# Análise de Incompatibilidade entre Licenças em Projetos de Código Aberto

**Letícia A. F. Gonçalves**[1], **Octávio T. R. Lage**[1]

[1]Pontifícia Universidade Catolica de Minas Gerais (PUC MG)
30140-002 — Belo Horizonte — MG — Brasil

{leticia.amanda,octavio.lage}@sga.pucminas.br

***Abstract.*** *The present article is dedicated to a comprehensive investigation of challenges related to licensing in open-source software projects. The research method involved the analysis of repositories hosted on GitHub and querying dependency data in the NPM Registry to identify the main causes of license violations between projects and their dependencies. The research was based on a sample of 1000 Node.JS projects, with the results revealing a limited violation, with only 3.5% of project-dependency relationships concerning copyleft-related license terms. Furthermore, the study identified the most widely adopted libraries in open-source projects, highlighting eslint (63.4%), typescript (51.6%), and prettier (47.8%) as the primary ones. The main contribution of the research was the library selection process based on license compatibility.*

***Resumo.*** *O presente artigo se dedica a uma investigação abrangente dos desafios relacionados ao licenciamento em projetos de software de código aberto. Para atingir esse propósito, empregou-se uma metodologia que envolveu a análise de repositórios hospedados no GitHub e consulta de dados de dependências no NPM Registry, visando identificar as principais causas de violação de licenças entre projetos e suas dependências. A pesquisa se baseou em uma amostra de 1000 projetos Node.JS, cujos resultados revelaram uma violação restrita, apenas 3,5% das relações entre projeto e dependência sobre os termos de licença relacionados a copyleft. Além disso, o estudo identificou as bibliotecas mais amplamente adotadas em projetos de código aberto, destacando eslint (63,4%), typescript (51,6%) e prettier (47,8%) como as principais. A principal contribuição da pesquisa foi processo de seleção de bibliotecas baseado na compatibilidade de licenças.*

**Bacharelado em Engenharia de *Software* - PUC Minas**
**Trabalho de Conclusão de Curso (TCC)**

Orientador de conteúdo (TCC I): Cleiton Tavares - cleitontavares@pucminas.br
Orientador acadêmico (TCC I): Laerte Xavier - laertexavier@pucminas.br
Orientador do TCC II: Rodrigo Baroni - baroni@pucminas.br

Belo Horizonte, 20 de Dezembro de 2023.

## 1. Introdução

A estratégia da Engenharia de Software Baseada em Reúso é bastante relevante no cenário de desenvolvimento de software, tornando-se comum a utilização de bibliotecas externas para aumentar a eficiência, a qualidade e reduzir custos na produção de

soluções de software [Nepomuceno et al. 2021]. No entanto, essas bibliotecas podem possuir diferentes tipos de licença, levando a problemas de compatibilidade e violação das licenças do projeto, o que têm se tornado uma questão cada vez mais significativa, inclusive para software de código aberto ou OSS (OSS, do inglês *Open Source Software*) [Duan et al. 2017, Kim et al. 2022]. Essas violações podem ocorrer de diversas formas em OSS, desde a não divulgação do código-fonte modificado como a GNU *General Public License* (GPL) exige, até o uso indevido de bibliotecas e componentes sem cumprir as condições da licença [Kim et al. 2022].

Na literatura, encontram-se diversos estudos propondo ferramentas para a identificação do reúso em OSS [Xu et al. 2021, Woo et al. 2021, Dutta et al. 2022, Reid et al. 2022]. Porém, esses estudos visam apenas a identificação e não expõem as violações de licenças originadas a partir do reúso. Encontram-se também estudos que quantificam violações de reúso de OSS [Kashima et al. 2011, Duan et al. 2017]. No entanto, eles não abordam todos os possíveis conflitos entre essas licenças de um projeto e suas bibliotecas. A incompatibilidade entre as licenças de diferentes bibliotecas utilizadas em um mesmo projeto de software é frequentemente observada em projetos de software livre, no qual as bibliotecas possuem licenças distintas e, em muitos casos, conflitantes [Higashi et al. 2022]. Diante disso, o **problema estudado neste trabalho é o potencial de pesquisa inexplorado a respeito da identificação e tratamento da violação de licenças de bibliotecas entre projetos de software de código aberto e suas dependências**.

A construção habitual de sistemas explorando componentes prontos pode parecer uma prática inofensiva. No entanto, é importante ressaltar que, frequentemente, os autores desses sistemas não possuem conhecimento suficiente sobre possíveis violações de licenças de bibliotecas envolvidas nos componentes utilizados e tal fato pode acarretar consequências graves [Hemel et al. 2011, Reid et al. 2022]. Portanto, estar atento às normas das licenças de software utilizados é crucial para evitar problemas em qualquer projeto [Xu et al. 2021]. A não observância das regras estabelecidas nas licenças e o reúso de software sem autorização podem gerar consequências legais para os infratores, o que pode resultar em multas e danos à reputação do projeto, empresa ou organização [Kashima et al. 2011, Xu et al. 2021]. Além disso, a violação de licenças de bibliotecas pode prejudicar os desenvolvedores originais, que podem perder receita e incentivos para continuar investindo em novos projetos.

Portanto, o **objetivo geral deste estudo é investigar violações de licenças de bibliotecas usadas entre projetos de software de código aberto e suas dependências**. Como objetivos específicos, pode-se destacar: (i) explorar os problemas mais comuns relacionados à compatibilidade entre as licenças de projeto e de bibliotecas; (ii) identificar os conflitos das licenças das bibliotecas mais comumente utilizadas em projetos de software aberto; e (iii) investigar soluções para mitigar os problemas identificados, como a adoção de licenças adequadas ou a substituição de bibliotecas incompatíveis.

Este trabalho encontra-se organizado em sete seções: a Seção 2 apresenta a fundamentação teórica, explorando conceitos aplicados no estudo; a Seção 3 destaca os principais trabalhos relacionados; a Seção 4 detalha os materiais e métodos utilizados; a Seção 5 apresenta e discute os resultados obtidos; a Seção 6 detalha as possíveis ameaças à validade observadas; por fim, a Seção 7 expõe as conclusões finais e trabalhos futuros.

## 2. Fundamentação Teórica

Em primeiro plano, destaca-se o conceito de engenharia de software baseada em reúso, seguido pela abordagem sobre bibliotecas de software, além de apresentar sobre licenças de OSS, e por fim, o conceito de violação de licenças de bibliotecas OSS.

### 2.1. Engenharia de Software Baseada em Reúso

A Engenharia de Software baseada em reúso é uma abordagem que visa aumentar a eficiência e qualidade dos projetos de software através do aproveitamento de artefatos de software já existentes [Taivalsaari et al. 2019]. O reúso de software consiste na utilização de artefatos previamente desenvolvidos em novos contextos ou na adaptação desses artefatos para novas necessidades. Evita-se assim o retrabalho e aumenta-se a produtividade, além de reduzir o tempo e os custos envolvidos no processo de desenvolvimento [Tung et al. 2014].

Essa abordagem pode ser aplicada em diversas etapas do ciclo de vida de desenvolvimento de software, desde a concepção até a manutenção, permitindo reutilizar requisitos, modelos, código-fonte, testes e documentação [Lampropoulos et al. 2018]. Outra vantagem importante é a redução do tempo de treinamento e capacitação de desenvolvedores, porque o conhecimento necessário para o uso de artefatos de software já existentes é mais fácil de ser adquirido do que o conhecimento necessário para a construção de novos artefatos [Nepomuceno et al. 2021].

### 2.2. Bibliotecas de Software

As bibliotecas de software consistem em conjuntos de códigos ou programas que disponibilizam uma série de funcionalidades para serem empregadas em outras aplicações de software [López de la Mora and Nadi 2018]. As bibliotecas de software são compostas por funções, classes, objetos, rotinas, entre outros componentes, fornecidos para serem incorporados em outros projetos de software [Alnusair et al. 2016]. Elas são amplamente utilizadas na indústria de desenvolvimento de software para aumentar a eficiência e a produtividade dos desenvolvedores, que podem reutilizar componentes já existentes em vez de escrever todo o código do zero [Ye 2009].

