Horizon 2020 Programme

METIS

Research and Innovation Action (RIA)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 945121

Start date : 2020-09-01 Duration : 48 Months

Enhanced version of the PSA calculation engine SCRAM

Authors : Mr. Charles DROSZCZ (GDS)

METIS - Contract Number: 945121

Project officer: Katerina PTACKOVA

Document title	Enhanced version of the PSA calculation engine SCRAM
Author(s)	Mr. Charles DROSZCZ
Number of pages	34
Document type	Deliverable
Work Package	WP7
Document number	D7.3
Issued by	GDS
Date of completion	2023-05-12 15:26:22
Dissemination level	Public

Summary

This report is focused on the presentation of the PSA tool SCRAM-Andromeda. This tool is developed for the METIS Project within Work Package 7 to perform the risk quantification. The main goal of WP7 is to provide an open-source PSA tool that can be freely used by the METIS community, and to address technical issues of seismic PSA.

Approval

Date	By
2023-05-12 16:07:45	Mr. Oleksandr SEVBO (Energorisk)
2023-05-12 16:26:07	Dr. Irmela ZENTNER (EDF)

METIS

Seismic Risk Assessment
for Nuclear Safety

Research & Innovation Action

NFRP-2019-2020

Seismic PSA Tool for the risk quantification - SCRAM - Andromeda

Deliverable D7.3

Version N°1

Authors: Charles DROSZCZ (GDS)

Disclaimer

The content of this deliverable reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

Document Information

Grant agreement	945121
Project title	Methods And Tools Innovations For Seismic Risk Assessment
Project acronym	METIS
Project coordinator	Dr. Irmela Zentner, EDF
Project duration	1st September 2020 – 31st August 2024 (48 months)
Related work package	WP 7
Related task(s)	Task 7.3 – Development of new assessment algorithms
Lead organisation	Oleksandr SEVBO (ER)
Contributing partner(s)	
Due date	
Submission date	
Dissemination level	Public

History

Version	Submitted by	Reviewed by	Date	Comments
N°1	Charles DROSZCZ (GDS)	Charisis CHATZIGOGOS (GDS) Pierre-Alain NAZÉ (GDS) Mohamed HIBTI (EDF) Saran Srikanth BODDA (NCSU) Abhinav GUPTA (NCSU) Oleksandr SEVBO (ER)	22/03/2023 22/03/2023 17/04/2023 29/04/2023 29/04/2023 07/05/2023	

Table of Contents

List of figures

List of tables

Abbreviations and Acronyms

Acronym	Description
BE	Basic Event
ET	Event Tree
FT	Fault Tree
MCS	Minimal Cut Sets
METIS	Methods And Tools Innovations for Seismic Risk Assessment
MSYS2	Software Distribution and Building Platform for Windows
OPSAMEF	Open-PSA Model Exchange Format
PSA	Probabilistic Safety Assessment
SCRAM	Command-line Risk Analysis Multi-tool
SPSA	Seismic Probabilistic Safety Assessment
WP	Work Package

Summary

This report is focused on the presentation of the PSA tool SCRAM-Andromeda. This tool is developed for the METIS Project within Work Package 7 to perform the risk quantification. The main goal of WP7 is to provide an open-source PSA tool that can be freely used by the METIS community, and to address technical issues of seismic PSA.

Keywords

Seismic risk quantification; system analysis; fault tree; event tree

Introduction

This report presents the work carried out within the framework of Task 7.3 of the METIS project which aims at providing a tool for calculating the seismic risk. The retained tool is based on the use of SCRAM [1] which performs risk quantification as part of systems analysis, and Andromeda [2] which is a PSA model management tool that can be used to implement, develop and modify a PSA model. This technical report includes an IT description of the current version of SCRAM tool and the work that has been performed for the compilation of SCRAM (developer level) and for its documentation (user level). A procedure for compiling SCRAM under Windows is provided, as well as the basic commands to use it. The integration of SCRAM in Andromeda as well as the use of SCRAM (in command line) or integrated into Andromeda are provided. An example of quantification is also provided.

1. PSA tool Andromeda-SCRAM

1.1. Context

Within the framework of European Project METIS, WP7 aims at developing an PSA tool and at addressing technical issues of seismic PSA by improving this code. This tool is based on the use of two pre-existing tools, modified and made compatible to one another within the framework of the METIS project. The first one is Andromeda Community (called Andromeda hereafter), which is a PSA model management tool. Andromeda allows manipulation of data used in a PSA (filter, sort, group), visualization (fault trees, event trees, lists of Basic events, parameters…), and can launch different solvers for quantification. The second tool is an open-source solver which will be available from now on via Andromeda and which is based on the open-source code SCRAM.

1.2. General description of Andromeda

Andromeda [2] is a contribution to PSA model management to enhance modern modeling principles and management for large, complex, and long-life PSA models. It is a way towards transparency and user-friendly practices for a wide panel of developers and users. The aim of Andromeda is to reinforce PSA analysis using state-of-the-art engineering tools belonging to the area of safety assessment software. Such tools require good quality assurance in addition to rigorous and effective ways of co-working in different engineering teams.

Based on the idea of a modular PSA [3], Andromeda was developed as a result of the PhD work of Thomas Friedlhuber in 2014 [4] under the supervision of Dr. Mohamed Hibti (EDF R&D) and Prof. Antoine Rauzy (Ecole Polytechnique).

Andromeda uses the Open-PSA Model Exchange Format (OPSAMEF) promoted by the Open PSA Initiative [5]. The OPSAMEF is an XML format [6] that defines an open and clear representation of PSA models [7]. This format is the one retained for the PSA tool developed in the METIS project [8].

Andromeda is based on the concept of Modular PSA in the sense that it copes with the challenge of increased complexity using the techniques of modularization and instantiation. Modularization targets to treat a model by smaller pieces (the "modules") to regain control over models. Instantiation aims to configure a generic model to different contexts. Both try to reduce model complexity [4].

In this perspective, Andromeda is constructed in a modular fashion with a central core and different modules each providing a different capability. Among the different modules, it is worth mentioning [4] (see Figure 1) the following:

- **PSA model editing**: it allows to edit a model by creating and modifying event trees, fault trees, parameters, etc.
- **Model comparison**: it allows to compare two models. For each type of object (basic event, event tree, fault tree, analysis case) Andromeda provides the detail of the difference in a graphical as well as in a textual form.
- **Model fusion**: it allows to merge two models after comparing them. Andromeda requires a model A, model B and a base model (that can be one of the two previous models). Andromeda can merge the model A with the model B. In case of conflict (different versions of the same object with respect to the base model), Andromeda asks the user to fix the conflict by selecting the option A or B to be considered into the merged model.
- **Version management**: it allows to use the git platform for the version management of the PSA model. The version management can be done either locally or from/to a remote repository. The version management allows to keep track of all the model modifications, have multiple

analysts working simultaneously on the same model (thanks to the branch concept), restore the previous correct version in case of error, *etc*.

- **Scripting**: Andromeda has a Jython API. This means that it provides the user with a set of functions that can be used into a script for model modification. The use of scripts is particularly recommended in case of massive and repetitive modifications.
- **Documentation**: for several elements of the model, Andromeda allows the user to fill up a wiki page. In this way the user can write the documentation at the same time as the model construction. This capability also helps to keep the documentation updated coherently with the model changes.
- **Dependency and Cartography**: this module provides the backward and forward dependency of each model element. For example, for a given basic event, the forward dependency provides the parameters that are associated to the basic event, while the backward dependency provides all the fault trees, function event, event trees, etc. that make use of the considered basic event. A cartography is also provided to show the dependencies in a more visual way, and may be used to split large study cases in a coherent manner to perform parallel computation if needed [7].

Within the context of the METIS project, Andromeda is coupled with the quantification software SCRAM. Thanks to this coupling, quantification of PSA models will be possible in Andromeda using an open-source code.

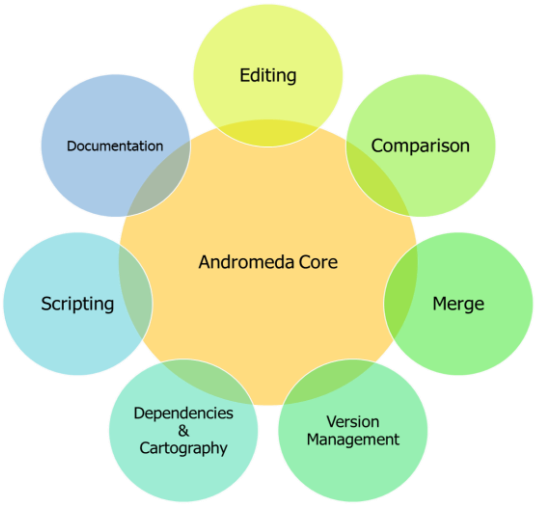


Figure 1: Andromeda's modules

1.3. General description of SCRAM

SCRAM is a Command-line Risk Analysis Multi-tool that can perform event tree analysis, static fault tree analysis, analysis with common cause failure models, probability calculations with importance analysis, uncertainty analysis with Monte Carlo simulations. The initial SCRAM project was developed by Olzhas Rakhimov from 2014 to 2018. The whole project is available on GitHub (https://github.com/rakhimov/scram) under the GNU General Public License v3.0. The project was developed in C++ to be available for all major OS (GNU/Linux, Mac OS X, Windows). SCRAM has been developed using Git as a source code management and versioning tool. The project is hosted on the GitHub platform and integrates automated testing tools, continuous integration and automated documentation generation tools.

The two classic types of PSA analyses that can be carried out with SCRAM are Event Tree analysis and Fault Tree analysis.

