)	Reference to requirement/spec chapter	Delayed output										
			No output										
			Spurious output										
			Wrong output										
...	...	...	...										

Table 1: Example of a prepopulated SFMECA table

3.3. Narrowing down the Fundamental SPFMs

Prepopulating failure modes with a narrowed down number of types of failure modes will greatly reduce the vast realm of SPFMs and will save you time consuming discussions during the workshop. The need for narrowing down the potential failure modes has also been discussed in standards and literature [2, 3, 4, 5]. In this approach we propose to narrow down the SPFMs to four (4) fundamental types of software failures:



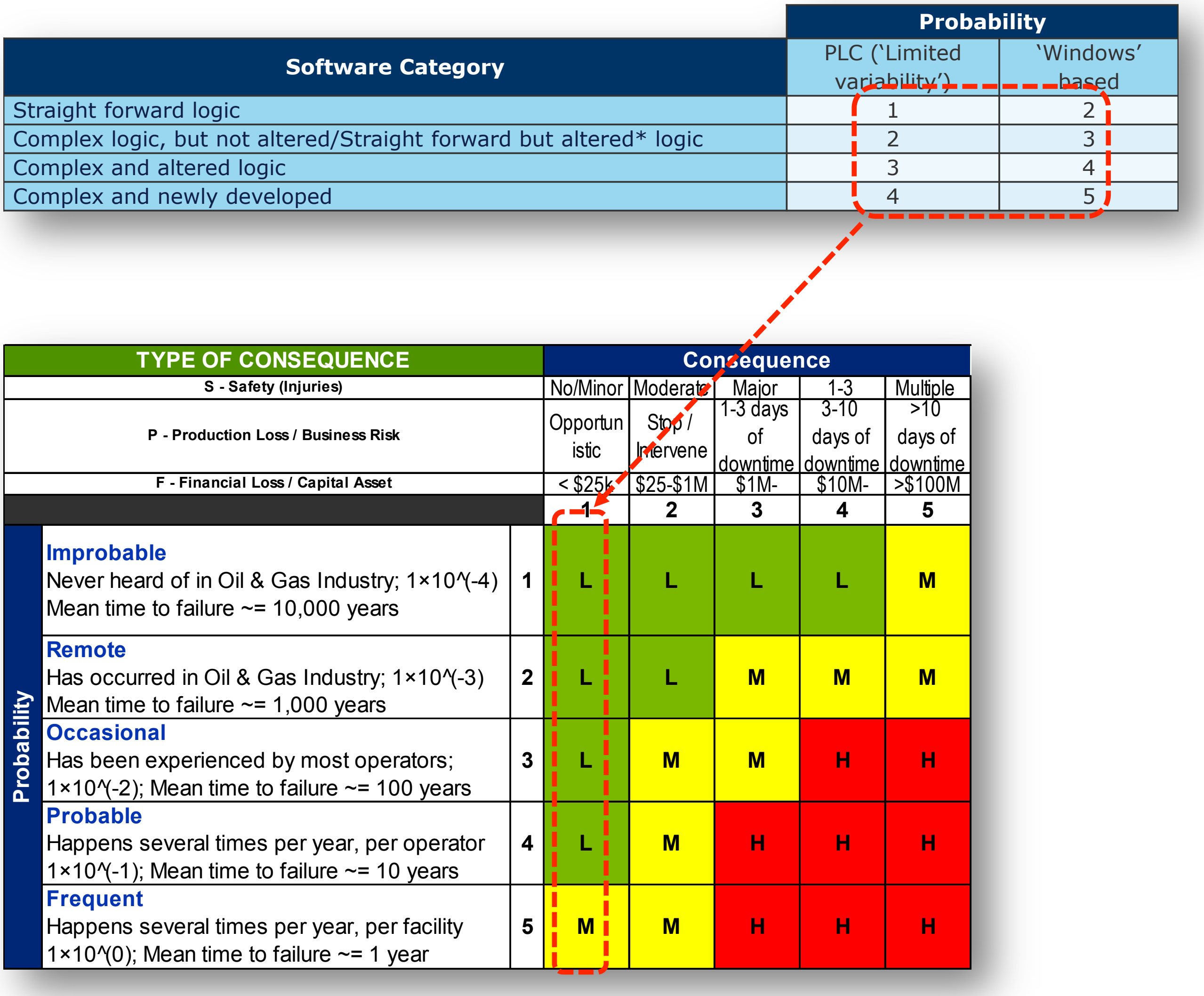
During the workshop we will eliminate SPFMs depending on their relevance and the function's criticality. Also the facilitator should pay attention to team's reaction; if the SPFM does not trigger interest then it can simply be deleted, otherwise when the team is in doubt or demonstrates interest then it should be kept in.

