What is the state of open source license clarity?

Summary

The mission of the ClearlyDefined project is to help free and open source software (FOSS) projects to be more successful through clearly defined project data. The initial focus for ClearlyDefined is licensing. Even if a FOSS project has published license information, that information may not be complete or clear enough for users of the code to easily comply with license obligations without significant research. The same applies to open source projects themselves: they need to understand the other FOSS code they include directly and indirectly either as embedded code or as external dependencies.

One of the first steps to support ClearlyDefined licensing for FOSS projects was to define a set of license clarity metrics (the "Licensed Score") and test them against a representative set of FOSS projects. This article provides the results from analyzing the Licensed Score of approximately 5000 of the most popular FOSS packages.

A summary of our conclusions from this analysis is:

- The "Licensed Score" overall is low. We expected higher license clarity scores for these popular projects.
- One size does not fit all: FOSS licensing documentation practices vary widely across programming languages and software package types and communities.
- We have set rather strict clarity scoring criteria that will need to evolve to account for the specialized conventions for each package type (Maven, Rubygems, npm, Pypi, etc.).

There are many caveats of course and low scores may be due to:

- bugs in the code we crafted to collect the data or compute scores,
- our lack of proper understanding of how different communities capture licensing,
- or other factors such as variations in the size of packages that we do not yet take into account. Some small npm packages may contain only a single JavaScript file while at least one Pypi Python package contained over 138,000 files (we removed it for now from the data set). Several packages still have a few thousand files. In such cases the comparison across package types may not be fair.

As we iron out bugs and learn more about the domain we expect scores to improve and be more accurate over time.

In spite of the low Licensed Scores reported from this analysis, we believe that open source software authors do care about licensing, but they are often not aware of best practices for license clarity. The npm community is a good example of how quickly an open source community can improve license clarity by combining public visibility, structured documentation, and author feedback/nudging (see https://npm1k.org).

How clear is the license of a FOSS project?

The ClearlyDefined Project has designed a Licensed Score metric to measure this. This report documents the results from an evaluation of the license clarity of about 5000 of the most popular open source packages. Some of the data confirmed our intuitions about the current state of license clarity, but we also made some surprising discoveries along the way.

Licensing is an essential element of free and open source software. License clarity matters because a software package may not be reusable without clear license terms. Without license information, a FOSS package is not really open source at all because you do not have any permission to use it without a license or evidence that the package is in the public domain. Therefore it is important to get access to clear licensing information for any FOSS package.

What does it mean for a package to be "ClearlyLicensed" ? The ClearlyDefined project has drafted a set of criteria for computing a “Licensed Score” to measure the license clarity for software projects and their distribution packages. The scoring system is based on the following weighted criteria with a maximum possible score of 100:

1. Declared - The license is documented at top-level or well-known locations in a project repo or distributed package using obvious, non-ambiguous conventions such as a package manifest, README, etc.
2. Discovered - The proportion of code files that have a license and copyright notice.
3. Consistency - The Declared licensing is consistent and aligned with the file-level Discovered license notices.
4. SPDX - The license(s) in use are included in the SPDX License List: this is a proxy to test if the licenses in use are common licenses - see https://spdx.org/licenses/.
5. License Texts - The full license text of all the licenses in use is included in the package.

See Table 5. below for information about the weighting of these criteria.

These criteria are trying to capture the essence of license clarity, and conversely what makes the license documentation of a software package ambiguous (and of course whether there is license documentation available or not).

Let’s look at what these criteria mean with a few examples:

1. The simple case is when there is no license information: you get a zero score failing all the criteria.
2. At the other end of the scale, we have a project that provides evident license information at the root of the codetree and is using common SPDX-listed licenses. Its files carry copyright and license notices that are consistent with the Declared licenses. The texts of all license in use are included. License-wise, things are crystal clear and you get a 100/100 score.
3. A JavaScript library is composed of a single JavaScript file with this comment ‘License: MIT‘. This is the Declared license. And since this is a single file, this is of course also Discovered and Consistent. This is an SPDX-listed license but we do not have the license text and there are many variations for this license text: it does not fulfil the License Texts criteria. There is some ambiguity that exists about which exact variant of the MIT license text may be in use and this text may be needed to comply with the license requirements. Hence this library would not get a maximum score of 100.
4. A Python package “manifest” (typically a setup.py file) documents a license in an attribute as Apache-2.0: this is the Declared license. But it also contains files with other Discovered licenses: some files are using a BSD license which is not Consistent with the Declared license. Therefore it would not get credits for Consistency.

