Directorate - General for Informatics
Directorate B – Digital Business Solutions
DIGIT.B.3 – Reusable Solutions

The European Commission's Open Source Software and Tools Inventory Methodology

Revised Version (Nov 2021)

Date: 03/11/2021
Doc. Version: Final

TABLE OF CONTENTS

TABLE OF FIGURES

1. DELIVERABLE OVERVIEW

1.1. Introduction

Building on top of the experience of creating two inventories around Open Source Software, processes and tools – initially in 2016, followed by its update two years later – this document, commissioned by DIGIT, the central IT functionary of the European Commission, attempts to offer a generic and replicable methodology for building and maintaining an inventory of free and Open Source Software, processes and tools.

The aim is for European public services and the private sector to benefit from this simplified presented methodology, and to provide the incentives for them to update and strengthen it, assisting in its future use and adoption by other entities.

The presentation and explanation of the generic methodology begins in Section 2 of the present document, which briefly defines the term “Open Source Software” and describes, concisely, the evolution of the methodology up to the optimal and ideal methodology (target scenario).

From Section 3 onwards, the methodology itself is then explained and described in detail and illustrated through examples related to the performance of projects under different European Commission initiatives in relation to open source.

2. BACKGROUND

2.1. Open Source

For the definition of Open Source Software (OSS), we refer to the Open Source Initiative (OSI) – https://opensource.org/osd, which states what OSS is and what criteria need to be met for it. Since this document is only about open source, the term is implied in multiple places when not written explicitly. Therefore, within the document, the term "software" has to be understood as "Open Source Software".

Additionally, in the context of the projects performed under different European Commission initiatives in relation to open source, OSS is defined as a computer software that is released under a licence, in which the copyright holder grants users the rights to use, study, change, and distribute the software and its source code to anyone and for any purpose. OSS might then be developed in a collaborative public manner. Open Source software is, therefore, a prominent example of open collaboration, meaning that, any capable user is able to participate online in its development, thus, making the number of possible contributors and iteration/version indefinite. Furthermore, the ability to examine the code facilitates public trust in the software[1].

2.2. Inception and evolution of the inventory methodology

The European Commission's open source inventory methodology was originally created as part of the 2016 EU-FOSSA Pilot project[2], which was formed in the wake of the Heartbleed[3] bug ***to assess potential security flaws hidden within the most critical Open Source Software that the European Commission was using***.

The pilot project was followed, in 2017, by a larger preparatory project, namely EU-FOSSA 2[4], which used the methodology to create a second version of the inventory. It was also used to create a fresh inventory for the European Council.

During these inventory exercises there were, essentially, three tasks executed:

1. Collect information about all the Open Source Software and tools in use.
2. Clean and group the data, apply filters and assessment criteria.
3. Establish a final inventory of the top 50 and 100 software in use, ordered by criticality[5].

[1] https://en.wikipedia.org/wiki/Open-source_software

[2] https://joinup.ec.europa.eu/collection/eu-fossa-2/solution/eu-fossa-pilot

[3] https://heartbleed.com/

[4] https://joinup.ec.europa.eu/collection/eu-fossa-2

[5] See section 3.7 and 3.8 for how to identify criticality.

This process allowed the European Commission to identify the Open Source Software it most used and relied upon – i.e., its most critical Open Source Software.

This, therefore, allowed the Commission to protect the identified software via security audits amongst other things.

2.3. Developing the Methodology

During its journey, from the pilot project to today, the methodology is undergoing significant improvements and has acquired a certain degree of maturity in the three areas presented below:

1. **Software Components Data Collection**: Represents the processes, tools and techniques to collect the component inventory and the mechanisms to process the consolidation of data with improved speed and accuracy.

2. **Assessment Metadata Collection**: Represents the processes, tools, and techniques to collect data for assessment of software such as "sustainability[6]" data (e.g., using information about communities behind the specific OSS), "vulnerability" data (about the known defects within the software) and "business criticality" data (as measured by the organisation).

3. **Filtering and Ranking**: Represents the processes, tools, and techniques to interactively filter/prioritise the inventory based on a set of criteria/thresholds.

The figure below summarises the progressive maturity improvements of the core aspects:

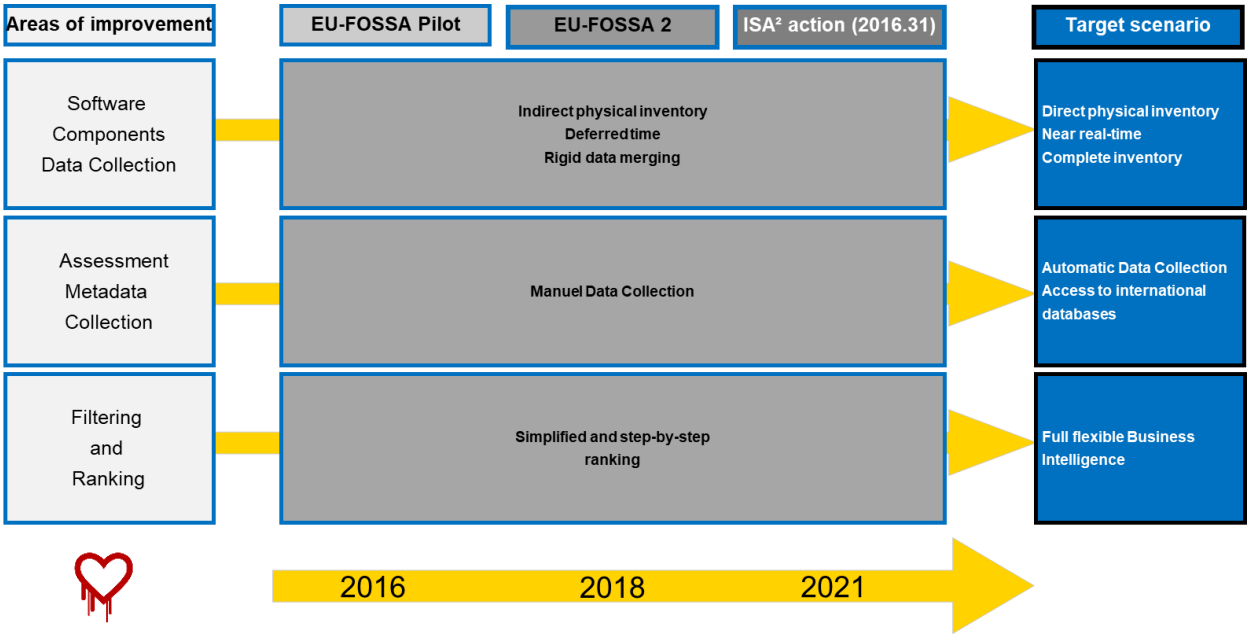


Figure 1: Evolution of inventory process from pilot to the optimal methodology (target scenario)

[6] See section 3.7 to understand how to establish the sustainability criteria.

As depicted in the above figure, the optimal methodology should be reached through the improvements of the three identified areas, namely, the software component data collection, the assessment metadata collection and the filtering and ranking processes.

The software component data collection, the assessment metadata collection and the filtering and ranking processes have been already improved across the execution of projects under different European Commission initiatives in relation to open source.

3. METHODOLOGY

From this section onwards, the document describes the methodology developed by the European Commission for creating an inventory of Open Source Software, processes, and tools.

3.1. Why create an inventory

What are the benefits for an organisation to spend considerable time and money to create an inventory of Open Source Software?

Most organisations use a software catalogue – or inventory – of proprietary software which comes with annual licence renewals and/or software support. Therefore, from one point of view, it makes sense to also keep an inventory of Open Source Software which has a series of extra benefits as we'll see further below. At a minimum, for OSS too, organisations ought to manage their open source for:

1. Licence compliance;
2. Support contracts.

Additional reasons and benefits

There are several reasons and benefits for creating an inventory of Open Source Software, processes and tools. Some of these include:

- Identification of what type of software is being used, where and by whom.;
- Identification of which software is critical for the organisation, and whether this software is sustainable[7].
- Identification of whether the critical software and related applications are well supported via support contracts.
- Identification of core/critical software for the organisation, which must be screened for security vulnerabilities (e.g. the Heartbleed bug, which remained undetected, caused over €600m in worldwide damage).
- Opportunity to understand the value of Open Source Software to the organisation, and to assess the reciprocal contribution the organisation makes to the open source community or eco-system.
- Opportunity to examine procurement policies in relation to open source.
- Possibility to highlight the usage of open source and examine whether it is used properly, or whether improvements need to be made internally.

[7] Sustainability here refers to the health of the software in terms of its community, e.g. if the software has only one person looking after its core development, then clearly its healthy continuity or sustainability would be questionable.

- Visibility on the associated open source processes, tools and frameworks relating to the software (and not just on the software itself).
- Improved understanding of what is in use and how transparent it is to the public with the generated code.
- Opportunity to consolidate the business applications built using open source. This would be a different sort of inventory or catalogue (e.g., PHP is an Open Source Software, but a corporate website or HR system built with PHP would be a business application. Potentially, an organisation can have an OSS inventory, and an open source applications catalogue).
- Reuse of software solutions already built and tested, lowering significantly any potential costs in unnecessary new-builds.

3.2. The methodology at a glance

The methodology has seven key steps:

1. Identify data sources
2. Acquire data
3. Consolidate and load into a database
4. Analyse, clean and enrich the data (e.g. duplicates, names, versions, dependencies and grouping)
5. Apply the business criticality and vulnerability criteria
6. Apply software sustainability criteria
7. Produce final results/reports

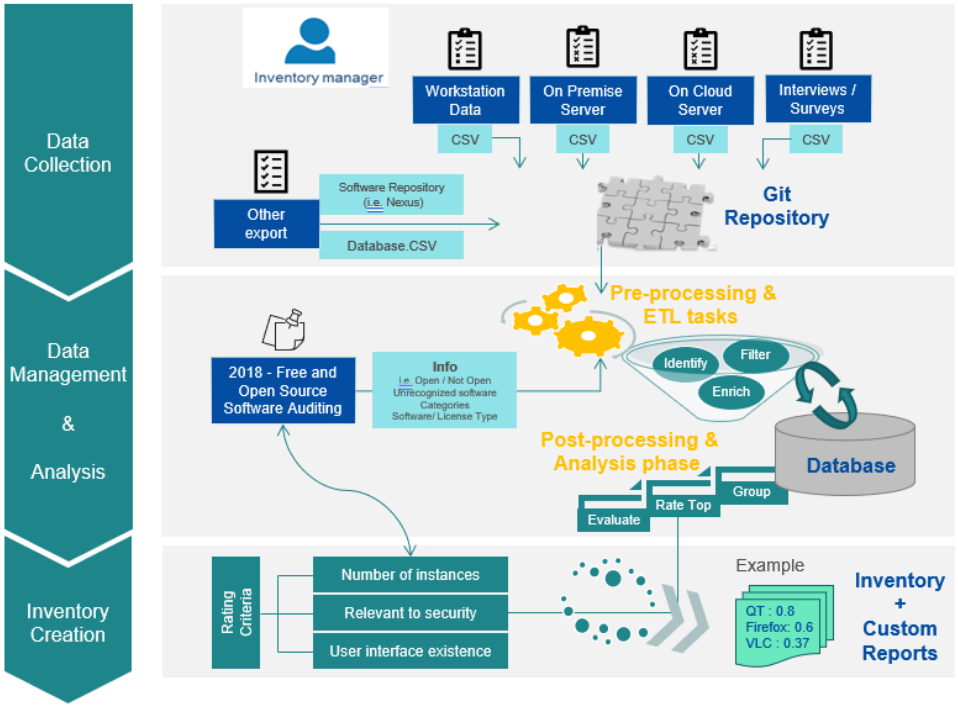


Figure 2: Methodology Overview diagram

3.3. Step 1: Identifying Data Sources

Step 1/7

Depending on the size of the organisation, Open Source Software data can be found in the following places:

- Data Centre
- Departmental systems/servers
- Virtual machines
- End user PCs
- Developer PCs
- Cloud systems
- Mobile devices
- Network switches, routers etc.

The types of data can be equally widespread to include:

- Operating systems such as GNU/Linux (various distributions).
- Applications running on servers for performance, messaging, email and connectivity.
- Software development tools and frameworks.
- User desktop tools such as web browsers, utilities, office suites, password managers; etc.

3.4. Step 2: Acquire data

Step 2/7

Each of the identified systems would been requested to provide outputs via a CSV file or other extracts. The multiple CSV (Comma-Separated Values) files will then be, during step 3, consolidated and uploaded into a database.

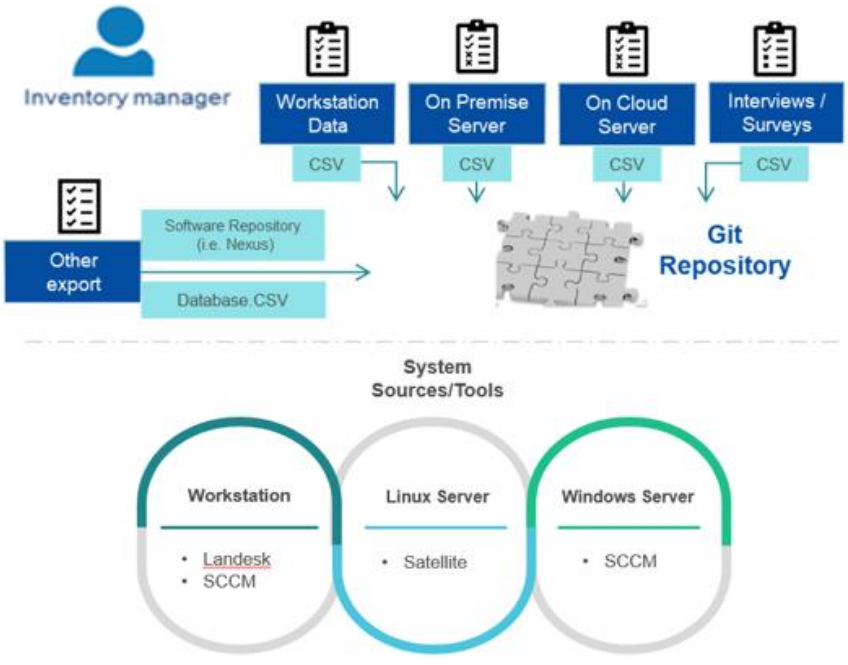


Figure 3: Phase 1 - Acquiring Data

During the execution of Step 2 – Acquire data, the following activities are performed in order to properly collect data:

a) **Brainstorming to agree on the CSV's mandatory and optional fields**

b) **Creation of CSV template and examples of the data that need to be acquired/gathered**

CSVs should cover software installed on user workstations and servers on premises or on cloud (production, QA, etc.). Example/sample files should be provided to the stakeholder along with the document to fill in.

Example: Since there are mandatory and optional fields included in the CSV template files, an example CSV template was provided with sample data in order to illustrate and provide guidance on how to properly fill in all the fields.

c) **Share the template CSV file with selected departments**

Departments fill in the data into these CSVs and send them back – this should be done, to the extent possible, via extract applications. In case data are provided via a database, an accurate definition of every field is expected.

Example: Sharing the template CSV via extract applications is an option, however all of the data were received via separate emails with attached CSV templates that were previously provided.

d) **Get data via interviews**

In certain cases, interviews can provide some meaningful data. Data can be collected during the interviews directly into CSV files or the interviewed organisations can be invited to fill them in afterwards and send them back to the Inventory Manager.

Example: There are cases in which, when a CSV template is provided, some additional clarifications may be required and/or requested for specific fields or columns. Thus, separate meetings/interviews can be scheduled to go through the CSV template in order to review and confirm with the stakeholder their understanding on how to properly include the relevant information.

e) **Anonymisation of the information**

It is recommended to anonymise the extracted data, prior to consolidating it with data from other sources. Anonymising data means ensuring it does not show personal information such as names of users, computer names or file paths etc.

Example: In cases where files received contain personal information, e.g names of users or computer names, the relevant fields will be replaced with "XXXXuser", "XXXXcomputer" etc.

3.5. Step 3: Consolidate and load into a database

Step 3/7

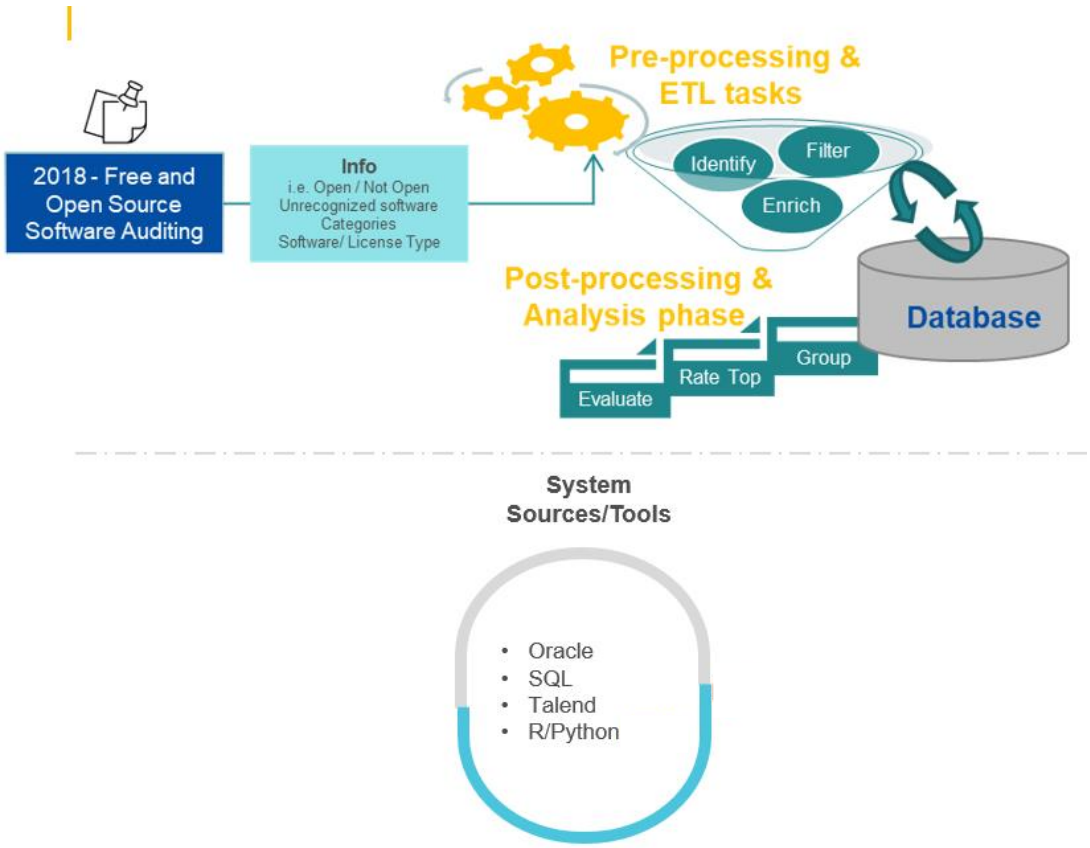


Figure 4: Phase2 - Data Management Data Model

3.5.1. Data Model

Once received, the data should be consolidated through a defined data model. Section a) of the present document contains the sample data model that has been used for the consolidation of the data received under the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, a component of the 2020 ISA² Sharing and Re-use action (2016.31).

3.5.2. Steps to loading data

In order to consolidate the database and load the information in it through Extract, Transform and Load (ETL) tasks, it is important to understand first the received data and doublecheck if it can be categorised as OSS. It is therefore important to proceed as follows:

a) Understanding the received data

1. Identify the software components.
2. Establish whether the software is open source or not:
   Example: To identify and establish that the software is open source or not, an online research is performed for each software.
3. Filter out non-Open Source Software:
   Example: Separated files are produced to divide the Open Source Software files from the non-Open Source Software ones.
4. Apply any needed data transformation:

Example: The data transformation step is explained in detail in the following Section 3.6, Step 4.

b) ETL Load

It is useful to use an ETL / BI tool, such as Talend, to load the data received into the database. A proposed and ideal model exists and has been used during the execution of the current project related to the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services (see Figure 12). Nonetheless, it is worth to note that the exact data model is determined by the type and characteristics of the data received.

3.6. Step 4: Analyse, clean and enrich the data

Step 4/7

In this step, we analyse, clean and group the received data as follows:

a) Categorisation of data

Group and categorise Open Source Software. All software that appears with similar names, should be represented with a single name. Prior to this step, a file is created in CSV format to gather all the software that is received in the appropriate format and, at a later stage, will be further analysed.

