Benchmark on dependability analysis of an Electrical power source.

Based on PSA 2019, paper "Dynamic Probabilistic Risk Assessment with PyCATSHOO: The Case of the Emergency Power Supply of a Nuclear Power Plant"

authors: Keoni Sanny[1], Claudia Picoco[1,2], and Tunc Aldemir[1]
presented by **Valentin Rychkov** [2]

[1]The Ohio State University
[2]Electricite de France

THE OHIO STATE UNIVERSITY

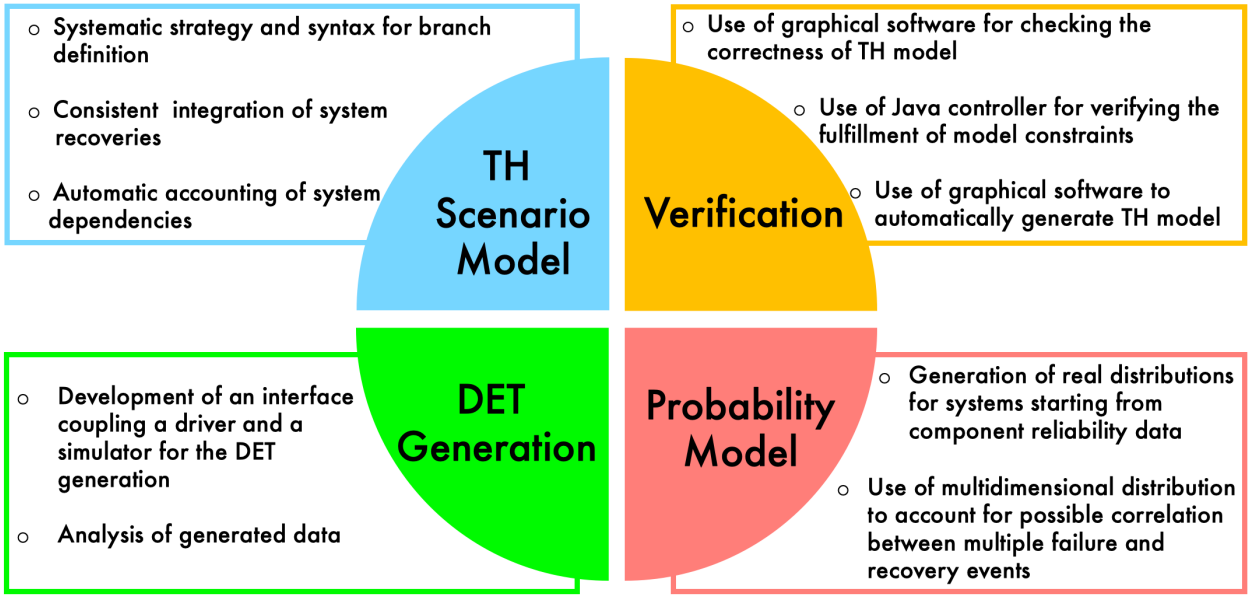
Dependability Benchmark - June 14, 2019

Outline

Outline

Context of the presented work

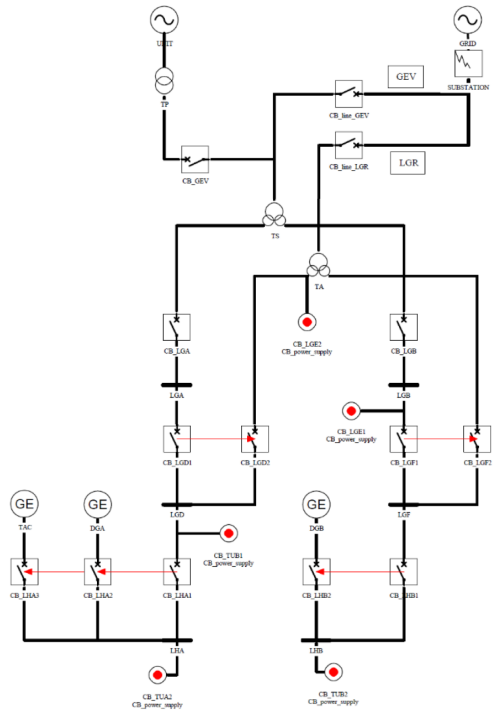
- This work is based on a paper presented at Probabilistic Safety Analysis Conference in Charleston (28.04-4.05 2019)
- This project was not to be meant a part the benchmark, but it is a part of the dissertation (by Claudia Picoco defended at Ohio State University on 04.04.2019): "Integrated Framework for Representing Recoveries Using the Dynamic Event Tree Approach"



The goal of the project

The goal of the project was to show that it is possible to extract the failure/recovery time distribution for a fairly complex system and inject it further into the Dynamic Event Tree analysis of the consequences of a LOOP-SBO event with possible recoveries.

- Includes repairable components, failures in function, failures on demand, cold redundancies, CCFs, reconfigurations, etc.
- Aims at supplying power to two busbars LHA and LHB,
- Has two main sources the GRID and the PLANT + backup sources including two diesel generators (DGA and DGB) and a gas turbine.



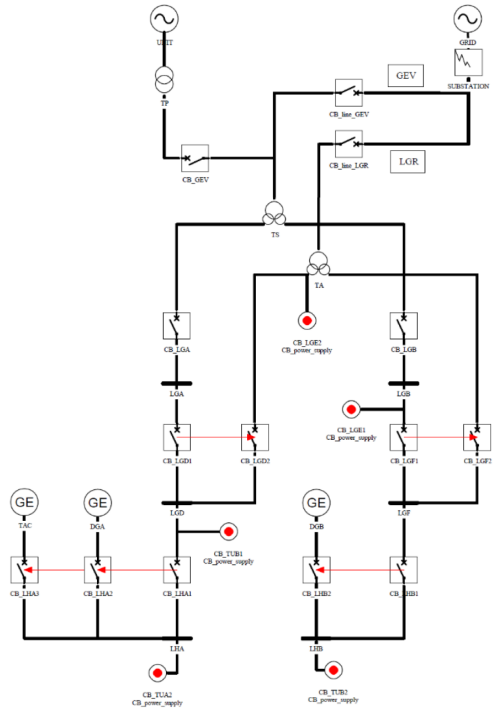
Outline

The Benchmark on EPSS

Proposed in: **M. BOUISSOU, A Benchmark on Reliability of Complex Discrete Systems: Emergency Power Supply of a Nuclear Power Plant, MARS 2017, Uppsala, 2017, 244, 200-216, Electronic Proceedings in Theoretical Computer Science (2017).**

Benchmark Problem

The objective of the benchmark is to compute the unreliability and the unavailability of the system for a mission time of 10000 h. Undesirable event to be quantified is the loss of power on both LHA and LHB.



Outline

PyCATSHOO Tool (1/2)

PyCATSHOO (PythoniC Object Oriented Hybrid Stochastic AuTomata) is a Python interfaced, object oriented tool to model a system dynamics using distributed, hybrid, stochastic automata:

- Developed by Électricité de France R&D [1] for dependability analysis of hybrid systems.
- Freeware available on http://pycatshoo.org
- Allows to integrate differental equations modeling physics.
- Allows modeling of:
  - Continuous, deterministic evolution,
  - Purely discrete events using discrete stochastic automata.
- Automata allows the system to be defined in terms of states and transitions among states

[1] H. CHRAIBI, J. C. HOUBEDINE, and A. SIBLER, PyCATSHOO: Toward a new platform dedicated to dynamic reliability assessments of hybrid systems, PSAM 13, Seoul (2016).

