СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА ПРОГРАМУВАННЯ ТА МАТЕМАТИКИ

Пояснювальна записка

до магістерської дипломної роботи

(освітньо-кваліфікаційний рівень)

на тему Розробка системи перевірки сумісності ліцензій програмного забезпечення

Виконав: студент 2 курсу, групи ІСТ-21зм

спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)

Садковський Микита Вадимович
(прізвище та ініціали)

Керівник доц., д.т.н. Лифар В. О.
(прізвище та ініціали)

Рецензент доц., к.т.н. Ратов Д. В.
(прізвище та ініціали)

Сєвєродонецьк - 2022 року

ЗМІСТ

РЕФЕРАТ

Пояснювальна записка складається з: 45 сторінок основного тексту, 16 рисунків, 2 таблиці, 1 додатку, 14 використаних джерел.

Метою даної магістерської дипломної роботи є розробка плагіна IntelliJ IDEA для роботи з ліцензіями Java-проєктів. Плагін повинен дозволити визначати допустимі ліцензії для кожного модуля проєкту та повідомляти програмісту про потенційні порушення

Об’єктом дослідження є розробка плагіна IntelliJ IDEA для роботи з ліцензіями Java-проєктів.

Предметом дослідження є методи та засоби проектування плагіна IntelliJ IDEA для роботи з ліцензіями Java-проєктів.

Головні завдання роботи:

1. Огляд предметної області.
2. Провести аналіз існуючих рішень.
3. Розробити архітектуру плагіна.
4. Реалізувати алгоритм контролю ліцензій проєкту:
   – реалізувати процедуру отримання інформації про ліцензії компонент проєкту;
   – реалізувати механізм визначення допустимих ліцензій для модуля проєкту на підставі ліцензій компонентів проєкту.
5. Реалізувати інтерфейс користувача плагіна.
6. Провести перевірку створеного інструменту.

Методи дослідження – аналіз предметної області буде виконаний з акцентом на ієрархічне представлення об'єктів та сформованими логічними зв'язки між процесами.

Ключові слова: ПРОЄКТ, ЛІЦЕНЗІЯ, ПЛАГІН, INTELLIJ IDEA, JAVA**,** IDE

ВСТУП

На сьогоднішній день існує велика кількість проєктів з відкритим вихідним кодом (open source projects). Чотири роки тому, у відкритому доступі на хостингу IT-проєктів GitHub існувало більше 180 000 відкритих проєктів [7], та їх кількість продовжує зростати.

Відкриті ліцензії програмного забезпечення дозволяють розробникам належним чином використовувати, модифікувати та розповсюджувати програмне забезпечення, але за порушення умов ліцензії передбачено відповідальність, що включає великі штрафи. Ліцензування відкритого програмного забезпечення складна область з великою кількістю нюансів. На даний момент налічується понад 450 різних відкритих ліцензій.

Зазвичай програмісти схильні повторно використовувати будь-який код, що знаходиться у відкритому доступі, не звертаючи уваги на його ліцензію. У великих IT-компаніях ліцензуванням займаються спеціальні юристи, які перевіряють ліцензії відкритого программного забезпечення, що використовується у продуктах компанії, і виявляють несумісності. Водночас у звичайних розробників часто не вистачає ресурсів для таких пильних перевірок.

Згідно з результатами дослідження фон Крога існують дві основні причини порушення ліцензій програмного забезпечення з відкритим вихідним кодом. Перша причина полягає в тому, що через обмеження в ресурсах та часі розробники часто повторно використовують сторонній код, щоб не витрачати сили на вирішення простих завдань. Друга причина полягає в тому, що через велику кількість відкритих ліцензій розробники програмного забезпечення часто не розуміють їх умови та відмінності між ними, що згодом може спричинити порушення ліцензування.

Для того, щоб вивчити поточну ситуацію із запозиченням коду та порушенням ліцензій, лабораторія Методів машинного навчання програмної інженерії JetBrains Research провела дослідження потенційних порушень ліцензування у популярних Java-проєктах на GitHub. В рамках дослідження

було виявлено та описано значну кількість відкритих ліцензій, а також складну систему їх сумісності. За результатами дослідження виявилося, що 9,4% методів Java-класів у цих проєктах потенційно могли бути скопійовані з порушенням ліцензування.

