NKS-304
ISBN 978-87-7893-381-2

Software reliability analysis for PSA

Ola Bäckström[1]

Jan-Erik Holmberg[2]

Mariana Jockenhövel-Barttfeld[3]

Markus Porthin[4]

Andre Taurines[3]

[1]Lloyd's Register AB, Sweden

[2]Risk Pilot AB, Sweden

[3]AREVA GmbH, Germany

[4]VTT Technical Research Centre of Finland

March 2014

Abstract

A project is ongoing, financed by Nordic nuclear safety research (NKS), The Finnish Research Programme on Nuclear Power Plant Safety (SAFIR2014) and Nordic PSA group (NPSAG), with the intent to provide guidelines to analyse and model digital systems in probabilistic safety assessment (PSA), using traditional reliability analysis methods (FMEA, Fault tree analysis).
This report discusses software reliability in this context. The report proposes a method for the evaluation and quantification of reactor protection system (RPS) software failures. The proposed method will use operational history to estimate the fatal failure probability within system software (operating system, runtime), and use an indirect method for the estimation of failure probability within application software (non-fatal and fatal failures). The quantification for application software is based on two main measures, complexity and the degree of verification and validation of the software.
Collection of data and its challenges will also be discussed. Some data collected for a software platform will be discussed, and used as an example of the difficultness — and challenge — to collect data.

Key words

PSA, Software reliability, Operational history data

NKS-304
ISBN 978-87-7893-381-2

Electronic report, March 2014
NKS Secretariat
P.O. Box 49
DK - 4000 Roskilde, Denmark
Phone +45 4677 4041
www.nks.org
e-mail nks@nks.org

NKS Report NKS-304

Software reliability analysis for PSA

Ola Bäckström[1]
Jan-Erik Holmberg[2]
Mariana Jockenhövel-Barttfeld[3]
Markus Porthin[4]
Andre Taurines[3]

[1]Lloyds Register AB, P.O. Box 1288, SE-172 25 Sundbyberg, Sweden
[2]Risk Pilot, Parmmätargatan 7, SE-11224 Stockholm, Sweden
[3]AREVA GmbH, P.O. Box 1109, 91001 Erlangen, Germany
[4]VTT Technical Research Centre of Finland, P.O. Box 1000, FI-02044 VTT, Finland

March 2014

Table of contents

Tables

Figures

Abbreviations

APU	Acquisition and processing unit
AS	Application software
BBN	Bayesian Belief Net
CCF	Common cause failure
CDF	Core damage frequency
DCU	Data communication unit
DCS	Data communication software
DLC	Data link configuration
EF	Elementary function
EFW	Emergency feedwater system
FRS	Functional requirement specification
HW	Hardware
I&C	Instrumentation and control
IEC	International Electrotechnical Commission
NKS	Nordic nuclear safety research
NPP	Nuclear power plant
NPSAG	Nordic PSA Group
OECD/NEA	Organisation for Economic Co-operation and Development, Nuclear Energy Agency
pfd	Probability of failure on demand
PSA	Probabilistic safety assessment
RPS	Reactor protection system
SAFIR	Finnish Research Programme on Nuclear Power Plant Safety
SIL	Safety Integrity Level (as defined in IEC 61508)
SIVAT	Simulation-based validation tool of TXS
SPACE	Specification and coding environment of TXS
SS	Subsystem
SyS	System software
SW	Software
TXS	TELEPERM® XS, product of AREVA
V&V	Verification and validation
VTT	Technical Research Centre of Finland
VU	Voting unit

Summary

Digital protection and control systems have been in operation for many years now (e.g. in France, Japan) and are appearing as upgrades in older nuclear power plants (NPPs) with increasing operating experience and are standard solutions for new NPPs. To assess the risk of NPP operation and to determine the risk impact of digital system upgrades on NPPs, quantifiable reliability models are needed along with data for digital systems that are compatible with existing probabilistic safety assessments (PSAs). Due to the many unique attributes of these systems (e.g., complex dependencies, software), several challenges exist in systems analysis, modelling and in data collection.

In particular, the assessment of software reliability is challenging. Software failures are in general mainly caused by systematic (i.e. design specification or modification) faults, and not by random errors. Software based systems cannot easily be decomposed into independent components, and the interdependence of the components cannot easily be identified and modelled. Applying software reliability models in the PSA context is hence not a trivial matter.

A project is ongoing, financed by Nordic nuclear safety research (NKS), The Finnish Research Programme on Nuclear Power Plant Safety (SAFIR2014) and Nordic PSA group (NPSAG), with the intent to provide guidelines to analyse and model digital systems in a PSA context, using traditional reliability analysis methods (FMEA, Fault tree analysis). The following focus areas have been identified for the activities:

1. Develop a taxonomy of hardware and software failure modes of digital components for common use.
2. Develop guidelines regarding level of detail in system analysis and screening of components, failure modes and dependencies.
3. Develop an approach for modelling and quantification of common cause failure (CCF) between components.
4. Develop an approach for modelling and quantification of software failures.

This report describes the approach for number four above, however it needs to be put in relation to the first three items.

The report will discuss a proposed method for evaluation and quantification of reactor protection system (RPS) software failures in nuclear PSA context. The proposed method will use operational history to estimate the fatal failure probability within system software (operating system, runtime), and use an indirect method for the estimation of failure probability within application software (non-fatal and fatal failures). The quantification for application software is based on two main measures, complexity and the degree of verification and validation of the software.

Collection of data and its challenges will also be discussed. Some data collected for a software platform will be discussed, and used as an example of the difficultness - and challenge - to collect data.

Acknowledgements

The work has been financed by NKS (Nordic nuclear safety research), SAFIR2014 (The Finnish Research Programme on Nuclear Power Plant Safety 2011–2014) and the members of the Nordic PSA Group: Forsmark, Oskarshamn Kraftgrupp, Ringhals AB and Swedish Radiation Safety Authority. NKS conveys its gratitude to all organizations and persons who by means of financial support or contributions in kind have made the work presented in this report possible.

Disclaimer

1 Introduction

Digital instrumentation and control (I&C) is becoming more and more common in nuclear power plants (NPPs). Turbine plant I&C and diverse other safety-related systems, which have minor role in probabilistic safety assessment (PSA) context, are already digital. Although quite a number of plants has received digital reactor protection (RPS) systems either as original equipment (e.g. China, France, Japan,) or in upgrade projects (e.g. Sweden, Switzerland, USA), most plants do not yet have digital reactor protection system. New-builds will have complete digital I&C.

Currently, no common approach is available in the NPP field for assessing safety and reliability of digital I&C and meeting related regulatory requirements. However, there is a tradition to try to find harmonised approaches for probabilistic safety assessment (PSA) and its applications and there is generally a strong interest to find solutions and guidelines on how to deal with digital I&C. Due to the absence of a common method for modelling software CCF in the PSA, generic conservative probabilities are usually used, which tend to be conservative and may ultimately prevent PSA results from providing proper risk insights.

This report presents a method for quantification of RPS software failures in nuclear PSA context. The aim is to define a simple yet sufficient model which describes the software failure impacts and provides a quantification approach for the failures. Treatment of common cause failures (CCF) between components is also discussed.

The work is part of the Nordic DIGREL project [1, 2, 3], and builds partly on the work on taxonomy of failure modes of digital components for the purposes of PSA conducted by the international OECD/NEA Working Group RISK [4].

2 Motivation for quantification of software faults

The modelling of programmable control systems in PSA is often considered as a difficult and tedious task, especially with regard to the software aspect of the systems. The reason that something is difficult to treat probabilistically is not a good argument to omit the representation of it (the software part) in PSA.

Software faults are important to the system and should be part of a safety analysis. The basic question: “What is the probability that a safety system or a function fails when demanded” is fully feasible and well-formed question for all components or systems independently of the technology on which the systems are based [5]. A similar conclusion was made in the workshop on Philosophical Basis for Incorporating Software Failures in a Probabilistic Risk Assessment [6]. As part of the open discussion, the panellists unanimously agreed that:

- software fails
- the occurrence of software failures can be treated probabilistically
- it is meaningful to use software failure rates and probabilities
- software failure rates and probabilities can be included in reliability models of digital systems.

Based on these conclusions, it is necessary to be able to estimate the failure probability of the software.

3 Definition for software faults

3.1 Example safety I&C architecture

DIGREL project primarily considers the RPS of a nuclear power plant, since it is considered more important for PSA than other I&C and it is considered a conceivable target for the activity. There is a general consensus that protection systems shall be included in PSA, while control systems can be treated in a limited manner. The system architecture and the mode of operation of protection systems versus control systems are different, which creates different basis for the reliability analysis and modelling.

Protection systems use microprocessors running in parallel in redundant divisions and they actuate functions on demand (e.g. when process parameter limits are exceeded). Control systems are versatile having both on demand and continuous functions and they do not necessarily have a redundant structure. Different roles of the protection and control systems are also reflected in the safety classification, meaning different safety and reliability requirements.

The differences between different I&C platforms and software may be significant, not only the physical design but also the functional, e.g. fault tolerant features and voting logic. On the other hand, due to the stringent design requirements for protection systems and common functional requirements for safety automation of light water reactors, there are important similarities between design solutions provided by different nuclear safety I&C vendors.

For the purpose of defining concepts and demonstrating modelling and quantification approaches, a generic safety I&C architecture is assumed. The example protection system consists of two diverse subsystems, called RPS-A and RPS-B, both divided into four physically separated divisions (see Figure 1). The platforms of both subsystems are assumed to be identical. The extent of diversity between RPS-A and RPS-B may vary, but we may generally assume that they perform different functions. The number of acquisition and processing units (APU) and voting units (VU) per each subsystem and division may vary, too, but here we assume that there can be more than one APU/VU per each subsystem and division.