Event tree analysis is an analysis of sequences given an initiating event and progression of states over functional events. The analysis uses the Boolean formulae and expressions per sequence to supply arguments for other types of analysis, such as fault-tree, probability, and uncertainty analyses.

Fault trees use various types of gates (AND, OR, ...) and events to represent Boolean formulae and to model systems for analysis. Minimal Cut Sets are determined and the total probability of the top event is calculated given the failure probabilities of the Basic Events.

SCRAM uses the Open-PSA Model Exchange Format (OPSA-MEF) for coding its inputs (fault trees and event trees).

1.4. General IT description of SCRAM

In this section, we describe from an IT point of view the SCRAM project as it was initially, before its use within the METIS project (computer description, dependencies, branch analysis). We also detail the main changes made to make SCRAM compilable on Windows, and provide a compilation guide.

1.4.1. Analysis of the SCRAM project

The SCRAM project (Recursive acronym for SCRAM Command-line Risk Analysis Multi-tool) is an open-source project under GPL-3.0 license, developed by Olzhas Rakhimov. The sources of the project are accessible on Github at the following URL: https://github.com/rakhimov/scram. The last commit of the project dates from July 3, 2019, and the SSL certificate of the project website, scram-pra.org, has expired since September 2020. The continuous integration of the project is also in failed state for 2 years. The project is no longer maintained.

SCRAM is a free software – redistributable and modifiable under the terms of the GNU General Public License. This license implies a "contaminating" Copyleft license, which means that any product or project, derived from or using licensed code, must comply with and apply the even said license. The main principles of this license are as follows:

- Anyone is free to use the project for any purpose,
- Anyone has guaranteed access to the sources and is free to modify them for their own needs,
- Anyone is free to redistribute a copy of the project,
- Anyone is free to share their changes.

The SCRAM project is implemented in C++, with the C++17 standard, due to a recommendation from its optional TCMalloc dependency. The project uses the CMake meta-generator, generating the files needed for the build chain.

1.4.2. SCRAM project dependencies analysis

Hereafter is a brief description of the dependencies of the SCRAM project.

1.4.2.1. Qt & Qtango

Qt is a C++ development framework. It is a library enriching the standard functionalities of the languages, but more specifically used in the design of applications with graphical interfaces. In the

context of SCRAM, Qt is a dependency of Qtango, for the realization of the GUI module of the software. Using Qt for any proprietary project requires a commercial license. Qtango is an overlay of Qt.

1.4.2.2. TCMalloc/JEMalloc

TCMalloc and JEMalloc are libraries implementing memory allocators. The allocators are memory managers working during dynamic allocations. The objective is to optimize the use of system calls, limiting costly interactions with the operating system during manipulation of memory.

TCMalloc is a solution developed by Google, optimized and recognized for its performance in terms of CPU cycles per allocation. This optimization has the cost of less optimal recycling of the memory.

JEMalloc is an independent open-source project, standardized in the Unix operating system FreeBSD. It has the benefit of a small footprint in terms of memory consumption, with an optimized recycling. The manager tends to maintain a memory footprint as close as this which was provided by the developer.

1.4.2.3. LibXML2

Library integrated into the GNOME project, tending to be a standard in terms of manipulation of xml files. Due to its popularity, its use is widely democratized. We will therefore have the different classes representing each token of the XML format, a parser, a write driver, as well as a read driver.

1.4.2.4. Boost

Boost is a generalist library widely used in science, generally experimenting field for future standard features of C++.

1.4.3. Analysis of project branches

Due to the status of the project, only the "develop" branch is considered workable, containing the latest changes from the maintainer. For information, and due to various common configurations on the Github platform, we observe some usage of the developer:

- develop: main branch of the project,
- master: abandoned branch, source of development,
- gh-source: probable attempt to link to a Jenkins,
- gh-pages: source branch of Github Pages (documentation),
- coverity-scan: source branch for coverage test plugin,
- l10n_develop: added Czech language support in the GUI module,
- win32: configuration for continuous integration to produce a 32-bit binary artifact.

After some code modifications (see §1.4.4) and verification of the correct compilation, the "develop" branch (the most up-to-date) has been merged with the "master" branch (see §1.4.6).

1.4.4. First modifications of SCRAM

Within the METIS project and since the initial project was no longer maintained and the initial developer cannot be reached, the whole project has been cloned from the rakhimov/scram repository into the SCRAM-NG/scram repository: https://github.com/SCRAM-NG/scram.

Some modifications of the code were necessary to make it usable and compilable. The main modifications are the following;

- disabling the GUI build target since the GUI is not desired for the METIS project (ANDROEMDA is used as a Graphical User Interface):

  File CMakeLists.txt, option(BUILD_GUI "Build the GUI front-end" OFF)

- updating of the boost library (version 1.75) and the boost::throw_function function,
- disabling the use of TCMalloc and JEMalloc (does not work correctly with MSYS2 - the building platform for Windows, recommended for the project).
- automatic deployment of the dependant shared object (dll) files:

  File CMakeLists.txt, option(PACKAGE "Package for distribution" ON)

1.4.5. Compilation of SCRAM under Windows

Since the code is not maintained since 2019, intensive work has been done to get familiar with this code and being able to recompile it. The choice was made to recompile the code on Windows (the initial developer planned compilation on Linux as well), using the Integrated Development Environment Visual Studio Code and the software distribution and building platform for Windows MSYS2 (linux-like).

The Appendix in §6 describes the procedure for compiling SCRAM under Windows using Visual Studio Code and MSYS2. Main steps are the following:

- Installation and configuration of the Linux-friendly MSYS2 environment on Windows,
- Installation and configuration of Visual Studio Code,
- Clone the SCRAM repository from GitHub,
- Compilation of SCRAM on Visual Studio Code.

1.4.6. Release of a new version

The "develop" branch has been merged to the "master" branch after the above modifications and after successfully compiling the source code. The new compiled version of scram is 0.17.0.

2. Using SCRAM

Within the framework of the METIS project, SCRAM can be used in several ways:

- From a developer's point of view, the source files and the compilation procedures are available (source files available on GitHub, compilation procedure in §6). This makes it possible to envisage an in-depth modification of the SCRAM code,
- From a user's point of view, the compiled version of SCRAM is available (*cf.* §1.4.6, or see Releases on the GitHub page of the project). This compiled version is usable in two different ways:
  - In command line, without any graphical interface,
  - Integrated in Andromeda.

2.1. Quantification principle for a PSA study using SCRAM

Given an Initiating Event (seismic initiator for a Seismic PSA) with a frequency of occurrence f_{IE}, and given the failure probability of the safety systems (P_{SM} for Safety Mission) which aim at mitigating the occurrence of the Initiating Event, the frequency of the Unacceptable Consequence f_{UC} is calculated.

This can be summarized in the basic Event Tree in Figure 2 here after:

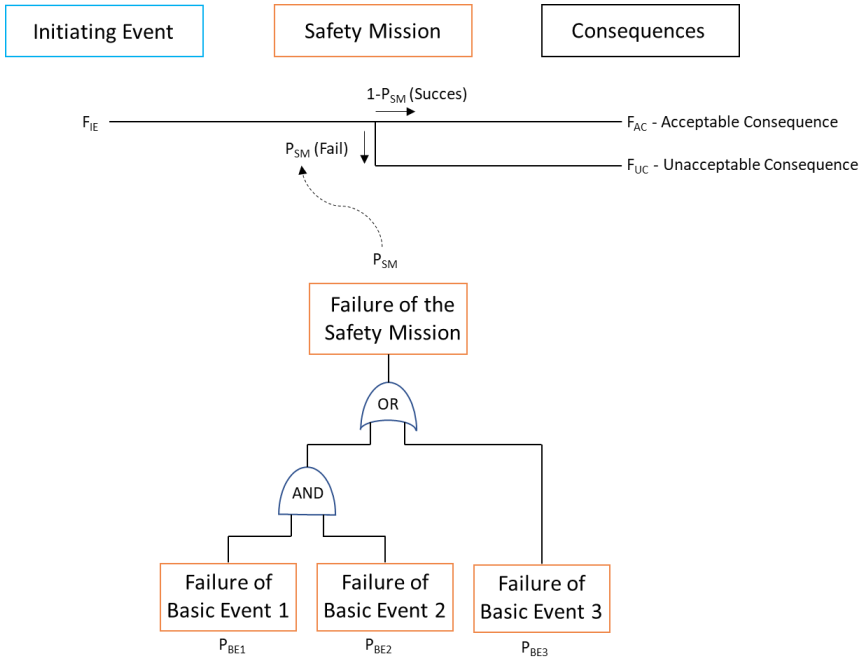


Figure 2: Example of a simple Event Tree and Fault Tree

The failure of the safety mission is modelled by a Fault Tree (see for example Figure 2) which contains gates (AND, OR, …) and events to represent Boolean formulas. Minimal Cut Sets of the Fault Tree are determined and the probability that the safety mission failed, P_{SM}, (the TOP event of the Fault Tree) is calculated.

The frequency of Unacceptable Consequence is then calculated by:

$$f_{UC} = f_{IE} \times P_{SM}$$

In a full-size PSA study, there are multiple event trees containing several safety missions, and multiple paths can lead to the Unacceptable Consequence.

SCRAM makes it possible to calculate the probability of failure of a safety mission at a time t (of a TOP Event), given the failure probabilities of the basic events depending on time, denoted Q(t). In other words, SCRAM makes it possible to calculate time-dependent unavailability of a safety system. The basic event failure models available in SCRAM are those commonly used:

- Repairable component (exponential distributions for failure and repair processes),
- Periodically tested component (exponential distribution for the failure process, constant fixed test interval, constant fixed repair time. Optional time to first test different than test interval),
- Probability – Constant unavailability (Failure probability per demand),
- Fixed mission-time component,

- Non-repairable component (exponential distribution),
- Weibull distribution (exponential distribution with scale and shape factors),
- User defined.