Щоб допомогти розробникам не робити таких помилок і не витрачати час на окремі перевірки, можна показувати необхідну інформацію прямо в інтегрованому середовищі розробки (Integrated Development Environment, IDE). Така можливість дозволить визначати сумісні ліцензії проєкту та повідомляти програміста про потенційні порушення. Завдяки цьому розробники зможуть уникати помилок, пов'язаних із ліцензіями, що може знизити кількість порушень ліцензування у відкритих програмних продуктах.

РОЗДІЛ 1 ОГЛЯД ПРЕДМЕТНОЇ ГАЛУЗІ

Відкрите програмне забезпечення (open source software) – це програмне забезпечення з відкритим вихідним кодом. Розробка перших відкритих проєктів почалася наприкінці 70-х – початку 80-х. Один з найбільш відомих таких проєктів - це система комп'ютерної верстки TeX, що розробляється з 1979 відомим вченим і програмістом Дональдом Кнутом [1]. Іншим проєктом стала операційна система GNU, розробка якої розпочалася у 1980 році відомим програмістом Річардом Столлманом. Однак широку популярність open source рух почав набирати після створення в 1998 році організації Open Source Initiative [9] (OSI), яка покликана пропагувати принципи open source розробки та підтримувати відкриті проєкти.

В рамках роботи організації Open Source Initiative було сформульовано визначення open source програмного забезпечення, яке містить вичерпну інформацію про програмне забезпечення, яке можна вважати відкритим. З повним текстом визначення можна ознайомитись на офіційному сайті Open Source Initiative.

Програмне забезпечення, як і інші форми інтелектуальної власності, підлягає ліцензуванню. Далі детально розглянемо ліцензії програмного забезпечення.

1.1 Ліцензії програмного забезпечення

Ліцензія програмного забезпечення – це правовий інструмент, що визначає використання, модифікацію та розповсюдження програмного забезпечення, захищеного авторським правом.

Усі ліцензії програмного забезпечення можна розділити на п'ять різних типів, кожен з яких має свій набір дозволів та обмежень використання.
Крім них існує ще статус “вільний доступ”, яке не є ліцензією, але теж є способом надання прав.

Різні методи передачі прав представлені на рисунку 1.1.

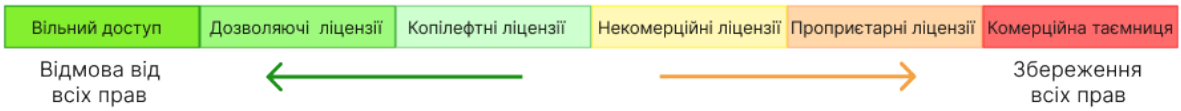

Рис.1.1 – Методи передачі прав на програмне забезпечення.

Варто зазначити, що ліцензії, що входять до категорії “Дозволяючі ліцензії” та “Копілефтні ліцензії”, є відкритими (open source), оскільки програмне забезпечення, ліцензоване таким чином, задовольняє визначення відкритого програмного забезпечення OSI.

Найбільш вичерпний список відкритих ліцензій можна знайти на офіційному сайті відкритого стандарту для передачі відомостей про програмне забезпечення The Software Package Data Exchange (SPDX), який включає відомості про ліцензії та авторські права.

Стисло розглянемо особливості кожного способу передачі прав.

Вільний доступ

Найбільш дозвільним інструментом передачі прав є статус вільний доступ (public domain). Роблячи своє програмне забезпечення з вільним доступом, автори відмовляються від усіх прав та дають можливість розпоряджатися програмним продуктом без жодних обмежень. У тому числі, у третіх осіб з'являється можливість зробити це програмне забезпечення закритим або навіть продати його. Поняття вільного доступу у різних країнах відрізняється, що іноді може призводити до юридичних труднощів.
Щоб уникнути цього, використовуються дозволяючі ліцензії.