3.4. Mapping the software probability of failure to a common HW & SW risk/criticality matrix

Software by itself is harmless; it is only when uploaded to microcontrollers and other CPUs controlling physical elements that their failures can lead to consequences in the physical world. This means that ultimately the consequences of software failures will lead to the same as for the hardware failures. Therefore the difficulty of using a common risk/criticality calculation matrix is not in determining the consequence categories but rather in mapping the scale of probabilities of failure. For hardware we usually dispose of manufacturer data and field experience but for software the problem is different. This approach proposes a simple and easy to use method of mapping the software probability of failure scale to the hardware scale that we will assume has already been determined for the usual hardware FMECA workshop. To do this we propose three criteria joined by a fourth overriding one:

- Technology robustness: for example Windows based technology is known to have a higher failure rate than PLC based technology…
- Logic complexity: the correlation between the logic complexity and defect-prone software has been validated in the literature [6, 7, 8]; when assessing a SPFM for its probability of failure, invite the software engineers to categorise the software function's complexity (e.g.: straight forward logic, medium or complex).
- Proven in use: when a software module or function has been proven in use and is not subject to any modification for the target system under analysis, the probability of failure can be lowered as the software defects are more likely to have been identified and removed throughout the years of its use in the industry. On the other hand when a known software function is to be tailored or modified, then new software defects may be introduced during its modification and the probability of failure increases in turn. For this reason, any alteration to reused software should disqualify it as being "proven in use", in other words altered software should not be considered as proven in use.
- SMEs' field experience: lastly the probability of failure can be overridden by the SMEs depending on the team's discussions as they may have field experience on known failure modes of certain software packages etc.

In the example below we have used the above criteria to create a scale of 1 to 5 which maps to the scale of hardware failure probabilities:



Software Category	Probability	
	PLC ('Limited variability')	'Windows' based
Straight forward logic	1	2
Complex logic, but not altered/Straight forward but altered* logic	2	3
Complex and altered logic	3	4
Complex and newly developed	4	5

TYPE OF CONSEQUENCE			Consequence				
S - Safety (Injuries)			No/Minor	Moderate	Major	1-3	Multiple
P - Production Loss / Business Risk			Opportunistic	Stop / Intervene	1-3 days of downtime	3-10 days of downtime	>10 days of downtime
F - Financial Loss / Capital Asset			< $25k	$25-$1M	$1M-	$10M-	>$100M
			1	2	3	4	5
Probability	**Improbable** Never heard of in Oil & Gas Industry; 1×10^(-4) Mean time to failure ~= 10,000 years	1	L	L	L	L	M
	Remote Has occurred in Oil & Gas Industry; 1×10^(-3) Mean time to failure ~= 1,000 years	2	L	L	M	M	M
	Occasional Has been experienced by most operators; 1×10^(-2); Mean time to failure ~= 100 years	3	L	M	M	H	H
	Probable Happens several times per year, per operator 1×10^(-1); Mean time to failure ~= 10 years	4	L	M	H	H	H
	Frequent Happens several times per year, per facility 1×10^(0); Mean time to failure ~= 1 year	5	M	M	H	H	H

Figure 3: Mapping the software probability of failure scale alongside the hardware provides a common risk/criticality calculation matrix

3.5. Mutual effects of SW/HW failures

It is worthwhile to assess the mutual effects of software and hardware failure modes. For example software has been known to escalate failure consequences in the past; a mechanical failure of a limit switch lead to severe damage of the equipment since the software misinterpreted the actual position of the equipment during critical manoeuvres. Designers of control systems should consider:

