License Incompatibilities in Software Ecosystems

Rolf-Helge Pfeiffer
IT University of Copenhagen
Copenhagen, Denmark
ropf@itu.dk

Abstract—Contemporary software is characterized by reuse of components that are declared as dependencies and that are received from package managers/registries, such as, NPM, PyPI, RubyGems, Maven Central, etc. Direct and indirect dependency relations often form opaque dependency networks, that sometimes lead to conflicting software licenses within these.

In this paper, we study license use and license incompatibilities between all components from seven package registries (Cargo, Maven, NPM, NuGet, Packagist, PyPI, RubyGems) with a closer investigation of license incompatibilities caused by the GNU Affero General Public License (AGPL).

We find that the relative amount of used licenses vary between ecosystems (permissive licenses such as MIT and Apache are most frequent), that the number of direct license incompatibilities ranges from low 2.3% in Cargo to a large 20.8% in PyPI, that only a low amount of direct license incompatibilities are caused by AGPL licenses (max. 0.04% in PyPI), but that a whopping 6.62% of Maven packages are violating the AGPL license of an indirect dependency. Our results suggest that it is not too unlikely that applications that are reusing packages from PyPI or Maven are confronted with license incompatibilities that could mean that applications would have to be open-sourced on distribution (PyPI) or as soon as they are publicly available as web-applications (Maven).

I. INTRODUCTION

Denmark is one of the most digitized countries in Europe [1]. After implementation of the 2011 Danish e-government strategy [2], citizens interact with the government mostly through e-government systems, such as, self-service web-applications and websites. Examples for such systems are sundhed.dk (health.dk) or borger.dk (citizen.dk). These e-government systems are usually implemented as closed-source systems by private contractors and often in a combination of C#, Java, JavaScript, etc.

Besides application of modern programming languages, contemporary software development relies to a large degree on code reuse. Software is usually built with the help of reusable components, which we call packages in this paper. Programming language ecosystem emerge usually around specific package managers, which download and setup needed dependencies from remote package registries, such as, NPM, Maven Central, PyPI, RubyGems, etc.

Code reuse via dependency on external packages is also encouraged by software quality models and tools, such as, the SIG/TÜV IT Evaluation Criteria for Trusted Product Maintainability [3]. It recommends using “libraries and frameworks over ‘homegrown’ implementations of standard functionality” [4] for example to keep the size of software under own maintenance small.

However the advantages of code reuse, e.g., increased software quality or developer productivity [5–7] are attended by drawbacks, such as, increased maintenance cost or legal issues [8–11].

This paper focuses on license incompatibilities, which are examples of such legal issues. For instance, in March 2020 it was reported that 577.148 software projects which are build on top of Ruby on Rails (rails) face license issues. An indirect dependency of the web-application framework rails (see mimemagic in Fig. 1) switched its license from the permissive MIT to the more restrictive GPL-2.0. The restrictive strong copyleft GNU General Public License (GPL) is viral in that all work that is distributed and that includes code under GPL has to adopt the same –or a compatible– license. Consequently, all dependents of mimemagic, i.e., rails and all applications build with it, would have to either switch to GPL too or they would all have to replace the indirect dependency mimemagic, with a suitably licensed package. Such a small GPL licensed package hidden in a dependency network would pose a legal issue for commercial vendors that create and ship software on top of rails. Usually they are not interested in open-sourcing their products on distribution, which is a requirement of the GPL. The younger AGPL is even more restrictive on when to open-source software that reuses correspondingly licensed packages. It prescribes that software has to be made publicly available when it accessed via a network and when it relies on AGPL licensed components. Consequently, if one of Denmark’s online e-government systems had a direct or indirect dependency to an AGPL licensed package, either it would have to be open-sourced or the respective dependency would have to be replaced with software under a compatible license.

To understand to which degree license incompatibilities pose an issue in seven major software ecosystems (Cargo (Rust), Maven (Java and other JVM languages), NPM (mostly JavaScript), NuGet (C# and other .Net languages), Packagist (PHP), PyPI (Python), RubyGems (Ruby)), we investigate empirically and quantitatively the following research questions on the Libraries.io dataset [12].

RQ1 Which licenses are mainly used in various software ecosystems?
RQ2 How many direct license incompatibilities exist in various ecosystems?
RQ3 How many direct license incompatibilities are caused by AGPL licenses?
RQ4 What is the impact of indirect AGPL license incompatibilities?

Our goal is to investigate how many cases similar to the motivational rails example exist in various ecosystems and to assess if there is a risk for Danish online e-government systems.