In order to conduct a Seismic PSA study, we need:

- The mean unavailability over an infinite time interval of the basic events, denoted Q_{mean}, $Q(t\rightarrow\infty)$, or Q,
- The frequency of the Seismic Initiating Event.

The calculation of the mean unavailability of Basic Events is possible using user-defined probability of unavailability. Under Andromeda, it is possible to use different models for the Basic Events (REPARAIBLE, TESTED, FIXED, MISSION-TIME). Andromeda implements the mean unavailability formula given the parameters of the model to be provided by the user (Failure rate, Time to first test, …). The formulations of these time-dependent failure models Q(t) and the corresponding average unavailability Q are provided in the Appendix in §7.

In the same way, it is possible in command line to use the mean probability calculated beforehand using the FIXED model (constant probability), or to use the user-defined model and by filling in the formula for the average unavailability.

3. SCRAM Workshop – Example of use

In this section, the example of the use of SCRAM to quantify the failure probability of a basic Fault Tree is given.

3.1. Application: Loss of Core Cooling system

We consider the Fault Tree used during the Seismic PSA Workshop ([10], 02/2022) within the METIS Project. The application concerns a simplified facility with two cooling and shut-down paths. Each path contains the following components: a) emergency diesel generator, b) bus (electrical cabinet), c) emergency feedwater system composed of a pump and a tank and d) cooling water supply system composed of a pump and a pipe. Both paths imply the same steam generator. The Fault Tree that represents the loss of the core cooling function (TOP event) is then composed of an AND gate between the gates that represent the loss of each shutdown path, see Figure 3. The Fault Tree associated with the loss of one shutdown path (for example, path 1) is given in Figure 4.

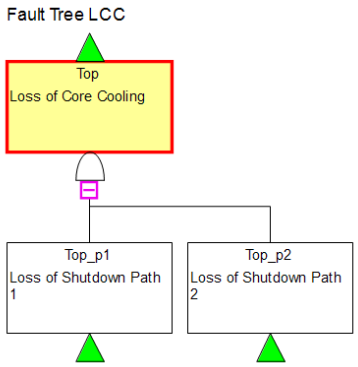


Figure 3: Loss of Core Cooling - Fault Tree

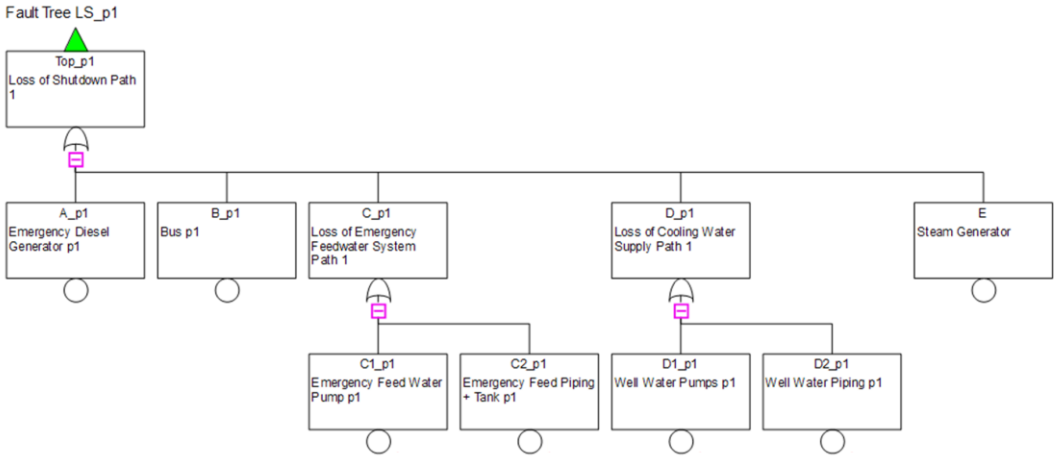


Figure 4: Loss of Shutdown Path 1 - Fault Tree

Fault Tree for path 2 is the same as for path 1, except that Basic Events ending with "p1" are replaced by new Basic Events ending with "p2". Basic Event E (steam generator) is common.

The Fault Tree for the Loss of Shutdown Path 1 is given in OPSA-MEF format in the Appendix in §8. The correspondence between the names of the Basic Events and Gates of the Fault Tree in Figure 4 (which has been built in Andromeda) and in Appendix §8 is given in Table 1 hereafter:

Type	Name in the Graphical Fault Tree (Figure)	Name in the text Fault Tree (Appendix §8)	Designation
Basic Event	A_p1	BE1	Emergency Diesel Generator
	B_p1	BE2	Bus
	C1_P1	BE3	Emergency Feed Water Pump
	C2_p1	BE4	Emergency Feed Piping
	D1_p1	BE5	Well Water Pump
	D2_p1	BE6	Well Water Piping
	E	BE7	Steam Generator
Gate	C_p1	G01	Loss of Emergency Feedwater System Path 1
	D_p1	G02	Loss of Cooling Water Supply Path 1
	Top_p1	TOP	Loss of Shutdown Path 1

Table 1: Basic Events and Gates of the Fault tree for Loss of Shutdown path 1

Failure probabilities used for the Basic Events are FIXED probabilities.

Given the fact that the fault tree is an OR gate between 7 Basic Events, the Minimal Cut Sets are trivially these 7 Basic Events. The TOP probability can be calculated using the rare-event approximation or the Min-Cut Upper Bound approximation:

$$P(Top)_{rare-event} = \sum_{i=1}^{7} P(BE_i)$$

$$P(Top)_{MCUB} = 1 - \prod_{i=1}^{7} (1 - P(BE_i))$$

The exact probability calculation can also be achieved using Binary Decision Diagram (BDD) based algorithms.

The results obtained by using these different ways of quantifying the probability of the Top event are provided in Appendix §8 using a given set of probabilities for the BEs.

The same exercise is conducted for the Fault Tree Loss of Core Cooling of Figure 3, which is an AND gate between the two paths (both paths must fail to fail the Top). The list of MCS is composed of 37 cut sets (1 of order 1, the Basic Event E, and 36 of order 2). The list of MCS and the Top failure probability calculations are provided in Appendix §9.

4. Conclusion

The PSA tool SCRAM-Andromeda developed within Work Package 7 of the METIS Project and destined to perform the risk quantification is presented in this report. The main goal of WP7 is to provide an open-source PSA tool that can be freely used by the METIS community, and to address technical issues of seismic PSA.

This technical report includes an IT description of the initial development of SCRAM and also the work that has been performed within METIS to make this tool compilable (developer level) and usable (user level). A procedure for compiling SCRAM under Windows is provided, as well as the basic commands to use it. The integration of SCRAM in Andromeda and the procedures for using SCRAM in command line or integrated into Andromeda are provided. Two examples of quantifications are also provided.

5. Bibliography

[1] Olzhas Rakhimov. SCRAM Fault tree analysis, 2016.

[2] Mohamed Hibti, Thomas Friedlhuber, and Antoine Rauzy. Overview of the open PSA platform. In *11th International Probabilistic Safety Assessment and Management Conference and the Annual European Safety and Reliability Conference 2012, PSAM11 ESREL 2012*, 2012.

[3] Hibti, Mohamed. Vers une EPS modulaire. In *Actes du congr´e lambda/mu*, La Rochelle, 2010.

[4] Friedlhuber T. Model Engineering in a Modular PSA, PhD Disertation, Ecole Polytechnique. Oct 2014.

[5] Epstein S., Rauzy A., and Reinhart FM. The Open PSA initiative for next generation probabilistic safety assessment. *Kerntechnik*, 2009.

[6] Bray T., Paoli J., Sperberg-McQueen M., Maler E., and Yergeau F. Extensible markup language (xml). World Wide Web Consortium Recommendation. 1998.

[7] Epstein S. and Rauzy A. Standard model representation format for probabilistic safety assessment, version 2.d. Technical report, The Open PSA Initiative. 2007-2008.

[8] M Hibti. Specification of the PSA format. METIS, Deliverable D7.1, 29/04/2022.

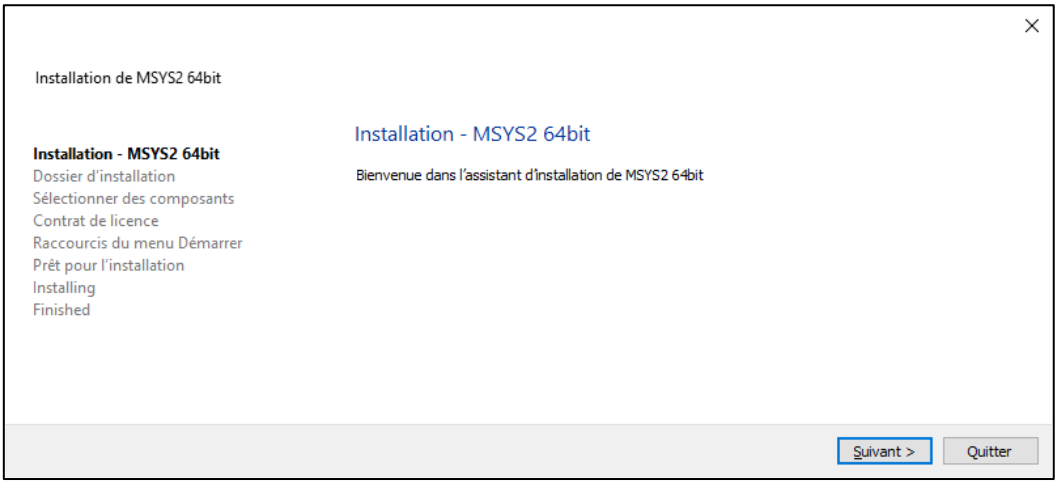
[9] M Hibti, T. Friedlhuber, and T Abdellatif. Model Cartography Diving in Complex PSA Models. *Proceedings of the 30th European Safety and Reliability Conference and 15th Probabilistic Safety Assessment and Management Conference*, 2020.

