MODIG – MODELLING THE RELIABILITY OF DIGITAL I&C IN MODERN NUCLEAR POWER PLANTS

Ola Bäckström[1], Jan-Erik Holmberg[2], Markus Porthin[3], Tero Tyrväinen[3]

[1] Lloyds Register: P.O Box 1288, Sundbyberg, Sweden, 17225, and ola.backstrom@lr.org
[2] Risk Pilot Ab: Metallimiehenkuja 10, Espoo, Finland, 02150, and jan-erik.holmberg@riskpilot.fi
[3] VTT Technical Research Centre of Finland Ltd.: P.O. Box 1000, Espoo, Finland, 02044, and tero.tyrvainen@vtt.fi and markus.porthin@vtt.fi

The Nordic R&D project MODIG (MODelling of DIGital I&C) aims to get a consensus approach for a reliability analysis of a plant design with digital I&C. The relevant part of the project for this paper is software failure probability quantification. To be able to define relevant software failure modes the I&C system needs to be split into a number of entities. The software entities are basically system software and application software. The system software can be further split into the run time environment and communication software. The failure modes applicable for each type of software differ. The approach to estimate the probability for various software failure modes is also discussed. System software failure probability estimate should be based on operational experience. Also the probability that an application software causes a fatal failure of the processor (crash) could be estimated based on operational experience. Non-fatal failures (functional failure without processor crash) for application software has to be treated differently, as sufficient operational data is not available. The non-fatal failure probability is suggested to be estimated based on an analytical approach using metrics of complexity and verification and validation..

I. INTRODUCTION

The fundamental challenge in nuclear power plant (NPP) engineering is to ensure and demonstrate the safety of the NPP, which is a complex sociotechnical system, by considering all types of hazards including rare and extreme conditions, all disciplines and all types of system elements and all life-cycle phases and activities. Deterministic and probabilistic analysis methods must be applied to show the compliance with regulatory requirements.

A digital I&C reactor protection system forms the center of intelligence in a modern nuclear power plant. A failure in the hardware or software has a potential to propagate through the reactor protection system and affect the overall safety of the station. The design of the system needs to meet deterministic criteria. Due to the complexity of the system the use of probabilistic techniques could be also very relevant in the demonstration of the defence-in-depth (DiD) concept. The digital I&C system should be included in the PSA, as I&C has a potential to significantly affect the overall plant reliability. The use of the PSA model for both DiD evaluation and PSA requires a proper definition and handling of the software system failure modes.

This paper will continue the discussion of the software failure modes and their quantification developed within the Nordic R&D project DIGREL. The NKS project DIGREL (2010–14) developed guidelines for analysis and modelling of digital systems in probabilistic safety assessment (PSA) for nuclear power plants. The project consisted of three interrelated activities. First, a taxonomy for failure modes of digital I&C systems was developed by a task group of OECD/NEA Working Group RISK (Ref. 1). Second, a fictive digital I&C PSA-model was developed for the demonstration and testing of modelling approaches (Refs. 2 - 6). Third, a method was developed for the quantification of software reliability in the context of PSA (Refs. 7 and 8).

As a result of DIGREL, there is a good understanding of sufficient level of details for PSA modelling and an approximate idea of treatment of software failures. Based on conclusions from the DIGREL-project, issues left out of the scope and discussions with stakeholders, a number of relevant issues have been identified to be studied further in the MODIG (Modelling of Digital I&C) project. This paper provides interim results from the MODIG project with regard to SW quantification.

II. SOFTWARE RELIABILITY ANALYSIS

In the context of PSA for NPPs, there is an on-going discussion on how to treat software reliability in the quantification of reliability of systems important to safety. It is mostly agreed that software could and should be treated probabilistically (Refs 9 and 10) but the question is to agree on a feasible approach.

