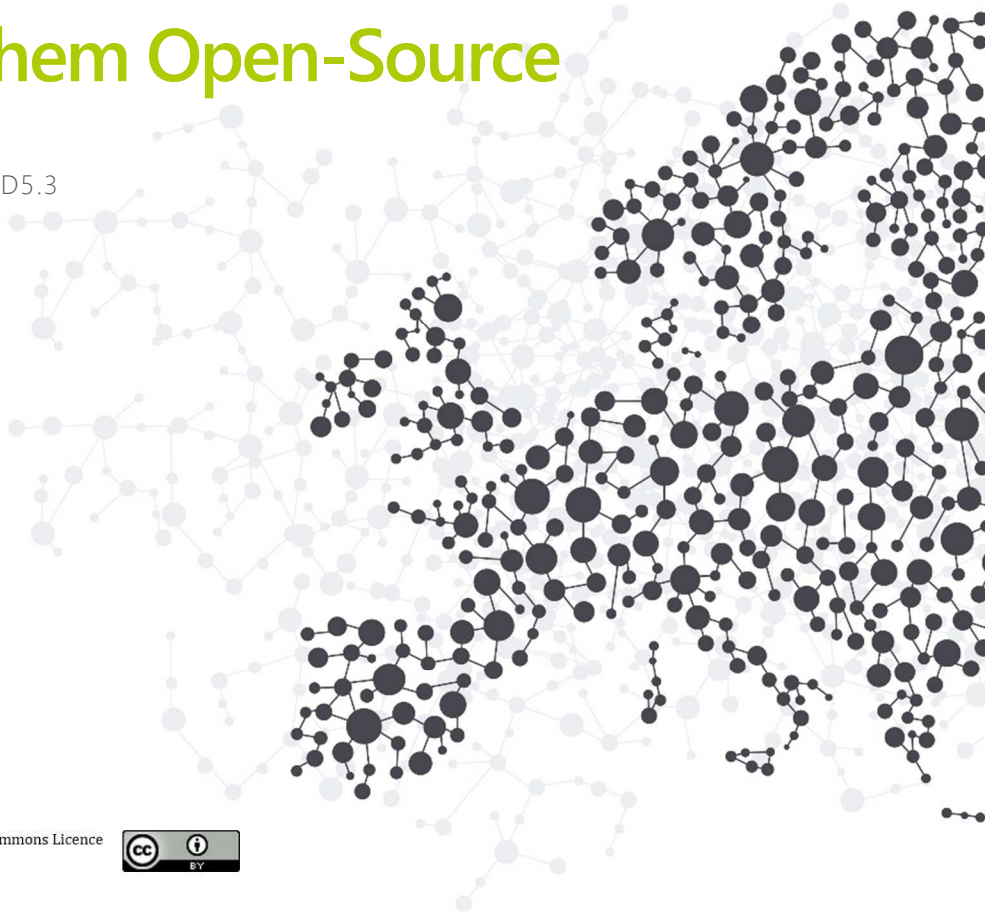
Definition and Implementation of Upgrades of openENTRANCE Models to Make Them Open-Source

DELIVERABLE NO. D5.3



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 835896

Deliverable No.	D5.3	Work Package No.	WP5
Work Package Title	A Suite of Modelling Tools		
Status	Final	(Draft/Draft Final/Final)	
Dissemination level	PU-Public	(PU-Public, CO-Confidential)	
Due date deliverable	2021.02.28	Submission date	2021.02.26
Deliverable version	V5		

Deliverable Contributors:	Name	Organisation	Date
Deliverable Leader	Paolo Pisciella	NTNU	2021.02.22
Work Package Leader	Luis Olmos Camacho	Comillas	
Contributing Author(s)	Pedro Crespo del Granado	NTNU	
	Theresia Perger	TU Wien	
	Erik Francisco Álvarez Quispe	Comillas	
	Sara Lumbreras Sancho	Comillas	
	Andrés Ramos Galán	Comillas	
	Sebastian Zwickl-Bernhard	TU Wien	
	Paolo Pisciella	NTNU	
	Karlo Hainsch	TU Berlin	
	Konstantin Löffler	TU Berlin	
	Hettie J. Boonman	TNO	
	Sandrine Charousset	EDF	
	Mohammadreza Ahang	NTNU	
Reviewer(s)	Daniel Huppmann	IIASA	
	Ingeborg Graabak	SINTEF Energy Research	
Final review and approval			

History of Change

Release	Date	Reason for Change	Status
Draft	2021.01.29	Initial draft	
Apply review feedback	2021.02.25	Inputs from quality reviewer	
Final version	2021.02.26	Polishing	

DISCLAIMER / ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 835896.

Table of Content

Glossary

Open-Source

An open-source model is a model whose code can be inspected, modified and enhanced by anyone without restrictions. It is distributed or released under a license approved by the Open Source Initiative (https://opensource.org/).

Copyleft license

A Copyleft license requires all modified and extended versions of a program to be open as well. Copyleft licenses require that anyone who redistributes the software, with or without changes, must allow future users to further copy and change it.

Permissive license

A Permissive license allows to use the original program with little restrictions on how you choose to use the source code and re-share the software. It is also known as "non-Copyleft" license and the main requirement is generally to include attribution of the original code to the original developers. A Permissive license does not have reciprocity obligations so there is no assurance that the software will always remain free and public. With a Permissive license, one can generally change and re-share someone else's program without disclosing the source code.

DOI

Digital Object Identifier; a code used to permanently and stably identify digital objects. DOIs provide a standard mechanism for retrieval of metadata about the object, and generally a means to access the data object itself.

FAIR

Guidelines to improve the Findability, Accessibility, Interoperability and Reusability of digital assets.

Open standards

A standard is a set of specifications adhered to by a producer, either tacitly or as a result of a formal agreement. An open standard is a standard that is freely available for adoption, implementation and updates.

Executive Summary

This deliverable presents a report of the work carried out by the modelling teams to produce open-source versions of their models. The core of the deliverable is defined by a set of tables, one for each modelling team, containing information about the steps taken to revise, restructure and simplify the source code of their models as well as providing them with a user guide and a suitable open-source license. Moreover, the deliverable contains the web addresses of the repositories where the models are stored and can be retrieved. Prior to the introduction of the modelling teams opening experiences, the deliverable provides a background on the concept of open-source models, its meaning in the context of the openENTRANCE project and provides a summary of the most utilized open-source licenses, categorizing them based on the level of freedom that they give to new users. The deliverable also discusses on the requirements that the models need to fulfill in terms of coding style and on their adherence to the so-called FAIR guidelines.

1. Introduction

1.1 Background

In openENTRANCE we are developing an Open Platform for analyses of the transition to low-carbon energy system. The Open Platform will include a suite of open linked models, a scenario explorer (database) and data. Furthermore, it will include a data format and a nomenclature. The Open Platform will gradually be made available to the research community in 2021. The open models described in this report are a part of this process. The linking process is based on the usage of a common interface and database, both described in deliverable D5.2 while the structure of the database, defined according to a predefined data format, is described in deliverable D4.2.

The idea of open-source has been generated by the information technology community. This sector is characterized by a strong development rate which could only occur on the premises of a tight collaboration. In particular, software development can leverage on the non-material characterization of the output to be shared among different developers around the world, which engage in an iterative improvement process. This collaboration scheme allows to obtain a fine-tuning of the results that is not possible to obtain in other sectors. Sharing ideas and building on each other's improvements is the basis for a healthy development of technology. With this premise, it is natural to seek the removal of possible barriers and limitations preventing this process of mutual knowledge exchange. In this respect, the creation of open-source licenses is a direct response to proprietary manufacturing. An open-source license, in its strongest form, dictates that the code needs to be openly available to anyone and that it is not possible to copyright or patent any derivative work or any improvement stemming from the initial product. As stated in the mission of the Open Source Initiative "Open source enables a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open-source is higher quality, better reliability, greater flexibility, lower cost, and an end to predatory vendor lock-in." The administration of open-source licensing is done by the Open-Source Initiative (OSI), founded in 1998. The purpose of the Initiative, besides administering open-source licensing around the world is promoting open-source development and facilitating community and educational initiatives.