Дозволяючі ліцензії

Дозволяючі (permissive) ліцензії надають можливість використовувати, модифікувати та поширювати відповідне програмне забезпечення, включаючи можливість ліцензувати модифікації програмного забезпечення пропрієтарними ліцензіями. Такі ліцензії часто вимагають вказувати оригінальних авторів програмного забезпечення при поширенні модифікованого програмного забезпечення. Крім цього, дозволяючі ліцензії часто включають відмову від будь-яких гарантій програмного забезпечення

та знімають будь-яку відповідальність з авторів продукту. Найбільш поширеними дозвільними ліцензіями є MIT, Apache 2.0 та BSD-3-Clause.

Іноді автори програмного забезпечення хочуть обмежити можливість ліцензування модифікацій продукту. Для цього використовуються копілефтні ліцензії.

Копілефтні ліцензії

Зазвичай виділяють два види копілефтних ліцензій: сильні копілефтні (strong copyleft) та слабкі копілефтні (weak copyleft) ліцензії.

Сильні копілефтні ліцензії – це ліцензії, які, як і дозволяючі, дозволяють вільно модифікувати та поширювати модифікований продукт, однак ліцензія модифікованого продукту має бути такою самою, як і на оригінальному програмному забезпеченні. Сильні копілефтні ліцензії також вимагають вказувати оригінальних авторів на модифікаціях товару. Таким чином, сильні копілефтні ліцензії дозволяють авторам обмежити використання свого програмного продукту та заборонити використання його модифікацій під іншими ліцензіями. Найбільш популярними сильними копілефтними ліцензіями є ліцензії сімейства GNU General Public License (GPL) [4, 13].

Слабкі копілефтні ліцензії відрізняються від сильних різними послабленнями в обмеженнях використання, модифікації та розповсюдження програмного продукту Найбільш відомим прикладом таких ліцензій є ліцензії сімейства GNU Lesser General Public License (LGPL). Вони, як і ліцензії сімейства GPL, дозволяють змінювати і поширювати модифікований код лише за умові збереження цієї ліцензії, однак при використанні коду у якості бібліотеки виконання цієї вимоги не обов'язкове.

Некомерційні ліцензії

Некомерційні (noncommercial) ліцензії надають права на модифікацію та розповсюдження програмного забезпечення, але тільки для некомерційної діяльності. Часто некомерційні ліцензії є копілефтними ліцензіями. Прикладами ліцензій такого типу є Java Research License (JRL) [5] та Aladdin

Free Public License (AFPL).

Пропрієтарні ліцензії

Пропрієтарні (proprietary) ліцензії – це ліцензії, які забезпечують автору програмного забезпечення монополію з його використання, копіювання і модифікацію. Програмне забезпечення, що знаходиться під цим типом ліцензій, є пропрієтарним. Вихідний код таких продуктів зазвичай закритий, але це не обов'язково.

Комерційна таємниця

Комерційна таємниця (trade secret) - інформація, яка має комерційну цінність через її невідомість третім особам, до якої немає вільного доступу на законній підставі. Програмне забезпечення, яке є комерційною таємницею, ніде не публікується. Вихідний код таких продуктів зберігається в таємниці, а за розголошення інформації про внутрішній устрій програмного забезпечення, що потрапляє під комерційну таємницю, передбачено відповідальність.

Сумісність типів ліцензій

Одним з найважливіших аспектів ліцензування програмного забезпечення є сумісність різних типів ліцензій. Загалом сумісність типів ліцензій представлена на рисунку 1.2.