O uso de biblioteca contribui para a diminuição da quantidade de erros e o aumento da qualidade do software, uma vez que as bibliotecas já foram testadas e validadas em outros projetos. As bibliotecas de software podem ser proprietárias ou de software livre, e a escolha de uma ou outra depende da estratégia de licenciamento adotada pelo desenvolvedor. A adoção de software proprietário pode ser motivada pela necessidade de obter uma solução mais exclusiva ou suporte técnico especializado e personalizado, oferecido pelos fornecedores dessas soluções. Em projetos de software livre, deve-se utilizar bibliotecas de software livre para garantir a compatibilidade de licenças entre os componentes do software.

### 2.3. Licenças de OSS

Licenças de OSS permitem aos usuários o acesso, uso, modificação e distribuição gratuita do software licenciado [Yun et al. 2017]. Ao contrário das licenças proprietárias, que impõem restrições ou proibições a essas ações, as licenças OSS se baseiam em princípios de colaboração, transparência, comunidade e distribuição do software [Liu et al. 2021].

Dessa forma, as licenças OSS concedem aos usuários um maior controle sobre o software utilizado, permitindo modificações de acordo com suas necessidades e compartilhamento com outros. Além disso, essas licenças incentivam a participação da comunidade no desenvolvimento do software, permitindo que usuários contribuam com correções de *bugs*, novos recursos e outras melhorias.

No que se refere ao reúso de software, as licenças de OSS são especialmente relevantes, uma vez que permitem que o código-fonte seja utilizado em outros projetos e o software modificado possa ser redistribuído sob as mesmas ou diferentes licenças de OSS [Ballhausen 2019]. Cabe salientar que existem diversas licenças de OSS disponíveis, cada qual com suas próprias condições e requisitos. Para uma visão mais abrangente das licenças de software de código aberto mencionadas acima, é recomendado consultar o apêndice A, deste artigo. Apêndice A, onde se encontra uma compilação detalhada das licenças identificadas pela OSI (do inglês, *Open Source Initiative*).

### 2.4. Violação de Licenças de Bibliotecas OSS

A violação de bibliotecas OSS ocorre quando uma pessoa ou organização usa software sem cumprir as obrigações definidas na licença do mesmo. Essas obrigações incluem, entre outras, a atribuição correta dos créditos do software, a disponibilização do código-fonte modificado e a redistribuição do software sob a mesma licença OSS original [Kapitsaki and Paschalides 2017]. As violações de OSS podem resultar em problemas legais para a pessoa ou organização que utiliza o software, porque a violação pode ser considerada uma infração de direitos autorais, uma vez que a licença de biblioteca OSS é uma forma de proteção de propriedade intelectual [Feng et al. 2019]. Além disso, a violação pode afetar a reputação da pessoa ou organização, especialmente se o software em questão é amplamente utilizado ou tem uma comunidade de desenvolvedores engajada.

Para evitar violações de licenças de bibliotecas em código aberto, é importante compreender os termos e condições da licença em questão antes de utilizar o software [Kapitsaki and Charalambous 2021]. As empresas que utilizam software de código aberto em seus projetos devem estabelecer políticas claras sobre seu uso e garantir que seus funcionários as sigam. Isso pode incluir a designação de um responsável pela conformidade, treinamento dos funcionários e a implementação de ferramentas para monitorar o uso do software [Monden et al. 2011]. A violação de licenças de bibliotecas OSS pode ter graves consequências legais e de reputação para indivíduos e organizações. Portanto, é fundamental entender as obrigações definidas pela licença de código aberto antes de utilizar o software e assegurar que essas obrigações sejam cumpridas para evitar violações.

## 3. Trabalhos Relacionados

Nesta seção, apresentam-se diversos trabalhos correlatos que abordam problemas comuns relacionados à compatibilidade entre as licenças de projeto e de bibliotecas através do reúso, exibindo as distintas abordagens e obstáculos encontrados para a concretização desse procedimento.

Golubev et al. (2020) abordaram o problema das violações de licenças em software de código aberto. Para o estudo, os autores compilaram um conjunto de 23.378 projetos Java populares do GitHub, pesquisaram clones de código e realizaram uma análise

original de possíveis empréstimos de código e violações de licenças no nível dos fragmentos de código. Como resultado, no conjunto de dados 94 licenças distintas, constatou-se que pelo menos 9,1% dos arquivos verificados não possuem cobertura de licença. Além disso, foram identificados 29,6% dos fragmentos de código que podem estar relacionados a possíveis empréstimos de código. Tanto o estudo citado quanto este artigo têm em comum a mineração de repositórios do GitHub e a análise de violações de licença em projetos de código aberto.

No intuito de detectar possíveis problemas de incompatibilidade de licenças de software de código aberto em imagens do Docker, Higashi et al. (2022) realizaram uma análise de 776 imagens do Docker publicadas no GitHub. Os autores utilizaram uma ferramenta de análise de licenças de código aberto, chamada *Fossology*, que identifica licenças de código aberto em pacotes de software. Eles identificaram aqueles que utilizavam componentes licenciados sob o GPL e verificaram se as licenças foram adequadamente respeitadas pelos mantenedores das imagens. Em decorrência disso, os resultados indicaram que uma proporção significativa dos pacotes (28,7%) não é compatível, e que 457 (58,9%) imagens apresentaram problemas de incompatibilidade relacionados a GPL. A relação com este trabalho está na utilização da metodologia para identificar incompatibilidades de licenças de bibliotecas OSS.

Liu et al. (2021) propuseram uma solução para auxiliar desenvolvedores na escolha de licenças de código aberto, considerando o impacto das dependências de software nessa seleção. Os autores desenvolveram uma aplicação capaz de coletar informações sobre as bibliotecas de software utilizadas pelos projetos, bem como as respectivas licenças de código aberto associadas a elas. Em seguida, eles conduziram uma análise estatística dos dados coletados para identificar padrões e tendências em relação ao uso de licenças de software livre em projetos que utilizam diferentes bibliotecas. Como resultado, as análises indicam que a escolha da licença de software livre mais apropriada para um projeto não depende apenas das características do projeto, mas também das licenças das bibliotecas de software utilizadas. O estudo em questão possui relação com o trabalho atual no âmbito da escolha de licenças de bibliotecas de software OSS, uma vez que essa decisão pode se tornar um fator crítico para o êxito de um projeto de código aberto.

Visando propor ideias para mitigar vulnerabilidades e incentivar pesquisas acadêmicas adicionais, Reid et al. (2022) abordaram a propagação de vulnerabilidades de segurança de software que são resultantes do reúso de componentes de software. Os autores conduziram um estudo de caso exploratório, analisando projetos de software amplamente reutilizados e identificando as vulnerabilidades conhecidas. Ao detectar as versões afetadas, foi utilizada a base do *World of Code* (WoC) para identificar outros projetos que possuem versões que precedem a versão afetada ou que são modificadas após ela sem aplicar a correção. O estudo encontrou 72.281 repositórios com casos de vulnerabilidades órfãs, mesmo em projetos atualmente ativos e altamente populares. Assim, tem-se que a demora na correção de vulnerabilidades, mesmo em projetos altamente populares, aumenta as chances de espalhamento para novos projetos. Ambos os estudos tratam da identificação de possíveis problemas no projeto a partir do reúso de código, havendo uma relação complementar para ajudar desenvolvedores a garantir que seu software seja seguro e legalmente compatível.