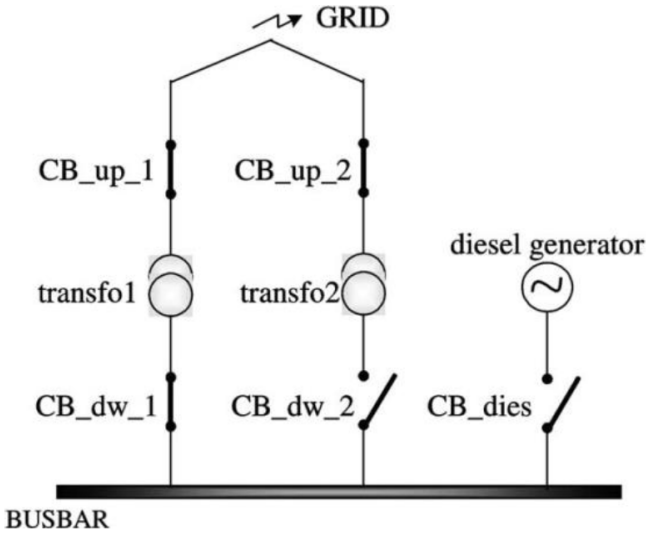
PyCATSHOO Tool (2/2)

PyCATSHOO modeling approach:

- System is divided into subsystems/components.
- Each subsystem/component is described as a set of automata, state variables and message boxes.
- System behavior is simulated using MonteCarlo sampling.
- Sequences that lead to desirable end states are traced and clustered.

Outline

Case Study 1: System & Scenario [2]



- The goal of the simple system is to supply electricity to a BUSBAR
- Electricity can be supplied to the *BUSBAR* either from the GRID or from a backup diesel generator.
- Short circuit propagation can disable the *GRID*. For example, if *transfo2* experiences a short circuit and *CB_up_2* fails to open.
- All components are repairable.
- For feeding the BUSBAR, the system prioritizes Line 1 over Line 2 and Line 2 over the diesel generator after every repair.

[2] M. BOUISSOU and J. BON, A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes, Reliab. Engng & System Safety, 82, 149-163 (2003).

PyCATSHOO Modeling and Data

Each component is defined by:

- A failure rate λ: average number of failures per hour.
- On-demand failure γ: the average number of failures per attempt.
- A recovery rate μ: average number of repairs per hour.

$$\begin{array}{|ll|}\hline \lambda = & 10^{-4}\ \textit{failures/hour} \\ \gamma = & 10^{-3}\ \textit{failures/attempt} \\ \mu = & 10^{-1}\ \textit{repairs/hour} \\ \hline \end{array}$$

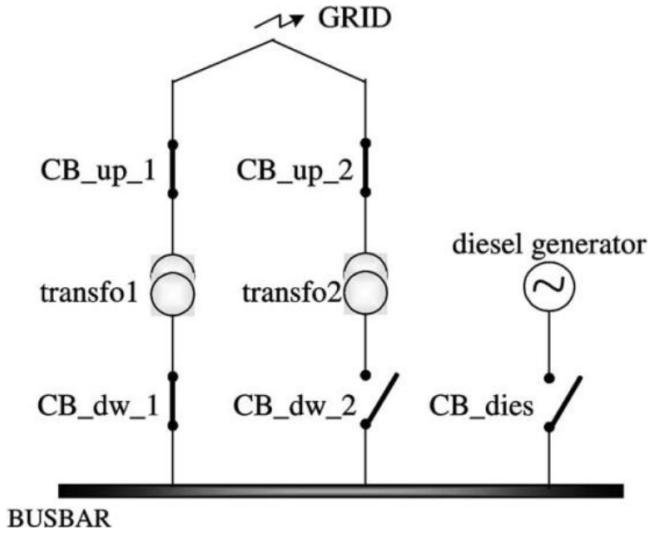
For in-operation failure and recovery, an exponential distribution has been assumed with parameters λ or μ for failure and repair, respectively.

Other assumptions

- All components can experience failure except the *BUSBAR* itself.
- The *GRID* is modeled as a single component and is also allowed to fail.
- Not every circuit breaker opening or closing is subject to failure on demand in the model.
- A line on which a failure has occurred is powered down so that other components on that line generally do not fail. For example, if *transfo1* has experienced a short circuit, *CB_dw_1* is not allowed to have a short circuit until Line 1 is back in operation.

Results

- The mission time is set to 10^4 h,
- The number of simulations is arbitrarily set to one million.



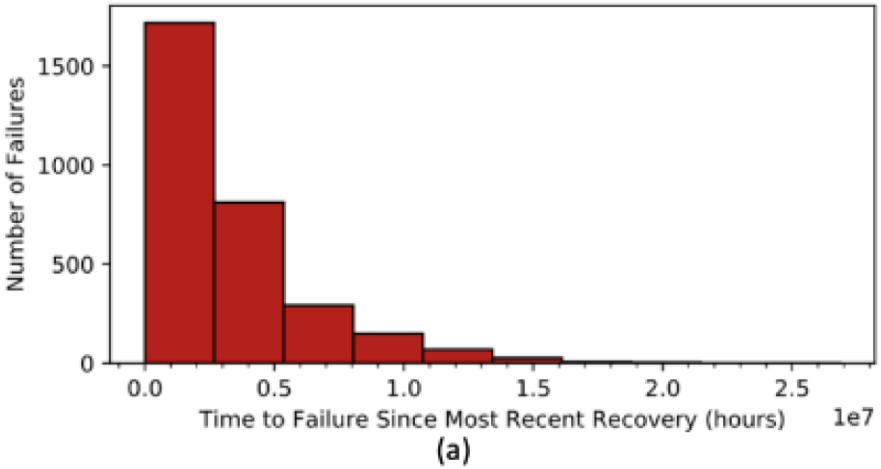
Number of Occurrences	Average failure time [h]	Prob.	Sequence
1016	4.9	0.33	Grid Failure, DG Fails to Start
1006	4.97	0.33	Grid Failure, CB Fails to Close
998	4.67	0.32	Grid Failure, DG Fails in Operation