Decide which software requires this categorisation (e.g. FirefoxA, FirefoxB => Firefox). As an example, the project team receives, from several stakeholders, software names such as the ones in the below table that are then grouped, based on our categorisation, under a "Parent" software.

Software names received "Child" software	Identified "Parent" software categorisation
Mozilla Fi	Firefox
Mozzilla Fi	
Firefox ES	
NodeJS	NodeJs
Node.js	
NodeJS-	
Nodejs	
node	

*Example taken from the project performed under the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, a component of the 2020 ISA² Sharing and Re-use action (2016.31).

b) Process evaluation and re-run all steps

All previous steps need to be re-done multiple times, to eliminate the potential mistakes or risks of duplicates and/or data overlooking. For instance, after the categorisation that is performed, the project team can always identify room for adjustments in the grouping of the software, as illustrated in the examples above.

In order to apply all above steps, a new table called 'Categorisations' is created – originating from the table that includes the software names received from the stakeholders – to check which software belongs to which category.

The table “Categorisations” includes the following grouping of data (sample data):

PARENT SOFTWARE	CHILD SOFTWARE	NUM_OF_INST
FireFox	Firefox	85081
FireFox	Mozilla Fi	320
FireFox	Mozilla Fi	82
FireFox	FireFox	10
FireFox	Firefox ES	6
FireFox	Selenium	1

*Example taken from the project performed under the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, a component of the 2020 ISA² Sharing and Re-use action (2016.31).

3.7. Assessment Criteria definition

Before we move on and execute steps 5 and 6 it is crucial – for the appropriate implementation of the methodology – to proceed with a thorough and accurate definition of the assessment criteria.

This methodology uses three types of criteria to filter and tag open source data, which are:

1. “**Business criticality**” criteria: Applied to software components and applications, it indicates how “heavily” the software is used within the organisation, based on the number of instances (number of installations) that each software has.
2. “**Vulnerability**” criteria: Applied to software components and applications, it shows how secure the software can be, based on its relevance to security and the exposure it has to users. The exposure is defined by the option of the software to have a user interface and/or a user contact.
3. “**Sustainability**” criteria: Applied to open source communities and projects, it indicates the strength of the software in terms of continuity. To assess and evaluate sustainability, a set of measurable metrics is defined on the basis of aspects that can affect and impact the sustainability of the targeted software. In general, such aspects include:

    - Community Activity
    - Performance
    - Quality and Security
    - Demographics and Diversity
    - Governance
    - Support

    In conjunction with the identification of aspects, a Metric Measurement Approach is developed which describes the process for measuring metrics used to evaluate sustainability.

    In our case, the project team has used the set of metrics developed and defined by WP1 of the EU-FOSSA Pilot project (See Annex 2: Metrics and Sustainability).

3.8. Step 5: Apply the business criticality and vulnerability criteria

Step 5/7

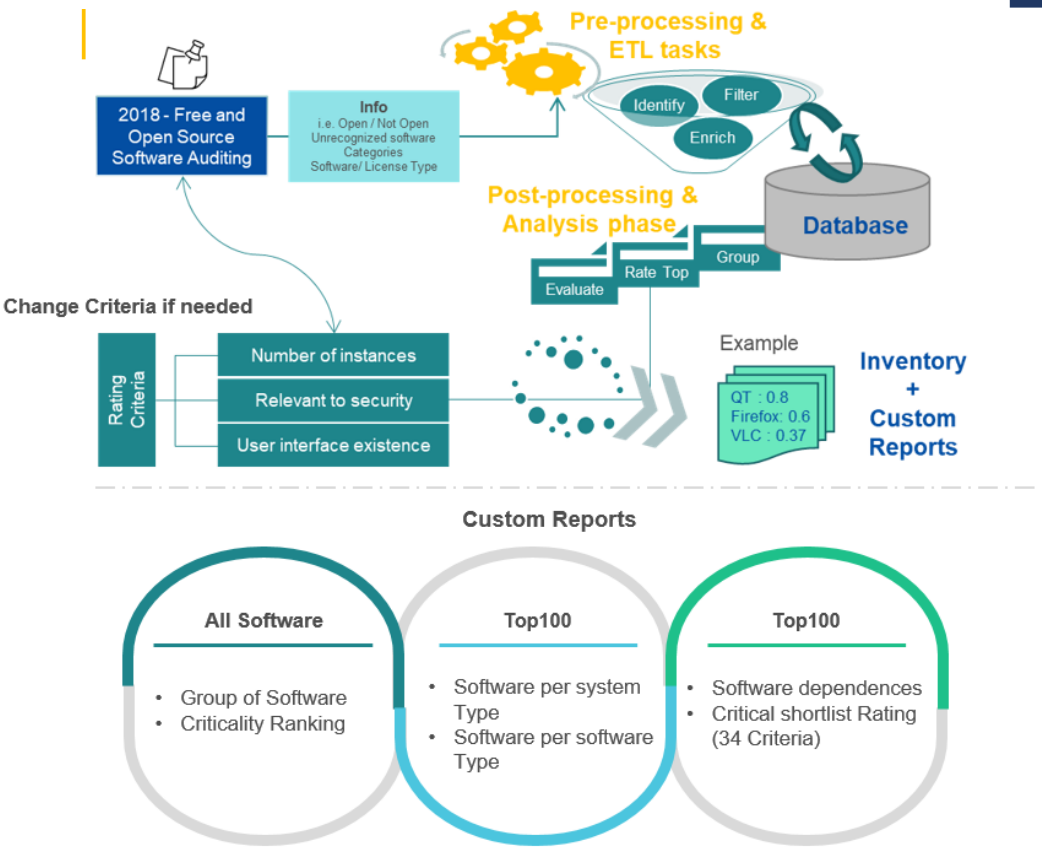


Figure 5: Phase 3 - Criteria and Inventory Creation Data Model

a) Number of instances analysis

A software list is created with all relevant categorisations and number of instances (number of installations) that each software has. The normalisation (normalised score) is applied on a scale ranging from 0 to 1, based on the most common software (maximum instances).

Rationale: The more a software is deployed, the more it impacts the infrastructure and/or user base, and the more damage a vulnerability could cause.

Rating: Normalisation ranges on a scale from 0 to 1, based on the most common software (maximum instances).

b) Security analysis

The exposure of the software is analysed to define the relevance of the security. A binary rating is introduced (security-related = 0.5, not security-related = 0).

Rationale: A vulnerability in a component related to a security aspect may increase the damage due to an exploit. Examples of security-related software are the solutions meant to secure communication, manage authentication, manage processes and permissions, etc.

Rating: A binary rating (security-related = 0.5, not security-related = 0).

c) User Interface analysis

We analyse whether the software has a User Interface or not and whether it can be easily hacked or not. A binary rating is introduced (exposed to users = 1, non-exposed to users = 0).

Rationale: A vulnerability in a component exposed to the end user (i.e. that offers an interface to end users) increases the risk of an exploit attacking the software. This criterion only applies to data centre infrastructure, since workstation users have a direct login to their machines.

Rating: A binary rating (exposed to users = 1, non-exposed to users = 0).

d) Index calculation

The total score – provided by the sum of the three above scores – is then normalised on a scale of 0 to 1 (dividing by 2.5, i.e. the sum of the highest values of the three criteria).

In this way the "**Business Criticality Index**" is created.

In the example displayed below, we notice that the initial list of the software names is sorted based on the number of instances, starting with the software with the maximum number of instances. The criticality indexes are applied accordingly:

➔ Interface

➔ User Contact

➔ Exposure to users

➔ Security

Criticality Indexes

SOFTWARENAME	A/A Software	NOOFINSTANCES	interface (0 = no 1 = yes)	user contact (0 = no 1 = yes)	Exposure to Users	Security (0 = no 1 = yes)
LibreOffice	1	10910	1	0	1	0
Ubuntu	2	1562	1	0	1	0.5
SonarQube	3	1105	1	0	1	0.5
Squash	4	946	1	0	1	0.5
Git	5	742	1	0	1	0
SILL	6	517	1	0	1	0

As an example, the criticality indexes for the software KeeFox are shown below:

KeeFox	57	293	1	0	1	0.5

A sample of application scores for the same software group is shown below.

SOFTWARENAME	A/A Software	NOOFINSTANCES	# Instances of the most common of the list	# Instances	Relative # of instances	Exposure to users	Relation with security	Score	Normalized score
LibreOffice	1	10910	10910	10910	1	1	0	2	0.8
Ubuntu	2	1562	10910	1562	0.143171402	1	0.5	1.643171402	0.657268561
SonarQube	3	1105	10910	1105	0.101283226	1	0.5	1.601283226	0.640513291
Squash	4	946	10910	946	0.086709441	1	0.5	1.586709441	0.634683776
Git	5	742	10910	742	0.068010999	1	0	1.068010999	0.4272044
SILL	6	517	10910	517	0.047387718	1	0	1.047387718	0.418955087

And, specifically, for Keefox:

SOFTWARENAME	A/A Software	NOOFINSTANCES	# Instances of the most common of the list	# Instances	Relative # of instances	Exposure to users	Relation with security	Score	Normalized score
KeeFox	57	293	10910	293	0.026856095	1	0.5	1.526856095	0.610742438

*Examples taken from the project performed under the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, a component of the 2020 ISA² Sharing and Re-use action (2016.31).

Outcome:

KeeFox has received a score of 1.526856095 (as a result of adding up the metrics "Relative # of instances", "Exposure to users" and "Relation with security"), putting the software at the 57th position.

However, to be able to analyse all metrics on the same scale, in a quantitative way, the scores have to be normalised by dividing them by 2,5 (the sum of the highest values of the three criticality indexes). The normalised score of KeeFox is therefore of 0.610742438.

Based on this last normalisation process, if all software were sorted by highest score first, KeeFox would move from its original 57th position to the 20th position. This means that compared to all the software in the list, KeeFox's **business criticality** climbed up 64,91% making KeeFox appear as a much more important and critical software than initially analysed.

Original Sorting based on Scoring

SOFTWARENAME	Normalized Sco	Original A/A Softwa	Current A/A Softwa	Check A/A	NOOFINSTANC
LibreOffice	0.8	1	1	↑0.00%	10910
Ubuntu	0.657268561	2	2	↑0.00%	1562
SonarQube	0.640513291	3	3	↑0.00%	1105
Squash	0.634683776	4	4	↑0.00%	946
Selenium	0.618515124	7	5	↑28.57%	505
FireFox	0.618331806	8	6	↑25.00%	500
CE gravity	0.614812099	11	7	↑36.36%	474
Rancher	0.614152154	19	8	↑57.89%	387
exodus-privacy (standalone)	0.61411549	20	9	↑55.00%	358
Opal	0.613932172	21	10	↑52.38%	357
SecureFlag Community	0.613455545	29	11	↑62.07%	355
Yunohost	0.613235564	34	12	↑64.71%	314
postfix.admin	0.613162236	35	13	↑62.86%	302
Tomcat	0.611512374	44	14	↑68.18%	283
Vault	0.611439047	45	15	↑66.67%	256
ARX Data Anonymization Tool	0.61136572	46	16	↑65.22%	249
CockroachDB	0.611109074	51	17	↑66.67%	246
Keycloak	0.611035747	53	18	↑66.04%	243
Escalation	0.610852429	56	19	↑66.07%	242
KeeFox	0.610742438	57	20	↑64.91%	231
CentOS	0.609129239	76	21	↑72.37%	223
Debian	0.609019248	77	22	↑71.43%	211

*Example taken from the project performed under the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, a component of the 2020 ISA² Sharing and Re-use action (2016.31).

Although a lot of manual work is needed, especially in the cases of new software, this data management stage should be as automated and parametrised as possible.

A mechanism that will detect and explore only new software additions – and function as a first stage of control, too – can be useful for the respective future projects.

3.9. Step 6: Apply software sustainability criteria

Step 6/7

For the top 30 results, apply the defined sustainability criteria. In our case, the project team has applied the ones defined in Annex 2: Metrics and Sustainability.

To illustrate how the below sustainability criteria metrics have been defined, the KeeFox software has been, again, selected as an example.

In the below case, the following metrics have been illustrated:

- **Code Activity** = Metric 1 from the Community Activity aspect
- **Release History** = Metric 2 from the Community Activity aspects
- **Number of Tickets** = Metric 4 from the Community Activity aspect

The below presented results were produced based on online research.

Code Activity = Metric 1 from the Community Activity aspect

Metrics family		KeeFox (Kee is the new version)
	Source	https://www.openhub.net/p/keefox
	# contrib who committed 80%	1
	# contrib over past year	1
	Contributors ratio	9%
	Textual metrics	Very dependant
1 - Code Activity	Normalized ratio	0%

*Example taken from the project performed under the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, a component of the 2020 ISA² Sharing and Re-use action (2016.31).

For example, the criteria of contributions, namely "#contrib who committed 80%" and "#contrib over the past year", that belong to the Code Activity (metric 1 from the Community Activity aspect, were taken from https://www.openhub.net/p/keefox). The data are shown in the screenshot below:

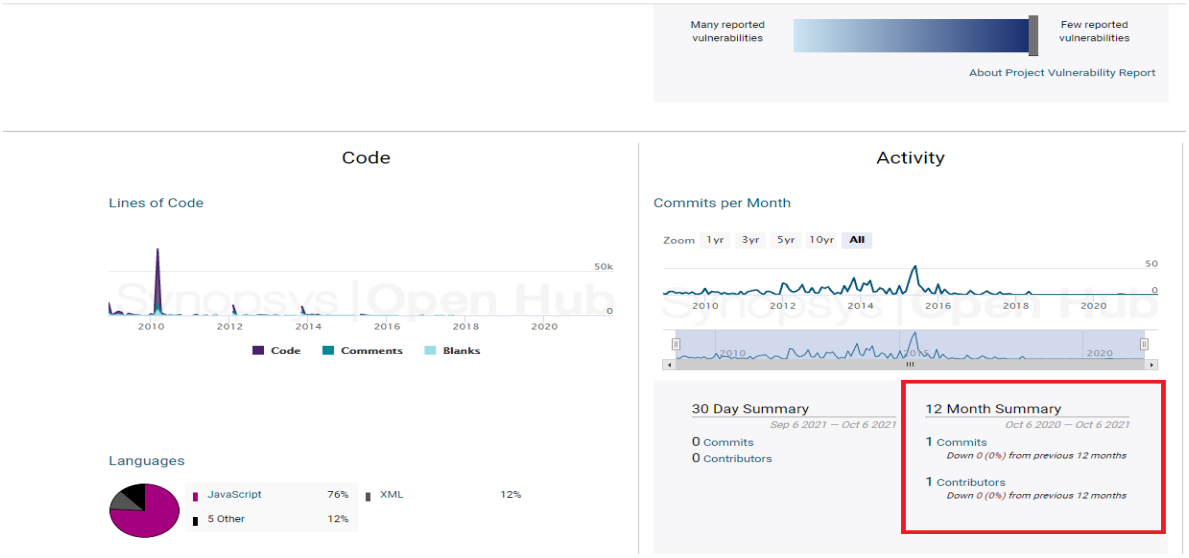


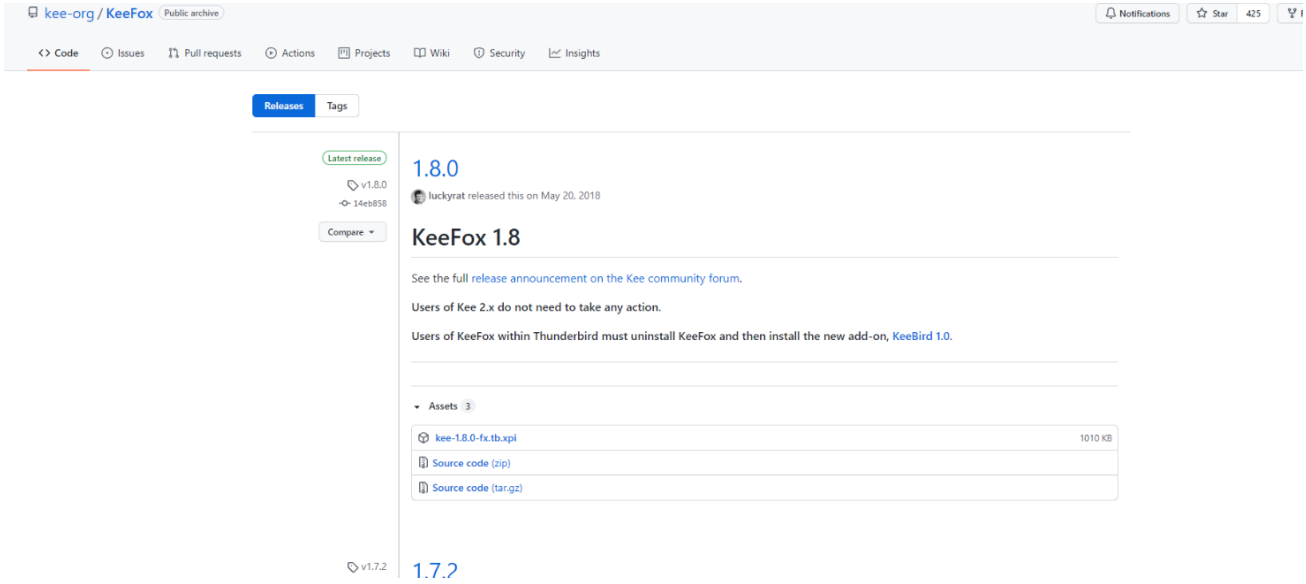
Figure 6: Example of Contributing Activity Metric

Release History = Metric 2 from the Community Activity aspect

Another example that could be used to illustrate how the "Release History" (metric 2 from the Community Activity aspect) was produced is taken from the following link:

https://github.com/kee-org/KeeFox/releases

The value "Managed" is provided since it seems that there is an informal approach, for release/publication when development objectives are achieved from the release history, as shown in the screenshot below.



Number of Tickets = Metric 4 from the Community Activity aspect

To illustrate how the sustainability criteria have been applied and results been produced, the "Number of Tickets"(metric 4 from the Community Activity aspect) will be used as a last example.

The defined available values for the "Number of Tickets" are:

1. Very active: there are, at least, 10 tickets created in the last week.
2. Active: there are, at least, 10 tickets created in the last two weeks.
3. Average: there are, at least, 10 tickets created in the last month.
4. Inactive: there are, at least, 10 tickets created in the last three months.
5. Very Inactive: rest of the values.

As depicted in the below screenshot – taken during the online research performed on https://github.com/kee-org/KeeFox/issues – the last ticket was opened on October 2020. Therefore, we can consider that the value to be provided is "**5 – Very inactive**" since it includes anything over three months old.

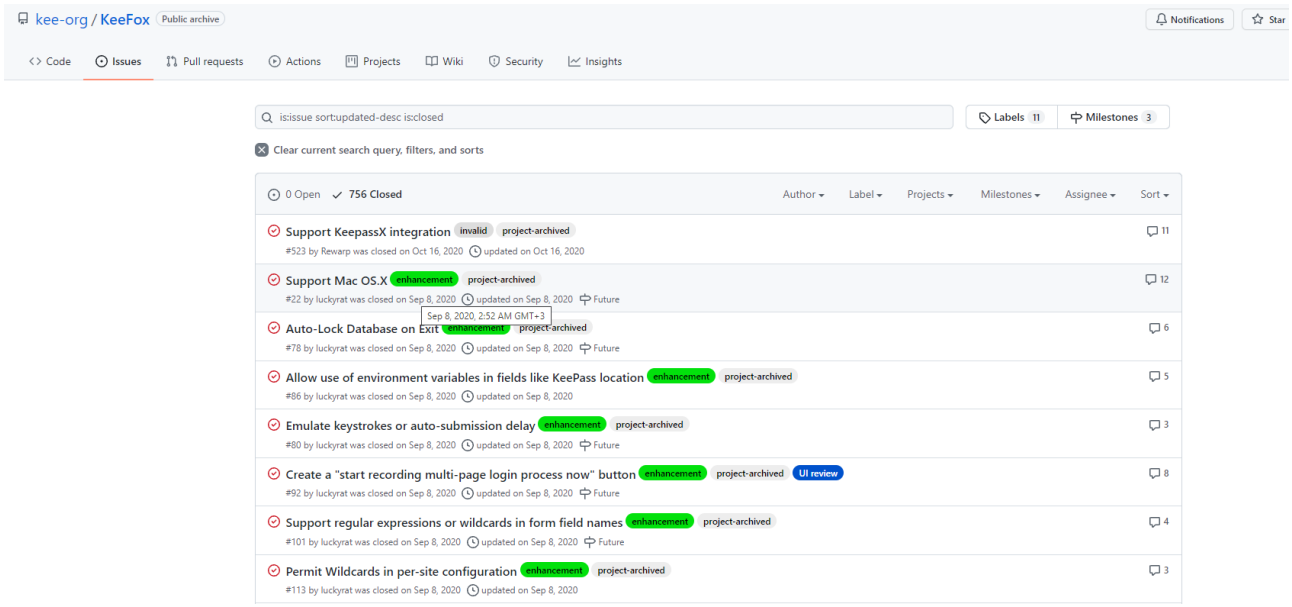


Figure 7: Activity Metric for KeeFox

This would produce the following results for metric 4 from the Community Activity aspect:

4 - Number of Tickets	Source	https://github.com/kee-org/KeeFox/issues
	at least 10 tickets over last:	more than three months
	Textual rating	Very inactive
	Normalised ratio	0%

*Example taken from the project performed under the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, a component of the 2020 ISA² Sharing and Re-use action (2016.31).