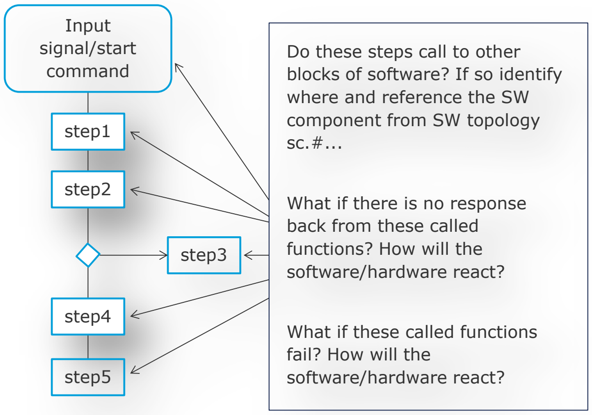
- How will the equipment react to a given software failure mode?
- How will the software react faced with a hardware failure, loss of sensor, erroneous sensor, sensor missing from design, or faulty position of the equipment?
- Are there common mode failures being introduced by hardware redundancy (redundant hardware running same software…)?

These questions are useful in that they will bring value and help elaborate on the failure consequences during the breaking down of critical function as seen in Figure 4.

3.6. Breaking down of critical functions

On top of identifying and ranking SPFMs, getting the SMEs to agree on a list of critical software functions that require special attention will also help capture interesting failure modes and mitigation actions. As it is not practical to test all theoretical branches of software execution paths which run up to 2^n paths where n is the number of conditions within a function, and if this function contains conditional loops then the result may lead to an infinite number of paths [6]. Inviting the team to also pre-select a list of critical functions will help cover more ground at a minor cost and will help keep the teams' attention fresh throughout the workshop. The list of critical functions could look like EDS, Safety critical functions, Regulatory sensitive functions, etc.

Zooming-in and breaking down these critical functions will help elaborate on the failure consequences and identify mitigation actions. One easy and intuitive way to do this is to draft the flow chart of a function's process flow and analyze what would be the cascading consequences of a function failing at a given step while considering software interlocks, timers, calls to libraries, detection of faulty sensors, etc. Figure 4 shows an example of this:



Do these steps call to other blocks of software? If so identify where and reference the SW component from SW topology sc.#…

What if there is no response back from these called functions? How will the software/hardware react?

What if these called functions fail? How will the software/hardware react?

Figure 4: Breaking down a critical function to elaborate on the failure consequences

3.7. Analyzing HMI failure modes

When coming across mimics or screenshots of HMIs within the design documents, existing or to be developed, the facilitator should ask the SMEs if there is logic embedded within the HMI or Supervisory Control And Data Acquisition (SCADA) that will be executed for a given command/push button or if it is simply requesting the execution of some program code to be executed in a remote PLC for example.

This probing will help bring forward implicit software functions (and SPFMs) executed within the HMI/SCADA software that may otherwise get overlooked. It is worth highlighting even if an HMI is simply waiting for a return from another code execution before completing a sequence or giving back the command to the operator; as we will show later in the examples, crashing of the HMI at the wrong time can also lead to serious undesired effects.

3.8. Determination and tracking of mitigation actions

Risk mitigation actions aim to reduce the consequences and probability of software defects from surfacing during operations. The implementation of the mitigation actions are typically confirmed during software verification and validation activities: design peer reviews, software code peer reviews, unit testing, Factory Acceptance Testing (FAT), Hardware In the Loop (HIL) testing.

One temptation that must be avoided during the workshop is to assume the mitigation actions to be already in place. Members of the workshop team may suggest lowering the probability of failure of a SPFM because they are assuming the suggested SPFM will already have been discovered by the time the system is deployed, because they are basing their assumption on the fact that the software is scheduled for testing before it leaves the factory. However the probability of failure should not be lowered on this basis, rather it should be kept as initially determined, and if the resulting level of risk requires action, then the action to include this failure mode in the test program will be recorded and tracked with the help of QA (rather than assuming the future test program will address the SPFM in question).