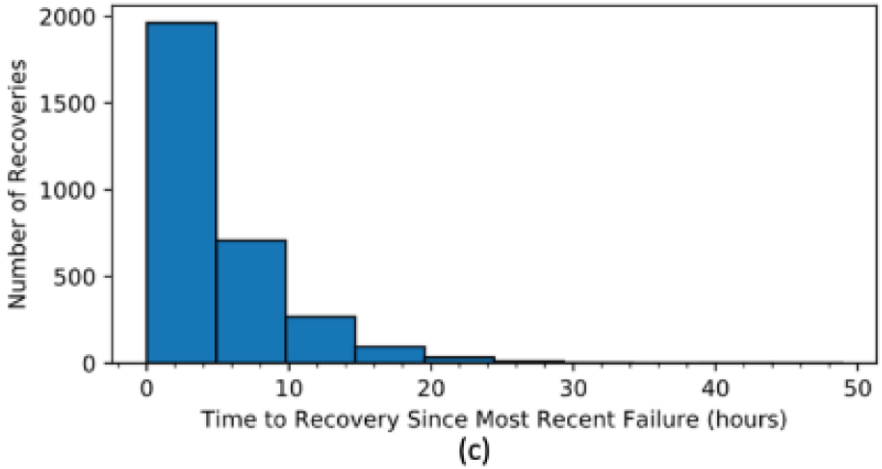
Post-Processing Analysis

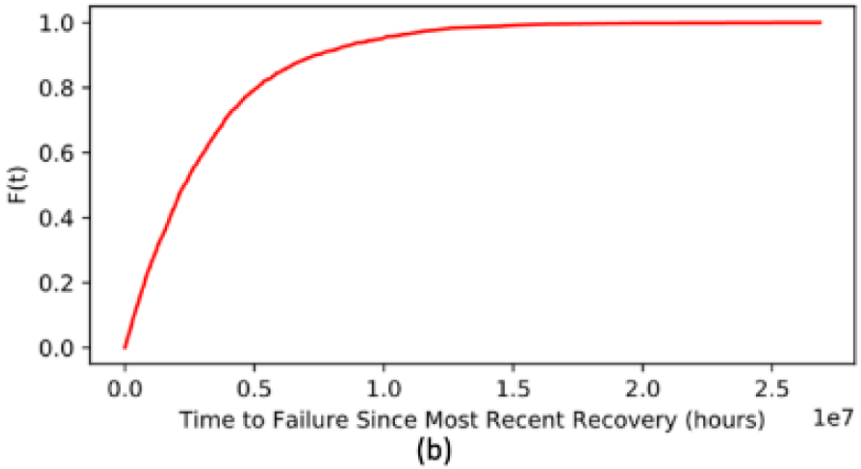
	PyCATSHOO	(Ref. *)
MTTF (h):	3.24e+06	3.29e+06
Unreliability	0.0031	0.0030
Unavailability	1.49e-06	1.51e-06

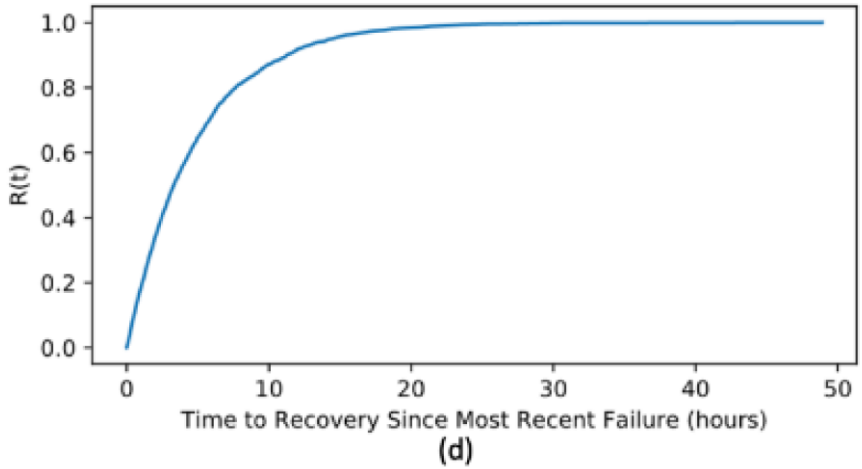
* M. BOUISSOU and J. BON, A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes, Reliab. Engng & System Safety, 82, 149-163 (2003).

Post-Processing Analysis

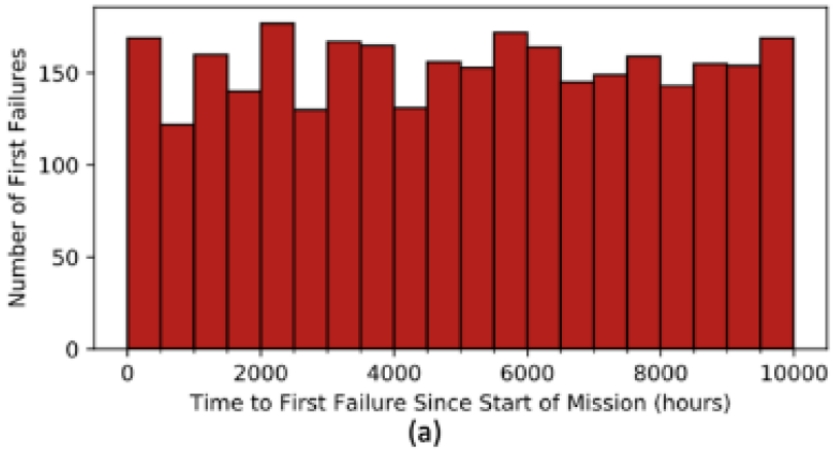






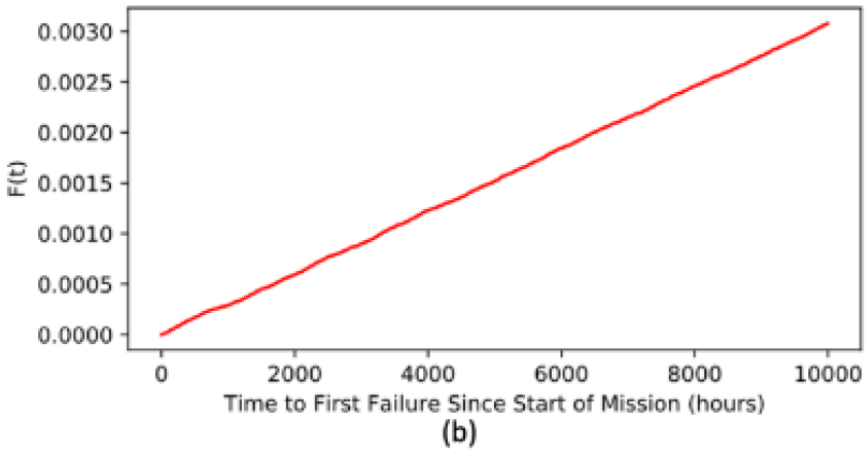


Post-Processing Analysis

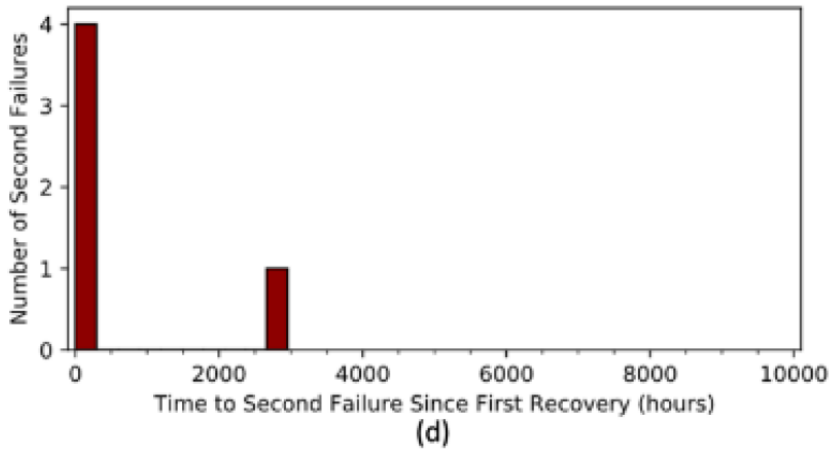


(a)

(c)



(b)



(d)

Model Architecture

Model size

- Generic part of the model: 600 lines (5 classes)
- System part of the model: 30 lines of the code

```
 10      ########################## Line 1 ###############
 11      class Line_1(Pyc.CComponent):...
167
168      ########################## Line 2 ###############
169      class Line_2(Pyc.CComponent):...
359
360      ######################### Diesel Generator Line #
361      class Line_D(Pyc.CComponent):...
485
486      ################################### Grid ######
487      class Grid(Pyc.CComponent):...
560
561      ################################### Busbar ###
562      class Bus(Pyc.CComponent):...
610
```