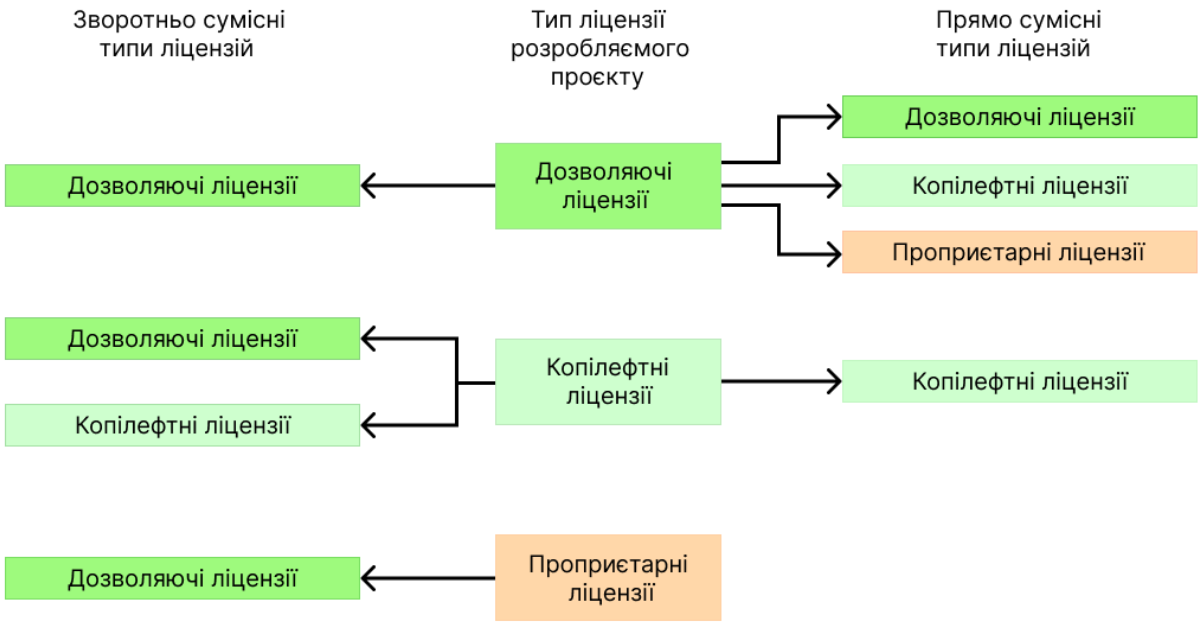

Рис. 1.2 – Сумісність типів ліцензій.

Пряма сумісність визначає, якою ліцензією можуть ліцензуватися похідні продукти, які використовують програмне забезпечення автора. Зворотна сумісність визначає, яку ліцензію автор може використовувати для ліцензування свого програмного продукту, виходячи з ліцензій програмного забезпечення, яке він використав.

Наприклад, у програмному проєкті, який має дозволяючу ліцензію можна використовувати лише програмне забезпечення, яке знаходиться під дозволяючою ліцензією. Водночас модифікації такого проєкту можна поширювати під різними ліцензіями, у тому числі пропрієтарними.

Слід зазначити, що у діаграмі відсутний тип некомерційних ліцензій, оскільки йому не можна визначити сумісність із іншими типами ліцензій. Сумісність із некомерційними ліцензіями необхідно розглядати окремо для кожної ліцензії, тому в рамках даної роботи вони розглядатися не будуть.

Сумісність відкритих ліцензій

Особливу увагу слід приділити сумісності відкритих ліцензій. Важливо, що використання сторонньої ліцензованої бібліотеки є повторним використанням коду цієї бібліотеки, а отже, вимагає дотримання всіх ліцензійних обмежень. Тому одним із завдань плагіна, що розробляється, є

пошук потенційних порушень ліцензійних обмежень та повідомлення розробника про знайдені порушення.

На рисунку 3 детально зображено сумісність найпопулярніших відкритих ліцензій.

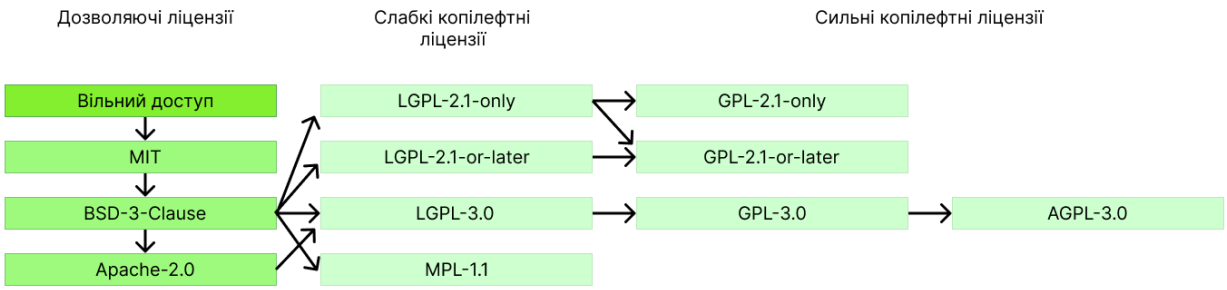

Рис. 1.3 – Сумісність популярних відкритих ліцензій.