3.10. Step 7: Produce final results/reports

Step 7/7

The last part is the creation of the inventory that includes the following tasks:

- Create the software inventory, including a number of custom reports.
- Prepare a publishable version of the inventory.
- Produce a summary and present to management.

3.11. Summary and conclusion on the Inventory Methodology and sample reports

3.11.1. Software inventory procedure from the Inventory Manager's perspective

Figure 8: Procedure from Inventory Manager's perspective

Data collection is a "pull" process starting with a periodic reminder (for example an e-mail) to the interested counterparties (the stakeholders owning the relevant data) sent by the process owner, or Inventory Manager (to be properly identified and appointed). The reminder message shall indicate a due date and a set of instructions for operators on how to execute the data extraction and allocation into the repository.

Once the Inventory Manager has received sufficient confirmation from all data providers, s/he will start the ETL sub-process to populate the inventory database and create the necessary ETL jobs based on the level of information received.

For the Data Centre, the underlying hypothesis is that all collected data are about known software. This means that all items treated in the inventory must have been previously recognised as software components or software products bearing some brand name (including in-house codes) that can be associated with an external manufacturer (or an organisational unit) or with a community.

As for desktops, it is expected that the full list of installed software is made available for the inventory. When the inventory database is populated, the Inventory Manager can manually adjust the ranking criteria based on a first set of quantitative criteria (possibly excluding some criteria and/or fixing thresholds) and interactively select the most relevant set of software applications/components.

Finally, the set of selected software (components) can be enriched with metadata such as licensing type, known vulnerabilities etc., and be prepared for the final ranking.

"Unknown" software (i.e. software not associated with a community, organisational unit of a stakeholder, or another identifiable manufacturer) is the first candidate for inspection, but this is out of the scope of this methodology (see Section 4, "Recommendations and Next Steps"). However, some additional processing can be applied to software recognised as open source, to decide how to contribute to their OSS communities.

One inventory use case could be to obtain a shortlist of critical software components, by applying criteria to the inventory items in order to rank them by criticality.

The final ranking is performed by the Inventory Manager, adjusting the previous ranking based on a second set of qualitative criteria (sustainability).

3.11.2. Sample reports

The inventory consists of a number of Custom Reports based on all available data. These Custom Reports are as follows:

a. Grouping of Software

PARENTSOFTWARE	SOFTWARENAME	NUM_OF_IN	EXTENSION	OPEN S
FireFox	Firefox	85081		TRUE
LibreOffice	libreoffice	85080		TRUE
Thunderbird	Thunderbird	85080		TRUE
adoptopenJDK	adoptopenJDK	85000		TRUE
openSC	openSC	85000		TRUE
ssh	ssh	85000		TRUE
VLC	VLC Media Player	85000		TRUE
VLC	vnc	85000		TRUE
Ubuntu	Ubuntu LTS	78000		TRUE
luks	luks	30000		TRUE
strongswan	strongswan	30000		TRUE
LibreOffice	LibreOffice	10493		TRUE
Debian	Debian	4746		TRUE
Syslog-ng	Syslog-ng	4742		TRUE
squid	squid	4501		TRUE
apache	apache	4500		TRUE
bacula	bacula	4500		TRUE
cups	cups	4500		TRUE
dovecot	dovecot	4500		TRUE
naemon	naemon	4500		TRUE
nginx	nginx	4500		TRUE
php	php	4500		TRUE
samba	samba	4500		TRUE
Ubuntu	Linux Ubuntu OS	1262		TRUE
Haproxy	Haproxy	1000		TRUE
Python	python	1000		TRUE

Figure 9: Grouping of Software Report

*Example taken from the project performed under the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, a component of the 2020 ISA² Sharing and Re-use action (2016.31).

b. Software by System Type

PARENTSOFTWARE	SYSTEMTYPE	NUM_OF_INST
7zip	Server	1
	Workstation	127
	(blank)	76
Acceleo	(blank)	304
achat	(blank)	1
adoptopenJDK	Workstation	85000
Aegisub	(blank)	1
aide-sociale	(blank)	3
AjaxControlToolkit	Server	1
Akelpad	Server	1
ALCASAR	(blank)	1
ALM	(blank)	1
Alternatiff	(blank)	2
Anaconda	Server	1
Android	(blank)	1
Angular	Mobile device	1
	Server	2
Ansible	Server	3
	(blank)	225
apache	Server	4605
Apache SolR	Server	1
Apache Ant	Server	1
Apache Ignite	Server	1

Figure 10: Software by System type Report

*Example taken from the project performed under the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, a component of the 2020 ISA² Sharing and Re-use action (2016.31).

c. Criticality Ranking

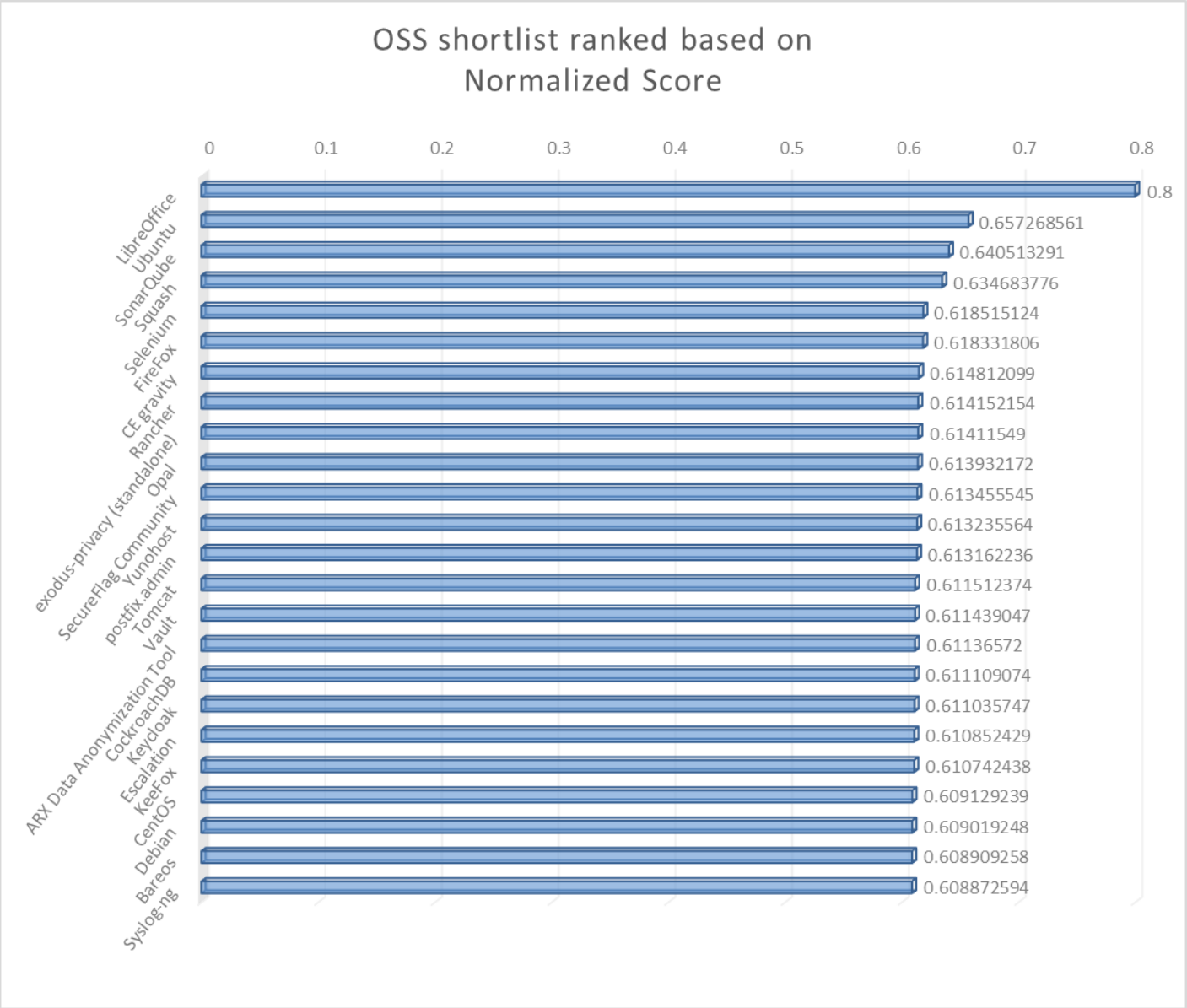


Figure 11: Criticality Ranking Report

*Example taken from the project performed under the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, a component of the 2020 ISA² Sharing and Re-use action (2016.31).

Additionally, the inventory consists of a number of Custom Reports based on **the top 30 results**. These Custom Reports are as follows:

a. Software by Software Type

Top Open Software	No of Instances	Software Type
LibreOffice	10910	Application software/Tool
Ubuntu	1562	Operating system
SonarQube	1105	Application software/Tool
Squash	946	Application software/Tool
Selenium	505	Runtime software platform
FireFox	500	Application software/Tool
CE gravity	474	Application software/Tool
Rancher	387	Development platform/framework
exodus-privacy (standalone)	358	Mobile software
Opal	357	Operating system
Tomcat	283	Runtime software platform
Vault	256	Application software/Tool
ARX Data Anonymization Tool	249	Application software/Tool
CockroachDB(not fully open-sourced, must purchase a license)	246	Runtime software platform
Keycloak	243	Application software/Tool
Escalation	242	Application software/Tool
KeeFox(Kee is the new version)	231	Application software/Tool
CentOS	223	Operating system
Debian	211	Operating system
Syslog-ng	207	Libraries
Rudder	189	Application software/Tools
ClamAV	184	Application software/Tool
Centreon	183	Application software/Tool

Figure 12: Software by Software Type Report

*Example taken from the project performed under the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, a component of the 2020 ISA² Sharing and Re-use action (2016.31).

b. Software Dependencies

Components	Number of Dependencies
FireFox	38
exodus-privacy (standalone)	31
CE gravity	22
Opal	18
LibreOffice	17
Ubuntu	16
Squash	12
Rancher	10
SonarQube	7
Selenium	7
Grand Total	178

Components	Number of Dependencies
glibc	5
zlib	5
glib2	4
bash	4
libX11	3
gtk2	3
systemd	2
libXext	2
libselinux	2
freetype	2
log4j	2
gdk-pixbuf2	2
libffi	2
bcmail-jdk14	2
commons-beanutils	2
lua	2
libXrender	2
gtk3	2
fontconfig	2
hamcrest	2
nss	2
Chart.yml	1
xml-apis	1

Figure 13: Software Dependencies Report

*Example taken from the project performed under the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, a component of the 2020 ISA² Sharing and Re-use action (2016.31).

c. Critical shortlist Rating (34 different criteria will be applied)[8].

Metrics family		LibreOffice	Ubuntu	SonarQube	Squash	Selenium	FireFox	CE gravity
	Source	https://www.libreoffice.org/	https://www.openhub.net/p/ubuntu	https://www.openhub.net/p?query=SonarQube	https://www.openhub.net/p?query=squash	https://github.com/SeleniumHQ/selenium.git	https://www.openhub.net/p/firefox	https://github.com/gravitational/gravity.git
	# contrib who committed 80%	4	1	11	4	5	240	7
	# contrib over past y	75	12	58	25	55	5226	23
	Contributors ratio	74%	49%	81%	67%	76%	82%	67%
	Textual metrics	Split	Average	Very split	Split	Split	Very split	Split
1 - Code Activity	Normalized ratio	75%	50%	100%	75%	75%	100%	75%
	source	https://wiki.documentfoundation.org/ReleasePlan	https://wiki.ubuntu.com/Releases	https://docs.sonarqube.org/latest/setup/upgrade-notes/	https://www.squashtest.com/community-roadmap-releases-tm?lang=en	https://github.com/SeleniumHQ/selenium/releases	https://www.mozilla.org/en-US/firefox/releases/	https://goteleport.com/gravity/docs/changelog/
	Textual metrics	Managed	Optimized	Managed	Managed	Managed	Optimized	Managed
2 - Release History	Normalized ratio	50%	100%	50%	50%	50%	100%	50%
	# commits last year	6789	117	1183	812	834	53205	402
	# commits last year top popular OpenHub Repository	884	884	884	884	884	884	884
	Ratio	768%	13%	134%	92%	94%	6019%	45%
	Textual metrics	Very active	Average	Very active	Very active	Very active	Very active	Active
3 - Number of Commits	Normalized ratio	100%	50%	100%	100%	100%	100%	75%
	Source	https://bugzilla.mozilla.org/buglist.cgi?short_desc=libreoffice&query_format=advance	https://bugzilla.mozilla.org/buglist.cgi?short_desc=ubuntu&short_desc_t	https://jira.sonarsource.com/issues/	https://bugzilla.mozilla.org/buglist.cgi?short_desc=squash&short_desc_	https://bugzilla.mozilla.org/buglist.cgi?short_desc_type=allwordssubstr&	https://bugzilla.mozilla.org/query.cgi	https://bugzilla.mozilla.org/buglist.cgi?resolution=---
	at least 10 tickets over last:	last week	last week	last week	more than three months	last month	last week	more than three months
	Textual rating	Very active	Very active	Very active	Very inactive	Average	Very active	Very inactive
4 - Number of Tickets	Normalized ratio	100%	100%	100%	0%	50%	100%	0%
	Textual rating	Optimized	Optimized	Optimized	Optimized	Managed	Optimized	Initial
5 - Communications	Normalized ratio	100%	100%	100%	100%	66%	100%	33%

Figure 14: Critical Shortlist Rating Report (Assessment of top items in the inventoried software against the criticality mechanism defined in the EU-FOSSA Pilot project)

*Example taken from the project performed under the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, a component of the 2020 ISA² Sharing and Re-use action (2016.31).

It should be noted that the set of sustainability criteria contains a number of qualitative criteria previously defined by the Inventory Manager, on the basis of a thoroughly designed and described Metric Measurement Approach. Therefore, in order to produce more reliable results, the metrics should be adjusted accordingly for each project (i.e. removed or replaced with a more quantifiable set of options, depending on its specifics and characteristics).

[8] See Deliverable 6 - Final Metrics Definition (europa.eu).

4. RECOMMENDATIONS, GOOD PRACTICES AND NEXT STEPS

As mentioned in chapter 2, to reach its full maturity, the methodology should improve the three identified areas, in order to:

1. Enable a direct and automated way to collect, in real-time, the data from the system/servers of the various stakeholders and, therefore, improve the completeness of the inventory since this would not depend on a manual selection of information provided by the stakeholders.
2. Enable an automated system to collect assessment metadata through international databases.
3. Use a Business Intelligence tool to proceed with the steps currently done manually via spreadsheet files.

An example of a targeted/target scenario, implementing an updated and optimal methodology, is described in paragraph 4.3 - "The target scenario – first step" and uses a public international organisation as "implementer" of a further mature/optimal methodology.

The target scenario is used as an example and for illustration purposes only, and recognises that each public or private administration will have its own target scenario and will, therefore, adapt the methodology to its own context and needs.

The below sections provide recommendations and good practices to apply while implementing the methodology as well as a projection on how the optimal methodology could be implemented by an organisation.

4.1. Recommendations

Based on the analysis performed in the framework of the present project, namely the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services, some recommendations are hereby provided on future actions that stakeholders, or any future interested Inventory Manager, may implement to enhance the efficiency and effectiveness of the Inventory Methodology and its related processes:

- Continue along the guidelines set by this project, enlarging its scope and consolidating processes and IT systems:
  - Industrialise the methodology described in these pages through the development of a maximum of automated processes;
  - Industrialise the processes and information system elements introduced in this document, transforming them in an "industrial" solution (see section 5.2).

- Adopt security practices into the software development/adoption lifecycle:
  - Select and install only secure and supported open source software;

- Actively maintain an accurate list of OSS components and applications;
- Identify vulnerabilities during development;
- Alert product/solution managers of potentially vulnerable applications based on the track of new vulnerabilities.

- Foster the adoption of a common Configuration Management Database (CMDB) consolidating all the different inventories.

- Apply and implement, to the extent possible, the good practices presented in the next section.

4.2. Good practices

a) Target data model definition

To consolidate the received data, the Inventory Manager should use a data model. The presented Target Data Model is an ideal conceptual, object-oriented model. Therefore, it is technology-independent and is not intended to be an image of the database the inventory tool will use. Thus, entities are not mapped one-to-one in database tables.

The below presented model is recommended by the project team as a good practice and has been used and verified during the performance of the current project related to the Open Source Software Inventory, Security, Sustainability and Funding Initiatives for European Public Services.

The model describes:

- **Entities**: coherent aggregates of information, related to real-world objects, ideas or contexts, which are commonly stored into database tables.
- **Attributes**: simple pieces of information (text, numbers, lists, etc.) belonging to an entity, which are commonly stored into database table columns.
- **Relationships**: connections that represent hierarchy or interaction between entities.

Each entity has the following properties:

- **Name**: a sequence of words that identifies the entity.
- **Description**: a short phrase that explains the role and information content of the entity.
- **Requirements**: a list of the project requirements that led to the definition of the entity.
- **Sources**: a list of the information sources from which the entities' information is gathered (e.g. Landesk, App-V, Satellite).
- **Type**: if the entity is a specialisation of another entity, the value is "Dependent"; else, the value is "Independent".

Attributes are organised by entities. Each attribute has the following properties:

- **Name**: a sequence of words that identifies the attribute.
- **Definition**: a short phrase that explains the role and information content of the attribute.
- **Required:** if the field required or not**.**
- **Is PK**: if the attribute is used to identify the entity it belongs to.
- **Is FK**: if the attribute references an external entity.

The model is built around the Software and System core entities. The Software entity aggregates all the information required to perform the software inventory, software attributes and meta-data, while the System entity contains the information related to the systems, physical or virtual, where the software is deployed. The data for the Software entity are manually and locally managed by the Inventory Manager, while the ones that belong to the System entity are automatically loaded from external systems (Landesk, App-V, Satellite and other CMDBs). A third entity, SoftwareInstance, represents the software that has been actually deployed, and works as a bridge between the two.

The Software entity is related to versions and licences. Each software version is tied to its evaluation criteria, which are evaluated to assess if the software must be included in the Critical Software Shortlist. The software classes that were declared as in-scope in the requirements are also modelled as specialisations of the Software entity.

The System entity is divided into workstations, servers and mobile devices. The first two system types are in-scope, while the last one is currently under evaluation. It will be excluded from the Data Model if definitively assessed as out-of-scope.

Both Software and System entities are related to the standards they comply with. As the standard inventory is a project requirement, a Standard entity contains all the information gathered from the information sources and can be considered as a fourth core entity.

Organisations that own or produce software, standards and/or systems are also related to the four main entities that have been modelled.

Details about project requirements are mapped to entities that answer to those requirements. The same operation is performed for data sources that have been currently identified as available.