## 4. Materiais e Métodos

Este trabalho submeteu uma pesquisa empírica do tipo quantitativa, de forma que realizou a mineração de dados para análise de violação ou incompatibilidade de licenças ocasionada pela utilização de bibliotecas de código. Os dados coletados são de repositórios de código aberto relevantes no GitHub e de suas respectivas dependências. Nesta seção, são expostas as fases executadas para alcançar os objetivos definidos para esta pesquisa.

### 4.1. Procedimentos

A coleta de informações e metadados de repositórios hospedados efetuou-se por meio da API (do inglês, *Application Programming Interface*) REST (do inglês, *REpresentational State Transfer*) do GitHub. O processo dividiu-se em cinco etapas (Figura 1). A etapa de coleta de repositórios utilizou ferramentas automatizadas de mineração de dados para obter dados relevantes sobre projetos de software aberto, como o responsável, a licença e a linguagem principal. A etapa de identificação de dependências e suas licenças consistiu em obter recursivamente a lista de dependências de cada projeto e suas respectivas licenças por meio de um algoritmo. A etapa manual de matriz de compatibilidade associou licenças de OSS baseadas nas licenças coletadas nas etapas anteriores, analisando os termos de cada licença. Esse passo possibilitou que a análise de compatibilidade de cada repositório fosse realizada de forma automática.

A etapa de análise de compatibilidade de licenças desempenhou a responsabilidade de identificar possíveis violações em cada repositório baseado nas saídas das etapas anteriores. Dessa forma, o *script* automatizado faz a comparação das licenças envolvidas em cada projeto com a matriz de compatibilidade preparada pelos autores. Por fim, a etapa de avaliação de objetivos analisou todos os dados coletados nas etapas precedentes para atingir os objetivos específicos apontados anteriormente. Nas seções a seguir, são apresentados detalhes sobre cada uma das etapas mencionadas.

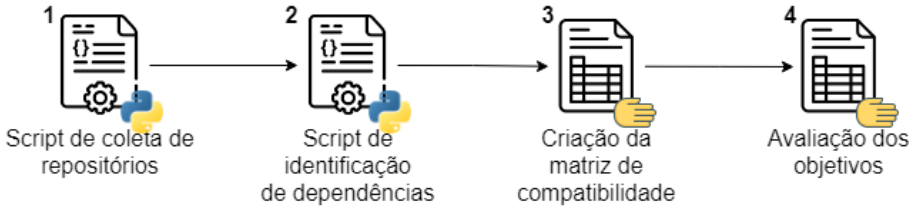


**Figura 1. Diagrama de procedimentos.**

### 4.2. Coleta de Repositórios

A primeira etapa constou no desenvolvimento de um *script* na linguagem de programação Python que utiliza a API REST do GitHub[1] para coletar 1.000 dados em repositórios de código aberto. O GitHub é um sistema de hospedagem e versionamento de código na Internet que disponibiliza suas APIs gratuitamente e é amplamente utilizado pela comunidade de OSS. Por meio dele e da API disponibilizada, adotaram-se os seguintes filtros de repositórios (FR), para a remoção de repositórios irrelevantes e para fins de reprodutividade ou repetibilidade deste estudo:

[1] Acesso à documentação da API REST do GitHub em https://docs.github.com/rest

- **FR1**: Possuir JavaScript (JS) ou TypeScript (TS) como linguagem de programação principal, objetivando a viabilidade desse estudo;
- **FR2**: Possuir licença descrita em seu repositório, para viabilizar que o mesmo possa ser analisada;
- **FR3**: Possuir mais de 100 estrelas, com o intuito de descartar repositórios pouco relevantes;
- **FR4**: Possuir o arquivo `package.json` no repositório para garantir que o repositório se trata de um projeto Node.js;
- **FR5**: Ordenar de modo decrescente pela quantidade de estrelas para realizar a seleção dos repositórios mais relevantes.

A coleta de dados efetivou-se exclusivamente em repositórios de software baseados em Node.js[2], ou seja, que empregam JavaScript ou TypeScript como linguagem principal. Essa restrição ocorre devido à inviabilidade de criar um processo único que analise a dependência de projetos em diversas linguagens, dada a singularidade de como cada uma representa em suas dependências. Além de ser uma tecnologia altamente popular, o Node.js possui um esquema claro de *frameworks* e bibliotecas necessárias para o sistema, permitindo uma análise objetiva de cada uma das dependências envolvidas no projeto.

### 4.3. Identificação das Dependências e suas Licenças

Ainda utilizando da API REST do GitHub, conduziu-se a identificação de dependências do projeto pelo arquivo `package.json`. Esse arquivo é criado automaticamente ao iniciar um novo projeto Node.js e todos os projetos devem incluí-lo. O propósito desse arquivo é listar todas as dependências necessárias e suas respectivas versões para executar o código. É importante destacar que as dependências de um projeto podem ter suas próprias submissões, tornando o processo de identificação de dependências mais complexo. Nesse caso, se fez necessário realizar uma busca em profundidade para obter todas as dependências do projeto, juntamente com suas respectivas licenças, a fim de garantir uma análise sobre todas as licenças envolvidas no projeto.

Após o processo de identificação das dependências em cada um dos projetos, tornou-se necessário prosseguir com uma identificação recursiva das licenças e suas dependências. Para cada dependência identificada em um projeto, foi consultada a licença dela em seu repositório, cruzando todas as licenças incluídas no projeto, a fim de obter todas as licenças desse projeto.

### 4.4. Matriz de Compatibilidade

A montagem da matriz de compatibilidade entre as licenças de OSS coletadas em etapas antecedentes envolveu uma análise minuciosa dos termos de cada licença relacionados ao uso e distribuição, os quais pertencem ao contexto de utilização de bibliotecas de código. Fez-se necessário reunir todas as licenças identificadas durante a pesquisa. Em seguida, qualquer termo e cláusula de cada licença foram cuidadosamente revisados para compreender suas restrições e requisitos específicos. Com base nessa análise, estabeleceram-se as relações de compatibilidade entre as diferentes licenças. A matriz de compatibilidade resultante é uma ferramenta valiosa que forneceu informações claras sobre quais licenças

[2]Node.js é um ambiente de tempo de execução JavaScript multiplataforma e de código aberto. Mais informações em https://nodejs.org/.

podem ser combinadas em um projeto de software sem violar as restrições legais impostas por cada uma delas. Esse processo manual assegurou uma compreensão precisa das implicações legais, possibilitando a avaliação da combinação de licenças de OSS em projetos descritos nas próximas etapas.

### 4.5. Avaliação dos Objetivos

Com o propósito de atingir os objetivos estabelecidos na análise de violação de licenças, foram formuladas questões de pesquisa (RQ, do inglês *Research Question*) que auxiliam na definição de metas claras e específicas. Cada questão de pesquisa está diretamente relacionada a um objetivo específico, possibilitando uma abordagem mais estruturada da pesquisa.

- **RQ1:** Quais são os principais problemas encontrados ao combinar diferentes licenças de projeto e bibliotecas em projetos de software aberto?
- **RQ2:** Quais são os conflitos de licenças de bibliotecas mais populares ou amplamente adotadas em projetos de software aberto?
- **RQ3:** Quais são as práticas eficazes para mitigar os problemas causados pela incompatibilidade entre licenças de projeto e de bibliotecas em projetos de software aberto?

A RQ1 satisfaz o primeiro objetivo específico, cujo propósito é investigar os problemas mais comuns relacionados à compatibilidade entre as licenças de OSS e suas dependências. A RQ2 cumpre o segundo objetivo específico, visando explorar os conflitos de licenças das bibliotecas mais utilizadas nesses projetos. Para ambas, foi observada a saída do *script* de análise de licenças, onde se tem uma lista dos projetos identificados com incompatibilidades, a biblioteca usada e a violação cometida. A RQ3 atende o terceiro objetivo específico, que propõe uma investigação de soluções para mitigação dos problemas identificados. Foram utilizados os resultados obtidos das RQs para explorar quais medidas podem ser tomadas para minimizar os casos de incompatibilidade, sendo feitos testes exploratórios para os casos comuns.