In general terms, open-source models are defined as models that are distributed under an open-source license [8,9]. In openENTRANCE, we use the term "open modelling framework" for a software tool that is released under an established open-source license. For the combination of a software tool and the related data for at least a case study that is released under an established open-source license we use the term "open model" (cf. opensource.org/licenses).

The open model must satisfy the four freedoms: any user must be allowed to run the model, to study how it works and modify it, to redistribute it and to distribute copies of any modified version. The published dataset must include all required information to replicate the scientific analysis. We include in this definition models that are released under an appropriate license but require proprietary software (e.g., GAMS). An open-source model and its code can be inspected, modified and distributed by anyone without restrictions. Open-source development promotes collaborative development, enforces transparency by allowing anyone to analyse the source code and supports rapid development of solutions.

An important impact of the utilization of open-source approaches is the promotion of open standards [10]; a standard is a set of specifications adhered to by a producer, either tacitly or as a result of a formal agreement [14]. An open standard is a standard that is freely available for adoption, implementation and updates. The possibility of inspecting the source code of a program allows other parties to develop interoperable technological solutions which can interact with the open code with full compatibility. In general, all models are considered as copyrighted in the moment they are created [11], and the rights to use that model or software depend on the license that the copyright holder provides. Open-source distribution is still based on copyright laws. In this case, the copyright holder equips the model with an open-source license, which allows anyone to not only run it, but also view and change the model and the code itself and, finally, to redistribute the resulting changes to others. There are different ways for a model to be open-source and many different open-source licenses are available. A license states the degree in which a given model and code can be changed and/or combined with other code. There are two major classes of open-source licenses: Copyleft, which requires that any new version of the model and code is distributed under the same license and Permissive licenses, which allows the model and code to be recombined to proprietary code and be redistributed in a non open-source way. These classes of licenses will be discussed in section 2.1. In the context of modelling, “open” means that the data and the code of the model are published and shared using one of the possible open-source licenses.

1.2 Objectives

Policy analysis models are very often coded as black-boxes. This is not only an obstacle to the peer-based implementation of the models but also poses challenges in terms of the credibility of the results. When it is not possible to analyse how the different assumptions and/or the utilized dataset have been processed in the implementation of a model it becomes hard to believe in the accuracy of the results. More openness increases transparency, reduces double-work and improves the overall quality of the modelling framework. The result of open modelling is an advance in the level of research and a gain for both the modelling community and for the overall society. The development of a simulation model, whether it is focused on

the energy system, the power system, the economy or the environmental footprint, is a very long and time-consuming process. Data gathering and pre-processing of these models require a large amount of resources. Even though there is substantial overlap in data requirements among different modelling groups, data collection is often done independently and in isolation by each of them. An open modelling approach should therefore require the openness of data and source code in order to warrant the widest possible collaboration among researchers. OpenENTRANCE was created with the idea of providing an open and widely accessible platform and an interoperable suite of models to design and test simulations about possible pathways for decarbonization of the energy system and their economic and environmental impact. Under openENTRANCE, the models can require proprietary software to run, but need to be released under an open-source license. In order to enforce the openness required by openENTRANCE it is important that the models comply with the FAIR guiding principles: the models and their code need to be **findable**, **accessible**, **interoperable** and **reusable**. In general, **findable** means that the data and model need to be discoverable with metadata, i.e., with descriptive tags, identifiable and locatable by means of a standard identification mechanism; as a mention by using a web search engine. By **accessible** we require the model and data to always be available and obtainable. This means that the model and data should be possibly uploaded in a repository or received on-demand. **Interoperable** means that the model and data can be exchanged with the largest possible degree of compatibility between different researchers and institutions. Finally, by **reusable** it is meant that the model and the data should be shared with the least restrictive license, so to allow the widest possible outreach. It is important to remember that albeit FAIR is a set of principles and not a standard, openENTRANCE requires the open models operating around its platform and the related data to follow these guidelines.

A more detailed interpretation of these principles applied to scientific data management has been provided by [2], and is as follows

Table 1 Interpretation of the FAIR principles

FAIR PRINCIPLES		
Findable	F1	data are assigned a globally unique and persistent identifier
	F2	data are described with rich metadata
	F3	metadata clearly and explicitly include the identifier of the data it describes
	F4	data are registered or indexed in a searchable resource
Accessible	A1	data are retrievable by their identifier using a standardized communications protocol
	A1.1	the protocol is open, free, and universally implementable

	A1.2	the protocol allows for an authentication and authorization procedure, where necessary
	A2	metadata are accessible, even when the data are no longer available
Interoperable	I1	data use a formal, accessible, shared, and broadly applicable language for knowledge representation
	I2	data use vocabulary that follows FAIR principles
	I3	data include qualified references to other (meta)data
Reusable	R1	data are richly described with a plurality of accurate and relevant attributes
	R1.1	data are released with a clear and accessible data usage license
	R1.2	data are associated with detailed provenance
	R1.3	data meet domain-relevant community standards

To ensure that models and data operating around the openENTRANCE platform are consistent with the FAIR principles the modelling teams have been required to make sure that the following points have been considered for their model

- Satisfy the best practices for software development (see Section 2.2.2.)
- Existence of a user guide
- Availability of the code for download
- Availability of input data, if possible, for a simple case.
- Availability of an instance of the output data
- Application for an open-source license

Every model needs to be released under an established open-source license and must satisfy the four freedoms mentioned at the beginning of the chapter. Moreover, the dataset needs to be accessible. To facilitate their future usage, the models need to follow best practices for writing code, that is a set of conventions to make the code easily readable and understood. Normally this boils down to the definition of proper commentary, with the breaking down of the code into modules and the addition of a brief description of each of these modules. Good coding practices also involve the definition of proper naming conventions that remain uniform throughout the code with brief descriptions of what each variable is for. Moreover, one must seek to maintain the code as simple as possible. Notice that simple does not necessarily coincide with short. Even if compactness might reduce the scrolling by the reader, allowing to see more content within the same page, this does not necessarily translate into more simplicity. Finally, the code should be designed with scalability and reuse in mind. This

normally translates into a clear separation of code and data management, the definition of pre-determined input structures for new data and avoiding hard coding of particular cases.

The models have been also required to be complimented by a user guide. Normally, a user guide contains information about the sphere of application of the model, it includes a description of the data, the model itself and of the output produced. After the models have been polished to comply with the coding best practices and provided with a user guide, they have been equipped with a digital object identifier (DOI) and uploaded in a repository such as an institutional repository or a service like Zenodo to make sure that the work is available beyond the end of the project. The repository should contain the code for the model itself as well as input data for running a test case or even a full-fledged case. Moreover, to allow possible users to understand the version of the model stored in the repository, the modelling teams have been requested to implement semantic versioning for their models, with each version identified by a triplet of integers, called identifiers, that increase as new versions of the code are developed:

- The Major identifier is used when a backward incompatible change is made to the model;
- The Minor identifier is used to mark changes that are backward compatible;
- The patch version is used for bug fixes.

In this manner, versioning will be useful in future utilizations of the code also for what concerns dependency management [12].