Стрілки між ліцензіями позначають їхню пряму сумісність. Важливо, що кожна розглянута на рисунку 3 ліцензія сумісна сама із собою. Крім цього, на діаграмі мається на увазі транзитивна сумісність ліцензій. Так наприклад, у продукті з ліцензією GPL-3.0 можна використовувати програмне забезпечення під ліцензією Apache-2.0, але не навпаки.

1.2 Дослідження порушень ліцензування ПЗ

Існує низка наукових праць, присвячених дослідженню порушень ліцензування у відкритому програмному забезпеченні. Одним з таких досліджень є робота Даніеля Германа та інших [2], в результаті якої було виявлено несумісність ліцензій залежностей пакетів дистрибутива Fedora-12 GNU/Linux. Код під ліцензіями GPL часто некоректно використовувався у пакетах із більш дозвільними ліцензіями.

У роботі Романського та інших [11] досліджено можливі порушення ліцензій під час міграції коду між модулями Python і постами сервісу StackOverflow, а також зроблено висновок про те, що пости StackOverflow часто містять код, ліцензія якого несумісна з ліцензією платформи.

Іншою подібною роботою є емпіричне дослідження несумісності ліцензій у відкритому програмному забезпеченні Арунеш Матура та інших. В

рамках цього дослідження були вивчені відкриті проєкти хостингу репозиторіїв Google Code, серед яких було виявлено та докладно описано низку проблем, пов'язаних з несумісністю ліцензій.

Хун'ю Чжан та ін. провели дослідження, присвячене автоматичній перевірці сумісності ліцензій, у ході якого було розроблено інструмент для визначення сумісності ліцензій проєктів з урахуванням сервісу Google Code Search. За результатами експериментів з даним інструментом було виявлено кілька нових порушень ліцензій у відкритому програмному забезпеченні.

Перелічені дослідження показують, що ліцензування відкритого програмного забезпечення є нетривіальною областю. Незважаючи на те, що написано безліч статей і створено цілу низку інструментів для автоматичного визначення ліцензії файлу та виявлення несумісності ліцензій, ця область потребує подальших досліджень.

РОЗДІЛ 2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

На даний момент існує кілька популярних інструментів, які надають можливість витягувати інформацію про ліцензії з Java-проєктів, включаючи інформацію про ліцензії залежностей проєкту та файли з вихідним кодом. Такі інструменти корисні для невеликих компаній та незалежних розробників, які не мають достатньо ресурсів для використання послуг професійних юристів для роботи з ліцензіями проєктів.

Після аналізу інструментів для роботи з ліцензіями проєктів було виділено такі параметри для порівняння:

• компоненти проєкту, які аналізують інструмент;

• тип інструменту (консольна програма, графічна програма, веб-сервіс, плагін для системи складання проєкту);

• здатність рекомендації основної ліцензії проєкту;

• здатність визначення потенційних порушень ліцензування;

• Відкритість вихідного коду.

Можливість витягувати інформацію не лише про основну ліцензію проєкту, а й про ліцензії залежностей та файлів з вихідним кодом безпосередньо визначає можливості самого інструменту.

Тип інструменту визначає зручність його використання. Звичайному користувачеві простіше працювати з інструментом, який має наочний графічний інтерфейс або інтегрований із популярною системою складання проєктів. Консольні програми зазвичай незручні для швидкого освоєння та використання. Часто їхні команди мають безліч параметрів, що потребує часу на їх вивчення та часто відштовхує програмістів від використання таких інструментів.