Model Architecture

Model size

- Generic part of the model: 600 lines (5 classes)
- System part of the model: 30 lines of the code

```
611  ######################## System Architecture ################################
612  # This class holds the composition and the architecture of the system
613  class MySystem(Pyc.CSystem):
614      def __init__(self, name):
615          Pyc.CSystem.__init__(self, name)
616
617          # Instantiation of all components
618          self.Grid = Grid("Grid")
619          self.Line_1 = Line_1("Line_1")
620          self.Line_2 = Line_2("Line_2")
621          self.Line_D = Line_D("Line_D")
622          self.Bus = Bus("Bus")
623
624          # Connect Message Boxes
625          # (Sending class, sending message box, Receiving class, receiving message box)
626          self.connect("Grid", "Grid-to-Line_1", "Line_1", "Line_1-from-Grid")
627          self.connect("Line_1", "Line_1-to-Grid", "Grid", "Grid-from-Line_1")
628          self.connect("Line_1", "Line_1-Bus", "Bus", "Bus-Line_1")
629          self.connect("Line_1", "Line_1-to-Line_2", "Line_2", "Line_2-from-Line_1")
630
631          self.connect("Grid", "Grid-to-Line_2", "Line_2", "Line_2-from-Grid")
632          self.connect("Line_2", "Line_2-to-Grid", "Grid", "Grid-from-Line_2")
633          self.connect("Line_2", "Line_2-Bus", "Bus", "Bus-Line_2")
634
635          self.connect("Grid", "Grid-to-Line_D", "Line_D", "Line_D-from-Grid")
636          self.connect("Line_1", "Line_1-to-Line_D", "Line_D", "Line_D-from-Line_1")
637          self.connect("Line_2", "Line_2-to-Line_D", "Line_D", "Line_D-from-Line_2")
638          self.connect("Line_D", "Line_D-Bus", "Bus", "Bus-Line_D")
639
640          self.connect("Grid", "Grid-Bus", "Bus", "Bus-Grid")
641
```

Model Architecture

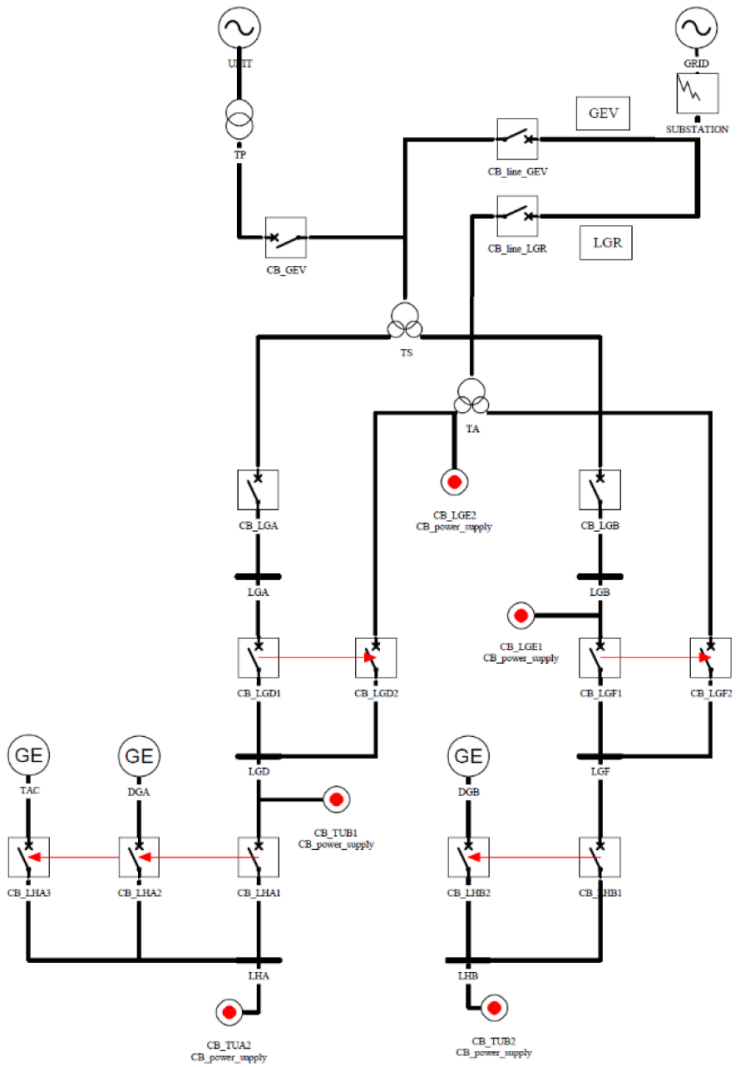
Verification: how to make sure that the model does what is supposed to do?

- Dynamic model $\leftrightarrow$ State Chart (generate a model from the code or create the code from the graphical model)
- Visual verification (simulator) or Automatic verification using test using YAKINDU tool by Itemis*
- Demo in YAKINDU

*On verification of dynamic models see C. Picoco et al, "Developing the control logic of thermal- hydraulic model for Dynamic Event Tree generation with RAVEN-MAAP5-EDF",(2018), BEPU 2018, or Claudia's PhD (on request)

Outline

Case Study 2: System & Scenario



- Goal of the EPSS: supply electricity to two busbar LHA and LHB that feed critical safety components
- There are five high voltage power sources: transmission network (*GRID*), the reactor itself (*UNIT*), two diesel generators (*DGA* and *DGB*), and a gas turbine (*TAC*).
- Low voltage part of the system is modeled as well including e backup batteries, AC/DC converters, etc.

PyCATSHOO Modeling and Data

Each component is defined by:

- A failure rate λ: average number of failures per hour.
- On-demand failure γ: the average number of failures per attempt.
- A recovery rate μ: average number of repairs per hour.

Data and parameters for this case have been taken from (Ref. *)

PyCATSHOO model consists:

- ▷ 24 classes,
- ▷ 69 automata (one for each component),
- ▷ 310 message boxes

---

* M. BOUISSOU, A Benchmark on Reliability of Complex Discrete Systems: Emergency Power Supply of a Nuclear Power Plant, MARS 2017, Uppsala, 2017, 244, 200-216, Electronic Proceedings in Theoretical Computer Science (2017).