In the remainder, this deliverable will touch on each of these points, to explain what they mean in the context of openENTRANCE and what has been the experience of each modelling team in following the steps of this opening process. Chapter 2 will elaborate in more detail on the degree of openness that a model can have, what it means to follow coding best practices and what are the requirements that openENTRANCE has set to certify a model as open. Chapter 3 contains reports in tabular form of the experience of each modelling team in the process of opening their model. Even though the planned title for this deliverable was “Definition and implementation of the upgrades to a selected subset of models in the suite to make them open-source” we have decided to slightly change it it to make it more compact and informative to the interested reader.

2. Defining an open model

We can define several degrees of openness for the implementation of models. On one side it is possible to implement a model as a black box, where no information is given for the details of

the mathematical formulation. In this case the program is only viewed in terms of inputs and outputs, without any knowledge of its internal structure. These programs are usually delivered as executable files accompanied by a use case example. On the other extreme one finds full open-access/open-source models. In this case the source code, together with a stylized case study for running the model are fully accessible through a specific web page or a centralized repository like GitHub. Notice that making a GitHub project public is not the same as licensing the project. Public projects are covered by GitHub's Terms of Service, which allow others to view and fork the project, but the work otherwise comes with no permissions. If the owner wants others to use, distribute, modify, or contribute back to the project, it is necessary to include an open-source license. For example, someone cannot legally use any part of a GitHub project in their code, even if it is public, unless the owner explicitly gives them the right to do so[1].

In what follows, we elaborate on the practical steps to be done to carry out the opening process.

2.1 Several degrees of openness

The degree of openness of a code is connected to the obligations and permissions that the developers have when they use, distribute or redistribute the code [6]. This is connected to the open-source license used for the code. Open-source licenses are legal and binding contracts between the author and the user of a software component, declaring that the software can be used in commercial applications under specified conditions. Without an open-source license, the software component is publicly unusable by others, even if the code has been posted on repositories such as GitHub. Each open-source license states what users are allowed to do with the software components, their obligations, and what they cannot do as per the terms and conditions. There are over 200 open-source licenses, varying in complexity and requirements. It is up to the organizations to choose which licenses are most compatible with their policies[2].

[1] https://opensource.guide/legal/
[2] https://resources.whitesourcesoftware.com/blog-whitesource/open-source-licenses-explained

Figure 1: Some popular types of licenses. Copyleft licenses are on the left, Permissive licenses are on the right

The main distinction for open-source licenses is between Copyleft and Permissive. This division is based on the requirements and restrictions that each license places on users. Copyleft is a method for making a program open which requires all modified and extended versions of the program to be open as well. Copyleft licenses require that anyone who redistributes the software, with or without changes, must allow other users to further copy and change the code[3]. On the other hand, a Permissive license allows the model and code to be recombined to proprietary code and be redistributed in a non open-source way. The main requirement is generally to include attribution of the original code to the original developers. A Permissive license does not have reciprocity obligations so there is no assurance that the software will always remain free and public. With a Permissive license, one can generally change and re-share someone else's program without disclosing the new code[4]. Here we report some of the most used types of licenses, classifying them as Copyleft or Permissive. Copyleft licenses are normally more adequate for research purposes, because they do not allow to "close" future evolutions of the code implementation, granting a higher level of innovation potential. On the other hand, Permissive licenses leave a larger margin of flexibility to the developers for commercial exploitation of the code. In what follows we report some of the most popular open-source licenses based on their categorization.

[3] https://www.gnu.org/Copyleft/
[4] https://fossa.com/blog/what-do-open-source-licenses-even-mean/

Popular Copyleft licenses

- GPL License: The GNU General Public License is a Copyleft license for software and other kinds of works. GPL is intended to guarantee the freedom to share and change all versions of a program to make sure it remains open software for all its users. For the developers and authors protection, the GPL clearly explains that there is no warranty for the software. The GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions[5].
- AGPL License: The Affero GPL license is usually applied to server-client applications. In the standard GPL license, the reciprocity clause comes into effect whenever a developer releases software. With AGPL, software teams can ensure all changes to the codebase becomes available to the public, even on server-side applications[6]. In other words, if one provides a service based on software conceived under AGPL licensing, then it is required to provide free access to the complete software for the service.

Popular Permissive Licenses

- BSD License: The Berkeley Software Distribution license is a simple license that merely requires that all code retain the BSD license notice if redistributed in source code format. It does not require that source code be distributed at all. This license has four clauses: redistributions of the source code must retain the same license type, redistribution in binary form must reproduce also the license disclaimer, advertising materials mentioning the use of software with this license must display an acknowledgement, names of the original producers cannot be used to promote products using the licensed code without explicit consent. Different versions of the BSD license have sparked after the original one; Among them, the modified BSD license removes the advertising clause.
  A BSD style license is a good choice for long duration research or other projects that need a development environment that has near zero cost, will evolve over a long period of time, and permits anyone to retain the option of commercializing results with minimal legal issues. Developers tend to find the BSD license attractive as it has very little legal restrictions, allowing them to use the code in a number of ways. In contrast, those who expect primarily to use a system rather than program it, or expect others to evolve the code, or who do not expect to monetize their work associated with the system (such as government employees), find the GPL attractive, because it forces code developed by others to be given to them and keeps their employer from retaining copyright and thus potentially "burying" or orphaning the software[7].

[5] https://www.gnu.org/licenses/gpl-3.0.en.html
[6] https://snyk.io/learn/agpl-license/
[7] https://www.freebsd.org/doc/en_US.ISO8859-1/articles/bsdl-gpl/article.html

- MIT License: The MIT open-source license is Permissive in nature and one of the simplest. It basically allows developers to modify or re-calibrate source code according to their preferences. The MIT license always includes a copyright statement and a disclaimer, which explains that the software is provided "as is" and that copyright holders will not be held liable for any claims or liabilities. Holders of the MIT license can produce, without restrictions, any derivative works from the original software and even reap commercial benefits from the sale of the secondary product. The MIT license is GPL-compatible, but unlike the traditionally restrictive GPL, it is not viral. This means that developers can modify the original code without "infecting" the resultant derivative code with the original license[8].
- Apache License: The Apache 1.1 licenses requirements are very similar to the BSD modified license requirements. The Apache 2.0 license adds that any modifications to the open-source code must be clearly marked with a change log. The Apache 2.0 license adds an important restriction, the patent non-assert clause, which essentially states that if you initiate patent litigation against any of the authors or contributors who helped create this open-source, you can no longer use the open-source[9]. The core specifications of the Apache License 2.0 consist of: 1) Software may be freely used, reproduced, modified, distributed or sold. 2) Software can be combined with other products and distributed or sold as packages. 3) Products derived or modified from licensed software can be distributed under other licenses. 4) A copy of the license must be redistributed along with any Apache software. 5) External contributions to the software are released under the ASF terms unless explicitly specified otherwise.

To provide an open-source license to a code one needs to add a LICENSE, LICENSE.txt or LICENSE.md file in the root directory of the repository where the code is stored. As an example, we report the procedure required in GitHub[10]:

1. Open your GitHub repository in a browser.
2. In the root directory, click on **Create new file**.
3. Name the file "LICENSE".
4. Click on **Choose a license template**.
5. Pick one of the licenses (all the ones mentioned in this article are there).

[8] https://www.kiuwan.com/blog/comparison-popular-open-source-licenses/
[9] http://sourceauditor.com/blog/tag/apache-license-obligations/
[10] https://www.freecodecamp.org/news/how-open-source-licenses-work-and-how-to-add-them-to-your-projects-34310c3cf94/

6. Once chosen, click on **Review and submit**.
7. **Commit** the file.

The selection of a license depends on many considerations. In general, if a program is developed for research purposes and the programmers wish to keep future developments of the source code open, they should opt for a Copyleft type license. If the developers want to allow others to use it for commercial purposes, a Permissive license is better suited, because it allows to maximize the usage of the code by others by allowing to redistribute the software without disclosing the modifications on the derived code.

