Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

ABDULLAH AL SAMMAN
MASTER THESIS

MODELING FLOSS DEPENDENCIES IN PRODUCTS

Submitted on 30 September 2020

Supervisors: Andreas Bauer, M. Sc., Prof. Dr. Dirk Riehle, M.B.A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander-Universität Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 30 September 2020

License

Erlangen, 30 September 2020

Abstract

As the software industry grows fast, the competition to produce high-quality software as quickly as possible, made the reusability of Free/Libre and Open Source Software (FLOSS) famous, but with features come challenges and risks. The main issue with FLOSS is license compliance among the system's different FLOSS components; therefore, license compliance tools arise, trying to avoid and solve the problem. This thesis investigates the license compliance tools to find how these products model the FLOSS dependencies. The need for a unified model is significant to reach a fully automated method to decide FLOSS components' usability without any issues.

Contents

1 Introduction

As the software industry grows, reusability is becoming crucial for vendors to produce high-quality software quickly. That made FLOSS very popular and used by all kinds of software vendors worldwide, but with such ease comes challenges. Managing FLOSS dependencies has many aspects companies need to handle before using FLOSS dependencies to prevent legal hazards like license noncompliance with other FLOSS dependencies. *"The main problem when working with multiple open-source licenses is that they are not necessarily compatible."* Among different challenges Luoto (2013) identified license incompatibilities as the main problem. Security risks raise the following questions. Does a dependency oppose a threat to software safety or other FLOSS dependencies? Does a component have vulnerabilities? Possible risks from patent infringement. The community behind the component, and does it have an up to date documentation? Does the component have a Software Bill of Material (SBOM) documents available in a machine-readable format like Software Package Data Exchange (SPDX)[1]? All the required information to manage these challenges must be documented next to the FLOSS component composition. A system architect usually doesn't consider this additional information to manage FLOSS dependencies. *"The architect's role as a key character in orchestrating different stakeholder requirements into a single system has been widely recognized"* (Rozanski and Woods, 2012). Still, the lake of legal experience or supervision from this key role could be a fatal blow to the software. All the mentioned challenges must be treated well in the early phase if the vendor wants to avoid legal problems. That's why companies using different tools to collect license compliance information then produce SBOM documents for future use. The wide usage of these tools as FLOSS products benefits all vendors and helps create communities around it. Each license compliance tool has its way of representing the FLOSS dependencies. Usually, license compliance tools offer more features than just listing all used packages and licenses. They provide license conflict analysis, and they use a machine-readable format for generating and sharing reports. All variation between license compliance tools should have something common to build models around it. The challenges with data gathering for the compliance process, usage of FLOSS in product, custom

[1][Accessed on 09.05.2020] https://spdx.dev/specifications/

reports, SPDX support, and a central system to manage products are found in the Bauer et al. (2020) case study. Suggest investigating if the different models behind license compliance tools can be based on a unified product architecture model. Such a unified model could simplify the use of FLOSS in commercial software. In this thesis, we want to investigate if the representation of license relevant information of different license compliance tools can be based on a common model representing a tool architecture. From this model, documents like the SBOM should be derived.

Clarification of **We** usage in this thesis context is to keep on the academic writing style, and all the work is done by Al Samman alone without any help.

2 Related Work

Hammouda et al. (2010) discussed that every architectural design must consider the common stakeholders' requirements and consider a new aspect of the legal concerns: using the FLOSS dependencies and components legal part of the vendor's offered software. They have proposed patterns used for managing the way different components can interact to ensure that all licenses of open source components comply with its license. They introduced several concept open source legality patterns under these three main categories: **a)** Interaction legality patterns. **b)** Isolation legality patterns. **c)** Licensing legality patterns. This paper shows how important to consider the legal aspect to ensure license compliance among used FLOSS dependencies and components in particular software.

Brøndum (2012) dissertation shows how vital it is to model and coordinate the dependency to satisfy the intended system's requirements. Then he illustrates the cases, which the architectural dependency analysis and modeling (ADAM) have been applied to discovering the hidden dependencies and connections to other remote or local services, which is not documented by the vendors. It then concludes with the importance of modeling and designing software systems affected by the dependence relationships with the primary system. By using ADAM and version control systems help refine and decompose the design elements.

Luoto (2013) main focus is the significance of open source license management on the architectural level. The developed tool uses UML profiles to create an open source license management framework. This work is proof that UML elements can be extended to represent license compliance information since it is a common approach to model software. Then scan the software for license conflict during the early phases of the development process or even for released software. But this approach does not cover dynamically linked libraries usage scenarios.

Sangal et al. (2005) focuses on managing large software systems' architecture by converting the system's statistical analysis data to a visual method known as the dependency structure method. It utilizes multiple algorithms to detect the design violations according to the user's design rules and apply it during the system development to decide which dependencies are acceptable. This work shows the importance of managing dependencies to keep the software in the top state to fulfill the customer needs, avoiding unnecessary and problematic dependencies by auditing every dependency relationship between the components, whether it is a FLOSS component.

3 Research Questions

In this section, our research questions are defined. In the introduction 1, all our motives for research on this subject explained. The overall goal is to analyze the well-established FLOSS license compliance and governance tools to see how they modeled FLOSS dependencies. To answers the following research questions:

RQ1: What models for modeling FLOSS dependencies in products exist?
Many vendors care to solve license compliance issues, using FLOSS license compliance tools to determine if they have any legal issues caused by license conflict within the software. Without even knowing how the license is represented, or the model used by the tool, running in the background. The companies or the community that developed license compliance tools are interested in improving it. Understanding the need for a unified model for license compliance tools leads us to question how the existing once model FLOSS dependencies to scan the intended software. A common FLOSS dependency model could help maintainers understand the rivels tool or decide which tool suits the vendor's needs.

RQ2: What dimensions are considered to model FLOSS dependencies?
Answering RQ1 suggests the significance of which dimensions matter in modeling FLOSS dependencies in license compliance tools. Each tool model will have design factors that lead to choosing a subset of all possible dimensions. Some tools prioritize scanning each file in the analyzed project to give accurate results for each package they scan. Other tools focus on packages they find in the analyzed projects; others look for contact information and copyright holders. The focus of license compliance tools, therefore its dimensions, shape the model of FLOSS dependencies.

RQ3: What are the core use cases for existing modeling solutions?
Use cases of each license compliance tools help reveal similarities and differences between them, allowing us to see the overall concept. The variety of our case study subjects will discover which models enable more functionality options in the tool, demonstrating the results of RQ1 and RQ2. Each case will be discussed individually, followed by a discussion in section 7.1 to share the findings and answer the research questions.

4 Research Methodology

This section describes in details the followed approach in this thesis, starting with data collection to the findings to produce a high-quality multiple case study. *"No matter what specific analytic strategy or techniques have been chosen, you must do everything to make sure that your analysis is of the highest quality"*. As Yin (2018) describes a high-quality case study analysis.

4.1 Multiple case study design

"A case study is an empirical method that investigates a contemporary phenomenon (the 'case') in depth and within its real-world context, especially when the boundaries between phenomenon and context may not be clearly evident." As Yin (2018) defines a case study, allowing the phenomenon to be studied in a real-world context with a flexible multi iteration process, not in a controlled environment. Furthermore, investigating FLOSS license compliance and governance tools makes conducting controlled experiments nearly impossible because they are used for many different use cases, not just one. This thesis fundamental inquiry questions how the existing FLOSS license compliance and governance tools models FLOSS dependencies, suggesting the significance of investigating a real-world context. Therefore, a case study is a proper way to answer research questions. Still, a single case study won't be enough to give a reliable answer due to a lack of heterogeneity. That's where a multiple case study comes in to provide a more generalized reliable answer. Comparing different case study subjects with each other gives an insight into how the chosen subjects' design model influences the supply chain tools and the FLOSS dependency management processes.

4.2 Case definition

"You need to define a specific, real-world 'case' to be the concrete manifestation of any abstraction." – Case specification by Yin (2018).
Our multiple case study subjects must fulfill six criteria to be considered a relevant case and a real-world case for our multiple case study research:

1. The case subject must be a FLOSS product; anyone can access the source code and the documentation without any commercial or proprietary restrictions. As we investigate the product, it has to be on GitHub or any other public version control system (VCS) with recent commits, pushes, and continuous integration (CI) feature enabled. With this, the product will be considered as an active one and a valid case.
2. Scanning projects or repositories for dependencies licenses or copyrights information using SPDX identifiers or license texts.
3. Product maturity is a requirement with real use cases, not just a written license crawler, although the product's age plays no role at all.
4. The ability to generate reports in machine-readable formats is required to share it with other tools or instances of it, such as SPDX documents from FOSSology to SW360.
5. The named product must have a model in its source code or specification to describe the model, e.g. XML Schema, supporting the use case of multiple license compliance.
6. Proper up to date documentation about the product specifications and working mechanisms.

We want to demonstrate choosing such criteria to present a proper multiple case study research with a common theoretical basis to cover as much information to answer our research questions.
In summary, all cases must be a well-established products with active development and the ability to scan for license compliance information, building tools supply chain using machine-readable formats aiming to automate the process of FLOSS usage in production.

4.3 Case selection

This section determines the selected candidate for our multiple case studies see table 4.2. As mentioned in the case definition 4.2, the six criteria are specified, and a lot of products can fit these criteria, see the list of candidates A. The selection technique used to finalize our selection is the Most Similar cases by Seawright and Gerring (2008), which have control variables in our case are the six criteria, see table 4.1. Besides, focusing on well-established products leads us to select mature and successful ones. It starts with FOSSology, one of the earliest and prime products in the license scanning business. ORT as a modern tool with the automation process of license compliance in its architecture. CycloneDX, with a model specification in XML schema format, which is also a part of its supply chain tools. It has different implementations for some ecosystems, e.g., Java, Python, and NPM. Tern as a license compliance tool for containers, e.g. docker. ScanCode Toolkit as a license scanner tool used as a third-party tool by ORT, Quartermaster and Tern. Finally, the Quartermaster as license compliance tool works during build time, unlike other tools integrate into the analyzed project building process.

Product	is FLOSS	License Scan	Mature	Machine-readable formats	has Model	Documentation
FOSSology	Yes	Yes	Yes	Yes	Yes	Yes
ORT	Yes	Yes	Yes	Yes	Yes	Yes
CycloneDX	Yes	Yes	Yes	Yes	Yes	Yes
Tern	Yes	Yes	Yes	Yes	Yes	Yes
Quartermaster	Yes	Yes	Yes	Yes	Yes	Yes
ScanCode Toolkit	Yes	Yes	Yes	Yes	Yes	Yes

Table 4.1: Selected candidates control variables.

This variation between existing products provides a broad overview and generalized answer to our research questions regarding the dependency modeling in products.

Product	Use Cases
FOSSology	LS, LC, CR, CLI, MA, WG.
OSS Review Toolkit (ORT)	LS, LC, CR, CLI, MA, E, DLDC.
CycloneDX	LS, LC, CR, CLI, VC, S, E.
Tern	LS, LC, CR, CLI, MA, VC, CS, E.
Quartermaster	LS, LC, CR, CLI, DLDC, BPC.
ScanCode Toolkit	LS, LC, CR, CLI, MA.

Legend: LS: License Scanner, **LC:** License Compliance, **CS:** Container Scanner, **S:** Specifications, **VC**: Vulnerabilities Check, **CR:** Copyrights, **E:** Extensions, **WG:** Web GUI, **CLI:** Command Line Interface, **MA:** Multiple Analyzers, **DLDC:** Dynamically Linked Dependency Check, **BPC:** Building Phase Scanner.

Table 4.2: Selected candidates for multiple case study.

4.4 Data sources

As research questions in chapter 3 suggest investigating FLOSS license compliance and governance tools focusing on how the FLOSS dependencies have been modeled. All case selection criteria have been specified in section 4.2 prepare for the data collection process starts with defining the data sources and where the data comes. According to Yin (2018), a case study should have at least one "source of evidence"; the more is better. He also identified the primary six sources of evidence: documentation, archival records, interviews, direct observations, participant observation, and physical artifacts. Because this thesis was written during the COVID-19 pandemic[1], the following sources of evidence were

[1] [Accessed on 28.05.2020] https://en.wikipedia.org/wiki/COVID-19_pandemic

eliminated. **Direct observations, interviews, and participant observation.**
As defined by Yin (2018), **archival records** take the form of data files and records, e.g. public use files, service records, organizational records, maps and charts, and surveys. The previously described source of evidence doesn't apply in this research context and hard to get such records if it exists. The remaining two sources of evidence supply the necessary data for this research.
Documentation: is documentary information whether paper or electronic form e.g. emails, adminstrative documents, notes, progress reports, announcements, proposals, internal reports, etc. As Yin (2018) defined, this research focuses on well-established products, looking into details, not just the popularity of the product or its place in the FLOSS license compliance and governance tools or its use cases or performance but also the documentation of different kinds:

- Scientific papers.
- Specifications.
- Github wikis.
- Mailing lists.
- Developers documentation.

this list of documentation sources is our main source of evidence for most of this research.
Physical artifacts: exemplary defined by Yin (2018), *"a technological device, a tool or instrument, a work of art, or some other physical evidence."* The other physical evidence category applies to the subjects' source code and the embedded logical models in the referenced source codes. As for citing the source codes, we mention the source file refers to it in the code line number if required.
Discussing the difficulty of unifying the process for all sources of evidence. The documentation holds a different kind of data source in different shapes, writing styles, and notations. It makes the possibility of using the automation process nearly impossible. Driving us to do everything manually and leaving it to the researcher ways of thinking to match and compare the findings. Considering the previous facts about this research data sources, the need to standardize the process for all evidence sources is in play:

1. Look for the following source of documentation in order:
   (a) Papers and publications.
   (b) Specification.
   (c) Mailing lists.
   (d) Developers documentation.

   (e) Github wikis.
   (f) Source code as a physical artifact.

2. Consider the usefulness of each source of documentation.
3. Repeat for each case study subject.

The significance of this process affects the ability to distinguish sources of evidence and reliability of findings.
Lastly, the online sources of evidence have no restriction of any kind, including websites, wikis, and documentation. It was taking into consideration must be maintained officially by the selected product, not by any other third party organization or maintainer. If any evidence sources are out of date, it causes immediate elimination of the product as a case study subject, not just the source of evidence; it's also stated in the case definition 4.2 since it's one of the six criteria to have proper up to date documentation.

4.5 Analysis method

As data sources for this thesis are specified in section 4.4, our thesis findings are all done manually without any assistance from an automation tool. We extract the existing product model, which can take multiple forms or shaps because each product has its implementation and interpretation of its model. Since we developed our strategy to analyze the finding by building UML diagrams, which illustrate the model itself and the significant points based on the case study subject model. Then defining questions to be answered, as Yin (2018) suggested. each case must answer the following questions:

1. What is a component?
2. How are licenses and copyrights represented?
3. How are dependencies represented?
4. How is metadata represented?
5. How are vulnerabilities represented?
6. What are the common use cases?

Using this question provides a unified way to describe all the case studies to distinguish differences and similarities among them. Finally, *"Within any general strategy, including one you might develop yourself, you should consider using any of five analytic techniques."*(Yin, 2018). The techniques are:

1. Pattern matching.

2. Explanation building.
3. Time-series analysis.
4. Logic models.
5. Cross-case synthesis.

Those techniques are derived from pattern matching, with different conditions and aspects of the case study. As for this thesis, we chose cross-case analysis, which is easier to combine with our strategy. This strategy will help this multiple case study research in this thesis to find its way to answer our research questions with generalized and reliable results. Qualitative Data Analysis (QDA) was used to analyze the gathered data for the cross-case analysis method. Linking RQs and research results within the theory are crucial, as Mayring (2014) explains the QDA steps. Since the QDA thematic analysis by Braun and Clarke (2006) is highly similar to what we want to achieve, except for the first two steps are adjusted to fit our need analyzing the gathered data, see table 9.2. Firstly, transcription is changed to collect data based on case study subject source code and documentation. Secondly, the coding process shifts to build UML diagrams based on step one for each subject. Finally, all other steps stay unchanged.

5 Cases

5.1 Case 1: CycloneDX

"CycloneDX is a lightweight software bill-of-material (SBOM) specification designed for use in application security contexts and supply chain component analysis"[1]. It is a FLOSS product licensed under the Apache-2.0 license. This product provides a machine-readable format and a way to share the software bill of material(SBOM) with further features in mind vulnerabilities, dependency graphs, and BOM descriptor. Considering the young age of the product (started 2017), it is well-established and widely used by other supply chain tools. Some use it in the supply chain as a source of information, e.g. *"OWASP Dependency-Track is an intelligent Software Supply Chain Component Analysis platform that helps identify and reduce the risk of using FLOSS dependencies."*[2] Other tools used as a reporting option, e.g. OSS Review Toolkit, see case 5.3. It's well-maintained with continuous development by a single maintainer. It also uses SPDX licenses list and scans for licenses with SPDX identifiers. The CycloneDX SBOM specification is a vendor-agnostic and language-independent solution to describe a product's composition. They aim for a machine-readable and easy to implement and parse representation as JSON or XML file. Apart from other goals, the specification should be extensible to support specific and future use cases and support software components and hardware, frameworks, containers, and operating systems. Besides a common goal of having a complete and accurate representation of a software and its components, it also covers the application security context. Interesting in this product is that they see components rather than a simple list and want to describe the component relationships in a more detailed fashion. Another aspect is the ability to describe the dependency on another service, which is not often covered by FLOSS license compliance and governance tools. These aspects of this product make it an excellent case study subject to understand how detailed component dependencies can be represented. Additional use cases can be covered by using CycloneDX extensions, for

[1][Accessed on 19.05.2020] https://cyclonedx.org
[2][Accessed on 19.05.2020] https://dependencytrack.org

example, the dependency graph functionality and the vulnerability description are provided as extra extensions.

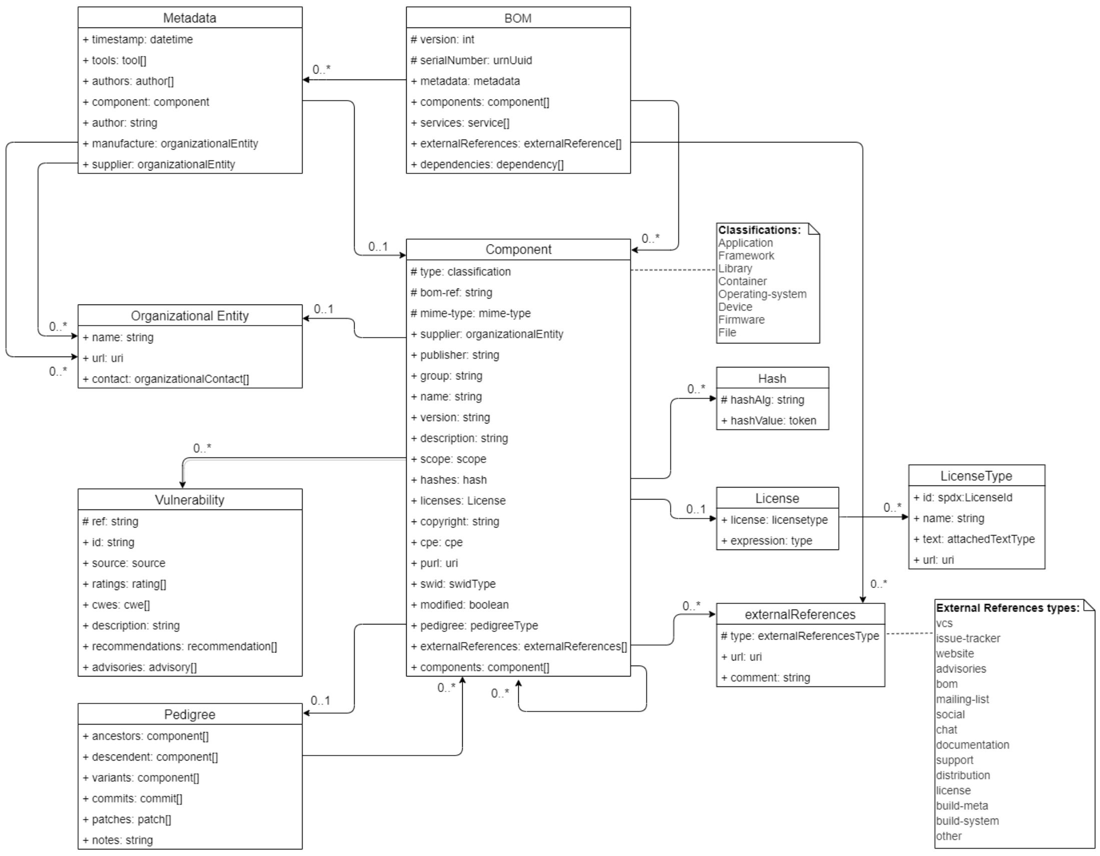


Figure 5.1: CycloneDX UML model.