Our results show that permissive licenses, such as, MIT and Apache are most frequent in the studied ecosystems. The number of direct license incompatibilities is not negligible, ranging from low 2.3% in Cargo over 4.7%, 5.2%, 6.2%, 9.1%, and 9.1% in NuGet, NPM, RubyGems, Maven, and Packagist respectively to a large 20.8% in PyPI. Only a low amount of direct license incompatibilities are caused by AGPL licensed packages (max. 0.04% in PyPI). Lastly, a whopping 6.62% of Maven packages are violating the AGPL license of an indirect dependency. However, only a few packages are responsible for this large number of indirect license incompatibilities in Maven.

In the remainder, we describe the impact of license incompatibilities and potential resolutions more detailed in Sec. II, we detail the our experiment setup in Sec. III, we describe our results in Sec. IV and we discuss our results in Sec. V and Sec. VII.

II. BACKGROUND

License Incompatibility Example: Fig. 1 illustrates an excerpt of the dependencies of the web-framework Ruby on Rails (rails) before Mar. 24th 2021. Via the package manager RubyGems, rails depends indirectly (by activestorage and marcel) on mimemagic, a library for detecting MIME types of files. mimemagic in turn depends on a single artifact from the xdg-shared-mime-info project. That artifact, an XML file that serves as database of MIME types, which is copied from the original project and reused in mimemagic. The XML file and the entire xdg-shared-mime-info are licensed under GPL-2.0 whereas mimemagic is under MIT license. These two licenses are incompatible, since GPL-2.0 is a so-called "viral" license that would require dependents to open-source their code under the same license, which the MIT license does not require.

After being informed about the license incompatibility, the mimemagic maintainer decides to resolve the license incompatibility by releasing new versions of mimemagic that switches license from MIT to GPL-2.0. The rails community discussed how to resolve the license incompatibility issue on their side. Since many commercial closed-source projects are built on top of rails, changing license to GPL-2.0 is not an option. Developers discuss if they can replace the dependency to mimemagic with another suitably licensed project. Finally, after consulting the mimemagic developers, the latter decide to license their project again under MIT, to remove the artifact that generated the issue from their sources, and to describe to developers how to obtain a copy of that artifact at runtime so that mimemagic can import it from the user’s filesystem.

Effects of License Incompatibilities: Imagine an e-government system that is built on top of rails by a third-party vendor and that rails were licensed under GPL. When the vendor hosts this system for a public sector institution, then it does not matter if any dependency is under GPL, since it only prescribes open-sourcing a system on distribution. That is, if the same vendor delivers the system to another party for hosting, the GPL prescribes that the sources of the system are made publicly available. Imagine now, that rails or its indirect dependency mimemagic were under AGPL. That would mean that as soon as the system is publicly accessible, its sources would have to be made publicly available too, no matter which party is hosting the system.

That is the motivation for this work. Usually, Danish e-government systems are closed source. However, since they are most often web-applications and since they usually rely on third-party packages from various ecosystems, it is important to access how likely the license of a dependency would create an issue for online e-government systems.

Terminology: Various ecosystems and package managers call reusable components differently. For example, in the Maven realm reusable components are called Jars, in Ruby they are called Gems, in Cargo they are called Crates and in other ecosystems, e.g., PyPI and NPM they are just called packages. Irrespective the ecosystem, we call reusable components that are distributed via package managers uniformly packages.

We call a dependency between two packages that are connected via a dependency relation in the dependency network a direct dependency. Contrary two packages possess an indirect dependency if the required package is not directly declared by the dependent package but by an intermediate.

We call the act of making the sources of a software system publicly available under a corresponding license open-sourcing.

Open-source software licenses fall into two major categories: permissive and protective licenses. The latter are often called copyleft licenses.

Permissive licenses restrict minimally how software can be used, modified, or redistributed. Examples of such licenses are the MIT, Apache-2.0, BSD-3-Clause, ISC, or the Unlicense. Permissive licenses permit relicensing of derivative work and allow for use of software also in proprietary software that is distributed.
There are two kinds of protective licenses: weakly and strongly protective licenses. Examples of weakly protective licenses are the LGPL-3.0 or MPL-2.0 and examples of strongly protective licenses are the GPL-3.0 or AGPL-3.0. Strongly protective licenses prescribe to open source derived software and to distribute it under the same license. Software under a weak protective license can be redistributed under another license, as long as that software is not modified and software that reuses such does not need to be open-sourced.