2.2 Best practices for model development

Establishing a model as open-source should not merely be limited to picking a proper license. Instead, the modeller should follow a sequence of steps to make the model suitable to be opened and used in a collaborative scheme. These steps are followed to ensure that the models abide to the FAIR principles, and that the model is fit to be utilized in future analyses by the research community. The openENTRANCE project has established a list of checkpoints that the models need to satisfy to be considered ready to become compatible with the requirements of the four freedoms and ensure that the coding is clear enough to be used afterwards by third parties to extend the functionalities or improve its performance.

2.2.1 Checkpoints for openness progression

In openENTRANCE we have defined a set of items to check whether the data and model are open and we have defined a table to monitor the state of openness of the different models over time. For what concerns the data, the project has required every model to be equipped with an interface making it able to interact with the openENTRANCE platform. Namely, every model operating around the common platform needs to write and read data in IAMC format, a data template developed by the Integrated Assessment Modelling Consortium and used in scenario analysis [13]. Moreover, each model's data handling has been adapted to consider possible aggregation and disaggregation routines to homogenize the internal data process and the input/output template in terms of granularity.

Besides complying with a common data structure for the communications with the scenario platform, the modelling teams have been required to adapt their models to satisfy the following requirements:

- Each model needs to be coded following the coding best practices, i.e., a set of rules that the scientific community adopts to make the code easily accessible for future development and use and, therefore, more adapted to be maintained and improved over a long period of

time. These rules also make the details of the model clearer to potential users, increasing the level of reliability of the models themselves. The coding best practices are further explained in Section 2.2.2.

- The models need to be provided with a user guide, describing the structure of the input data, the routines defining the model and the output results. Moreover, the user guide should contain a mathematical description of the model. The user guide needs to be made available as a web page, as a help included in the model or as a document stored with the model itself in a publicly accessible repository.
- The code of each model needs to be made available in a public repository,
- A stylised example of the input data (or a full-fledged case) should be made available together with the model.
- An example of the output results should be made available with the model.
- The model should be released under an open-source license.
- A video should be made to briefly describe the main features of the model and its usage.

The models have been going through a weekly revision to ensure that all these points are satisfied, keeping track of the progresses in the opening process by means of the following table, which displays the status at the moment of the writing of this document.

Table 2 Progresses in the opening process of the models

	Coding Best practices	User guide	Code available	Input data	Output results	Open-source license
EMPIRE	✓	✓ – GitHub	✓ - GitHub	✓ - GitHub	✓ - GitHub	MIT license
REMES EU	✓	✓ - GitHub	✓ - GitHub	✓ - Github	✓ - GitHub	APACHE 2.0 license
openTEPES	✓	✓ - paper	✓ - GitHub and webpage	✓ - GitHub and webpage	✓ - GitHub and webpage	GPL3 license
EXIOMOD	✓	✓ - Github	✓ - GitHub	✓ - GitHub	✓ - GitHub	Custom License. For the code, data comes with

						the EXIOBASE license
Plan4EU	✓	✓ - Zenodo and Gitlab	✓ - Gitlab	✓ - Gitlab	✓ - Gitlab	GNU Lesser General Public License version 3.0
GENESYS-MOD	✓	✓ - Gitlab	✓ - Gitlab	✓ - Gitlab	✓ - Gitlab	APACHE 2.0 license
GUSTO	✓	✓ - paper	✓ - GitHub	✓ - GitHub	✓ - GitHub	GNU license
FRESH:COM	✓	✓ - paper	✓ - GitHub	✓ - GitHub	✓ - GitHub	APACHE 2.0 license

The table columns are related to the achievement of models' openness and replicate the bullet points presented at the beginning of the section. Regular meetings have been held to monitor the process of opening the data and the model themselves.

2.2.2 Coding best practices

The openENTRANCE project requires every model to adapt its implementation based on a set of coding best practices. These practices aim at maximizing the capability of future collaboration and reusage of the models. In general, writing code should be made in a way to make it future-proof. Often, a lack of structure and commentary makes a code hard to read even for the original developer. Moreover, when coding, one needs to bear in mind that the largest share of time will be spent on maintaining and refining the code, rather than on writing the first prototype. For this reason, it is advisable to follow some basic rules when developing a code [5,7]. Before start coding one needs to define and design what he or she wants to develop. Poorly designed code usually takes more lines of instructions to perform the same tasks, often because of code duplication. An important aspect of improving design is to eliminate duplicate code. If the code is written to implement a mathematical model, then its implementation must come after a clear and understood representation in paper. It also is a good habit to read code written by experienced developers; this will allow to learn how to structure good code. Another good practice is to start writing simple and plain code and then improve progressively

it by including additional characteristics that make the code more complex. Moreover, one should start by developing a simplified version of the model to have it ready and debugged and tested with a case study before working on the full-scale version. Finally, it is important to protect the code against all the possible data values and special conditions. Commonly, starting and ending conditions in the time domain as well as checking the model behaviour for zero or negative values of some parameters must be carefully considered.

Software implementation can easily consist of thousands of lines of code. If this code is to be shared with other researchers, while ensuring a smooth transition of the workflow, it is paramount that this code is written by taking into account some principles to make it easier for others to understand and re-use the code. The code of open-source models should as much as possible be written using a modular structure, by breaking a lengthy workflow into pieces to make it easier to understand, share, describe, and modify it. Besides modularity, the coding best practices can be summarized by the following list [2]:

1. *Place a brief explanatory comment at the start of every program*
   Every module must have a brief explanation defining its purpose and input and output parameters. The comment should include at least one example of how the program is used.
2. *Decompose programs into functions*
   A function is a reusable section of software that can be treated as a black box by the rest of the program. The length of each function should not exceed 60 lines. Splitting the model in functions makes easier the development and reusability of those functions.
3. *Be ruthless about eliminating duplication*
   Avoid failures by eliminating the duplication of code snippets and data structures and substitute them by functions. Moreover, if possible and for standard operations, one should use well-maintained libraries that already perform the task instead of writing more code from scratch.
4. *Give functions and variables meaningful names*
   Using an organized convention for defining variables, parameters and functions avoid many future difficulties when modifying the code later. This helps documenting the purpose of each variable and function and make the overall program easier to read.
5. *Make dependencies and requirements explicit*
   Be consistent declaring dependencies among variables, parameters or modules of the code to avoid future misbehaviour. This can be done outside the program, by adding a text file (e.g., "`requirements.txt`") to the root directory of the project.
6. *Do not comment and uncomment sections of code to control a program's behaviour*
   This is an error prone use and makes it difficult or impossible to automate analyses. Instead, put if/else statements in the program to control what it does.
7. *Provide a simple example or test data set*
   Always use a small case study to check the model behaviour and the new developments.
8. *Submit code to a reputable DOI-issuing repository*

Finally, the way to make the model referenceable is getting a DOI for the source code. DOIs for software are provided by Figshare and Zenodo. Zenodo integrates directly with GitHub.

2.2.3 Stages in the upgrade of models to make them open

The opening process followed by the modelling teams in openENTRANCE bears some mutual similarities. All the models have gone through a code restructuring to make it easier for a potential user to read, understand and use the code. Most of the effort have been placed in reorganizing the code into modules, by defining different files, each dedicated to a particular role in the model. All the files are normally called by a main routine, which controls the behaviour of the software. The codes have been also equipped with comments to explain the operations of each module and the modellers have ensured that the variables follow a naming convention that makes it easy to understand. This has been done by assigning variables meaningful names and often including a description of the variable after its introduction in the code. The modellers have also made sure that data and model are clearly separated. Modelling teams have provided a user guide for each of the models, located in the same repository where the models are stored. Typically, these guides cover a description of the model and its uses, description of the input data, description of the model formulation and explanation of the output data. Some models also provide a "Quick user guide" containing the essential information for running the model.