[10] Renault P. Educational example of a seismic PSA for a critical facility. From hazard to CDF – Current practice. Andromeda-Scram & Seismic-PSA theory Workshop, 25/02/2022.

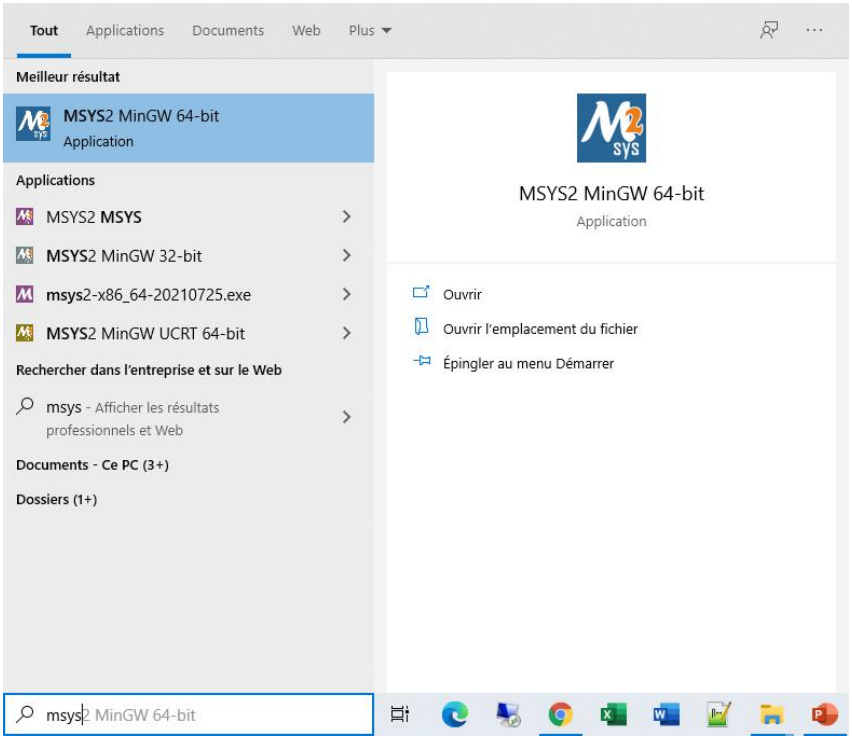
6. APPENDIX A – Compilation of SCRAM

Installation and configuration of the Linux-like MSYS2 environment on Windows

Go to https://www.msys2.org/#installation and download the executable. Run the executable as administrator, follow the instructions to complete the installation.

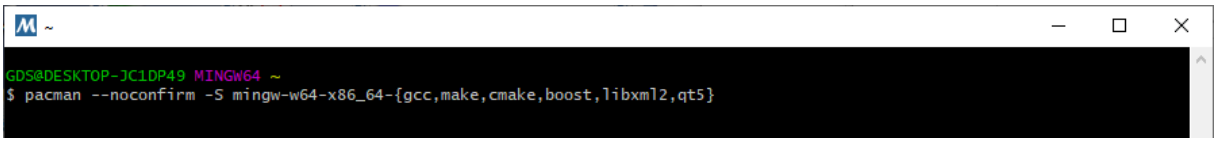


Once the installation is complete, launch the MSYS2 MingGW 64-bit console:

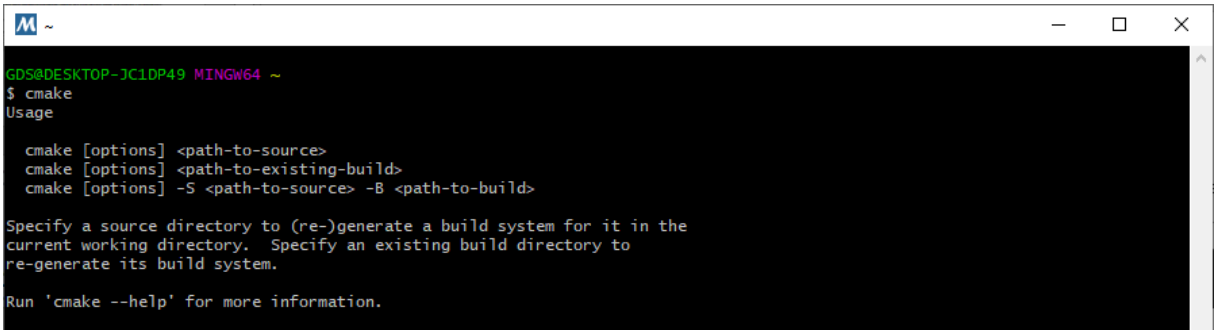


Install scram dependencies:

```
pacman --noconfirm -S mingw-w64-x86_64-{gcc,make,cmake,boost,libxml2,qt5}
```



Check the correct installation of cmake:



Install Git:

```
pacman --noconfirm -S git
```

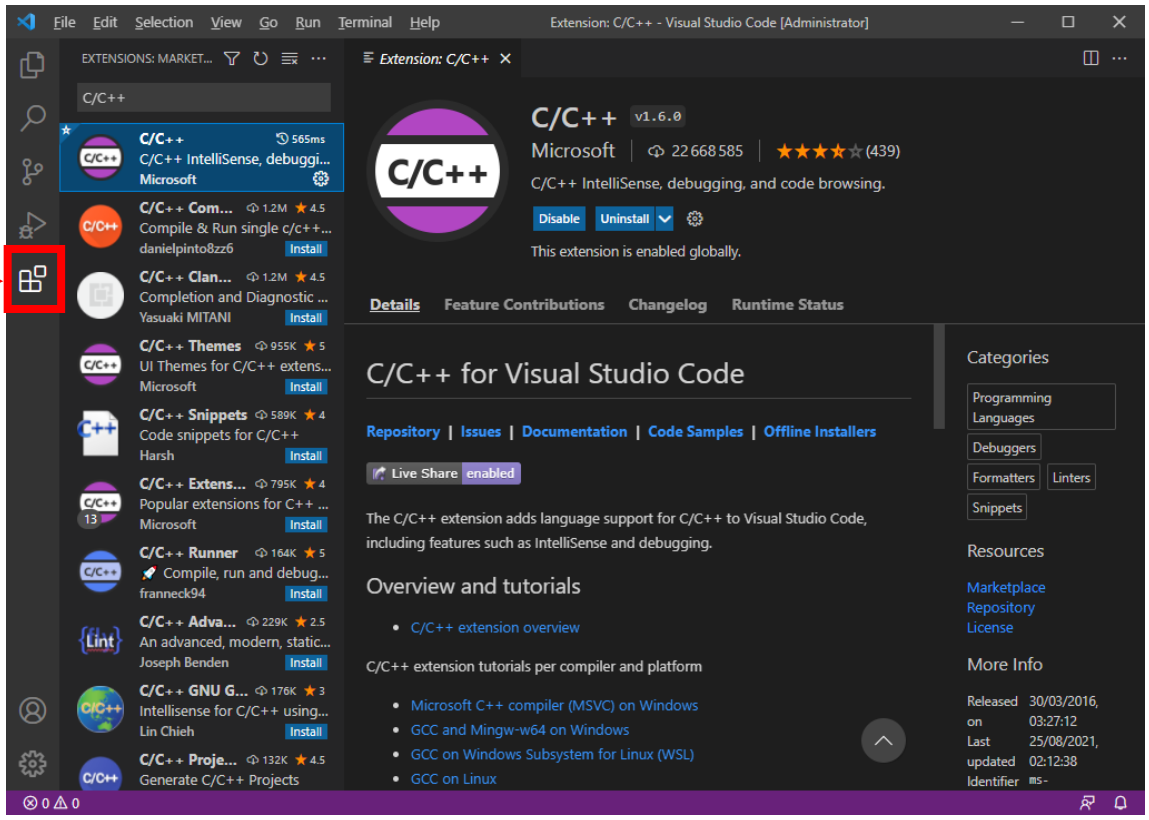
Add **c:\msys64\mingw64\bin** to the Path environment variable.