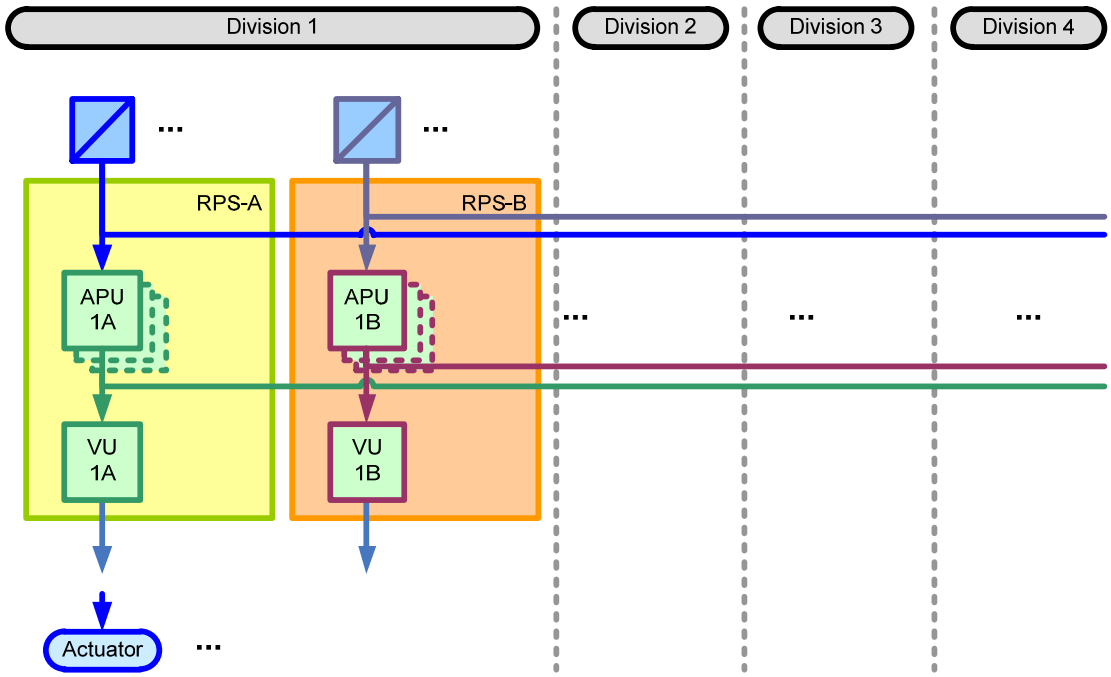

Figure 1. Example I&C system architecture.

3.2 Failure analysis of digital protections systems

With regard to the analysis and modelling of protection systems, the following levels of details can be distinguished [4]:

1. the entire system (subsystems RPS-A and RPS-B in this example)
2. a division (4 divisions in this example)
3. I&C units, i.e., APUs and VUs. In addition there are data communication units (DCU) which are usually integrated with APUs and VUs
4. modules (inside the I&C units)
5. basic components (of which modules are built).

At the system and division level, there are basically two failure modes: “failure to actuate the function” and “spurious actuation”.

At lower levels (I&C unit, module, basic component), it is relevant to consider more aspects of failure modes, *i.e.*:

- The fault location (in which hardware or software module the fault is located)
- Failure effect:
  - Fatal, ordered failure (generation of outputs ceases, outputs are set to specified, supposedly safe values),
  - Fatal, haphazard failure (generation of outputs ceases, outputs are in unpredictable states),
  - Non-fatal, plausible behaviour (generation of outputs continues, an external observer cannot determine whether the I&C unit or the hardware module has failed or not),
  - Non-fatal, non-plausible behaviour (generation of outputs continues, an external observer can decide that the I&C unit or the hardware module has failed).

- Detection situation: On-line detection, off-line detection, revealed only by demand, spurious effect.

Safety I&C relies on a high degree of self-monitoring, self-tests, plausibility checks and engineered monitoring functions, all designed to detect faults, errors and potentially unsafe situations, and to convert them into "safe" states. Therefore, for the further analysis, regarding the main I&C units of the RPS, APU and VU, the relevant failure effects can be restricted into "fatal, ordered failure" and "non-fatal, plausible behaviour", and later called "fatal" respectively "non-fatal" failure in this report.

In a fatal failure the processor stops, the watchdog or exception handler responds to the situation and a "safe state" is activated. Depending on how the "safe state" actuation is defined and the considered system level failure modes in PSA, fatal failures may be ignored (no consequences) i.e. the system shows a passive behaviour and lead to "no actuation signal" (fail-safe configuration, such as TELEPERM® XS) or the safe state is used to initiate a fail-safe actuation. Fatal failure affects all functions implemented in the I&C unit.

Non-fatal, plausible behaviour can cause inappropriate responses (failure to respond on demand, spurious actuation). Good design aims at minimizing the area of impact of non-fatal failures to an as small as possible area (e.g. one function) by separating the code of individual I&C functions, by separating application and system software etc.

Fatal haphazard failure is a practically eliminated failure state for APU and VU. It may be relevant for some other types of modules which are not supervised by watchdogs and have not exception handler. In those cases, fatal haphazard failure can be usually associated with one of the failure modes of the hardware.

Non-fatal, implausible behaviour is similar to fatal, ordered failure from the response point of view.

The combination of fault location, failure effect, detection situation together with the fault tolerant design of the system are usually sufficient to determine the functional end effect, such as

- Loss of all functions (outputs) of the I&C unit (APU/VU),
- Loss of a specific function (no actuation on demand),
- Spurious actuation.

The above list is not exhaustive, and, *e.g.*, for voting logics or in case of intelligent validation of input signals the functional end effect may be more complex (*e.g.* degraded voting logic). Anyway, the module level (both hardware and software) seems to be sufficient to analyse dependencies important to PSA, at least for protection systems.

3.3 Software fault modes

The qualitative part of the software fault mode analysis is focused on

a) identification of safety-critical software modules in I&C units

b) identification of possible effects of postulated faults in the safety-critical software modules

c) identification of defensive measures against the software faults.

The approach is to successively postulate a single software fault in each software module regardless of the likelihood of such faults, and to determine the maximum possible extent of the failure, regardless of the measures taken by design or operation to limit that extent. The following software PSA modules are considered [4]:

- System software (SyS). This includes the operating system and runtime environment (interaction between application and operating system).
- Elementary functions (EFs)[1]. There is one such module per EF. A virtual EF could be created for each hardware module for which one wants to consider failures due to its software and / or hardware design.
- APU functional requirements specification modules (APU-FRS). There is one such module per application function required of an APU. Their purpose is to allow the representation of errors in functional requirements specifications of the acquisition and processing functions.
- APU application software modules (APU-AS). There is one such module per application function implemented by an APU. Their purpose is to allow the representation of errors in the implementation of application-specific acquisition and processing software. If desired, a virtual module may be used to represent postulated errors in the data tables specifying the hardware configuration and the data communication of the APU.
- Proprietary software (Propr. SW) in I&C. There are other modules than the processor module. Specific pieces of software may be present in hardware modules in APU, DCU, VU or any other module of the system (e.g. power supply) other than SyS and AS.
- VU functional requirements specification modules (VU-FRS). There is one such module per voting function required of a VU. Their purpose is to allow the representation of errors in functional requirements specifications of the voting functions.
- VU application software modules (VU-AS). There is one such module per voting function implemented by a VU. Their purpose is to allow the representation of errors in the implementation of application-specific voting software. If desired, a virtual module may be used to represent postulated errors in the data tables specifying the hardware configuration and the data communication of the VU.
- Data communication software (DCS). There could be a dedicated operating system software in the DCSs.
- Data link configuration (DLC). There is one such module per network in the system.

Given the taxonomy of end effects at I&C level, Table 1 summarises the maximum failure effect of a postulated software fault in each of the software PSA modules:

- FF-1SS: Failure of one Function (or more) in one subsystem. This case refers to non-fatal software failures that result in the misbehaviour of one or more I&C functions in one subsystem. The I&C functions that are dependent on the failed

[1] For TELEPERM® XS EF are called function blocks. EF can be considered as part of the system software. However, all the application-specific processing is done in the code of the elementary functions modules. For this reason, EF could be considered as part of the application software.

functions could also fail. Those dependent functions are necessarily in the same subsystem.

- FF-1D-1SS: Failure of one Function (or more) in only one division in one subsystem. This case refers to non-common cause, non-fatal software failures of I&C functions without vote.
- FF-AllSS: Failure of one Function (or more) in all subsystems
- 1APU/1VU: Failure of one set of redundant APUs/VUs. This case refers to fatal software failures affecting only one set of redundant APUs/VUs (necessarily in the same subsystem).
- MAPU-1SS: Failure of multiple sets of redundant APUs in only one subsystem
- 1SS: Loss of one subsystem.
- MAPU-AllSS: Failure of multiple sets of redundant APUs in both subsystems
- 1SS-APU: Loss of one Subsystem and of one or more sets of redundant APUs in the other subsystem.
- SYSTEM: Loss of both subsystems.

Table 1. Effects of software module faults [4].The cells not filled are considered not relevant

	SW fault location									
Effect	**SyS**	**EF (in APU)**	**APU-FRS**	**APU-AS**	**Propr. SW**	**VU-FRS**	**VU-AS**	**EF (in VU)**	**DCS**	**DLC**
FF-1SS	R	R	R	R		R	R	R		
FF-1D-1SS	R	R	R	R						
FF-AllSS	R	R								
1APU	R	R	R	R	R					
1VU	R				R	R	R	R		
MAPU-1SS	R	R			R					
1SS	R	R	R		R	R	R	R	R	R
MAPU-AllSS	R	R			R					
1SS-APU	R	R			R					
SYSTEM	R	R			R	R	R	R	R	

R: Relevant.

3.4 Evaluation of relevant software faults

Table 1 includes a number of possible failure effects for different software faults. Although it would be impractical to take all of them into consideration in the PSA model, the most relevant can be identified. In this report, the software faults and effects proposed in Table 2 will be considered.

Table 2. Screening of relevant software fault cases for PSA modelling.

	SW fault location									
Effect	**SyS**	**EF (in APU)**	**APU-FRS**	**APU-AS**	**Propr. SW**	**VU-FRS**	**VU-AS**	**EF (in VU)**	**DCS**	**DLC**
FF-1SS			4a	4a		4b	4b			
FF-1D-1SS			4c	4c						
FF-allSS										
1APU/1VU			3a	3a		3b	3b			
MAPU-1SS										
1SS	2a	2a	2a		2a	2a	2a	2a	2b	2b
MAPU-AllSS										
1SS-APU										
SYSTEM	1	1			1			1	1	

1. Software fault causing loss of both subsystems (SYSTEM). This is a complete CCF covering all subsystems that have the same SyS. The probability of such an event is naturally extremely low, but the basic event can be used to evaluate the level of hardware diversity in the actuation of safety functions. It is only reasonable to consider a fatal failure consisting in a crash of the processing units, i.e., transition of the computers to a shut-down state. This maximal end effect covers all the other principally possible end effects. Software fault can be located in SyS, EFs, proprietary SW-modules in APUs/VUs, DCS, but it can be represented in a model by a single basic event.

