Quantification of reactor protection system software reliability based on indirect and direct evidence

Ola Bäckström[2], Jan-Erik Holmberg[3], Mariana Jockenhoevel-Barttfeld[4], Markus Porthin[1], Andre Taurines[4]
[1]VTT Technical Research Centre of Finland, Espoo, Finland
[2]Lloyd Register Consulting, Stockholm, Sweden
[3]Risk Pilot, Espoo, Finland
[4]AREVA GmbH, Erlangen, Germany

Abstract:
This paper presents a method for the quantification of software failures in a reactor protection system in the context of probabilistic safety assessment (PSA) for a nuclear power plant. The emphasis of the method is on the quantification of the failure probability of an application software module, which can lead to the functional failure modes: failure to actuate on demand a specific instrumentation and control (I&C) function or spurious actuation of a specific I&C function. The quantification is based on two main metrics, complexity of the application software and the degree of verification and validation of the software. The relevance of common cause failures and an analysis of the impact of fatal and non-fatal failures on the system will be covered by the discussion. Collection of operational data and challenges to use it for software reliability quantification will also be discussed. The outlined quantification method offers a practical and justifiable approach to account for software failures that are usually ignored in current PSAs.

Keywords: PSA, Software reliability, Operational history data

1. INTRODUCTION

Currently, no common approach is available in the nuclear power plant (NPP) field for assessing safety and reliability of digital instrumentation and control (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.

This paper presents a method for quantification of software failures in a reactor protection system (RPS) 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 paper is organised as follows. In Chapter 2 the software fault modes are first identified. In Chapter 3the list of evidence is discussed, taking into consideration the verification and validation (V&V) and the complexity of the software as well as software operating experience. In Chapter 4 the proposed quantification method is outlined. Finally conclusions are presented in Chapter 5.

The work is part of the Nordic DIGREL project [1, 2, 3]. This work 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. DEFINITION FOR SOFTWARE FAULTS