PyCATSHOO Modeling and Data

Components part (3600 lines)

```
 9    ################################## Grid ########
10    class Line_G(Pyc.CComponent):...
99    ################################## Line GEV ######
100   class Line_GEV(Pyc.CComponent):...
231   ################################## Line LGR ######
232   class Line_LGR(Pyc.CComponent):...
383   ######################## Common Cause Failures ###
384   class CCF(Pyc.CComponent):...
524   ######################## Transformer TS ##########
525   class TS(Pyc.CComponent):...
650   ######################## Battery_B Line ##########
651   class Line_LBB(Pyc.CComponent):...
964   ######################## Battery_A Line ##########
965   class Line_LBA(Pyc.CComponent):...
1316  ######################## Transformer TA ##########
1317  class TA(Pyc.CComponent):...
1438  ######################## CB_LGD2 #################
1439  class CB_LGD2(Pyc.CComponent):...
1548  ######################## CB_LGF2 #################
1549  class CB_LGF2(Pyc.CComponent):...
1658  ######################## CB_LHA1 #################
1659  class CB_LHA1(Pyc.CComponent):...
1809  ######################## CB_LHB1 #################
1810  class CB_LHB1(Pyc.CComponent):...
1944  ######################## Line_TS #################
1945  class Line_TS(Pyc.CComponent):...
```

```
2058  ######################## Bus_LGD #################
2059  class Bus_LGD(Pyc.CComponent):...
2170  ######################## Bus_
2171  class Bus_LGF(Pyc.CComponent):...
2280  ######################## Bus_LHA #################
2281  class Bus_LHA(Pyc.CComponent):...
2389  ######################## Bus_LHB #################
2390  class Bus_LHB(Pyc.CComponent):...
2480  ######## Count System Failures and Recoveries ####
2481  class counter(Pyc.CComponent):...
2667  ######################## Line_TS #################
2668  class Line_TS1(Pyc.CComponent):...
2781  ######################## Reactor Line ############
2782  class Line_U(Pyc.CComponent):...
2933  ######################## Diesel Generator A Line ##
2934  class Line_DGA(Pyc.CComponent):...
3229  ######################## Gas Turbine Line ########
3230  class Line_TAC(Pyc.CComponent):...
3395  ######################## Diesel Generator B Line ##
3396  class Line_DGB(Pyc.CComponent):...
```

PEP 8: too many leading

System part (180 lines)

```
3601  ############################ System Architecture ################################
3602  # This class holds the composition and the architecture of the system
3603  class MySystem(Pyc.CSystem):...
```

PyCATSHOO Modeling and Data

The developer: an inexperienced person (a M.S. student) without knowledge of PyCATSHOO and Python

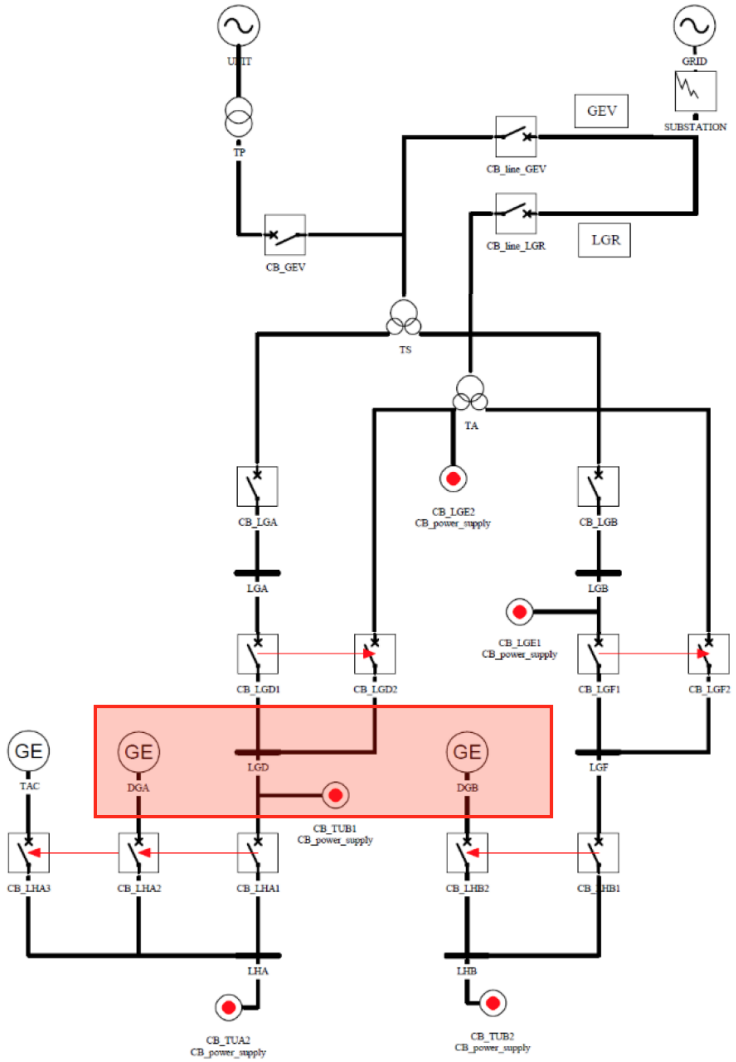
- A straightforward modeling approach without object oriented and functional programming tricks.
- The model is probably not reusable - small changes in the system model may require substantial changes in the code
- If you want to know more how design patterns influence the maintainability of the PyCATSHOO code, check the paper by I. Rychkova (Paris 1-Sorbonne)[3]

[3] I. Rychkova et al. "Measuring Maintainability of DPRA Models: A Pragmatic Approach." In ER Forum/Demos, pp. 58-71. 2017.

Other assumptions

- Each battery can supply power for exactly 1 hour before losing charge.
- Possible failure of the busbars is modeled with $\lambda = 2 \times 10^{-7}$
- House load functioning (i.e., reactor feeding just the systems within the plant, when the *GRID* or the connection to the *GRID* is lost) cannot be repaired until connection to the *GRID* is restored.
- Not all circuit breaker openings/closings are subject to failure in the model

CCFs

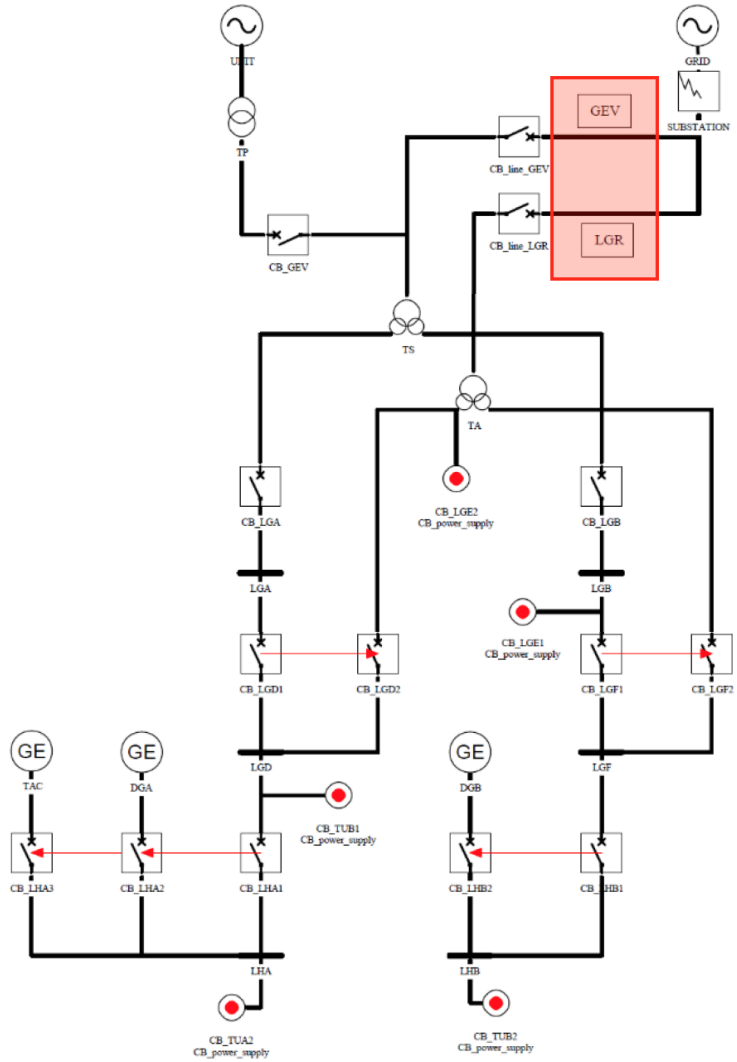


DGA and DGB can experience CCF :

- when one or both of them attempt to start (with $\gamma = 2 \times 10^{-4}$)
- when one or both of them are operating (with $\lambda = 5 \times 10^{-5} h^{-1}$)

CCFs

- Distribution lines GEV and LGR can experience a CCF in operation with (with $\lambda = 10^{-6} h^{-1}$)



Results

- The mission time is set to 10^4 h,
- The number of simulations is arbitrarily set to ten million.