For each model, an open-source license has been selected, based on the future intended use envisioned for the model. Considering the wide range of available types of licenses, the modelling teams have often used specialised websites providing aid for the selection of an appropriate one for their model. The website "Choose a License" (https://choosealicense.com/) has proved to be rather valuable in the process of selection. Modelling teams have often opted for Permissive licenses. These licenses are less restrictive when it comes to defining possible uses of future versions of their models. Among the different licenses, the Apache 2.0 license has been slightly preferred over other ones.

Each model has been stored into online repositories. Most of the modelling teams have stored their model and data in a GitHub repository. Two of the models can be found in a GitLab repository and another is in a dedicated repository of the institution where the model has been developed.

Even if this is not a requirement for an open model, some of the modelling teams have rewritten their code using alternative programming languages. Translating the code has often proved a time-consuming activity, sometimes requiring the code to be restructured, which has proved challenging, especially if the new programming language is lacking features that the previous language had and that made it simple to program a particular model implementation.

Opening the models have presented some challenges in defining the right trade-off for the amount of commentaries to make the code easy to read but not too verbose. Moreover, many modelling teams have found the definition of an appropriate user guide not always an easy task. This is due to the fact that having a large experience with a model often leads to overlooking the difficulties that it might present in terms of code readability. Also, the reorganization of the code based on modules is not always easy to carry out, especially if the code considers the inputs for the case studies in different blocks. In summary, the part presenting the largest number of difficulties is often defined by the implementation of the coding best practices. This is however a price that needs to be paid at the moment of defining the code, to be able to implement further developments in an efficient manner.

3. The opening process of the models

The suite populating the openENTRANCE initial framework consists of 13 models, each covering different aspects of the decarbonization policy analysis spanning from the macro-economic perspective to the analysis of the utilization of resources by local communities and prosumers, passing through models for the long-term development of the energy system. To engage in cooperation around the openENTRANCE platform, 9 of these models have gone through an opening process to ensure future usability by the interested research communities and interoperability with other models. The remaining models (namely SCOPE, EMPS-W and e-TRANSPORT) are protected by industry rights and they have not gone through the opening process, even though they have been used in openENTRANCE case studies. Finally, the Frigg model was not originally planned to be a part of the openENTRANCE modelling suite. It has recently been included but is not opened as part of this deliverable. The Frigg model will probably be opened in a later stage of the project. Besides the opening process, the models will be subject to progressive updates and, as part of the project activities, there will be the definition of an alert system to warn subscribers of the platform when a new version of a model is released.

There are three classes of models that have operated in the openENTRANCE project: the first group has been natively designed as open, the second is the group of models that have been opened as part of the project and finally, a third class is composed of the models that have not gone through any opening process. The models belonging to the first group were already equipped with a suitable open license. In this case the modelling teams have mainly worked in fulfilling the FAIR principles, while the second group has been subject to the entire sequence of steps described in the previous chapter, namely ensuring that the code implementation has been following the coding best practices, making the code, the input data and an instance of the

output available in a repository, defining the user guide and finally, equipping the model with a suitable open-source license. In this chapter we report the effort of each modelling team in opening their model and ensuring their adequacy and compliance with the FAIR principles.
The status of each model in openENTRANCE is summarized in the following table

Table 3 Status of the open models

Model name	Status of Model	Use of proprietary language or software
EMPIRE	Transformed into an open model	No proprietry element
REMES	Transformed into an open model	Proprietary language to process information (VBA) Proprietary modelling software (GAMS)
openTEPES	Transformed into an open model	No proprietary element
EXIOMOD	Transformed into an open model, but not an open-source model, is custom license model	Coding language is proprietary (GAMS). Matlab is used for processing data, representing that in the common format.
Plan4EU	Designed as an open model	No proprietary element, except the MILP solver which can be Cplex, SCIP or Coin-OR
GENESYS-MOD	Built as an open model	Using GAMS proprietary language
GUSTO	Built as an open model	No proprietary element
FRESH:COM	Transformed into an open model	No proprietary element
Frigg	To be made open at a later stage	To be made open at a later stage

3.1 Modelling teams experience in the opening process

In the following tables we report, for each modelling team, a schematic summary of their experience in the process of adapting their model to the requirements of openENTRANCE. A table has been devised for each modelling team, and within the table the considered model will be briefly introduced, listing the original developing team and the model initial status related to the open-source requirements. This will be followed by a description of the steps followed by the modelling team in making sure that their model complies with the coding best practices, then by a small description of the structure of the user guide and by the decision on the chosen open-source license. Next, there will be a pointer to the repository storing the code and the data for each model. Finally, each modelling team has included comments on relevant difficulties and obstacles (if any) encountered in the process of making their model open. The following tables summarize the efforts made by the modelling teams for opening their models.

openTEPES: Open Generation and Transmission Operation and Expansion Planning Model with RES and ESS	
Model introduction	The openTEPES model is a decision support system for defining the generation and transmission expansion plan of a large-scale electric system at a tactical level, defined as a set of generation and network investment decisions for future years. The expansion candidates, generators, ESS (Energy Storage System) and lines, are pre-defined by the user, so the model determines the optimal decisions among those specified by the user. It determines automatically optimal expansion plans that satisfy simultaneously several attributes. Its main characteristics are: • **Static:** the scope of the model corresponds to a single year at a long-term horizon, 2030 or 2040 for example. • **Stochastic:** several stochastic parameters that can influence the optimal generation and transmission expansion decisions are considered. The model considers stochastic medium-term yearly uncertainties (scenarios) related to the system operation. These operation scenarios differ with respect to the level and profile of the renewable energy sources generation output and the electricity demand. The objective function considers the two main quantifiable costs: 1) generation and transmission investment cost (CAPEX); and 2) expected variable operation costs (including generation

	emission and reliability costs) (system OPEX). The model formulates a stochastic optimization problem including generation and network binary investment decisions and operation decisions (commitment, startup and shutdown decisions are also binary). The operation model is a network constrained unit commitment (NCUC) model based on a tight and compact formulation if the problem including operating reserves and a DC power flow (DCPF). Network ohmic losses are considered proportional to the line flow. It considers different energy storage systems, e.g., pumped-storage hydro, battery, etc. openTEPES allows computing the optimal trade-off between the investment in generation/transmission and the investment or use of storage capacity. The openTEPES model has been developed using Python 3.8.6 and Pyomo 5.7.3 and it uses Gurobi 9.1.1 as a commercial MIP solver for which a free academic license is available. Using Pyomo makes the model definition independent from the choice of the preferred solver. Other solvers can be used to compute a solution for the instance of the problem, including free solvers such as SCIP, GLPK 4.65 and CBC. It has been developed by Andres Ramos, Erik Alvarez and Sara Lumbreras, from Instituto de Investigación Tecnológica, Escuela Tecnica Superior de Ingenieria - ICAI UNIVERSIDAD PONTIFICIA COMILLAS. Alberto Aguilera 23, 28015 Madrid, Spain Andres.Ramos@comillas.edu Erik.Alvarez@comillas.edu https://pascua.iit.comillas.edu/aramos/Ramos_CV.htm It is based on the previous commercial TEPES model, and has been opened as part of the openENTRANCE project. The openTEPES does not have all the functionalities of the original TEPES model, but it includes the basic functions to respond to the needs of the project.
Steps followed on the definition of the coding best practices	The model has been developed with a “Simplicity and Transparency in Power Systems Planning” idea to promote an easier manipulation and understanding of its structure. For this reason, we have followed the recommendations of this