   For this event, a single generic probability needs to be estimated, denoted here P(SYSTEM-SyS fatal CCF).

2. Software fault causing loss of one subsystem (1SS). This is a complete CCF causing a fatal failure which crashes the processing units in one subsystem, i.e.,transition of the computers to a shut-down state. The software fault can be located in

   a) the SyS, EF (APU/VU), APU-FRS, proprietary SW-modules in APUs/VUs, VU-FRS or VU-AS,

   b) DCS or DLC.

   Difference is that in case of fatal failure in DCS or DLC (b), VUs run and can take safe fail states. In case (a), the whole subsystem stops running and also takes a safe state.

   For each case, a generic probability needs to be estimated, denoted here P(1SS-SyS fatal CCF) resp. P(1SS-DCU fatal CCF).

3. Software fault causing failure of redundant set of APUs (3a, see Table 2) or VUs (3b) in one subsystem (1APU, 1VU, respectively). This is a fatal fault causing loss of all functions. The fault can be in APU/VU-FRS or APU/VU-AS.

   There is a variant, where the software fault could cause the failure of multiple sets of APUs in one subsystem (MAPU-1SS). It remains to be analysed case-specifically whether there is a need to consider such CCF.

   For this event, a single generic probability needs to estimated, denoted here as P(AS fatal fault).

4. Software fault causing a failure of one or more application functions. This is a non-fatal failure and can be failure to actuate the function or spurious actuation. The fault can be in the APUs (4a), VUs (4b) or have effect only in one division (4c). For instance, there can be safety functions which are actuated on 2-o-o-4 basis or are not implemented in all divisions. Cases 4a – 4c are modelled by application function and failure mode specific basic events.

The relationship between AS fault and FRS fault can be taken into account in a Bayesian manner, i.e.,

P(AS fault) = P(AS fault | FRS fault)P(FRS fault) + P(AS fault | no FRS fault)P(no FRS fault).

In addition, in order to distinguish between fatal and non-fatal failure, we need to estimate the fraction of AS faults causing fatal respective non-fatal failures.

Table 3 includes a principal decomposition of probability parameters related to faults in AS or FRS. In section 5.3, handling of AS faults is further developed to better match the proposed quantification and modelling approach.

Table 3. Principal probability parameters related to I&C failures caused by application software faults (fault in AS or FRS).

Parameter	**Description**	**Comment**
P(APU-FRS fault) P(VU-FRS fault)	Probability of a fault in FRS. Fault itself does not cause anything, but it increases the likelihood of an AS fault. AS fault can be fatal or non-fatal.	FRS specific value. FRS may be common to more than one AS.
P(APU-AS fault | APU-FRS fault) P(VU-AS fault | VU-FRS fault)	Probability of an AS-fault given FRS-fault. AS-fault causes fatal or non-fatal failure of APU/VU.	FRS fault is not necessarily critical to cause a failure of AS function, i.e., P(AS fault | FRS fault) < 1
P(APU-AS fault | no APU-FRS fault) P(VU-AS fault | no VU-FRS fault)	Probability of an AS-fault given *no* FRS-fault. The AS-fault is caused by the implementation or translation error from FRS to AS. AS-fault causes fatal or non-fatal failure of APU/VU.	Can be assumed to be a generic value
P(APU fatal | APU-AS fault) P(VU fatal | VU-AS fault)	Fraction of fatal failures	Can be assumed to be a generic value
P(APU-AS non-fatal | APU-AS fault) P(VU-AS non-fatal | VU-AS fault)	Fraction of non-fatal failures. Non-fatal failure can cause failure to actuate or spurious actuation	P(AS non-fatal | AS fault) = 1 – P(AS fatal | AS fault)
P(APU-AS no actuation | APU-AS non-fatal) P(VU-AS no actuation | VU-AS non-fatal)	Fraction of non-fatal failures causing failure to actuate.	
P(APU-AS spurious | APU-AS non-fatal) P(VU-AS spurious | VU-AS non-fatal)	Fraction of non-fatal failures causing spurious actuation.	P(APU-AS spurious | APU-AS non-fatal) = 1 – P(APU-AS no actuation | APU-AS non-fatal)

3.5 Software failure modes for an example safety function

As an example, the emergency feedwater system (EFW) pump and the safety core cooling are considered [1]. The failure modes of the pump for this safety function are

- failure to start
- spurious stop.

The above failures may be caused by several reasons, among others failures of the safety I&C, making the link to the fault tree models of RPS. We denote the start signal of EFW-pump by EFW-ON and the stop signal by EFW-OFF.

Assuming similar RPS architecture as in Figure 1, the signal path from the measurements to the pump goes via APUs and a division specific VU. The design principle of RPS is that given the critical input signals from the measurements, 2-o-o-4 is enough to create the actuation signals in APUs (EFW-ON). APUs send the signal to all VUs, which vote again by 2-o-o-4 principle, causing the start signal EFW-ON.

EFW-pumps may also be supervised by a pump leakage protection function. If a leakage is detected in the pump room, the protection system shall stop the pump (only the specific pump). The signal path is the same as for EFW-ON signal, but the measurements are different and the output signal is designated as EFW-OFF. The difference is also that EFW-OFF is a division specific safety function (only the leaking train is stopped).

The following fail-safe principles have been assumed:

- Voting units are assumed to fail to provide EFW-ON and EFW-OFF signals if power supply fails or if there is an internal voting unit failure (i.e. the default value is 0).
- At loss of communication between VU and APU due to a detected failure in the APU, VUs change the voting rule from 2-o-o-4 to 2-o-o-3 in case of a single failure. In this report, CCF due to a systematic SW fault is considered. In this case the preferred state is actuation of EFW-OFF.
- In case of APU safety functions, detected failures of input signals from measurements or from other APUs cause an actuation (i.e. the default value is 1) in a 2-o-o-4 condition.
    - EFW-ON is actuated by 2-o-o-4 (e.g. based on the calculation of the second MIN value of the measurements) low water level condition in the reactor pressure vessel, denoted by the acronym RPV-LL. There are four measurement sensors, one in all four divisions, which information is shared by all divisions.
    - EFW-OFF is actuated by 2-o-o-2 leakage protection signal in each EFW train, denoted by EFW-LEAK-x, x = 1, 2, 3, 4. There are two measurement sensors per each division, and this information is *not* shared between divisions. EFW-OFF stops the affected EFW train in case of pipe break in an EFW train.

Table 4 summarises the application software modules considered in this example.

Table 4. Example application software modules in VUs and APUs.

Unit	Application software	Condition
VU	EFW-ON	2-o-o-4 EFW-ON from APUs 1–4
	EFW-OFF	EFW-OFF from the same division's APU
APU	EFW-ON	2-o-o-4 RPV-LL from APUs 1–4
	EFW-OFF	EFW-LEAK from the same division's APU
	RPV-LL	2-o-o-4 RPV water level below "very low level" measurement from division 1–4 RPV level measurement sensors (can be realised so that each APU calculates the 2nd MIN value and threshold and exchanges the result with each other)
	EFW-LEAK	1-o-o-2 water level in the EFW pump room over the leakage criterion from the same division's leakage detection sensors

A schematic fault tree for the failure to actuate EFW-ON in division x (x = 1, 2, 3 or 4) is shown in Figure 2 and spurious EFW-OFF in Figure 3. The fault tree is developed down to boxes of hardware and software modules failure modes. Software modules failure modes are listed in Table 5. Hardware modules failure modes are omitted since they are out of the scope of the report. Only one redundancy (division 4) is developed at the APU level. The other divisions are identical.

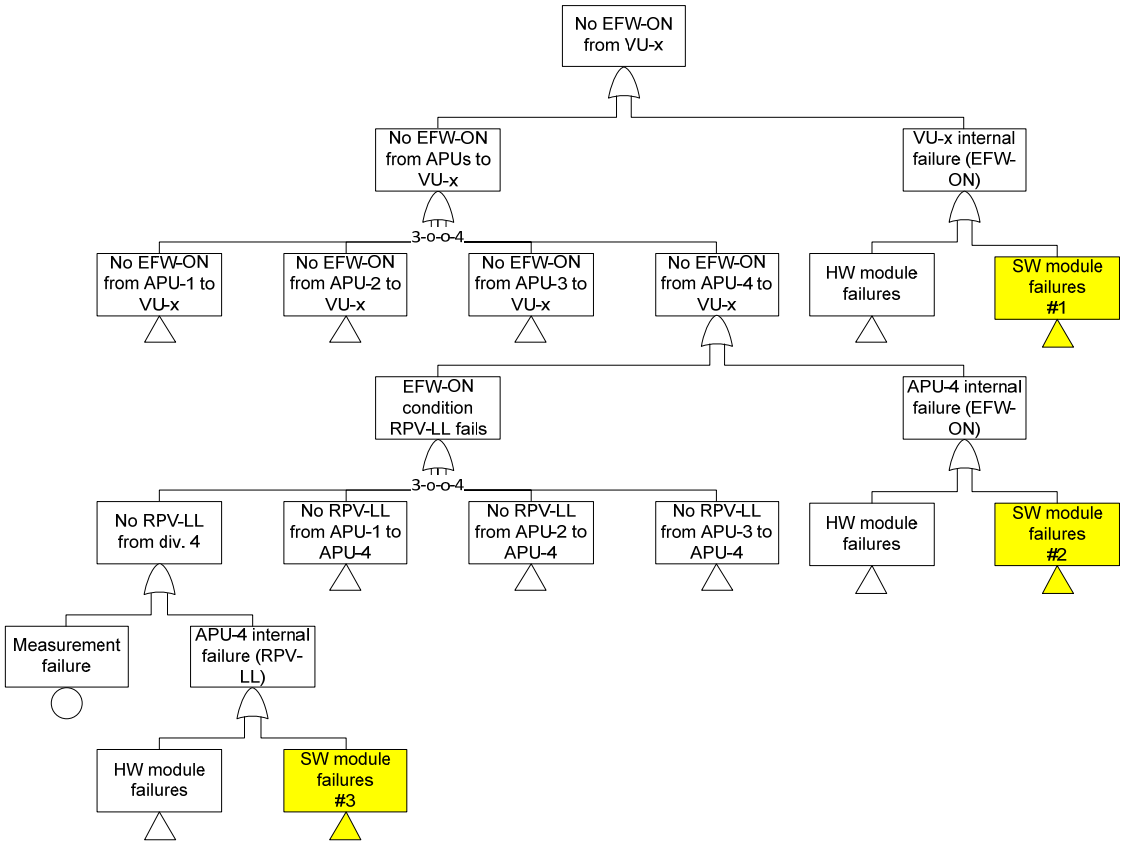

Figure 2. Schematic fault tree for failure to actuate EFW-ON in division x. Yellow transfer gates include software module basic events listed in Table 5.