Software reliability estimation methods described in academic literature are not applied in real industrial PSAs for NPPs. Software failures are either omitted in PSA or modelled in a very simple way as common cause failures related to the application software (AS) of operating system (platform). It is difficult to find any basis for the numbers used except the reference to a standard statement that 1E-4 per demand is a lower limit to reliability claims, which limit is then categorically used as a screening value for software CCF.

A schematic overview of a digital I&C system is presented by Fig.1. The figure presents a RPS which contains two systems RPS-A and RPS-B. Each system contains Application Processing Units (APU) and Voting Units (VU). MU represents the processor unit for the operator. Communication between redundancies is indicated.

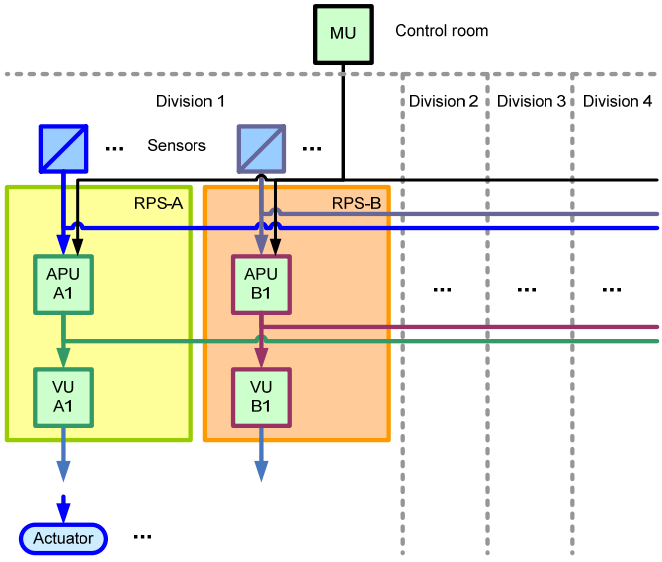

Fig. 1. An overview of a RPS system in a NPP.

The software quantification method is based on the defined cases from Ref. 8 and Ref. 1. The following software modules are considered:

- System software (SyS).
- Elementary functions (EFs).
- APU functional requirements specification modules (APU-FRS).
- APU application software modules (APU-AS).
- Proprietary software in I&C.
- VU functional requirements specification modules (VU-FRS).
- VU application software modules (VU-AS).
- Data communication software (DCS).
- Data link configuration (DLC).

The systems software (SyS) consists of the operating system and the runtime environment (interaction between application and operating system). The application software (AS) is the representation of the application functions in form of code. The application software is executed and controlled by the system software (run time environment) during an operating cycle. The elementary functions (EFs) are reusable, closed, and classifiable pieces of software, capable of processing signals, from which I&C functions can be assembled using function diagrams. The data communication software and the data link configuration implements the data communication. It is part of the platform software. The proprietary software is code that is embedded in specific hardware modules and its source code is generally not available for the end user.

In Ref. 1 a maximum extent of software failures is discussed. It is impractical to take all of the potential software failures into account in a PSA, and therefore a list of the most relevant faults and effects are considered further in MODIG. These are presented in TABLE I.

TABLE I. Faults and effects considered

Effects	Definition of effects	Software fault location					
		SyS	APU-FRS	APU-AS[1]	VU-FRS	VU-AS[1]	DCS
SYSTEM	Loss of complete system	case 1					case 1
1SS	Loss of one subsystem	case 2a	case 2a		case 2a	case 2a	case 2b
1APU-1SS	Loss of one group of redundant APU in one subsystem		case 3a	case 3a			
1VU-1SS	Loss of one group of redundant voters in one subsystem				case 3b	case 3b	
1AF-1SS	Loss of one function in all divisions of one subsystem		case 4a	case 4a	case 4b	case 4b	
1AF-1D-1SS	Loss of one function in one division of one subsystem		case 4c	case 4c			

The failure modes for software failures can be divided into fatal and non-fatal failures. The fatal failures will stop all on-going processes running on the processor/system. The non-fatal failures will only affect the output of the current process. Fatal and non-fatal failures may have the same impact on the signal generated; this is simply a matter of configuration and definition of end-states.