	document to make the model easy to understand and, consequently, to modify. Every parameter, variable, objective function and constraint was named in a user-readable way as well as each of them has a comment with a short explanation. Following the best-practices of open-sources, the openTEPES model was split into the following modules: • openTEPES.py: Main script to select the test case and solver. • openTEPES_InputData.py: This script reads the input data from CSV files and processes the data in the adopted format to define parameters and variables. • openTEPES_ModelFormulation.py: The objective function and constraints are defined. • openTEPES_ProblemSolving.py: This script allows us to select the solver options and sequence to solve the model. • openTEPES_OutputResults.py: To write results in a suitable format and save it into CSV files for subsequent user analysis. Moreover, to ensure a clear separation between the model and its data the model was structured so that both the results and the input data are in CSV format, are read and written using separate scripts, and different from the model formulation.
A small description of the user guide	The user guide was published on a website that can be found in the following link: https://pascua.iit.comillas.edu/aramos/openTEPES/index.html The guide is composed of 7 sections: introduction, input data, output results, mathematical formulation, research projects, download, and contacts. It describes the model and its functionalities and the mathematical formulation, and relevant results that can be obtained. Moreover, research projects, publications, and links to contact for additional information and download the model and test case are considered.
Chosen open license	Taking into account the purposes of the model, the GNU General Public License 3.0 was selected. More details about the license can be found at https://www.gnu.org/licenses/gpl-3.0.html
Where is the model stored and how can a	The code and a small test case can be downloaded from the user guide using the link:

potential user get access and download it	https://pascua.iit.comillas.edu/aramos/openTEPES/index.html under the section "Download" and also from GitHub https://github.com/IIT-EnergySystemModels/openTEPES.
Relevant difficulties, or obstacles, found in the development of the model as an open one	Our research group background has a strong and large experience on modelling in GAMS, but making the model open, it was a challenge because we had to learn new languages, Python for general data manipulation and Pyomo for writing the optimization problem.

GUSTO	
Model introduction	GUSTO is a mixed-integer linear program that optimizes both the energy technology investment decision (portfolio optimization) and the energy technology dispatch on a local or neighbourhood level. The framework gives the following objective functions: (a) minimizing total (annual) costs of supply, (b) minimizing total greenhouse-gas emissions, (c) maximizing local self-consumption, and (d) scheduled generation compliance within the neighbourhood. It is a tailor-made extension of the existing open-source model urbs. The Technical University of Munich initially developed the latter. The functional extension of the model was carried out at the Vienna University of Technology by Sebastian Zwickl-Bernhard (zwickl@eeg.tuwien.ac.at). The model is written in Python 3.6 and Pyomo 5.6.7 and uses Gurobi 9.1.0 as a commercial MIP solver for which a free academic license is available. However, a free solver such as GLPK can be used since the Pyomo framework is independent of solver choice. GUSTO has been opened as part of the openENTRANCE project.
Steps followed on the definition of the coding best practices	Comments are added at the beginning of a code block or in the corresponding function documentation (this applies to the mathematical model in Pyomo) to ensure the readability. The model is divided into several modules (e.g., an individual python script *model.py* for the mathematical framework or *plot.py* for the visualization of the results). Every parameter, variable, objective function, and constraint was named in a

	user-readable form as well as each of them has a comment with a short explanation. The data (.xlsx files) and the model (.py scripts) are clearly separated.
A small description of the user guide	The user guide can be found in the GitHub repository under the following link: https://github.com/sebastianzwickl/GUSTO. Note that two documentations form the user guide. On the one hand, the comprehensive description of the underlying model *urbs*. On the other hand, two recent scientific publications regarding the tailor-made extension of the model.
Chosen open license	For GUSTO, the GNU General Public License v3.0 has been selected.
Where is the model stored and how can a potential user get access and download it	The model framework is available at GitHub, and can be found at the following link: https://github.com/sebastianzwickl/GUSTO
Relevant difficulties, or obstacles, found in the development of the model as an open one	As mentioned above, GUSTO is an extension of an existing open-source model. Thus, there are already several experiences on possible difficulties in the development of open-source models, which concern the following points: • The perspective of the developer of the model when developing the user guide may result in this user-guide not being understandable and intuitive to third parties. • Comments in the source code increase the readability of the code, but can also make it messy at the same time. • Besides, some difficulties can arise in the course of third parties using the model. These must be taken into account (e.g., obstacles in setting up the appropriate environment necessary to perform initial model runs).

FRESH:COM FaiR Energy SHaring in local COMmunities	
Model introduction	FRESH:COM is a linear optimization model that allocates electricity in local energy communities via peer-to-peer trading. The allocation mechanism considers (i) the minimization of grid purchases by the community (e.e. of an external supplier),

	and (ii) the willingness-to-pay of the individual prosumers and thus the objective function maximizes the community's social welfare over a whole year. The model considers decentralized photovoltaic (PV) systems and battery energy storage systems (BESS) of prosumers. FRESH:COM was developed by TU Wien, at the Department of Energy Systems and Electrical Drives, Energy Economics Group (EEG). The main contributor is Theresia Perger (contact perger@eeg.tuwien.ac.at). The model is implemented in Python 3.7.2 as a Jupyter notebook 6.1.1, using the Pyomo environment 5.6.6 and Gurobi 9.0.1 as a solver. While Python, Jupyter, and Pyomo are open-source tools, Gurobi is commercial, but can be used with an academic license free of charge. However, any other solvers of choice can be used too. Opening FRESH:COM was part of the openENTRANCE project, as there was a previous, non open-source version of the model.
Steps followed on the definition of the coding best practices	The open-source code available publicly on GitHub is implemented as a Jupyter notebook (.ipynb file). The advantage of coding notebook-style is that the code can be arranged in blocks embedded between text, tables, and figures explaining the method step by step. The notebook shows a small use case to demonstrate the functionality of the model. Variables are explained in the nomenclature at the beginning of the Jupyter notebook. The variable names were chosen so that they are understandable for human readers and in line with the nomenclature of the user guide. The code is arranged in a block structure, with text and comments at the beginning of each block. For better readability, data can be shown in table format. The GitHub repository includes input data as .xlsx files, separated from the model itself. Users of FRESH:COM can easily use their own input data, which have to be converted to the right format first.
Small description of the user guide	The Jupyter notebook, where the model is implemented in,

	contains a user guide and demonstrates a small use case. Further, for details of the model, it is referred to the main user guide: T. Perger, L. Wachter, A. Fleischhacker, H. Auer, PV sharing in local communities: Peer-to-peer trading under consideration of the prosumers' willingness-to-pay, In: *Sustainable Cities and Society* (2021), DOI: https://doi.org/10.1016/j.scs.2020.102634 The main topics covered in the user guide are: Methodology and mathematical description of the model, validation of the model, and results of a case study including sensitivity analysis. Description of input and output data are included as well.
Chosen open license	Following the best-practice of many open-source models, the Apache 2.0 license was selected. The Apache 2.0 license details can be found at https://www.apache.org/licenses/LICENSE-2.0
Where is the model stored and how can a potential user get access and download it	The model and input data can be found on FRESH:COM's public GitHub repository at https://github.com/tperger/FRESH-COM. There is no DOI for the code yet.
Relevant difficulties, or obstacles, found in the development of the model as an open one	In the case of FRESH:COM, an existing model has been opened and the following difficulties were faced: All parts of existing code must be reviewed and annotated with comments, some variables had to be renamed and some structural changes were implemented.