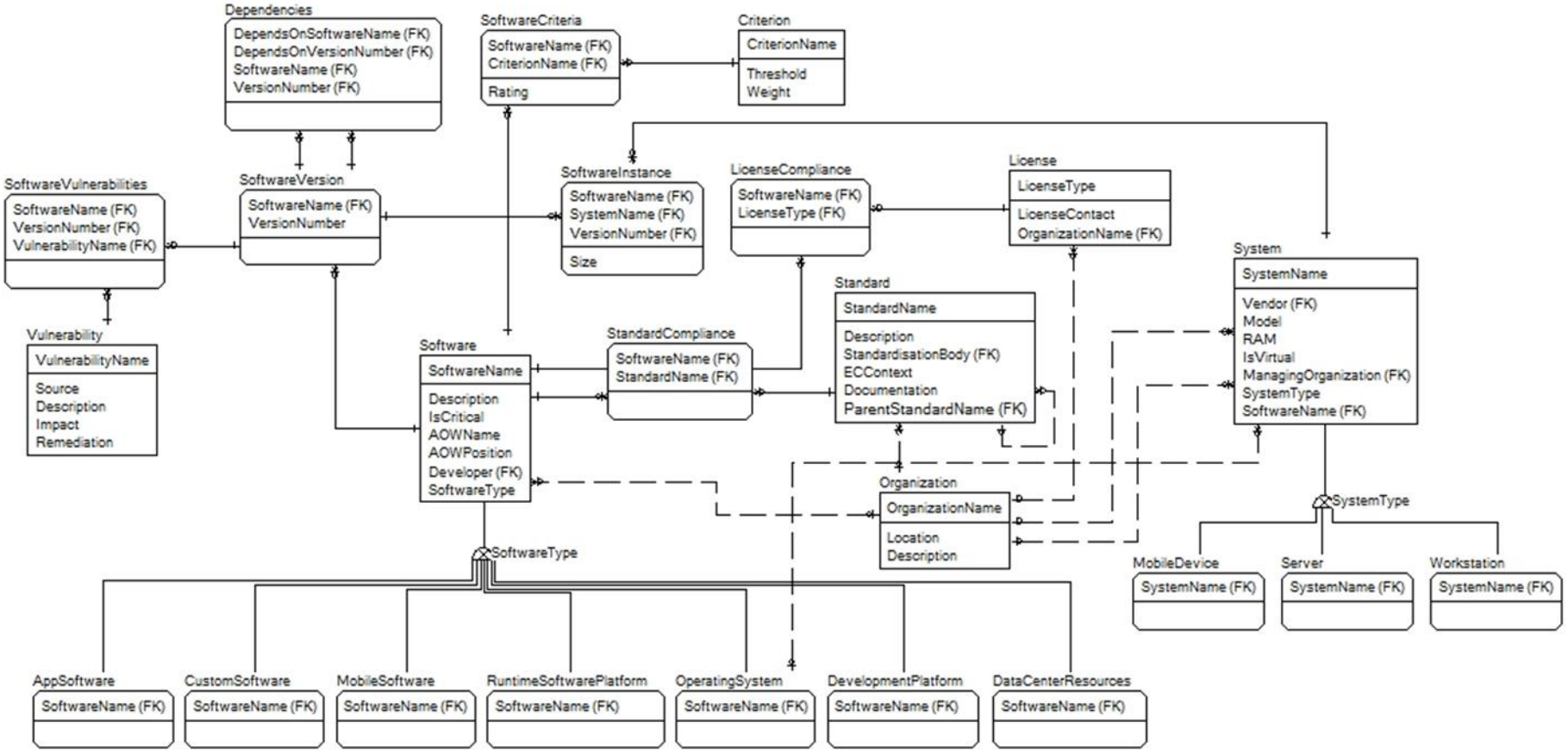


Figure 15: Target Data Model Diagram

More detailed information about the model is provided in Annex 3: Detailed Description of Target Data Model.

b) Focus on the internal software development/acquisition processes by adopting best practices and solid solutions.

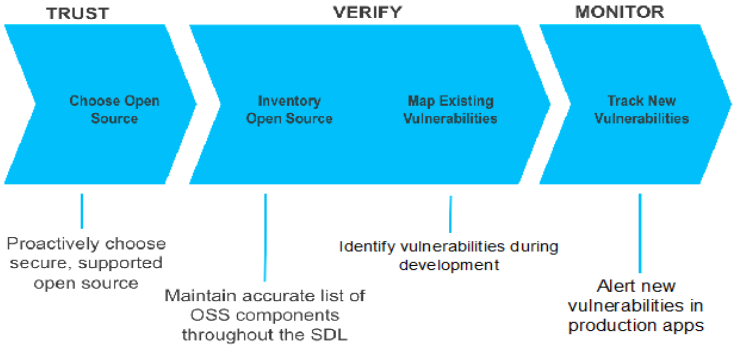


Figure 16: Proposed high level to-be approach

A best practices solution would combine elements of TRUST, VERIFICATION, and MONITORING:

1 –TRUST means providing developers and architects with a way to choose open source components that are free of known vulnerabilities and have active community support. This is a proactive step that reduces risks downstream in the software development process, and is the most cost-effective means of risk reduction.

2 – VERIFICATION means maintaining an accurate inventory of open source software and being able to map all its known vulnerabilities, in any and all applications, at any point in the SDL.

3 – MONITOR means being able to monitor the released code for newly discovered vulnerabilities and alert the right people for remediation. With over 4,000 new vulnerabilities each year, a comprehensive solution should be to continuously monitor the constant stream of new vulnerabilities and automatically notify the administrator of any new vulnerabilities in the open source components used in deployed applications, including which applications use the code, how critical the vulnerability is, and how to remediate it.

c) Handling “unknown” software

An additional, and very important, aspect is the enlargement of the software component scope to include “unknown” software. In such cases, the added value of the “target” scenario can be significantly higher than the previous relative projects, including this one.

As described in the data collection section[9], this project is based on the hypothesis that only "known" software components/applications will be dealt with.

Management of "unknown" software is a strong constraint and a complex aspect. It requires the handling of large amounts of raw inventory data. Moreover, "unrecognised objects" have to be collected and matched with some "known" data patterns in order to understand their nature (source code, executable, scripts etc.), and professional tools will be needed to scan and recognise them.

Despite the complexity of the abovementioned process, from a security point of view, the most interesting elements are the "unknown" software components, which is why the project team strongly recommends considering this aspect as a priority in future projects.

[9] Section 3.4. Step 2 : Acquire data

4.3. The target scenario – first step

This section serves as an example of a target scenario, which uses a public international organisation (PIOX) as "implementer" of a more mature/optimal methodology.
The target scenario is used as an example – and for illustration purposes only – and recognises that each public or private administration will have its own target scenario and will adapt the methodology to their own context and needs.

The first step after the conclusion of this project should be to start a programme to reach the "target" scenario, with robust and agreed processes and an industrial-grade IT support solution.

The suggested "target" scenario is as follows:

- PIOX makes recurring automatic inventories to collect the software components that are in place (development and production);
- PIOX has a consolidated CMDB which is regularly enriched with inventory data;
- PIOX has a consolidated repository where it stores a "reference" copy of any in-house developed or downloaded/used software (source, executable, data etc.);
- On a regular basis, PIOX conducts automatic verifications that code present on the systems corresponds to the "reference" copy;
- PIOX has a policy to apply a form of licensing to its in-house developed software and has a policy to evaluate whether to submit this software to a public community or to contribute to an OSS initiative;
- PIOX has a policy to foster employees' contribution to open software communities with the products of their work;
- On a regular basis, PIOX scans the code repository with appropriate tools to find any possible "alien" or "unlicensed" software component.

A detailed analysis regarding the tools that can be used to support the open source inventory and their ranking is the objective of another specific deliverable. Below, the overall features of the "target" processes/solution are presented:

1. Industrial automatic discovery and inventory tool, able to collect all the information about software components;
2. Automatic inquiry of large internet databases to find additional metadata (licensing form, community dimension, vulnerabilities etc.);
3. Semi-automatic semantic web engine capable to enrich an initial list of standards;
4. Graphic editing of the standard taxonomy;
5. "Business Intelligence" dashboard with customisable ranking criteria/rules;
6. Automatic publishing of the inventory and ranking as open-data on http://open-data.europa.eu/.

The "target" recurring processes are therefore the following:

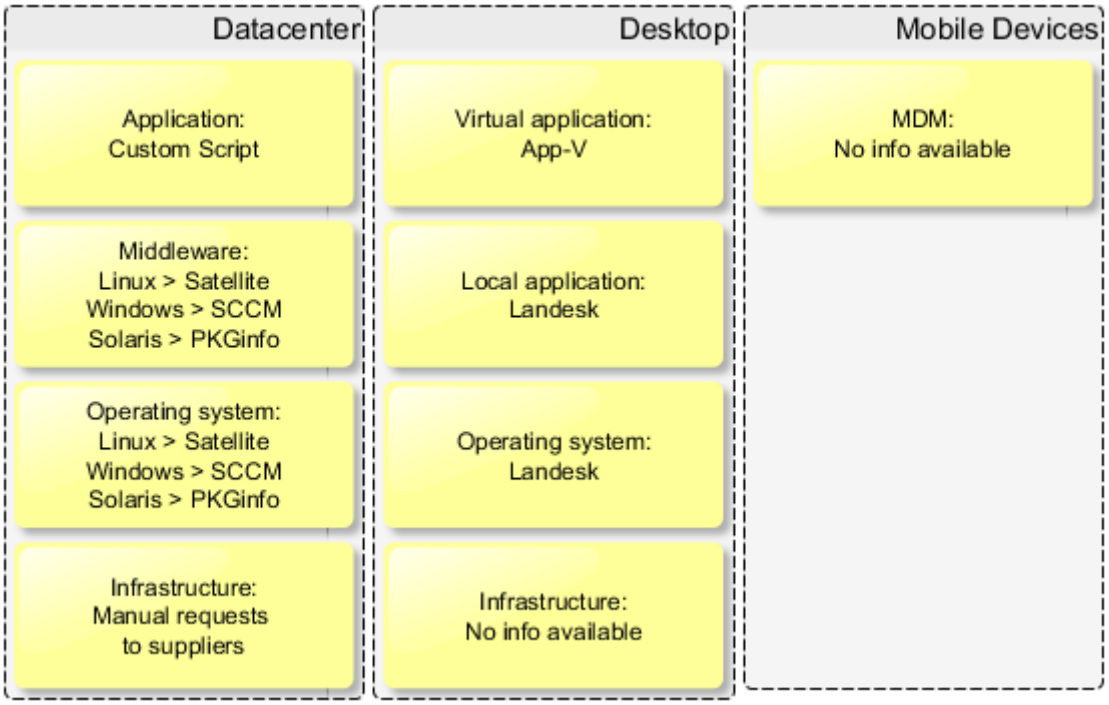
1. Automatic and semi-transparent Open Source Software component inventory and classification.
2. Automatic inquiry of internet databases.
3. Semi-automatic ranking.
4. Selection of candidates for the code review.

This ideal situation will be enriched and described as the project progresses and will, eventually, provide a set of pragmatic recommendations to improve procedures, tools and data quality.

ANNEX 1: AVAILABLE INFORMATION SOURCES FOR EUROPEAN COMMISSION

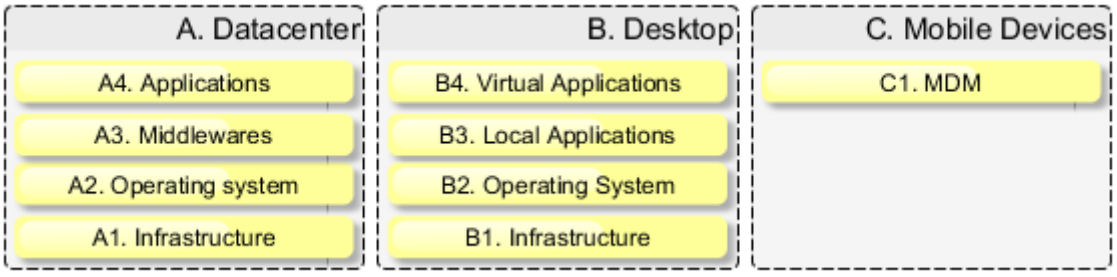
To help in understanding the process of collecting open source information, it may be useful to see what the European Commission did. For the EC, the following information sources were identified:

Figure 17: Information sources (European Commission)



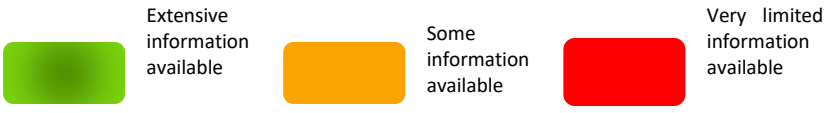
Data collection high level scope

The data collection covered the following high-level scope:



The sources of the inventory therefore covered three major areas: datacentres, desktops and mobile devices.

In the next paragraphs, this figure will be further detailed with the quality of the coverage for each area, indicated by the colours used to represent it:



Limitations

A – Datacentre

DIGIT Datacentre team does not directly control all machines under its responsibility (for example, DIGIT B uses physical / virtual machines not entirely controlled by the DIGIT Datacentre team). Due to the lack of information on the machines (physical or virtual) out of such control, such machines will not fall in the scope of the present study.

A4 - Applications

The applications (hosted or housed) running on the servers present in the DIGIT Datacentre are mostly not controlled by DIGIT. DIGIT handles the requests to make available to the users a specific environment (Infra/OS/middleware) but has no specific rights to consolidate and manage the applications running over these environments.

In order to build and consolidate such inventory, custom scripts may be developed to identify the applications and the specific libraries installed on these servers, at least for hosted servers.

At a first stage, a simple script may explore recursively some of the usual standard installation paths to build an initial inventory. At a later stage, the standard installation paths shall be defined.

A1 - Infrastructure

This layer groups all the possible open source software embedded inside physical devices such as routers, load balancers, SANs, switches, firewalls…

To build an OSS inventory for such devices, manual requests will need to be addressed to manufacturers of these devices. In order to optimize the timeframe, only a shortlist of main devices and appliances will be subject to these manual requests.

B – Desktop

Only Standard workstations & laptops provided by DIGIT were considered here. The BYOD will remain out-of-scope. Similarly, some specific workstations are also excluded as OLAF (Anti-Fraud Office) and JRC (Joint Research Centre).

The list of orders for approved software is stored in the ABAC database, but it is not in an exploitable state, as it is composed of scanned orders in landscape view.

B1 - Desktop infrastructure

In the scope of OSS study, no relevant information can be provided even if some infrastructure information is available through LanDesk inventory tool.

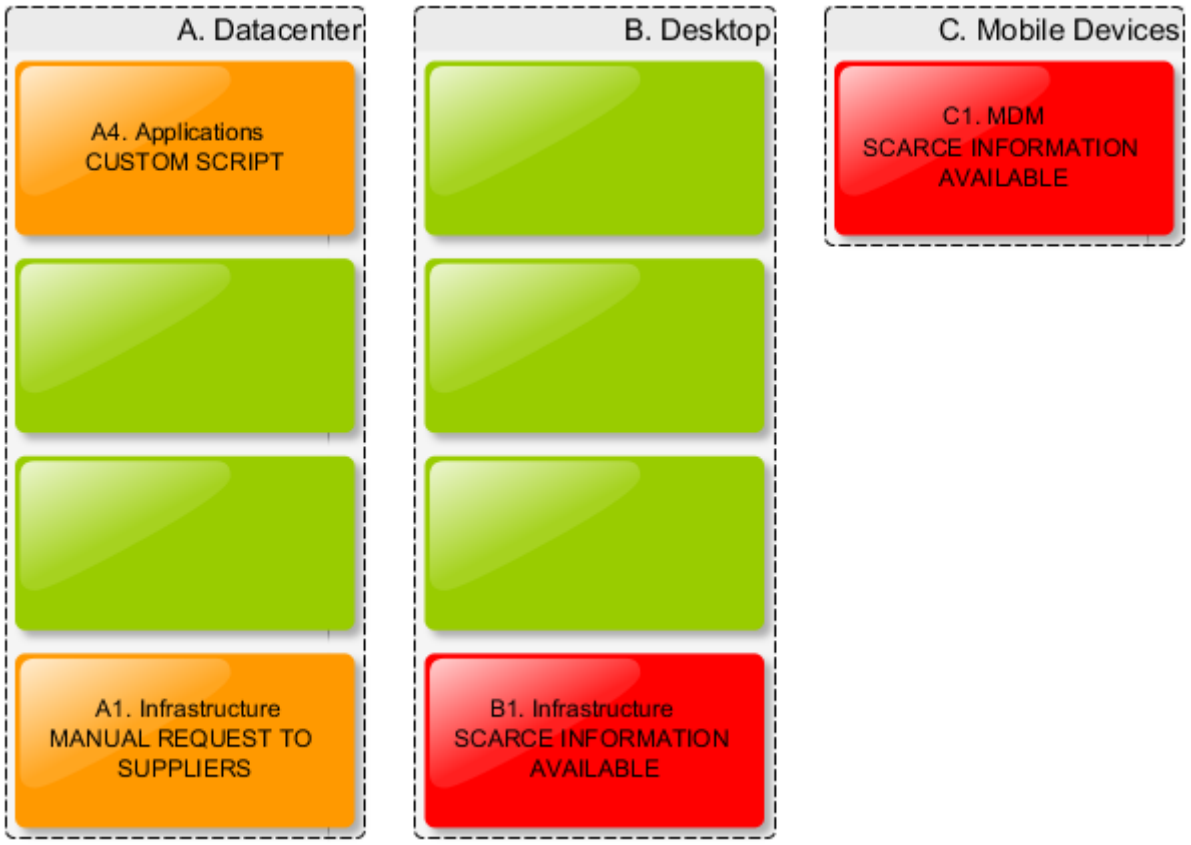
C – Mobile Devices

On mobile devices under provided by DIGIT, only the “MobileIron” agent is installed through MDM channel. This platform, in the configuration purchased by DIGIT, does not include any OSS software. No inventory tool is currently implemented/activated.

As DIGIT does not manage the installed Apps on Mobile devices, this domain will temporarily remain out-of-scope.

The figure below summarizes the approach adopted to manage the limitations to the various areas mentioned above:

Figure 18: High-level approach to manage limitations (European Commission)



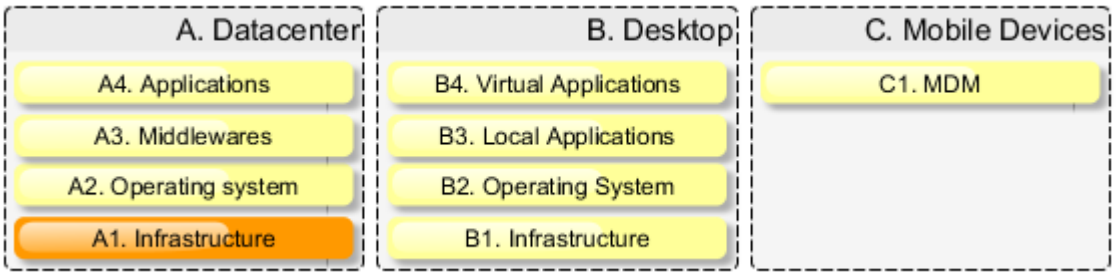
A – Datacenter

A1 - Infrastructure

This layer groups all the possible open source software embedded inside physical devices such as routers, load balancers, SANs, switches, firewalls…

Currently there are no inventories of the software components (firmware) of those devices.