## 5. Resultados e Discussões

Esta seção detalha a caracterização dos objetos de estudo, bem como a análise das dependências e os resultados e implicações analisadas.

### 5.1. Caracterização dos Repositórios

No desenvolvimento do *script* de coleta de repositórios, buscou-se inicialmente utilizar a API GraphQL devido à sua flexibilidade na seleção dos dados a serem recuperados. Entretanto, devido a algumas questões relacionadas a filtros e instabilidades ocasionais, optou-se por prosseguir exclusivamente com a API REST.

Em virtude da decisão de não empregar a API GraphQL, ao invés de adquirir os 1.000 repositórios em conjunto para as linguagens, foram coletados 1000 repositórios de JavaScript e 1000 TypeScript separadamente. Na API, foram implementados filtros relacionados à linguagem e ao tipo de licença, sendo este último diferenciado devido à aplicação de licenças personalizadas, o que impossibilitou a análise minuciosa termo a

termo. Cada consulta foi estruturada de modo a ordenar os resultados em ordem decrescente de estrelas e um indicador de popularidade, com a inclusão do filtro *stars*:<100 para identificar repositórios com mais de 100 estrelas.

Após a aquisição de 2.000 repositórios (1000 JS e 1000 TS) consolidados em uma única lista, realizou-se a subsequente filtragem. Esta etapa consistiu na seleção dos repositórios com licenças de uso inequivocamente definidas, além da identificação daqueles inquestionavelmente associados à plataforma Node.JS. A identificação se deu através da presença do arquivo `package.json` no diretório principal do repositório, sendo importante mencionar que a ausência deste arquivo foi considerada uma exceção.

Posteriormente, o conteúdo do arquivo `package.json` foi extraído para acessar detalhes como a lista de dependências utilizadas tanto em ambientes de produção quanto de desenvolvimento. Após a conclusão dessa fase de coleta de dados, realizou-se a seleção dos 1.000 principais repositórios com a maior quantidade de estrelas dentre os 2.000, como parte do processo contínuo de análise. Os repositórios que não se enquadraram nesse critério foram cuidadosamente arquivados em um documento separado, contendo as razões de descarte.

### 5.2. Análise das Dependências

No contexto da análise de dependências, procedeu-se à investigação de cada dependência extraída dos projetos coletados, utilizando a API do Registro Público de Pacotes do *Node Package Manager* (NPM), o NPM Registry.[3] O propósito subjacente a essa pesquisa consistiu em identificar a dependência compatível com a versão especificada, bem como em coletar informações essenciais, incluindo o nome, a licença e a URL do repositório do projeto.

Inicialmente, a proposta envolvia a inclusão de bibliotecas instaladas localmente ou referentes a *links* externos na análise. No entanto, deparou-se com um desafio de magnitude significativa relacionado ao grau de dependência subjacente a essas bibliotecas. Portanto, foram deliberadamente excluídas desta análise as bibliotecas instaladas localmente ou aquelas que referiu-se a *links* externos, devido à complexidade inerente associada à avaliação de todos os cenários. Em alguns casos, a cadeia de dependências poderia se estender indefinidamente, culminando em um nível de recursividade extremamente elevado.

Em geral, ao considerar a avaliação de conformidade das dependências, o enfoque tradicional se restringe ao primeiro nível, ou seja, às dependências diretas do projeto principal. Contudo, ao buscar uma análise recursiva abrangendo todas as dependências até sua conclusão, a complexidade resultante revelou-se excessiva e, em algumas instâncias, até de caráter infinito. Com o intuito de simplificar e tornar mais gerenciável essa avaliação, restringiu-se a análise ao primeiro nível de dependências, mantendo um grau determinado de autonomia nessa abordagem. Isso permitiu direcionar a atenção para as dependências imediatamente relacionadas ao projeto, sem se aprofundar em uma análise infinita das camadas de dependências secundárias.

[3]Registro público de pacotes baseados em Node.js. Saiba mais em https://docs.npmjs.com/cli/v8/using-npm/registry

### 5.3. Resultados e Implicações

Após a obtenção de informações relativas a todas as dependências, elaborou-se uma tabela destinada à avaliação de possíveis infrações em cada relação entre projeto e dependência. Essa análise baseou-se principalmente nos termos que regulam a distribuição e o uso de cada licença que constituem o escopo deste estudo. Durante a análise, situações em que a clareza em relação à licença estava ausente foram consideradas. A partir desse ponto, cada relação foi classificada por três categorias: **compatível**, **incompatível** ou **desconhecida**.

A matriz de compatibilidade criada engloba 41.068 resultados, dos quais 35 foram identificados como incompatíveis. Para uma compreensão mais aprofundada dos resultados incompatíveis, recomenda-se o trecho da matriz de compatibilidade detalhada no Apêndice B, deste artigo. As razões para a incompatibilidade incluem:

- AGPLv3.0, uma licença *copyleft* que exige que todos os usuários que distribuam o software em um ambiente de rede, incluindo servidores, forneçam o código-fonte completo do software, enquanto a GPLv2.0 não exige essa medida;
- A licença AGPLv3.0 é *copyleft* e exige que os projetos utilizem a mesma licença ou uma compatível, o que a licença MIT não atende a todos os requisitos da AGPLv3.0;
- A família de licenças EPL possui *copyleft*, exigindo que os projetos derivados utilizem uma licença igual à de sua dependência ou mais recente;
- A licença GPLv2.0 exige que todos os termos de sua licença sejam mantidos em projetos derivados;
- A menos que seja especificado (gplv2.0-or-later), o *copyleft* não permite o uso de uma versão mais recente da GPLv2.0;
- A licença GPLv2.0 também exige que todos os termos de sua licença sejam mantidos em projetos derivados.

Além disso, classificaram-se como compatíveis aquelas licenças do projeto que estavam conforme todos os termos relacionados à distribuição e ao uso da dependência, considerando que a utilização ocorre por meio do gerenciador de pacotes Node.js, em vez de bibliotecas locais ou repositórios específicos. As categorizadas como desconhecidas foram aquelas relações onde as bibliotecas não tinham licenças definidas, uma vez que não era possível determinar a real intenção dos autores. A ausência de definição de uma licença difere da adoção de uma licença de domínio público, como a *Creative Commons Zero* e a *The Unlicense*, por exemplo, nas quais os autores renunciam a todos os direitos associados ao projeto.

### 5.4. Discussão dos Resultados e Achados

Nesta subseção, apresentam-se as respostas às questões de pesquisa fundamentais que direcionam o estudo.

#### 5.4.1. Problemas na combinação de diferentes licenças de projetos e bibliotecas (RQ1)

Ao combinar diferentes licenças de projeto e bibliotecas em projetos de software aberto, o estudo identificou um problema predominante: as cláusulas relacionadas ao *copyleft*.

As cláusulas de *copyleft* presentes em algumas licenças de código aberto podem impor restrições significativas quando combinadas com diferentes licenças em um projeto. Essas cláusulas exigem que qualquer derivado seja também licenciado sob os mesmos termos de *copyleft*, o que pode limitar a escolha e a flexibilidade dos desenvolvedores ao criar projetos de software aberto. No contexto do teste realizado, concentrou-se em bibliotecas de código sem modificação obtidas diretamente do NPM *Registry*. Não foram observadas outras violações das cláusulas de uso e distribuição de software de código aberto.

Na amostra coletada, as licenças que tiveram suas cláusulas relacionadas a *copyleft* violadas foram Eclipse Public License 2.0, GNU General Public License (GPL) 2.0, GNU GPL 3.0, GNU Affero General Public License (AGPL) 1.0, GNU AGPL 3.0 e Common Public Attribution License 1.0. Todas essas licenças exigiram que projetos derivados sejam licenciados pelos mesmos termos, com exceção de alguns casos onde a especificação da licença vem acompanhada da expressão *or-later* (do inglês, ou posterior), sinalizando que a licença de projetos derivados pode ser uma versão mais recente da licença de sua dependência.