III. METHOD

Fig. 2 illustrates the dataflow in our study setup. A couple of Bash, Python, and SWI Prolog scripts automatically process and analyze the dependency networks of various software ecosystems and generate the results in the following section (Sec. IV). A reproduction kit containing all these scripts and configurations is available online.

Our study is based on the Libraries.io dataset [12]. For various ecosystems, that dataset contains names, licenses, and dependency links of all registered packages. Note, that the dataset provides only the license of the most current release of a package, i.e., not the licenses of earlier versions of a package. License information per package is stored via Software Package Data Exchange (SPDX) identifiers – or a comma separated list of these –, which we use for this study.

We focus our study on the seven ecosystems Cargo, Maven, NPM, NuGet, Packagist, PyPI, and RubyGems since a) these ecosystems are most relevant for Danish e-government systems and b) the Libraries.io dataset contains dependency links for these ecosystems.

The filter action in Fig. 2 illustrates that our scripts download the Libraries.io dataset (more than 20GB in size) and extract all information relating to packages from the seven ecosystems and their dependencies. Furthermore, the available data is reduced to retain only package identifiers, names, and license information. We keep only those dependency links that are labeled as compile- or runtime-dependency in the original dataset since such dependencies may create license incompatibilities.

Since the Libraries.io dataset contains license information only on a package level, we reduce the dependencies between versions of packages to logical dependencies. That is, we reduce the directed multigraph from the input dataset into a directed graph, e.g., as soon as one version of a package eventually depended on a version of another package, a single dependency link is retained. The filtered data is stored for all seven ecosystems in two CSV files, one for the packages and another for the dependency links.

In a next step (illustrated as convert action in Fig. 2), these two CSV files are converted into SWI Prolog knowledge bases. We do that, since license compatibility is a constraint satisfaction problem, which can be efficiently encoded and resolved with Prolog programs. Manually, we create a set of license incompatibility rules as facts in a Prolog knowledge base. Listing 1 shows Prolog facts that declare AGPL-3.0 incompatible with the respectively stated license. These facts are translated from the license incompatibility matrix from Kamocki et al.'s [14] Public License Selector.

During translation, we map the used license identifiers to the respective SPDX identifiers.

To study RQ1 (Which licenses are mainly used in various software ecosystems?), we just count frequencies of licenses per ecosystem and compute their relative frequency compared to the total number of packages in the relative ecosystem. To investigate RQ2 (How many direct license incompatibilities exist in various ecosystems?), we let the Prolog program compare all packages between which a dependency link exists. For each such pair of nodes, the program checks if a license incompatibility rule – as in Listing 1 – holds true. For RQ3 (How many direct license incompatibilities are caused by

Listing 1. Excerpt of Prolog license incompatibility rules.

| License Combination | Incompatible | License Combination | Incompatible |
|---------------------|--------------|---------------------|--------------|
| AGPL-3.0, LGPL-2.1   | incompatible  | AGPL-3.0, LGPL-2.1-only | incompatible |
| AGPL-3.0, CDDL-1.0   | incompatible  | AGPL-3.0, LGPL-2.0-only | incompatible |
| AGPL-3.0, LGPL-2.1   | incompatible  | AGPL-3.0, LGPL-1.0   | incompatible |
| AGPL-3.0, AGPL-1.0-only | incompatible | AGPL-3.0, LGPL-1.0-only | incompatible |

1 It does not necessarily contain dependency links for all other ecosystems in the dataset.

2 Other dependencies types, e.g., test, development dependencies, etc. usually do not create a license incompatibility since they are not distributed with the resulting system.
we do the same as for RQ2 only that we restrict the analysis to dependency links that point to a package with any \texttt{AGPL} license, i.e., \texttt{APGL-1.0}, \texttt{APGL-2.0}, etc.

As mentioned above, packages in the Libraries.io dataset can have more than one license assigned as a comma separated list of SPDX identifiers. The semantics of a comma in a list of licenses is not specified in the dataset description\(^\text{16}\). We interpret such lists to prescribe a disjunction of possible licenses, i.e., a list of possible licenses to choose from depending on the use case. Therefore, when searching for license incompatibilities, we analyze the cross-product of license combinations and record a license incompatibility only when all possible license pair combinations are incompatible. Or vice versa, if one combination allows for a compatible use of a package, we do not mark the dependency as incompatible.