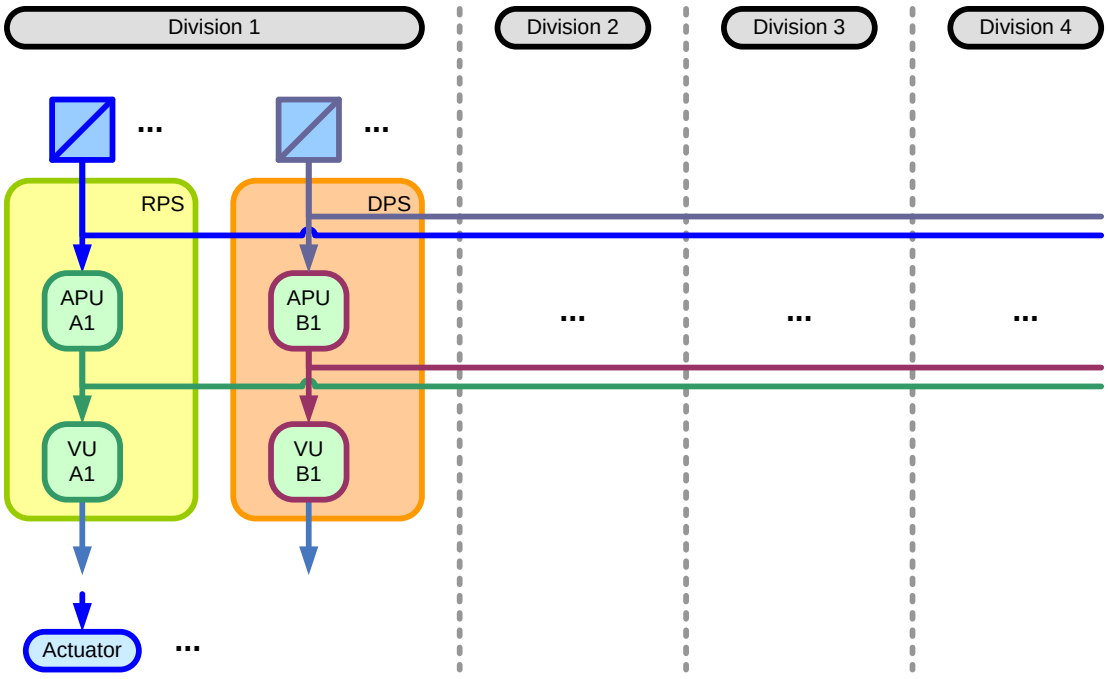
2.1. Example safety I&C architecture

DIGREL project primarily considers the RPS of a nuclear power plant, since it is considered to be more important for PSA than other I&C systems and it is considered a conceivable target for the activity.

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 reactor protection system (RPS) and diverse protection system (DPS), both divided into four physically separated divisions.

The extent of diversity between RPS and DPS may vary, but we may generally assume that they perform different functions. The platforms of both subsystems are assumed to be identical, in order to include the platform CCF in consideration. 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.



2.2. 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).*
- APU functional requirements specification modules (APU-FRS). 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). Their purpose is to allow the representation of errors in the implementation of application-specific acquisition and processing software.
- Proprietary software (Propr. SW) in I&C. Other modules than the processor module. Specific pieces of software 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). Their purpose is to allow the representation of errors in functional requirements specifications of the voting functions.
- VU application software modules (VU-AS). Their purpose is to allow the representation of errors in the implementation of application-specific voting software.
- Data communication software (DCS). Operating system of DCUs.
- Data link configuration (DLC).

Two failure effects are considered from the hardware module point of view. A software fault leads either a *fatal* or a *non-fatal* failure of the hardware module (e.g. the processor module). A fatal failure means that generation of outputs is ceased and e.g. in the processor module the exception handler sets the output values into defined fail-safe values. A non-fatal failure means that the module continues to operate, but one or more output values are wrong.

Depending on the location of the software fault, failure effect and system architecture, one or more units in one or more subsystems can be impacted. The report [4] presents a list of maximum failure extents of a postulated event. Because it would be impractical to take all of them into consideration in the PSA model, the most relevant can be identified. The following software faults and effects are considered in this paper:

1. Software fault causing loss of both subsystems (SYSTEM, case 1). This is a complete CCF covering all platforms that have the same system software. 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 leading to a crash of the processing units, i.e., no output signals coming from the processors. 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., no output signals coming from the processors within this subsystem. 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.

   The difference is that in case of fatal failure in DCS or DLC (case 2b), VUs run and can take default values. In case (2a), the whole subsystem stops running. For each case, a generic

* 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.

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 (case 3a) or VUs (3b) in one subsystem. This is a fatal fault causing loss of all functions of corresponding units. The fault can be in FRS or AS.

   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 (Boolean output is assumed). The fault can be in the APUs (case 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.

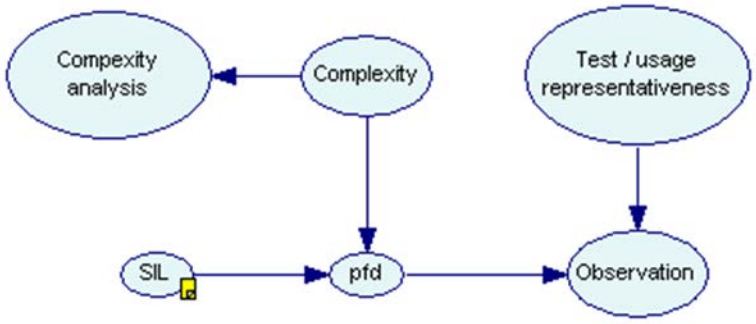
3. LIST OF EVIDENCE

In this chapter the relevant pieces of evidence considered for the quantification of software failure probability are identified.

3.1. Description of the relevant evidence

Figure 2 illustrates a Bayesian Belief Net (BBN) for quantification of software reliability proposed in [5]. 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.

Figure 2. 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 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 judgments. 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 under predict an already known operational failure rate.

3.2. Discussion on V&V procedure and 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 [6]. 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 V&V 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 V&V 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.