5.1.1 What is a component?

As the documentation of bill of materials (BOM) schema v1.2 implies, each FLOSS dependency is represented as a component with the necessary information, e.g. name, publisher, copyrights, etc. The predefined type doesn't describe software components only, but hardware devices or software containers are mentioned as a note in the figure 5.1. Additionally, a mime-type can be defined to describe the kind of a component. Apart from general identification properties such as name, group, and version, a package uniform resource locator (PURL)[3] can be used to accurately identify a software component. With other identification options through CPE[4] is a structured naming scheme for information technology systems. SWID is defined as an ISO standard under ISO/IEC 19770-2:2015[5],

[3][Accessed on 06.06.2020] http://www.purlz.org
[4][Accessed on 06.06.2020] https://nvd.nist.gov/products/cpe
[5][Accessed on 06.06.2020] https://www.iso.org/standard/65666.html

establishes specifications for tagging software to optimize its identification and management. Hashes for each component also allows checking if the component has changed. Authorship of a component can be presented as a simple author and publisher string, or as a supplier, a more sophisticated organizational entity with detailed contact information, see figure 5.1.

```
 <...
 type="bom:classification" [1]
 mime-type="bom:mimeType" [0..1]
 bom-ref="xs:string" [0..1]
 Allow any attributes from any namespace (lax validation).
>
   <bom:supplier> bom:organizationalEntity </bom:supplier> [0..1]
   <bom:author> xs:normalizedString </bom:author> [0..1]
   <bom:publisher> xs:normalizedString </bom:publisher> [0..1]
   <bom:group> xs:normalizedString </bom:group> [0..1]
   <bom:name> xs:normalizedString </bom:name> [1]
   <bom:version> xs:normalizedString </bom:version> [1]
   <bom:description> xs:normalizedString </bom:description> [0..1]
   <bom:scope> bom:scope </bom:scope> [0..1]
   <bom:hashes > [0..1]
      Start Sequence [0..*]
         <bom:hash> bom:hashType </bom:hash> [1]
      End Sequence
   </bom:hashes>
   <bom:licenses > [0..1]
      Start Choice [1]
         <bom:license> bom:licenseType </bom:license> [0..*]
         <bom:expression> xs:normalizedString </bom:expression> [0..1]
      End Choice
   </bom:licenses>
   <bom:copyright> xs:normalizedString </bom:copyright> [0..1]
   <bom:cpe> bom:cpe </bom:cpe> [0..1]
   <bom:purl> xs:anyURI </bom:purl> [0..1]
   <bom:swid> bom:swidType </bom:swid> [0..1]
   <bom:modified> xs:boolean </bom:modified> [0..1]
   <bom:pedigree> bom:pedigreeType </bom:pedigree> [0..1]
   <bom:externalReferences> bom:externalReferences
   </bom:externalReferences> [0..1]
   <bom:components > [0..1]
      Start Sequence [0..*]
         <bom:component> bom:component </bom:component> [1]
         Allow any elements from a namespace other than this schema's
         namespace (lax validation). [0..*]
      End Sequence
   </bom:components>
   Allow any elements from a namespace other than this schema's
   namespace (lax validation). [0..*]
</...>
```

Listing 5.1: CycloneDX XML reference v1.2 component.

5.1.2 How are licenses and copyrights represented?

A component can reference licenses. The definition of a license consists of an SPDX expression string, which describes the multiple licenses relationship to each other compliance information and one or more structures of license type entities, see lines 20 to 25 in listing 5.1. An example of an expression is “Apache-2.0 AND (MIT OR GPL-2.0-only)”, which can be represented as a structured license type object with an SPDX identifier, a name, license text, and a URL to the license definition, see listing 5.2 and figure 5.1. This definition also describes the option to document the license if it does not exist in the SPDX license list by license name and text. It's vital to notice the field id only accepts SPDX valid ids, not an SPDX expression. In contrast, copyright information is specified as a simple string.

```
 <...>
Start Choice [1]
   <bom:id> spdx:licenseId </bom:id> [0..1]
   <bom:name> xs:normalizedString </bom:name> [0..1]
End Choice
<bom:text> bom:attachedTextType </bom:text> [0..1]
<bom:url> xs:anyURI </bom:url> [0..1]
Allow any elements from a namespace other than this schema's namespace
(lax validation). [0..*]
</...>
```

Listing 5.2: CycloneDX XML reference v1.2 licenseType.

5.1.3 How are dependencies represented?

It is possible to document dependencies to other components in different ways. The simplest way is to reference subcomponents in the component's property of a component with full BOM. Another way is to use the pedigree property, which allows us to document complex supply chain scenarios from the beginning to the end, including the creation, distribution, modification, and redistribution of components. It is possible to describe how a component deviates from an ancestor, descendant, or variant with the commits property. The last way is to use the dependency graph extension. The graph representation of the component dependencies is defined besides the components using references to the components.

```xml
<bom serialNumber="urn:uuid:3e671687-395b-41f5-a30f-a58921a69b79"
version="1"
 xmlns="http://cyclonedx.org/schema/bom/1.1"
 xmlns:dg="http://cyclonedx.org/schema/ext/dependency-graph/1.0">
<components>
    <component type="framework" bom-ref="pkg:maven/org.example.acme/
    web-framework@1.0.0">
        ...
    </component>
    <component type="library" bom-ref="pkg:maven/org.example.acme/
    persistence@3.1.0">
        ...
    </component>
    <component type="library" bom-ref="pkg:maven/org.example.acme/
    common-util@3.0.0">
       ...
    </component>
</components>
<dg:dependencies>
    <dg:dependency ref="pkg:maven/org.example.acme/web-framework@1.0.0">
        <dg:dependency ref="pkg:maven/org.example.acme/common-util@3.0.0"/>
    </dg:dependency>
    <dg:dependency ref="pkg:maven/org.example.acme/persistence@3.1.0">
        <dg:dependency ref="pkg:maven/org.example.acme/common-util@3.0.0"/>
    </dg:dependency>
    <dg:dependency ref="pkg:maven/org.example.acme/common-util@3.0.0"/>
</dg:dependencies>
</bom>
```

Listing 5.3: Dependency graph example.

For arbitrary external dependencies, which are not part of the BOM, the externalReference property exists. *"External references provide a way to document systems, sites, and information that may be relevant but which are not included with the BOM."*Springett (2020b) as the documentation describes. There are 15 predefined types of external references, which are listed as a note in figure 5.1. If the first 14 types do not fit, you can go to the last option **"other"** and describe it as you need, aside with URI and text, see listing 5.4. Meaning any useful information helps with dependencies documentation not included in the BOM structure, it is welcomed to be added here, one of CycloneDX's most strong suites. Dependencies on other services can also be documented in the top-level BOM element in a detailed form. Here it is possible to describe endpoints of a provider, including their URLs, with information about the license of service, a classification of the provided data, and a flag for a required authentication, see listing 5.5. Documenting how a component is linked to another from a technical perspective (static linking vs. dynamic linking) is not possible yet.

```
 <...
 type="bom:externalReferenceType" [1]
 Allow any attributes from any namespace (lax validation).
>
  <bom:url> xs:anyURI </bom:url> [1]
  <bom:comment> xs:string </bom:comment> [0..1]
</...>
```

Listing 5.4: CycloneDX XML reference v1.2 externalReference.

```
 <...
 bom-ref="xs:string" [0..1]
 Allow any attributes from any namespace (lax validation).
>
  <bom:provider> bom:organizationalEntity </bom:provider> [0..1]
  <bom:group> xs:normalizedString </bom:group> [0..1]
  <bom:name> xs:normalizedString </bom:name> [1]
  <bom:version> xs:normalizedString </bom:version> [0..1]
  <bom:description> xs:normalizedString </bom:description> [0..1]
  <bom:endpoints > [0..1]
    Start Sequence [0..*]
      <bom:endpoint> xs:anyURI </bom:endpoint> [1]
    End Sequence
  </bom:endpoints>
  <bom:authenticated> xs:boolean </bom:authenticated> [0..1]
  <bom:x-trust-boundary> xs:boolean </bom:x-trust-boundary> [0..1]
  <bom:data > [0..1]
    Start Sequence [0..*]
      <bom:classification> bom:dataClassificationType
      </bom:classification> [1]
    End Sequence
  </bom:data>
  <bom:licenses > [0..1]
    Start Choice [1]
      <bom:license> bom:licenseType </bom:license> [0..*]
      <bom:expression> xs:normalizedString </bom:expression> [0..1]
    End Choice
  </bom:licenses>
  <bom:externalReferences> bom:externalReferences
  </bom:externalReferences> [0..1]
  <bom:services > [0..1]
    Start Sequence [0..*]
      <bom:service> bom:service </bom:service> [1]
      Allow any elements from a namespace other than this schema's
      namespace (lax validation). [0..*]
    End Sequence
  </bom:services>
  Allow any elements from a namespace other than this schema's
  namespace (lax validation). [0..*]
</...>
```

Listing 5.5: CycloneDX XML reference v1.2 service.

5.1.4 How is meta-data represented?

Meta-data can be documented in two different ways. The first is the meta-data property of the top-level BOM element mostly describes authorship information of the whole BOM. Furthermore, it allows documenting tools that were involved in the creation of the BOM document, see figure 5.1. The second, BOM descriptor extension, adds more metadata options about manufacturers and suppliers to document much more reliable detailed information about the components in case of third party existence. This extension works only for BOM v1.1 as for v1.2, its incorporated. In general, extension possibilities should make it easy to define custom meta-data that need to be part of the SBOM.

5.1.5 How are vulnerabilities represented?

The CycloneDX describes itself as an SBOM specification with a security context in mind. The vulnerability extension provides the functionality to document security vulnerabilities. The vulnerability property of a component can represent an identifier common vulnerabilities and exposures (CVE)[6] and a URL to the source, as well as scores, ratings, and recommendations to assist with the decision-making process. This extension schema developed in cooperation with Sonatype[7], which is a tool to automate FLOSS governance. However, this information can be bundled in the BOM structure for each component or bundled at the end of the BOM as other extensions do; both are valid for all other use cases. This extension relies on bom-ref to associate the component with the vulnerability element, recommending PURL usage as bom-ref value to ensure unique reference, see the example listing 5.6.

5.1.6 What are the common use cases?

This product already has a list of possible use case scenarios Springett (2020a), demonstrating the power of CycloneDX:

1. **License compliance:** Document all license compliance information for FLOSS dependency, using SPDX licenses list, identifiers, and expressions.
2. **Package evaluation:** PURL helps standardize the package metadata so it can be located regardless of the source.
3. **Integrity verification:** List all needed hashes for cryptography use.
4. **Known vulnerabilities:** Identifying known vulnerabilities in components can be achieved by using three fields: CPE, SWID, and PURL. Not all fields apply to all types of components, and not all sources of vulnerability intelligence support all three fields. The use of multiple sources may be required to obtain accurate and actionable results.
5. **Inventory:** Provide a complete list of all first party and third party dependencies.
6. **Authenticity:** Digital signatures may be applied to BOM for security purposes.
7. **Assembly:** BOM elements can be nested in each other to represent a tree of dependencies.
8. **Dependency graph:** Provide a graph data representation for each dependency and how depend on other dependencies.

[6][Accessed on 06.06.2020] https://cve.mitre.org
[7][Accessed on 28.05.2020] https://www.sonatype.com

9. **Provenance:** CycloneDX is capable of representing component authorship and the suppliers from which components were obtained. Textual fields representing the author(s) and publisher(s) can be used, as well as SWID metadata or complete inline SWID documents.

10. **Pedigree:** Represent detailed information about the ancestors, descendants, and variants.

11. **Service definition:** Description of depending services available as web service and otherwise not visible as a dependency of a component.

12. **Packaging and distribution:** For software produced for the consumption of others, it is essential to apply additional metadata about the provided software, including detailed component information, manufacturer and supplier information, and the tools used to create the BOM.

13. **Exploitability:** The vulnerability and exploitability (VEX) use cases are also possible through the use of the optional vulnerability schema extension.

```xml
<?xml version="1.0"?>
<bom serialNumber="urn:uuid:3e671687-395b-41f5-a30f-a58921a69b79"
version="1" xmlns="http://cyclonedx.org/schema/bom/1.1"
     xmlns:v="http://cyclonedx.org/schema/ext/vulnerability/1.0">
  <components>
    <component type="library"
    bom-ref="pkg:maven/com.fasterxml.jackson.core/jackson-databind
    @2.9.9">
      ...
      <purl>pkg:maven/com.fasterxml.jackson.core/jackson-databind@2.9.9
      </purl>
      <v:vulnerabilities>
        <v:vulnerability ref="pkg:maven/com.fasterxml.jackson.
        core/jackson-databind@2.9.9">
          <v:id>CVE-2018-7489</v:id>
          <v:source name="NVD">
            <v:url>https://nvd.nist.gov/vuln/detail/CVE-2018-7489
            </v:url>
          </v:source>
          <v:ratings>
            <v:rating>
              <v:score>
                <v:base>9.8</v:base>
                <v:impact>5.9</v:impact>
                <v:exploitability>3.0</v:exploitability>
              </v:score>
              <v:severity>Critical</v:severity>
              <v:method>CVSSv3</v:method>
              <v:vector>AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H</v:vector>
            </v:rating>
          </v:ratings>
          <v:cwes>
            <v:cwe>184</v:cwe>
            <v:cwe>502</v:cwe>
          </v:cwes>
          <v:description>FasterXML jackson-databind
            before 2.7.9.3, 2.8.x
            before 2.8.11.1 and 2.9.x before 2.9.5 allows
            unauthenticated remote code execution
            ...
           </v:description>
          <v:recommendations>
            <v:recommendation>Upgrade</v:recommendation>
          </v:recommendations>
          <v:advisories>
            <v:advisory>https://github.com/FasterXML/jackson-databind/
            issues/1931</v:advisory>
            <v:advisory>http://www.securityfocus.com/bid/103203
            ...
          </v:advisories>
        </v:vulnerability>
      </v:vulnerabilities>
    </component>
  </components>
</bom>
```

Listing 5.6: Vulnerability information included in BOM.

5.2 Case 2: Fossology

"FOSSology is an open-source license compliance software system and toolkit"[8]. It is one of the top well-established products in the license scanning business, started as a product inside the Hewlett-Packard[9]. The product was published as FLOSS in December 2007, licensed under the GPL-v2.0 license. Later on, it was transferred to the linux foundation[10] and hosted by it as one of open compliance program[11]; also, the product is a member of automation compliance tooling[12]. *"FOSSology implemented an architecture of pluggable agents that can be composed into a pipeline of tasks applied to an uploaded open source component."* As Jaeger et al. (2017) mentioned that the used architecture in FOSSology enables new features to be implemented as separate models, e.g. SPDX2 report agent[13]. This product offers different ways of scanning software. FOSSology uses three agents Nomos, Monk, and Ninka, with different approaches and algorithms to find any license information embedded in all the analyzed project files. Also, it allows automated decisions based on the agents joint effort if all agents agree on the same license decision. The goal was not just to scan and resolve license compliance issues only but also to reuse previous scan information by storing it in a database for a future scan to enhance scan times and performance. Generating reports in a machine-readable format has it's significant to all supply chain tools for different use cases. FOSSology offers to make reports not only SPDX format but also in Debian copyright format. In version 3.1 of the product, an exciting feature was developed, importing SPDX documents to share reports about FLOSS dependencies with other FOSSology instances or even other supply chain tools. It also runs via the command line in the repository directory, or you can upload the compressed software directly to the FOSSology web application and process the findings. Such flexibility and features made the product famous and used by many prominent organizations and universities around the globe, e.g. Siemens, Toshiba, and the University of Nebraska.

[8][Accessed on 20.05.2020] https://www.fossology.org/about/project-governance/#overview
[9][Accessed on 20.05.2020] https://en.wikipedia.org/wiki/Hewlett-Packard
[10][Accessed on 02.06.2020] https://www.linuxfoundation.org/
[11][Accessed on 07.06.2020] https://compliance.linuxfoundation.org/
[12][Accessed on 07.06.2020] https://automatecompliance.org/
[13][Accessed on 02.06.2020] https://fossology.github.io/agentlist.html

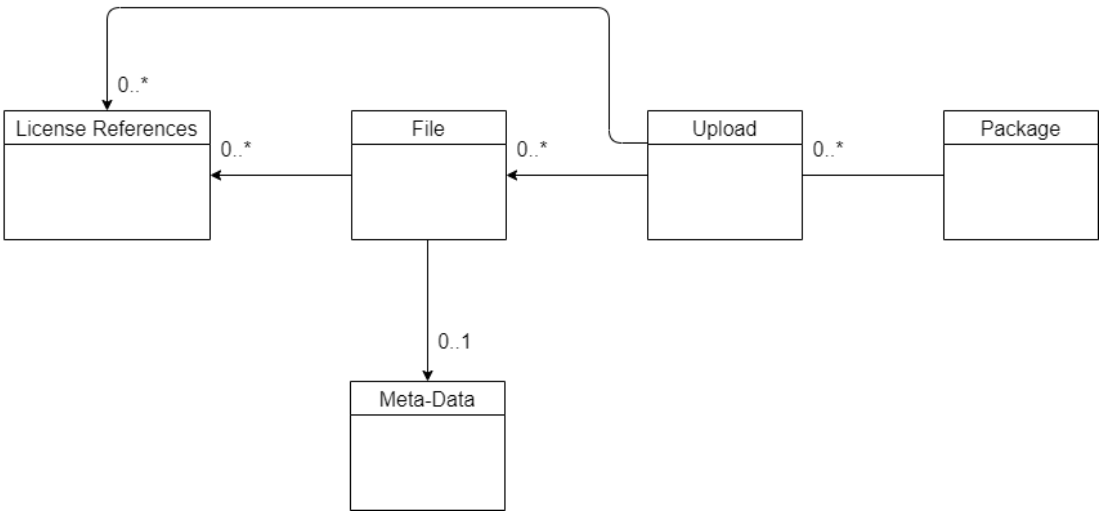


Figure 5.2: Fossology generlized model.

5.2.1 What is an Upload?

FOSSology is a web server, to use it the user must upload the target software to it, that's why each software is referred to as an upload in FOSSology GUI and database. The individual upload/component has a different representation, not like the other cases. Based on the entity relationship diagram (ERD)[14], see figure 5.3. The upload/component is divided into three database tables **a)** upload, **b)** uploadtree, and **c)** pfile. The upload table contains the primary data on the uploaded software like the file name and the way the target software was uploaded, e.g. GitHub, local file, or form file server. As for pfile table contains all files hashes to distinguish each file uniquely. The uploadtree table includes the relation between all the files with its uploaded target software; therefore, the relationship is described precisely between file and the parent, no matter what it is. All three tables enable the analyzed project to be described pretty accurately as an individual component allowing it to take shape, see figure 5.2.