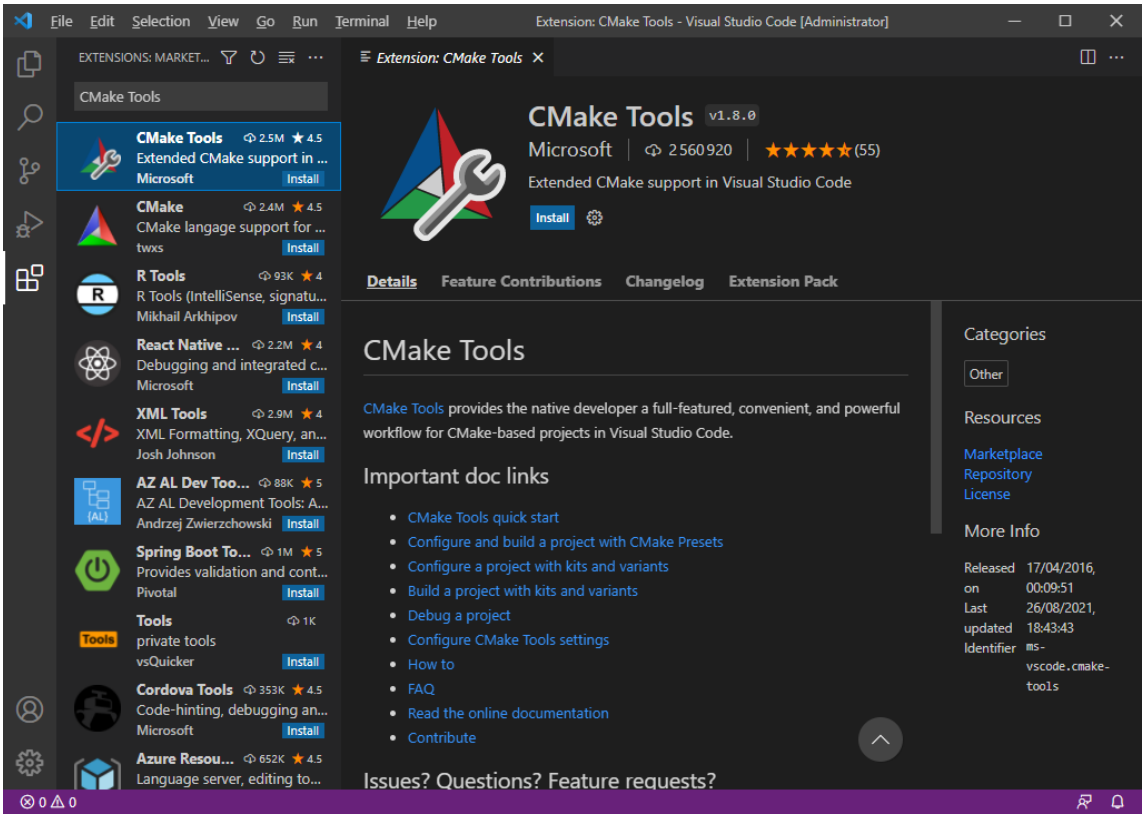
Installation and configuration of Visual Studio Code

Go to https://code.visualstudio.com/Download and download the executable for Windows. Run it and follow the instructions to complete the installation.

Launch VS Code and install the following extensions by clicking on the Extension tab on the left:

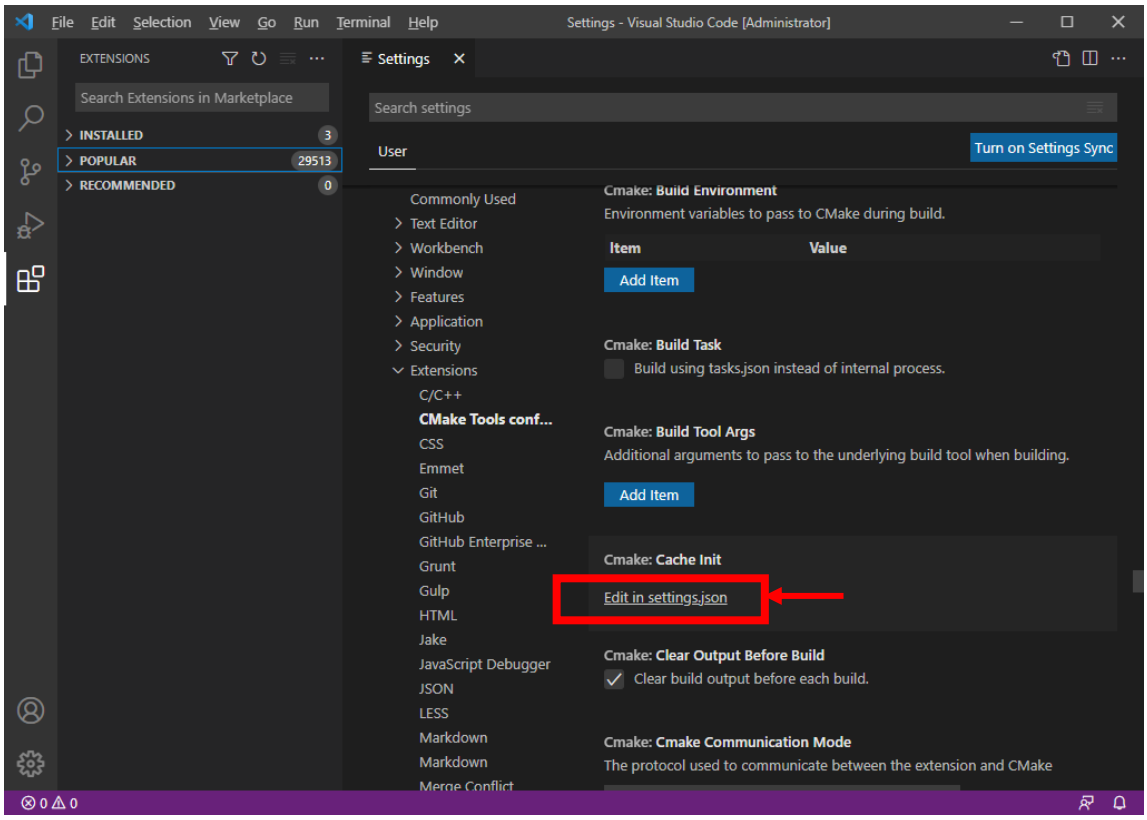
- C/C++
- Cmake Tools





In the top menu, go to File>Preferences>Settings

In the Settings file that opens, find the line "Edit with settings.json" and click on it.



In the settings.json file that opens, add the following line:

```
"cmake.generator": "MinGW Makefiles",
```

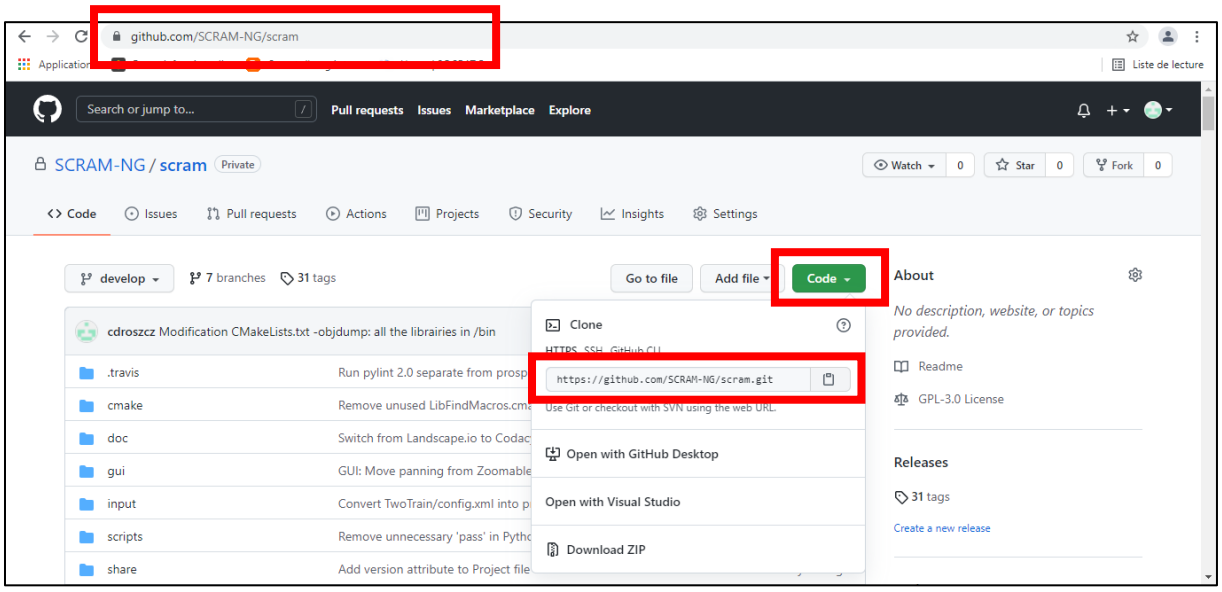
Save the file.

Clone the SCRAM repository from GitHub

Go to the GitHub page of the scram project: https://github.com/SCRAM-NG/scram

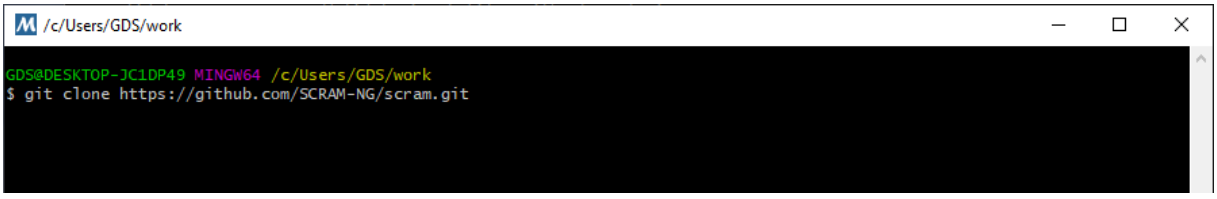
Log in.