Properties of two most common sequences:

- 40 Occurrences
- Average failure time is 105.16 hours
- P = 0.094
- Sequence
  - CCF GEV LGR
  - House load failure in operation
  - CCF DGA DGB
  - TAC failure in operation

- 31 Occurrences
- Average failure time is 9.96 hours
- P = 0.073
- Sequence
  - CCF GEV LGR
  - House load failure in operation
  - Diesel Generator A failure in operation (long),
  - TAC failure in operation,
  - Diesel Generator B failure in operation (short)

Post-Processing Analysis

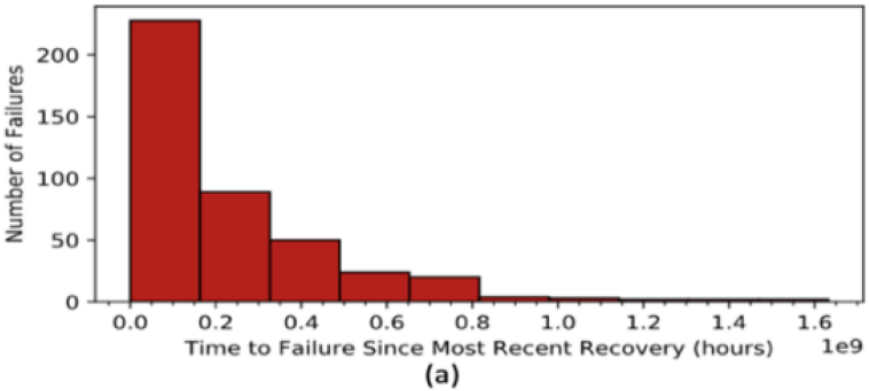
	PyCATSHOO
MTTF (h)	2.35e+08
Unreliability	3.86e-05
Unavailability	1.57e-07
Simulation time 10^7 sequences	7h with 2nd Core i7 (2012)

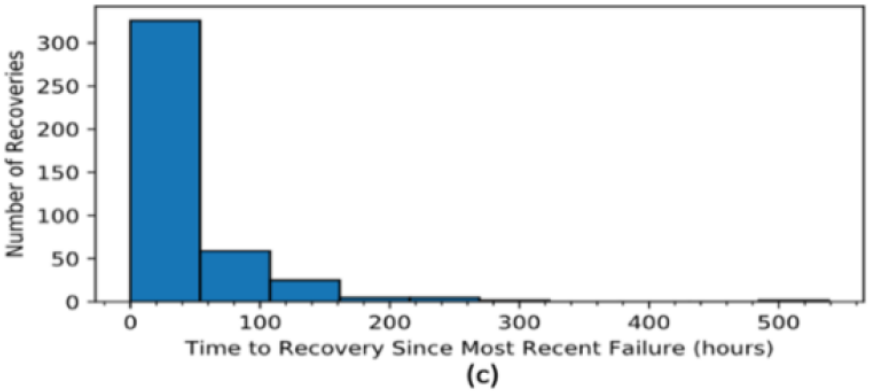
Table: Case Study 2 Results

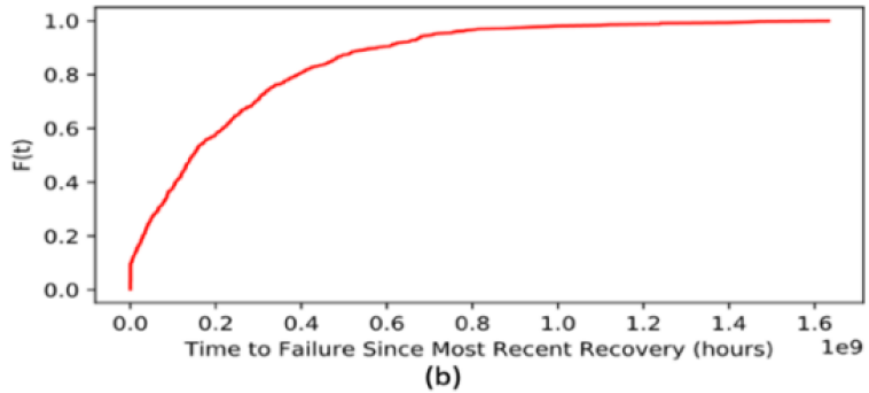
- (Ref. 1) founds unreliability of the system = 3.84e-05

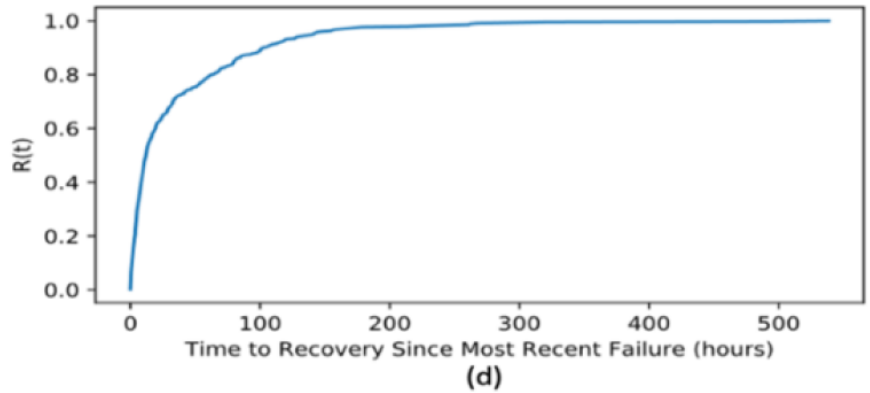
1. M. BOUISSOU, O. BCKSTRM, RORY GAMBLE, PAVEL KRCAL, and W. WANG, The IAB Quantification Method for Large Dynamic Systems in Practice: Two Use Cases.