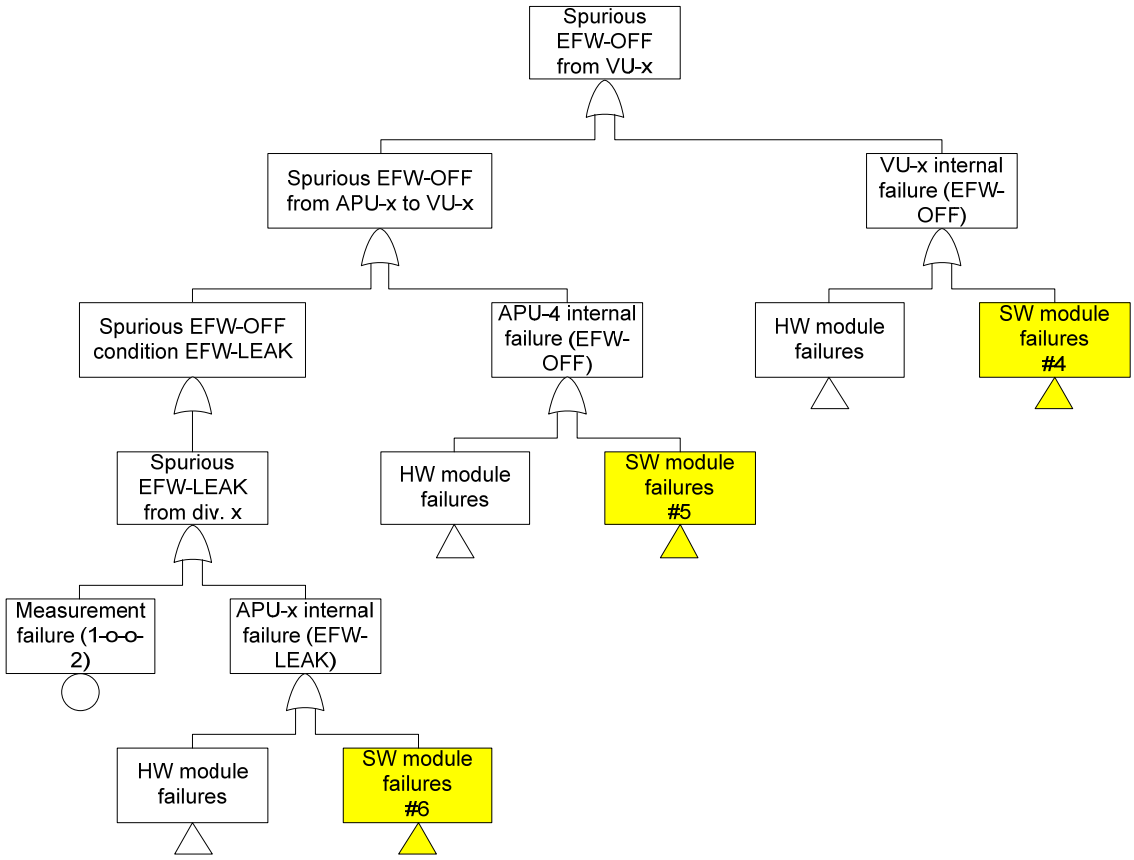

Figure 3. Schematic fault tree for spurious EFW-OFF in division x. Yellow transfer gates include software module basic events listed in Table 5.

Table 5. Software module level failure modes for the example safety function. Indexes #1–#6 refer to the transfer gates of the fault trees above.

Unit	EFW function failure mode	Software failure modes	
		Fatal faults	**Non-fatal faults**
Failure to actuate EFW-ON			
VU	#1 No EFW-ON from VU	• SYSTEM level CCF in SW modules (case 1) • 1SS level CCF in SW modules (case 2a) • 1VU level CCF in SW modules (case 3b)	• EFW-ON application SW fault in VU (case 4b)
APU	#2 No EFW-ON from APU to VU	• SYSTEM level CCF in SW modules (case 1) • 1SS level CCF in SW modules (case 2a and 2b) • 1APU level CCF in SW modules (case 3a)	• EFW-ON application SW fault in APU (case 4a)
	#3 EFW-ON condition RPV-LL fails in APU (3-o-o-4)		• RPV-LL application SW fault in APU (case 4a)
Spurious EFW-OFF			
VU	#4 Spurious EFW-OFF	• SYSTEM level CCF in SW modules (case 1) • 1SS level CCF in SW modules (case 2a) • 1VU level CCF in SW modules (case 3b)	• EFW-OFF application SW fault in VU (case 4b)
APU	#5 Spurious EFW-OFF from APU to VU	• SYSTEM level CCF in SW modules (case 1) • 1SS level CCF in SW modules (case 2a and 2b) • 1APU level CCF in SW modules (case 3a)	• EFW-OFF application SW fault in APU (case 4a)
	#6 Spurious EFW-OFF condition EFW-LEAK in APU		• EFW-LEAK application SW fault in APU (case 4a)

4 List of evidence

4.1 Description of the relevant evidence

Figure 4 illustrates a Bayesian Belief Net (BBN) for quantification of software reliability proposed in [10]. This model includes three main pieces of evidence which are proposed to be used in the quantification of probability of failure on demand (pfd) of an AS: Safety Integrity Level (SIL) class, software complexity and observations from usage and tests. The main rationale for the model is that development process and product quality affect the reliability of the software, which in turn affects the amount of discrepancies observed during usage and tests.

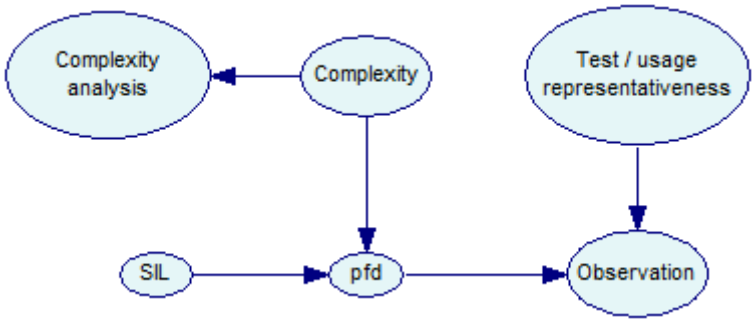

Figure 4. A BBN for assessing software reliability using SIL class, software complexity and usage and test observations as evidence.

The SIL class is assumed to give information about the quality of the software development process, including verification and validation (V&V) and installation tests. Product quality is represented by complexity of the software solution, with the assumption that more complex software is more likely to fail. However, complexity of software is not easy to define and measure accurately, so one may have to rely on indicative complexity metrics or expert judgements. Still, receiving even indirect evidence on the complexity of the software influences the beliefs on its reliability.

The observation node in the BBN includes all usage and test observations done after the installation tests, e.g. maintenance and periodical tests are included in this node. Normally no errors are found in the software at this stage, and known errors are fixed. The value of this information depends on the representativeness of the observations with respect to the possible and foreseeable state space of the software. Since this state space is huge, the representativeness of tests and even of operation experience has traditionally been seen as weak by regulators, and they would rather rely on the quality of software V&V measures.

Although the absence of findings during tests or usage do not guarantee a low software pfd, their presence help to calibrate the weight of the other BBN nodes, because a reliable model cannot underestimate an already known operational failure rate.

4.2 Discussion on basis V&V procedure, SIL

The IEC 61508 standard defines a generic approach for analysis of systems comprised of electrical and/or electronic and/or programmable electronic elements. The standard introduces the concept safety integrity level, SIL [7]. The different levels have different requirements on how the equipment should be manufactured and tested and also on how the software is being developed. The standard does however not include a quantitative assessment of the software reliability.

Safety integrity level and corresponding assumed failure probability and failure rate are presented in the table below.

Table 6. Probability intervals of safety integrity levels [7].

Safety integrity level (SIL)	**Average probability of a dangerous failure on demand of the safety function (PFDavg)**
4	$\geq 10^{-5}$ to $< 10^{-4}$
3	$\geq 10^{-4}$ to $< 10^{-3}$
2	$\geq 10^{-3}$ to $< 10^{-2}$
1	$\geq 10^{-2}$ to $< 10^{-1}$
Safety integrity level (SIL)	**Average frequency of a dangerous failure of the safety function [h^{-1}] (PFH)**
4	$\geq 10^{-9}$ to $< 10^{-8}$
3	$\geq 10^{-8}$ to $< 10^{-7}$
2	$\geq 10^{-7}$ to $< 10^{-6}$
1	$\geq 10^{-6}$ to $< 10^{-5}$

It was extensively discussed within the report [8] if the SIL level could be used to estimate the software failure probability, and the conclusion from that project was that the software failure probability could be argued to be represented by the SIL level (conservatively). That is however considered to be conservative by software experts, and it could therefore be further argued that the software failure probability should at least not be above the lower bound of the SIL.

The basic idea in this report is that the main process quality indicator that can be used to estimate the quality of the software is the safety class or SIL class of the system. In nuclear context, cat. A usually is interpreted as SIL3, see for example 2010 edition of the Safety Critical Systems Handbook [9]. We might assume that in some cases the vendor or the utility may want to claim that more efforts have been done, a better value for the design characteristics and V&V process may be justified (cat. A+). Further taking into account that the method will be applied to cat. B systems (SIL2), a lower value for V&V may be assumed. This reasoning leads to a categorisation of V&V into three levels (relevant to nuclear PSA applications):

- low = cat. B = SIL 2
- medium = cat. A = SIL 3
- high = cat. A+ = SIL 4.