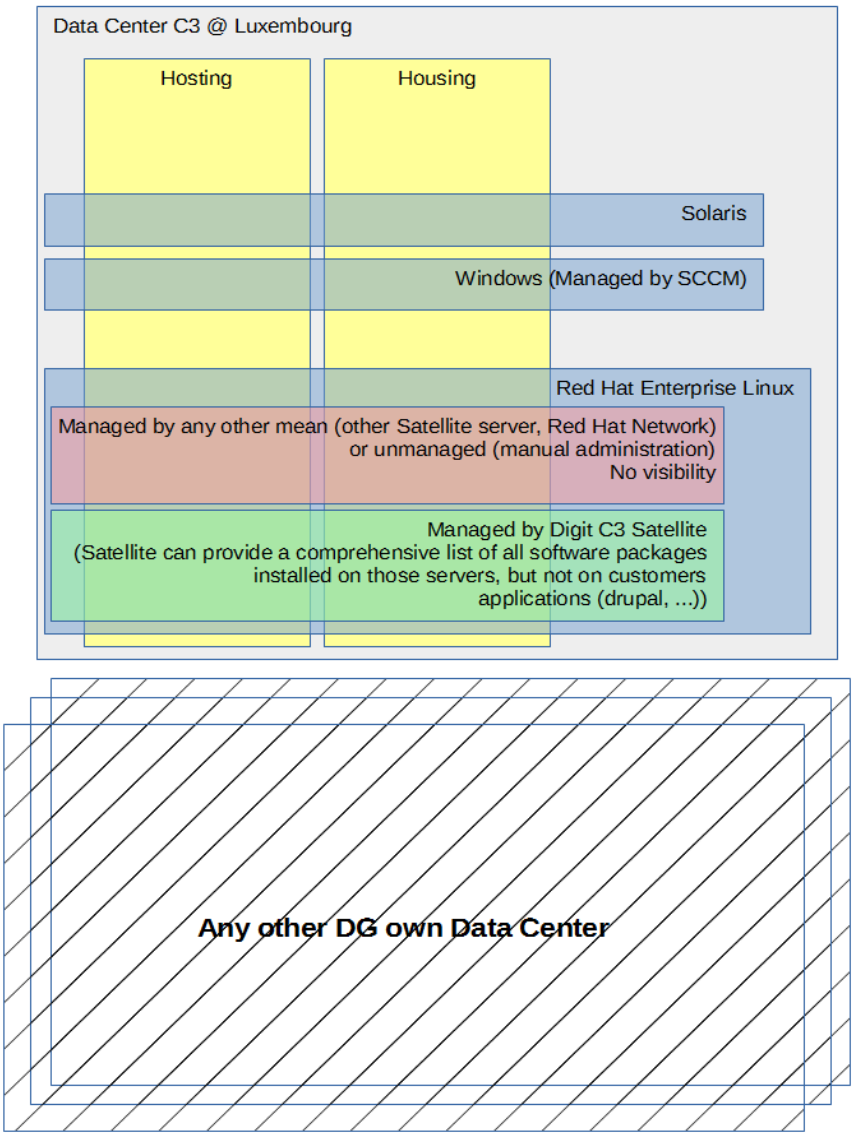
The recommended methodology is to start such an inventory from the list of devices and to contact the vendors in order to get information about the software they run. As this is a long and manual process, it is suggested to perform it based on a very limited set of devices (2 or 3). Even if the output of such a limited sample won't be exploitable as is, the benefit will be that the structure and the process of collecting the information will be in place, and the exercise could be continued later on.



A2 - Operating systems

The following picture describes the situation of the operating systems managed by DIGIT.

Figure 19: Outline of DIGIT-operating systems



DIGIT C3 manages a datacentre in Luxembourg. This datacentre provides hosting and housing services. Among the servers, either in the housed or hosted part, three major operating systems are supported: Red Hat Enterprise Linux, Solaris and Windows:

- Windows servers are managed by Microsoft System Centre Configuration Manager (SCCM);
- Solaris servers are manually managed by the team (i.e. no centralised configuration tool used);
- Linux servers are either managed by the Red Hat Satellite tool from DIGIT C3 (green box in the figure), or are managed by any other means (pink box on the figure), such as:
    - By another Satellite server operated by the customer;

- Directly connected to the Red Hat Network;
- Or unmanaged (manual administration).

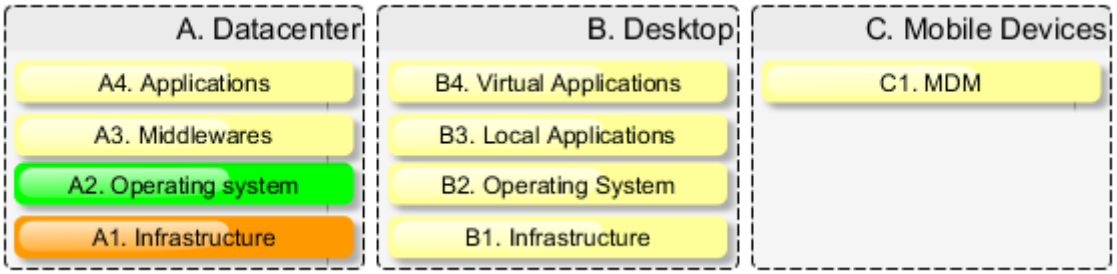
Additionally, other Directorates General also manage their own infrastructure (represented with the hatched boxes).

DIGIT has no visibility on the servers represented by the pink and hatched boxes in Figure 19, as they are not under its control. For these reasons, this methodology will focus on the other areas:

- Windows systems, expected to run little to no open source software, from SCCM exports;
- Solaris systems, from manual export (pkginfo command);
- Linux systems managed by DIGIT C3 Satellite server, from the following commands: spacewalk-report inventory and spacewalk-report system-packages-installed. The latter command outputs the list of all packages, and of their versions, installed on all the systems managed by the satellite server. This includes the libraries installed on the systems.

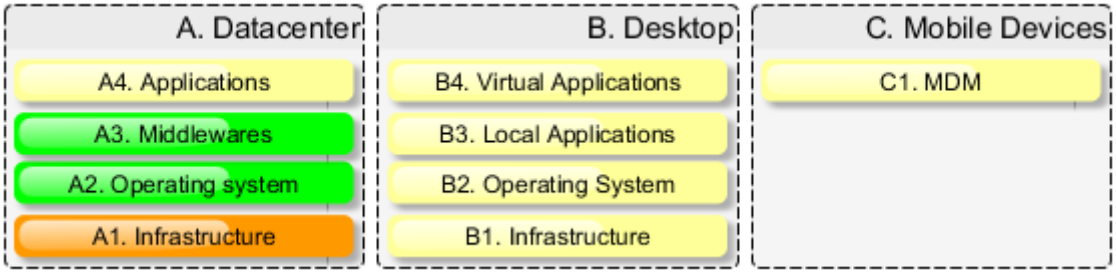
However, only the software installed using the respective software management tools from each OS will be collected (i.e. package manager for Linux and Solaris, and Add/Remove software for Windows). This means that any application added to the system through any other way will not be reported through these methods. This can include:

- Source code compiled on the system;
- Executable copied on the system;
- Applications downloaded from a git/svn repository;
- Webapps for Apache, Tomcat, Weblogic, etc. provided by the users.



A3 - Middleware

The middleware layer includes the application servers or database servers. As this software is installed through the usual package manager of the distribution, the scope and limitations of the previous section 0, “Operating systems”, apply to the present section as well.



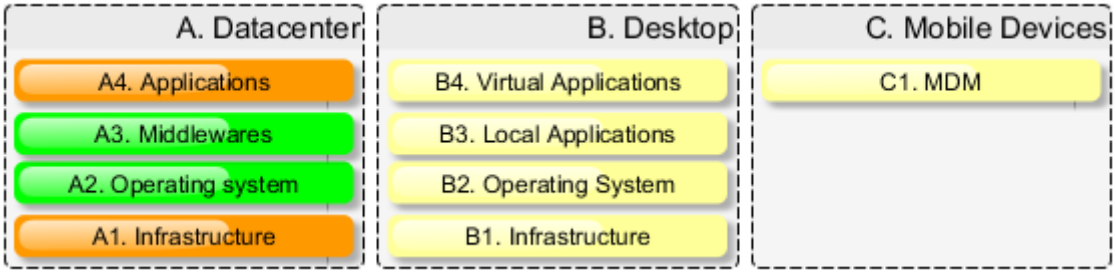
A4 - Applications

The applications are the software hosted by the application server (Tomcat, Weblogic, Apache and Coldfusion). Those applications are provided by the users, and DIGIT has no visibility on them. No inventory currently exists listing the various applications the application servers run. Thus, the only way to keep this layer in the scope cannot be, as for the other layers, to rely on existing tools or inventories, but to develop a script that shall discover the applications inside the application servers.

Based on information gathered from DIGIT C2 technical teams on the standard configuration of various application server types, the script will establish a list of files, looking in specific paths (/var/lib/tomcat…). The collected information may include the file name, the libraries, the version…

However, it is acknowledged that:

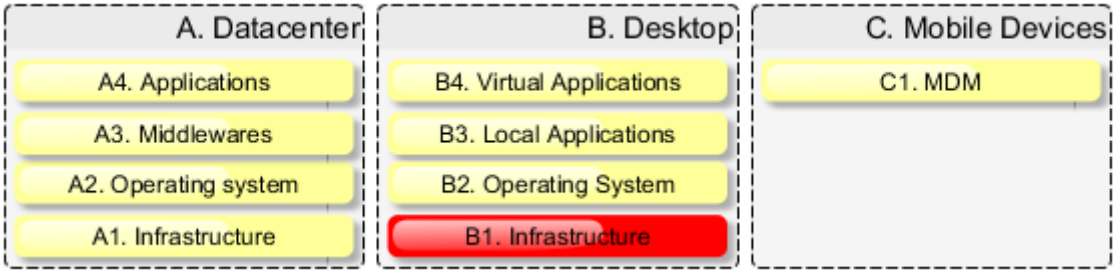
- The configuration of application servers may vary from one to another, thus the script may not see the webapp files if they are stored in a non-standard path;
- The quality of the script result may not provide the requested information on the application (licence type, version, etc.). This will be clarified at the early stages of the testing of the script.



B – Desktop

B1 - Infrastructure

In this section, “infrastructure” includes landline phones, printers, copiers, video conferencing devices and similar items. The firmware of those devices has not been listed and no inventory is currently available to rely on, in order to select the open source components. For this reason, this layer is not covered by the methodology.



B2 – Operating System & B3 – Local Applications

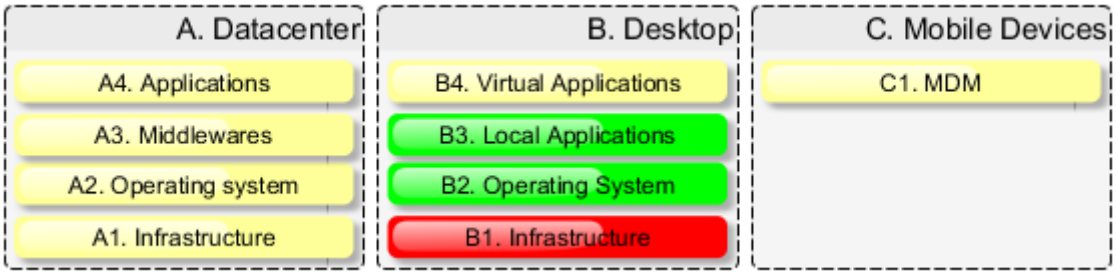
The information on the operating systems and local applications installed on the workstations is managed by Landesk, a tool operated by DIGIT A2.

In the case of typical workstation users not having administrative rights on his computer, there is no risk that a software component not managed by Landesk be installed on the machines.

However, roughly 10% of users do have administrative rights, and so, can install any software on their machine. If they do so, Landesk will discover it and it will appear in a daily report.

Should the admin user decide to disable Landesk on his computer, the system would be automatically banned from Active Directory.

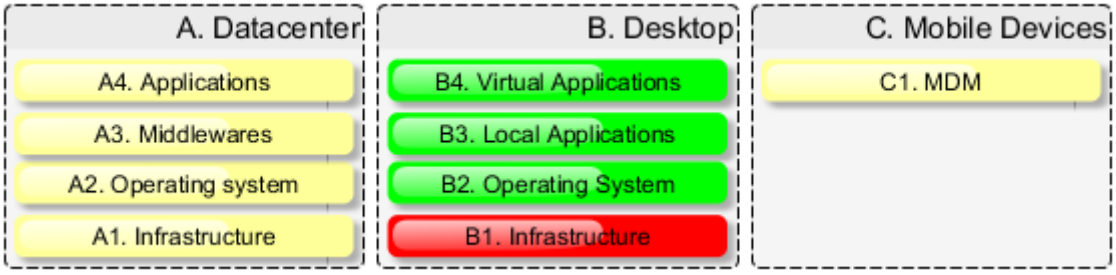
For all those reasons, Landesk is considered a reliable source of information on all the applications installed on the workstations managed by the DIGIT.



B4 – Virtual Applications

Besides the local applications installed on the workstations, DIGIT A2 also provides virtual applications through the Microsoft App-V technology.

The App-V service already can export the catalogue of virtual applications and their usage.

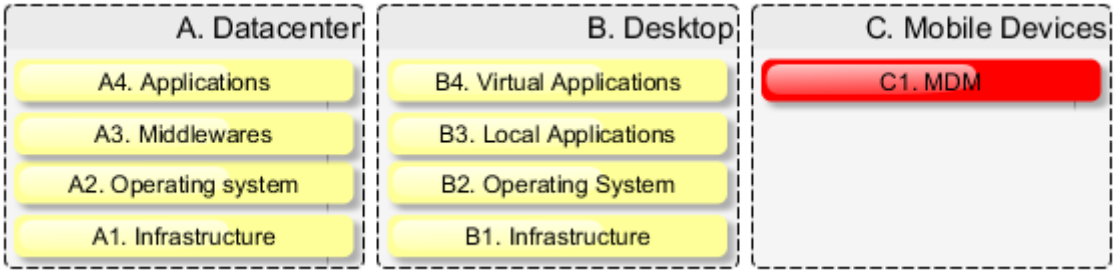


C – Mobile devices

C1 - MDM

The mobile devices are managed by the MDM system. However, the MDM tool cannot collect all the applications installed the mobile devices. Hence, there is no current inventory, nor any current tool in place that would build such an inventory of open source mobile device applications. Moreover, as far as the MDM security layer is concerned (for instance, securing e-mail application), and from the customer's understanding, no substantial open source software is installed.

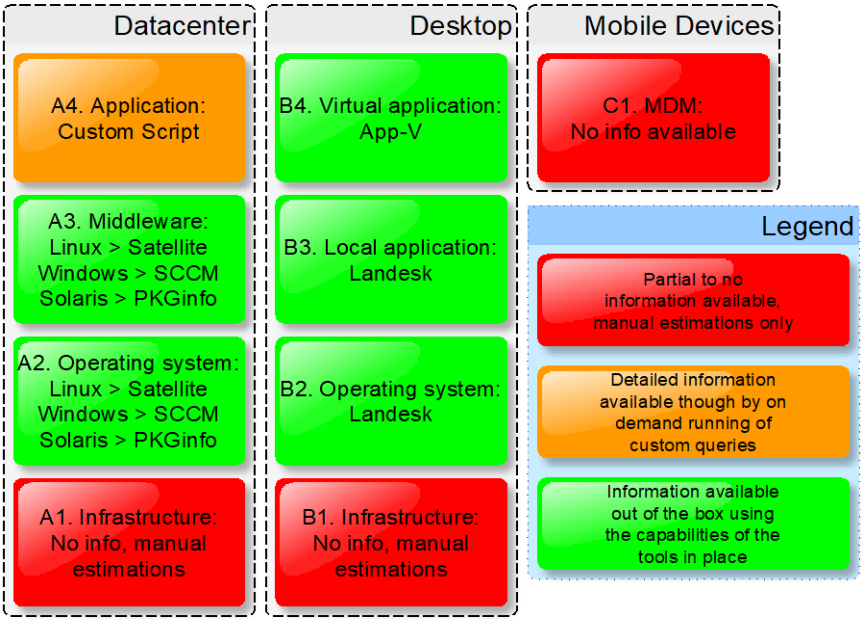
Eventually, even if the methodology described in the present chapter could very well cover the mobile devices, such devices will remain out of scope in the pilot scenario due to the lack of information available at the issue of this release of the document.



Summary of coverage and readiness of the information sources

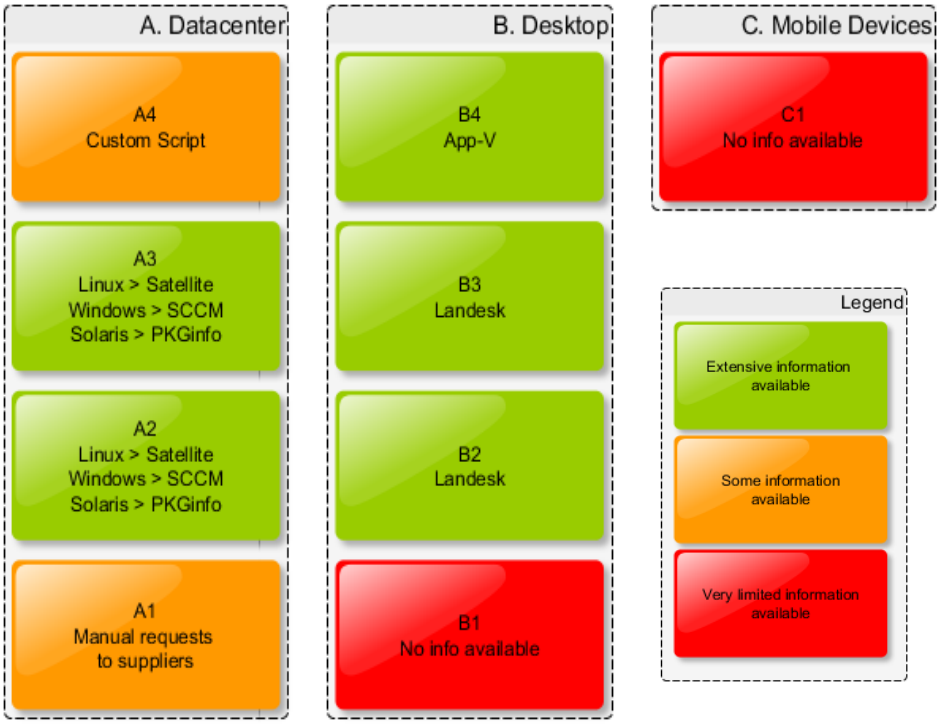
Based on the various sources of information that will be used to build the inventory, the general figure can now be instantiated as follows.

Figure 20: Coverage of inventory with information sources (European Commission)



Another way to qualify the information sources is to rate to what extent the information can be accessed. The following figure gives an overview of this situation.

Figure 21: Readiness of the information sources (European Commission)



ANNEX 2 : METRICS SUSTAINABILITY CRITERIA

A1 Introduction

1.1. Objective of this Document and Intended Audience

This document represents the deliverable 6 included within TASK-04: Final metrics definition.

The objectives of this document are:

- To identify and categorise the aspects that can affect the sustainability of FOSS projects;
- To provide a list of the most relevant metrics that can be used to evaluate the sustainability of FOSS projects;
- To provide a tool to measure these metrics.

This document is addressed to the areas interested in the use of these metrics to evaluate the sustainability of FOSS projects.

1.2. Document Structure

This document consists of the following sections:

- Section 1: **Introduction**, which describes the objectives of this deliverable and the intended audience, the structure of the document and the key success factors.
- Section 2: **Metrics to analyse the sustainability of FOSS projects,** which identifies and describes the metrics and respective categories that can be used to evaluate the sustainability of these projects.
- Section 3: **Metric Measurement Approach, which** describes the process for measuring the metrics.

1.3. Key Success Factors

All the steps described in Section 2 – Metrics to analyse the sustainability of FOSS projects, will ensure the fulfilment of the key success factors related to this deliverable:

- FOSSA outcomes provide new tools for CISO to measure the risk level of open source components.

1.4. Deliverables

1 *Deliverable 4: Analysis of Software Development Methodologies Used in FOSS communities*

B1 Metrics to Analyse the Sustainability of FOSS Projects

If you are going to rely on a FOSS community contribution-based project for your own project, you want to ensure that the community will continue to support it throughout the lifecycle of your project. For any FOSS project, the sustainability of its communities is fundamental for its long term success.

There are many different aspects of a FOSS project that can affect the community sustainability: Good project management, an effective structure of governance, fair licensing, leadership, community activity and performance, and support from external entities are key for healthy and sustainable FOSS communities.

In this section, we will identify the aspects that can affect the sustainability of FOSS projects, and we will design a set of measurable metrics that can be used to evaluate the sustainability of these projects

1.5. Identification and Analysis of the Complete Set of Aspects that Can Affect the Sustainability of the FOSS Projects

In order to identify and analyse the complete set of aspects that can affect the sustainability of the FOSS projects, we researched and gathered information from several sources:

1 Everis FOSS expert team
2 The websites of the communities that were analysed in Deliverable 4
3 Relevant websites and research papers (see Section 4. Bibliographical References)

The information gathered was analysed and, as a result, we defined six categories of metrics, as follows:

1. Community Activity

The overall activity of the community and how it evolves over time is a useful metric category for all open source communities.