License clarity of Popular Packages

To evaluate the current state of open source license clarity, we selected approximately 5,000 popular packages from five prominent application package managers: Maven, Pypi, Rubygems, npm and NuGet. We collected

license data for these packages and calculated a license clarity score for each one. See the Methodology section below for details about our process.

Table.1 Tallies for the score and each scoring element by package type						
package type	**gem**	**maven**	**npm**	**nuget**	**pypi**	**avg / total**
score median	60	27	60	30	60	45
score average	52.3	25.1	59.1	23.6	54.7	43.1
declared	987	100	968	634	990	3679
discovered average	3.2	44.4	8.7	1.1	17.2	14.5
consistency	95	25	147	4	347	618
spdx	828	615	889	248	708	3288
full text	164	25	203	1	261	654
package counts	**998**	**925**	**970**	**1000**	**999**	**4892**

First, the overall license clarity of these 5K packages is rather low: the overall median and average scores are about 45/100. (Table 1.) Only 194 of our 4,892 packages have a license clarity score of at least 80 (out of 100 points) (Table 4.). This is not a big surprise because we started the ClearlyDefined project to help improve this situation, but it is critical to get empirical data to document and evaluate the current state of license clarity. On the bright side, about 75% of these packages have a “Declared” license which is encouraging, but having such a declaration may not be helpful because of the overall lack of consistency with file-level license notices.

Table 2. Scoring elements percentage by package type						
package type	gem	maven	npm	nuget	pypi	averagel
score average	52.3	25.1	59.1	*23.6*	54.7	43.1
percentage with declared	98.9	*10.8*	**99.8**	63.4	99.1	75.2
discovered average	*3.2*	**44.4**	8.7	1.1	17.2	14.5
percentage with consistency	9.5	2.7	15.2	*0.4*	**34.7**	12.6
percentage with spdx	83.0	66.5	**91.6**	*24.8*	70.9	67.2
percentage with full text	16.4	2.7	20.9	*0.1*	**26.1**	13.4

Table 3. Number of packages by license score brackets and by type						
score bracket	gem	maven	npm	nuget	pypi	total
0	6	227	2	338	8	581
1 to 9	0	8	0	2	0	10
10 to 19	4	126	0	13	1	144
20 to 29	0	113	0	8	0	121
30 to 39	149	341	64	400	231	1185
40 to 49	294	37	151	211	212	905
50 to 59	11	30	17	6	43	107
60 to 69	437	16	582	22	223	1280
70 to 79	76	9	84	0	196	365
80 to 89	18	16	36	0	70	140
90 to 99	3	2	17	0	15	37
100	0	0	17	0	0	17
package counts	998	925	970	1000	999	4892

The criteria we defined for the Licensed Score are rather strict: four of the five scoring elements are binary (winner-take-all elements) which means that a Licensed Score can degrade fairly quickly. We think that these elements represent fairly well if the licensing documentation is obvious and clear or if it requires further review to understand and determine what are the license terms of an open source package. Practically, when a license score is under 80/100, the license documentation likely requires some additional review, research and analysis.

Table 4. Number of packages over a score by type						
package type	gem	maven	npm	nuget	pypi	total
over 40	839	110	904	239	759	2851
over 50	545	73	753	28	547	1946
over 60	534	43	736	22	504	1839
over 70	97	27	154	0	281	559
over 80	21	18	70	0	85	194

There are only a few (17) packages with a “perfect” score of 100. All of these are npm packages. This and the overall strong standing of npm packages with the highest average score may be explained by the fact that npm is

newer than other package types included in this study and by some important characteristics of the npm format and community:

- npm packages are usually small (contain fewer and smaller files),
- the npm tool displays a warning when a license tag is missing,
- license tags should use normalized SPDX licenses, and
- there was a campaign to document which of the most popular 1000 npm packages were missing a license.

These factors probably encouraged npm package authors to provide better license documentation (See also http://npm1k.org/). We think that the explicit actions taken by the maintainers of the npm tools and the npm community to provide some feedback on license clarity has contributed to the overall npm good scores.

For the 559 packages with a score of 70/100 or more, Pypi comes on top, followed by npm and Rubygems. Yet NuGet and Maven are far behind.