The basic idea in this paper is that the main process quality indicator that can be used to assess the quality of the software is the safety class or SIL class of the system.

3.3. Discussion on complexity of software

Concerning the reliability of software, one of the most important properties is its complexity. Complexity of a system is a parameter that is dependent on the size of the system, how many inputs 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 V&V is applied.

There is no widely accepted general method to calculate software complexity. The general idea in this work is to adapt existent methods, e.g. the TOPAAS approach [7] or the ISTec approach [8] as an input to the calculation, or use a simplified method to estimate the complexity of the software.

3.4. Discussion on software user experience

The discussion on user experience is based on the TELEPERM® XS (TXS) system platform developed at AREVA GmbH. 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 [9]). 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 [10].

For each of these non-conformances, it was analyzed 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 2a).
- *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) may happen. 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 and DPS). For this reason this CCF affects the APs and VUs within one subsystem (case 2b).
- *Same signal trajectory:* this group encompasses all CCFs 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 TXS system software to protect the processor against software exceptions from the application software processing. Exceptions can be captured by pre-checking algorithms, which detect the exception and handle it by substituting suitable values for computation if necessary. If the handling of the exception fails (system software failure due to e.g., failure of pre-checking routines, failure of the 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 sets 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 (e.g., RPS or DPS, see Figure 1), such that this CCF trigger mechanism is restricted to affect at most all APUs within one subsystem (case 3).
    - 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).

The operating experience of the potential CCF causes addressed in [9] is summarized in Table 1. The CCF triggering mechanisms and the latent fault location, i.e., system software (SyS), application software (AS) or communication of processors (DLC) is also indicated. Note that all observed failures of the TXS platform correspond to single failures with no evidence of CCF events.

Table 1. Assessment of software CCF triggering mechanisms based on TXS operating experience

CCF triggering mechanism		Latent fault location			Fault effects (see Ch. 2.2)	Failures in operation	Accumulated operation time [h]	Failure rate [1/h]	Event duration [h]
		SyS	FRS/ AS	DL C					
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	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 [9], one dependent failure is assumed for the failure rate calculation.

(2) This value is estimated (not reported in [9]).

(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.

The triggering mechanisms “temporal effects”, “faulty telegrams” and “same signal trajectory” (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. The downtimes caused by the failures with CCF potential are very short (see Table 1). Taking diversity requirements between both subsystems into account no relevant/realistic CCF mode of the TXS software which causes the complete failure of the system (both subsystems) can be identified.

4. PROPOSED QUANTIFICATION METHOD

The proposed method to quantify the failure probability of software is presented in this chapter. Chapter 4.1 presents an introduction to the quantification method. The approaches considered for the estimation of system and application software failure probabilities can be found in Chapters 4.1 and 4.3, respectively.

4.1. Introduction to the quantification method

Different quantification methods are appropriate for different type of software modules. 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.

For application software (AS) 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). Analysis of faults in FRS are part of the analysis of faults in AS.

Fault in EF can in principle cause any of the end effects. 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.

4.2. System software

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, 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 in the near future, 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 judgment in PSA when no proper data is available.

For comparison purposes, and for the use within the DIGREL example PSA model, the tentative probabilities presented in the table below can be used:

Table 2. System software fault related basic events.

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

4.3. Application software

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

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

A non-fatal failure in one application software can produce an incorrect output (no actuation when demanded or spurious actuation). If there is a strict separation between the system and application software (such is the case for TXS), a non-fatal failure does not affect the other processes running in the same processor.

In the proposed quantification method, indirect evidence is applied for the failure probability estimates of application software modules using the metrics Complexity and V&V.

In the baseline risk assessment – the SIL is used as the estimator of the V&V process. The value 0 for the V&V denotes a very simple or non-existing V&V process not fulfilling any SIL class requirements. The complexity of a system is defined as high, medium or low.

Table 3 presents 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.