The Community Activity provides a first view into how much the community is doing, and it can be used to track the different activities that the community conducts, such as:

1. How many people took part in a relevant amount of a particular activity, like code development, code review, bug fixing?
2. Number of commits, releases, tickets
3. Communications activity (Mailing list, posts, forums, chat history)

4. Number of adoptions/implementations by external organisations / communities
5. Software evolution in terms of code, architecture and bug resolution, which is an indicator of the maturity of the project

2. Performance

Performance allows you to analyse how processes and people are completing their tasks. For example, you can measure:

1. How long processes take to finish, like implementing a new feature, fixing a bug, or conducting code review.
2. The time that it takes to resolve or close tickets
3. The time spent conducting code review

3. Quality and Security

Quality and security are two very important factors to evaluate for the sustainability of a project, for two main reasons:

1. A methodology that checks the quality of the code and ensures that different types of testing are conducted, which will also help the project to be of greater interest to the communities.
2. A project that has included security from the design stage, and implements it throughout its lifecycle, has a much better chance to live longer, because the identified security risks will be mitigated.

4. Demographics and Diversity

Demographics give us an overview of the developers and users around a project, and the companies that engage in it. This includes hosting and support providers, consultancy and customisation services, and companies that integrate the software with other products as part of solutions.

The number of companies involved in a project is an important indicator, since such companies will clearly have a strong interest in the sustainability of the software.

A sustainable project accumulates partners and providers of increasing specialisation. Likewise, if there are signs of service companies moving away from supporting the project this may be an indicator of underlying problems. As a result, projects that have been in production for a long time have a better chance to stay in the long run.

Another factor to take into consideration is the existing knowledge in the external market, regarding the language and platforms used in the project. This factor is extremely important because a project based on a very specific piece of knowledge that is not easily found or not of

interest to the outside community of developers may find it difficult to stay in the long term, therefore directly affecting the sustainability of the project as a whole.

Diversity is an important factor in the resilience of communities. In general, the more diverse communities are—in terms of people or organisations that participate—the more resilient they are. For example, when a company decides to leave a FOSS community, the potential problems that the departure may cause are much smaller if its employees were contributing 5% of the work rather than 85%.

For the organisations that support the project, it is quite useful to look at their diversity in several ways:

1. Do they operate only in one country, or are they geographically spread out? And if so, in different continents?
2. Are they a mix of small and large companies?
3. Do they target a single sector or multiple industry sectors?

5. Governance

Governance is essential for the sustainability and evolution of a FOSS project and its associated communities.

It gives information on:

1. How the project is organised
2. Who is who in the project
3. If a roadmap exists
4. How well documented the project is
5. The licensing structure

6. FOSS Support

Support, either financial, tangible assets or workforce, is needed to ensure the sustainability of the FOSS project and its associated communities. This support can take various forms:

1 Financial

2 Infrastructure assets

3 Human Resources

1.6. Design of a Set Of Metrics

The objective of this task is to define a set of metrics with detailed aspects that will make it easy to measure the sustainability of the FOSS projects.

After the information gathering and the analysis conducted in task *2.1 Identification and analysis of the complete set of aspects that can affect the sustainability of FOSS projects*, a total of 34 metrics were defined and grouped in the six categories identified. Table 1 shows the categories with their corresponding metrics.

Table 1: Categories with their corresponding metrics

Category	No.	Metric Name
Community Activity	1	Code Activity (contributions and contributors)
	2	Release History
	3	Number of Commits
	4	Number of Tickets
	5	Communications (Mailing list, posts, forums, chat history)
	6	Number of Adoptions/Implementations by External Organisations / Communities
	7	SW Evolution (code, architecture, bug/feature)
	8	Programming Language Used
	9	Project Domain (OS, Application SW, IDE, Application servers, Libraries, desktop Environments and frameworks). I.e. Apache, Linux, Eclipse, Mozilla, Ant, GNoME, KDE)
	10	Source Code (repositories like CVS/SVN for code base, GitHub, source forge).
Performance	11	Time to Resolve Tickets
	12	Time Spent in Code Reviews
	13	Pending Work
Quality and Security	14	Security Requirements
	15	Threat Modelling
	16	Security Code reviews
	17	Security Testing
	18	Vulnerability Management
	19	Software Development Methodologies
	20	SLA

Category	No.	Metric Name
Demographics and Diversity	21	Longevity
	22	Real Knowledge Existent in the market of the language and Platforms Used.
	23	People Participating
	24	Organisation Participating
	25	Geographically distributed user community
Governance	26	Project Management
	27	Project Roadmap
	28	Project Structure
	29	Documentation
	30	Licensing
	31	Training
FOSS Support	32	Funding - Monetary
	33	Work force
	34	Infrastructure assets

1.7. Define Metrics Criteria

In order to design the forms that will be used to compile all the information for each metric, we defined the following criteria:

1. **Metric Name**: Descriptive name of the metric.
2. **Description:** what the metric should accomplish.
3. **Unit of Measurement:** it refers to the way the metric will be measured: a number, a maturity level, etc.
4. **Method:** it defines how the metric will be measured.
5. **Measurement:** it defines the actual measurement of the metric, i.e. the maturity level.
6. **Result:** the formula applied to measure the metric.

All the information of each metric is documented in the following forms, grouped in one of the 6 categories defined in *Task 2.1 Identification and analysis the complete set of aspects that can affect the sustainability of FOSS projects*

2.3.1. Community Activity

M1	Metric Name	Code Activity (contributions and contributors)
Description	For a project to be sustainable it must have contributors, and its codebase needs to be evolving. One can track this by looking at the project's revision control system and looking at the pattern of contributions. This metric measures the amount of committers that contribute to a majority of the commits in the project.	
Unit of Measurement	Ratio of contributors	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the pattern of contributions, to identify the number of contributors who submitted 80% of the total contributions in a specific period of time (mostActiveContributors80). Formula to calculate the ratio of contributors: **Contributors ratio = (mostActiveContributors80 / (mostActiveContributors80 + 1% x totalContributors)) x (totalContributors/ totalContributors + 10)**	
Measurement	1. **Very split:** Ratio value within the upper 20% of the maximum ratio 2. **Split: Ratio** value ranked between 79% and 60% of the maximum ratio 3. **Average:** Ratio value ranked between 59% and 40% of the maximum ratio 4. **Dependant:** Ratio value ranked between 39% and 21% of the maximum ratio 5. **Very dependant**: Ratio value within the lowest 20% of the maximum ratio	

M2	Metric Name	Release History
Description	This metric measures the approach followed for releases that provide information on the update frequency 1. Regular releases (disruption in the cycle might indicate sustainability or governance issues, in which case the best way to find out is to go into the project communications area and see if there is an issue) 2. Releases on a “need to have" basis. Some projects make releases as and when they feel ready, so they do not follow an established frequency. 3. When do releases occur? On the weekends (suggesting a hobby) or during the week (suggesting a business)?	
Unit of Measurement	Release frequency	
Method	Look at the release pattern for a certain period of time	
Measurement	**1** **Optimised:** formal approach, regular releases are planned and delivered periodically, with the exception of security fixes. **2** **Managed:** informal approach, release is published when development objectives are achieved. **3** **Initial**: informal approach, release is published without clear definition criteria.	

M3	Metric Name	Number Of Commits
Description	The number of commits gives a general idea about the volume of the development effort.	
Unit of Measurement	Number of commits	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the number of code commits done by contributors during - last year. The number of most active contributors will be those that submitted 50% of the total contributions Formula to calculate the ratio: **Commits Ratio = (nCommitsLastYear / nNumberCommitsLastYearTopPopularGitHubRepository) *100**	
Measurement	**1** **Very active:** Ratio value within the upper 51% of the maximum ratio **2** **Active**: Ratio value ranked between 26% and 50% of the maximum ratio **3** **Average:** Ratio value ranked between 6% and 25% of the maximum ratio **4** **Inactive**: Ratio value ranked between 1% and 5% of the maximum ratio **5** **Very Inactive:** Ratio value within the lowest 1% of the maximum ratio	

M4	Metric Name	Number Of Tickets
Description	The number of tickets opened provides information about how many bugs are reported or the new functionalities that are proposed.	
Unit of Measurement	Ratio of tickets created	
Method	This analysis will be carried out by checking the community's main tasks or ticket repository. The information to look for will be when the tickets are created	
Measurement	**1** **Very active**: there are, at least, 10 tickets created in the last week. **2** **Active**: there are, at least, 10 tickets created in the last two weeks. **3** .**Average:** there are, at least, 10 tickets created in the last month. **4** **Inactive**: there are, at least, 10 tickets created in the last three months. **5** **Very Inactive:** rest of the values	

M5	Metric Name	Communications (Mailing list, posts, forums, chat history)
Description	The number of messages in mailing lists or posts in forums gives an idea of how many discussions are being held in public. However, this metric needs to differentiate the types of activities that are conducted in the communications, which can range from some serious discussions to unnecessary flame wars (in this case, the communication channel should not be accounted for).	
Unit of Measurement	Number of active communication channels	
Method	This analysis will be carried out by checking official communication channels provided by the community. The information to look for will be the number of active communication channels used by the community.	
Measurement	**1** **Optimised:** More than three communication channels are used (different mailing lists, IRC, wiki, user forums and web post are used for the project). **2** **Managed:** At least three communication channels are used in the project. **3** **Initial:** less than three channels are used for exchanging information.	

M6	Metric Name	Number of Adoptions/Implementations by External Organisations / Communities
Description	Software downloads provide information about the global interest in the project Each distribution platform provides its own metrics to describe popularity. For example, on GitHub, watchers, stars, and forks are the strongest indicators of a project's popularity and use. On WordPress.org, you can see the number of downloads a plugin receives, as well as its average user rating. If distributed via package manager (e.g., Rubygems, NPM), you can see the number of installs. These indicators show how much the project is used.	
Unit of Measurement	Interest level	
Method	This analysis will be carried out by checking distribution platforms. The information to look for will be the identification and measurement of the interest, in order to rank it within the levels defined. This level of interest will be measured by means of doing the following assessment: Taking the 5 most downloaded/popular projects, an average will be assessed (Av). The level of popularity (using the Alexa ranking) of the project or the number of downloads (P) will be divided by that average. The result is the adoptions ratio (Ra). **Ra = P / Av**	
Measurement	**1** **Very Interesting**: The ratio value is larger than 1 **2** **Interesting**: The ratio value is between 1 and 0,51 **3** **Normal** The ratio value is between 0,50 and 0,26 **4** **Disappointing**: The ratio value is between 0,25 and 0,11 **5** **Very disappointing:** The ratio value is smaller than 0,10	

M7	Metric Name	SW Evolution (code, architecture, bug/feature)
Description	This metric evaluates the evolution level of the software development cycle: 1 Code development follows a methodology 2 Improvements were made to the architecture supporting the software development 3 Improvements were made to the bug fixing process	
Unit of Measurement	Maturity level	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the project's development lifecycle and the evaluation of these three parameters: 1 Code development follows a methodology 2 Architecture Improvements 3 Improvements bug fixing process	
Measurement	**1** **Optimised:** The community applies all three parameters **2** **Addressed:** They accomplish two of the three parameters analysed **3** **Partially Addressed:** They accomplish one of the parameters **4** **Initial:** They don't address any of the parameters analysed	

M8	Metric Name	Programming Language Used
Description	This metric evaluates the use of a stable and widely used programming language	
Unit of Measurement	Use of the programming language	
Method	This analysis will be carried out by checking the community website and wiki. The goal is to measure the maturity of the programming language used using TIOBE Index as indicator. http://www.tiobe.com/tiobe_index	
Measurement	**1** **Very popular:** First 5 entries from TIOBE **2** **Popular**: Languages ranked from 6 to 15 from TIOBE **3** **Average**: Languages ranked from 16 to 20 from TIOBE **4** **Unusual:** Rest of the languages from TIOBE	

M9	Metric Name	**Project Domain (OS, Application SW, IDE, Application servers, Libraries, desktop Environments and frameworks. I.e. Apache, Linux, Eclipse, Mozilla, Ant, GNoME, KDE…)**
Description	The sustainability of the projects increases if they belong to the most common domains: Operating Systems (OS), Application Software, Integrated Development Environments (IDE), Application Servers, Libraries, Desktop Environments and Frameworks. Examples of projects in these domains include Linux, Eclipse, Apache, Ant, Mozilla, GNOME, KDE, and ArgoUML This metric will evaluate if the project belongs to one of these domains.	
Unit of Measurement	Domain type	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the project's domain: 1. Common: Operating Systems (OS), Application Software, Integrated Development Environments (IDE), Application Servers, Libraries, Desktop Environments and Frameworks. Example projects under these domains include Linux, Eclipse, Apache, Ant, Mozilla, GNOME, KDE, and ArgoUML. 2. Not common	
Measurement	**1** Common Domain **2** Not common domain	

M10	Metric Name	**Source Code (repositories like CVS/SVN for code base, GitHub, source forge).**
Description	This metrics measures if the developer uses existing repositories to produce quality code. 1. Repositories maintaining the code base (e.g., CVS/SVN, change log) are data sources that contain information on the underlying software and its development process, ensuring that everything is commented. Comments are clear and free of misspellings, and the project includes extensive tests. 2. External sources, like SourceForge.net, repositories hosting thousands of FOSS projects	
Unit of Measurement	Position in Alexa ranking	
Method	This analysis will be carried out by checking the Alexa ranking for open source project hosting: http://www.alexa.com/topsites/category/Computers/Open_Source/Project_Hosting	
Measurement	**1** **Popular Repository**: 1st, 2nd, 3rd positions **2** **Common Repository:** 4th, 5th, 6th positions. **3** **Independent Repository:** From 7th up to 15th positions. **4** **Marginal Repository**: Not ranked in the first 15 positions in Alexa ranking.	

2.3.2. Performance

M11	Metric Name	Time to Resolve Tickets
Description	This metric measure the Time it takes to resolve or close tickets. This metric shows how the project is reacting to new information that requires another action, such as fixing a reported bug or implementing a requested new feature.	
Unit of Measurement	Average period to resolve a ticket	
Method	This analysis will be done by looking at the software development statistics during a certain period of time (for example, 6 months) The formula to calculate the average time is as follows: **Average time = sum(ticket solving time)/number of tickets**	
Measurement	**1** **Optimised:** Average_time < 5 days **2** **Defined:** 10 days > Average_time >= 5 days **3** **Managed:** 15days > Average_time >= 10 days **4** **Basic**: 15days <= Average_time **5** **No data about this**	

M12	Metric Name	Time Spent in Code Reviews
Description	These metric measures the Time spent in code reviews —from the moment a change to the code is proposed, to the moment it is accepted—, and it shows how long it takes to upgrade a proposed change to the quality standards expected by the community. Other metrics deal with how well the project is coping with pending work, such as the ratio of new to closed tickets, or the backlog of still incomplete code reviews. Those parameters tell us, for example, whether or not the resources put into solving issues are enough.	
Unit of Measurement	Average time to do code reviews. (Considering the minimum number of code reviews before being accepted or rejected)	
Method	This analysis will be done by looking at the annual community reports. The formula to calculate the average time is as follows: **Average time = sum(code review acceptance time)/number of code reviews**	
Measurement	**1** **Optimised:** Average_time <= 3 days **2** **Defined:** 7days>= Average_time > 3 days **3** **Managed:** 15days>= Average_time > 8 days **4** **Basic:** Average_time > 15 days **5** **No data about this**	

M13	Metric Name	Pending Work
Description	This metric measures the ratio of new to closed tickets, or the backlog of incomplete code reviews This parameter is also an indicator of whether or not the resources put into solving issues are enough.	
Unit of Measurement	Ratio of new and closed tickets	
Method	The ratio between closed tickets (issues) and new ones will be done, if possible, taking a month as timeframe. The formula to calculate this ratio is as follows: **SolvingRatio = NewTickets/ClosedTickets * 100**	
Measurement	1 **Optimised:** SolvingRate <=33% 2 **Controlled:** 33% < SolvingRate <= 66% 3 **Managed:** 66% < SolvingRate <= 100% 4 **Overloaded:** 100% > SolvingRate	

2.3.3. Quality and Security

M14	Metric Name	Security Requirements
Description	This metric measures the existence and maturity level of the definition of security requirements in the early stages of the SDLC	
Unit of Measurement	Maturity level	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the definition of security requirements. If possible, the information will be verified by contacting the community.	
Measurement	**1** **Optimised:** Specific requirements (defined at the initial phases) **2** **Defined**: Within business requirements **3** **Managed:** Security requirements defined as needed **4** **Initial:** No Security Requirements	

M15	Metric Name	Threat Modelling
Description	This metric measures the existence and maturity level of threat modelling	
Unit of Measurement	Maturity level	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the definition of the approach to threat modelling. If possible, the information will be verified by contacting the community.	
Measurement	1 **Optimised:** They have threat modelling and countermeasures are implemented or in the process of being implemented (managed) 2 **Managed**: No formal threat modelling, however some countermeasures are implemented (from previous experiences) 3 **Initial:** No threat modelling	

M16	Metric Name	Security Code Reviews
Description	This metric measures the existence and maturity level of security procedures such as code reviews	
Unit of Measurement	Maturity level	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the definition of the security code review process (security code reviews is being responsibly conducted). If possible, the information will be verified by contacting the community.	
Measurement	**1** **Formal:** Security code reviews conducted by a specific team **2** **Informal:** Security code reviews conducted by community members **3** **No** security code reviews conducted	

M17	Metric Name	Security Testing
Description	This metric measures the existence and maturity level of security procedures such as security testing (white box /black box)	
Unit of Measurement	Maturity level	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be if the definition of the security testing process (security testing is being conducted, specifying in which SDLC phase). If possible, the information will be verified by contacting the community.	
Measurement	**1** **Optimised:** Security testing conducted during development **2** **Defined:** Security testing conducted during testing **3** **Managed**: Security testing conducted before release **4** **Basic:** No security testing or conducted after release (user finds a vulnerability)	

M18	Metric Name	Vulnerability Management
Description	This metric measures the existence and maturity level of vulnerability management.	
Unit of Measurement	Maturity level	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the definition of the vulnerability management process. If possible, the information will be verified by contacting the community.	
Measurement	1 **Optimised:** Vulnerability management conducted by a dedicated team 2 **Defined:** Vulnerability management conducted as part of the security team´s responsibilities 3 **Managed:** Vulnerability management conducted by a closed group (community leaders, vulnerability stakeholders, trusted members)	

M19	Metric Name	Software Development Methodology
Description	This metric measures the existence and maturity level of the software development methodologies used	
Unit of Measurement	Maturity level	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the software development methodology used in the project. If possible, the information will be verified by contacting the community.	
Measurement	**1** **Optimised:** Use of a standard methodology (i.e. Scrum, Agile, Kanban, Waterfall) **2** **Managed:** Use of their own documented methodology **3** **Basic:** Random, individual contributions	

M20	Metric Name	SLA
Description	An SLA that defines the parameters for ticket resolution, bug fixing, etc… This metric measures the existence and maturity level of an SLA	
Unit of Measurement	Maturity level	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the definition of an SLA in the project. If possible, the information will be verified by contacting the community.	
Measurement	1 **Formal:** An SLA exists and is managed 2 **Informal:** An SLA does not exist, however, there is an informal procedure to resolve the issues	

2.3.4. Demographics and Diversity

M21	Metric Name	Longevity
Description	This metric measure how long the project has been in a "live" or production status. Some open source projects are long-lived, leading more conservative organisations to adopt the software, and maintain its use for longer, and resulting in a longer-term investment in its sustainability. If a project has survived long enough to undergo several technology replacement cycles, this is a good indication that it is going to be around for years to come. The warning signs appear when there seems to be subsequent migrations from one project community to another. Eventually, even a large, mature project will start to suffer if this happens.	
Unit of Measurement	Start year of the project	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the starting date of the project. If possible, the information will be verified by contacting the community.	
Measurement	1 **Reference Project in FOSS environment:** Project started before 2000 2 **Veteran Project:** Project started between 2000 and 2005 3 **Experimented Project:** Project started between 2005 and 2010 4 **Adult Project:** Project started between 2010 and 2015 5 **Beginner Project:** Project started after 2015	

M22	Metric Name	**Real Knowledge Existent in the Market about the Language and Platforms Used.**
Description	The PYPL PopularitY of Programming Language Index is created by analysing how often language tutorials are searched on Google: the more a language tutorial is searched, the more popular the language is assumed to be. It is a leading indicator. The raw data comes from Google Trends.	
Unit of Measurement	PYPL index	
Method	This analysis will be carried out by checking the website: http://pypl.github.io	
Measurement	1 **Popular programming language**: PYPL share >10% 2 **Common programming language**: 10% >= PYPL share >5% 3 **Specialised programming language**: 5%>= PYPL share	