GENeSYS-MOD	
Model introduction	The Global Energy System Model (GENeSYS-MOD) is a cost-optimizing linear program, focusing on long-term pathways for the different sectors of the energy system, specifically targeting emission targets, integration of renewables, and sector-coupling. The model minimizes the objective function, which comprises total system costs (encompassing all costs occurring over the modelled time period). It builds upon the well-established Open-Source Energy System Model (OSeMOSYS) and was developed by researchers at TU Berlin (contact: genesysmod@coaltransitions.org). The model is written in GAMS and requires proprietary LP and DNLP solvers to solve. For solvers such as GUROBI, an academic

	license can be acquired but every other commercial solver can as well be used. While GENeSYS-MOD was designed to be open-source and available before the openENTRANCE project, a GitLab including a more sophisticated user guide featuring a quick start guide and a toy data set was developed during the project.
Steps followed on the definition of the coding best practices	Comments are added at the beginning of a code block as well as in the header of all input data sheets to improve usability. Additionally, all sets, parameters, variables, equations and files are named in user-readable ways. The code is divided into different input files which all serve a distinct model function (e.g., loading the data, defining the equations or handling the result processing). The data (.xlsx files) and the model (.gms scripts) are clearly separated. Also, a readme.md containing all basic information is found in the main folder.
A small description of the user guide	The user guide can be found in the GitLab repository under the following link: https://git.tu-berlin.de/genesysmod/genesys-mod-public/-/tree/main/Docs) Note that two documentations form the user guide. On the one hand, an extensive "Technical Manual" describing in detail all requirements and components of GENeSYS-MOD, including a description of a toy data set which can serve to validify the correct implementation of the model for other users. On the other hand, a "Quickstart Guide" which condenses the essential information and steps which have to be taken to start a successful model run.
Chosen open license	GENeSYS-MOD is published under the Apache 2.0 license (http://www.apache.org/licenses/LICENSE-2.0).
Where is the model stored and how can a potential user get access and download it	The model framework is available at GitLab, and can be found at the following link: https://git.tu-berlin.de/genesysmod/genesys-mod-public
Relevant difficulties, or obstacles, found in the development of the model as an open one	As mentioned above, GENeSYS-MOD is an extension of an existing open-source model. Thus, there are already several experiences on possible difficulties in the development of open-source models, which concern the following points:

	• The perspective of the developer of the model when developing the userguide may result in this user-guide not being understandable and intuitive to third parties. • Comments in the source code increase the readability of the code, but can also make it messy at the same time. • Besides, some difficulties can arise in the course of third parties using the model. These must be taken into account (e.g., obstacles in setting up the appropriate environment necessary to perform initial model runs).

REMES:EU Regional Economic Model with focus on the Energy System	
Model introduction	REMES:EU is a Computable General Equilibrium model used to simulate the activities and money flows in the European economy as a consequence of a particular policy or shock. The model computes sectoral value added, overall GDP level, prices and other macroeconomic values by ensuring the equilibrium between supply and demand of each commodity considered in the economic system. The model has been developed as a joint project between SINTEF, department of Technology and Society and NTNU, department of Industrial Economics and Technology Management. The model has been developed by Gerardo A. Perez-Valdès (GerardoA.Perez-Valdes@sintef.no), while at NTNU, Paolo Pisciella (paolo.pisciella@ntnu.no) has provided a set of extensions. The model is implemented in GAMS 24.7.3, using the MPSGE framework, a language used for the formulation of Arrow-Debreu economic equilibrium models. The model is structured as a Mixed Complementarity Problem and is solved using the Path solver version 4.7.04. REMES:EU has been opened as a part of the openENTRANCE project.
Steps followed on the definition of the coding best practices	The model has been restructured in a modular way. It is composed of three basic routines: a main routine, handling the general operations, an input file defining the value of the user defined parameters, and a model file containing the model itself. Besides, a module is used to initialize the model variables

	and a module includes initial values for GDP and population growth. Variables are explained when they are defined in the code and their names have been chosen to reflect, to the largest extent possible, their meaning. The code has a block structure with each block considering a different task. Comments are present at the beginning of each block.
Small description of the user guide	The user guide consists of 4 sections. The first section introduces the model and its typical uses; the second section explains the structure of the used data, the third section defines the CGE model and explains the code structure; the fourth section contains complementary information and a list of the main model parameters. The user guide is available, together with the model, in the GitHub repository https://github.com/paopis/REMES-EU
Chosen open license	The Apache 2.0 license was selected for REMES:EU. This license is Permissive and allows future commercial uses. More details can be found on the link https://www.apache.org/licenses/LICENSE-2.0
Where is the model stored and how can a potential user get access and download it	The model code and input data are stored on the GitHub repository https://github.com/paopis/REMES-EU
Relevant difficulties, or obstacles, found in the development of the model as an open one	The main difficulties in opening the REMES:EU model have been related to the restructuring of the model to follow the best practices. The implementation of shocks to the initial state is coded in several parts of the model and it is not immediately clear how to bundle all the shocks in one single area of the code. This reorganization has been done, to the extent possible, by defining an external file with all the input parameters.

EXIOMOD	
Model introduction	EXIOMOD is an economic model able to measure the environmental and economic impacts of policies[1]. As a multisector model, it accounts for the economic dependency between sectors. It is also a global and multi-country model with consistent bilateral trade flows between countries at the detailed commodity level. Based on national account data, it can provide compressive scenarios regarding the evolution of key economic variables such as GDP, value-added, turn-over, (intermediary and final) consumption, investment, employment, trade (exports and imports), public spending or taxes. Thanks to its environmental extensions, it makes the link between the economic activities of various agents (sectors, consumers) and the use of a large number of resources (energy, mineral, biomass, land, water) and negative externalities (greenhouse gases, wastes). The model is written in GAMS and requires a license for the PATH solver. In order to transform GAMS output into excel files in the format for the OpenEntrance platform, an additional license for Matlab is required. The model is a custom-license model. That is, the model is openly available for everyone. However, for using it for commercial purposes, a license is needed.
Steps followed on the definition of the coding best practices	The code is divided into different input files. Each input file contains on top a clear description of the purpose of that file. Also, all variables, equations, parameters and sets are thoroughly described in the script files. The code is divided into different folders, each having their own function. This is also clearly described in the user-guide of the model. The base code – which should in principle not be changed - is places in a separate folder from the code that could be adjusted by the user.
A small description of the user guide	The user guide will be found in the Gitlab repository under the highest folder under the name: “Open-Source-EXIOMOD-

	getting-started.docx". The user guide describes: - How to install the model to your computer. - Which files to run for running the model. In which order the model runs the files, and what is the purpose of each file in the model. - The folder structure of the model. Which files need to be changed in order to tune the model for your project.
Chosen open license	EXIOMOD is a custom-license model. It is available for everyone with a GitHub account. However, when using for commercial purposes, a license is required. Contact hettie.boonman@tno.nl for a license.
Where is the model stored and how can a potential user get access and download it	EXIOMOD is placed on a publicly available GitHub repository: www.github.com/TNO/EXIOMOD-open
Relevant difficulties, or obstacles, found in the development of the model as an open one	EXIOMOD was initially not developed as an open-source model. The model that is publicly available for the audience uses EXIOBASE 3.0, which also contains an underlying license (Exiobase - Terms of Use.). This license also holds for users of the model EXIOMOD. We published a full working model on a GitHub account; however, we might slightly aggregate the underlying database.