- The highest scoring NuGet packages are in the 60 to 69/100 bracket (Table 4.) and this is the lowest of all package types. There are possibly two reasons for this: a NuGet package payload is primarily a set of compiled binaries without many ways to add license details. The .nuspec file format historically has had little support to document licenses, though recently they have introduced some improvements. Until these changes are adopted by the NuGet package authors, most packages only have a license URL to document licensing. This is a weak license clue for automated discovery and a URL can change or not be available in the future.
- Ruby programmers seldom provide file-level comments and less so than Python and JavaScript programmers.
- npm packages more consistently provide a declared license and use common SPDX-listed licenses.
- Pypi packages more consistently provide a license text and align their declared vs. discovered licenses.
- Maven JAR packages have the highest discovered average and the lowest declared average. They typically do not contain a package manifest (POM file) and extra documentation or files beyond the code itself. We selected the source JAR packages to get actual source code vs. binaries but Java JAR packages commonly miss key, top level licensing information.

Now let's now look at each scoring element separately and drill down by package type:

1. Declared - Clearly defined top-level, declared license
2. Discovered - License and copyright are available at the file level
3. Consistency - License consistency between file- and top-level
4. SPDX - License is from the SPDX License List
5. License texts - License Texts are provided

Declared Licenses

Only 100 of 925 (about 11%) Maven JAR packages are reported with a top-level Declared License. Maven JAR packages seldom contain much meta information: the POM manifest file is not often included and the META-INF rarely has much data. 634 of 1000 (about 24%) NuGet packages have such license information which is the next lowest. This is likely a problem with the NuGet license documentation conventions where the .nuspec format only had support for an optional license URL until recently. In contrast almost all npm, Rubygems and Pypi packages have a Declared License.

Discovered Licenses

Table 5. Number of packages by percentage of discovered license brackets and by type

label	gem	maven	npm	nuget	pypi	total
0	808	334	770	939	454	3305
1 to 9	109	44	56	18	189	416
10 to 19	26	13	20	24	74	157
20 to 29	15	14	22	3	51	105
30 to 39	11	33	8	10	39	101
40 to 49	11	12	3	3	28	57
50 to 59	6	51	26	3	48	134
60 to 69	2	53	13	0	37	105
70 to 79	3	60	8	0	24	95
80 to 89	4	121	11	0	30	166
90 to 99	3	185	7	0	24	219
100	0	5	26	0	1	32
package counts	998	925	970	1000	999	4892

This is the scoring element with the lowest scores overall: the documentation of a license at the file level seems poor. There are 3,305 of 4,892 packages (about 67%) where no file contain a detected license and copyright notice. When looking at the breakdown by package type we can see significant differences:

- Rubygems. Anecdotally, Ruby programs seem to contain fewer code comments than for most other languages and this seems to extend to license notices that are usually comments in the code. This lack of comments is likely a trait of the Rubyist programmer community and therefore not something that will change easily.
- NuGet packages comprise mostly compiled binaries and have a low score there.

- In contrast, Maven has the highest average file-level license coverage with an average of 44% (Table 2.). There could be several explanations: the original Java codebase from Sun with many file-level license and copyright notices may have influenced the community; or Java developers have long used advanced code editors (IDEs) with comment generation features that can inject a license notice header comment in each newly created file.

License Consistency

The low scores for consistency between Declared (top-level) and Discovered (file-level) license documentation are worrisome. There could be many reasons for this. First, this is a derived scoring element that does not kick in if there is no declared license or no file-level license. Also, documentation, example and build scripts may be using other licenses that the Declared ones: this is not taken yet into account. As a result, there are only a few NuGet and Maven packages that consistently document both the top-level and file-level licenses. In contrast there are 147 out of 970 npm packages and 347 out of 999 Pypi packages with consistent top-level and file-level license documentation. Inconsistency between Declared and Discovered licensing is one of the key problem areas for license clarity: if the existing license is not well documented as a top level declaration, this Declared license is less effective and useful as further investigation is needed to uncover the actual licenses in use.

SPDX Licenses

Beside many Declared licenses, the other bright spot is the relatively high number of packages that use SPDX-listed licenses. About 3,288 of 4,892 (~67%) packages use exclusively licenses that are listed in the SPDX License List. Yet, some common licenses, such as simple public domain dedications, are not included in the SPDX License List so the actual license clarity may be better than the score we compute. The npm community scores high as they have adopted SPDX license expressions to document licenses in a package.json file. Rubygems comes second: they encourage using SPDX license ids, but not expressions yet.

License texts

The scores for providing a full license text are aligned with the other results: Maven and NuGet score low while Pypi makes the best showing followed by npm and Rubygems. This scoring element is important as a common requirement of most simple and permissive licenses such as the MIT license is to include the license text with the source and/or binaries.