To retrieve the URL of the git repository, click on Code, and copy the address that appears:



In the MSYS2 MingGW 64-bit console, go to the working directory, and clone the GitHub repository:

```
git clone https://github.com/SCRAM-NG/scram.git
```

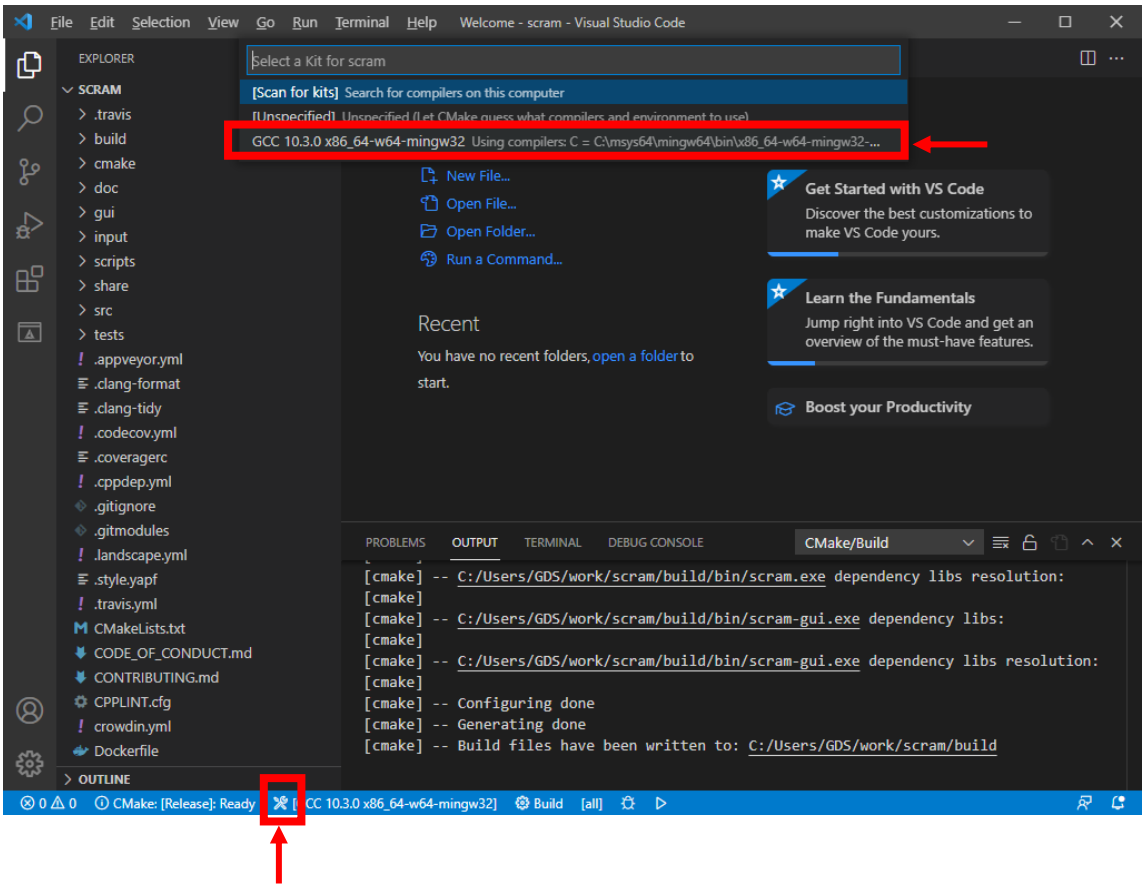


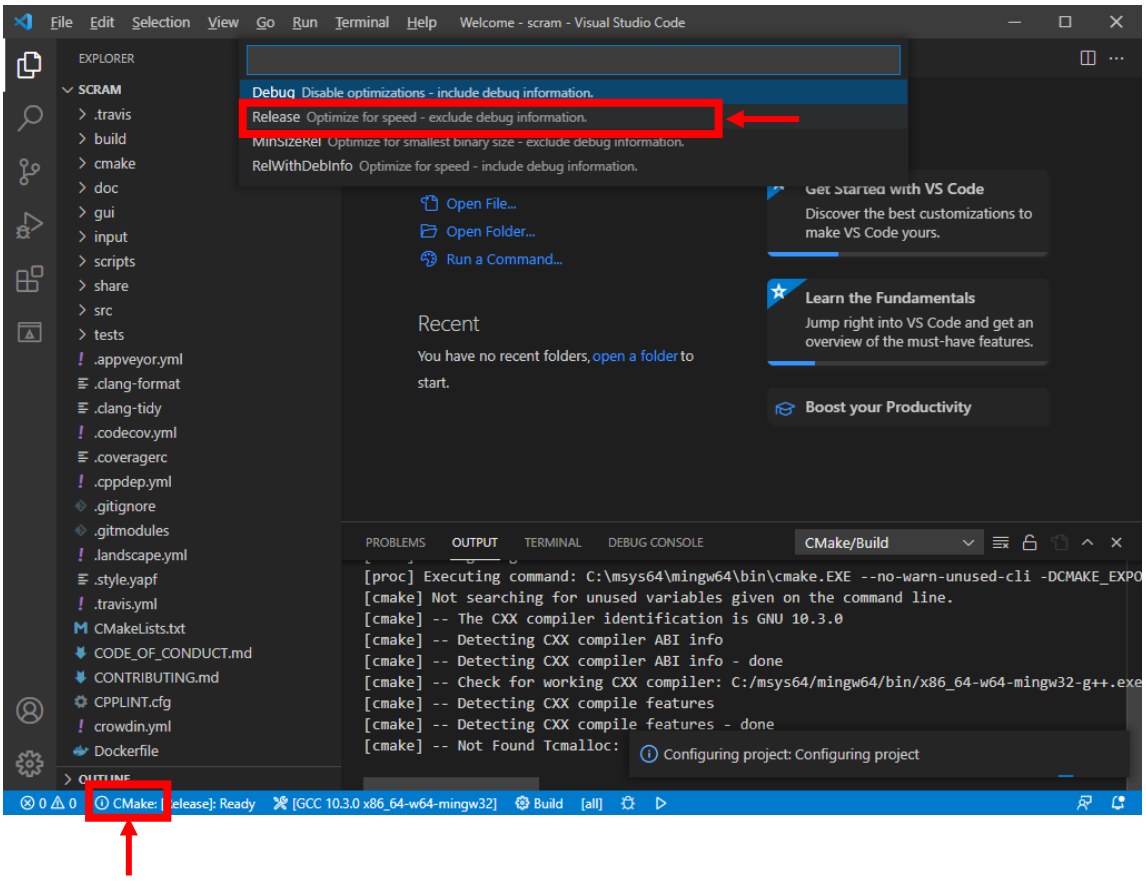
Login and password of your GitHub account will be requested. Once the operation is complete, the scram folder is created in the working directory.

Compilation of SCRAM on Visual Studio Code

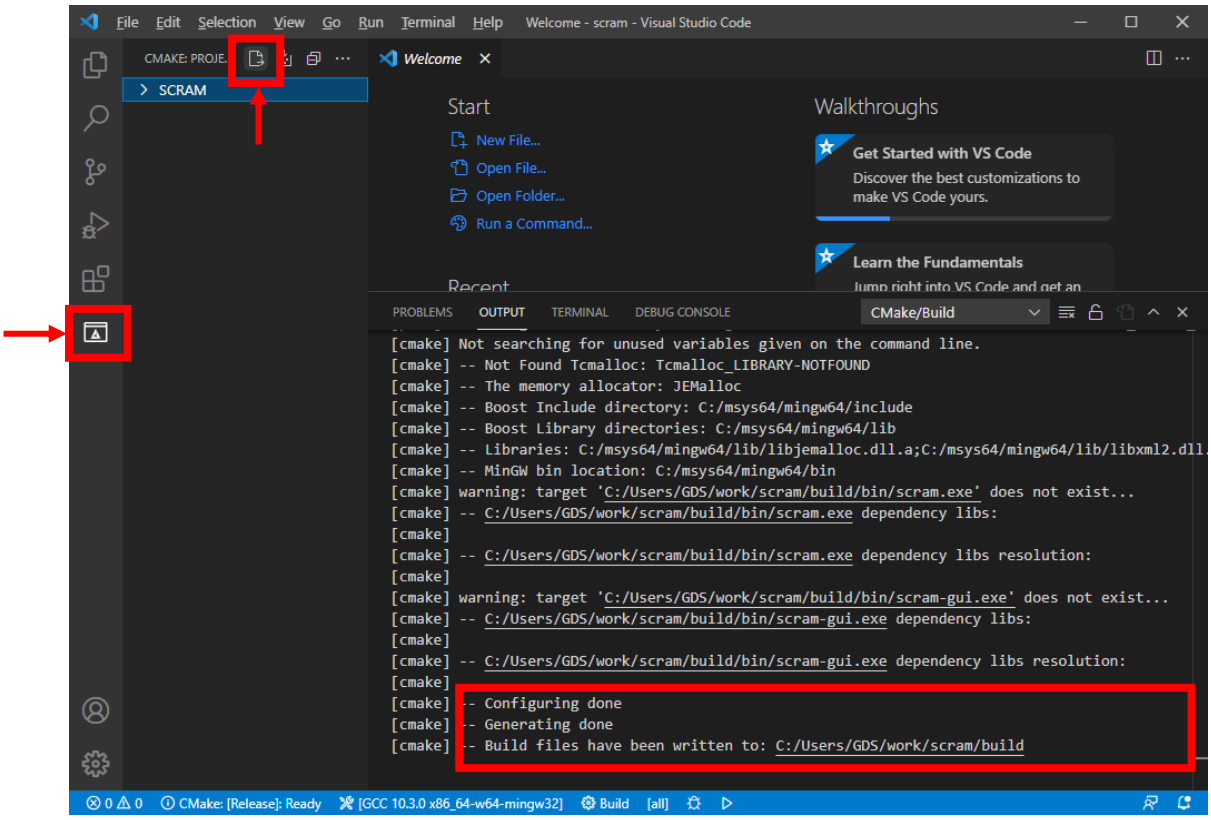
In Visual Studio Code, open the cloned scram directory.