A questão das cláusulas de *copyleft* ao combinar licenças de projeto e bibliotecas foi de extrema importância para a comunidade de software aberto. Destacou-se o desafio enfrentado pelos desenvolvedores ao tentar criar projetos que possam aproveitar ao máximo as vantagens do código aberto, como a reutilização de bibliotecas, enquanto ainda respeitam as condições específicas de licenciamento de cada componente. Essas limitações podem afetar a viabilidade e a escalabilidade de projetos de código aberto, bem como a facilidade de colaboração entre desenvolvedores e comunidades. Portanto, compreender e abordar as complexidades das cláusulas de *copyleft* tornou-se essencial para promover o crescimento e a sustentabilidade da comunidade de software aberto.

### 5.4.2. Nível de adoção de bibliotecas em projetos de software aberto (RQ2)

A pesquisa dedicou-se à análise das bibliotecas mais populares e amplamente adotadas em projetos de software aberto, abrangendo tanto o ambiente de desenvolvimento quanto o de produção. A Tabela 1 destaca a frequência das principais bibliotecas. Optou-se por omitir da tabela aquelas bibliotecas cuja frequência foi inferior a 25%, concentrando-se nas mais significativas em termos de adoção.

A popularidade das bibliotecas *eslint*, *typescript* e *prettier* na comunidade de desenvolvimento pode ser explicada por diversos fatores que contribuem para sua ampla adoção. O *eslint*, destaca-se ao identificar erros e inconsistências nos padrões de código, aprimorando assim a clareza e a legibilidade do código. O *typescript*, oferece a vantagem da tipagem estática, auxiliando na prevenção de erros comuns durante a compilação, o que proporciona um ambiente de desenvolvimento mais seguro. O *prettier* se destaca ao automatizar a formatação do código, poupando tempo e mantendo a consistência no estilo de codificação. Em resumo, essas ferramentas compartilham a missão de aprimorar a qualidade e a manutenção do código, tornando-se escolhas valiosas para os desenvolvedores.

As bibliotecas com licenças que, por algum motivo, foram violadas atingiram, no máximo, uma frequência de apenas 0,5% na amostra. Apesar de licenças que possuem

| Biblioteca | Licença | Clientes | Frequência |
|---|---|---|---|
| eslint | MIT | 634 | 63.4% |
| typescript | Apache-2.0 | 516 | 51.6% |
| prettier | MIT | 478 | 47.8% |
| husky | MIT | 326 | 32.6% |
| eslint-plugin-import | MIT | 305 | 30.5% |
| @types/node | MIT | 302 | 30.2% |
| @babel/core | MIT | 285 | 28.5% |
| jest | MIT | 283 | 28.3% |
| @typescript-eslint/parser | BSD-2-Clause | 277 | 27.7% |
| @typescript-eslint/eslint-plugin | MIT | 277 | 27.7% |
| lint-staged | MIT | 254 | 25.4% |
| rimraf | ISC | 253 | 25.3% |
| react | MIT | 252 | 25.2% |
| eslint-config-prettier | MIT | 252 | 25.2% |

**Tabela 1. Bibliotecas amplamente utilizadas. Fonte: Dados da pesquisa**

cláusulas de *copyleft* serem menos utilizadas, indicou-se que a vasta maioria dos projetos respeitam essas licenças, evitando potenciais implicações legais e fortalecendo os princípios fundamentais da comunidade de código aberto. Essa possível descoberta demonstrou que a comunidade de desenvolvedores, de forma geral, está comprometida em manter a integridade e os valores do software aberto, ao mesmo tempo em que aproveita as vantagens das bibliotecas mais amplamente adotadas para impulsionar o reúso.

### 5.4.3. Boas práticas para mitigação da incompatibilidade entre licenças de projeto e bibliotecas (RQ3)

A incompatibilidade entre licenças de projeto e bibliotecas representa um desafio significativo no desenvolvimento de software de código aberto, notadamente quando cláusulas de *copyleft* estão envolvidas. A Figura 2, entendida como umas das principais contribuições do trabalho, exibe o diagrama de procedimentos recomendados para orientar a seleção de bibliotecas a serem integradas em projetos de software. A omissão da prática de consulta a assessoria jurídica especializada é justificada, uma vez que pode ser aplicada em qualquer estágio do diagrama, sempre que surgirem dúvidas relacionadas a possíveis violações.

A seguir, são apresentadas algumas das boas práticas que podem ser empregadas para mitigar essas incompatibilidades:

- **Seleção criteriosa de bibliotecas**: um passo inicial crucial consiste em realizar uma escolha criteriosa das bibliotecas a serem incorporadas ao projeto. Isso implica em buscar bibliotecas que adotem licenças compatíveis com a licença do projeto principal, considerando as cláusulas de *copyleft* e outros termos relevantes. Yun et al. (2017) propõem um processo de governança de OSS que inclui a inspeção de arquivos de licença. Esta inspeção pode ser realizada manualmente ou usando ferramentas automatizadas, e tem como objetivo identificar as licenças