Cases 1, 2a and 2b are system failure modes, representing fatal failures. Case 2a means loss of one subsystem (represented by a fatal failure of APUs and VUs) and case 2b means loss of communication within a subsystem. The difference between 2a and 2b 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.

Cases 3 and 4 are application module failure modes and can be fatal or non-fatal failures. A fatal failure is a failure where the process stalls, which means that it will be possible for an external observer to know that the system has stopped (Case 3). A non-fatal failure does not stop the process but can yield an incorrect set of signals (Case 4).

The quantification method depends on the type of software module. System software (types 1 and 2 in TABLE I) and application software modules (types 3 and 4 in TABLE I) are considered relevant to model and quantify in PSA.

Fault in EF can in principle cause any end effect. The case “fatal failures affecting redundant units” is covered by the SyS fault. Non-fatal failures are covered by corresponding AS-fault. The most likely fault is not EF fault itself but that the EF is used in a wrong way in the AS. 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, and an explicit modelling of the proprietary SW module can be omitted. Faults in DCS and DLC may require some special treatment, due to possibly unique end effects, not necessarily covered by cases 1 and 2 (which is why DCS/DLC is treated with case 2b).

CCF between identical software modules in redundant trains is a general assumption. Failures in the SyS software are assumed to apply to all trains, in which the SyS software is used (Case 1 or 2a). With regard to AS faults, the definition of fatal failures is assuming that all redundant processors are affected, which means that the conditional probability is 1. A redundant processor is defined as running the same type of application software as the other trains (there may be small deviations in the running AS, but the majority should be the same). Non-fatal failures are considered to be completely dependent for each similar AS module. The CCF assumptions are considered conservative, but there are very scarce data and since the modules are operating under the same conditions with same signal trajectories then it would require justification to claim that they are not affected by CCF.

It is important to recognize that for the quantification of software failures, it may be necessary to differentiate between failures detected during operation and failures occurring during a transient. As this analysis is primarily focusing probability per demand (following initiating event); to be able to use operational experience from full power operation some supporting arguments should be provided.

1 Note that the APU-AS and VU-AS software modules consider the elementary functions (function blocks) involved in the application functions implemented in the APU/VU.

III. SYSTEM SOFTWARE (SyS) QUANTIFICATION

The failures of SyS should preferably be estimated for the system in question from operational history, since it is practically impossible and not meaningful to analyse system software more in detail (it is a "black box"). In addition, SyS is a vendor generic product (not plant specific), which can be taken as a justifying argument to utilize evidence from several NPPs.

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

For SyS type of failures, it can be assumed that the failures could happen regardless if the plant is in normal operation or a transient is occurring. The run time environment and messages sent between units will still function in the same way as during normal operation. Failures of the SyS will be discovered, since they should lead to a fatal failure.

From the TXS experience, following estimates of failure probabilities (per demand) have been done for TXS (Ref. 8):

- P[System-SyS, case 1] ≈ 2E-9
- P[1 SubSystem, case 2a] ≈ 2E-6
- P[1 SubSystem, case 2b] ≈ 1E-5.

The failure probability is calculated based on an estimate of the failure rate and an exposure time using a one-staged Bayesian model:

$$\lambda = \frac{2n + 1}{2T}$$

This calculation yields an estimate per hour, which is then multiplied with the studied transient time (for example 24 hours, as in the estimate above).

IV. APPLICATION SOFTWARE (AS) QUANTIFICATION

Whilst the system failures can be estimated using operation experience, this is not the case for the application software. The approach for application software quantification therefore needs to be treated separately. A fundamental question with regard to application software quantification is to what extent information from other software can be used to predict the behaviour of a specific software, as the different software may be doing vastly different tasks.

As the source of non-fatal failures is considered to be the application software failures, it is important to correctly treat dependencies between application software.