At the bottom of the window, click on the Kit to select the GCC compiler and on Cmake to select the Release build target:

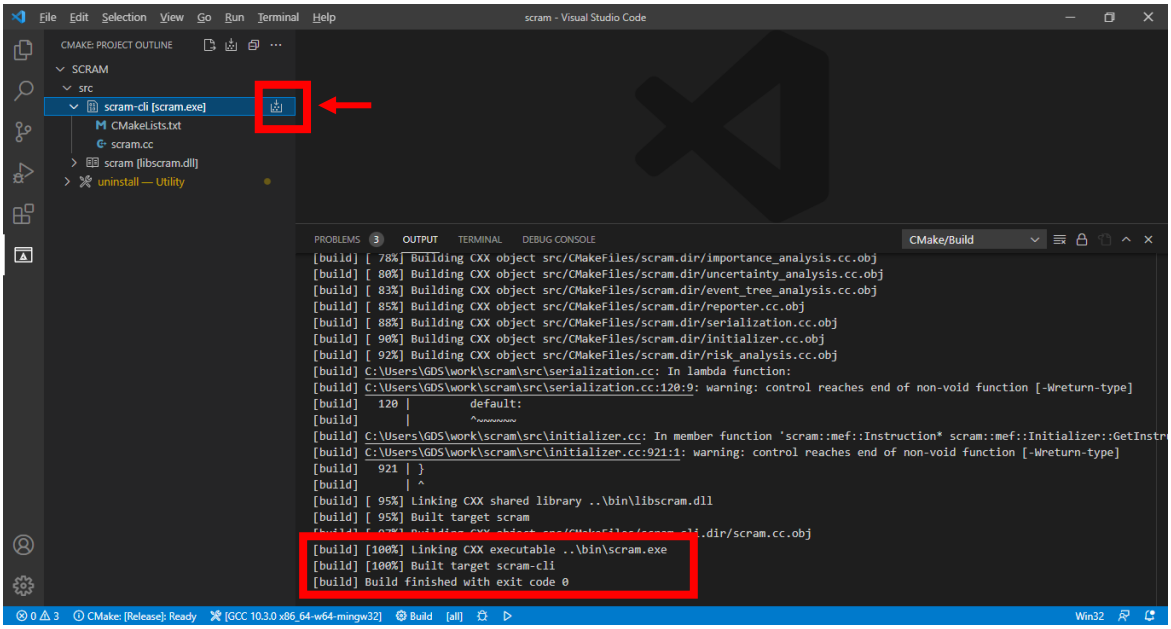




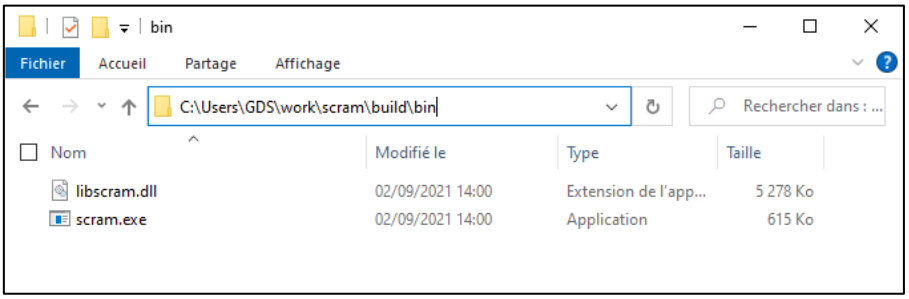
Click on the Cmake button on the left, then on Configure:



Run the scram-cli (for command-line) target build:



Check that you have the **scram.exe** executable and the **libscram.dll** binary in the created bin directory:



By opening a console, call the scram.exe executable with the --help option then --version for verification:

```
C:\Users\GDS\work\scram\build\bin>scram.exe --help
Usage:    scram [options] input-files...

Options:
  --help                Display this help message
  --version             Display version information
  --project path        Project file with analysis configurations
  --allow-extern        **UNSAFE** Allow external libraries
  --validate            Validate input files without analysis
  --bdd                 Perform qualitative analysis with BDD
  --zbdd                Perform qualitative analysis with ZBDD
  --mocus               Perform qualitative analysis with MOCUS
  --prime-implicants    Calculate prime implicants
  --probability         Perform probability analysis
  --importance          Perform importance analysis
  --uncertainty         Perform uncertainty analysis
  --ccf                 Perform common-cause failure analysis
  --sil                 Compute the Safety Integrity Level metrics
  --rare-event          Use the rare event approximation
  --mcub                Use the MCUB approximation
  -l [ --limit-order ] int Upper limit for the product order
  --cut-off double      Cut-off probability for products
  --mission-time double System mission time in hours
  --time-step double    Time step in hours for probability analysis
  --num-trials int      Number of trials for Monte Carlo simulations
  --num-quantiles int   Number of quantiles for distributions
  --num-bins int        Number of bins for histograms
  --seed int            Seed for the pseudo-random number generator
  -o [ --output ] path  Output file for reports
  --no-indent           Omit indentation whitespace in output XML
  --verbosity int       Set log verbosity
```

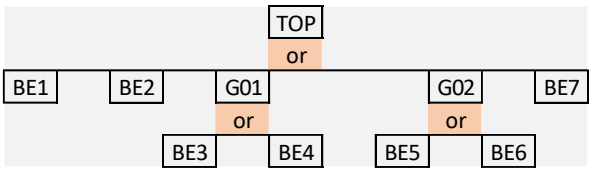
7. APPENDIX B – Failure models for the Basic Events in SCRAM

Model	Parameters	Formulation – Unavailability Q(t)	Long-term unavailability Q
REPAIRABLE	$\lambda\ (FR)$: Failure rate $\mu = 1/MTTR$: Repair rate (MTTR: Mean Time to Repair)	$Q(t) = \frac{\lambda}{\lambda+\mu}\left(1 - e^{-(\lambda+\mu)t}\right)$ $Q(0) = 0;\ Q(\infty) = \frac{\lambda}{\lambda+\mu}$	$Q = Q(\infty) = \frac{\lambda}{\lambda+\mu}$
	Optional: q : Prob/demand	$Q(t) = qe^{-(\lambda+\mu)t} + \frac{\lambda}{\lambda+\mu}\left(1 - e^{-(\lambda+\mu)t}\right)$ $Q(0) = q;\ Q(\infty) = \frac{\lambda}{\lambda+\mu}$	
TESTED	$\lambda\ (FR)$: Failure rate TI: Test Interval	$Q(t) = \begin{cases} 1 - e^{-\lambda t} & \text{if } t < TI \\ 1 - e^{-\lambda (t \bmod \mathrm{TI})} & \text{if } t > TI \end{cases}$ $Q(0, nTI^+) = 0$	$Q = Q_{mean} = \frac{1}{TI}\int_0^{TI} Q(t)dt = 1 - \frac{1}{\lambda TI}(1 - e^{-\lambda TI})$
	Optional: $TF\ (TTF)$: Time to First Test	$Q(t) = \begin{cases} 1 - e^{-\lambda t} & \text{if } t < TF \\ 1 - e^{-\lambda ((t-TF) \bmod \mathrm{TI})} & \text{if } t > TF \end{cases}$ $Q(0, TF^+, nTI^+) = 0$	
	Optional: $TR\ (MTTR)$: Repair Time	$Q(t) = \begin{cases} 1 - e^{-\lambda t} & \text{for } t < TF \\ Q(TI) = 1 - e^{-\lambda TI} & \text{for } t = TF + nTI \\ Q(TI) + \left(1 - Q(TI)\right)\left(1 - e^{-\lambda (t-TI)}\right) & \text{for } TI < t < TI + TR \\ 1 - e^{-\lambda (t-TI)} & \text{for } TI + TR < t \end{cases}$ $Q(0) = 0$ $Q(TF + nTI^+) = 1 - e^{-\lambda TI}$ $Q(TF + nTI + TR) = Q_{TR} = 1 - e^{-\lambda TR}$	$Q = Q_{mean} = 1 - \frac{1}{\lambda TI}(1 - e^{-\lambda TI}) + (1 - e^{-\lambda TI})\frac{TR}{TI}$
	Optional: q: Probability	$Q(t) = \begin{cases} 1 - (1-q)e^{-\lambda t} & \text{for } t < TF \\ Q(TI) = 1 - (1-q)e^{-\lambda TI} & \text{for } t = TF + nTI \\ Q(TI) + \left(1 - Q(TI)\right)\left(1 - (1-q)e^{-\lambda (t-TI)}\right) & \text{for } TI < t < TI + TR \\ 1 - (1-q)e^{-\lambda (t-TI)} & \text{for } TI + TR < t \end{cases}$ $Q(0) = q$ $Q(TF + nTI^+) = Q_{TI} + (1 - Q_{TI})q = 1 - (1-q)e^{-\lambda TI}$ $Q(TF + nTI + TR^+) = Q_{TR} = 1 - (1-q)e^{-\lambda TR}$	$Q = Q_{mean} = 1 - \frac{1}{\lambda TI}(1-q)(1 - e^{-\lambda TI}) + \left(1 - (1-q)e^{-\lambda TI}\right)\frac{TR}{TI}$
FIXED	q: Probability	$Q(t) = Q = q$	$Q = Q(\infty) = Q_{mean} = q$
MISSION-TIME	$\lambda\ (FR)$: Failure rate TM: Mission-Time	$Q(t) = Q = 1 - e^{-\lambda TM}$	$Q = Q_{mean} = 1 - (1-q)e^{-\lambda TM}$
	Optional: q: Prob/demand	$Q(t) = Q = 1 - (1-q)e^{-\lambda TM}$	
UNREPARAIBLE	$\lambda\ (FR)$: Failure rate	$Q(t) = 1 - e^{-\lambda t}$ $Q(0) = 0;\ Q(\infty) = 1$	$Q = Q_{mean} = 1$

	Optional: q: Prob/demand	$Q(t) = 1 - (1 - q)e^{-\lambda t}$ $Q(0) = q;\ Q(\infty) = 1$	