Mitigation actions must therefore be considered as actions to be carried out and tracked via documented evidence confirming that the mitigating verification & validation activities have been carried-out. Teaming-up with the software quality assurance responsible will help keep track of these actions. This is especially important for new software as the likelihood of failure of new software is at its maximum until all verification and validation actions have been carried-out, including testing of failure modes.

4. SPFMs found during BOP SFMECAs

DNV has so far tested the proposed approach for BOP control system FMECAs on three different occasions. The approach has proven to be effective in revealing SPFMs that would have otherwise been overlooked during the normal system FMECA. DNV was able to identify a combined amount of 224 failure modes, 11 of which were High risks, 22 were Medium and 191 were Low. The following Table 2 provides a sample set of the types of software failure modes and mutual effects from hardware failures that have been identified using this approach:

FUNCTION/ITEM	FAILURE MODE	CAUSE	GLOBAL EFFECT	MITIGATION ACTION
Fluid recovery	Unexpected malfunction	Erroneous SW configuration	Stack malfunction	Display readback values to help identify wrong configuration Include default setpoints in commissioning procedure
HMI unlatch function	No output	SW defect	manual unlatch impossible	Implement time-out logic in interlock
SCADA	No output	SW defect	Unable to control annular pressure leading to collapse of casing	Implement high pressure shutdown logic in PLC code, addition of this failure mode in HIL testing program
EDS	Spurious output	SW defect	Blowout	Extensive testing of new logic, confirm achievement of required SIL level (see IEC 61508)
I/O module	No output	HW failure	Gas, mud, wellbore fluid discharge in environment	Software implementation of failure handling logic and alarm signaling loss of I/O module
Intrinsically safe barriers	No output	HW failure	Damage to pump	Software implementation of alarm signaling loss of barrier
SW valve command	Unable to complete command	Mechanical problem	Failure to unlatch, Blowout, Discharge into sea	Additional logic enabling SW interlock to be aware of valve position at end of interlock sequence, SW Functional spec to be updated.

Table 2: Sample of software failure mode identified using the proposed SFMECA approach

The sample set given in Table 2 are examples of the most significant risks that could later be mitigated at a minor cost. The first half of the table contains examples of software failure modes while the second half contains hardware failure modes for which consequences were found to be aggravated by the software not being designed to handle a given hardware failure. Mitigation actions are shown to vary from simple additions of time-out logic in the software code to the implementation of software watchdogs and additions of more situational awareness logic also to be described in the functional design specifications. In any case the failure modes should be added to the testing program(s) i.e. Unit, FAT & HIL testing where applicable.

5. Discussion

The subjectivity of the approach discussed in chapter 3.4 can be debated. One could see the probability of failure scale of the hardware aspect in Figure 3 as having a more mathematical approach where failure categories range from 1 X 10$^{(-4)}$ to 1 X 10$^{(0)}$. However the determination of these categories is also very subjective as they are deeply dependent on the experience of the FMEA team. Furthermore, the calculated risks are in practice subject to arbitration since the team can also override the result when it agrees to do so; for example if the team is on the fence between choosing a failure category, they can decide to tip the scale to the highest level to ensure that the risk will mandate a mitigation action if the team feels the need to do so.

One limitation that has been observed is relation to the maturity level of the software design information. At the time of a SFMECA analysis, a too low level of maturity could potentially be in conflict with certain offshore rules. For example NORSOK Z-013 states that the basis for a risk analysis must be documented, which would render the output of a premature system FMEA to be non-compliant. However the proposed methodology in this paper can be applied even when software design information is limited or available only in part. Therefore despite the often low level of software design information or maturity in the design phase of offshore drilling control systems, the SFMECA will still drive the quality and reliability of the system. A SFMECA update can always be performed on the system once the software design has been significantly updated.

6. Conclusion

As pointed-out in IEC 60812; a FMECA should not be used as the single basis for judging whether or not the risk of a system is acceptably small. "[…] *more influential parameters (and their interactions) can be taken into account, e.g. exposure time, probability of avoidance, latency of failures, fault detection mechanisms*". However this approach has been observed to add value to the conventional methodology observed in the offshore industry where system FMECAs often only address software failures via the failures modes of hardware hosting the software.