It remains to be discussed how cat. A+ differs from cat. A. A possible interpretation: implementation of the RPS in two subsystems executing different functions allows claiming a higher reliability for the total solution, by crediting the functional diversity.

4.3 Discussion on complexity of software

Concerning the reliability of software, one of the most important properties is its complexity. Especially when used with several different operational profiles, much too complex software is more prone to present failures. A measure of the software complexity is therefore a key contributor to the assessment of its reliability.

There is however no widely accepted general method to calculate software complexity. Those which exist are difficult to be automated, a very important feature for the analysis of the, usually, very large amount of software.

In this section two recent approaches for the estimation of software complexity are presented, TOPAAS (Task Oriented Probability of Abnormalities Analysis for Software) and the ISTec's approach.

4.3.1 TOPAAS Approach

The TOPAAS method is based on critical software properties and expert opinions, which are captured into a factor model to calculate the probability of failure of software modules (see [11]). The most important factors that determine the software quality are first defined and for each of these factors performance levels (e.g., good, less good, average, bad) are decided by experts. Finally, numbers are assigned to these levels, expressing the importance of the factor and the impact of the level. If n factors F_i are considered, the failure probability by demand P is calculated as $P = P_B \ F_1 \ F_2 \ ... \ F_n$, with P_BPB as the base failure rate, which is conservatively set to 1. If a factor Fi is unknown, then this factor is set to 1. This implies that missing information results in a conservative bias of the model. The lowest attainable level of software failure probability P is assumed to be 1E-5 per demand. An example of a standard software module of a valve control device is presented. For this example, the failure probability for the software module is calculated to 4E-5 per demand, which is conservatively rounded to 1E-4 per demand.

The TOPAAS defines an approach for the calculation of the probability of failure in software intensive systems. One of the inputs for this method is the software complexity and for its estimation TOPAAS uses the McCabe index. Another product characteristic which is taken into account is the number of lines of code. In case an input is not known, the correspondent contribution is simply a multiplication by one.

4.3.2 ISTec Approach

In 2010 the institute ISTec (Institut für Sicherheitstechnologie) produced a document [12] where a method for the special case of the assessment of the complexity of safety relevant software in NPP was proposed. That is a very special kind of software, because it is produced using integrated tool environments: the software is designed by using graphical tools where function blocks are interconnected to form logic diagrams. From that graphic specification a code generator build the final source code.

One important aspect of the ISTec's approach is its easiness of automation. In fact, the method was prototypically applied to approximately 1000 logic diagrams describing the reactor control and limitation system in a German NPP (based on TXS).

In this approach, two levels of complexity are identified: function blocks and logic diagrams, the first on the most elementary level and the second reflecting the way the function blocks are connected to each other. For each of these two levels, rules are established for the calculation of complexity. These rules were implemented either using commercially available software or as automated tools created by ISTec for this purpose.

In order to make the method as applicable as possible, two complementary approaches were considered for the measurement of complexity. The white box view is based on the static analysis of the source code of the function block, which delivers standard complexity indicators: number of knots, McCabe index, code length, etc. Notice that the last two indicators are also used in the TOPAAS method. Since the access to the source code is not always possible, the complementary black box view is used, which is based on the analysis of the information present in the software user documentation (number of parameters, memory requirement, number of pages of text, etc.).

For the analysis of the logic diagrams only the black box approach was applied. For each logic diagram, the following quantities were calculated:

- Number of function blocks
- Number of input signals
- Number of output signals
- Number of upstream logic diagrams
- Number of downstream logic diagrams
- Normalized complexity of the function block interconnections (similar to the McCabe index)
- Number of changeable parameters
- Number of internal memories
- Complexity of the function blocks themselves (information coming from the function block analysis)

Each of these quantities is a component of a so-called complexity vector of the logical diagram. All complexity vectors of an I&C system can be averaged in order to have an aggregated indicator of its software complexity. This averaged complexity vector can then be further used for the assessment of the software reliability.

4.4 Discussion on test and user experience

This chapter presents a discussion on test and user experience based on the TELEPERM XS (TXS) system platform developed at AREVA GmbH.

4.4.1 Verification and Validation process of TXS

The main steps involved in the verification and validation process of TXS are detailed in this chapter.

The implementation of the software of TXS I&C systems is performed effectively in the specification and coding environment (SPACE), where the function diagram editor, code generators, and verification functions are integrated. This means that the application-specific data for the software and network architecture of the TXS computers are specified graphically in form of function, arrangement and network diagrams and then centrally stored in a data base. This data base is used to generate the complete plant-specific software code by means of automatic code generators.

During the detailed design and manufacturing phase of a TXS system, the verification and validation of the software is executed in three steps (see [15]):

1. *Formal check of the project data base*: the verification steps and tests which follow the formal specification serve mainly to prove that
    - The specification is clear, accurate and free of contradictions and the specification fulfils the process engineering requirements.

    Prior to the code generation, the documentation tools are called to check the data base contents for consistency. Source code for the application software can only be generated from a consistent specification. The same checks are also performed by the code generators during the source code generation. If an inconsistency is detected by the code generators, source code files will not be generated.
2. *Visual check of the specification, system meetings*: this step verifies that the process system requirements have been fulfilled. This is established by manually checking the specification (represented as function diagrams) against the requirements.

3. *Functional tests in the simulation environment*: the SPACE engineering system and the simulation-based validation tool (SIVAT) allow the specification and performance of functional tests on the configured I&C functions within a software simulation environment. The simulation results serve to prove the functionality of the I&C functions. Depending on the functions involved, the functionality can also be validated with a closed-loop simulation (e.g., using an engineering simulator in combination with models of the plant).

The verification and validation is completed with the system and functional tests in the test field (system integration phase) and during commissioning.

In the system integration phase, the software is loaded into the processors and the system functions are tested. This functional test consists in feeding input signals into a computer and checking the output signals using test computers. Additionally, a service computer is used to test how the signals are acquired and processed "inside" of the computer that is being tested.

The scope and coverage of functional tests during the detail design phase and the system integration phase is defined in line with the international requirements given in e.g., IEC 61513 and IEC 60880. The scope of functional tests is documented in a test concept report, where the objectives and coverage of the tests are defined as a basis for the elaboration of a test plan and test specifications. All implemented I&C functions in all redundancies are tested. There are different strategies for the functional tests involving different depth levels, e.g., test of all signals, test of the structure (e.g., all connections within one function), "negative tests" (to proof that an undesired behaviour of the system cannot occur), failure-performance tests (to proof if the function is still available in case of a failure of parts of the I&C system).

Finally during the commissioning phase, the functionality of the I&C system is tested in combination with the plant, i.e., no specific software tests are performed.

Since the software is tested extensively in the frame of V&V, software identity and integrity tests are considered to be sufficient for periodical tests. The software identity and integrity testing is performed through the evaluation of check sums. This test is automatically performed through the TXS self-monitoring system. The aim of this test is to proof that the loaded software in the processors did not change. The test involves the calculation of the check sum of the software code of a processor and the comparison of this value with the check sum of the original code loaded in this processor. The fact that both check sums are exactly the same confirms that the software code in this processor did not change. Note that if a fault in the FRS or a fault in the implementation of the application functions is not detected during V&V, these will not be detected during periodical tests.

If the software is updated or an I&C system modernization takes place, a V&V strategy has to be elaborated. The scope of the tests to be considered is defined by means of an impact analysis, where the repercussion of the changes (e.g., new software, new system parts) on the complete system is analysed, including the parts which remain unchanged. Functional tests in the SIVAT simulation environment are included. Depending on the scope of the changes, simulation tests on a representative environment and on the field may also be included, completed by commissioning tests.

Furthermore, according to [16] all these V&V steps in TXS, required by the nuclear regulations for the performance of category A functions (IEC 60880), can also be

understood as fulfilling the correspondent V&V requirements of the SIL 3 (IEC 61508, industrial regulation).

4.4.2 Operating experience of TXS

The operating experience of the TXS platform has been assessed in 2008 and it is based on the user experience of more than 60 nuclear-related plants worldwide (see [14]). These I&C systems are permanently in operation, are broadly monitored, and have been working reliably and accumulating applicable operating experience for over thirteen years. During the considered operating time and until the present no CCF caused by the TXS platform was experienced.

Problems observed during power operation, deficiencies of released products and components found during engineering, design or testing activities but also deficiencies found during internal/external audits are documented in non-conformance reports (NCR). After a NCR initialization, the further processing of NCR is tracked in a data base by a dedicated team separately and independently from the engineering and product development teams in order to ensure an independent evaluation. The NCR data base contains those faults and failures which constitute a significant deviation of released products from their specification [13].