IV.A. Application Software Failure Modes

The failure modes for application software can be divided into fatal and non-fatal failures (case 3 and case 4 in TABLE I). The fatal failures in application software will stop all ongoing processes and thereby also affect the other application software running on the processor. The non-fatal failures will only affect the output of the corresponding application software.

As the fatal failures will affect all ongoing processes, these types of failures should be handled by the process and the error treatment. As the error treatment needs to be comprehensive in a reactor protection system, the fraction of failures leading to exceptions is expected to be small. Because of this the fatal failures' fraction of application software failures was estimated to be below 5% in Ref. 8.

IV.A.1. Application Software Modules

The non-fatal failure modes for the application software require a higher modelling resolution, and the method splits the application software into application software modules. An application software module (AS module) is a piece of software that is representing a specific functionality.

Each AS module usually corresponds to one individual function diagram group dedicated to a specific task. Depending on the specific case the application software can be represented by one or more AS modules. An AS module shall be defined so that one AS module is only used as a single entity in the analysis.

In Fig. 2 an AS is presented, with two possible ways to define the AS module. As the output from this software is through the same interface, there is no reason to split the AS module into more than one AS module, so a proper definition would be according to the solid red line in this example.

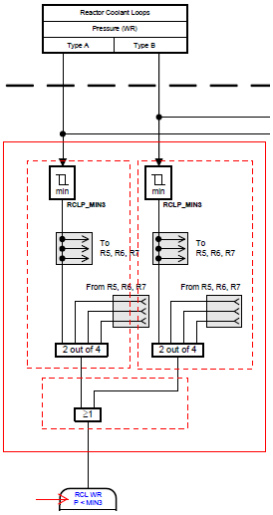

Fig. 2. Illustration of the definition of application software modules.

IV.B. Application Software Quantification Method – Fatal Failures

Fatal failures in any application software should be treated by the error handling system to avoid a fatal failure that would affect other ongoing processes. The error handling has to handle some types of faults that could occur in the application software, but these potential faults should be recurring in all processes. Hence, it should be reasonable to claim that these failures can be considered to be of same type and therefore analysed together.

Optimally, the estimate should be based on failures during demands in addition to information from failures during normal operation. There is more operational experience during operation, but it could be claimed that this operational experience does not cover potential failures at demands. The amount of demands is however not easily estimated. Assuming that the error handling will be challenged during operation in the same way as during a demand, the addition of failure probability from demands will be negligible compared to the probability of failure during operation. This assumption should be further justified. The fatal failures are hence estimated using the operational experience for processors.

Using the data for TXS (Ref. 11) there is experience from 44 million processor hours and no fatal failure has been observed. Since we are assuming CCF between all redundancies, only experience from one train can be claimed. A one stage Bayesian approach (see section III) yields an estimate of 1.1E-8 failures/hour and processor. Considering 24 hours mission time, the fatal failure probability per processor (CCF is assumed between redundant units) is estimated to 3E-7 per demand for redundant units.

IV.C. Application Software Quantification Method – Non Fatal Failures

The estimation of non-fatal failure probability is challenging, since this type of failure is not automatically detected by the system and may be very hard for an observer to identify. The probability estimation for non-fatal failures has to therefore be based on engineering judgements. There is simply insufficient operational experience to make good claims for non-fatal failures.

The potential impacts of non-fatal failures are also more challenging to estimate. To make it possible to estimate the impact of an AS failure AS-modules are defined, see section IV.A.1.

The approach is based on a Bayesian Belief Network (BBN) approach, which is using a matrix of complexity and verification and validation to form the initial probability per demand failure estimate. The assumption is that an increased

level of V&V should decrease the failure probability, and a lower complexity should also lead to a lower probability of failure.