Despite having been observed on a small sample set, this approach is based on proven concepts and has already showed promise in the few attempts performed on BOP control systems as it helped shed light on otherwise overlooked SPFMs. In three separate trials of applying this approach DNV was able to identify a combined amount of 224 failure modes at a low cost (one to two extra days for the SFMECA were added to the hardware FMECA which typically ran between 7 to 10 days). The identification of these failure modes lead to mitigation actions that improved the hardware design and also enhance the verification and validation strategy. Since these SPFMs and their test cases were identified early in the system life cycle, the findings could be addressed with the most cost effectiveness.

7. Acknowledgements

This paper is supported by DNV GL Approval Center Korea. I also acknowledge the encouragement and assistance provided by my colleague and mentor, David N. Card, ISDS Senior Principal Specialist. Mr. Card is listed in Who's Who in Science and Technology and has received several awards for his professional activities from CSC, NSF, NASA, IEEE, and AIAA.

8. Definitions

8.1. Terms

Commercial Off-The-Shelf (COTS): COTS products are ready-made packages sold off-the-shelf to the acquirer who had no influence on its features and other qualities. Typically the software is sold pre-wrapped with its user documentation [ISO/IEC 25051:2006(E)].

Critical: Any function or component whose failure could interfere significantly with the operation or activity under consideration [DNV-OS-D203].

Defect: Non-fulfilment of a requirement related to an intended or specified use [ISO 9000: 2005].

Factory Acceptance Tests (FAT): Acceptance testing (see above) of a component, sub-system or system before delivery and integration.

Failure: The termination of the ability of a functional unit to perform a required function on demand [IEC 60812:2006]

Failure mode: A defined manner in which a failure can occur [IEC 60812:2006]. Failure modes can be seen as scenarios for how a system can go wrong.

Fault: Abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function excluding the inability during preventive maintenance or other planned actions, or due to lack of external resources [IEC 61508-4].

Firmware: The combination of a hardware device and computer instructions and data that reside as read-only software on that device [IEEE 610.12:1990].

Functional requirement: A requirement that specifies a function that a system or system component must be able to perform [IEEE 610.12:1990].

Hardware In the Loop (HIL), a testing technique using simulators to simulate external conditions.

Peer review: A process of subjecting an author's work to the scrutiny of others who are experts in the same field.

Proven in use: The proven-in-use analysis should investigate failure data from previous systems that have been operated in a controlled way, e.g., all errors and software changes must have been recorded [DNV-OS-D203]

Redundancy: The existence of more than one means for performing a required function or for representing information [IEC 61508-4]. Redundancy prevents the entire system from failing when one component fails.

Requirement: A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents [IEEE 610.12:1990].

Reused software: Software integrated into the system that is not developed during the project, i.e., both standard software and non-standard legacy software. Software can be reused "as-is" or be configured or modified [DNV-OS-D203].

Review: Activity undertaken to determine the suitability, adequacy and effectiveness of the subject matter to achieve established objectives [ISO 9000:2005].

Risk: The qualitative or quantitative likelihood of an accident or unplanned event occurring, considered in conjunction with the potential consequences of such a failure. In quantitative terms, risk is the quantified probability of a defined failure mode times its quantified consequence [DNV-OSS-300].

Safety integrity level (SIL): A relative level of risk-reduction provided by a safety function, or to specify a target level of risk reduction [IEC 61508].

Software: Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system [IEEE 610.12:1990].

Software Component: A software component is an interacting set of software modules [DNV-OS-D203].

Software Module: Separately compliable or executable piece of source code. It is also called "Software Unit" or "Software Package" [ISO/IEC 12207:2008]. A small self-contained program which carries out a clearly defined task and is intended to operate within a larger program [DNV-OS-D203].

Specification: A document that specifies, in a complete, precise, verifiable manner, the requirements, design, behaviour, or other characteristics of a system or component, and, often, the procedures for determining whether these provisions have been satisfied [IEEE 610.12:1990].