For each of these non-conformances, it was analysed if the non-conformance had the potential –if undetected- to be triggered by a CCF initiator and what could be the impact (resulting down time). From this analysis, the following software CCF triggering mechanisms have been identified as relevant:

- *Temporal effects*: this group encompasses all CCF which may be triggered by time-dependent effects (internal trigger mechanisms), such as the depletion of resources by time (e.g., leakages in the memory allocation), or by accumulated time of operation. Affected by this CCF cause are all processors with the same operating time, which usually includes all processors of one subsystem (case 2 in Table 2).
- *Faulty telegrams*: this group considers all CCF which may be triggered by the transmission of information via serial data links. The failure mechanism is given by the existence of an undetected random failure in a sending CPU causing transmission of invalid data. If the system software of the receiver processor contains an undetected fault in the validation of the received data (e.g., wrong implementation of message checking), the corrupt data remain undetected. If these corrupt data are processed an exception (interrupt) is activated. The impact of this CCF cause is restricted to all units with direct communication. According to the architecture of Figure 1, communication exists between APUs and between APUs and VUs within one subsystem (no communication between RPS-A and RPS-B). For this reason this CCF affects the APs and VUs within one subsystem (case 2 in Table 2).
- *Same signal trajectory*: this group encompasses all common cause failures which may be triggered by a sequence of input data from the field (external trigger mechanism). It cannot be ruled out that the function computers which have the same operating system and same application software and process exactly the same signal trajectories may fail simultaneously. This CCF cause presumes that a very rare (not tested) signal trajectory may be combined with a latent hardware or software fault. For the analysis of this trigger mechanism it is convenient to differentiate between two categories:

- Category 1: the latent fault is located within the software, e.g., systematic fault in a TXS function block – elementary function – involved in the application software,) and leads to a *fatal failure.* In the case of a latent software fault, the fault has an impermissible interference (exception) on the system behaviour, such as the incorrect computation of a command executed with inoperable values due to a design error (e.g., division by zero, logarithm of negative values). As a consequence, the application function can no longer be processed as designed. For such cases, pre-checks and remedial actions are implemented in the application and system software to handle the faults on the level of the affected function and to protect the processor against a stop due to software exceptions. Faults can be captured by pre-checking algorithms, which handle the situation by substituting suitable values for computation if necessary. If the pre-check of data fails (lack in the pre-checking routines, inadequate activation of remedial actions, faulty/incomplete implementation of remedial actions) an exception handler is activated that interrupts the cyclic process in the processor module and set the signal values into defined fail-safe values. Trajectories with exactly the same sequence of data may only happen between the APUs within one subsystem, such that this CCF trigger mechanism is restricted to affect at most all APUs within one subsystem (e.g., RPS-A or RPS-B of Figure 1; see also case 3 in Table 2).
- Category 2: the latent fault is located in the FRS or in the application software and leads to a *non-fatal failure*. This CCF initiator presumes that a not-tested signal trajectory may be combined with a latent fault in the FRS or in the application software (e.g., incorrectly designed set point value). In this case the requested function is not executed (or a different response than the requested is obtained) but the TXS processing unit continues to operate cyclically. The impact extent is restricted in this case to the specific application function (case 4 in Table 2).

The operating experience of the potential CCF causes addressed in [14] is summarized in Table 7. The CCF triggering mechanisms and the latent fault location, i.e., system software (SyS), application software (AS), communication of processors (DLC) or hardware (HW) is also indicated. Note that all observed failures of the TXS platform correspond to single failures with no evidence of CCF events.

The fatal failure modes triggered by “temporal effects” and “faulty telegrams” in the system software and by “same signal trajectory” in the application software (category 1) lead to a processor shut down via an exception handler (detected common cause failures). This is followed by either a CPU reset including the start-up self-tests or by the immediate shut down of the processor. Because of the specific features of the TXS platform, no erroneous signal, even temporary, is possible before the processor shutdown. The downtimes caused by the failures with CCF potential are very short (see Table 7).

Table 7. Assessment of software CCF triggering mechanisms using the TXS operating experience.

CCF triggering mechanism		Latent fault location			Fault effects (see Table 2)	Failures in operation	Accumulated operation time [h]	Failure rate [1/h]	Event duration [h]
		SyS	FRS/ AS	DLC					
Temporal effects		x			case 2a	0(1)	3.4E+6	1.5E-7	0.25(2)
Faulty telegrams		x		x	case 2b	2	3.4E+6	7.4E-7	0.25
Same signal trajectory	Cat. 1	x	x		case 3	0(1)	3.1E+7(4)	1.6E-8	0.10
	Cat. 2		x		case 4	-(3)	-	-(3)	-

(1) Although no failures during the TXS accumulated operating experience are reported in [14], one dependent failure is assumed for the failure rate calculation.

(2) This value is estimated (not reported in [14]).

(3) A precise failure probability for application software faults triggered by the same signal trajectory cannot be predicted using operating experience of TXS because the possible influence mechanisms can only be detected in case of a demand of the function. No such a case has occurred until now

(4) The operating experience for this category reported in [14] does not allow the distinction between fatal and non-fatal failures. As a conservative assumption, the operating experience presented in this case can be considered as only due to fatal failures.

Note that the occurrence of a fatal failure in the TXS system software would not create spurious signals due to the strict separation between application and system software.

Taking diversity requirements between both subsystems (RPS-A and RPS-B in Figure 1) into account no relevant/realistic CCF mode of the TXS software which causes the complete failure of the system (both subsystems) can be identified.

5 Quantification method

5.1 Introduction to the quantification method

The quantification method depends on the type of software module. System software and application function software modules are considered relevant to model and quantify in PSA. The other SW modules could be ignored since their faults are implicitly covered by other cases.

Fault in system software (SyS) may cause in principle any type of end effect. The proposal here is, however, that only fatal failure of one subsystem (1SS) or both subsystems (SYSTEM) are considered. It is analytically very difficult to examine the reliability of a SyS but operating experience could be used as evidence. This approach is outlined in the next chapter.

Analysis of faults in application software (AS) is the main issue of the report. For them an analytical approach is suggested taking into account the complexity of the application function and the level of V&V process. Various failure effects and failure extents are considered using generic fractions (i.e. conditional probabilities). This approach is outlined in chapter 5.3.

Analysis of faults in FRS are part of the analysis of faults in AS.

Fault in EF can in principle cause any of end effect. The case ”fatal failures affecting redundant units” is covered by the SyS fault. Non-fatal failures are covered by corresponding AS-fault. It may be of interest to study whether some extra complex EF

is used in several AS, which causes a dependency between AS-modules. The most likely fault is not EF fault itself but that the EF is used in a wrong way in the AS – use of EFs is thus part of analysis P(AS-fault). Therefore there is no need to explicitly model EF faults.

Faults in proprietary SW modules are covered by HW faults from the end effects point of view. Therefore there is no need to explicitly model these proprietary SW module faults.

Faults in DCS and DLC may require some special treatment, due to possibly unique end effect, not necessarily covered by cases 1 and 2. However, the case ”fatal failures affecting redundant units” is covered by SyS fault, and thus faults in DCS and DLC are omitted.

5.2 System software (SyS)

The failures of a SyS should preferably be estimated for the system in question from operational history. The main challenge is to find historical events that have caused a complete fatal failure of the whole system.

Fatal failure of SyS is assumed to cause at least the failure of one subsystem (1SS). With sufficient data, this failure mode should be possible to estimate. The value calculated from operating experience represents thus the unavailability of one subsystem.

Depending on the degree of similarities between application functions of the two subsystems similarities, a fraction of faults may cause the failure of both subsystems (SYSTEM). The assessment of fraction requires an analysis of degree of diversity between the application functions of the two subsystems. Diversity assessment is out of the scope of this report and may be considered next year, if examples are available. For time being, it is suggested that without an analysis of degree of diversity, CCF between subsystems should be assumed (since it has not be ruled out). Tentatively a factor 0.1 may be used, which is a common CCF judgement in PSA when no proper data is available.

To summarise the SyS fault related basic events listed in Table 8 could be considered in the DIGREL example PSA model.

Table 8. SyS fault related basic events.

SW failure event	Tentative probability	Evidence
SW fault 1: SYSTEM-SyS fatal CCF	1E-7	Engineering judgement, 10% from 1APU/1VU (same signal trajectory, cases 3a/3b). This is where the diversity between application functions have an influence
SW fault 2a: 1SS-SyS fatal CCF	1E-6	Analysis of TXS operating experience
SW fault 2b: 1SS-DCU fatal CCF	1E-5	Analysis of TXS operating experience

Note that the failure probabilities given in Table 8 can be considered as inputs to model the software failure probabilities in the PSA using basic events.

5.3 Application software

5.3.1 Introduction to application software evaluation

The estimate of the application software failure probability is dependent on the processes that are run on the processor. On each processor, several application software may run.

A fault in one application software, which causes a fatal failure of the processor, affects also the other application software running in the same processor. Hence, a fatal failure can affect other processes running in the same processor– but only in the configuration that the information output stops.

A non-fatal failure in one application software can produce an incorrect output (or of course no output, but this is the same as incorrect output) but does not affect the other applications running in the same processor.

The failure probability of the application software is considered in this method dependent on the verification and validation procedure and the complexity of the software.

In section 5.3.2 a baseline failure estimate is developed, based on an estimate of complexity and V&V. This baseline failure estimate is representing the failure probability of the one AS (regardless of it is fatal or non-fatal). In the following section it is discussed how this baseline failure probability is expected to be used and modelled in different cases.

5.3.2 Baseline failure estimate, prior

The software fault probability in a system is hard to estimate. As shown in previous sections, the system can be broken down to a set of software fault parameters. But, the failure data available for software faults are not given on this level of detail. In fact, it is hard to find any collection of failure data.

In this section, we try to establish some baseline failure estimates. To do this, we have defined two types of measures for the system:

- Complexity
- Verification and validation

Complexity in a system is a parameter that is dependent on the size of the system, how many inputs that are handled, if there are delays or hold circuits and so on. It is a measure defined with the intent that complicated software should be more likely to produce a critical fault than simple software, given that the same level of verification and validation is applied. The complexity in a system is defined in the baseline estimate as a high, medium and low.

The verification and validation measure is believed to impact the software fault probability in the way that critical faults are expected to be much rarer in a system with high verification and validation principles. This could be compared with the SIL-system described in IEC-61508, where the obvious purpose is to reduce the failure probability of the system by increasing the requirements on the V&V process. Therefore, in the baseline risk assessment – the SIL is used as the estimator of the V&V process.

For a suggested method to be more than just another way of assessing software reliability, experience from different industries has been studied. The idea is to find examples that will justify some baseline failure probabilities for critical failures.

Below is a table with some initial assumptions on SW fault probabilities. It shall be noticed that we are assuming that a critical software fault will be CCF related between redundant AS that have the same task.

Table 9. Baseline failure probability estimates for application software modules.

		Complexity		
		High	**Medium**	**Low**
V&V	**0**	1.0E-1	1.0E-2	1.0E-3
	1	1.0E-2	1.0E-3	1.0E-4
	2	1.0E-3	1.0E-4	1.0E-5
	3	1.0E-4	1.0E-5	1.0E-6
	4	1.0E-5	1.0E-6	1.0E-7

The upper bound, a failure probability of 1E-1 per demand, would represent very complex software developed with a very simple verification and validation principle. In practice, this would not be applicable to the nuclear domain within RPSs. If such a system should be developed, the assumption that such a software should fail 1 time out of 10 is maybe a bit conservative, but yet reasonable.

As was discussed in section 4.2, the SIL level could be used as an estimate of the software failure probability. The lower failure probability for each SIL could be claimed to be an estimate of the highest failure probability the software can have. Hence, the leftmost column in Table 9 (High complexity software) could be argued to be representing the lower bound of the SIL bounding failure probability in Table 6.