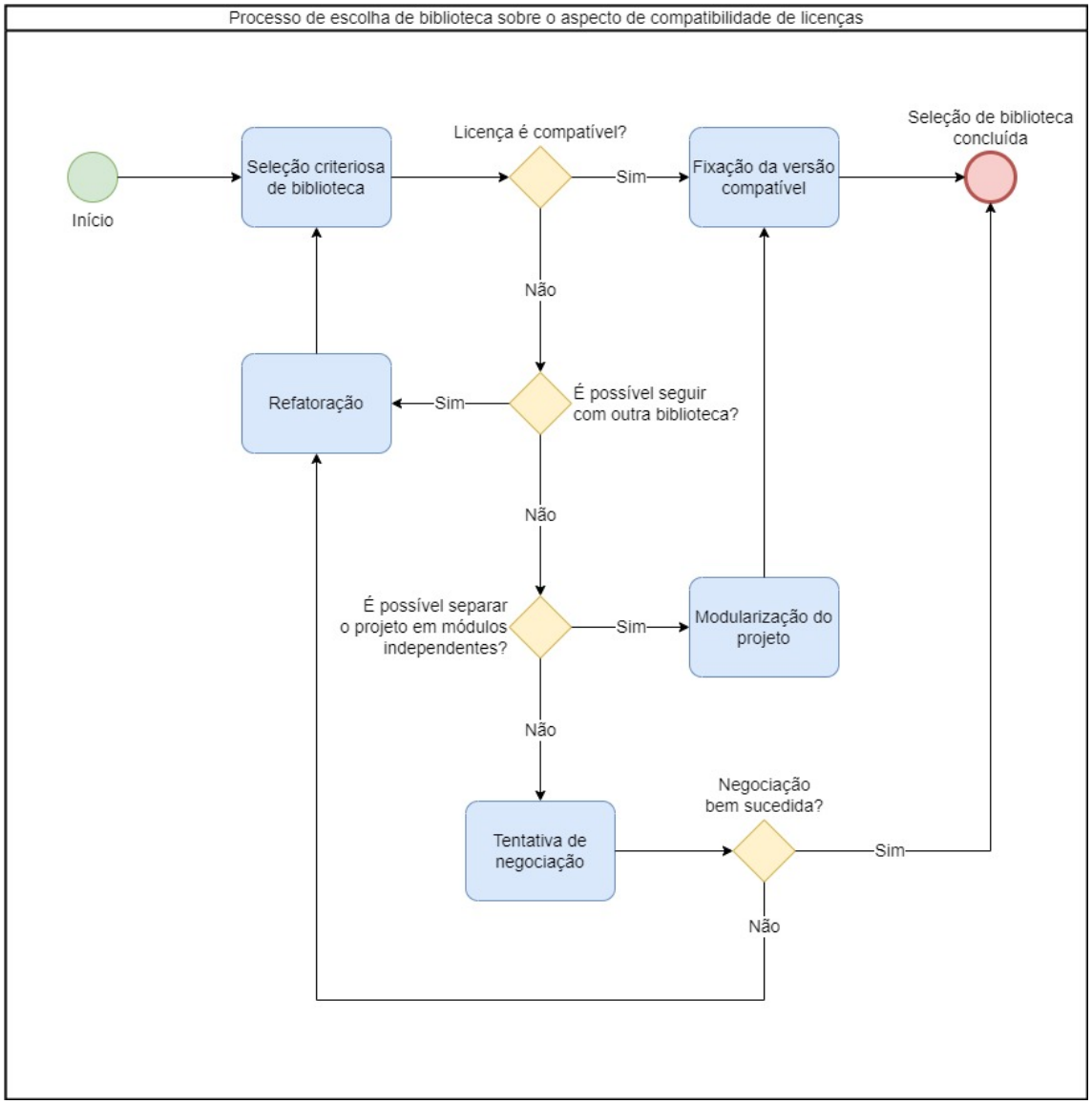


**Figura 2. Diagrama do processo de escolha de bibliotecas sobre o aspecto de compatibilidade de licenças.**

de todos os projetos OSS usados no desenvolvimento do software.

- **Atenção as versões**: projetos e bibliotecas sofrem atualizações e suas licenças podem ser afetadas. Por isso, é importante se atentar a versão da dependência que está utilizando. É recomendável estabelecer requisitos de versão específicas no projeto, garantindo a compatibilidade, pois uma vez publicado um sistema sob uma versão de licença, você sempre poderá usar esse sistema sob os termos dessa licença [Laurent 2004]. Em caso de atualização da licença do projeto, deve-se realizar uma nova verificação para averiguar se a biblioteca tem uma versão compatível.
- **Desenvolvimento modular**: a divisão do projeto em módulos independentes, cada um com sua licença específica, pode ser uma estratégia para contornar a incompatibilidade entre licenças. Essa abordagem mantém separadas as partes do projeto que possam ser afetadas por diferentes licenças, simplificando assim o

cumprimento dos termos de licenciamento.

- **Negociação de acordos personalizados**: em casos excepcionais, torna-se viável estabelecer contato com os detentores dos direitos autorais das bibliotecas e buscar acordos personalizados que permitam o uso da biblioteca em questão em um projeto com licença diferente. Essas negociações podem ser complexas, mas podem resultar em soluções adaptadas a cenários específicos.
- **Refatoração ou substituição de bibliotecas**: quando a incompatibilidade entre as licenças é insuperável, a possibilidade de refatorar partes da biblioteca ou substituí-la por alternativas com licenças mais compatíveis pode ser considerada. Embora possa ser uma tarefa árdua, essa abordagem permite manter a conformidade com as licenças e preservar a integridade do projeto.
- **Assessoria jurídica especializada**: em casos complexos ou quando há incertezas jurídicas, é altamente recomendável buscar assessoria jurídica especializada em direito de software. Apesar de não ser necessário ter conhecimentos específicos em leis para a escolha de uma licença [Laurent 2004], advogados com experiência nessa área podem oferecer orientações específicas e essenciais para assegurar o cumprimento das exigências legais.

## 6. Ameaças à Validade

No que diz respeito à ameaça à validade interna, as amostras foram selecionadas com critérios específicos, como a presença do arquivo `package.json` e o uso das linguagens JavaScript ou TypeScript, excluindo projetos que não acatassem a esses critérios. Isso resultou na escolha de repositórios que constituem uma amostra específica, mas representativa da comunidade de código aberto. No entanto, tal seleção impôs limitações à generalização dos resultados obtidos que podem não aplicar-se a todos os tipos de projetos de código aberto.

Quanto à ameaça à validade externa, os resultados obtidos com base em repositórios específicos do GitHub podem não aplicar-se a outras plataformas de desenvolvimento de código aberto. Além disso, as condições e práticas de licenciamento podem mudar ao longo do tempo.

A ameaça à validade de conclusão vincula-se à possibilidade de interpretação inadequada e à elaboração de conclusões incorretas com base nesses resultados. Tratando-se da identificação de licenças incompatíveis, foi fundamental não apenas destacar a existência dessas incompatibilidades, mas também compreender o impacto real que elas podem ter em projetos de código aberto. Realizou-se uma análise detalhada das implicações concretas e as incompatibilidades entre as licenças identificadas. Entretanto, nem sempre foi possível quantificar o impacto real dessas incompatibilidades nos projetos de código aberto.

## 7. Conclusões

Neste estudo, foram investigados os desafios relacionados ao licenciamento em projetos de software de código aberto, com ênfase três questões centrais. Os resultados obtidos revelaram problemas comuns que giram em torno da compatibilidade entre as licenças de código aberto e suas dependências, destacando o *copyleft* como o único termo violado no escopo da pesquisa. Além disso, foram propostas boas práticas e um processo para mitigar as preocupações relacionadas a violações de licenças em projetos de código aberto.

Paralelamente, foram identificadas as bibliotecas mais amplamente adotadas em tais projetos: eslint (63,4%), typescript (51,6%) e o prettier (47,8%). O somatório ultrapassa 100%, pois os projetos usualmente utilizam mais de uma biblioteca.

Este trabalho se alinha com pesquisas correlatas, seja utilizando mineração de dados no GitHub [Golubev et al. 2020] e uma matriz de compatibilidade [Liu et al. 2021], bem como se mostrando complementar, sugerindo passos a serem executados na escolha de uma biblioteca [Yun et al. 2017] e trazendo dados acerca da violação e uso de bibliotecas e licenças [Higashi et al. 2022, Liu et al. 2021]. Contudo, vale ressaltar que este estudo também reconhece suas próprias limitações, como a restrição da identificação de licenças através do serviço do NPM ou repositório no GitHub e a falta de definição de uma licença específica em alguns projetos, dificultando a identificação de possíveis violações. Além disso, a análise restrita aos projetos desenvolvidos em JavaScript ou TypeScript que utilizam Node.JS pode ter impactado a compreensão do alcance real dessas questões em projetos de código aberto.

Como resultado, evidenciou-se uma melhor compreensão do cenário atual de reutilização de bibliotecas em projetos de código aberto hospedados no GitHub, ao identificar as principais causas de violação de licenças entre projetos e suas dependências, além de propor soluções para mitigar os problemas encontrados. Espera-se que esse estudo possa ajudar os desenvolvedores de software a identificar e tratar adequadamente as violações de licenças em seus projetos.

Para pesquisas futuras, recomenda-se a exploração de abordagens mais abrangentes que incluam diferentes linguagens de programação e ambientes de desenvolvimento. Além disso, é crucial avaliar a eficácia das práticas propostas e considerar a viabilidade de mecanismos automatizados para garantir a conformidade com as licenças de bibliotecas em projetos de software. Essas iniciativas de pesquisa têm o potencial de contribuir significativamente para aprimorar a gestão de licenças e promover maior compatibilidade no desenvolvimento de código aberto, destacando-se como uma possível ferramenta automatizada para análise de licenças.

## Pacote de Replicação

O pacote de replicação deste trabalho encontra-se disponível em:

`https://github.com/ICEI-PUC-Minas-PPLES-TI/plf-es-2023-1-tcci-0393100-pes-leticia-e-octavio`

## Referências


Alnusair, A., Rawashdeh, M., Alhamid, M. F., Hossain, M. A., and Muhammad, G. (2016). Reusing software libraries using semantic graphs. In *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*, pages 531–540.

Ballhausen, M. (2019). Free and open source software licenses explained. *Computer*, 52(6):82–86.

Duan, R., Bijlani, A., Xu, M., Kim, T., and Lee, W. (2017). 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*, CCS '17, page 2169–2185, New York, NY, USA. Association for Computing Machinery.