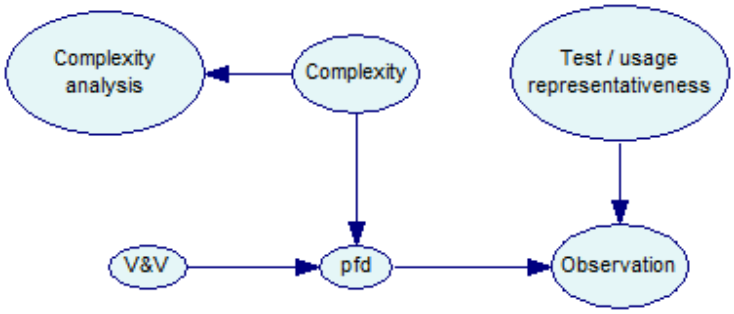

Fig. 3. A BBN for assessing software reliability using V&V class, software complexity and usage and test observations as evidence.

The estimate of the failure probability for an application software module is suggested to be based on:

$$E[P_{NSA}|F] = 1E\text{-}6 * F,$$

where F is a shaping factor. The shaping factor F is determined by Table III. Table II defines the V&V classification that is used in the method (used as input in Table III).

TABLE II. Verification and validation classification

V&V	Safety class in nuclear (Ref. 12)
0	*Non-nuclear safety*
1	C
2	B
3	A
4	-

If it can be claimed that the V&V level is greater than corresponding to safety class A in the nuclear domain, this can be represented by V&V level 4. It can be noticed that the considered factor in Table III is using the same factor (10) as between the different SIL levels in IEC61508 (Ref. 13).

The complexity is defined in High, Medium and Low – where the calibration is set so that a normal AS module for a NPP is Medium complexity. This is why factor 1 is used for Medium complexity and V&V level 3. The estimation of complexity is performed using a rather simplistic method, SICA. SICA is described in a separate paper at the PSAM conference. SICA is determining the complexity based on simple rules related to the number and complexity of function blocks, interconnections between function blocks, and the number of inputs and outputs.

TABLE III. Suggested shaping factor (F) values for different software categories

		Complexity		
		High	Medium	Low
V&V	0	10000	1000	100
	1	1000	100	10
	2	100	10	1
	3	10	1	0.1
	4	1	0.1	0.01

The distribution for the estimated failure probability is assumed to be a beta distribution Beta(α,β) with the above mean, and with an α of 0.5 and $\beta = \alpha(1 - \text{mean value}) / \text{mean value}$ to represent a wide distribution.

The non-fatal failure probability estimated using the above method is then split in spurious signal and no-signal failure probability respectively using the fraction 0.2 for spurious (active failure) and 0.8 for no signal (passive failure) (Ref. 8).

IV.C.1. Bayesian Update, Pooling of Data

The derived non-fatal failure probability can be updated using a Bayesian approach, should there be sufficient data available. In case a Bayesian update is performed, pooling of operational experience is a very important aspect.

When the data is collected and evaluated it can then either be pooled, i.e. several components are considered identical, or the data can be treated individually for each component. If the data is pooled, then the operational history is added up (sum of failures and hours) to one "super component". A condition for pooling is that the data is homogeneous, i.e. that the components are failure prune to the same extent. If homogeneity cannot be demonstrated then the data needs to be treated as separate pieces of evidence.

The collection and data analysis for software application modules have some challenges, of which following is of especial interest:

- Each software module is its own entity, and the software modules are therefore not (generally) exactly the same
- There are very few failures observed. Further, if a failure was observed, the software would be repaired (fault would be removed)

Despite the challenges, if a Bayesian update should be performed on the estimated failure probability (pfd), data need to be pooled (there will most likely not be enough data to modify the prior pfd without pooling). Pooling of software modules cannot be justified by statistical data with the scarce data available. If homogeneity is claimed it should be based on engineering judgements of the actual software modules and evidence for the statement should be discussed/presented.

IV.C.2. Common Cause Failure between AS Modules

In addition to the general assumptions on CCF (see section II), there could also potentially be CCF between AS modules (using same elementary functions, partly being the same, using almost identical inputs etc.). Generally, when data are pooled this should indicate that there is likelihood of a CCF. To account for these types of dependencies CCF between AS modules should be considered. A simple Beta-factor method is suggested.