M23	Metric Name	People Participating
Description	This metric evaluates the different groups and number of active members that are participating as contributors or supporters of this community. Having a diversity of contributors indicates that there's a community of users who rely on and care about improving the software. Contributors need not be only technical. Look for those contributing to documentation processes, posting on support forums, or filing issues and feature requests. They can be grouped as: 1 Developers 2 Documenters 3 Supporters	
Unit of Measurement	Number of active groups	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the number of working groups or teams within the community. If possible, the information will be verified by contacting the community.	
Measurement	**1** **High:** Three or more groups **2** **Medium**: Two groups **3** **Low:** One group	

M24	Metric Name	Organisations Participating
Description	This metric evaluates the number of different organisations that are participating as contributors or supporters of this community. There are many open source projects that can meet the above mentioned criteria, but if none of the peers are using the project (or haven't even heard of it), that could be a major red flag. Many companies proudly showcase the open source projects they're built on, and Google searches can often reveal those that don't.	
Unit of Measurement	Levels, indicating the number and relevance of supporting organisations	
Method	This analysis will be carried out by checking community website and wiki. The information to look for will be the organisations that support the project. If possible, the information will be verified by contacting the community.	
Measurement	**1** **Level 1**: Several big technological organisations participate in the project **2** **Level 2:** Only one big technological organisation participates in the project **3** **Level 3:** Several organisations participate in the project **4** **Level 4:** One organisation participates in the project **5** **Level 5:** No participating organisations	

M25	Metric Name	Geographically Distributed User Community
Description	This metric evaluates how geographically spread out the user community is.	
Unit of Measurement	Number of continents	
Method	This analysis will be carried out by checking the community website and wiki. Identify the home country/continent of the current top contributors (100).	
Measurement	**1** **Geographically widely spread:** more than 4 continents **2** **Geographically spread:** Between 2 and 4 continents **3** **Geographically concentrated:** Less than 2 continents	

2.3.5. Governance

M26	Metric Name	Project Management
Description	This metric measures the existence and maturity level of the project management cycle	
Unit of Measurement	Maturity level	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the project's management cycle conducted by the community. If possible, the information will be verified by contacting the community.	
Measurement	**1** **Optimised:** Project Management is defined and implemented **2** **Defined:** Project Management is defined and documented, but does not completely follow the agreed methodology **3** **Managed:** Project management is conducted in an informal way **4** **Initial:** Project management is conducted as needed	

M27	Metric Name	Project Roadmap
Description	This metric evaluates the existence and maturity level of a project roadmap	
Unit of Measurement	Maturity level	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the community's project roadmap. If possible, the information will be verified by contacting the community.	
Measurement	**1** **Optimised:** Project roadmap is defined and implemented **2** **Defined:** Project roadmap is defined and documented, but does not completely follow the agreed methodology **3** **No** project roadmap	

M28	Metric Name	Project Structure
Description	This metric evaluates if there is a formal structure for the project. 1 How is the project organised? 2 Who is behind the project, in terms of number of people? 3 Are they fully committed to the project or is it a partial assignment, done on a voluntary basis?	
Unit of Measurement	Documentation coverage defined in 3 levels	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the project structure (organogram). If possible, the information will be verified by contacting the community.	
Measurement	**1** **Optimised:** A formal structure with roles and responsibilities is defined, following an enterprise approach **2** **Managed:** An informal structure, with roles and responsibilities defined, although it may not be complete (i.e. no security roles) **3** **Initia**l: Only leader and contributor roles are defined.	

M29	Metric Name	Documentation
Description	This metric will indicate the level of the documentation existent in the project. 1 Is it a readme file or a dedicated documentation site? 2 Does it have technical documentation that covers how to install, and specifies requirements, dependencies? 3 Does it have a user manual? 4 Does it have general documentation?	
Unit of Measurement	Documentation coverage defined in 3 levels	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the documentation of the project. If possible, the information will be verified by contacting the community.	
Measurement	**1** **Full documentation:** a) developer guides (code style, code review, security review, development environment), b) user manual, c) technical manual (for system administrator), d) support wikis. **2** **Partial documentation:** Only main documentation is developed, user-oriented and for developers **3** **Basic documentation**: Only two types of documentation are developed, mainly user-oriented	

M30	Metric Name	Licensing
Description	This metric will indicate how serious the project is in terms of providing intellectual property. 1 Is the project properly licensed? 2 What type of license is provided? 3 Does it contain a license file or just a reference to a license in the readme? 4 Do files contain the proper headings, where required?	
Unit of Measurement	Intellectual property level	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the license file of the project. If possible, the information will be verified by contacting the community.	
Measurement	**1** **Optimised:** Project has a license history, up-to-date license that contains proper headings **2** **Defined**: Project incorporates a license file with proper headings. **3** **Managed:** Project incorporates a license file without proper headings.	

M31	Metric Name	Training
Description	This metric measures if the project has provisions for regular training to ensure the quality of project deliverables	
Unit of Measurement	Training programmes coverage defined in 3 levels	
Method	Identification of the regular training provided by the project	
Measurement	**1** **Optimised:** Project has a complete set of documentation for newcomers (How to contribute, how community works, tools), and a mentor is assigned to help them to get started. **2** **Managed:** Project has a complete set of documentation for newcomers (How to contribute, how community works, tools) **3** **Basic:** Project has some informal information for newcomers (How to contribute, how community works, tools)	

2.3.6. FOSS Support

M32	Metric Name	Funding - Monetary
Description	This metric measures if the project is being supported by some kind of monetary funding from an external source	
Unit of Measurement	Funding level	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the "Thanks" or "acknowledgment" part in the project/community website. If possible, the information will be verified by contacting the community.	
Measurement	1 **Optimised:** Different external organisations fund the project directly, or it is funded from a private organisation that does business with the FOSS 2 **Managed:** Different external organisations fund different projects in the same community. 3 **Basic:** No funding by third-party organisations, just individual donations.	

M33	Metric Name	Workforce
Description	This metric measures if the project is being supported by external volunteers who provide support in development, documentation or issue management tasks	
Unit of Measurement	Workforce level	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the "Thanks" or "acknowledgment" part in the project/community website. If possible, the information will be verified by contacting the community.	
Measurement	1 **Optimised:** there are paid human resources in all areas of the project, working exclusively in that area. Volunteers can also be part of the project 2 **Dedicated:** there are paid human resources working in one or more areas of the project. Volunteers can also be part of the project 3 **Volunteering:** There are only volunteers in the project.	

M34	Metric Name	Infrastructure Assets
Description	This metric measures if the project is being supported by the provision of equipment or software licenses from an external source This provision can come from a monetary donation or an actual asset donation	
Unit of Measurement	Type of infrastructure	
Method	This analysis will be carried out by checking the community website and wiki. The information to look for will be the "Thanks" or "acknowledgment" part in the project/community website. If possible, the information will be verified by contacting the community.	
Measurement	**1** **Dedicated:** Community is the infrastructure owner **2** **Mixed:** Dedicated and shared infrastructure. **3** **Shared:** Infrastructure assets are shared with other communities	

C1 Metrics measurement approach

Following the criteria defined and agreed upon in Section *2.3 Define Metrics Criteria*, we conducted the following activities to measure the metrics designed in Section *2.2 Design of a Set of Metrics*:

3.1. Tool to measure the metrics

1. Development of an Excel sheet, with all the metrics that were defined in Section *2.2 Design of a Set of Metrics* and all the metrics criteria defined in Section *2.3 Define Metrics Criteria*
2. Definition of a unit of measurement for each metric
3. Development of method to measure each metric. This method could be a formula to calculate the ratio of two values, or data obtained from the project website.
4. Each measurement is normalised, so all the metrics can be analysed on the same scale, in a quantitative way
5. To show the results in a graphic way, easy to understand, a set of example graphs are produced, to represent the results in a graphical way.

   To view the measurement tool, click on the icon below:

Metrics measurement tool

3.2. Frequency of the measurement

Bitergia, a company focused on software development analytics, indicates in the article 'On the Importance of Quarterly Reports: OPNFV and OpenStack as use cases', that measurement of all the metrics should be conducted at least on a quarterly basis.

3.3. Responsible for the measurement

A team should be appointed to conduct the metric measurement of the selected FOSS projects.

For successful measurements, the team should have a suitable level of relevant skills and experience.

These skills include:

- Analytical thinking, to notice discrepancies and inconsistencies in available information.
- Communication skills, oral and written, to ensure that important information is shared with others appropriately and to communicate results
- Specific knowledge for particular categories, e.g. project management knowledge for the governance category, security knowledge for the Quality and Security category, etc.
- Experience in conducting metrics evaluations
- Teamwork

3.4. Results

Once the measurement is conducted, 8 types of graphs can be produced, as follows:

1. One for each of the categories defined in Section 2.1 Identification and Analysis of the Complete Set of Aspects that Can Affect the Sustainability of the FOSS Projects
2. A graph comparing each community against all 6 categories.