For investigation of RQ4 (What is the impact of indirect \texttt{AGPL} license incompatibilities?), we want to incorporate the transitive structure of dependency relations. Since license incompatibilities can propagate through a dependency network, as in the motivating example with \texttt{rails} and its indirect dependency \texttt{mimemagic} (Sec. II), we identify all those packages that are under an \texttt{AGPL} and for which a dependent possesses an incompatible license. Thereafter, we identify all packages that are on a path that ends in a package with a direct dependency to an \texttt{AGPL} licensed package with a license that is incompatible to an \texttt{AGPL} license.

Consider an exemplary dependency network as in Fig. 3. Let us assume that all packages (nodes) in that network are under MIT license, except of packages \texttt{i}, which is under \texttt{AGPL-3.0}.

For RQ1, we would report that 87.5% of all licenses in the ecosystem are MIT and that 12.5% of all licenses are \texttt{AGPL-3.0}.

The MIT license is incompatible with \texttt{AGPL-3.0}. Consequently, there is a direct license incompatibility between package \texttt{e} and package \texttt{i}. Therefore, we would report for RQ2 and RQ3 respectively, that there are 1 direct license incompatibility, i.e., 12.5% of all dependency links point to a package where the dependency has a license that is incompatible with the dependent.

The impact of the incompatibility with \texttt{i}'s \texttt{AGPL-3.0} is quite large since packages \texttt{a}, \texttt{b}, \texttt{c}, and \texttt{d} depend directly or indirectly on package \texttt{e}. That is, 50% of the ecosystem depend on a package that reuses a package with a license that is incompatible to the respective own license. By transitivity, they have an incompatible license too.

| Packages | Dependencies | Packages\_direct | Packages\_indirect | Incompatibilities | Incompatibilities\% |
|----------|--------------|-----------------|-------------------|------------------|-------------------|
| Cargo    | 35,635       | 19,968          | 5,918             | 29,717           | 453               | 2.3%              |
| Maven    | 184,871      | 426,804         | 101,788           | 83,083           | 39,002            | 9.1%              |
| NPM      | 1,275,011    | 4,927,446       | 912,541           | 362,470          | 257,593           | 5.2%              |
| NuGet    | 199,447      | 488,385         | 145,996           | 53,451           | 22,949            | 4.7%              |
| Packagist| 313,278      | 473,083         | 162,630           | 150,648          | 43,196            | 9.1%              |
| PyPI     | 231,690      | 152,779         | 49,197            | 182,493          | 31,816            | 20.8%             |
| Rubygems | 161,608      | 276,580         | 108,552           | 53,056           | 17,267            | 6.2%              |

**Table I.** DESCRIPTIVE STATISTICS OF LICENSES AND INCOMPATIBILITIES.

**IV. Results**

1) **RQ1:** Which licenses are mainly used in various software ecosystems? Fig. 4 illustrates the share of the most common licenses per ecosystem. For clarity, we plot only the seven most common licenses per ecosystem. A plethora of less frequent licenses –some of which are only used by a single package– occupy the remaining percentages in Fig. 4.

The MIT license is most frequent in Cargo (≈ 54%), NPM (≈ 56%), Packagist (≈ 64%), PyPI (≈ 38%), and RubyGems (≈ 64%). For Maven, the Apache-2.0 license is most common (≈ 41%) with a considerable share in Cargo (≈ 28%) too. In the other five ecosystems the Apache-2.0 license is less frequent with ca. 8% in PyPI, ca. 7% in NuGet, and ca. 4% in NPM, RubyGems, and Packagist respectively. The only ecosystem in this study with a major share of ISC licensed packages is NPM where around a quarter of all packages carry it.

Generally, permissive licenses are most common in the ecosystems. Except of NuGet, where the majority of packages do not carry any license, the majority of packages are permissively licensed.

A considerable amount of packages do not carry a license at all. In NuGet the majority of packages (≈ 63%) do not carry a license. The amount of such packages is lower in the other six ecosystems: Maven (≈ 20%), RubyGems (≈ 24%), PyPI (≈ 14%), Packagist (≈ 9%), and NPM (≈ 7%). In Fig. 4 and in the Libraries.io dataset packages without assigned license are

![Figure 4. Distribution of most common licenses per ecosystem.](image-url)
labeled as “None”. Package without license are essentially not meant for reuse.

Except of PyPI, strongly protective license are quite rare in the ecosystems. PyPI sports the highest share of strongly protective licenses. There, GPL-3.0 and AGPL-3.0 account together for ≈ 9.2% (≈ 6.1% and ≈ 3.1% respectively). In Packagist GPL-2.0 and GPL-3.0 are the common strongly protective licenses, together with a share of only ≈ 4.2%. For the remaining ecosystems, these numbers are even lower: in Maven ≈ 3.0% of packages are GPL-3.0 licensed, in Cargo these are ≈ 2.3%, in RubyGems packages with GPL-2.0 or GPL-3.0 together account for ≈ 1.8%, and NPM and NuGet have ≈ 1.0% of packages under GPL-3.0 respectively.