If the complexity in the software is lower, then the software fault probability should be lower than what the SIL level is indicating. If a piece of software is of low complexity, but has the same type of validation, how much better could the software be claimed to be? In this process we have assumed that software with low complexity would be a factor of 100 better than the software with high complexity.

Justification

If a functional processor is considered medium complex software with a fairly strict V&V process, the grading according to the suggested method would be Medium on complexity and 3 on V&V. This would give a failure probability per demand of 1E-5, which, referring to previously applied data in nuclear PSAs may be a bit low — where a commonly used probability of failure for a functional processor in an I&C system is 1E-4 per demand. Certain I&C system suppliers base this on their claim that they have followed the requirements for SIL 4 developing the software. It is believed to be reasonable with the 1E-5 pfd assumption for such software, since the involved probabilities are based on very rough estimates.

Data collected by AREVA [14] states that the fatal failure rate for the software in the TXS system is in the range of 1.6E-8 per hour, based on operating experience (for details see section 4.4.2). The failure rate represents failures due to impermissible interferences from the application software on the system processors. The failure

probability for the system can be calculated as the probability that the system is failed when an initiating event occur plus the likelihood that the system stops during a transient. The repair time is hence also relevant (0.1h) for impermissible interferences from the application software with fatal consequences. The transient time is assumed to be 24 hours. Based on this the failure probability of AS (fatal failures) is calculated to 3.9E-7. Hence, compared to the baseline failure probabilities of Table 9, it indicates a SIL3 application of Low complexity or a SIL4 application of Medium complexity. This seems reasonable.

In the presentation *Reliability of New Plant Automation of Loviisa NPP* [18] the failure probabilities used for the software at digital safety I&C were discussed. The probability of software failures affecting more than one division is 5E-5 (within same automation system). The presentation also considered data used for CCF between different systems, but that is not relevant in this context. The failure probability is reasonably consistent with the baseline failure probability for Medium complex software for SIL 3.

A study of software related failure within the U.S. Public Switched Telephone Network (PSTN) is presented in the paper *Sources of failure in the public switched telephone network* [19]. The telephone switching network performs a fairly simple task by connecting point A with point B, but requires a very complex computing system. Software for a switch with even a relatively small set of features may comprise several million lines of code. It is pointed out that the telephone switch manufacturer's software development process typically includes elaborate quality assurance functions. Relating this to Table 9 would give a highly complex system, complexity grade High, with an advanced verification and validation process, V&V grade 4, which in its turn gives a probability of failure per demand of 1E-5. The author of [19] has studied the system outages from April 1992 to March 1994 and concludes that the PSTN averaged an availability rate better than 99.999% during that time period. The software related failures, including those in recovery mode, account for 14% of the number of total outages. This would indicate that the failure probability of the system would be at least 14% of 1E-4 ~ 1E-5. This would then be reasonably consistent with the table above.

In the paper *Failure modes in medical device software: an analysis of 15 years of recall data* [20] Dolores R. Wallace and D. Richard Kuhn examine recall data for medical devices due to software failures using the U.S. Food and Drug Administration database of medical device failures. It is concluded that 6% of the 2,792 medical devices recalled between 1983 and 1991, including devices not containing software, were recalled due to software related failures. For the devices recalled between 1992 and 1997 the data were not complete, but the results were within the same range. The years 1994, 1995, 1996 had 11%, 10% and 9% of the software recalls which, according to the authors, could be a result of the rapid increase of software in medical devices. The conclusion from the paper is that there should be more quality assurance procedures to mitigate software failure. Based on this we could assume that the V&V processes are of reasonably low score and thereby would fit in the low grade V&V. For reasonably complicated software this would indicate an annual failure probability of 1E-2. This estimate is however very uncertain, but is yet reasonable when compared with the table.

The TOPAAS method [11] developed by the Dutch Rijkswaterstaat includes considerably more grading points than the method suggested here. The example presented in the TOPAAS document includes between 10,000 and 50,000 lines of code and is stated to have a fairly simple logic. The product fulfils the requirements for SIL 1 and it is assumed to have medium-low complexity. The suggested method in this document would yield a probability of failure per demand of 1E-4 –1E-3. The

calculated failure probability based on the method in [11] is 2.2E-5. Even though there is a difference between the methods, the TOPAAS method shows that the failure probability can be significantly lower than the SIL level indicated failure probability – and there is a reasonable consistence between the methods.

5.3.3 Outline of representations

The data collected and presented in previous section does not discuss "what is a system" or what is the "origin of the fault"? It is however reasonable to believe that as soon as a fault is detected, the software is considered faulty. Within a process there may in reality be several AS running – in parallel. If there is a fault in one of them, this will only affect the other AS if the overall process is stopped, halted. This would cause all the AS on that processor to stop working and give no output.

Hence, a failure in the specification or the programming of one of the processes may influence the others, but only if the process runs into a fatal fault. This is a subset of the failures for the AS. Table 9 is considered to be an estimate of all the failures, fatal and non-fatal, for an AS. Table 9 is also considered to include failures in the FRS, or rather, the complexity in the FRS defines the complexity in the software – and the V&V is the measure of the implementation process.

Illustrated by Figure 5 is the process of estimating fatal and non-fatal failures and also the split of non-fatal failures into failure to actuate or spurious actuation failure cases.

It can be noticed from Figure 6 and Figure 7 that regardless if the modelled failure mode is failure to actuate or if it is spurious actuation, it is estimated given the full fraction of non-fatal failures in the current model (i.e., being a fraction of the total probability P(AS1 fault)).

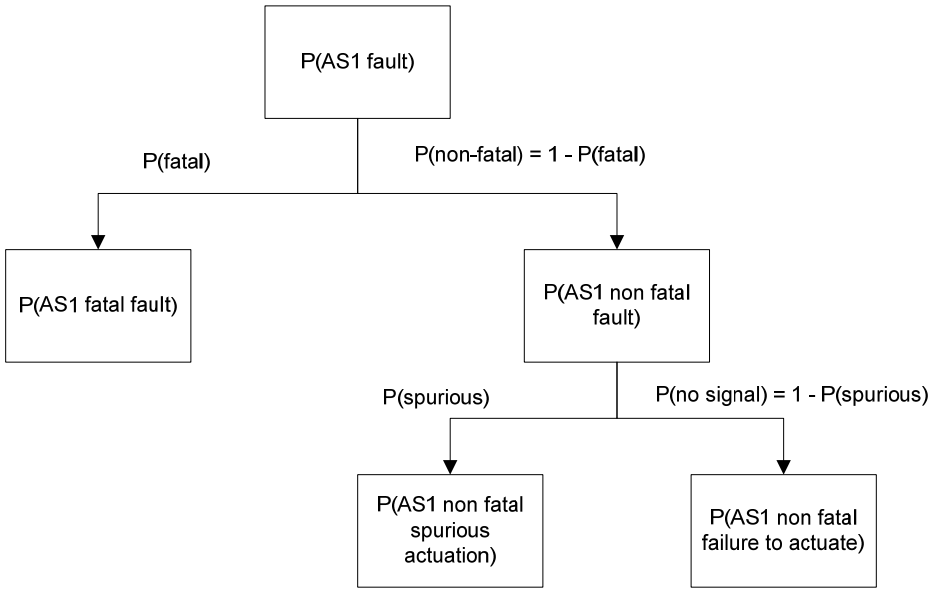

Figure 5. How to estimate the software fault probability based on Table 7 in fatal and non-fatal (spurious and no signal scenarios).

For the estimation of the fractions failure to actuate resp. spurious actuation it has to be taken into account that in general a significant effort is spent to implement fault propagation barriers, self-tests, self-monitoring and plausibility checks in soft- and

hardware to capture any detected malfunctions inside a system, and to direct these detected malfunctions into safe states. Therefore, non-fatal failures leading to spurious actuations should be considered as less frequent than non-fatal failures leading to failures on demand.

Given the fact that there is at this stage of this project no evidence available how to split the non-fatal failures into fractions representing failure to actuate resp. spurious actuation one could conservatively assume the same failure probability for non-fatal failures leading to failure to actuate and to spurious actuation (for this topic, future collection of evidences is of high interest).

Referring back to Table 3, the first three rows in the table are represented by the P(AS1 fault) in the figure, whereas rows four and five in the table are representing the fatal/non-fatal failure fractions and rows six and seven are representing the failure to actuate and spurious actuation cases.

An alternative approach to estimate fatal failure probability in application software could be to use operational history, if sufficient history is available. Which method to use to estimate the fatal failure probability will be discussed further in the continued work. Also, as described in section 3, which of the fatal and non-fatal failure modes that are relevant in the fault trees are also dependent on the system functionality. The figures below illustrate the situations for no-signal scenario (in a fail-safe and non-fail-safe configuration) and spurious signal (in a fail-safe and non-fail-safe configuration). It shall be noticed that a fatal failure in another process may influence the behaviour of the studied AS, as discussed above.

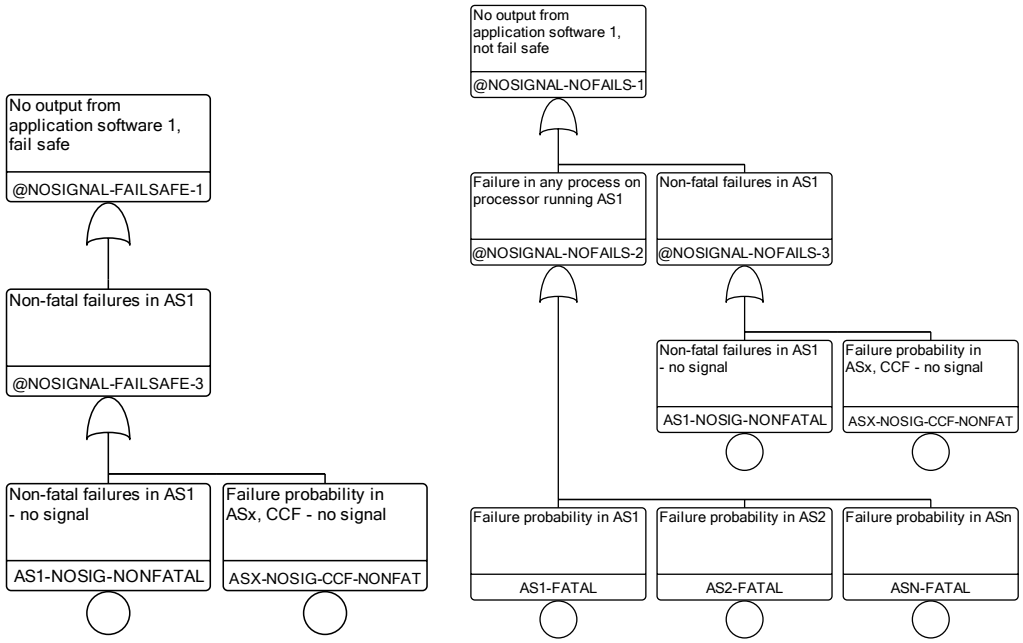

Figure 6. Left figure: No signal in a fail-safe configuration. Right figure: No signal in a non-fail-safe configuration.