[14][Accessed on 09.06.2020] https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/

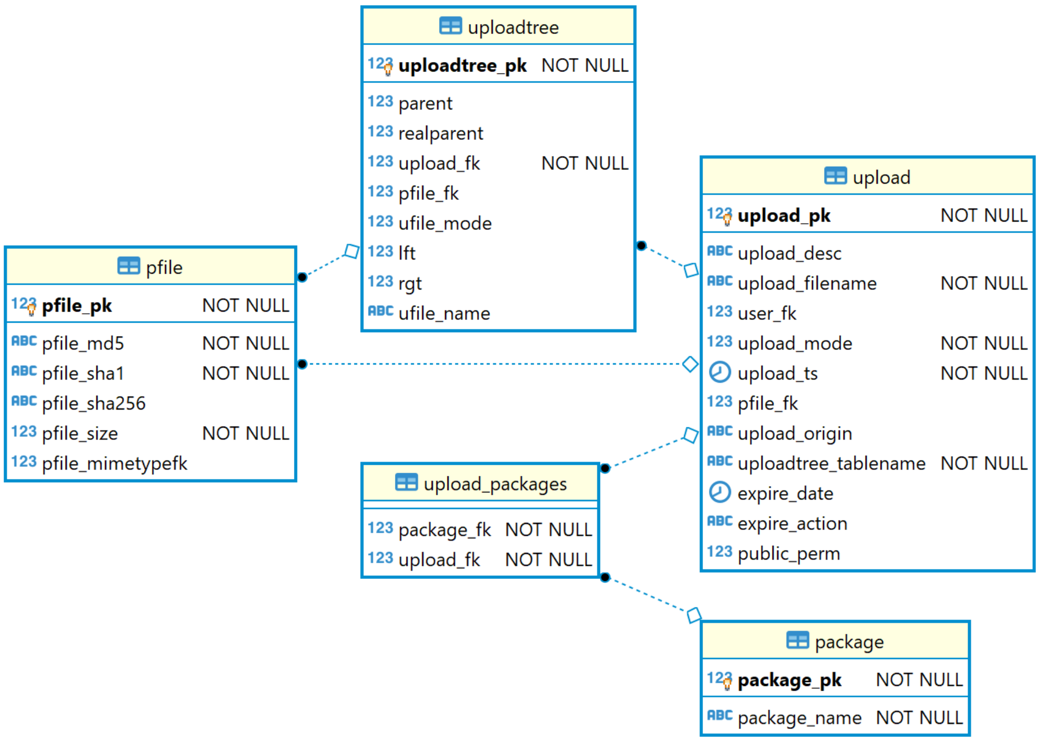


Figure 5.3: FOSSology upload ERD.

5.2.2 How are licenses and copyrights represented?

FOSSology scans each file of the uploaded software for license match, meaning each file has a list of possible licenses. Base on the ERD, see figure 5.4. The table pfile contains file information and hashes to ensure unique identification of each file. The license_file table links both the file with its license combined with vital information about the scanner, agent, or user how decided the license time stamped. The table license_ref contains all known licenses with full details, name of the license with its full text; It also defines license compatibility with GPL-v2 and GPL-v3, free software foundation (FSF)[15] status, and open source initiative (OSI)[16] approval of the license as an open source. This way, the analyzed project license and all included dependencies inside scanned for license compliance, if an issue is identified, e.g. GPL license incompatibility FOSSology give warnings about it. The license is represented as a database entity related to each software file and the upload. The upload relation to license_ref represents the whole analyzed project license or licenses. Copyrights are scanned for each file, just like licenses, see figure 5.5.

[15] [Accessed on 08.06.2020] https://www.fsf.org
[16] [Accessed on 08.06.2020] https://opensource.org

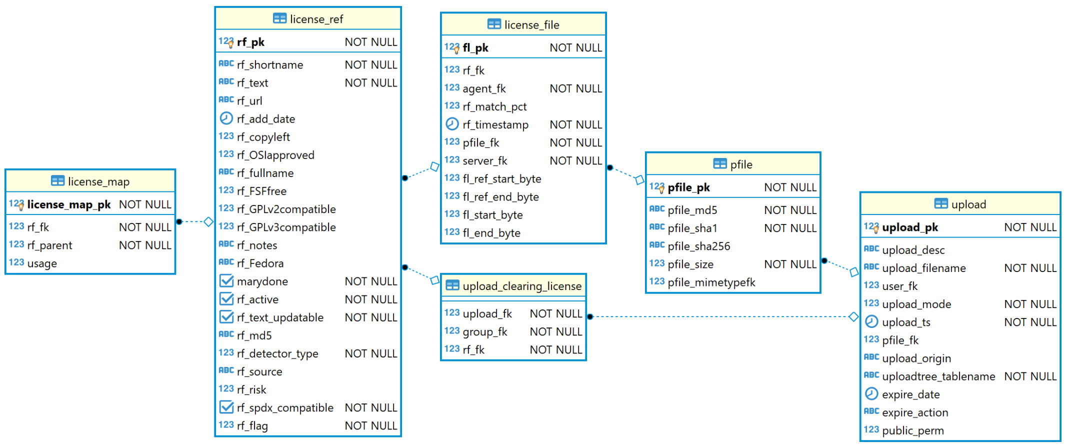


Figure 5.4: FOSSology license ERD.

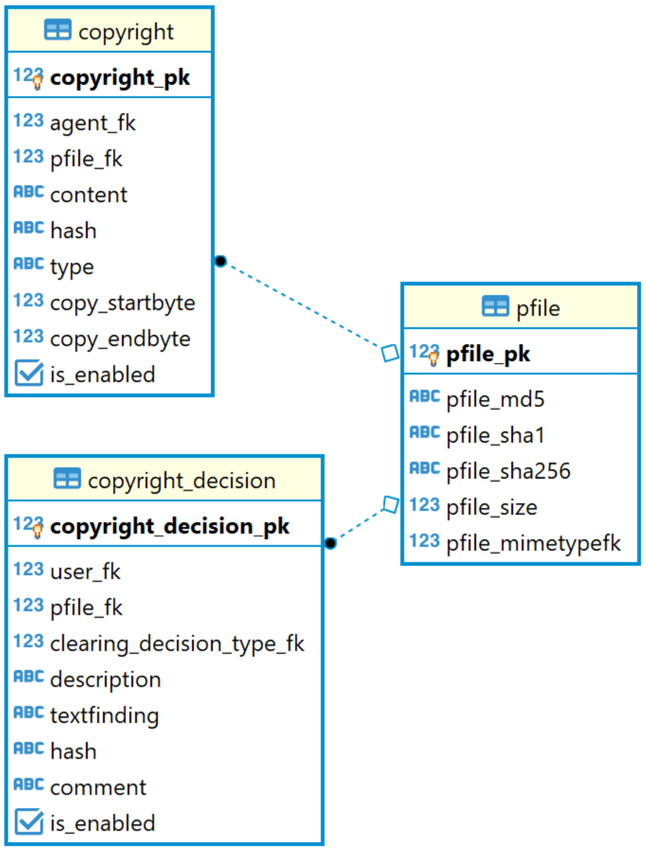


Figure 5.5: FOSSology copyright ERD.

5.2.3 How are dependencies represented?

The table uploadtree describes all relations between files, directories, special and links to its upload and packages. The file mode allows every file to be individually specified. The parent field is designated to determine which package, project, container, directory, or artifact the file belongs to and how the upload files depend on each other, see figure 5.3. Documenting how a component is linked to another from a technical perspective (static linking vs. dynamic linking) is not possible yet. Also, checking dynamically linked libraries not possible, e.g. Gradle build files.

5.2.4 How is metadata represented?

The package agent[17] searches for known packages using RPM[18] and Debian package manager[19] to store all meta-data in the database, see figure 5.6. The diagram shows how the package meta-data is stored in the database. The table pkg_rpm stores all data from RPM, and pkg_deb stores all data from the Debian package manager. In addition to the author's table, stores all data about the agent how unpacked file, in this case, package. Also, the mime type table helps add more metadata to each file.

[17][Accessed on 08.06.2020] https://fossology.github.io/pkgagent.html

[18][Accessed on 08.06.2020] https://rpm.org

[19][Accessed on 08.06.2020] https://en.wikipedia.org/wiki/Dpkg

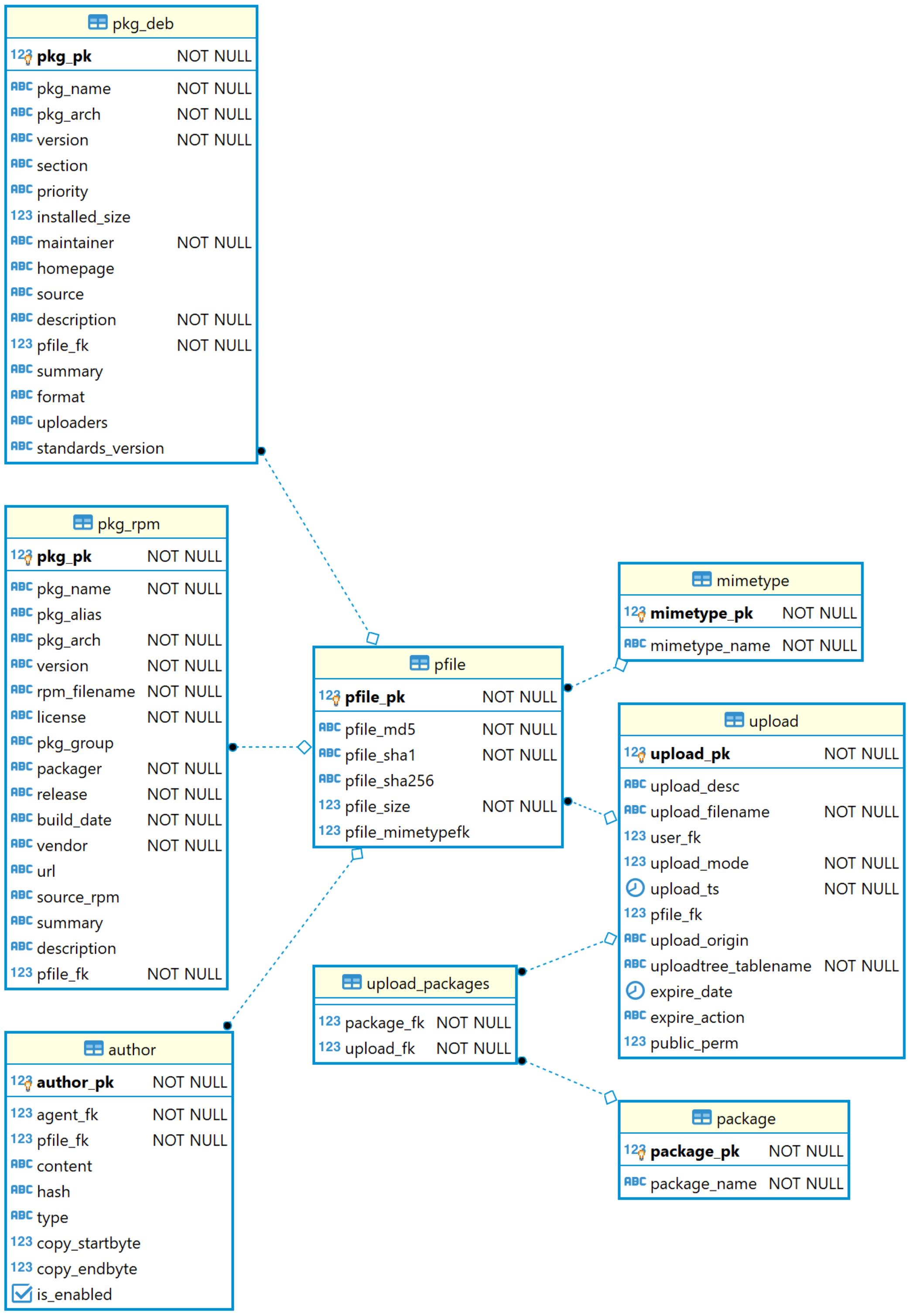


Figure 5.6: FOSSology RPM and Debian package manager ERD.

5.2.5 How are vulnerabilities represented?

FOSSology itself doesn't consider vulnerabilities a core use case, but a tool in the supply chain. *"SW360 is a server with a REST interface and a Liferay portal application to maintain your projects/products and the software components within. It can manage SPDX files for checking the license conditions and maintain license information."*[20] As one of the core use cases of SW360 is to check and manage vulnerabilities for the analyzed project.

5.2.6 What are the common use cases?

1. **License compliance:** The primary use case of FOSSology is to scan software finding all included licenses inside and review all possible conflicts or problems in license compatibility with each other.
2. **Export/Import/Edit SPDX reports:** Enabling users or organizations to share their work on a certain software via a machine-readable format SPDX tag value or RDF in this case, also allows editing the shared report and regenerate it again.
3. **Auto Deciding:** This feature is also a use case; FOSSology uses different scanners. These scanners work together if the findings are identical, the license will be auto-identified.
4. **Correction of License texts:** FOSSology allows the user to edit all licensing texts with the flexibility to add it as a variant or modify the existing ones.
5. **Copyrights statements:** Find all copyrights statements included in all of the analyzed project files.
6. **Suppliers contact information:** List all possible contact information like addresses and emails embedded in software files.

[20] [Accessed on 08.06.2020] https://github.com/eclipse/sw360

5.3 Case 3: OSS Review Toolkit (ORT)

"The OSS Review Toolkit (ORT) assists with verifying FLOSS license compliance by checking a project's source code and its dependencies"[21]. This product is licensed under the Apache-2.0 license. It started in 2017 with automation processes in mind to ensure FLOSS license compliance and give evaluation about the issues and document the analysis findings with different machine-readable formats to share it within a supply chain. All ORT tools are programmed in kotlin, a modern language that runs on the java virtual machine (JVM)[22]. This product is part of automation compliance tooling (ACT)[23], where they aim for efficient and effective exchange of SBOM to enable license compliance. ORT divided its functionalities in different tools as part of one toolkit. These tools are:

1. **Analyzer:** Scans a source code directory my facilitating various package managers and pull meta-data from ClearlyDefined[24] about the software packages included in the source code. The result of an analysis run is a dependency tree that can be represented as JSON, XML, or YAML document and share back the concluded results with ClearlyDefined via cd-upload command. It also allows defining custom policy rules to check for during the analysis for packages and dependencies.

2. **Downloader:** Job is vital to analyze transitive dependencies, the analyzer uses it as an intermediate tool, or you can use it as a separate tool before running the analyzer.

3. **Scanner:** Scans for licenses in all software packages included in the directory or the analyzed project. It uses a different license scanner, e.g. Licensee, ScanCode Toolkit, which stores the scanned packages and results locally or in a file server or in a PostgreSQL database to be reused in the future scans saving time and resources.

4. **Evaluator:** Checks license compliance and looks for issues to be resolved; it can also run custom license policy checks.

5. **Reporter:** Task is to generate machine-readable formats; it supports the following formats:

   (a) Amazon OSS Attribution Builder[25].

   (b) CycloneDX BOM, see the case 5.1.

[21][Accessed on 11.06.2020] https://github.com/oss-review-toolkit/ort

[22][Accessed on 20.09.2020] https://www.infoworld.com/article/3272244/what-is-the-jvm-introducing-the-java-virtual-machine.html

[23][Accessed on 18.06.2020] https://automatecompliance.org/community/projects/

[24][Accessed on 11.06.2020] https://clearlydefined.io/about

[25][Accessed on 11.06.2020] https://github.com/amzn/oss-attribution-builder

(c) Excel sheets.

(d) Notice summarized or full license with texts.

(e) Static HTML.

(f) Web App.

ORT has two planned tools, advisor to cover vulnerabilities checks and documenter to generate the review process outcomes, including the legal conclusion. Even though this product is not complete as designed yet, see figure E, reaching it's full potential still needs development and time. However, the currently available features and core use cases are worthy of being reviewed and considered an important case study, and the modeling done here will shape the future of FLOSS automated governance.

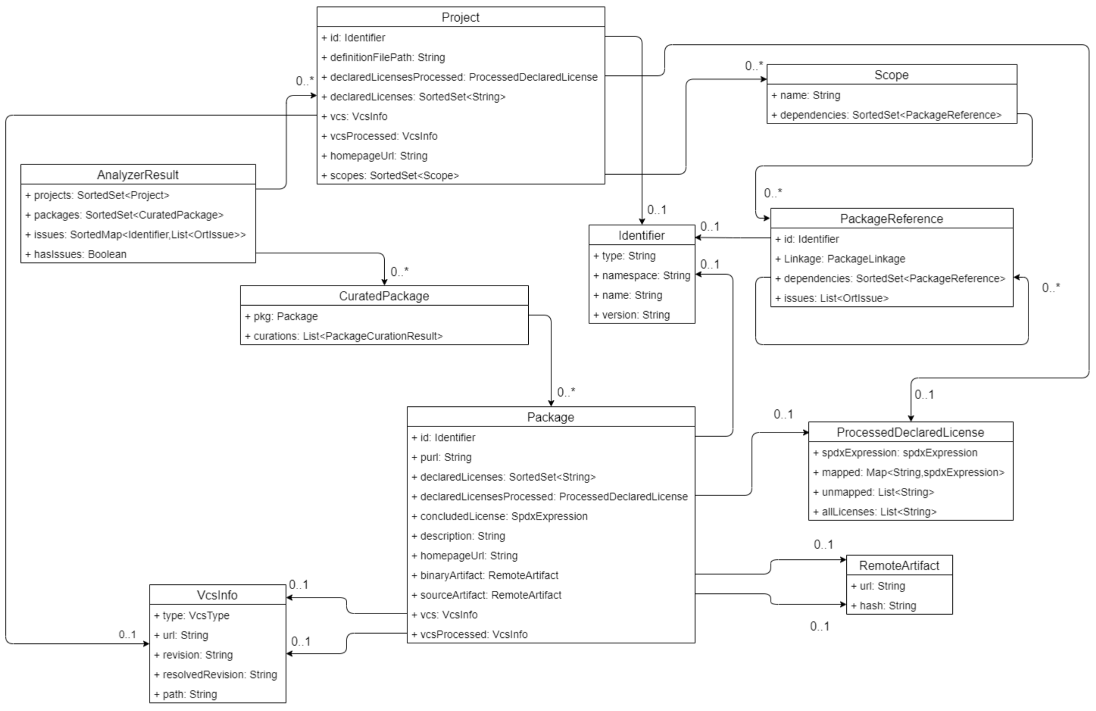

Project
+ id: Identifier
+ definitionFilePath: String
+ declaredLicensesProcessed: ProcessedDeclaredLicense
+ declaredLicenses: SortedSet<String>
+ vcs: VcsInfo
+ vcsProcessed: VcsInfo
+ homepageUrl: String
+ scopes: SortedSet<Scope>
Scope
+ name: String
+ dependencies: SortedSet<PackageReference>
AnalyzerResult
+ projects: SortedSet<Project>
+ packages: SortedSet<CuratedPackage>
+ issues: SortedMap<Identifier,List<OrtIssue>>
+ hasIssues: Boolean
PackageReference
+ id: Identifier
+ Linkage: PackageLinkage
+ dependencies: SortedSet<PackageReference>
+ issues: List<OrtIssue>
Identifier
+ type: String
+ namespace: String
+ name: String
+ version: String
CuratedPackage
+ pkg: Package
+ curations: List<PackageCurationResult>
Package
+ id: Identifier
+ purl: String
+ declaredLicenses: SortedSet<String>
+ declaredLicensesProcessed: ProcessedDeclaredLicense
+ concludedLicense: SpdxExpression
+ description: String
+ homepageUrl: String
+ binaryArtifact: RemoteArtifact
+ sourceArtifact: RemoteArtifact
+ vcs: VcsInfo
+ vcsProcessed: VcsInfo
ProcessedDeclaredLicense
+ spdxExpression: spdxExpression
+ mapped: Map<String,spdxExpression>
+ unmapped: List<String>
+ allLicenses: List<String>
RemoteArtifact
+ url: String
+ hash: String
VcsInfo
+ type: VcsType
+ url: String
+ revision: String
+ resolvedRevision: String
+ path: String
0..*
0..*
0..*
0..*
0..*
0..*
0..1
0..1
0..1
0..1
0..1
0..1
0..1
0..1
0..1
0..1

Figure 5.7: ORT UML model.

5.3.1 What is a component?

The component here is divided into **a)** projects and **b)** packages. The project contains all necessary information about the root software, which will be analyzed and scanned for sub-packages or sub-projects. The analyzer fills the packages used for describing underlying components related to the root software with all necessary information and meta-data with the help of package managers supported by ORT, e.g. Gradle, Maven, NPM, and PIP, see figure 5.7.

5.3.2 How are licenses and copyrights represented?

Both packages and the analyzed project has a known license provided by the package managers and ClearlyDefined, but that does not mean there are no hidden license compliance issues with underlying packages under the hood. That's why ORT separated licenses representation to declared licenses, processed declared licenses, and concluded licenses. The declared licenses are the listed ones by package managers or ClearlyDefined. ORT analyzer treats the processed declared licenses; mapping the licenses by its SPDX identifier if it does not have one will be counted as an unmapped license. The concluded license is an SPDX expression to express multiple license scenarios with ease as the final documented result by ORT analyzer, see lines 22 to 25 in listing 5.8.

5.3.3 How are dependencies represented?

ORT model represents the dependency tree for each package in the analyzed project with a set of scopes referencing the package reference, which considers the relationship with other packages. It also specifies if the package is dynamically or statically linked[26] library or subproject in the analyzed project itself, and it can be used recursively as deep as needed by the analyzer, see figure 5.7. the dependencies tree is essential; all dependencies source code will be downloaded and scanned for license compliance with the declared project license, see listings 5.7 and 5.8.