Only in PyPI a noticeable share of packages is licensed under a version of AGPL (3.1%, i.e., 7,307 packages). For the other ecosystems, this share is below one percent (Cargo: 0.78%, Maven: 0.74%, NPM: 0.24%, NuGet: 0.1%, Packagist 0.21%, and RubyGems: 0.18%).

2) RQ2: How many direct license incompatibilities exist in various ecosystems?: Tab. II lists descriptive statistics for the seven studied ecosystems. Column “|Packages|” shows the total number of packages, “|Dependencies|” shows the total number of logical dependency links in the dependency network (see Sec. II). “|Packages\textsubscript{direct}||” shows the total number of packages that neither depend on another package nor are required by any other package. “|Packages\textsubscript{com}||” shows the number of packages that are connected as dependencies or dependents in the dependency network, and “|Incompatibilities|” and “Incompatibilities\%” show the absolute and relative amount of dependency links where the license of the dependency is incompatible with the license of the dependent package.

Obviously, Cargo is the smallest ecosystem with 35,635 packages of which 29,717 (ca. 83% of all packages) contribute to the dependency network. NPM is the largest ecosystem with 1,275,011 packages, where 912,541 (ca. 72% of all packages) are linked in the dependency network. The sizes of the other five ecosystems are in between these two extremes, ranging from 161,608 packages in RubyGems to 313,278 in Packagist. The share of packages that are connected in the dependency network in the other ecosystems ranges from a meager 27% in NuGet over 33% (RubyGems), 45% (Maven), and 48% Packagist to 79% in PyPI.

Cargo sports the lowest amount of license incompatibilities with 2.3% of all dependencies that link packages with incompatible licenses (corresponding to 453 incompatibilities dependency links). In NuGet these are 4.7% (22,949). In absolute numbers, NPM’s 257,593 license incompatibilities are the largest of all ecosystems. But due to its total size, only 5.2% of all dependency links connect packages with incompatible licenses. In the remaining ecosystems, the share of incompatible licenses raises from 6.2% in RubyGems (17,267) over 9.1% in both Maven (93,002) and Packagist (41,196) to a staggering 20.8% in PyPI (31,816).

3) RQ3: How many direct license incompatibilities are caused by AGPL licenses?: Tab. II lists the number of dependency links between packages where the dependency is under an AGPL license and the dependent is under an incompatible license (column |Incompatibilities\textsubscript{AGPL}|| lists absolute numbers and Incompatibilities\%\textsubscript{AGPL} lists the ratio compared to all dependency links).

The only ecosystem without any license incompatibilities “caused” by dependencies to AGPL licensed packages is Cargo. NPM and Maven have the highest absolute amount of such license incompatibilities with 777 and 148 respectively. However, in general the ratio of incompatibilities caused by packages that incorrectly depend on AGPL licensed packages is low. It is consistently below half per mill. The highest relative share of incompatibilities caused by AGPL licensed packages have Cargo (0.04%) and Maven (0.03%). Remember however, that PyPI is the ecosystem with the highest amount of AGPL licensed packages in this study, see Sec. IV-1.

4) RQ4: What is the impact of indirect AGPL license incompatibilities?: Besides absolute and relative numbers of direct license incompatibilities that are caused by incorrectly depending on AGPL licensed packages, Tab. II lists the number of packages that are affected by such an incompatibility (see columns “|Packages\textsubscript{affected}||” for absolute and “Packages\textsubscript{affected}\%” for relative numbers). All packages that either directly or indirectly depend on a package with a license incompatibility are affected by it.

The absolute and relative numbers of packages affected by an incompatibility with an AGPL license are non-existent in Cargo and very low for NuGet and RubyGems (0.03% with 52 and 53 affected packages respectively), for Packagist (0.04%), and for PyPI (0.05%). For NPM, the share of packages that are affected by a license incompatibility with an AGPL licensed package is 2.38%, which corresponds to 30,377 packages in total. However, these affected packages are fairly wide spread over the ecosystem. In NPM none of the AGPL licensed dependencies with an incompatibly licensed dependent affects
more than thousand other packages. That is, it is not a few AGPL licensed packages that are dependencies of many other packages with an incompatible license.