The estimation of the Beta-factor is based upon the assumption that a CCF between software would be triggered by a number of inputs that are in states that are not properly treated. Therefore, the CCF would be triggered by identical inputs. Given this assumption, if there are no identical inputs, the Beta-factor would be 0 and if all inputs are the identical the Beta-factor could in worst situation be 1. The estimate of the Beta-factor could be based on some indicators (like in Human Reliability Analysis methods). In the rough method outlined, only two indicators are used for screening of the similarity:

- Amount of identical Inputs
- Level of similarity of the functions

If identical functions and inputs are used, the Beta-factor should be assumed to be 1.0. If the AS:s are similar, but not exactly the same, a rough estimate is proposed in Table IV. "Similar" is in the current method assumed as a screening rule based on that the logic is reasonably same. Formal rules for "similar" are yet to be developed.

TABLE IV. Rough estimation of Beta-factor when AS1 and AS2 is similar and not exactly the same

		AS2 complexity		
		High	Medium	Low
AS1 complexity	High	InputsIdentical/ InputsTotal	(InputsIdentical/ InputsTotal) /2	Beta = 0
	Medium		InputsIdentical/ InputsTotal	(InputsIdentical/ InputsTotal) /2
	Low			InputsIdentical/ InputsTotal

V. JUSTIFICATION FOR THE SOFTWARE QUANTIFICATION METHOD

A remaining task that is planned for the project in future is to further define the claims and evidence for the software approach used. The list of claims below is a draft list, which will need to be refined. The claims are defined on three levels: a) software system (SyS) claims, b) software failure mode (FM) claims and c) failure data (FD) claims.

The claims that need to be addressed are exemplified by:

- (SyS claim) Hardware and software failure modes can be treated separately.
- (SyS claim) The software subsystem can be divided into SyS module, DCS module and AS modules.
- (FM claim) The failure modes of the system can be described by Fatal and Non-Fatal failures.
- (FM claim) SyS software failures are fatal failures and non-fatal failures can be neglected.
- (FD claim) The failure probability for SyS can be estimated using operational experience gathered during operation.
- (FD claim) AS Non-fatal failures probability are dependent on complexity, level of verification and operational profile of software.

VI. FAULT TREE REPRESENTATION

The software failure modes that shall be represented for a specific signal to a certain component depends on the system configuration. Fig. 4 presents an example on the failure modes that could be relevant with regard to signals from application software AS1.

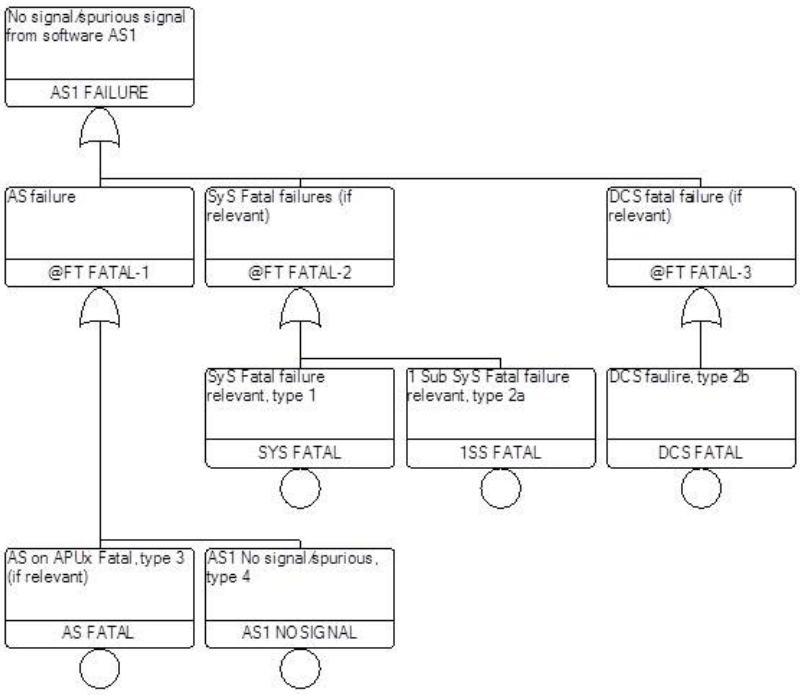

Fig. 4. An example of software failure modes.