System Design Review: A review conducted to evaluate the manner in which the requirements for a system have been allocated to configuration items, the system engineering process that produced the allocation, the engineering planning for the next phase of the effort, manufacturing considerations, and the planning for production engineering [IEEE 610.12:1990].

Traceability: Linkage between requirements and subsequent work products, e.g. design documentation and test documentation [DNV-OS-D203.]

Traceability matrix: A matrix that records the relationship between two or more products of the development process; for example, a matrix that records the relationship between the requirements and the design of a given software component [IEEE 610.12:1990].

Validation: Confirmation, through the provision of objective evidence that the requirements for a specific intended use or application have been fulfilled [ISO 9000:2005].

Verification: Tasks, actions and activities performed to evaluate progress and effectiveness of the evolving system solutions (people, products and process) and to measure compliance with requirements. Analysis (including simulation, demonstration, test and inspection) are verification approaches used to evaluate: risk; people, product and process capabilities; compliance with requirements, and proof of concept [INCOSE SE 2004].

Verification strategy: Identification (e.g., list) of verification activities to be performed, along with verification methods, objectives, and responsibility assigned to them. The purpose of this strategy is to minimize redundancy and maximize effectiveness of the various verification activities [DNV-OS-D203].

8.2.Abbreviations

BOP: Blow Out Preventer

CPU: Central Processing Unit

DNV: Det Norkse Veritas (Classification society now merged with GL)

EDS: Emergency Disconnect System

FMEA: Failure Mode and Effects analysis

FMECA: Failure Modes, Effects and Criticality Analysis

HMI: Human Machine Interface

OEM: Original Equipment Manufacturer

PLC: Programmable Logic Controller

PFM: Potential Failure Mode

SCADA: Supervisory Control And Data Acquisition

SFMECA: Software FMECA

SPFM: Software Potential Failure Mode

9. References

[1] *Software Management for Integrated control system based on ISDS notation*, Drillship & Semi-Rig & Jackup Rig, session 5, Offshore Korea conference, November 13th 2014

[2] *Integrated Software Dependent System*, Offshore Class notation DNV-OS-D203, B.RAMS.3

[3] Ristord, L. & Esmenjaud, C., 2001, *FMEA Per-ored on the SPINLINE3 Operational System Software as part of the TIHANGE 1 NIS Refurbishment Safety Case*. CNRA/CNSI Workshop 2001–Licensing and Operating Experience of Computer Based I&C Systems. Ceské Budejovice–September 25–27, 2001.

[4] Lutz, R.R. & Shaw, H–Y., 1999a, *Applying Adaptive Safety Analysis Techniques*. Proceedings of the Tenth International Symposium on Software Reliability Engineering, Boca Raton, FL, Nov 1–4, 1999.

[5] James R. Kotterman, *Narrow(er) FOCUS, Tapping into a powerful new method to better identify potential failure modes*. Quality Progress Journal -p.23-28, January issue 2015

[6] McCabe, *Structured Testing: A Software Testing Methodology Using The Cyclomatic Complexity Metric*, National Bureau of Standards Special Publication 500-99.

[7] T.J. McCabe, "*A Complexity Measure*" IEEE Transactions on Software Engineering. Vol. SE-2, no. 4. Dec. 1976. Pp. 308-320

[8] T.J. McCabe, and associates, Inc. *Structured Testing Workbook*, 14th Edition

Biography

Patrick Rossi
Senior ISDS Approval Engineer
Integrated Software Dependent Systems
DNV GL, Approval Centre Korea
patrick.rossi@dnvgl.com,
patrick_rossi@hotmail.com

Patrick graduated with a Masters degree in Automated Production Engineering in 2000. He held different posts in various projects ranging from manufacturing quality assurance to software quality assurance in the aerospace, automation, maritime and offshore industries in Europe until 2013 when he joined the South Korean approval centre as an Integrated Software Dependent System (ISDS) senior approval engineer for offshore newbuild projects. Patrick is lead surveyor for ISDS projects for high tech drilling vessels destined for the North Sea and is a contributor for the ISDS class notation targeted at increasing software dependent system reliability in the offshore drilling industry.