To the contrary, Maven is the ecosystem with the highest relative amount of AGPL-based license incompatibilities (6.62%) with in total 12,236 affected packages. Here, a low number of packages that incorrectly depend on a package under AGPL license affects a large amount of other packages. Fig. 5 illustrates all those packages from the Maven ecosystem (green, blue, and orange nodes on top) that depend on an AGPL licensed package (bottom nodes in pink) and that affect more than one thousand packages. Sizes of nodes are adjusted to the PageRank of the respective package in the Maven dependency network. The larger a node the more central it is in the Maven ecosystem, i.e., the more packages depend either directly or indirectly on the respective package. The three packages from the Spring framework\(^\text{17}\) (org.springframework:spring-context-support, org.springframework:spring-webmvc, org.springframework:spring-context-support) are all quite central to the dependency network. They are respectively ranked to be the 171\(^{th}\), 351\(^{st}\), and 352\(^{nd}\) most central (important) nodes in the dependency network. Less central but still fairly important are the jfree, jasperreports, grails, and xhtmlrenderer packages. All of these depend on only three packages com.lowagie.itext, com.itextpdf:itextpdf, and com.itextpdf:itext-pdfs. Each of these depends affects more than 11,500 other packages.

The two neo4j packages org.neo4j:neo4j-kernel and org.neo4j:neo4j are less central than the springframework packages in the dependency graph (ranks 465 and 761 respectively) but still, they affect more than 1,300 packages each. Interestingly, for these two packages the license incompatibility is caused by three packages (org.neo4j:neo4j-remote-graphdb and org.neo4j:neo4j-lucene-upgrade, org.neo4j:neo4j-online-backup respectively), which are created by the same organization.

Consequently, it is not too unlikely in Maven to indirectly depend on a package that has a license incompatibility with an AGPL licensed package. Furthermore, in the Maven ecosystem we find a case similar to the example in the introduction, see Sec. 1. Interestingly, like with rails the Spring framework is also a web-development framework.

V. DISCUSSION

As sketched in Sec. II, there are multiple ways of resolving a license incompatibilities. For example, a restrictive license of a dependency can be changed by the respective project owners to a more permissive license that is compatible to those of the dependents. Alternatively, the dependent project changes its license from a more permissive to a more restrictive license that is compatible to the one of its dependency or it replaces the dependency altogether with another suitably licensed package.

Not all license incompatibilities are necessarily an issue. For example, imagine a system that relies for operation on the monitoring tool Prometheus\(^\text{18}\), which is licensed under the Apache-2.0 license, and the dashboard visualization tool Grafana, which is licensed under the AGPL-3.0 license. Such system, does not have to be licensed under AGPL-3.0 and be provided as open-source, since it is not derived work from Grafana but rather reuses a certain web-API during operation. However, if this system was derived from Grafana, e.g., a modified version of it that is used to display data via a web-application, it would have to be open-sourced under a respective license. These semantic differences of the purpose of package reuse cannot be inferred via static analysis, i.e., they are not represented in the results of this study. The purpose of this study is only to access the size of a potential issue and to indicate potential refactoring candidates.

License incompatibilities as described in this paper may not be an issue in non-US American/Anglo-Saxon legislations. For example, a French court ruled recently that GPL copyright claims are invalid in France and may only be enforced via contractual disputes\(^\text{19}\) Additionally, it is unclear if licenses, such as, the GPL are actually a copyright or a contract in non US legislations\(^\text{15}, \text{16}\).

Threats to validity:

The quality of our results depend on the accuracy of dependency link information in the Libraries.io dataset, which we trust to be accurate.

We are no lawyers. That is, all reported results assume that the license incompatibility model, which we adapt from Kamocki et al.’s\(^\text{14}\), accurately captures incompatible licenses. Likely this model is coarse grained, since license incompatibility also depends on the use case of software that

---

17. The Spring framework: org.springframework is a web-development framework.
18. Prometheus is licensed under Apache-2.0 license.
19. French copyright law allows only to enforce such claims via contractual disputes, not via statutory remedies.
reuses other packages. Our model is use case agnostic and therefore might be too strict.

We rely on the latest available Libraries.io dataset [12], which was released on Jan. 12th, 2020. The dataset contains license information only statically on package level. That is, for each package only one or more licenses are given. However, a package might change its license(s) over time. For example, versions of com.lowagie.text below version 5.0.0 were released under the more permissive MPL or LGPL [20] and first with version 5.0.0 the license was switched to AGPL-3.0. Since the most current, this is the only license that is recorded for the package. Consequently, our results in Sec. IV may be overestimating AGPL-based license incompatibilities, since certain versions of packages that in our study exhibit license incompatibilities may depend on suitably licensed earlier versions.