Post-Processing Analysis

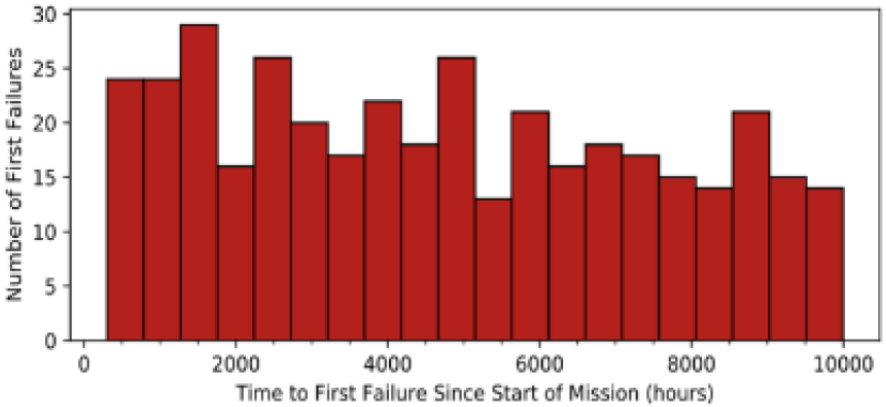


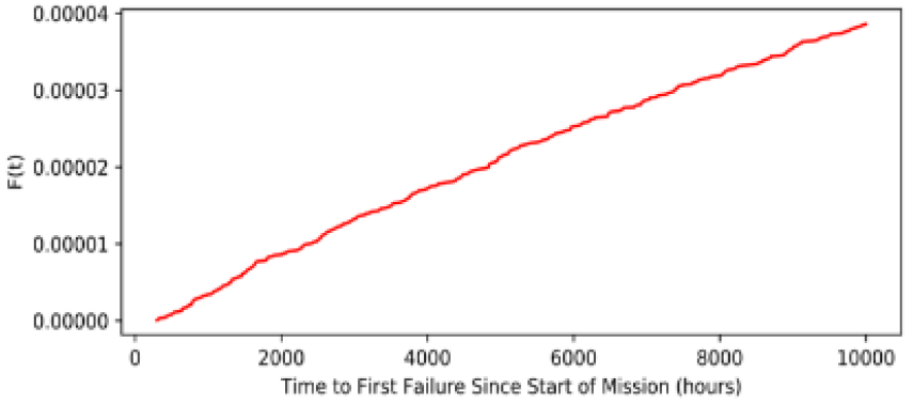




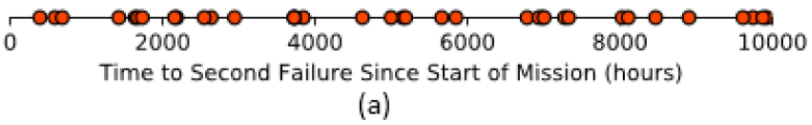


Post-Processing Analysis

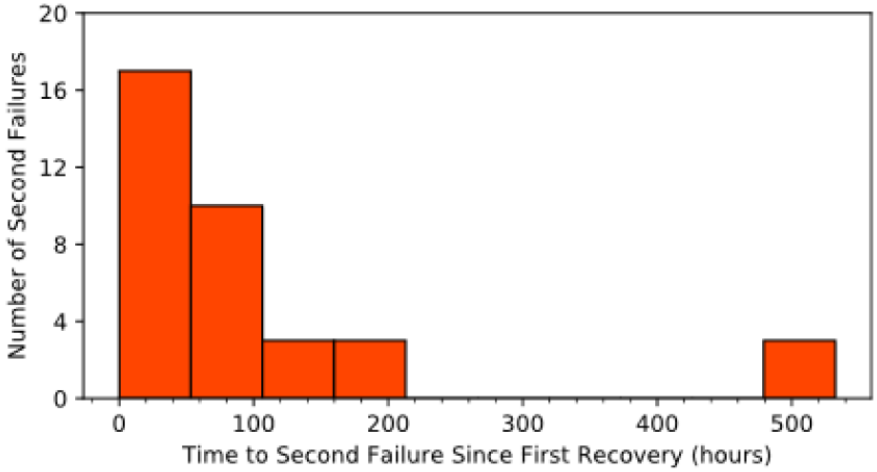




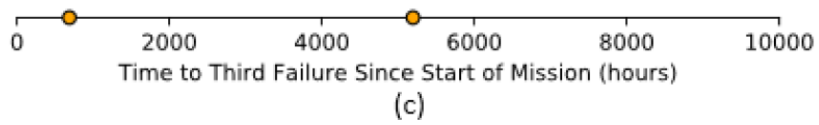
Post-Processing Analysis



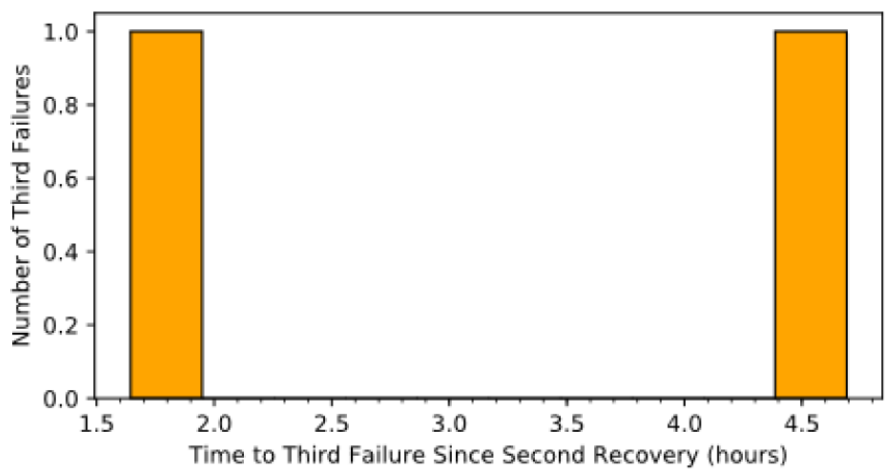
(a)



(b)



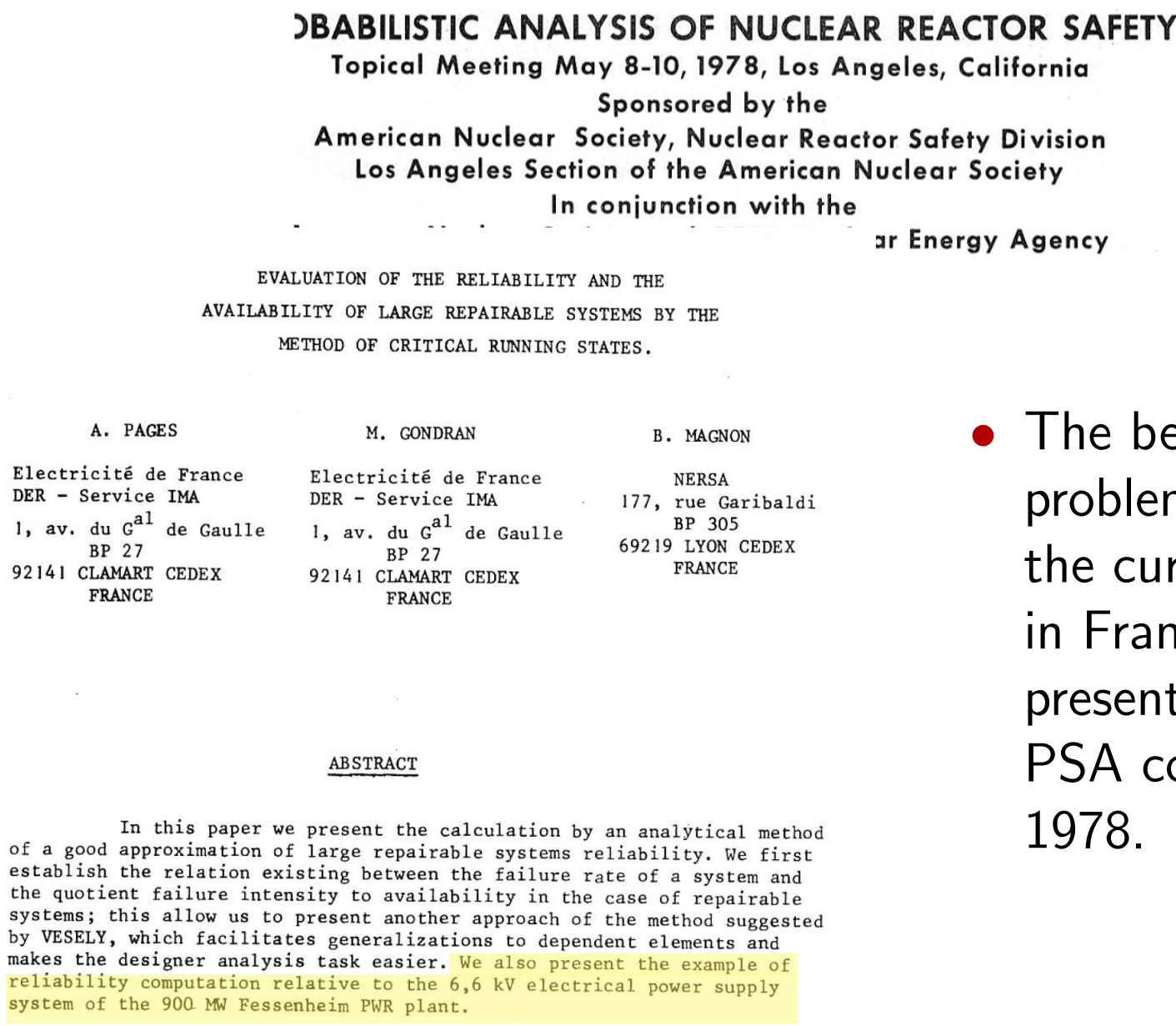
(c)