A sample of the graphs is shown in Figures 1 through 7

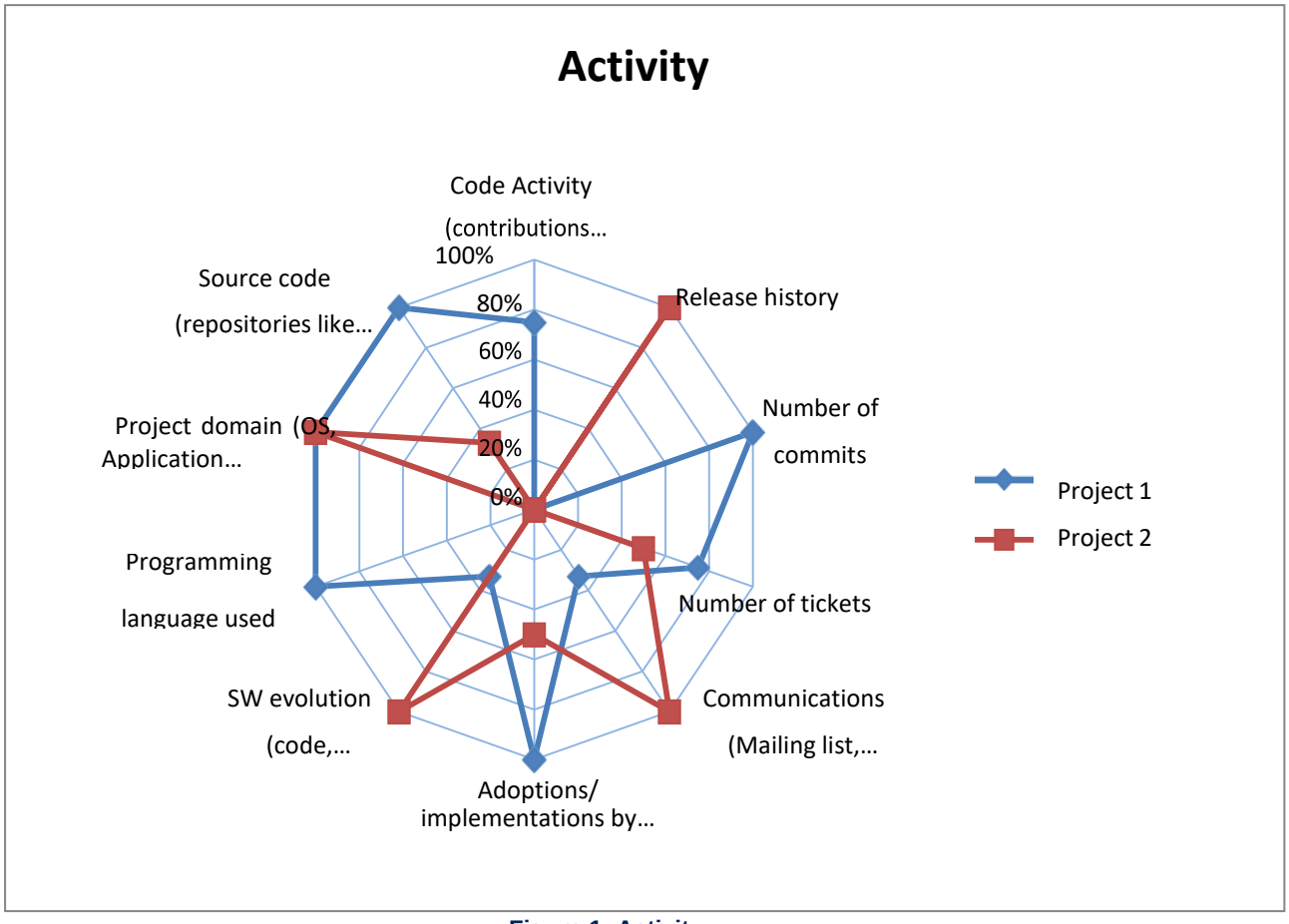


Figure 1: Activity

Figure 2: Performance

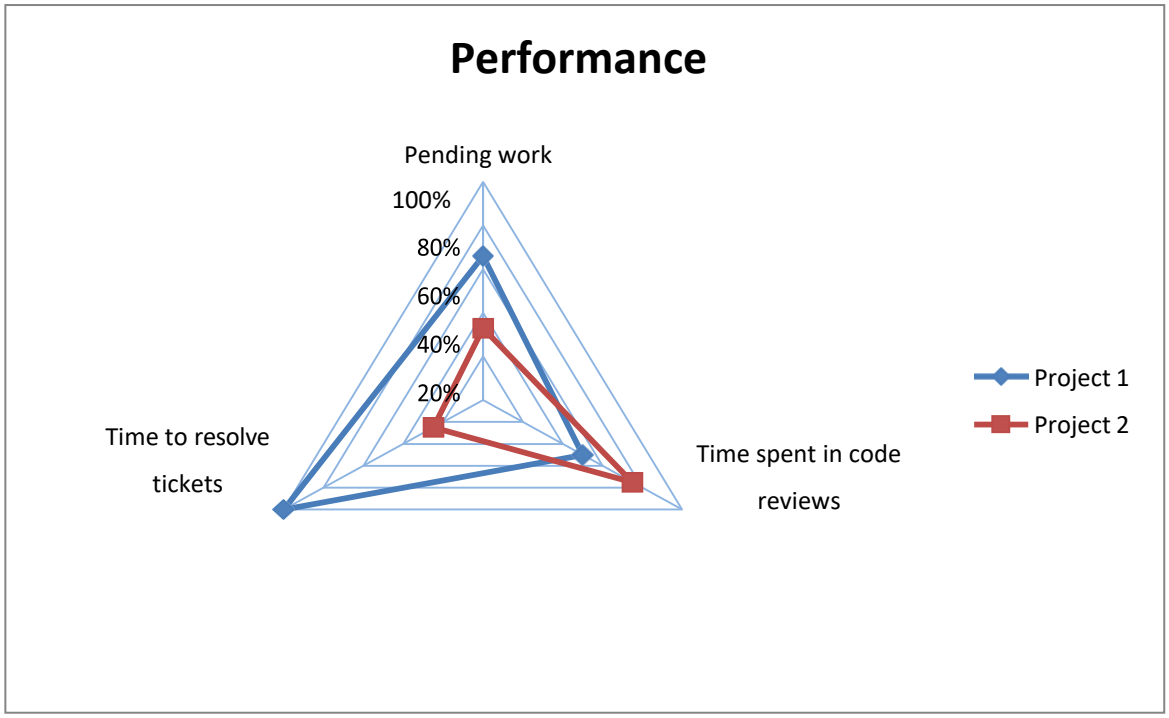


Figure 3. Quality and Security

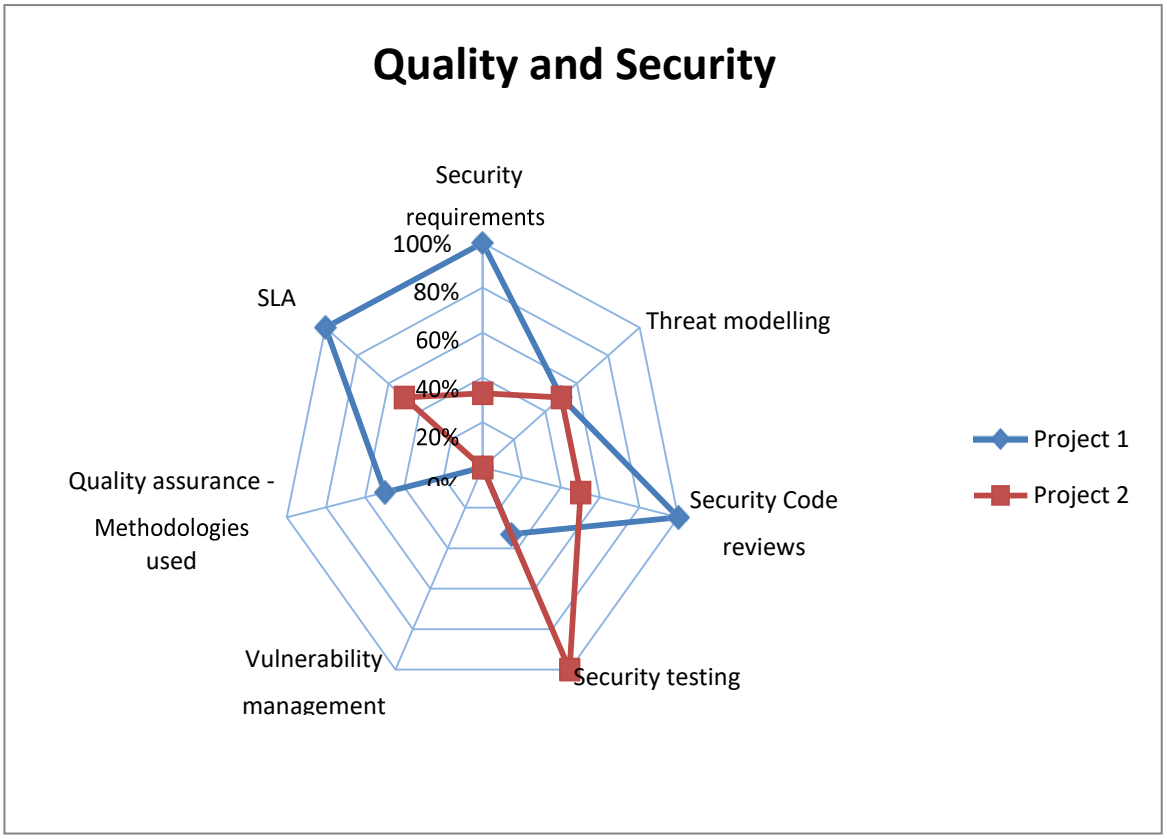


Figure 4: Governance

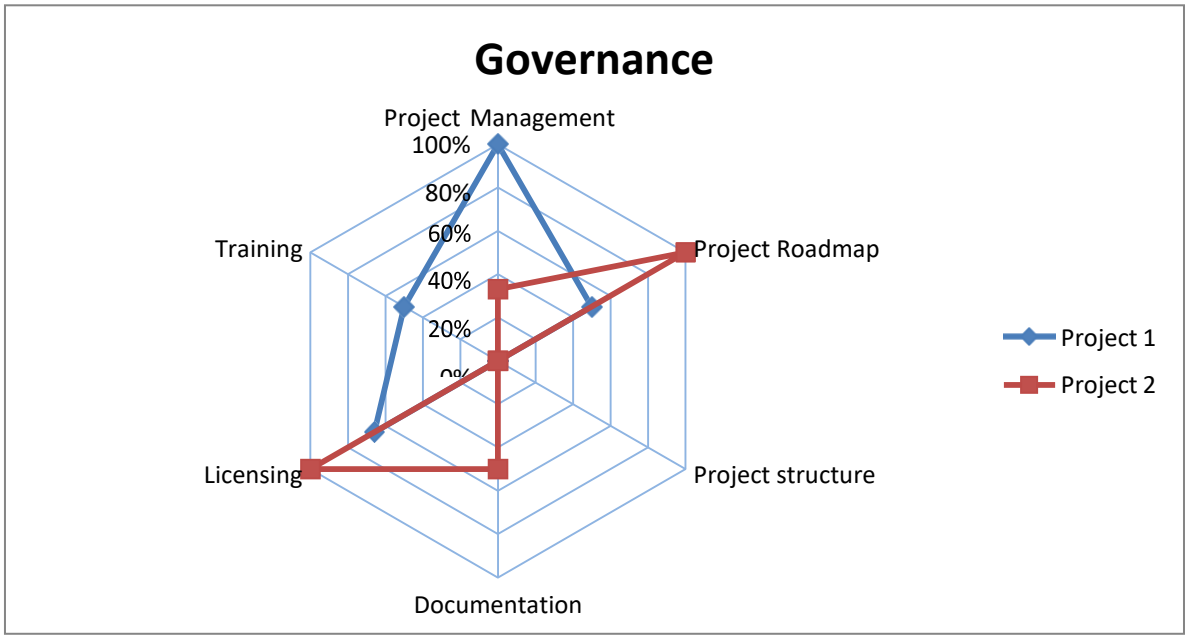


Figure 5. Demographics and Diversity

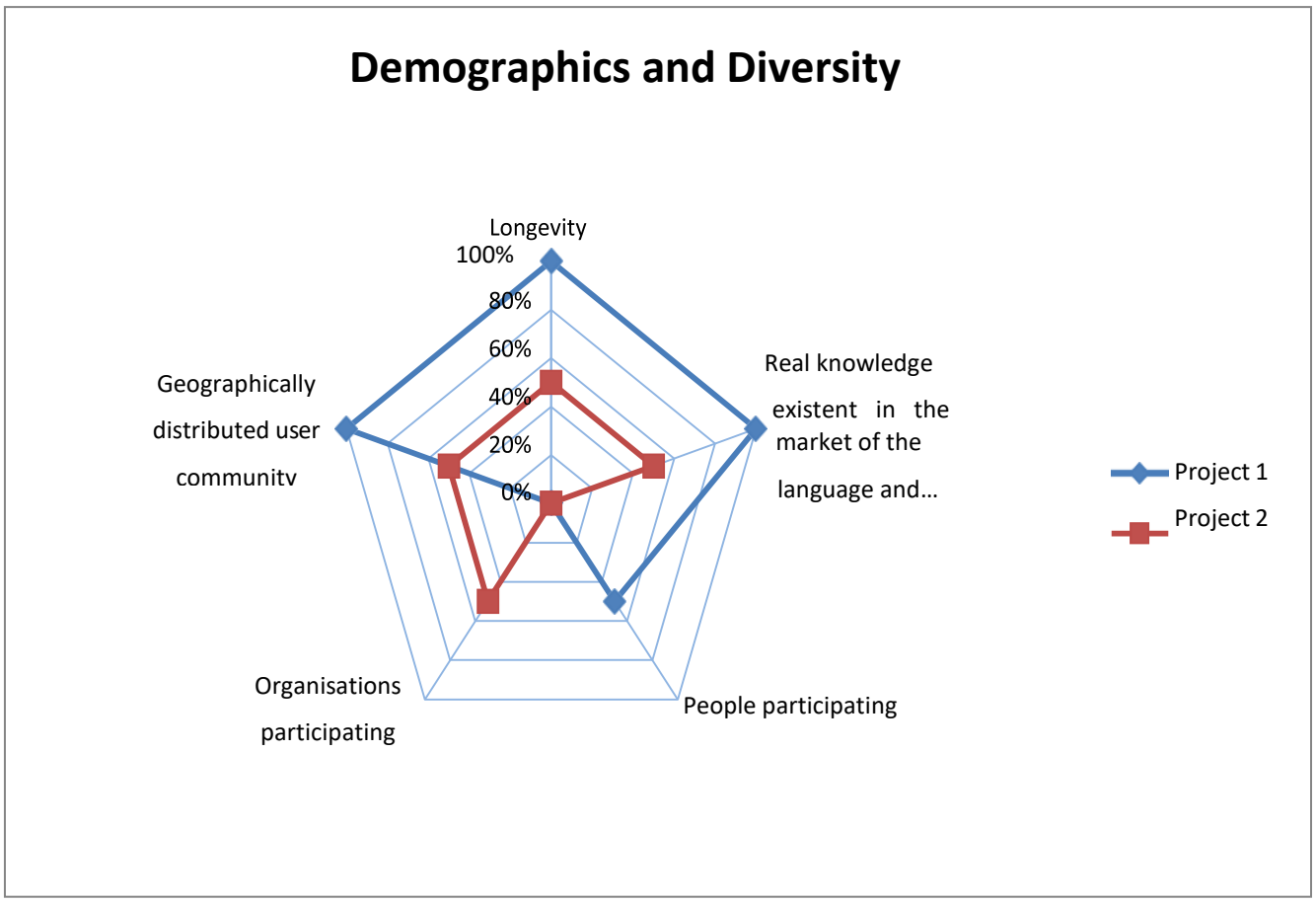


Figure 6. FOSS Support

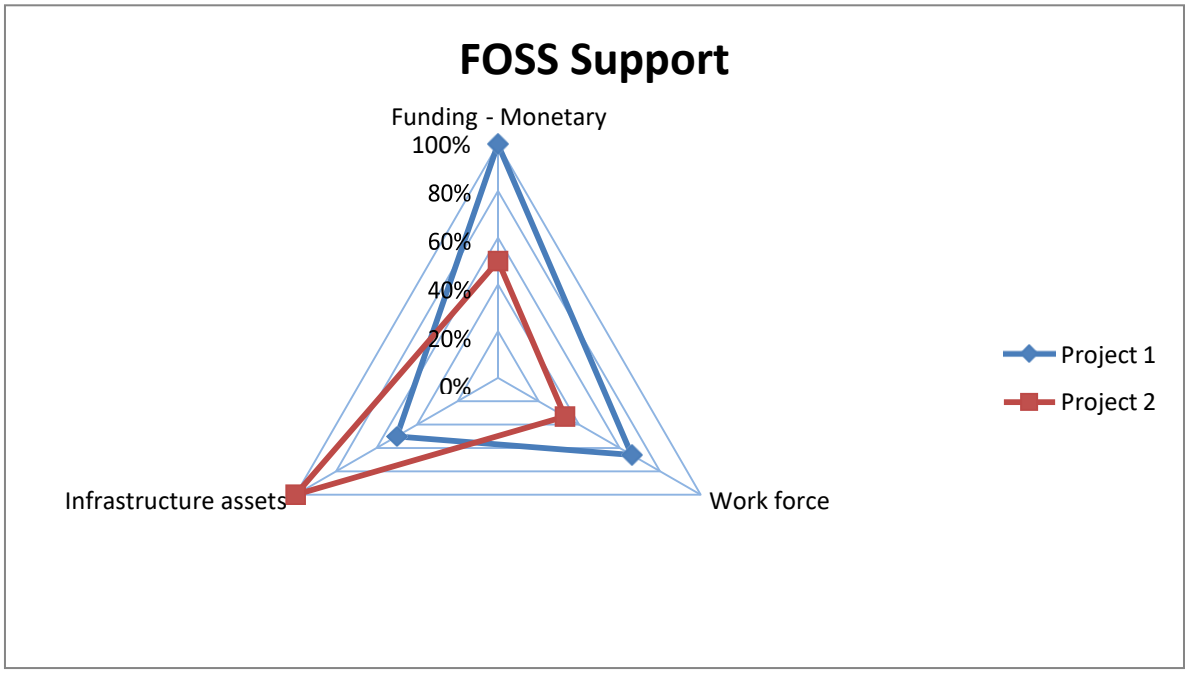


Figure 7. Comparison of Projects and Categories

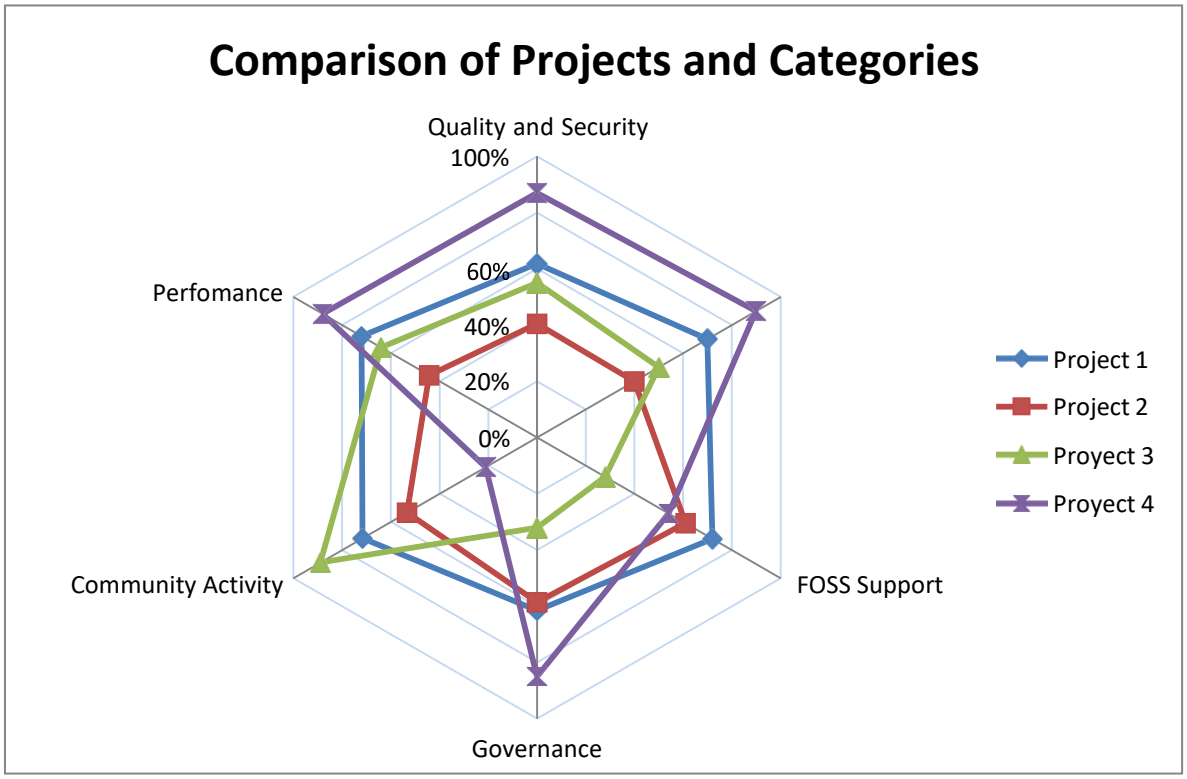
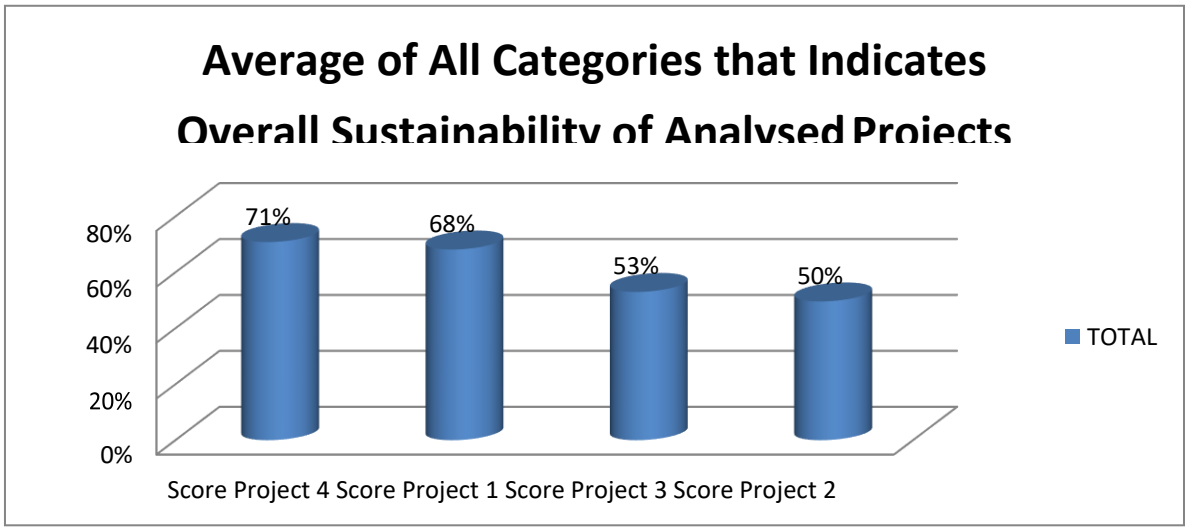


Figure 8. Average of All Categories that Indicates Overall Sustainability of Analysed Projects



ANNEX 3: DETAILED DESCRIPTION OF TARGET DATA MODEL

Entity			
Name	**Definition**		**Type**
AppSoftware	The entity describes application / app software.		Dependent
Criterion	The entity describes a quality criterium used to assess if related software belong to the Critical Software Shortlist.		Independent
CustomSoftware	The entity describes software defined ad-hoc.		Dependent
DataCenterResources	This layer groups all the possible open source software embedded inside physical devices such as routers, load balancers, SANs, switches, firewalls…		Dependent
Dependencies	The entity lists all the software on which a software depends on.		Dependent
DevelopmentPlatform	The entity describes a software development platform or tool.		Dependent
License	The entity describes a software license and its terms.		Independent
LicenseCompliance	The materialised relationship connects a software to the licenses it complies with.		Dependent
MobileDevice	The entity describes a portable device (smartphone, tablet, etc.).		Dependent
MobileSoftware	The entity describes software that has been developed for mobile devices.		Dependent
OperatingSystem	The entity describes an operating system.		Dependent
Organization			Independent
RuntimeSoftwarePlatform	The entity describes a web server, DBMS, application server or any kind of runtime/middleware;		Dependent
Server	The entity describes a computer used for hosting purposes.		Dependent
Software	The entity describes software.		Independent
SoftwareCriteria	The materialised relationship connects softwares with their related quality criteria.		Dependent
SoftwareInstance	The entity represents a deployed software, hence it relates with one or more hosts.		Dependent
SoftwareVersion	The entity describes the version of a Software.		Dependent
SoftwareVulnerabilities	The materialised relationship connects a software version with its related detected vulnerabilities.		Dependent
Standard	The entity describes a standard, whose characteristics are: openness, transparency and being based on consensus.		Independent
StandardCompliance	The materialised relationship connects a software to the standards it complies with.		Dependent
System	The entity represents a real machine or device on which software has been installed.		Independent
Vulnerability	The entity describes a vulnerability which was found on a specific version of a software.		Independent
Workstation	The entity describes a desktop or laptop device.		Dependent
Attribute(s) of "AppSoftware" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SoftwareName	The name that identifies the software.	Yes	Yes
Attribute(s) of "Criterion" Entity			
Name	**Definition**	**Is PK**	**Is FK**
CriterionName	The name that identifies the criterion.	Yes	No
Threshold	The specific criticality threshold for the criterion.	No	No
Weight	Measures the relevance of the criterion and influences how it is taken into account when assessing software criticality.	No	No
Attribute(s) of "CustomSoftware" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SoftwareName	The name that identifies the software.	Yes	Yes
Attribute(s) of "DataCenterResources" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SoftwareName	The name that identifies the software.	Yes	Yes

Attribute(s) of "Dependencies" Entity			
Name	**Definition**	**Is PK**	**Is FK**
DependsOnSoftwareName	The name that identifies a software on which the software under analysis depends on.	Yes	Yes
DependsOnVersionNumber	The name that identifies the version of a software on which the software under analysis depends on.	Yes	Yes
SoftwareName	The name that identifies the software.	Yes	Yes
VersionNumber	Reports the version the software is, or was.	Yes	Yes
Attribute(s) of "DevelopmentPlatform" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SoftwareName	The name that identifies the software.	Yes	Yes
Attribute(s) of "License" Entity			
Name	**Definition**	**Is PK**	**Is FK**
LicenseType	The specific type of the license, which refers to a specific standard.	Yes	No
LicenseContact	The name of the reference person for the license.	No	No
OrganizationName	The name that identifies the organisation that defined the license.	No	Yes
Attribute(s) of "LicenseCompliance" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SoftwareName	The name that identifies the software.	Yes	Yes
LicenseType	The specific type of the license, which refers to a specific standard.	Yes	Yes
Attribute(s) of "MobileDevice" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SystemName	The name that identifies the system.	Yes	Yes
Attribute(s) of "MobileSoftware" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SoftwareName	The name that identifies the software.	Yes	Yes
Attribute(s) of "OperatingSystem" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SoftwareName	The name that identifies the software.	Yes	Yes
Attribute(s) of "Organization" Entity			
Name	**Definition**	**Is PK**	**Is FK**
OrganizationName	The name that identifies the organisation.	Yes	No
Location	The physical location (i.e. place) the headquarters of the organisation is stationed.	No	No
Description	Further details on the organisation.	No	No
Attribute(s) of "RuntimeSoftwarePlatform" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SoftwareName	The name that identifies the software.	Yes	Yes
Attribute(s) of "Server" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SystemName	The name that identifies the system.	Yes	Yes
Attribute(s) of "Software" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SoftwareName	The name that identifies the software.	Yes	No
Description	Further details about the software.	No	No
IsCritical	Tells if the software belongs to the Software Critical Shortlist.	No	No
AOWName	The application owner name.	No	No
AOWPosition	The application owner position.	No	No
Developer	The development entity that designed the software.	No	Yes
SoftwareType	It defines the type of the System: application software, custom software, mobile software, runtime platform, operating system, development platform or data center resources.	No	No
Attribute(s) of "SoftwareCriteria" Entity			
Name	**Definition**	**Is PK**	**Is FK**

SoftwareName	The name that identifies the software.	Yes	Yes
CriterionName	The name that identifies the criterion.	Yes	Yes
Rating	The value of the criterion for the specific software.	No	No
Attribute(s) of "SoftwareInstance" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SoftwareName	The name that identifies the software.	Yes	Yes
SystemName	The name that identifies the system.	Yes	Yes
VersionNumber	Reports the version the software is, or was.	Yes	Yes
Size	The memory space (in MB) the instance needs.	No	No
Attribute(s) of "SoftwareVersion" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SoftwareName	The name that identifies the software.	Yes	Yes
VersionNumber	Reports the version the software is, or was.	Yes	No
Attribute(s) of "SoftwareVulnerabilities" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SoftwareName	The name that identifies the software.	Yes	Yes
VersionNumber	Reports the version the software is, or was.	Yes	Yes
VulnerabilityName	The name that identifies the vulnerability type.	Yes	Yes

Attribute(s) of "Standard" Entity			
Name	**Definition**	**Is PK**	**Is FK**
StandardisationBody	The organisation that defined the standard.	No	No
StandardName	The name that identifies the standard.	Yes	No
Description	Further details about the standard.	No	No
StandardisationBody	A reference to the Standard content.	No	Yes
ECContext	The European Community Context the standard is related to.	No	No
Documentation	The documentation that the standard have, in text format.	No	No
ParentStandardName	The name that identifies the standard which references or contains this standard.	No	Yes
Attribute(s) of "StandardCompliance" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SoftwareName	The name that identifies the software.	Yes	Yes
StandardName	The name that identifies the standard.	Yes	Yes
Attribute(s) of "System" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SystemName	The name that identifies the system.	Yes	No
Vendor	The Organization that produces the system.	No	Yes
Model	The specific model of the machine, comprehensive of producer and version.	No	No
RAM	It measures the Random Access Memory size of the machine.	No	No
IsVirtual	It tells if the machine is a Virtual Machine.	No	No
ManagingOrganization	The name that identifies the organisation that manages the system.	No	Yes
SystemType	It defines the type of the System: mobile device, server or workstation.	No	No
SoftwareName	The name that identifies the software.	No	Yes
Attribute(s) of "Vulnerability" Entity			
Name	**Definition**	**Is PK**	**Is FK**
VulnerabilityName	The name that identifies the vulnerability type.	Yes	No
Source	The affected software's source code.	No	No
Description	Further details on the vulnerability.	No	No
Impact	An indicator of the expected harm received if the vulnerability is actually exploited.	No	No
Remediation	The description of the required actions to resolve the vulnerability.	No	No
Attribute(s) of "Workstation" Entity			
Name	**Definition**	**Is PK**	**Is FK**
SystemName	The name that identifies the system.	Yes	Yes

APPENDIX: ABBREVIATIONS AND ACRONYMS

ABAC / ABAC Asset	Corporate Ordering and Asset management system
BYOD	Bring Your Own Device
DIGIT	Directorate-General for Informatics
CMDB	Configuration Management Data Base
CSV	Comma-Separated Values
EC	European Commission
ETL	Extract, Transform and Load
FOSSA	Free & Open Source Software Application
HR	Human Resources
IT	Information Technology
MDM	Mobile Device Management
NIST	National Institute of Standards and Technology
OS	Operating System
OSI	Open Source Initiative
OSS	Open Source Software
OSVDB	Open Source Vulnerability Database
PC	Personal Computer
RHEL	Red Hat Enterprise Linux
SAN	Storage Area Network
SCCM	System Centre Configuration Manager
SDL	Software Development Library
SLA	Service Level Agreement
Svn	Subversion