The dependency links available in the Libraries.io dataset are a lower bound approximation of the “real” dependency links in the ecosystems. The dataset only includes those dependency links that are statically declared in the package metadata. For example, PyPI and Maven packages can execute code during installation of packages, which can be used to install other dependencies on top of those that are statically declared. These are not subject of this study.

The Libraries.io dataset provides package licenses as comma separated lists of one or more licenses. The semantics of such commas in a list of licenses is not specified in the dataset description [16]. Thus, we interpret that lists prescribe a disjunction of possible licenses, i.e., a list of possible licenses to choose from depending on the use case. When searching for license incompatibilities, we analyze the cross-product of license combinations and record a license incompatibility only when all possible license pair combinations are incompatible. That might be an overly permissive interpretation, since some projects want to express via multiple licenses that the provided sources are made up by source under the given licenses, i.e., a conjunction of these.

VI. FUTURE WORK

In future, we plan to execute the experiment again on another dataset, which captures licenses per versions of packages and not per package in general. This will allow us to assess if the reported high impact of license incompatibilities with a few AGPL licensed packages in Maven is an artifact or if it adequately represents reality. We are currently working on creating such an updated dataset ourselves, since the Libraries.io dataset appears to be stale [21]. Furthermore, working on more recent data will allow us to appropriately describe the current impact of AGPL licensed packages.

In our results, the amount of license incompatibilities that are caused by incorrect use of AGPL licensed packages is quite low or negligible for all ecosystems except Maven, see Sec. IV. Likely, the reason for this is that the AGPL becomes increasingly popular first recently [22], i.e., after the latest release of the Libraries.io dataset from Jan. 12th, 2020.

VII. CONCLUSIONS

In this paper, we investigate empirically and quantitatively to which degree license incompatibilities pose an issue in the seven major software ecosystems Cargo, Maven, NPM, NuGet, Packagist, PyPI, and RubyGems.

Our results (Sec. IV) show that in most ecosystems the permissive MIT and Apache licenses are most common, only in NuGet it is most common that packages do not carry any license. PyPI is the only ecosystem with a noteworthy amount of packages under AGPL-3.0 but only 3.1% of all packages carry it.

The amount of license incompatibilities between packages that are directly connected via dependency links differs across the ecosystems. The lowest amount of dependency links that cause a license incompatibility can be found in Cargo (2.3% (453 dependencies)). This number increases from 4.7% (22,949) in NPM to 5.2% (257,593), 6.2% (17,267), 9.1% (39,002), and 9.1% (43,196) in NPM, RubyGems, Maven, and Packagist to 20.8% (31,816) in PyPI. However, only a low amount of direct license incompatibilities are caused by packages that depend on AGPL licensed packages but possess an incompatible license themselves (max. 0.04% in PyPI).

When studying the amount of packages that are either directly or indirectly affected by a license incompatibility with AGPL licensed packages, we find that 6.62% (12,236) of all Maven packages are affected but that only a few packages are causing such indirect license incompatibilities.

A. Implications for Practitioners

Our results illustrate that license incompatibilities in general and license incompatibilities caused by inappropriately depending on AGPL licensed packages are more frequent in some ecosystems compared to others. Consequently, we recommend that practitioners not only check direct dependencies for compatible licenses but also indirect dependencies especially when working with Maven and PyPI. Some package managers, e.g., mvn with the dependency:tree of npm with the list command, allow to inspect direct and indirect dependencies simultaneously. In other package managers, such as, pip, dependency trees can be inspected via tools like pipdeptree [23]. On top of that, there exist tools that can be integrated to the development process to support developers in creating software free of license incompatibilities, e.g., the IDE plug-in Sorrel [17].

B. Implications for Researchers

We only studied license incompatibilities within ecosystems. Modern applications, in particular web-applications, typically reuse code from multiple ecosystems. It remains an open question to which degree software projects from various domains are affected by license incompatibilities within ecosystems or to which degree they introduce even new incompatibilities on top of existing ones.