Dutta, S., Garbervetsky, D., Lahiri, S. K., and Schäfer, M. (2022). Inspectjs: Leveraging code similarity and user-feedback for effective taint specification inference for javascript. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, ICSE-SEIP '22, page 165–174, New York, NY, USA. Association for Computing Machinery.

Feng, M., Mao, W., Yuan, Z., Xiao, Y., Ban, G., Wang, W., Wang, S., Tang, Q., Xu, J., Su, H., Liu, B., and Huo, W. (2019). Open-source license violations of binary software at large scale. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 564–568.

Golubev, Y., Eliseeva, M., Povarov, N., and Bryksin, T. (2020). A study of potential code borrowing and license violations in java projects on github. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, page 54–64, New York, NY, USA. Association for Computing Machinery.

Hemel, A., Kalleberg, K. T., Vermaas, R., and Dolstra, E. (2011). Finding software license violations through binary code clone detection. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, page 63–72, New York, NY, USA. Association for Computing Machinery.

Higashi, Y., Fukui, K., Yutaro, K., and Ohira, M. (2022). A preliminary analysis of gpl-related license violations in docker images. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 444–448.

Kapitsaki, G. M. and Charalambous, G. (2021). Modeling and recommending open source licenses with findosslicense. *IEEE Transactions on Software Engineering*, 47(5):919–935.

Kapitsaki, G. M. and Paschalides, D. (2017). Identifying terms in open source software license texts. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, pages 540–545.

Kashima, Y., Hayase, Y., Yoshida, N., Manabe, Y., and Inoue, K. (2011). An investigation into the impact of software licenses on copy-and-paste reuse among oss projects. In *2011 18th Working Conference on Reverse Engineering*, pages 28–32.

Kim, D., Kim, E., Cha, S. K., Son, S., and Kim, Y. (2022). Revisiting binary code similarity analysis using interpretable feature engineering and lessons learned. *IEEE Transactions on Software Engineering*, pages 1–23.

Lampropoulos, A., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., and Stamelos, I. (2018). React - a process for improving open-source software reuse. In *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 251–254.

Laurent, A. M. S. (2004). *Understanding open source and free software licensing: guide to navigating licensing issues in existing & new software*. "O'Reilly Media, Inc.".

Liu, Z., Zhang, Z., Wang, Z., Peng, J., and Wu, S. (2021). Choosing an open source license based on software dependencies. In *2021 IEEE International Conference on Software Engineering and Artificial Intelligence (SEAI)*, pages 30–36.

López de la Mora, F. and Nadi, S. (2018). Which library should i use?: A metric-based comparison of software libraries. In *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*, pages 37–40.

Monden, A., Okahara, S., Manabe, Y., and Matsumoto, K. (2011). Guilty or not guilty: Using clone metrics to determine open source licensing violations. *IEEE Software*, 28(2):42–47.

Nepomuceno, F., Castro, J., Moreira, A., Almeida, E., and Gimenes, I. (2021). Requirements engineering and enterprise architecture-based software discovery and reuse. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–19.

Reid, D., Jahanshahi, M., and Mockus, A. (2022). The extent of orphan vulnerabilities from code reuse in open source software. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, ICSE '22, page 2104–2115, New York, NY, USA. Association for Computing Machinery.

Taivalsaari, A., Mikkonen, T., and Mäkitalo, N. (2019). Programming the tip of the iceberg: Software reuse in the 21st century. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 108–112.

Tung, Y.-H., Chuang, C.-J., and Shan, H.-L. (2014). A framework of code reuse in open source software. In *The 16th Asia-Pacific Network Operations and Management Symposium*, pages 1–6.

Woo, S., Park, S., Kim, S., Lee, H., and Oh, H. (2021). CENTRIS: A precise and scalable approach for identifying modified open-source software reuse. *CoRR*, abs/2102.06182.

Xu, X., Zheng, Q., Yan, Z., Fan, M., Jia, A., and Liu, T. (2021). Interpretation-enabled software reuse detection based on a multi-level birthmark model.

Ye, Y. (2009). Peer to peer support for the reuse of open source software libraries. In *2009 IEEE International Conference on Information Reuse & Integration*, pages 284–289.

Yun, H. Y., Joe, Y. J., and Shin, D. M. (2017). Method of license compliance of open source software governance. In *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 83–86.

## A. Apêndice - Lista de Licenças OSI

A *Open Source Initiative* (OSI) é uma organização dedicada à promoção e defesa do software de código aberto por meio da certificação de licenças de código aberto. As licenças reconhecidas pela OSI são:

- 1-clause BSD License
- Academic Free License ("AFL") v. 3.0
- Adaptive Public License 1.0
- Apache License, Version 2.0
- Apache Software License, version 1.1 (Apache-1.1)
- Apple Public Source License 2.0
- Artistic License (Perl) 1.0
- Artistic License 1.0 (Artistic-1.0)
- Artistic License 2.0
- Attribution Assurance License
- Boost Software License 1.0
- BSD+Patent
- Cea Cnrs Inria Logiciel Libre License, version 2.1 (CECILL-2.1)

- CERN Open Hardware Licence Version 2 – Permissive
- CERN Open Hardware Licence Version 2 – Strongly Reciprocal
- CERN Open Hardware Licence Version 2 – Weakly Reciprocal
- COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0
- Common Development and Distribution License 1.0
- Common Public Attribution License Version 1.0 (CPAL-1.0)
- Common Public License Version 1.0
- Common Public License, version 1.0 (CPL-1.0)
- Computer Associates Trusted Open Source License 1.1 (CATOSL-1.1)
- Cryptographic Autonomy License version 1.0 (CAL-1.0)
- CUA Office Public License (CUA-OPL-1.0)
- Eclipse Public License -v 1.0
- Eclipse Public License 1.0 (EPL-1.0)
- Eclipse Public License version 2.0
- eCos License version 2.0
- Educational Community License, Version 1.0 (ECL-1.0)
- Educational Community License, Version 2.0 (ECL-2.0)
- Eiffel Forum License, version 1 (EFL-1.0)
- Eiffel Forum License, Version 2
- Entessa Public License Version. 1.0 (Entessa)
- EU DataGrid Software License (EUDatagrid)
- European Union Public License, version 1.2 (EUPL-1.2)
- Fair License (Fair)
- Frameworx License 1.0 (Frameworx-1.0)
- GNU Affero General Public License version 3
- GNU General Public License version 2
- GNU General Public License version 3
- GNU General Public License, version 1
- GNU Lesser General Public License version 2.1
- GNU Lesser General Public License version 3
- GNU LGPL
- GNU Library General Public License version 2
- Historical Permission Notice and Disclaimer (HPND)
- IBM Public License Version 1.0 (IPL-1.0)
- Intel Open Source License (Intel)
- IPA Font License (IPA)
- ISC License (ISC)
- Jabber Open Source License
- JAM License
- LaTeX Project Public License, Version 1.3c (LPPL-1.3c)
- Lawrence Berkeley National Labs BSD Variant License (BSD-3-Clause-LBNL)
- Licence Libre du Québec — Permissive (LiLiQ-P) version 1.1
- Licence Libre du Québec — Réciprocité (LiLiQ-R) version 1.1
- Licence Libre du Québec — Réciprocité forte (LiLiQ-R+) version 1.1
- Lucent Public License Version 1.02 (LPL-1.02)
- Lucent Public License, Plan 9, version 1.0 (LPL-1.0)
- Microsoft Public License (MS-PL)
- Microsoft Reciprocal License (MS-RL)
- MirOS License (MirOS)
- MIT No Attribution License
- MITRE Collaborative Virtual Workspace License (CVW)
- Motosoto Open Source License — Version 0.9.1 (Motosoto)
- Mozilla Public License (MPL), version 1.0 (MPL-1.0)