plan4EU	
Model introduction	Plan4EU is a modelling suite comprising 3 nested layers: I) at capacity expansion model (CEM) for generation/storage investment and transmission/distribution expansion (not yet implemented), ii) a seasonal storage valuation (SSV) model and iii) a European operational dispatch model (Unit Commitment EUC). The CEM is based on Benders decomposition, SSV on SDDP and EUC on Lagrangian relaxation or MILP. It is focused on electricity systems comprising generation/storage and demand-response assets, and a network. Time and geographic

	granularity are parameters (usually 1 or more years, hourly resolution, Europe/ country resolution but it may be downscaled to regional resolution). It takes into account uncertainties (demand, hydro inflows, availability of assets, maximum generation of PV and Wind Power...) It is implemented within the C++ SMS++ modelling and optimisation library developed by University of Pisa, in the plan4res project. Optimisation solvers used are: • For solving MILPs: any commercial or open-source solver such as Cplex, SCIP or Coin-or. • The open-source Stopt library (developed by EDF) • The suite of Solvers implemented at UniPi : Non-Differentiable Optimization solvers in particular generalized Bundle algorithms, Dynamic programming solvers.
Steps followed on the definition of the coding best practices	The sources are managed on Gitlab (gitlab.com/smsp, gitab.com/stochastic-control), the whole documentation is managed via DOxygen and available on the Gitlab, as well a testing database and environment
A small description of the user guide	The main model equations are available in https://zenodo.org/record/3904272#.YBBnfuhKjIU The exhaustive source documentation is available in https://gitlab.com/smspp/doxygen-env
Chosen open license	SMS++ will be published under the GNU Lesser General Public License version 3.0
Where is the model stored and how can a potential user get access and download it	The model is available at the link https://gitlab.com/smspp
Relevant difficulties, or obstacles, found in the development of the model as an open one	No particular difficulties have been faced in the opening process of plan4EU.

EMPIRE	
Model introduction	The EMPIRE model has been developed using Pyomo and it can use solvers such as Gurobi and CPLEX. EMPIRE is a power system model including generation, storage, and transmission capacity expansion. It is designed to determine optimal capacity investments under operational uncertainty, while also incorporating long- and short-term dynamics. To achieve these objectives, EMPIRE is a stochastic linear program endogenously considering uncertainty on an hourly operational resolution of (1) nationally aggregated load and (2) availability of variable renewable supply. The model considers net transfer capacity (NTC) of power exchange between countries, up-ramping constraints for generators and investment and operation of storage technologies. EMPIRE has three key advantages in contrast to other power sector models. The first is the special handling of challenges given by the variability of renewable technologies, in particular with wind and solar power, which highly impacts the supply and demand balance. Another major contribution of EMPIRE is that it simultaneously incorporates short- and long-term dynamics, in conjunction with short-term uncertainty. Dynamics refer to multiple investment periods coexisting with multiple sequential operational decision periods, while uncertainty is enhanced through multiple input scenarios that captures different operating conditions. Lastly, EMPIRE uses representative time periods (days or weeks) with an hourly resolution within each investment period to preserve computational tractability. The team at NTNU started developing EMPIRE in 2010. The development started as part of Christian Skar's doctoral degree, and it has been developed and expanded by several researchers since then. The current research team is continuously improving it to maintain topicality, applicability, and efficiency. Therefore, EMPIRE contains recent research results, up-to-date data and a clean code basis.
Steps followed on the definition of the coding best practices	EMPIRE is a capacity expansion model formulated as a stochastic programming optimization problem. The code is divided into two parts include scenario generation algorithm “scenario_random.py” and core part to make instances “Empire.py”. In addition, the model uses “run.py” in order to

	control the number of stochastic scenarios and run the model. The model has a folder called "Data handler" that includes all necessary data and also after running the model it is possible to get results in a folder called "Results" (.csv files).
A small description of the user guide	The user guide was published on a website that can be found in the following link: https://github.com/openENTRANCE/linkages/tree/main/EMPIRE/doc
Chosen open license	EMPIRE is licensed under the MIT License.
Where is the model stored and how can a potential user get access and download it	The code can be downloaded from the GitHub repository at: https://github.com/ntnuiotenergy/OpenEMPIRE
Relevant difficulties, or obstacles, found in the development of the model as an open one	No particular difficulties have been faced in the opening process of EMPIRE.

4. Discussion & Conclusions

The objective of this document has been twofold: providing an explanation of the necessary steps needed to define an open-source model and report the experiences by the openENTRANCE modelling teams in opening the models they have developed and are maintaining. The experiences brought by the modelling teams have also served as the basis to summarize the typical efforts that have been made to open the models and to discuss about the typical actions and decisions taken. Opening the models has not only been limited to selecting a suitable license, but it has extended to a process focused on preparing the model code to be readable and upgradable by third parties with the lowest possible effort. The openENTRANCE project requires open models to be Findable, Accessible, Interoperable and Reusable, which implies that the definition of an open-source model should in principle not only be limited to the choice of a suitable license, but every model needs to be prepared to abide to the FAIR principles. These requirements have been fulfilled by restructuring the code of the involved models in order to satisfy a set of rules, called best coding practices, which require to write the code in a modular way and ensuring that each code block is properly commented. The codes have been placed in an online repository together with an example of their input data and examples of output results. In addition, every model has been equipped with a user guide, to make their implementation and use more transparent. Finally, the problem of the selection of an open-source license has been explored in some detail, both by providing a detailed exposition of the most widely used licenses and by reporting the experiences of the modelling teams in choosing a license.

This deliverable can serve as a basis both for project stakeholders and for third parties, such as modelling teams interested in engaging in collaboration with the openENTRANCE platform also beyond the lifespan of the project, to understand the steps that have been taken to make the models suitable for a transparent interaction around a common platform. This will give researchers and modelers the possibility to use the models that have been included in the initial suite of tools operating around the openENTRANCE platform and facilitate future interactions between the created platform and third parties interested in exploiting its potentialities using their own models.

5. References

[1] PEP 20 – "The Zen of Python," (https://www.python.org/dev/peps/pep-0020/).

[2] G. Wilson et al. "Good Enough Practices in Scientific Computing," Oct 2016 (arxiv.org/abs/1609.00037).

[3] M. Wilkinson, M. Dumontier, B. Mons "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific Data*, March 2016.

[4] D. Huppmann, "Principles of open-source and collaborative scientific programming for energy modelling," (https://data.ene.iiasa.ac.at/teaching/).

[5] A. Ramos, "Good optimization modeling practices with GAMS" (https://pascua.iit.comillas.edu/aramos/simio/transpa/s_GoodOptimizationModelingPractices.pdf).

[6] Lin et al., "Open Source Licenses and the Creative Commons Framework: License Selection and Comparison," *Journal of Information Science and Engineering,* 22(1), 1-17, 2006.

[7] B. W. Kernighan and P. J. Plauger, "The Elements of Programming Style," *McGraw Hill, New York*, 1978 .

[8] S. S. Levine, M. J. Prietula., "Open Collaboration for Innovation: Principles and Performance," *Organization Science*. 25 (5): 1414–1433, 2013.

[9] E. S. Raymond, "The cathedral and the bazaar: musings on Linux and Open Source by an accidental revolutionary,". *OReilly*. ISBN 978-0-596-00108-7, 2001.

[10] F. Almeida, J. Oliveira and J. Cruz, "Open standards and open source: enabling interoperability," *International Journal of Software Engineering & Applications (IJSEA),* 2(1), 1-11, 2011.

[11] S. Day O'Connor, "Copyright Law from an American Perspective". *Irish Jurist*. 37: 16–22, 2002.

[12] T. Preston-Werner, "Semantic Versioning 2.0.0. Creative Commons," https://semver.org/spec/v2.0.0.html, 2013.

[13] D. Huppmann, J. Rogelj, E. Kriegler, V. Krey and K. Riahi, "A new scenario resource for integrated 1.5 C research," *Nature climate change*, 8(12), 1027-1030, 2018.

[14] P. A. David and S. Greenstein, "The economics of compatibility standards: an introduction to Recent Research," *The Economics of Innovations and New Technology* (1:1/2), 3-41. 1990.