5.3.4 How is metadata represented?

ORT offers multiple ways to document the analyzed project and packages metadata, see figure 5.7. package and project classes store data until the results are finished. Those classes contain essential meta-data about itself, e.g. homepage URL, and PURL. VcsInfo used as a separate entity specifying information about the source code of the analyzed project on any public VCS. The binary artifacts can also be added with URLs and hashes, see listing 5.8; it shows how dependencies are described for each package and its sub-packages. ORT also can pull meta-data from ClearlyDefined about the analyzed project, allowing the user to add, edit, and fix the missing meta-data. Afterward, it can be pushed back to ClearlyDefined after the curation process is done as updated meta-data, see figure 9.4; this figure shows a defined structure by ClearlyDefined and used by ORT.

[26][Accessed on 18.06.2020] https://github.com/oss-review-toolkit/ort/blob/master/model/src/main/kotlin/PackageLinkage.kt

5.3.5 How are vulnerabilities represented?

ORT has planned for the advisor tool, which scans the analyzed project vulnerabilities based on the analyzer results and documents it for later use, not implemented yet.

5.3.6 What are the common use cases?

1. **License scan:** ORT has multiple options to use as a scanner during the analysis looking for licenses, but the default one is ScanCode Toolkit, see case 5.6.
2. **License compliance:** One of the most core use cases in this product is to download all included packages source codes and check for license incompatibilities under the hood of the analyzed project.
3. **Outdated software packages:** The evaluator will take analyzer scan results and work to see if there are any problems regarding the components, including the version of the components.
4. **Missing meta-data:** The analyzer results will show if there are any missing meta-data about any package of the analyzed project.
5. **Custom rules evaluation:** The evaluator forces the user to provide a set of rules consulted by a legal representative to give the final evaluation of the analyzer results if there are any violations within the analyzed project regarding licenses.
6. **Copyrights:** The scanners utilized by ORT look for copyright notices during the scanning of the analyzed project.
7. **Vulnerabilities check:** is a planned feature.

```
"analyzer" : {
    "start_time" : "2020-06-16T10:54:47.770881Z",
    "end_time" : "2020-06-16T10:55:16.814027Z",
    "environment" : {
      "ort_version" : "0.1.0-SNAPSHOT",
      "java_version" : "11.0.7",
      "os" : "Linux",
      "variables" : {
        "JAVA_HOME" : "/opt/java/openjdk",
        "GOPATH" : "/go"
      },
      "tool_versions" : { }
    },
    "config" : {
      "ignore_tool_versions" : false,
      "allow_dynamic_versions" : false
    },
    "result" : {
      "projects" : [ {...} ],
      "packages" : [ {
        "package" : {
            "id" : "NPM::acorn-jsx:5.2.0",
            "purl" : "pkg:npm/acorn-jsx@5.2.0",
            "declared_licenses" : [ "MIT" ],
            "declared_licenses_processed" : {
              "spdx_expression" : "MIT"
            },
            "description" : "Modern, fast React.js JSX parser",
            "homepage_url" : "https://github.com/acornjs/acorn-jsx",
            "binary_artifact" : {
              "url" : "",
              "hash" : {
                "value" : "",
                "algorithm" : ""
              }
            },
            "source_artifact" : {
              "url" : "https://registry.npmjs.org/acorn-jsx/-
              /acorn-jsx-5.2.0.tgz",
              "hash" : {
                "value" : "4c66069173d6fdd68ed85239fc256226182b2ebe",
                "algorithm" : "SHA-1"
              }
            },
            "vcs" : {
              "type" : "Git",
              "url" : "git+https://github.com/acornjs/acorn-jsx.git",
              "revision" : "c30617bd8d3763ee96fc76abfc0a9bb00e036d68",
              "path" : ""
            },
            "vcs_processed" : {
              "type" : "Git",
              "url" : "https://github.com/acornjs/acorn-jsx.git",
              "revision" : "c30617bd8d3763ee96fc76abfc0a9bb00e036d68",
              "path" : ""
            }
          },
          "curations" : [ ]
        }
      } ],
    },
    "scanner" : null,
    "evaluator" : null
}
```

Listing 5.7: Analyzer result packages JSON sample.

```
"analyzer" : {
    "start_time" : "2020-06-16T10:54:47.770881Z",
    "end_time" : "2020-06-16T10:55:16.814027Z",
    "environment" : {
      "ort_version" : "0.1.0-SNAPSHOT",
      "java_version" : "11.0.7",
      "os" : "Linux",
      "variables" : {
        "JAVA_HOME" : "/opt/java/openjdk",
        "GOPATH" : "/go"
      },
      "tool_versions" : { }
    },
    "config" : {
      "ignore_tool_versions" : false,
      "allow_dynamic_versions" : false
    },
    "result" : {
      "projects" : [ {
        "id" : "NPM::mime-types:2.1.27",
        "definition_file_path" : "package.json",
        "declared_licenses" : [ "MIT" ],
        "declared_licenses_processed" : {
          "spdx_expression" : "MIT"
        },
        "vcs" : {
          "type" : "",
          "url" : "https://github.com/jshttp/
          mime-types.git",
          "revision" : "",
          "path" : ""
        },
        "vcs_processed" : {
          "type" : "Git",
          "url" : "ssh://git@github.com/jshttp/mime-types.git",
          "revision" : "47b62ac45e9b176a2af35532d0eea4968bb9eb6d",
          "path" : ""
        },
        "homepage_url" : "",
        "scopes" : [ {
            "name" : "dependencies",
          "dependencies" : [ {
            "id" : "NPM::mime-db:1.44.0"
          } ]
        }, {
          "name" : "devDependencies",
          "dependencies" : [ {
            "id" : "NPM::eslint-config-standard:14.1.1"
          }, {
            "id" : "NPM::eslint-plugin-import:2.20.2",
            "dependencies" : [ {
              "id" : "NPM::array-includes:3.1.1",
              "dependencies" : [ {
                "id" : "NPM::define-properties:1.1.3",
                "dependencies" : [ {
                  "id" : "NPM::object-keys:1.1.1"
                } ]
              }
            }],
        }],
      }],
    }],
    "packages" : [ {...} ],
    },
    "scanner" : null,
    "evaluator" : null
}
```

Listing 5.8: Analyzer JSON result sample.

5.4 Case 4: Tern

"Tern is an inspection tool to find the metadata of the packages installed in a container image"[27]. This product is licensed under the BSD-2 License, which started in 2017 by VMWare, as a member of ACT[23]. Tern has a different use case, unlike the others, it specializes in scanning containers and image layers for packages licenses, and it is wholly programmed in python and shell, see figure 5.8. Tern has four primary elements working together to perform the following steps:

1. Mount the BaseOS layer to build the container.
2. Scans for installed packages in that layer.
3. Repeat steps 1 and 2 for the rest of the container layers.
4. Generate reports.

These four elements are[28]:

1. **The Cach:** This is the database where filesystem identifiers can be queried against to retrieve package information. That is useful as many containers are based on other container images. If Tern had come across the same filesystem in another container, it could retrieve the package information without spinning up a container[28]. It helps save resources and time, reusing filesystem identifiers.
2. **The Command library:** This is a database of shell commands that may create a container's layer filesystem. There are two types of shell commands - one for system-wide package managers and one for custom shell commands or install scripts. The library is split in this way to account for situations where whole root filesystems are imported to create a new container. When Tern uses an external file scanner, it bypasses the command library altogether and relies on the external tool's results. The approach allows Engineers to compare results from different tools available to them as they have always done, but on container images.
3. **The Analyzer:** Tern has a dedicated analyzer to the type of image being analyzed. Currently, it can analyze only images created by Docker. The inspection part can be done using Tern's native analyzer or an external tool. The analyzer will collate the metadata it can get in image objects, which encapsulate data for each layer and each package found. It also encapsulates

[27][Accessed on 23.06.2020] https://github.com/tern-tools/tern

[28][Accessed on 24.06.2020] https://github.com/tern-tools/tern/blob/master/docs/architecture.md

notes while execution takes place. The native analyzer logical flow in figure 5.9.

4. **The Formatter:** Generates reports in different machine-readable representations, the available formats are text, JSON, YAML, and SPDX tag-value. It provides SBOM to see the underlying hidden software dependencies in the container and helps with the decision-making process, how to deal with the software and its risks.

Tern allows us to choose between the native analyzer and ScanCode Toolkit as an extension scanning for licenses, packages, and copyrights to analyze license compliance. Tern also performs vulnerability checks by utilizing another extension for this task. CVE Binary Tool is a command line tool which scans for several common, vulnerable components (openSSL, libpng, libxml2, expat, and others) to let you know if your system includes common libraries with known vulnerabilities with additional report options. Tern allows BOM generation from the docker image or docker file.

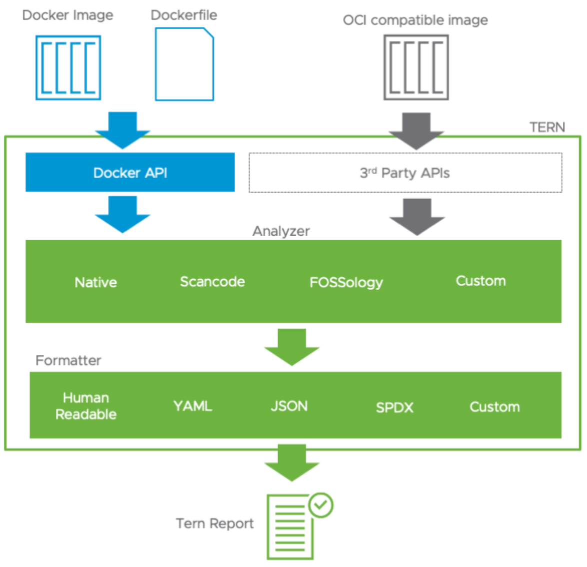


Figure 5.8: Tern general architecture diagram.[28]

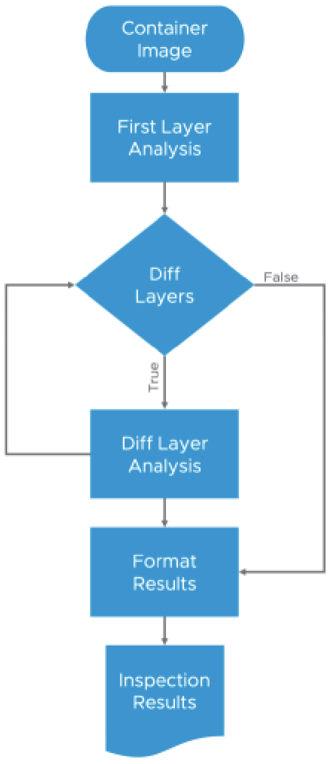


Figure 5.9: Tern analyzer logical model.[28]

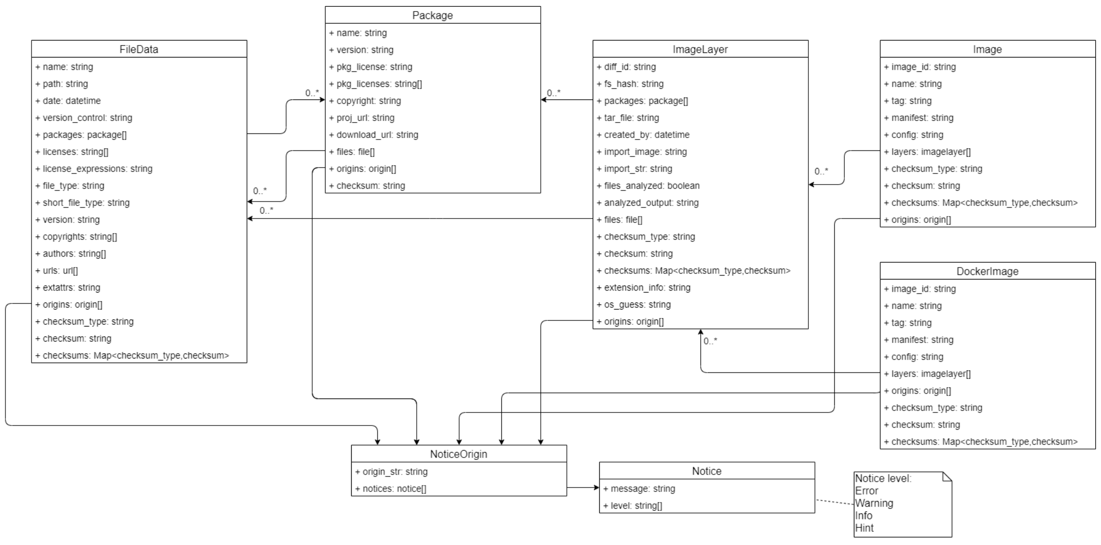


Figure 5.10: Tern UML model.

5.4.1 What is a component?

Tern divides the component into three primary elements: **a)** image layer, **b)** package, and **c)** file. Since Tern is specialized in container scans dividing container to layers as the docker file does, each image layer holds a list of all packages included in it. The package contains all critical information with a full list of all included files. The file contains all necessary information about each file, including a list of multiple packages that might belong to it. Tern uniquely represents the component compared to other case study subject, see figure 5.10.

5.4.2 How are licenses and copyrights represented?

As mentioned in section 5.4.1, the package has a declared license as meta-data are pulled from the Tern cache. The other represents all included licenses in the package with copyright. As for file, it has a list of all licenses in the file with the possibility to express multiple licenses scenario through SPDX expressions and the list can be left empty in this case, it also has a list of copyrights in case of multiple exists in the file, see figure 5.10. If any notice exists about any packages regarding license compliance, it is represented in a list of origins, which filled by the analyzer while it runs, see the listing 5.10 lines 80 to 91.

5.4.3 How are dependencies represented?

Tern represents the dependency is a list of packages and files included in each image layer, as for packages, it holds a list of all included files in it. The file has a list of packages that might belong to it if some packages have common files with another. This how Tern represents dependency, including the image layer in it, see figure 5.10. Also, see the listing 5.10 lines 42 to 44, which explains the relationship between layers and packages in SPDX format, this example has only one layer.

5.4.4 How is metadata represented?

The architecture introduces Tern[28]. All images meta-data will be pulled from the docker hub as it supports only docker for now. The docker image and inctance contains all meta-data about the container. As for packages and files, meta-data is coming from the cache, and if there are any missing pieces of information, Tern will issue a warning to fill the missing ones, see the listing 5.10 lines 54 to 60 and 70 to 76. Each element has its meta-data included in it, as the Tern data-model indicates, see figure 5.10. Also, see the listing 5.9, the lines from 3 to 65 show the meta-data about the container, which comes from the docker hub.

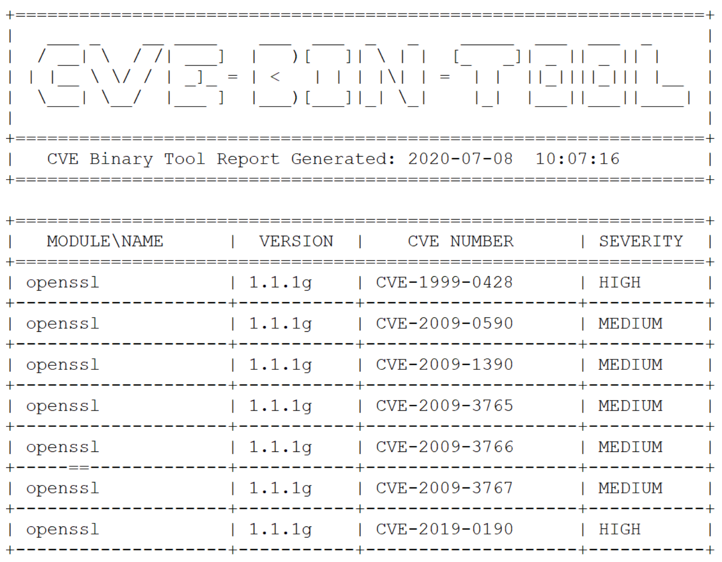
5.4.5 How are vulnerabilities represented?

The vulnerability checks are provided by the extension CVE Binary Tool[29], which reports its findings separated from the analyzer. The available output formats for CVE Binary Tool extension are only YAML and JSON, which looks like the default analyzer output except for the analyzer_output field. It contains the vulnerabilities for each layer as a table with extra information about each layer scan, see figure 5.11.

5.4.6 What are the common use cases?

1. **License scans for containers:** This is the primary use case of Tern to search for packages licenses included in each layer of the container reporting the missing information.
2. **License compliance:** Tern does not look for licenses only, but also gives warnings about possible compliance issues in the analyzed project.
3. **Missing meta-data:** The results of the scans also contains warnings about the missing meta-data about the analyzed project, e.g. copyrights, hashes, URLs, and licenses.
4. **Copyrights:** The default analyzer and ScanCode Toolkit extension both perform copyrights scans during the analysis.
5. **Vulnerability checks:** Use a third-party extension CVE Binary Tool to perform these checks to provide CVEs list for the analyzed project.

[29][Accessed on 23.06.2020] https://github.com/intel/cve-bin-tool



```
+=================================================================+
|                                                                 |
|   ___ _    __ ____    ___  ___  _  _    _____  ___  ___  _      |
|  / __| \  / /| ___]  |   )[   ]| \ | |  [_   _]| _ || _ || |    |
| | |__ \ \/ / | _]_ = | <    | | | |\| | =  | |  ||_||||_||| |__ |
|  \___| \__/  |___ ]  |___)[___]|_| \_|    |_|  |___||___||____| |
|                                                                 |
+=================================================================+
|   CVE Binary Tool Report Generated: 2020-07-08  10:07:16        |
+=================================================================+

+=================================================================+
|   MODULE\NAME        |  VERSION  |    CVE NUMBER       | SEVERITY |
+=================================================================+
| openssl              | 1.1.1g    | CVE-1999-0428       | HIGH     |
+----------------------+-----------+---------------------+----------+
| openssl              | 1.1.1g    | CVE-2009-0590       | MEDIUM   |
+----------------------+-----------+---------------------+----------+
| openssl              | 1.1.1g    | CVE-2009-1390       | MEDIUM   |
+----------------------+-----------+---------------------+----------+
| openssl              | 1.1.1g    | CVE-2009-3765       | MEDIUM   |
+----------------------+-----------+---------------------+----------+
| openssl              | 1.1.1g    | CVE-2009-3766       | MEDIUM   |
+-----==---------------+-----------+---------------------+----------+
| openssl              | 1.1.1g    | CVE-2009-3767       | MEDIUM   |
+----------------------+-----------+---------------------+----------+
| openssl              | 1.1.1g    | CVE-2019-0190       | HIGH     |
+----------------------+-----------+---------------------+----------+
```