Licenses Score Calculation

The current Licensed Score calculation is based on the following Weight and Style for the 5 scoring elements. The “binary” scoring Style means Yes or No. The “progressive” scoring Style means that the score is calculated based on the percentage of files that have such a given attribute.

Scoring element	**Weight**	**Style**
Clearly defined top-level, declared license	30	binary
File-level license and copyright	25	progressive
License Consistency	15	binary
SPDX Standard Licenses	15	binary
License Texts	15	binary

Table 5. Table of scoring elements weights and styles.

Improving License Clarity

A primary goal for ClearlyDefined is to help FOSS projects improve the clarity of licensing for their code. The Licensed Score is intended to be an incentive for improvement. Given the weighting of our scoring elements, the easiest way to improve the Licensed Score of a package is to provide a top level Declared License, typically in a package manifest, NOTICE, LICENSE, COPYING or README file.

The next most obvious area for improvement is to add file-level license and copyright notices to all files in a software package. As a progressive scoring element, this means that you get some improvement even if you add this to only a few files. A great approach to do this is to follow the https://reuse.software/ guidelines published by the Free Software Foundation Europe; or to check the (new) way implemented by the Linux kernel maintainers (https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/process/license-rules.rst).

Both approaches build on the SPDX License Identifier conventions and the SPDX License List. They provide a good way to state a file-level license in a brief, clear and concise manner that is as equally easy to read (by people) and to parse with code. It is also a convention that can be supported by automated processes relatively easily.

Yet, while desirable, adding file-level notices may not fit the conventions of some programming communities (e.g. Rubygems) and therefore we may need to adapt the scoring in the future to better adapt to the ways of each package community.

Another easy improvement is to ensure that the license text is present in a package. A top level declaration that a package is using a BSD license is a good start, but this can be ambiguous because there are so many variations of the BSD licenses. Providing both a precise SPDX License Identifier and the full license text in a package is always a good idea for clarity and removes any licensing ambiguities.

Next steps

The score computation has already been integrated in ClearlyDefined server-side computation and UI code but not all packages used here have an updated score yet. It would probably be very useful to define a set of the most popular FOSS projects and packages to continuously monitor over time - a bit like a stock index - to track whether and how Licensed Score is improving over time. The set of packages used in this analysis is reasonable for a first pass at calculating Licensed Score by package type.

A Licensed Score is available for every software package processed in ClearlyDefined and is listed in the web UI in two variants

- The "tools" score which is the score computed based on the data collected from tools, and
- The "effective" score which is the score computed as part of the curation process after curation of the licensing data.

The scoring also provides a guideline to support the ongoing process to select those components that most need curation and review.

Finally there are clear differences in Licensed Score between package types based on a combination of factors including:

- How and where the package format provides a place for license information,
- Conventions in the community for documenting licenses for a project and for the preferred type of license - e.g. Permissive or Copyleft, and
- Initiatives within a package community to improve license clarity.

Therefore outreach from the ClearlyDefined project should be specific to each package community. Based on observation of the successful improvement of npm licensing practices in some areas, a promising approach would be to reach out to the package management tool authors and public repository maintainers to encourage them to provide positive feedback to package authors about the clarity of their license information. The immediate feedback that these tools can provide could have a big impact on a whole programming language and package ecosystem. This can be reinforced by adopting a structured approach to document licenses (e.g. SPDX-style expressions) to use in package manifests files and publishing a web-based leader board of popular packages with poor license clarity as done in the npm community.

Details about software package types

We have talked in this article about "packages", "package managers", manifests and repositories. A package is an installable software archive that contains installable software code (either source, binary or mixed) and sometimes contains data about itself (often in what is called a package manifest). Package archives are made available through internet or web-based "repositories" that provide an index of available packages and support the download of package archives and package manifests. There are also "package manager" tools that can help you to download and install package archives, and interpret the package manifest format and data.

Note that each programming language has produced its own unique package type, archive format, manager, repository, manifest format, etc. even though the languages are similar and provide similar services using similar data. Each package type and manager has its own unique conventions to handle documentation of the licensing for a package.

For instance, Node.js packages are primarily written in JavaScript. They are also called Node modules or npm packages. We use "npm" as a package type. npm is also the name of primary package management command line tool to install Node modules. Node modules are distributed through repositories called registries that are web-based servers. The primary public registry is at npmjs.com, and we call this the npm registry.