To simplify the FT modelling, all failure modes are not necessarily represented for each application software specifically. Software failure modes that affects the function of the system studied may instead be modelled on a higher level.

VII. CONCLUSIONS

This paper presents an approach on how to analyze a digital I&C for a NPP. The approach is dividing the software system into subcomponents, which form their own failure modes in the PSA model.

For system software failures the reliability is estimated based on operational experience. The analysis of application software reliability is also based on operational experience with regard to fatal failures. Estimation of the probabilities for application software non-fatal failures can hardly be performed based on operational experience, as it is hard to collect this data on an appropriate level and even if you could — you would not have sufficient amount of operational experience. Therefore the method suggests an analytical approach using a metric of complexity and verification and validation. If operational experience data is available, the estimates can be updated in a Bayesian manner. Pooling of data is an open issue and the way it is done can significantly affect the results.

ACKNOWLEDGMENTS

The work has been financed by NKS (Nordic nuclear safety research) and SAFIR2018 (The Finnish Research Programme on Nuclear Power Plant Safety 2015–2018). 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. Software module diagram example was provided by Mariana Jockenhövel-Barttfeld and Andre Taurines from AREVA GmbH.

REFERENCES

1. OECD, “Failure modes taxonomy for reliability assessment of digital instrumentation and control systems for probabilistic risk analysis”, NEA/CSNI/R(2014)16, OECD/NEA/CSNI, Paris (2015)
2. 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)
3. 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)
4. 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)
5. AUTHÉN, S., HOLMBERG, J.-E., LANNER, L. & TYRVÄINEN, T., “Guidelines for reliability analysis of digital systems in PSA context — Phase 4 Status Report”. NKS-302, Nordic nuclear safety research (NKS), Roskilde (2014)
6. AUTHÉN, S., HOLMBERG, J.-E., TYRVÄINEN, T., ZAMANI, L., “Guidelines for reliability analysis of digital systems in PSA context – Final Report”, NKS-330, Nordic nuclear safety research (NKS), Roskilde (2015)
7. BÄCKSTRÖM, O., HOLMBERG, J.-E., JOCKENHÖVEL-BARTTFELD, M., PORTHIN, M. & TAURINES, A., ”Software reliability analysis for PSA”. NKS-304, Nordic nuclear safety research (NKS), Roskilde. (2014)
8. BÄCKSTRÖM, O., HOLMBERG, J.-E., JOCKENHÖVEL-BARTTFELD, M., PORTHIN, M., TAURINES, A. & TYRVÄINEN, T., “Software reliability analysis for PSA — Final report”. NKS-341, Nordic nuclear safety research (NKS), Roskilde. (2015)
9. DAHLL, G., LIWÅNG, B. & PULKKINEN, U., “Software-Based System Reliability”, Technical Note, NEA/SEN/SIN/WGRISK(2007)1, Working Group on Risk Assessment (WGRISK) of the Nuclear Energy Agency, Paris (2007)
10. 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)
11. JOCKENHOEVEL-BARTTFELD, M., “Estimations of baseline probabilities of application software failures”, AREVA, D02-IN-PEPS-G-14-0007 rev B. (2014)
12. INTERNATIONAL ELECTRONIC COMMISSION., “Nuclear power plants — Instrumentation and control important to safety — Classification of instrumentation and control functions”, IEC 61226, ed. 3.0, International electrotechnical commission, Geneva (2009)
13. INTERNATIONAL ELECTRONIC COMMISSION, “Functional safety of electrical/electronic/programmable electronic safety-related systems (E/E/PE, or E/E/PES), IEC 61508, ed. 2.0,” Geneva, Switzerland (2010)