Figure 5.11: Tern CVE binary tool extension sample.

```
# This report was generated by the Tern Project
# ('package', '2.1.0')
image:
  Image__checksum: ...
  Image__checksum_type: sha256
  Image__checksums: []
  config:
    architecture: amd64
    config:
      AttachStderr: false
      ...
      Cmd:
     - /bin/sh
      —...
      Domainname: ''
      Entrypoint: null
      Env:
      —...
      ExposedPorts:
        8080/tcp: {}
      Healthcheck:
        Interval: 300000000000
        Test:...
        Timeout: 3000000000
      Hostname: ''
      Image: sha256:...
      Labels:
        maintainer: steve.springett@owasp.org
        vendor: OWASP
      OnBuild: null
      OpenStdin: false
      StdinOnce: false
      Tty: false
      User: '1000'
      Volumes: null
      WorkingDir: /opt/owasp/dependency—track/
    container: sha256:...
    container_config:
      AttachStderr: false
      AttachStdin: false
      AttachStdout: false
      Cmd: ...
      Domainname: ''
      Entrypoint: null
      Env: ...
      ExposedPorts:
        8080/tcp: {}
      Healthcheck:
        Interval: 300000000000
        Test: ...
        Timeout: 3000000000
      Hostname: b019223e956a
      Image: sha256:...
      Labels:
        maintainer: steve.springett@owasp.org
        vendor: OWASP
      OnBuild: null
      OpenStdin: false
      StdinOnce: false
      Tty: false
      User: '1000'
      Volumes: null
      WorkingDir: /opt/owasp/dependency—track
    created: '2020-03-22T23:02:49.8701459Z'
    docker_version: 19.03.8
    history: &id001
   - created: '2019-08-20T20:19:55.062606894Z'
      created_by: '...'
   - created: '2019-08-20T20:19:55.211423266Z'
      created_by: '/bin/sh -c #(nop) CMD ["/bin/sh"]'
      empty_layer: true
    ...
   - created: '2020-03-22T23:02:49.8701459Z'
      created_by: '/bin/sh -c #(nop) HEALTHCHECK &{["CMD-SHELL"]'
      empty_layer: true
    os: linux
    rootfs:
      diff_ids:
     - sha256: ...
     - sha256: ...
      ...
      type: layers
  history: *id001
  image_id: ...
  layers:
 - analyzed_output: ''
    checksum: ...
    checksum_type: sha256
    checksums: {}
    created_by: '/bin/sh -c #(nop) ADD
    file: ...
    diff_id: ...
    extension_info: {}
    files:
    ...
 - analyzed_output: ''
    ...
    manifest:
   - Config: ...config.json
      Layers:
     - a61298a1d179786c2c176dc6c3e20c2d0cad.../layer.tar
     - .../layer.tar
     - .../layer.tar
     - .../layer.tar
     - .../layer.tar
      RepoTags: &id002
     - owasp/dependency—track:latest
    name: owasp/dependency—track
    origins:
   - notices: []
      origin_str: 'Docker image: owasp/dependency-track:latest'
    repotag: owasp/dependency—track:latest
    repotags: *id002
    tag: latest
```
Listing 5.9: Tern YAML report sample.

```
SPDXVersion: SPDX-2.2
DataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
DocumentName: Tern report for alpine
DocumentNamespace: https://spdx.org/spdxdocs/tern-report-...-alpine
LicenseListVersion: 3.8
Creator: Tool: tern-12ba057e29095100995d11f973fc76cd1a5c0f70
Created: 2020-06-25T14:32:50Z
DocumentComment: <text>This was generated by the Tern</text>
PackageName: alpine
SPDXID: SPDXRef-a24bb40132-alpine-latest
PackageVersion: latest
PackageDownloadLocation: alpine:latest
FilesAnalyzed: false
PackageLicenseConcluded: NOASSERTION
PackageLicenseDeclared: NOASSERTION
PackageCopyrightText: NOASSERTION
Relationship: SPDXRef-...-alpine-latest CONTAINS SPDXRef-...
PackageName: layer.tar
SPDXID: SPDXRef-50644c29ef
PackageFileName: fa27c...d432/layer.tar
PackageDownloadLocation: fa27...ad432/layer.tar
FilesAnalyzed: false
PackageChecksum: SHA256: 5064...dd0a
PackageLicenseConcluded: NOASSERTION
PackageLicenseDeclared: NOASSERTION
PackageCopyrightText: NOASSERTION
PackageComment: <text>
Layer: 70b38e3fcc:
  info: Found 'Alpine Linux v3.12' in /etc/os-release.
  info: Layer created by commands: /bin/sh ...
  info: Retrieved by invoking listing in command_lib/base.yml
versions:
  in container: ...
proj_urls:
  in container: ...
names:
  in container: ...
licenses:
  in container: ...
</text>
Relationship: SPDXRef-... CONTAINS SPDXRef-musl-1.1.24-r8
Relationship: SPDXRef-... CONTAINS SPDXRef-busybox-1.31.1-r16
...
PackageName: musl
SPDXID: SPDXRef-musl-musl-1.1.24-r8
PackageVersion: musl-1.1.24-r8
PackageDownloadLocation: NONE
FilesAnalyzed: false
PackageLicenseConcluded: NOASSERTION
PackageLicenseDeclared: LicenseRef-c7ea3b7
PackageCopyrightText: NONE
PackageComment: <text>
musl:
  warning: No metadata for key: copyright
  warning: No metadata for key: download_url
  warning: No metadata for key: checksum
  warning: No metadata for key: files
  warning: No metadata for key: pkg_licenses
</text>
PackageName: busybox
SPDXID: SPDXRef-busybox-busybox-1.31.1-r16
PackageVersion: busybox-1.31.1-r16
PackageDownloadLocation: NONE
FilesAnalyzed: false
PackageLicenseConcluded: NOASSERTION
PackageLicenseDeclared: LicenseRef-c66410f
PackageCopyrightText: NONE
PackageComment: <text>
busybox:
  warning: No metadata for key: copyright
  warning: No metadata for key: download_url
  warning: No metadata for key: checksum
  warning: No metadata for key: files
  warning: No metadata for key: pkg_licenses
</text>
...
LicenseID: LicenseRef-5a8406a
ExtractedText: <text>Original license: BSD-2-Clause AND BSD-3-Clause</text>
LicenseID: LicenseRef-c66410f
ExtractedText: <text>Original license: GPL-2.0-only</text>
LicenseID: LicenseRef-1eaea05
ExtractedText: <text>Original license: ISC</text>
LicenseID: LicenseRef-c7ea3b7
ExtractedText: <text>Original license: MIT</text>
LicenseID: LicenseRef-f262406
ExtractedText: <text>Original license: Zlib</text>
LicenseID: LicenseRef-d312664
ExtractedText: <text>Original license: MIT BSD GPL2+</text>
LicenseID: LicenseRef-f30c02b
ExtractedText: <text>Original license: MPL-2.0 GPL-2.0-or-later</text>
LicenseID: LicenseRef-de5acdd
ExtractedText: <text>Original license: OpenSSL</text>
```
Listing 5.10: Tern SPDX tag value report sample.

5.5 Case 5: Quartermaster

"Quartermaster is an integrated free and open-source software (FOSS) toolchain that implements industry best practices of license compliance management".[30] This product is licensed under the GPL-3.0 license; it started in 2017 with Siemens support as a FLOSS product and a member of ACT[23]. It has a unique feature unlike all previous cases; it runs in building/compiling time. The Quartermaster process starts right before building the analyzed project and finishes with reporting after it is done. Still, it would be best if you made sure the analyzed project runs very well without any issues before integrating Quartermaster to its build process. It also uses a flexible architecture dividing it to the following modules:

1. **Analyzers module:** This module is responsible for scanning the analyzed project for license information, such as the available two in this case:
    - **SPDX-Identifier:** The analyzer looks for SPDX-identifier across the analyzed project files.
    - **ScanCode Toolkit:** Quartermaster also utilizes the well-known scanner ScanCode Toolkit as one of its modules, see case 5.6.
2. **Builders:** It holds the supported building systems modules responsible for building the analyzed project, e.g. Gradle, Maven, and Python builder.
3. **Manifests:** Contain modules that define the reports outputs in machine-readable formats, in this case, SPDX.
4. **Packages:** It holds the supported package manager, e.g. Debian package manager.
5. **Reporters:** It holds reporting modules responsible for generating reports with the Manifests module, e.g. SPDX, after the build is finished as the last step.

Each module communicates with others via the gRPC framework[31] managed by the master process; this flexible architecture made the module programming language choices unlimited as long it supports gRPC. Quartermaster aggregates the acquired knowledge in a graph database Dgraph[32] managed by the master process.

[30][Accessed on 30.06.2020] https://qmstr.org/documentation/
[31][Accessed on 30.06.2020] https://grpc.io/
[32][Accessed on 30.06.2020] https://dgraph.io/

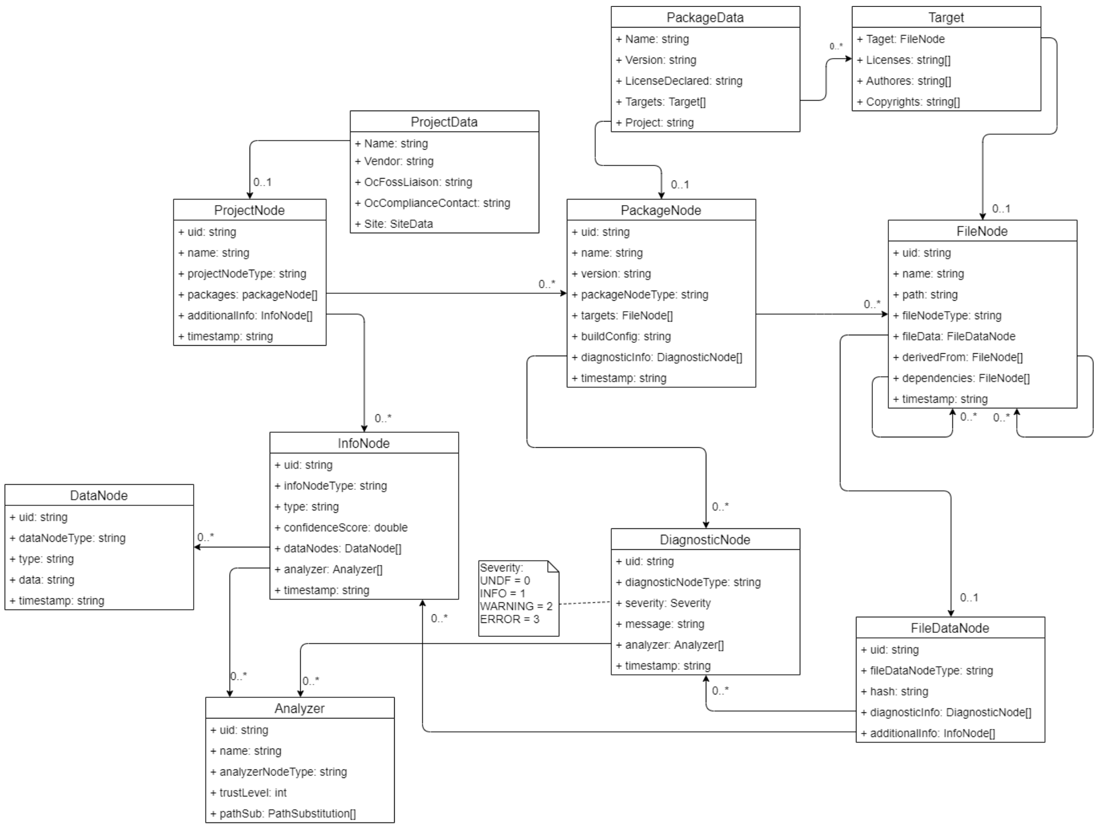


Figure 5.12: Quartermaster UML model.

5.5.1 What is a component?

Quartermaster divides the analyzed project into three primary elements **a)** project, **b)** packages, and **c)** files each project has a list of included packages. The packages also have a list of included files. The corporation between these elements represents the component, see figure 5.12.

5.5.2 How are licenses and copyrights represented?

The package node has the package declared license by the vendor and a list of targets. These are the files included in the package. This node has a data node class that links all these targets with its copyrights, authors, and a list of licenses found in the file. Then these results generated in the reporting phase as a machine-readable format. In addition to the diagnostic node, allow the analyzer to document any SPDX license expressions well, scanning the analyzed project, and reported after finishing the build phase with severity level indicating the significance of this finding, mentioned as a note in figure 5.12. All SPDX identifiers are stored as a map of a string with struct to contain all the matched license information[33]. If the found license doesn't exist in the SPDX identifiers list, it will be added as a diagnostic node with a severity level of an error, see listings 5.12 and 5.11.

5.5.3 How are dependencies represented?

Quartermaster illustrates dependencies on a file-level; in the file node, it has a list of itself as a dependency field, which holds all dependent files. All related files are listed here, and Quartermaster also has a derived field that holds all parent files of the selected file node, see figure 5.12. It also considers dynamically linked libraries via supported building systems, e.g. Gradle.

5.5.4 How is metadata represented?

In addition to significant meta-data included in each element, there is a flexible way to add more meta-data from the analyzers or the user via info node, which can be added to the project node, and file data node as much as needed. Info node type can determine the type of this node, e.g. meta-data; see figure 5.12. The meta-data about the analyzed project is in a YAML report see listing 5.11, and as for the analyzer results found in the SPDX tag value file, see listing 5.12.

5.5.5 How are vulnerabilities represented?

As far as we know, Quartermaster doesn't have any extensions or any services yet to perform vulnerabilities, or any other supply chain tools that can integrate with it.

5.5.6 What are the common use cases?

1. **License scan:** During the build phase, all used source code scanned for license generating report contains all license compliance information. If everything is set up properly and the meta-data in the project source code is well-maintained. It also distinguishes between test files and releases files; the scanned is only release files.
2. **License compliance:** The analyzer module utilizes different scanners; all of them give reportings about possible license compliance issues within the analyzed project.
3. **Copyrights:** The analyzer looks for copyrights statements during the analysis, reporting its findings in different available machine-readable formats.

[33][Accessed on 02.07.2020] https://github.com/QMSTR/qmstr/blob/master/modules/analyzers/spdx-identifier-analyzer/spdx_identifiers.go

4. **Missing meta-data:** Quartermaster CLI allows the user to fill and store the missing meta-data for any project, file, or package-node utilize it later for reporting.

```yaml
project:
  name: "json-c"
  metadata:
    Vendor: "Endocode"
    OcFossLiaison: "Mirko Boehm"
    OcComplianceContact: "foss@endocode.com"
  analysis:
   - analyzer: spdx-identifier-analyzer
     name: "Simple SPDX Analyzer"
     trustlevel: 300
     config:
       workdir: "/buildroot"
   - analyzer: scancode-analyzer
     name: "Scancode Analyzer"
     trustlevel: 400
     config:
       workdir: "/buildroot/jsonc"
       resultfile: "/buildroot/scancode.json"
       #cached: "true"
   - analyzer: test-analyzer
     name: "Simple CI Test Analyzer"
     config:
       workdir: "/buildroot"
       tests: "TestPackageNode"
   - analyzer: spdx-analyzer
     name: "SPDX Analyzer"
     trustlevel: 300
     config:
       spdxfile: "/buildroot/SPDX.tag"
  reporting:
   - reporter: test-reporter
     name: "Test Reporter"
     config:
       siteprovider: "Endocode"
   - reporter: qmstr-reporter-html
     name: "HTML Reporter"
     config:
       #generatehtml: "no"
       siteprovider: "Endocode"
       baseurl: "http://localhost:8080/"
       outputdir: /buildroot/
   - reporter: package-manifest-reporter
     name: "Package manifest Reporter"
     config:
       outputdir: "/buildroot"
```

Listing 5.11: Quartermaster YAML report sample.[33]

[33][Accessed on 13.07.2020] https://github.com/QMSTR/qmstr-demo/demos/jsonc/qmstr.yaml

```
SPDXVersion: SPDX−1.2
DataLicense: CC0−1.0
DocumentComment: <text>This is the spdx document for the Calculator
    </text>

## Creation Information
Creator: Person: Math Codder
Creator: Organization: Endocode AG
Created: 2018−05−16T00:00:00Z
CreatorComment: <text>This is an example of an SPDX spreadsheet
    format for CI demo tests</text>

## Review Information
Reviewer: Person: Joe Reviewer
ReviewDate: 2018−05−17T00:00:00Z
ReviewComment: <text>LGTM</text>

## Package Information
PackageName: libjson−c
PackageVersion: Version 0.1
PackageDownloadLocation: http://www.example.org/thecalc
PackageSummary: <text>A simple calculation utility</text>
PackageSourceInfo: <text>Version 0.1 of the Calculator application</text>
PackageFileName: thecalc−0.1.tar.gz
PackageSupplier: Organization:Endocode AG
PackageOriginator: Organization:Endocode AG
PackageChecksum: SHA1: 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
PackageVerificationCode: 4e3211c67a2d28fced849ee1bb76e7391b93feba (
    SpdxTranslatorSpdx.rdf, SpdxTranslatorSpdx.txt)
PackageDescription: <text>This utility calculates.</text>

PackageCopyrightText: <text> Copyright 2017, 2018 Endocode AG</text
    >

PackageLicenseDeclared: GPL−3
PackageLicenseConcluded: GPL−3
PackageLicenseInfoFromFiles: GPL−3
PackageLicenseInfoFromFiles: GPL−3
PackageLicenseComments: <text>Just GPL 3</text>


# File Info

FileName: Calculator/add.c
FileType: SOURCE
FileChecksum: SHA1: 7c2f3b2fb26ad864f443bbecee8a059c91c83d26
LicenseConcluded: GPL−3
LicenseInfoInFile: GPL−3
FileCopyrightText: <text>Copyright 2018 Endocode AG</text>
ArtifactOfProjectName: The Calculator
ArtifactOfProjectHomePage: http://www.endocode.com/
ArtifactOfProjectURI: http://www.endocode.com/
FileComment: <text>perform addition</text>
```

Listing 5.12: Quartermaster SPDX tag value report sample.[33]

[33][Accessed on 13.07.2020] https://github.com/QMSTR/qmstr-demo/demos/jsonc/SPDX.tag

5.6 Case 6: ScanCode Toolkit

"A typical software project often reuses hundreds of third-party packages. License and origin information is not always easy to find and not normalized: ScanCode Toolkit discovers and normalizes this data for you."[34] ScanCode Toolkit is a FLOSS product licensed under Apache-2.0, CC0-1.0, and multiple licenses for its plugins, e.g. MIT, GPL-3.0, and BSD. This product is one of the best, earliest (started 2013) and most used in this field of license scanning business. It is utilized by other products responsible for FLOSS license compliance and governance tools, see cases 5.3, 5.4 and 5.5. ScanCode Toolkit has a flexible architecture that divides the process into modules; each module can have one or more plugins programmed in C++ or python. The plugin phases are divided into three types[35]:

1. **Pre:** Before the scan begins, these plugins are responsible for extracting the analyzed project archives or handling specific file types.
2. **Scan proper:** During the scan, these are responsible for investigating the analyzed project files for licenses, keywords, phrases, copyrights, and metadata or specific license policies or rules.
3. **Post:** After the scan finishes, those are responsible for generating reports in machine-readable formats.

ScanCode Toolkit performs scans on a code-base in the following steps:

1. Collect an inventory of the code files and classify the code using file types.
2. Extract files from an archive using a general-purpose extractor.
3. Extract texts from binary files if needed.
4. Use an extensible rules engine to detect open source license text and notices.
5. Use a specialized parser to capture copyright statements.
6. Identify packaged code and collect metadata from packages.
7. Report the results in a format of your choice, e.g. JSON, for integration with other supply chain tools, see listing 5.13.

The scan results contain the following information for each file:

1. File path.
2. Detected licenses.
3. Copyrights statements.

[34][Accessed on 05.07.2020] https://github.com/nexB/scancode-toolkit

[35][Accessed on 05.07.2020] https://scancode-toolkit.readthedocs.io/en/latest/plugins/plugin_arch.html

4. Location of the findings in lines.

5. References for the recognized licenses.

ScanCode Toolkit can be used for individual users in multiple platforms, e.g. Windows, Linux, and macOS via CLI. There is also a tool to visualize the scan results to help the user evaluate licenses, and other notices identified, provided by nexB called ScanCode Workbench[36].

```
output formats:
--json FILE Write scan output as compact JSON to FILE.
--json-pp FILE Write scan output as pretty-printed JSON to FILE.
--json-lines FILE Write scan output as JSON Lines to FILE.
--csv FILE Write scan output as CSV to FILE.
--html FILE Write scan output as HTML to FILE.
--custom-output FILE Write scan output to FILE formatted with the
  custom Jinja template file.
--custom-template FILE Use this
  Jinja template FILE as a custom template.
--spdx-rdf FILE Write scan output as SPDX RDF to FILE.
--spdx-tv FILE Write scan output as SPDX Tag/Value to FILE.
--html-app FILE (DEPRECATED: use the ScanCode Workbench app
instead) Write scan output as a mini HTML application to FILE.
```

Listing 5.13: ScanCode Toolkit reporting options in v3.1.1.