Maven
JAR packages uploaded to the Maven Central repository do not always contain a package manifest (the Maven "pom.xml"). This is not required by Maven and even though there is a convention to include this manifest in the META-INF subdirectory of a JAR, relatively few packages do it. We counted only 159 packages out of a thousand that contained a pom.xml manifest. As a result, there are only a few Maven packages that contain a structured license declaration. The pom.xml file that applies to a binary package may be available separately from the Maven repository or the corresponding source code repository, but that will typically remain separate from the code and is therefore not very helpful for clarity license documentation of a package archive.

Furthermore, JAR packages are typically consumed as compiled binaries that contain mostly .class files without the license files that may exist in the project source codebase (such as a git repo). Even source code JAR packages seldom contain more than the source code used to compile .class files. Some JAR packages include their license text in the META-INF directory, but this is rare. To address some of these issues for this study, we analyzed only "source" JARs in order to have at least some license information when this is present in the source code files.

Note that the POM of a JAR uploaded and released on Maven Central is now required to have license information as explained here https://maven.apache.org/repository/guide-central-repository-upload.html and here https://central.sonatype.org/pages/requirements.html.

RubyGems
A RubyGems ".gem" archive package is guaranteed to contain a YAML-structured metadata file derived from the ".gemspec" manifest found in the source code repository. However, declaring a license tag in a .gemspec file is not required, and such a license tag is not fully specified, leading to ambiguities. A license tag is a list of licenses strings, but there is no indication in the RubyGems specification that defines the relationship between multiple licenses when there is more than one. Do all licenses apply or is there a choice of licenses?

The license scanner (ScanCode toolkit) assumes by default that all the licenses apply (a conjunction of multiple licenses). This is not likely to be true in all cases, but short of clarity in the RubyGems specification or other

contextual information available in a given package, it is the safest assumption to make. This situation does not have a direct impact on the license clarity score, though it could lower the score of a RubyGems package with multiple licenses in the future. The very simple license expression syntax adopted by the SPDX community would be a great improvement here.

Pypi
Python packages come in various forms. The two primary forms are the built "wheel" archive and the source distribution archive. Because not all packages are available as wheels, we have used source distributions instead of wheels. These source distribution archives (tarballs) are created from the development source code files and typically contain a package manifest and other documentation files. The manifest is typically a "setup.py" Python file that contains some structured license information. No license information is required to release publicly a package on Pypi but when using the "qualifiers" to document licensing, the license must be picked in a list (https://pypi.org/pypi?%3Aaction=list_classifiers) Note also that the wheels packaging does not automatically include a license text that may be present in the source code checkout: this may lead to lower presence of License Text in the built archives.

npm
An npm JavaScript package archive always contains a "package.json" manifest that supports a license tag. The license tag is not mandatory when uploading a package, but npm does report a warning when a package.json manifest has no license information. When present, the license should be an SPDX license expression, but there are some exceptions. As a result, npm packages more often have better defined top-level declared licensing than other packages that do not enforce or encourage a standard to document licensing information.

NuGet
A NuGet package archive (for .NET) always contains a ".nuspec" manifest file. This manifest only supported a license URL to document a license until recently, which is rather limited. The format has been updated and will include SPDX license expressions (See https://github.com/NuGet/Home/wiki/Packaging-License-within-the-nupkg-(Technical-Spec) and https://github.com/NuGet/Home/wiki/Packaging-License-within-the-nupkg#spdx-expressions)
NuGet packages usually contain mostly compiled binaries, but they can include additional text files (such as license texts).

Methodology

The selection of 1000 popular packages for each package type was more art than science. In some cases, a public package repository provides some indication of popularity such as a number of downloads. We used this number even though that may be a weak indicator of the real popularity for a package. In other cases (such as Maven Central) there is little information available. We tried then to check if a package manifest contained some other clues (such as a reference to GitHub repo) and used the stars and fork counts on the corresponding repo as a proxy for popularity.

Computing the Scores

We computed the license clarity scores as designed and specified at https://github.com/clearlydefined/license-score/blob/master/ClearlyLicensedMetrics.md

We collected all the data details in the files available at https://github.com/clearlydefined/license-score/tree/master/scoring

This repository also contains instructions and a Python script to reproduce the computations used in this article.

The score computations have been integrated into the ClearlyDefined server and UI code. When browsing component definition details at https://clearlydefined.io/definitions there is a score badge that comes with two "licensed" scores: the base score as computed on the base package and the "effective" score which a "curated" score that is computed on the updated package if it has been curated and reviewed.

Credits

Many thanks the **ClearlyDefined** project contributors for their review and feedback and in particular Jeff McAffer, Carol Smith, Steven Esser, Luis Villa, Dennis Clark and Nicolas Schifano.