(d)

Outline

Knowing the roots



)BABILISTIC ANALYSIS OF NUCLEAR REACTOR SAFETY

Topical Meeting May 8-10, 1978, Los Angeles, California

Sponsored by the

American Nuclear Society, Nuclear Reactor Safety Division

Los Angeles Section of the American Nuclear Society

In conjunction with the

ar Energy Agency

EVALUATION OF THE RELIABILITY AND THE AVAILABILITY OF LARGE REPAIRABLE SYSTEMS BY THE METHOD OF CRITICAL RUNNING STATES.

A. PAGES

Electricité de France
DER - Service IMA
1, av. du G^{al} de Gaulle
BP 27
92141 CLAMART CEDEX
FRANCE

M. GONDRAN

Electricité de France
DER - Service IMA
1, av. du G^{al} de Gaulle
BP 27
92141 CLAMART CEDEX
FRANCE

B. MAGNON

NERSA
177, rue Garibaldi
BP 305
69219 LYON CEDEX
FRANCE

ABSTRACT

In this paper we present the calculation by an analytical method of a good approximation of large repairable systems reliability. We first establish the relation existing between the failure rate of a system and the quotient failure intensity to availability in the case of repairable systems; this allow us to present another approach of the method suggested by VESELY, which facilitates generalizations to dependent elements and makes the designer analysis task easier. We also present the example of reliability computation relative to the 6,6 kV electrical power supply system of the 900 MW Fessenheim PWR plant.

- The benchmark problem is as old as the current NPPs fleet in France, it was first presented by EDF at a PSA conference at 1978.

Knowing the roots

This study uses similar reliability data and finds (using semi-analytical methods) similar results:

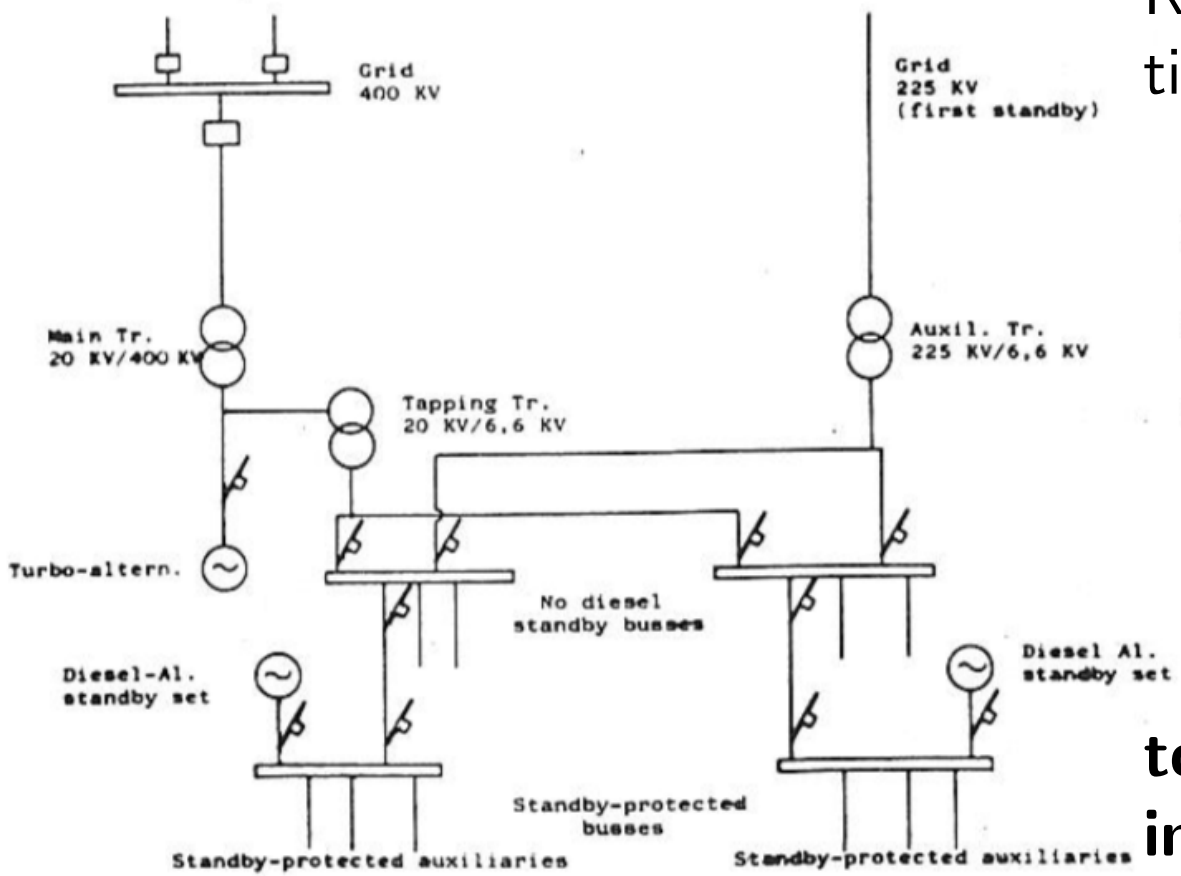


Figure 1 Schematic of power supply to auxiliaries in a French nuclear power plant 1976

Results for 8700h mission time (VL = Voltage loss),

$$\text{Proba. (VL)} = 2.7\ 10^{-5}$$

$$\text{Proba. (VL} > 1\text{ h)} = 2.2\ 10^{-5}$$

$$\text{Proba. (VL} > 20\text{ h)} = 1.3\ 10^{-5}$$

to compare with $3.8e^{-5}$ **in the benchmark.**

Conclusion

- This benchmark presentation is a by-product of a larger project.
- We used PyCATSHOO in the cheapest possible way to obtain the information we needed (the time distributions for Emergency Power Source failure/repair to inject in subsequent analysis).
- The overall project took 10 days for a M.S. student who didn't have a prior knowledge of neither PyCATSHOO nor Python (had only an experience with MATLAB)
- Model verification (in proper sense of this term) is missing, but possible. (see smaller case)
- Software design patterns[4] may be used for modeling languages like PyCATSHOO to optimize the model construction and maintenance.

[4] Wolfgang, P.: Design patterns for object-oriented software development. Reading, Mass.Addison-Wesley (1994)

Thank you for your attention!