- Mozilla Public License 1.1 (MPL-1.1)
- Mozilla Public License 2.0 (MPL-2.0)
- Mulan Permissive Software License v2 (MulanPSL — 2.0)
- Multics License (Multics)
- NASA Open Source Agreement v1.3 (NASA-1.3)
- NAUMEN Public License (Naumen)
- Nokia Open Source License Version 1.0a (NOKIA)
- Non-Profit Open Software License version 3.0 (NPOSL-3.0)
- NTP License (NTP)
- Open Group Test Suite License (OGTSL)
- Open Logistics Foundation License v1.3
- Open Software License 2.1 (OSL-2.1)
- Open Software License, version 1.0 (OSL-1.0)
- OpenLDAP Public License Version 2.8 (OLDAP-2.8)
- OSET Public License version 2.1
- PHP License 3.0 (PHP-3.0)
- PHP License 3.01 (PHP-3.01)
- Python License (Python-2.0)
- RealNetworks Public Source License Version 1.0 (RPSL-1.0)
- Reciprocal Public License 1.5 (RPL-1.5)
- Reciprocal Public License, version 1.1
- SIL OPEN FONT LICENSE (OFL-1.1)
- Simple Public License (SimPL-2.0)
- Sun Industry Standards Source License (SISSL)
- Sun Public License, Version 1.0 (SPL-1.0)
- The 2-Clause BSD License
- The 3-Clause BSD License
- The CNRI portion of the multi-part Python License (CNRI-Python)
- The European Union Public License, version 1.1
- The MIT License
- The Nethack General Public License (NGPL)
- The OCLC Research Public License 2.0 License (OCLC-2.0)
- The Open Software License 3.0 (OSL-3.0)
- The PostgreSQL Licence (PostgreSQL)
- The Q Public License Version (QPL-1.0)
- The Ricoh Source Code Public License (RSCPL)
- The Sleepycat License (Sleepycat)
- The Sybase Open Source Licence (Watcom-1.0)
- The Universal Permissive License (UPL), Version 1.0
- The University of Illinois/NCSA Open Source License (NCSA)
- The Unlicense
- The Vovida Software License v. 1.0 (VSL-1.0)
- The W3C® SOFTWARE NOTICE AND LICENSE (W3C)
- The wxWindows Library Licence (WXwindows)
- The X.Net, Inc. License (Xnet)
- The zlib/libpng License (Zlib)
- Unicode, Inc. License Agreement — Data Files and Software
- Upstream Compatibility License v1.0
- Zero-Clause BSD (0BSD)
- Zope Public License 2.1
- Zope Public License Ver.2.0 (ZPL-2.0)

## B. Apêndice - Matriz de compatibilidade

| project_name | project_license | version_spec | depends_on | dependency_name | dependency_version | dependency_license |
|---|---|---|---|---|---|---|
| botpress/botpress | mit | @bpinternal/readiness@0.0.1 | production | @bpinternal/readiness | 0.0.1 | agpl-3.0 |
| marktext/marktext | mit | cfonts@^2.10.1 | development | cfonts | 2.10.1 | gpl-2.0 |
| agalwood/Motrix | mit | cfonts@^3.2.0 | development | cfonts | 3.2.0 | gpl-3.0-or-later |
| xiandanin/magnetW | gpl-3.0 | cfonts@^2.1.2 | development | cfonts | 2.10.1 | gpl-2.0 |
| Zettlr/Zettlr | gpl-3.0 | citeproc@^2.4.63 | production | citeproc | 2.4.63 | (cpal-1.0 or agpl-1.0) |
| date-fns/date-fns | mit | cloc@^2.2.0 | development | cloc | 2.11.0 | gpl-2.0 |
| CesiumGS/cesium | apache-2.0 | cloc@^2.8.0 | development | cloc | 2.11.0 | gpl-2.0 |
| nextui-org/nextui | mit | commitlint-plugin-function-rules@^1.7.1 | development | commitlint-plugin-function-rules | 1.7.1 | gpl-3.0-or-later |
| developit/htm | apache-2.0 | eslint-config-developit@^1.1.1 | development | eslint-config-developit | 1.2.0 | gpl-3.0 |
| developit/microbundle | mit | eslint-config-developit@^1.2.0 | development | eslint-config-developit | 1.2.0 | gpl-3.0 |
| developit/mitt | mit | eslint-config-developit@^1.2.0 | development | eslint-config-developit | 1.2.0 | gpl-3.0 |
| keplergl/kepler.gl | mit | eslint-config-developit@^1.2.0 | production | eslint-config-developit | 1.2.0 | gpl-3.0 |
| preactjs/preact | mit | eslint-config-developit@^1.1.1 | development | eslint-config-developit | 1.2.0 | gpl-3.0 |
| wulkano/Kap | mit | ffmpeg-static@^4.4.1 | production | ffmpeg-static | 4.4.1 | gpl-3.0-or-later |
| juice-shop/juice-shop | mit | fuzzball@^1.3.0 | production | fuzzball | 1.4.0 | gpl-2.0 |
| r-spacex/SpaceX-API | apache-2.0 | fuzzball@^2.1.2 | production | fuzzball | 2.1.2 | gpl-2.0 |
| simple-icons/simple-icons | cc0-1.0 | get-relative-luminance@1.0.0 | development | get-relative-luminance | 1.0.0 | gpl-3.0 |
| showdownjs/showdown | mit | grunt-endline@^0.7.0 | development | grunt-endline | 0.7.0 | gpl-3.0 |
| carhartl/jquery-cookie | mit | gzip-js@∼0.3.0 | development | gzip-js | 0.3.2 | gpl-2.0 |
| jquery/jquery | mit | gzip-js@0.3.2 | development | gzip-js | 0.3.2 | gpl-2.0 |
| js-cookie/js-cookie | mit | gzip-js@^0.3.2 | development | gzip-js | 0.3.2 | gpl-2.0 |
| standard/standard | mit | hallmark@^4.2.0 | development | hallmark | 4.2.0 | gpl-3.0 |
| antfu/vitesse | mit | https-localhost@^4.7.1 | development | https-localhost | 4.7.1 | agpl-3.0 |
| umami-software/umami | mit | is-localhost-ip@^1.4.0 | production | is-localhost-ip | 1.4.0 | gpl-3.0 |
| Leaflet/Leaflet | bsd-2-clause | leafdoc@^2.3.0 | development | leafdoc | 2.3.0 | gpl-3.0 |
| hediet/vscode-drawio | gpl-3.0 | ovsx@^0.1.0-next.e000fdb | development | ovsx | 0.1.0 | epl-2.0 |
| bailicangdu/node-elm | gpl-2.0 | pm2@^2.8.0 | production | pm2 | 2.10.4 | agpl-3.0 |
| cnodejs/nodeclub | mit | pm2@* | production | pm2 | 5.3.0 | agpl-3.0 |
| nylas/nylas-mail | mit | pm2@2.4.0 | development | pm2 | 2.4.0 | agpl-3.0 |
| supabase/supabase | apache-2.0 | prettier-plugin-sql-cst@^0.5.0 | development | prettier-plugin-sql-cst | 0.5.0 | gpl-3.0-or-later |
| jshint/jshint | mit | results-interpreter@∼1.0.0 | development | results-interpreter | 1.0.0 | gpl-3.0 |
| iissnan/hexo-theme-next | mit | stylint@^1.5.9 | development | stylint | 1.5.9 | gpl-3.0 |
| simple-icons/simple-icons | cc0-1.0 | svg-path-segments@1.0.0 | development | svg-path-segments | 1.0.0 | gpl-3.0 |
| dream-num/Luckysheet | mit | vuepress-plugin-code-copy@^1.0.6 | development | vuepress-plugin-code-copy | 1.0.6 | gpl-3.0-or-later |
| Meituan-Dianping/mpvue | mit | weex-vdom-tester@^0.2.0 | development | weex-vdom-tester | 0.2.0 | gpl-3.0 |