[36][Accessed on 05.07.2020] https://github.com/nexB/scancode-workbench

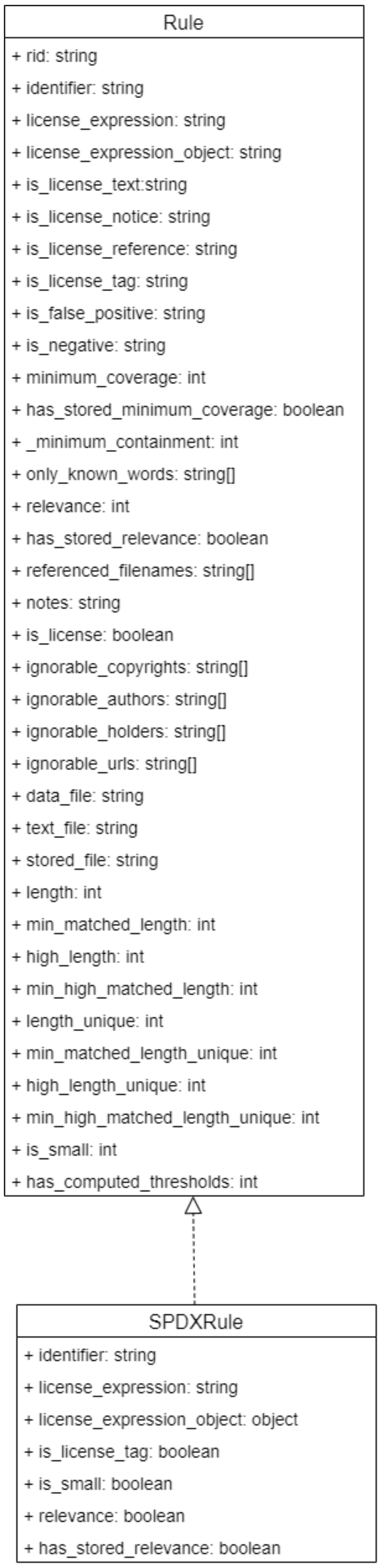

Rule
+ rid: string
+ identifier: string
+ license_expression: string
+ license_expression_object: string
+ is_license_text:string
+ is_license_notice: string
+ is_license_reference: string
+ is_license_tag: string
+ is_false_positive: string
+ is_negative: string
+ minimum_coverage: int
+ has_stored_minimum_coverage: boolean
+ _minimum_containment: int
+ only_known_words: string[]
+ relevance: int
+ has_stored_relevance: boolean
+ referenced_filenames: string[]
+ notes: string
+ is_license: boolean
+ ignorable_copyrights: string[]
+ ignorable_authors: string[]
+ ignorable_holders: string[]
+ ignorable_urls: string[]
+ data_file: string
+ text_file: string
+ stored_file: string
+ length: int
+ min_matched_length: int
+ high_length: int
+ min_high_matched_length: int
+ length_unique: int
+ min_matched_length_unique: int
+ high_length_unique: int
+ min_high_matched_length_unique: int
+ is_small: int
+ has_computed_thresholds: int
SPDXRule
+ identifier: string
+ license_expression: string
+ license_expression_object: object
+ is_license_tag: boolean
+ is_small: boolean
+ relevance: boolean
+ has_stored_relevance: boolean

Figure 5.13: ScanCode Toolkit license and rules UML model.

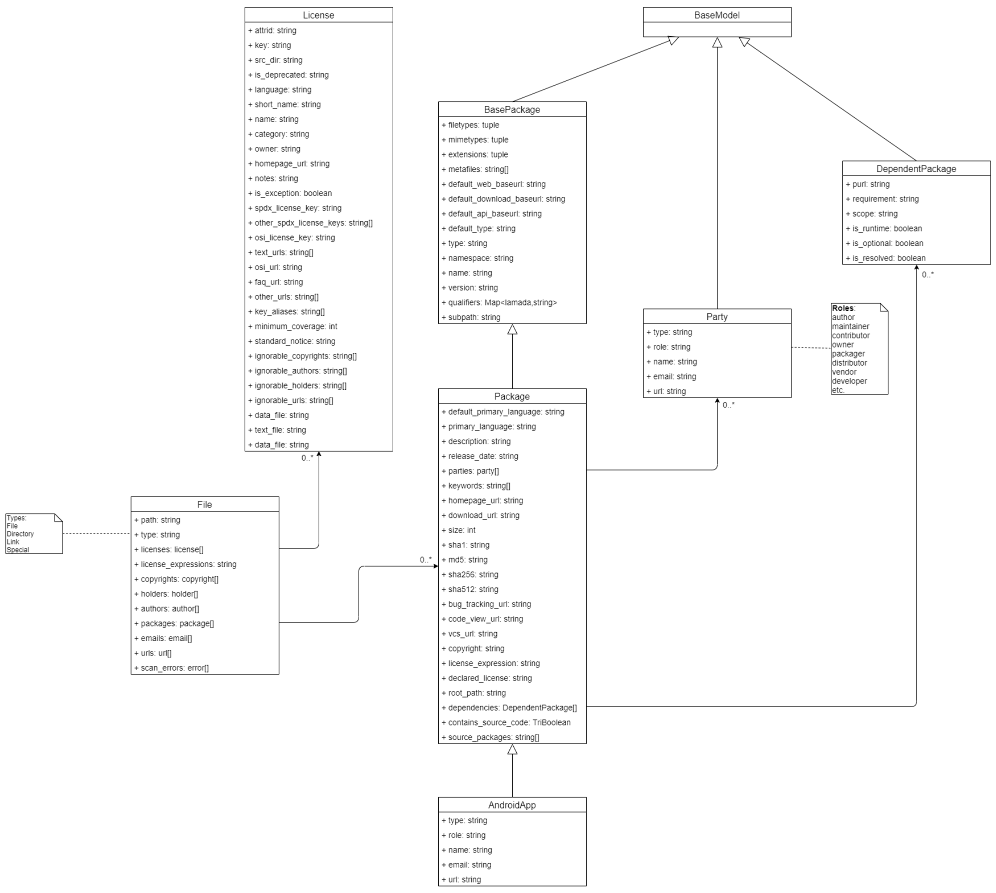


Figure 5.14: ScanCode Toolkit UML model.

```
        @attr.s()
        class AndroidApp(Package):
            filetypes = ('zip archive',)
            mimetypes = ('application/zip',)
            extensions = ('.apk',)
            default_type = 'android'
            default_primary_language = 'Java'
```

Listing 5.14: Custom package example.

5.6.1 What is a component?

ScanCode Toolkit considers a component as **a)** packages and **b)** files. The model has a base package that holds the necessary information about it. A successor to the base package, which is package inherits all the base package information and methods with far more extended information options. ScanCode Toolkit has an extended list of package types inherited from the package class has specified information about it, e.g. AndroidApp, see figure 5.14 and listing 5.14. The file contains all critical information, e.g. licenses, copyrights, holders, and authors, including packages list, in case the file is repeated in multiple packages, see figure 5.14.

5.6.2 How are licenses and copyrights represented?

The list of known licenses is stored inside the license module in a data directory to reference it during the scans. As for the runtime, the model has a license class that holds viable information to link it to the default rules or customized once, e.g. SPDX rules; see figure 5.13. The package itself has a license expression, which accepts SPDX expressions and a declared license by the vendor, see figure 5.14 and see sample 5.15, the lines 38 to 76 showing how the license/licenses represented in the JSON report.

5.6.3 How are dependencies represented?

ScanCode Toolkit model the dependencies on both package level and file level, unlike other cases in this thesis. With a list of dependent package class, ScanCode Toolkit describes all the related packages to the scanned one, with URLs and requirement for the dependent package, see figure 5.14. Documenting how a component is linked to another from a technical perspective (static linking vs. dynamic linking) is not covered by ScanCode Toolkit but in another planned product called DependentCode[37] by the same vendor nexB.

5.6.4 How is metadata represented?

During the scan phase, ScanCode Toolkit gathers meta-data about the package from the available source code, which depends on the source code's well-being and maintainability by the vendor. The package class has the most of meta-data about each package from the analyzed project. In addition to a list of responsible parties, e.g. vendor, developer, and communities, as ScanCode Toolkit describes it, provided as a note, with all vital contact information, see figure 5.14 and see sample 5.15; it shows how all contact information, copyrights, and holders represented. In addition to the configuration was passed on to ScanCode Toolkit, see lines 2 to 31. See sample in SPDX tag value 5.16.

5.6.5 How are vulnerabilities represented?

ScanCode Toolkit roadmap mentions a planned supply chain tool called VulnerableCode to scan for vulnerabilities. It uses CVE and NVD, which still work in

[37][Accessed on 05.07.2020] https://github.com/nexB/dependentcode

progress[38].

5.6.6 What are the common use cases?

1. **License scan:** ScanCode Toolkit primary function to scan each file for the included licenses, demonstrating the licenses' detailed information.
2. **License compliance:** ScanCode Toolkit can provide compliance information via the customizable rules to search for during the scan.
3. **Copyrights:** The user can configure ScanCode Toolkit to look for copyrights in the same scan, represented as a list for each scanned file.
4. **Custom license policy scanner:** ScanCode Toolkit can do more by defining custom rules to check during the scan.
5. **Contact Information and URLs:** It also looks for emails, URLs, and contact information or parties presenting it as a list.
6. **Missing meta-data:** ScanCode Toolkit reports missing notices and licenses texts inside the analyzed project.

[38][Accessed on 05.07.2020] https://scancode-toolkit.readthedocs.io/en/latest/contribute/roadmap.html

```
{
  "headers": [
    {
      "tool_name": "scancode-toolkit",
      "tool_version": "3.1.1",
      "options": {
        "input": [
          "."
        ],
        "--copyright": true,
        "--email": true,
        "--json-pp": "fossology-scancode-result.json",
        "--license": true,
        "--package": true,
        "--url": true
      },
      "notice": "Generated with ScanCode and provided on an \"AS IS\"
      BASIS, WITHOUT WARRANTIES\nOR CONDITIONS OF ANY KIND,
      either express or implied. No content created from\nScanCode
       should be considered or used as legal advice.
       Consult an Attorney\nfor any legal advice.\nScanCode is a free
           software
        code scanning tool from nexB Inc. and others.
        \nVisit https://github.com/nexB/scancode-toolkit/ for support
            and download.",
      "start_timestamp": "2020-07-09T090533.267094",
      "end_timestamp": "2020-07-09T101146.642187",
      "message": null,
      "errors": [],
      "extra_data": {
        "files_count": 4120
      }
    }
  ],
  "files": [{
    ...
        {
          "path": "fossology/Dockerfile",
          "type": "file",
          "licenses": [
            {
              "key": "fsf-ap",
              "score": 100.0,
              "name": "FSF All Permissive License",
              "short_name": "FSF All Permissive License",
              "category": "Permissive",
              "is_exception": false,
              "owner": "Free Software Foundation (FSF)",
              "homepage_url": "http://www.gnu.org/prep/maintain/
              html_node/License-Notices-for-Other-Files.html",
              "text_url": "",
              "reference_url": "https://enterprise.dejacode.com/
              urn/urn:dje:license:fsf-ap",
              "spdx_license_key": "FSFAP",
              "spdx_url": "https://spdx.org/licenses/FSFAP",
              "start_line": 5,
              "end_line": 8,
              "matched_rule": {
                "identifier": "fsf-ap.LICENSE",
                "license_expression": "fsf-ap",
                "licenses": [
                  "fsf-ap"
                ],
                "is_license_text": true,
                "is_license_notice": false,
                "is_license_reference": false,
                "is_license_tag": false,
                "matcher": "2-aho",
                "rule_length": 35,
                "matched_length": 35,
                "match_coverage": 100.0,
                "rule_relevance": 100
              }
            }
          ],
          "license_expressions": [
            "fsf-ap"
          ],
          "copyrights": [
            {
              "value": "Copyright Siemens AG 2016,
               fabio.huser@siemens.com",
              "start_line": 2,
              "end_line": 3
            },
            {
              "value": "Copyright TNG Technology Consulting
              GmbH 2016-2017, maximilian.huber@tngtech.com",
              "start_line": 2,
              "end_line": 3
            }
          ],
          "holders": [
            {
              "value": "Siemens AG",
              "start_line": 2,
              "end_line": 3
            },
            {
              "value": "TNG Technology Consulting GmbH",
              "start_line": 2,
              "end_line": 3
            }
          ],
          "authors": [],
          "packages": [],
          "emails": [
            {
              "email": "fabio.huser@siemens.com",
              "start_line": 2,
              "end_line": 2
            },
            {
              "email": "maximilian.huber@tngtech.com",
              "start_line": 3,
              "end_line": 3
            },
            {
              "email": "fossology@fossology.org",
              "start_line": 14,
              "end_line": 14
            }
          ],
          "urls": [],
          "scan_errors": []}
        },
        ...
  }],
```
Listing 5.15: ScanCode Toolkit scan JSON report sample.

```
# Document Information

SPDXVersion: SPDX-2.1
DataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
DocumentComment: <text>Generated with ScanCode and provided on
 an "AS IS" BASIS, WITHOUT WARRANTIES
OR CONDITIONS OF ANY KIND, either express or implied.
No content created from
ScanCode should be considered or used as legal advice.
Consult an Attorney for any legal advice.
ScanCode is a free software code scanning tool from
nexB Inc. and others.
Visit https://github.com/nexB/scancode-toolkit/ for support and
    download.
</text>

# Creation Info

Creator: Tool: scancode-toolkit 3.1.1
Created: 2020-07-09T11:51:08Z

# Package

PackageName: fossology
PackageDownloadLocation: NOASSERTION
PackageVerificationCode: da39a3ee5e6b4b0d3255bfef95601890afd80709
PackageLicenseDeclared: NOASSERTION
PackageLicenseConcluded: NOASSERTION
PackageLicenseInfoFromFiles: 0BSD
PackageLicenseInfoFromFiles: 389-exception
PackageLicenseInfoFromFiles: AAL
PackageLicenseInfoFromFiles: ADSL
PackageLicenseInfoFromFiles: AFL-1.1
PackageLicenseInfoFromFiles: AFL-1.2
PackageLicenseInfoFromFiles: AFL-2.0
PackageLicenseInfoFromFiles: AFL-2.1
PackageLicenseInfoFromFiles: AFL-3.0
PackageLicenseInfoFromFiles: AGPL-1.0-only
PackageLicenseInfoFromFiles: AGPL-1.0-or-later
PackageLicenseInfoFromFiles: AGPL-3.0-only
PackageLicenseInfoFromFiles: AGPL-3.0-or-later
...

#File

FileName: ./fossology/.travis.yml
FileChecksum: SHA1:
LicenseConcluded: NOASSERTION
LicenseInfoInFile: GPL-2.0-only
LicenseInfoInFile: LGPL-2.1-only
FileCopyrightText: <text>Copyright Siemens AG, 2014-2019</text>
...
# File

FileName: ./fossology/Dockerfile
FileChecksum: SHA1:
LicenseConcluded: NOASSERTION
LicenseInfoInFile: FSFAP
FileCopyrightText: <text>Copyright Siemens AG 2016, fabio.
    huser@siemens.com
Copyright TNG Technology Consulting GmbH 2016-2017,
    maximilian.huber@tngtech.com
</text>
...
# Extracted Licenses
...
LicenseID: LicenseRef-scancode-bsd-simplified-darwin
LicenseComment: <text>See details at https://github.com/nexB/
    scancode-toolkit/blob/develop/src/licensedcode/data/licenses/bsd-
    simplified-darwin.yml
</text>
ExtractedText: <text>See details at https://github.com/nexB/scancode
    -toolkit/blob/develop/src/licensedcode/data/licenses/bsd-simplified
    -darwin.yml
</text>
...
LicenseID: LicenseRef-scancode-cpl-0.5
LicenseComment: <text>See details at https://github.com/nexB/
    scancode-toolkit/blob/develop/src/licensedcode/data/licenses/cpl
    -0.5.yml
</text>
ExtractedText: <text>See details at https://github.com/nexB/scancode
    -toolkit/blob/develop/src/licensedcode/data/licenses/cpl-0.5.yml
</text>
```
Listing 5.16: ScanCode Toolkit SPDX tag value report sample.

6 Limitation

In this section, we discuss the four tests Yin (2018) mentioned to check the quality of the research method at various levels and the whole case study research.

6.0.1 Construct validity

Construct validity concerns that the case study design's correct operational measures have been followed to ensure the soundness of the studied concept. In this research, all of the gathered information is maintained by the studied products directly, pulling out the core models used, and analyzing it, establishing the chain of evidence. Besides using multiple data sources for each case as one of the recommended tactics by Yin (2018) to ensure high construct validity. The operation of combining the available documentation with the physical artifacts (source code) gave an integral whole to the cases, discovering the FLOSS dependencies model within.

6.0.2 Internal validity

This test seeks to establish a causal relationship, whereby certain conditions leads to another set of conditions, as distinguished from spurious relationships. Internal validity only applies to explanatory case studies. The used method in our case is multiple case study, focusing on analyzing the well-established products and how they model FLOSS dependencies. We don't address rival explanation or explanation building, but we use logical models and pattern matching. The possible threats to internal validity come from the entire manual process of analyzing the gathered data, which means research bias is potential in the cross-analysis phase. We grounded our reasons for this process in section 4.5.

6.0.3 External validity

External validity cares for the generalization of the research findings. This research is not based on quantitive methods; we can't claim statistical generalization to a large population where sampling applies. But we can affirm generalization with respect to the most similar cases and replication logic. Choosing well-established products for our research and using the cross-case analysis to compare these cases among each other, gives strong findings generalization.

6.0.4 Reliability

Reliability concern is the ability to repeat the followed processes in the research, leading to the same findings. We guarantee following the same procedures, processes, and strategies, leading to the same results, but what can change is the data and models, because the chosen products are continually developing. The repetition of these findings must consider the dates of gathering the data, including the documentation and the physical artifacts.

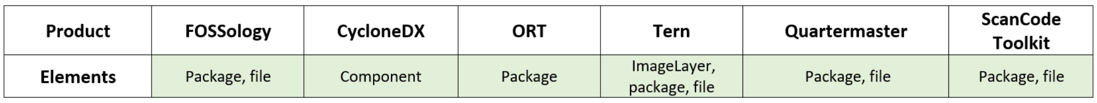
7 Conclusion

7.1 Discussion

This section will discuss (significant) findings of this thesis regarding our strategy questions and research questions.

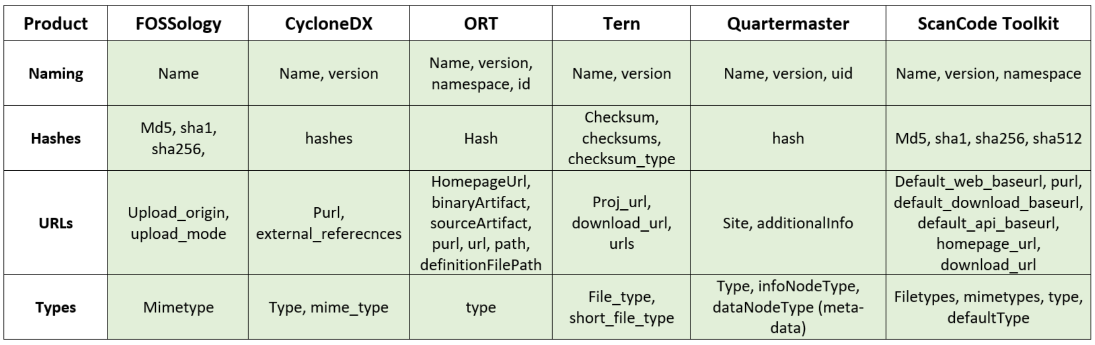
7.1.1 Component representation

In all cases, the component has different characteristics and shared aspects; see figure 7.2 shows all common elements across all UML diagrams of the cases. These aspects are naming, hashes, URLs, and types. Naming is always about the package name or the file name as separated entities and the packages' versions. Hashes utilized by these tools intensively to ensure that the long list of files and packages are unique to avoid duplication during the analysis, in some cases used for vulnerability checks, derived origin, or authenticity of the available source code. URLs have many varieties, which describes the analyzed project homepage, source code, download, or even binary URL. Types show how each case outlines the analyzed project type accurately on file or package level and mime types with different representations to add more details about each package or component.For instance, CycloneDX has eight classifications of types to describe the component, e.g. application, device, or firmware; you can also add a mime-type composed of any letter and numbers, see case 5.1 and figure 5.1. As for uncommon aspects of the component, some cases have more to attach to it, e.g. VCS information, description of the component, the analyzed project primary language, or keywords. The critical part of the component representation comparison is how each case represents the underlying model of a component. As for the elements refer to primary parts that compose the model of the component, e.g. package linked to alist of files compose the component from the product model perspective, see figure 7.1.



Product	FOSSology	CycloneDX	ORT	Tern	Quartermaster	ScanCode Toolkit
Elements	Package, file	Component	Package	ImageLayer, package, file	Package, file	Package, file

Figure 7.1: Component elements comparsion.