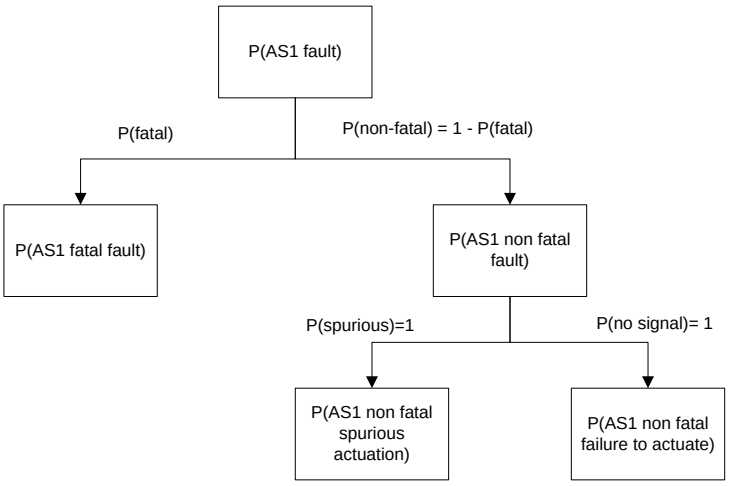
Table 3. Baseline failure probability estimates for application software modules.

		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 V&V 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. If the complexity of the software is medium or low, then the software failure probability should be lower than indicated by the SIL level. If a piece of software is of low complexity, but has the same type of validation as one of higher complexity, 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. The table is justified based on some available data (e.g. the data presented in section 3.4), and also justified with practice regarding used data in PSA. However, more operational history data would be welcome.

Figure 3 illustrates the process of calculating the failure probabilities for fatal and non-fatal probabilities.

Figure 3. Procedure to estimate the software fault probability in fatal and non-fatal failures (spurious and no signal scenarios).



An alternative approach to estimate fatal failure probability in application software could be to use operational history, if sufficient history is available. The method to be used for the fatal failure

probability estimation will be discussed further in the continued work. It shall also be noticed that which of the fatal and non-fatal failure modes that are relevant in the fault trees are also dependent on the system functionality (the safe state of the system).

5. CONCLUSION

This paper has outlined the background and the ideas behind a method for quantification of software faults to be included in a nuclear PSA context.

The paper has outlined which parts of the software should be represented in the PSA, which failure modes are relevant to consider and how to quantify the faults.

Fault modes associated to the system software is proposed to be modelled in the PSA with three types of basic events:

- Fatal CCF of the complete system software (failure probability: 1E-7),
- Fatal CCF of one subsystem (e.g., RPS or DPS, 1E-6) and
- Fatal fault of one DCU within one subsystem (1E-5).

The proposed failure probabilities of system software faults should be evaluated based on operating experience for the actual system/platform. The failure probabilities above are based on operating experience for the TXS platform. In the case of TXS, fatal failures of the system software are detected by the system, which sets the signal values into defined fail-safe values. For the estimation of the failure probability of application software a quantification method was proposed. This method considers the metrics complexity and V&V of the software as indirect evidences. Baseline failure probabilities for application software modules were estimated depending on the metrics. This is still an on-going research project and there are several items being investigated further. However, the outlined quantification method offers already a practical and justifiable approach to account for software failures that are usually ignored in current PSAs.

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 paper possible.

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] Porthin, M., Holmberg, J-E., Modelling software failures using Bayesian nets, VTT Research Report VTT-R-08279-12, 2013

[6] International Electrotechnical Commission, “Function Safety of Electrical/Electronic/Programmable Safety-Related Systems,” Parts 1-7, IEC 61508, various dates
[7] Rijkswaterstaat Ministerie van Verkeer en Waterstaat, TOPAAS: Een structurele aanpak voor faalkansanalyse van software intensieve systemen, 01.04.2011
[8] Komplexitätsmessung der Software Digitaler Leittechniksysteme, ISTec-A-1569, J. Märtz H. Miedl A. Lindner, Ch. Gerst, 2010
[9] 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
[10] AREVA GmbH, Managing operating experience with TXS. Report PTLS-G/2010/en/0046 Rev. A; 28.01.2008