8. APPENDIX C – Fault Tree of the Loss of Shutdown Path 1

Fault Tree Loss of Shutdown path 1




```
<opsa-mef>
<define-fault-tree name="LSP1">
<define-gate name="TOP">
<or>
<basic-event name="BE1"/>
<basic-event name="BE2"/>
<basic-event name="BE7"/>
<gate name="G01"/>
<gate name="G02"/>
</or>
</define-gate>
<define-gate name="G01">
<or>
<basic-event name="BE3"/>
<basic-event name="BE4"/>
</or>
</define-gate>
<define-gate name="G02">
<or>
<basic-event name="BE5"/>
<basic-event name="BE6"/>
</or>
</define-gate>
</define-fault-tree>
<model-data>
<define-basic-event name="BE1">
<float value="0.033"/>
</define-basic-event>
<define-basic-event name="BE2">
<float value="0.0074"/>
</define-basic-event>
<define-basic-event name="BE3">
<float value="0.077"/>
</define-basic-event>
<define-basic-event name="BE4">
<float value="7.7e-12"/>
</define-basic-event>
<define-basic-event name="BE5">
<float value="0.11"/>
</define-basic-event>
<define-basic-event name="BE6">
<float value="1.1e-11"/>
</define-basic-event>
<define-basic-event name="BE7">
<float value="0.11"/>
```

```
</define-basic-event>
</model-data>
</opsa-mef>
```

Basic Events and Gates

Type	Name in the Graphical Fault Tree (Figure)	Name in the text Fault Tree (Annex)	Designation	Probability (constant)
Basic Event	A_p1	BE1	Emergency Diesel Generator	3.30E-02
	B_p1	BE2	Bus	7.40E-03
	C1_p1	BE3	Emergency Feed Water Pump	7.70E-02
	C2_p1	BE4	Emergency Feed Piping	7.70E-12
	D1_p1	BE5	Well Water Pump	1.10E-01
	D2_p1	BE6	Well Water Piping	1.10E-11
	E	BE7	Steam Generator	1.10E-01
Gate	C_p1	G01	Loss of Emergency Feedwater System Path 1	
	D_p1	G02	Loss of Cooling Water Supply Path 1	
	Top_p1	TOP	Loss of Shutdown Path 1	

Minimal Cut Sets

The list of Cuts Sets is provided by Andromeda, with the probability of each Cut Set and the contribution to the TOP failure probability.

Rank	P	Contribution (%)	Order	Event 1
1	0.11	36.9	1	E
2	0.11	36.9	1	D1_p1
3	0.077	25.8	1	C1_p1
4	0.033	11.1	1	A_p1
5	0.0074	2.5	1	B_p1
6	1.1E-11	0	1	D2_p1
7	7.7E-12	0	1	C2_p1

Quantification

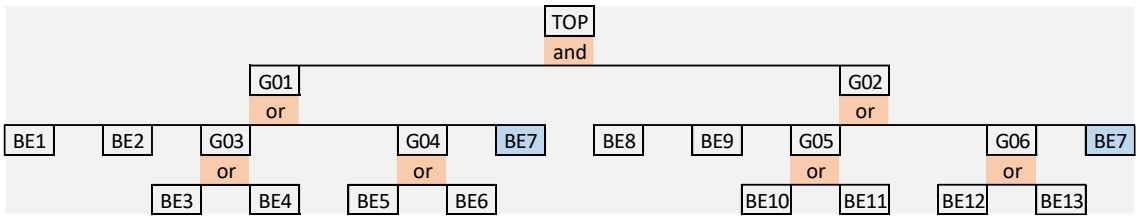
TOP failure probability is calculated with SCRAM in command-line, using different possible approximations to quantify the TOP.

Approximation	Top Probability
Rare-event	0.3374
MCUB	0.29825
Exact (BDD)	0.29825

The TOP probability can be calculated as well directly in Andromeda.

9. APPENDIX D – Fault Tree of the Loss of Core Cooling

Fault Tree Loss of Core Cooling




```
<opsa-mef>
<define-fault-tree name="LCC">
<define-gate name="TOP">
<and>
<gate name="G01"/>
<gate name="G02"/>
</and>
</define-gate>
<define-gate name="G01">
<or>
<basic-event name="BE1"/>
<basic-event name="BE2"/>
<basic-event name="BE7"/>
<gate name="G03"/>
<gate name="G04"/>
</or>
</define-gate>
<define-gate name="G02">
<or>
<basic-event name="BE8"/>
<basic-event name="BE9"/>
<basic-event name="BE7"/>
<gate name="G05"/>
<gate name="G06"/>
</or>
</define-gate>
<define-gate name="G03">
<or>
<basic-event name="BE3"/>
<basic-event name="BE4"/>
</or>
</define-gate>
<define-gate name="G04">
<or>
<basic-event name="BE5"/>
<basic-event name="BE6"/>
</or>
</define-gate>
<define-gate name="G05">
<or>
<basic-event name="BE10"/>
<basic-event name="BE11"/>
</or>
</define-gate>
```

```
<define-gate name="G06">
<or>
<basic-event name="BE12"/>
<basic-event name="BE13"/>
</or>
</define-gate>
</define-fault-tree>
<model-data>
<define-basic-event name="BE1">
<float value="0.033"/>
</define-basic-event>
<define-basic-event name="BE2">
<float value="0.0074"/>
</define-basic-event>
<define-basic-event name="BE3">
<float value="0.077"/>
</define-basic-event>
<define-basic-event name="BE4">
<float value="7.7e-12"/>
</define-basic-event>
<define-basic-event name="BE5">
<float value="0.11"/>
</define-basic-event>
<define-basic-event name="BE6">
<float value="1.1e-11"/>
</define-basic-event>
<define-basic-event name="BE7">
<float value="0.11"/>
</define-basic-event>
<define-basic-event name="BE8">
<float value="0.033"/>
</define-basic-event>
<define-basic-event name="BE9">
<float value="0.0074"/>
</define-basic-event>
<define-basic-event name="BE10">
<float value="0.077"/>
</define-basic-event>
<define-basic-event name="BE11">
<float value="7.7e-12"/>
</define-basic-event>
<define-basic-event name="BE12">
<float value="0.11"/>
</define-basic-event>
<define-basic-event name="BE13">
<float value="1.1e-11"/>
</define-basic-event>
</model-data>
</opsa-mef>
```

Basic Events and Gates

Type	Name in the Graphical Fault Tree (Figure)	Name in the text Fault Tree (Annex)	Designation	Probability (constant)
Basic Event	A_p1	BE1	Emergency Diesel Generator	3.30E-02
	B_p1	BE2	Bus	7.40E-03
	C1_p1	BE3	Emergency Feed Water Pump	7.70E-02

	C2_p1	BE4	Emergency Feed Piping	7.70E-12
	D1_p1	BE5	Well Water Pump	1.10E-01
	D2_p1	BE6	Well Water Piping	1.10E-11
	E	BE7	Steam Generator	1.10E-01
	A_p2	BE8	Emergency Diesel Generator	3.30E-02
	B_p2	BE9	Bus	7.40E-03
	C1_p2	BE10	Emergency Feed Water Pump	7.70E-02
	C2_p2	BE11	Emergency Feed Piping	7.70E-12
	D1_p2	BE12	Well Water Pump	1.10E-01
	D2_p2	BE13	Well Water Piping	1.10E-11
Gate	C_p1	G01	Loss of Emergency Feedwater System Path 1	
	D_p1	G02	Loss of Cooling Water Supply Path 1	
	Top_p1	TOP	Loss of Shutdown Path 1	

Minimal Cut Sets

Rank	P	Contribution (%)	Order	Event 1	Event 2
1	1.10E-01	71	1	E	
2	1.21E-02	7.8	2	D1_p1	D1_p2
3	8.47E-03	5.5	2	C1_p2	D1_p1
4	8.47E-03	5.5	2	C1_p1	D1_p2
5	5.93E-03	3.8	2	C1_p1	C1_p2
6	3.63E-03	2.3	2	A_p1	D1_p2
7	3.63E-03	2.3	2	A_p2	D1_p1
8	2.54E-03	1.6	2	A_p1	C1_p2
9	2.54E-03	1.6	2	A_p2	C1_p1
10	1.09E-03	0.7	2	A_p1	A_p2
11	8.14E-04	0.5	2	B_p1	D1_p2
12	8.14E-04	0.5	2	B_p2	D1_p1
13	5.70E-04	0.4	2	B_p1	C1_p2
14	5.70E-04	0.4	2	B_p2	C1_p1
15	2.44E-04	0.2	2	A_p2	B_p1
16	2.44E-04	0.2	2	A_p1	B_p2
17	5.48E-05	0	2	B_p1	B_p2
18	1.21E-12	0	2	D1_p2	D2_p1
19	1.21E-12	0	2	D1_p1	D2_p2
20	8.47E-13	0	2	C1_p2	D2_p1
21	8.47E-13	0	2	C2_p2	D1_p1
22	8.47E-13	0	2	C2_p1	D1_p2
23	8.47E-13	0	2	C1_p1	D2_p2
24	5.93E-13	0	2	C1_p2	C2_p1
25	5.93E-13	0	2	C1_p1	C2_p2
26	3.63E-13	0	2	A_p1	D2_p2

27	3.63E-13	0	2	A_p2	D2_p1
28	2.54E-13	0	2	A_p1	C2_p2
29	2.54E-13	0	2	A_p2	C2_p1
30	8.14E-14	0	2	B_p1	D2_p2
31	8.14E-14	0	2	B_p2	D2_p1
32	5.70E-14	0	2	B_p1	C2_p2
33	5.70E-14	0	2	B_p2	C2_p1
34	1.21E-22	0	2	D2_p1	D2_p2
35	8.47E-23	0	2	C2_p2	D2_p1
36	8.47E-23	0	2	C2_p1	D2_p2
37	5.93E-23	0	2	C2_p1	C2_p2

Quantification

Approximation	Top probability
Rare-event	0.1617
MCUB	0.1550
Exact (BDD)	0.1498