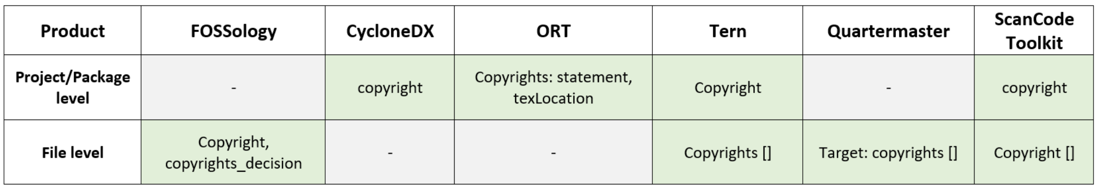


Product	FOSSology	CycloneDX	ORT	Tern	Quartermaster	ScanCode Toolkit
Naming	Name	Name, version	Name, version, namespace, id	Name, version	Name, version, uid	Name, version, namespace
Hashes	Md5, sha1, sha256,	hashes	Hash	Checksum, checksums, checksum_type	hash	Md5, sha1, sha256, sha512
URLs	Upload_origin, upload_mode	Purl, external_referecnces	HomepageUrl, binaryArtifact, sourceArtifact, purl, url, path, definitionFilePath	Proj_url, download_url, urls	Site, additionalInfo	Default_web_baseurl, purl, default_download_baseurl, default_api_baseurl, homepage_url, download_url
Types	Mimetype	Type, mime_type	type	File_type, short_file_type	Type, infoNodeType, dataNodeType (meta-data)	Filetypes, mimetypes, type, defaultType

Figure 7.2: Component common aspects.

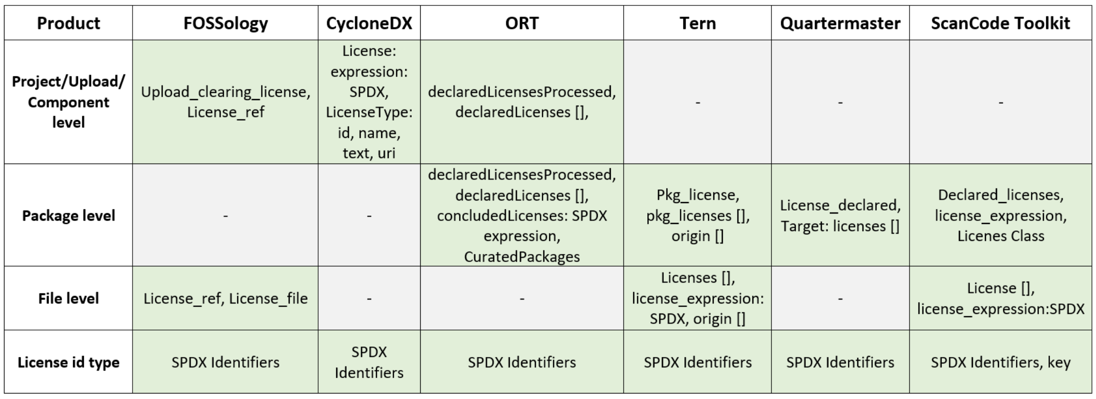
7.1.2 Licenses and copyrights representation

Since the cases consider the components on different levels as mentioned in component representation 7.1.1, it is required to slice every level individually to clearly see the similarities and differences. Regarding licenses, see figure 7.4. The primary levels are (project, upload, component) level, package level, and file level. Four of the cases ORT, Tern, Quartermaster, and ScanCode Toolkit consider the package declared licenses by the vendor, in addition to SPDX expression to express multiple licenses situation and list of all included licenses in the package. FOSSology, CycloneDX, and ORT consider overall license on (project, upload, component) level. FOSSology, Tern, and ScanCode Toolkit also acknowledge the licenses on file level by listing all existing licenses in the file with an SPDX expression to get a hold on the situation with multiple licenses. All the cases represent the licenses with a simple string or a list of strings containing the SPDX identifiers as value. ORT, ScanCode Toolkit, FOSSology, and CycloneDX model the license in their unique way to fit the needs of its logic and design, as for the other cases, the license is embedded in their model. Copyrights are considered only on two levels (project, package) level and file level; see figure 7.3. FOSSology, Tern, and Quartermaster recognize the copyrights on file level the others consider on a project, package level as for Quartermaster, and ScanCode Toolkit considers both. Tern has a separate representation of notices. In all the cases, copyrights are stored as string or list of strings, ORT, and FOSSology add the copyrights statements' location in the scanned files.



Product	FOSSology	CycloneDX	ORT	Tern	Quartermaster	ScanCode Toolkit
Project/Package level	-	copyright	Copyrights: statement, texLocation	Copyright	-	copyright
File level	Copyright, copyrights_decision	-	-	Copyrights []	Target: copyrights []	Copyright []

Figure 7.3: Copyrights common aspects.

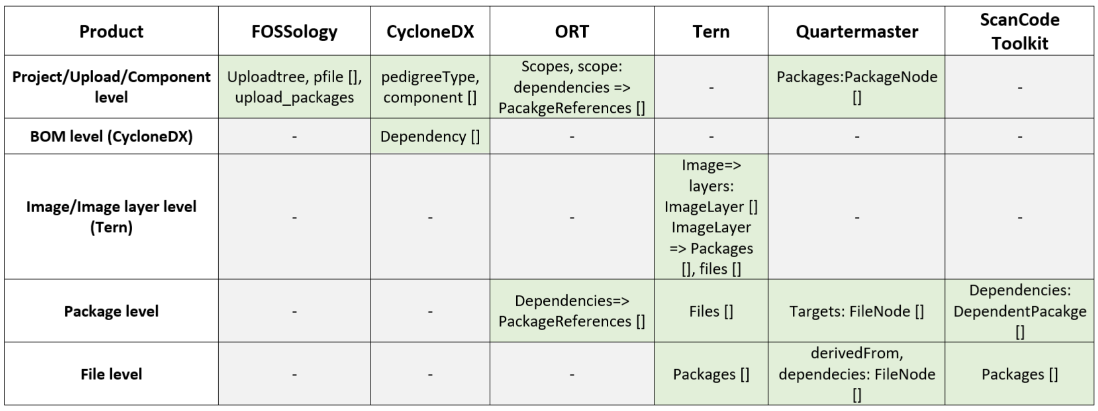


Product	FOSSology	CycloneDX	ORT	Tern	Quartermaster	ScanCode Toolkit
Project/Upload/ Component level	Upload_clearing_license, License_ref	License: expression: SPDX, LicenseType: id, name, text, uri	declaredLicensesProcessed, declaredLicenses [],	-	-	-
Package level	-	-	declaredLicensesProcessed, declaredLicenses [], concludedLicenses: SPDX expression, CuratedPackages	Pkg_license, pkg_licenses [], origin []	License_declared, Target: licenses []	Declared_licenses, license_expression, Licenes Class
File level	License_ref, License_file	-	-	Licenses [], license_expression: SPDX, origin []	-	License [], license_expression:SPDX
License id type	SPDX Identifiers	SPDX Identifiers	SPDX Identifiers	SPDX Identifiers	SPDX Identifiers	SPDX Identifiers, key

Figure 7.4: Licenses common aspects.

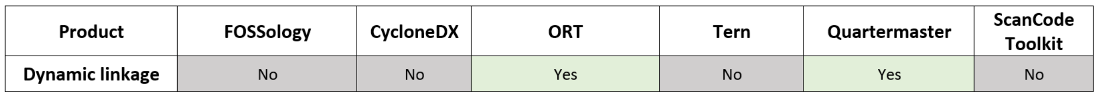
7.1.3 Dependency representation

As a reminder, the dependency is a reference to another component with linking information. Most cases allow the dependency representation on different levels; see figure 7.5. FOSSology represents dependency relationships only on the upload level. On the other hand, CycloneDX has multiple options to represent component to component relationship and older/parent versions through pedigree type. Tern, as a container license compliance tool, also has an extra level to describe dependent packages for each image layer. There is variation between all the cases, but it can be categorized to file level dependent Tern, Quartermaster, ScanCode Toolkit, and FOSSology and Package level dependent CycloneDX and ORT. The older scanners, e.g. FOSSology and ScanCode Toolkit, have no consideration for dynamically linked libraries. The only two products that consider the dynamically linked libraries are ORT and Quartermaster, with abilities to download and scan the remote library, then add all license compliance information to the final results, see figure 7.6. Only two of the products represent dependencies as a graph ORT and CycloneDX; the ORT analyzer uses the dependency graph as an intermediate result to export it to the other tools, e.g. reporter, scanner, or downloader. CycloneDX represents the dependency graph in the BOM as a tree; the older versions use dependency graph extension allowing the users to see how all packages are dependent on each other in a machine-readable format of your choice. Products like Quartermaster have a way to describe dependencies on file level. They can be represented as a tree, but no other additional uses for these data as a dependency tree or as a graph within the product itself.



Product	FOSSology	CycloneDX	ORT	Tern	Quartermaster	ScanCode Toolkit
Project/Upload/Component level	Uploadtree, pfile [], upload_packages	pedigreeType, component []	Scopes, scope: dependencies => PacakgeReferences []	-	Packages:PackageNode []	-
BOM level (CycloneDX)	-	Dependency []	-	-	-	-
Image/Image layer level (Tern)	-	-	-	Image=> layers: ImageLayer [] ImageLayer => Packages [], files []	-	-
Package level	-	-	Dependencies=> PackageReferences []	Files []	Targets: FileNode []	Dependencies: DependentPacakge []
File level	-	-	-	Packages []	derivedFrom, dependecies: FileNode []	Packages []

Figure 7.5: Dependency common aspects.

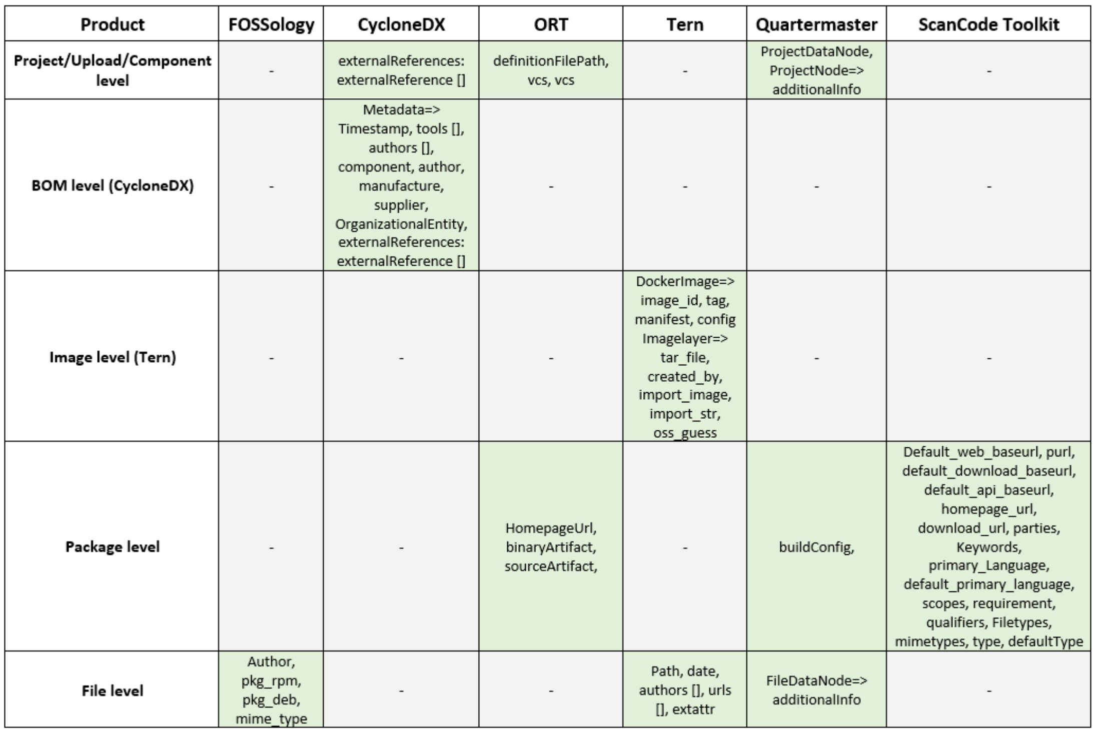


Product	FOSSology	CycloneDX	ORT	Tern	Quartermaster	ScanCode Toolkit
Dynamic linkage	No	No	Yes	No	Yes	No

Figure 7.6: Products with dynamic linkage.

7.1.4 Meta-data representation

Meta-data is defined as data on data or information on information, with time enigmatic of this definition is cleared, the refined definition appeared. Metadata is structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource by Riley (2017). All cases that have mandatory filed must be filled to store, analyze, and processes the analyzed project. These fields are necessary meta-data, also directly related to the primary use case license compliance. Still, all other fields that describe the analyzed project are additional meta-data. For example, case 5.6 gathers all possible meta-data during the analysis, e.g. URLs, keywords, and primary language, also see figure 7.7. Meta-data spread on different levels for each case, but what makes CycloneDX and Quartermaster unique, allowing analyzer or the user to add meta-data dynamically as much as needed, unlike the others, see figure 7.7. ORT is also unique because it connects to ClealyDefined to pull and push curated meta-data about the analyzed project. Tern pulls information from docker hub since it is specialized for container scans and its database. ScanCode Toolkit looks for meta-data that only provided in the analyzed project by the vendor; if the analyzed project is not well-maintained, it fails to deliver good up to date results. FOSSology uses RPM and Debian package to pull meta-data of the well-known packages.



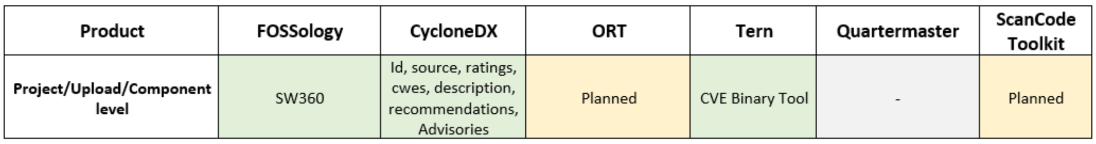
Product	FOSSology	CycloneDX	ORT	Tern	Quartermaster	ScanCode Toolkit
Project/Upload/Component level	-	externalReferences: externalReference []	definitionFilePath, vcs, vcs	-	ProjectDataNode, ProjectNode=> additionalInfo	-
BOM level (CycloneDX)	-	Metadata=> Timestamp, tools [], authors [], component, author, manufacture, supplier, OrganizationalEntity, externalReferences: externalReference []	-	-	-	-
Image level (Tern)	-	-	-	DockerImage=> image_id, tag, manifest, config Imagelayer=> tar_file, created_by, import_image, import_str, oss_guess	-	-
Package level	-	-	HomepageUrl, binaryArtifact, sourceArtifact,	-	buildConfig,	Default_web_baseurl, purl, default_download_baseurl, default_api_baseurl, homepage_url, download_url, parties, Keywords, primary_Language, default_primary_language, scopes, requirement, qualifiers, Filetypes, mimetypes, type, defaultType
File level	Author, pkg_rpm, pkg_deb, mime_type	-	-	Path, date, authors [], urls [], extattr	FileDataNode=> additionalInfo	-

Figure 7.7: Meta-data common aspects.

7.1.5 Vulnerabilities representation

Most of the project does not support vulnerability checks by default. Some use extensions, e.g. Tern and FOSSology, some have planned the feature, but it is not available yet, see figure 7.8. The only project that supports vulnerabilities built-in is CycloneDX v1.2, which was released 26.05.2020 during this thesis. FOSSology and Tern externalize the vulnerability check to extensions or another supply chain tool, so there is no clear standard linking these products. On the other hand, CycloneDX has ways to reference its vulnerability by CVE id, sources with a list of common weakness enumeration (CWE)[1] to distingue the vulnerability type, scores, rating, and recommendation to find the solution to this vulnerability.

[1] [Accessed-on 08.08.2020] https://cwe.mitre.org/



Product	FOSSology	CycloneDX	ORT	Tern	Quartermaster	ScanCode Toolkit
Project/Upload/Component level	SW360	Id, source, ratings, cwes, description, recommendations, Advisories	Planned	CVE Binary Tool	-	Planned

Figure 7.8: Vulnerabilities common aspect.

7.2 Summary

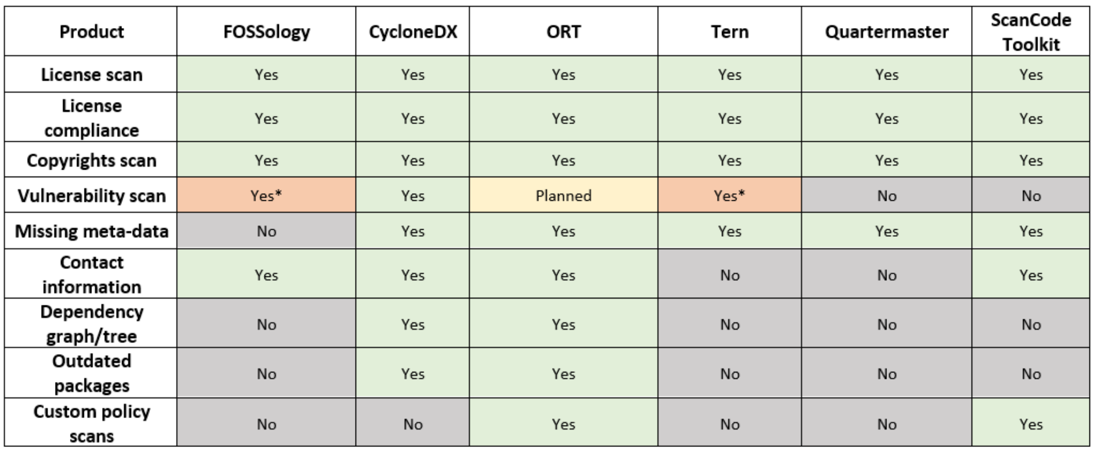
After we presented the results of our thematic analysis across the cases, pointing out the essential aspects. Answering the research questions follow it as a result.

RQ1: "What models for modeling FLOSS dependencies in products exist?"
The cases we discussed earlier and the common aspects show existing product models the dependency on a project and package levels like ORT and CycloneDX. FOSSology has a flat-level characteristic, which considers the dependency as a file relationship across all the analyzed project files. Tern has considered the dependency relationship across the package and file level with an extra consideration about each image layer of the container, which depends on a list of packages. ScanCode Toolkit has both modeled the dependency across two levels package and file. Quartermaster has an explicit representation for dependence and derived origin on a file level, besides the project level. The answer shows various ways to represent FLOSS dependency in the studied cases with critical similarities and differences.

RQ2: "What dimensions are considered to model FLOSS dependencies?"
As the discussion exhibits, everything spins around the analyzed project licenses in a query to find any compliance problems within the included packages. All the cases model the licenses in a significant way with a detailed sight to serve the primary concern of avoiding legal matters, leading us to these products' fundamental **dimensions**. Firstly, **licenses** shape the base model in all the cases, showing how the license details are managed, including the declared licenses by the vendor, the concluded license, multiple license scenarios, and the scan results. Secondly, the ability to share **reports in a machine-readable format** allows these products to integrate into the tools supply chain and other possible production lines. Thirdly, **copyrights** are a critical aspect of scanning projects considered by all the cases on different levels. Fourthly, **meta-data** filling the gaps, storing these forms of data, and keep the analyzed projects well maintained will enable products to provide more use cases on this matter, e.g. risk assessment. Finally, **vulnerabilities** are a crucial dimension, even though not all the instances of the study consider it built-in, provided ways to scan for it via extensions or other supply chain tools. In summary, all five dimensions shaps most

of the existing products' models; later on, new use cases and needs will increase these dimensions to provide a more automated way of deciding how to manage FLOSS dependencies in software.

RQ3: "What are the core use cases for existing modeling solutions?"
Of the identified nine use cases, we identified three use cases present in each case, see figure 7.9. These core use cases are license scan, license compliance, and copyrights. Five out of six cases allow the user to update the missing meta-data to help keep the SBOM and the analyzed project well-maintained, one of the crucial and common use cases among the products. Four out of six cases scan for contact information of copyright holders, maintainers, developers, and responsible parties is also a widespread use case among the products. One out of six cases has a vulnerability scan builtin and two out of six features using extensions or other supply chain tools Tern, and FOSSology with SW360. There are less common use cases with scores of two out of six, dependency graph/tree, outdated packages, and custom policy scans.



Product	FOSSology	CycloneDX	ORT	Tern	Quartermaster	ScanCode Toolkit
License scan	Yes	Yes	Yes	Yes	Yes	Yes
License compliance	Yes	Yes	Yes	Yes	Yes	Yes
Copyrights scan	Yes	Yes	Yes	Yes	Yes	Yes
Vulnerability scan	Yes*	Yes	Planned	Yes*	No	No
Missing meta-data	No	Yes	Yes	Yes	Yes	Yes
Contact information	Yes	Yes	Yes	No	No	Yes
Dependency graph/tree	No	Yes	Yes	No	No	No
Outdated packages	No	Yes	Yes	No	No	No
Custom policy scans	No	No	Yes	No	No	Yes

*: indicate that vulnerability scans done by extension or other supply chain tools.