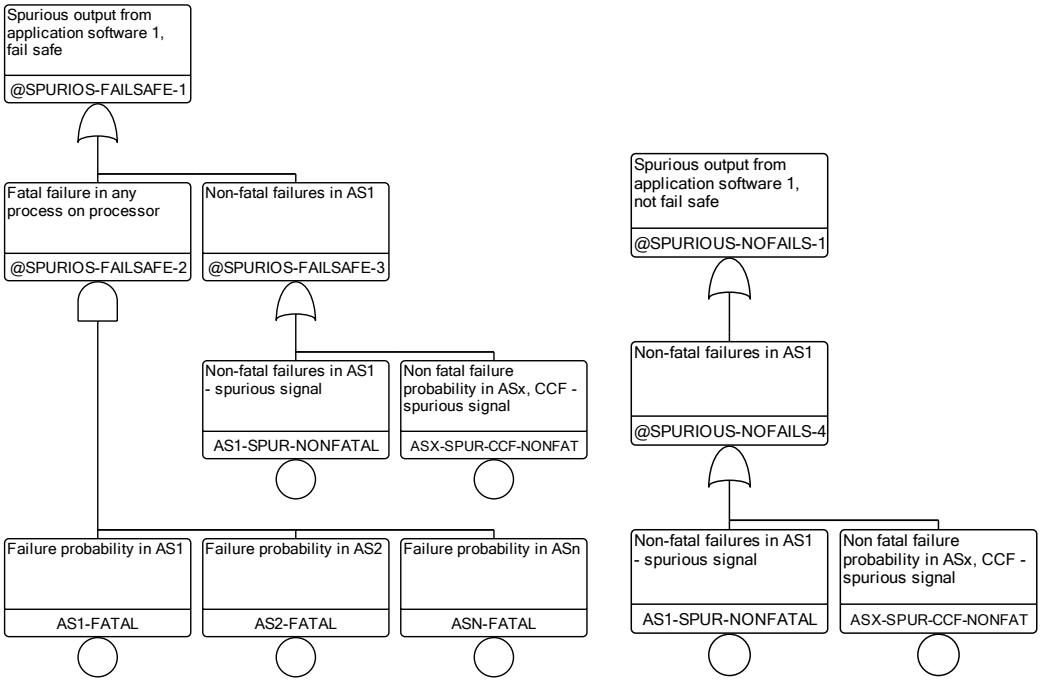

Figure 7. Left figure: Spurious signal in a fail-safe configuration. Right figure: Spurious signal in a non-fail-safe configuration.

5.3.4 Summary of quantification of application software failure probability

The baseline failure probability for the application software is estimated from Table 9. The failure probability is dependent on the software complexity and the V&V program. In case there are observations, after the installation tests, these should be possible to use as evidence for a better approximation of the software failure probability. This part of the method is however not defined yet.

Further investigation on this topic includes the estimation of the fraction for fatal and non-fatal failures. The fatal failure probability may also be estimated based on operational history.

Which failures that can cause spurious and no signals are dependent on the system layout. For example, if the system is designed in a way that the safe end state will not cause spurious stop signal to the system – then only the non-fatal failures will have a potential for causing spurious signals. Fatal failures would in this example only cause no signal scenarios.

CCF is especially interesting for functional requirements, and these are in their turn especially interesting for non-fatal failures. A simple application model for CCF is being considered.

6 Plan for 2014

In 2014, the plan is to perform an example including several types of software basic events in order to test and demonstrate the method.

Guidelines document will be prepared for the method. Some issues to address in the guidelines are

- How to assess complexity?
- Can better V&V-class than SIL 3 be justified sometimes, i.e., SIL 4 representing requirements which are stricter than cat. A (extra V&V measures are taken)?
- How to assess CCF between application SW modules which have common functional requirements specifications?
- The estimate of application software failure probabilities needs further discussion regarding fatal failure treatment, fractions of fatal failure or failure to actuate / spurious actuation (if applicable).
- How to take user experience (additional tests and experiences) into account?

7 References

1. Authén, S, Björkman, K., Holmberg, J.-E., Larsson, J. Guidelines for reliability analysis of digital systems in PSA context — Phase 1 Status Report, NKS-230 Nordic nuclear safety research (NKS), Roskilde, 2010.
2. Authén, S., Gustafsson, J., Holmberg, J.-E. Guidelines for reliability analysis of digital systems in PSA context — Phase 2 Status Report, NKS-261 Nordic nuclear safety research (NKS), Roskilde, 2012.

3. Authén, S., Holmberg, J.-E., Guidelines for reliability analysis of digital systems in PSA context - Phase 3 Status Report, NKS-277, Nordic nuclear safety research (NKS), Roskilde, 2013.
4. Failure modes taxonomy for reliability assessment of digital I&C systems for PRA, report prepared by a task group of OECD/NEA Working Group RISK, draft January 2014.
5. Recommendations on assessing digital system reliability in probabilistic risk assessments of nuclear power plants, NEA/CSNI/R(2009)18, OECD/NEA/CSNI, Paris, 2009.
6. Chu, T.L., Martinez-Guridi, G., Yue, M., Samanta, P., Vinod, G., and Lehner, J., Workshop on Philosophical Basis for Incorporating Software Failures into a Probabilistic Risk Assessment," Brookhaven National Laboratory, Technical Report, BNL-90571-2009-IR, November 2009.
7. International Electrotechnical Commission, "Function Safety of Electrical/Electronic/Programmable Safety-Related Systems," Parts 1-7, IEC 61508, various dates.
8. Björkman, K., Bäckström, O., Holmberg, J.-E., Use of IEC 61508 in Nuclear Applications Regarding Software Reliability — Pre-study, VTT, VTT-R-09293-11
9. Smith, D.J. Simpson, K.G.L. Safety Critical Systems Handbook Safety Critical Systems Handbook. A Straightforward Guide to Functional Safety: IEC 61508 and Related Standards Including: Process IEC 61511, Machinery IEC 62061 and ISO 13849, 3rd edition, 2010.
10. Porthin, M., Holmberg, J-E., Modelling software failures using Bayesian nets, VTT Research Report VTT-R-08279-12, 2013.
11. Rijkswaterstaat Ministerie van Verkeer en Waterstaat, TOPAAS: Een structurele aanpak voor faalkansanalyse van software intensieve systemen, 01.04.2011
12. Komplexitätsmessung der Software Digitaler Leittechniksysteme, ISTec-A-1569, J. Märtz H. Miedl A. Lindner, Ch. Gerst, 2010.
13. AREVA GmbH, Managing operating experience with TXS. Report PTLS-G/2010/en/0046 Rev. A; 28.01.2008
14. AREVA GmbH, Quantitative Bewertung des Einflusses eines GVA auf die Unverfügbarkeit von Leitsystemen in TELEPERM XS Gerätetechnik. Work Report NLR-G/2009/de/0001 Rev. A; 04.09.2009
15. AREVA GmbH, Overview of the TXS CORE Software (for Releases 3.1.x to 3.3.x) TELEPERM XS Manual TXS-1073-76-V3.0
16. VDI/VDE 3528- Gesellschaft Mess- und Automatisierungstechnik (GMA), Requirements of repetition parts and criteria for their use in the instrumentation and control to safety in nuclear power plants. Part 1, ICS 27.120.20, August 2011
17. Proceedings of the DIGREL seminar "Development of best practice guidelines on failure modes taxonomy for reliability assessment of digital I&C systems for PSA", October 25, 2011, VTT-M-07989-11, Espoo, 2011.
18. Jänkälä, K., Reliability of New Plant Automation of Loviisa NPP, Presentation at NKS seminar at VTT September 2010
19. Kuhn, R., Sources of Failure in the Public Switched Telephone Network, Computer 00-18-9162/97/$10.00 ©1997 IEEE
20. Wallace, D.R., Kuhn, R, Failure modes in medical device software: An analysis of 15 years of recall data, Paper

Title	Software reliability analysis for PSA
Author(s)	Ola Bäckström[1], Jan-Erik Holmberg[2], Mariana Jockenhövel-Barttfeld[3], Markus Porthin[4], Andre Taurines[3]
Affiliation(s)	[1]Lloyd's Register AB, Sweden, [2]Risk Pilot AB, Sweden, [3]AREVA GmbH, Germany, [4]VTT Technical Research Centre of Finland
ISBN	978-87-7893-381-2
Date	March 2014
Project	NKS-R / DIGREL
No. of pages	32
No. of tables	9
No. of illustrations	7
No. of references	20

Abstract

A project is ongoing, financed by Nordic nuclear safety research (NKS), The Finnish Research Programme on Nuclear Power Plant Safety (SAFIR2014) and Nordic PSA group (NPSAG), with the intent to provide guidelines to analyse and model digital systems in probabilistic safety assessment (PSA), using traditional reliability analysis methods (FMEA, Fault tree analysis).

This report discusses software reliability in this context. The report proposes a method for the evaluation and quantification of reactor protection system (RPS) software failures. The proposed method will use operational history to estimate the fatal failure probability within system software (operating system, runtime), and use an indirect method for the estimation of failure probability within application software (non-fatal and fatal failures). The quantification for application software is based on two main measures, complexity and the degree of verification and validation of the software.

Collection of data and its challenges will also be discussed. Some data collected for a software platform will be discussed, and used as an example of the difficultness — and challenge — to collect data.

Key words PSA, Software reliability, Operational history data