REFERENCES

[1] “Digital Economy and Society Index Report 2019,” https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=59975, accessed: 2021-10-19.
[2] D. Regeringen, D. Regioner, and KL., “The digital path to future welfare: Egovernment strategy 2011-2015,” Tech. Rep., 2011. [Online]. Available: https://joinup.ec.europa.eu/sites/default/files/document/2016-06-engelsk_strategi_tilgaengelig.pdf
[3] “SIG/TÜV iT Evaluation Criteria Trusted Product Maintainability, Version 12.0,” https://www.softwareimprovementgroup.com/wp-content/uploads/2020-SIG-TÜViT-Evaluation-Criteria-Trusted-Product-Maintainability.pdf, accessed: 2020-10-27.
[4] J. Visser, S. Rigal, R. van der Leek, P. van Eck, and G. Wijnholds, “Building maintainable software, java edition: Ten guidelines for future-proof code,” 2016.
[5] V. R. Basili, L. C. Briand, and W. L. Melo, “How reuse influences productivity in object-oriented systems,” Communications of the ACM, vol. 39, no. 10, pp. 104–116, 1996.
[6] W. C. Lim, “Effects of reuse on quality, productivity, and economics,” IEEE software, vol. 11, no. 5, pp. 23–30, 1994.
[7] P. Mohaghegh, R. Conradi, O. M. Killi, and H. Schwarz, “An empirical study of software reuse vs. defect-density and stability,” in Proceedings. 26th International Conference on Software Engineering. IEEE, 2004, pp. 282–291.
[8] H. Orsila, J. Geldenhuys, A. Ruokonen, and I. Hammouda, “Update propagation practices in highly reusable open source components,” in IFIP International Conference on Open Source Systems. Springer, 2008, pp. 159–170.
[9] C. Vendome, “Assisting software developers with license compliance,” 2018.
[10] A. Mathur, H. Choudhary, P. Vashist, W. Thies, and S. Thilagam, “An empirical study of license violations in open source projects,” in 2012 35th Annual IEEE Software Engineering Workshop. IEEE, 2012, pp. 168–176.
[11] R. Duan, A. Bijlani, M. Xu, T. Kim, and W. Lee, “Identifying open-source license violation and 1-day security risk at large scale,” in Proceedings of the 2017 ACM SIGSAC Conference on computer and communications security, 2017, pp. 2169–2185.
[12] J. Katz, “Libraries.io Open Source Repository and Dependency Metadata,” Jan. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3626071
[13] G. M. Kapitsaki, N. D. Tseliakas, and I. E. Fokarakis, “An insight into license tools for open source software systems,” Journal of Systems and Software, vol. 102, pp. 72–87, 2015.
[14] P. Kamocki, P. Stránská, and M. Sedláček, “The public license selector: making open licensing easier,” in Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016), 2016.
[15] E. Szatler et al., “Gpl-“ viral” license or “ viral” contract,” Masaryk University Journal of Law and Technology, vol. 1, no. 1, pp. 67–79, 2007.
[16] A. Guadarrunz, “Viral contracts or unenforceable documents? contractual validity of copyleft licenses,” EIPR, vol. 26, no. 8, pp. 331–339, 2004.
[17] D. Pogrebnoy, I. Kuznetsov, Y. Golubev, V. Tankov, and T. Bryksin, “Sorrel: an ide plugin for managing licenses and detecting license incompatibilities,” arXiv preprint arXiv:2107.13315, 2021.

**REFERENCES**

1. https://www.sundhed.dk/
2. https://www.borger.dk
3. https://www.theregister.com/2021/03/25/ruby_rails_code/
4. https://tldrlegal.com/license/gnu-affero-general-public-license-v3-(agpl-3.0)
5. https://github.com/freedesktop/xdg-shared-mime-info
6. https://github.com/freedesktop/xdg-shared-mime-info/blob/master/data/freedesktop.org.xml.in
7. https://github.com/mimemagicrb/mimemagic/issues/97
8. https://github.com/mimemagicrb/mimemagic/commit/c0f7b6b21a192629839db8794d08f9ff7e
9. https://github.com/rails/rails/issues/41757
10. https://github.com/rails/rails/issues/41750
11. https://www.gnu.org/licenses/copyleft.html
12. https://www.gnu.org/licenses/license-compatibility.html
13. https://github.com/HelgeCPH/license_compat_study/
14. https://spdx.org/licenses/
15. https://libraries.io/data?projectFields
16. https://prometheus.io/
17. https://thehftguy.com/2021/08/30/french-appeal-court-affirms-decision-that-copyright-claims-on-gpl-are-invalid-must-be-enforced-via-contractual-dispute/
18. https://en.wikipedia.org/wiki/IText
19. https://github.com/librariesio/libraries.io/issues/2744
20. https://www.synopsys.com/blogs/software-security/using-agpl-adoption/
21. https://github.com/naiquevin/pipdeptree