Figure 7.9: Common core use cases.

The answer to the research questions are not the only contribution of this paper; it also provided a full-scale comparison between the features of the selected products with a detailed view of the potentials for each product and its use cases.

8 Future Work

After concluding our research by answering the research questions, presenting the possible future work is critical to make use of these findings.

In this thesis, the selected license compliance tools we discussed are widely used in a real-world context, not just theoretical concepts. Interviewing the communities or the companies behind these tools will allow us to perform data triangulation using multiple resources to collect the data, as Carter Nancy and J. (2014) described it. Also, performing theory triangulation by further analysis and interpretation of the data by Carter Nancy and J. (2014) will adjust the finding, giving more reliable answers to the research questions.

During the case selection 4.3 processes, we chose six candidates according to our case definition 4.2, but the list contains more than 25 other candidates; see table A. Conducting additional single or multiple case studies based on the listed products will increase this thesis's generalization. It is worth to mention this list also contains a lot of widely used license scanners standalone or as a utilized tool by other products.

Furthermore, our case definition 4.2 demonstrates the sharable reports in a machine-readable format as one of our six criteria, indicating that the products have an unlimited probability of building a supply chain of tools. Some of the selected products have already been a part of the supply chain of tools, e.g. CycloneDX with Dependency Track and FOSSology with SW360. Arising the questions, How the modeling of dependencies in products affected the supply chain tools? How the supply chain tools appropriated the imported data? Which use cases do these supply chain tools have?

We looked into the products based on the vendors' available documentation and physical artifacts; we investigated the defined use cases. Still, further investigation is required to ensure how these products operate in a real-world context. Interview companies and software businesses that use these products will identify the most critical use cases with valuable knowledge about the drawbacks, missing features, and functionalities. Companies will also provide vital information about these products and the FLOSS integration process into their production lines and products. Exhibiting how crucial license compliance is, maybe they will share some of the current challenges and problems.

Interviews will complement this thesis, provide more real-world knowledge and expertise to enhance the results with an additional case study research about more well-established license compliance tools will definitely improve the findings and generalization.

9 Acknowledgements

I would like to acknowledge Prof. Dr. Dirk Riehle and my supervisor Andreas Bauer as the second readers of this thesis. I am gratefully indebted to Sebastian Schmid for his precious feedback and comments on this thesis's research method.

Finally, I must express my very profound gratitude to my wife, my kids, and my mother for providing me with unfailing support and continuous encouragement throughout my years of absence to study in Germany and through the process of researching and writing this thesis.

This accomplishment would not have been possible without them. Thank you.

Appendix A   Case Study Candidates List

Project Name	Publisher	License	Version	Github	Website
CycloneDX	Apache	Apache-2.0	v1.1	https://github.com/CycloneDX	https://cyclonedx.org
Quartermaster	Endocode AG	GPL-3.0	v0.5.1	https://github.com/QMSTR	https://qmstr.org/documentation/
SW360	Eclipse	EPL-2.0	v8.2.0-M1	https://github.com/eclipse/sw360	https://projects.eclipse.org/proposals/sw360
OSS Review Toolkit	HERE	Apache-2.0	-	https://github.com/oss-review-toolkit/ort	
AboutCode Toolkit	nexB	Apache-2.0	v4.0.1	https://github.com/nexB/aboutcode-toolkit	https://aboutcode.readthedocs.io/en/latest/aboutcode-toolkit/home.html
FOSSA-CLI	Fossa	MPL-2.0	v1.0.27	https://github.com/fossas/fossa-cli/releases	https://fossa.com
OSS Discovery	OpenLogic	AGPL-3.0	v2.3.2	https://github.com/openlogic/ossdiscovery	http://ossdiscovery.sourceforge.net
Debian/copyright	Debian	-	v1.0		https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/#introduction
Apache Whisker	Apache	Apache-2.0	v0.1 SNAPSHOT		http://creadur.apache.org/whisker/
OSS Attribution Builder	Amazon	Apache-2.0	v0.9.0	https://github.com/amzn/oss-attribution-builder	
OSS Police	Ruian Duan,	GPL-3.0	-	https://github.com/osssanitizer/osspolice	
Tern	VMware	BSD-2-Clause	v2.0.0	https://github.com/tern-tools/tern	
SPDX	Linux Foundation	CCA-3.0	v2.2	https://github.com/spdx/spdx-spec	https://spdx.org
Augur	Augur Labs	MIT	v0.12.0	https://github.com/chaoss/augur	http://www.augurlabs.io
Fossology	Linux Foundation	GPL-2.0	v3.8.0	https://github.com/fossology/fossology	https://www.fossology.org
in-toto	NYU Secure Systems Lab	Apache-2.0	v0.4.2	https://github.com/in-toto/	https://in-toto.io
OWASP Dependency-Track	OWASP Foundation	Apache-2.0	v3.8.0	https://github.com/DependencyTrack/dependency-track	https://owasp.org
kernel-spdx-ids	Linux Foundation	Apache-2.0	-	https://github.com/swinslow/kernel-spdx-ids	
REUSE	Free Software Foundation Europe	Multi-Licenses	v0.10.0	https://github.com/fsfe/reuse-tool.git	https://reuse.software/
ScanCode Toolkit	nexB	Multi-Licenses	v3.0.2	https://github.com/nexB/scancode-toolkit	https://scancode-toolkit.readthedocs.io/en/latest/
Qualipso	-	-	-	http://qualipso.icmc.usp.br/	
npm-spdx	Linux Foundation	Apache-2.0	-	https://github.com/swinslow/npm-spdx	
SPDX Online Tool	Linux Foundation	Apache-2.0	-	https://github.com/spdx/spdx-online-tools	
License Checker LC	Ben Boyter	MIT	v1.3.1	https://github.com/boyter/lc	
Apache Rat	Apache	Apache-2.0	v0.13	https://github.com/apache/creadur-rat	https://creadur.apache.org/rat/#Who_Develops_Rat
Go-License-Detector	Source{d}	Apache-2.0	v3.1.0	https://github.com/src-d/go-license-detector	
Go-License-Discovery	Jfrog	Apache-2.0	-	https://github.com/jfrog/go-license-discovery	https://jfrog.com
License Classifier	Google	Apache-2.0	-	https://github.com/google/licenseclassifier	
License Check	Google	BSD 2-Clause Simplified	-	https://github.com/google/licensecheck	
LiD	Code Aurora Forum	BSD 3-Clause		https://github.com/codeauroraforum/lid	
NPM License Checker	DavGlass	BSD 3-Clause	v25.0.1	https://github.com/davglass/license-checker	
NPM License Crawler	M Wittig	BSD 3-Clause	V0.1.4	https://github.com/mwittig/npm-license-crawler/	
dpkg-licenses	Daniel Alder	GPL-3.0	-	https://github.com/daald/dpkg-licenses	
Askalono	Amazon	Apache-2.0	v0.4.2	https://github.com/amzn/askalono	
Black Duck Docker Inspector	Black Duck Software	Apache-2.0	v9.0.1	https://github.com/blackducksoftware/blackduck-docker-inspector	
Barista	Amazon	Apache-2.0	v0.4.2	https://github.com/Optum/barista	https://optum.github.io/barista/docs/architecture
Licensee	Ben Balter	MIT	v9.14.0	https://github.com/licensee/licensee	

Table 9.1: Case study candidates list here for full version.

Appendix B FOSSology ERD

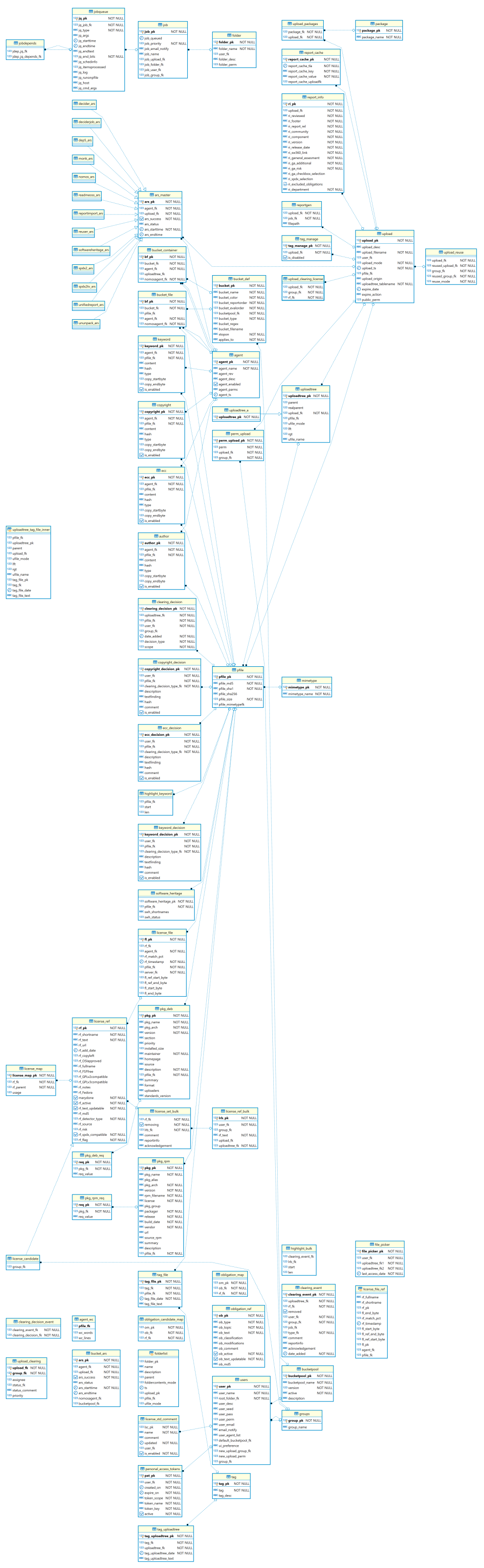


Figure 9.1: FOSSology v3.8.0 full ERD.

Appendix C Thematic Analysis Phases

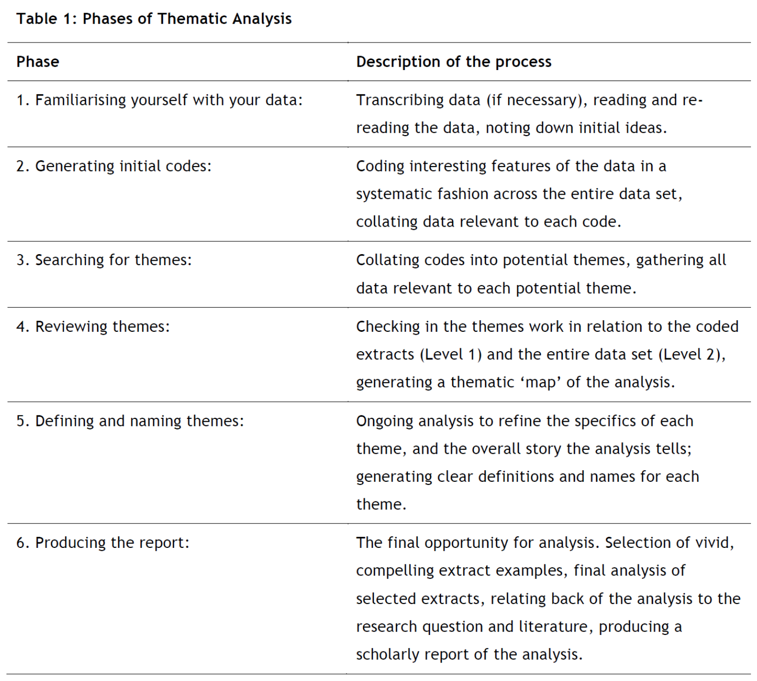


Table 1: Phases of Thematic Analysis

Phase	Description of the process
1. Familiarising yourself with your data:	Transcribing data (if necessary), reading and re-reading the data, noting down initial ideas.
2. Generating initial codes:	Coding interesting features of the data in a systematic fashion across the entire data set, collating data relevant to each code.
3. Searching for themes:	Collating codes into potential themes, gathering all data relevant to each potential theme.
4. Reviewing themes:	Checking in the themes work in relation to the coded extracts (Level 1) and the entire data set (Level 2), generating a thematic 'map' of the analysis.
5. Defining and naming themes:	Ongoing analysis to refine the specifics of each theme, and the overall story the analysis tells; generating clear definitions and names for each theme.
6. Producing the report:	The final opportunity for analysis. Selection of vivid, compelling extract examples, final analysis of selected extracts, relating back of the analysis to the research question and literature, producing a scholarly report of the analysis.

Figure 9.2: Thematic analysis phases by Braun and Clarke (2006).

Appendix D Clearly Defined UML

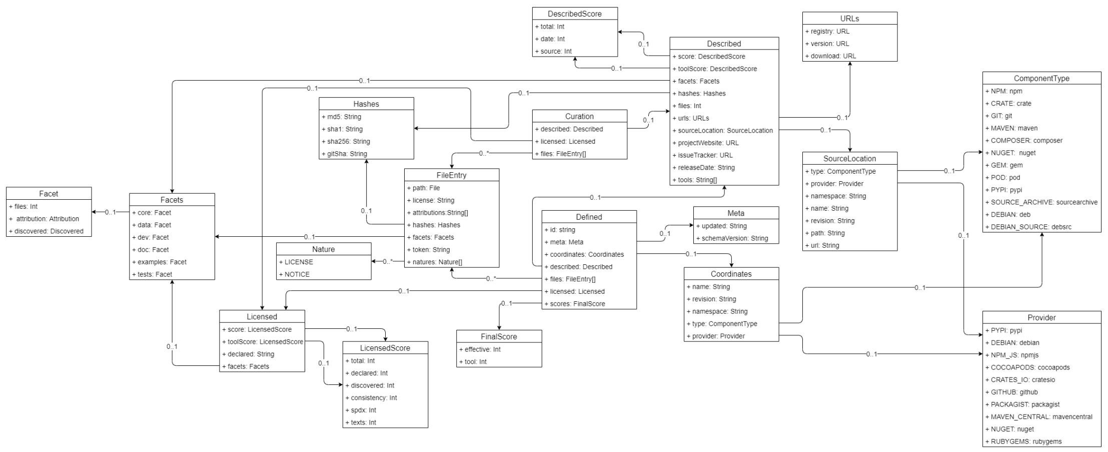


Figure 9.3: Clearly Defined UML[1].

Appendix E OSS Review Toolkit design

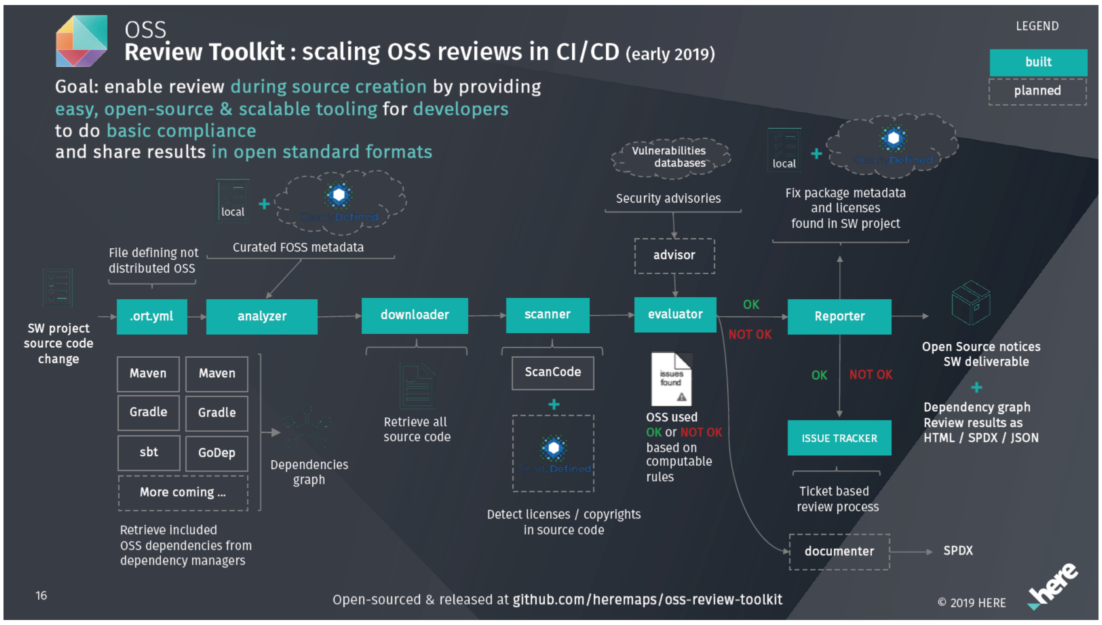

OSS
Review Toolkit : scaling OSS reviews in CI/CD (early 2019)
Goal: enable review during source creation by providing
easy, open-source & scalable tooling for developers
to do basic compliance
and share results in open standard formats
LEGEND
built
planned
Vulnerabilities databases
Security advisories
local
Fix package metadata and licenses found in SW project
Curated FOSS metadata
File defining not distributed OSS
advisor
SW project source code change
.ort.yml
analyzer
downloader
scanner
evaluator
OK
NOT OK
Reporter
Open Source notices SW deliverable
+
Dependency graph Review results as HTML / SPDX / JSON
Maven
Maven
Gradle
Gradle
sbt
GoDep
More coming ...
Retrieve included OSS dependencies from dependency managers
Dependencies graph
Retrieve all source code
ScanCode
Detect licenses / copyrights in source code
issues found
OSS used OK or NOT OK based on computable rules
OK
NOT OK
ISSUE TRACKER
Ticket based review process
documenter
SPDX
16
Open-sourced & released at github.com/heremaps/oss-review-toolkit
© 2019 HERE

Figure 9.4: OSS Review Toolkit design[2].

[1] Accessed-on[25.07.2020] https://github.com/oss-review-toolkit/ort/tree/master/clearly-defined/src/main/kotlin

[2] Accessed-on[24.09.2020] https://events19.linuxfoundation.org/wp-content/uploads/2018/07/Starting-and-scaling-an-Open-Source-Office-v2.pdf

References

Bauer, A., Harutyunyan, N., Riehle, D. & Schwarz, G.-D. (2020). Challenges of tracking and documenting open source dependencies in products: A case study. *IFIP International Conference on Open Source Systems*, 25–35.

Braun, V. & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology*, *3*(2), 77–101.

Brøndum, J. (2012). *Architectural dependency analysis and modelling* (Doctoral dissertation). University of New South Wales.

Carter Nancy, D. A. w. B. J., Bryant-Lukosius Denise & J., N. A. (2014). The use of triangulation in qualitative research. *Oncology nursing forum*, *41*(5), 545.

Hammouda, I., Mikkonen, T., Oksanen, V. & Jaaksi, A. (2010). Open source legality patterns: Architectural design decisions motivated by legal concerns. *Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments*, 207–214.

Jaeger, M. C., Fendt, O., Gobeille, R., Huber, M., Najjar, J., Stewart, K., Weber, S. & Wurl, A. (2017). The fossology project: 10 years of license scanning. *IFOSS L. Rev.*, *9*, 9.

Luoto, A. (2013). A uml profile approach to managing open source software licensing.

Mayring, P. (2014). Qualitative content analysis: Theoretical foundation, basic procedures and software solution.

Riley, J. (2017). Understanding metadata. *Washington DC, United States: National Information Standards Organization (http://www. niso. org/publications/press/UnderstandingMetadata. pdf)*, 23.

Rozanski, N. & Woods, E. (2012). *Software systems architecture: Working with stakeholders using viewpoints and perspectives.* Addison-Wesley.

Sangal, N., Jordan, E., Sinha, V. & Jackson, D. (2005). Using dependency models to manage complex software architecture. *Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 167–176.

Seawright, J. & Gerring, J. (2008). Case selection techniques in case study research: A menu of qualitative and quantitative options. *Political research quarterly*, *61*(2), 294–308.

Springett, S. (2020a). *Cyclonedx use cases* [accessed on 30.05.2020]. https://cyclonedx.org/use-cases

Springett, S. (2020b). *Cyclonedx xml reference external references* [accessed on 28.05.2020]. https://cyclonedx.org/docs/1.2/#type_externalReferences

Yin, R. K. (2018). Case study research and applications. *Design and methods*, *6*.