Незважаючи на те, що список ліцензій всіх компонентів проєкту сам по собі є корисною інформацією для програміста, йому, як і раніше, необхідно розбиратися у виборі коректної ліцензії для свого проєкту. Інструменти для роботи з ліцензіями проєктів можуть спростити цей процес

та надати користувачеві підказки, які допоможуть коректно вибрати ліцензію та уникнути всіх складнощів, пов'язаних із ліцензуванням програмного забезпечення.

Визначення потенційних порушень ліцензування є однією із цілей інструментів для роботи з ліцензіями проєктів. Ця можливість дозволяє попередити програміста про порушення ліцензування під час використання бібліотеки або файлів з вихідним кодом, ліцензії яких не сумісні з основною ліцензією проєкту, що розробляється.

Відкритий вихідний код інструмента дозволяє використовувати його безкоштовно, що актуально для незалежних розробників та невеликих компаній. Крім того, у сторонніх розробників з'являється можливість адаптувати код відкритого інструменту під свої потреби, що потенційно збільшує область застосування даного інструменту.

У таблиці 2.1 наведено порівняння деяких популярних інструментів для роботи з ліцензіями Java-проєктів.

Порівняння показало, що більшість інструментів працюють лише з основними ліцензіями проєктів та їх вихідним кодом, не опрацьовуючи ліцензії залежностей проєкту. Жоден із розглянутих безкоштовних інструментів не надає користувачеві рекомендації щодо вибору основної ліцензії проєкту та не повідомляє про несумісні ліцензії. Крім того, жоден з інструментів не вбудований у інтегроване середовище розробки (IDE).

Інтеграція інструменту для роботи з ліцензіями з популярною IDE полегшує встановлення та налаштування інструменту, а також дає можливість програмісту користуватися таким інструментом, не виходячи з звичного редактора коду, що підвищує мотивацію програміста приділяти необхідну увагу ліцензіям проєкту. Завдяки інтеграції інструмента в IDE середовище розробки може у фоновому режимі сканувати проєкт на наявність ліцензій і відстежувати потенційні порушення. Все це дозволить програмісту впевнитись у тому, що всі ліцензійні вимоги до проєкту дотримані.

Інстумент	Ліцензії	Тип	Рекомендації	Порушення	Open
askalono	Проєкт, вихідний код	Консольний додаток	-	-	+
FOSSology	Проєкт, залежності, вихідний код, бінарні файли	Веб-сервіс	+	+	-
go-license-detector	Проєкт, вихідний код	Консольний додаток	-	-	+
license-checker	Проєкт, вихідний код	Консольний додаток	-	-	+
licensee	Проєкт	Консольний додаток	-	-	+
scancode-toolkit	Проєкт, вихідний код, бінарні файли	Консольний додаток	-	-	+
FOSSA	Проєкт, залежності, вихідний код, бінарні	Веб-сервіс	+	+	-

	файли				
License Maven Plugin	Вихідний код	Maven плагін	-	-	+
Gradle License Plugin	Проєкт, залежності	Gradle плагін	-	-	+
Gradle License Report	Залежності	Gradle плагін	-	-	+

Таблиця 2.1 – Популярні інструменти для роботи з ліцензіями проєктів

РОЗДІЛ 3 АРХІТЕКТУРА ПЛАГІНА

Плагін License Detector Plugin (LDP) для роботи з ліцензіями Java проєктів в IntelliJ IDEA розроблено в рамках проєкту дослідницької лабораторії машинного навчання в галузі програмної інженерії JetBrains Research. Важливою особливістю розробки даного плагіна є небажаність використання сторонніх щодо компанії JetBrains інструментів для детектування ліцензій проєкту. Тому всі залежності проєкту є або продуктами, або внутрішніми розробками компанії JetBrains.

LDP написаний мовою Kotlin і підтримує роботу з 16 найбільш найпопулярнішими ліцензіями Java-проєктів. Плагін складається з двох модулів - Core та Integration. Архітектура плагіна представлена рисунку 3.1. Обидва модулі залежать від компонентів IntelliJ Platform SDK [4] — Набір інструментів, необхідних для створення плагінів до платформи IntelliJ Platform [3].

Перший із модулів, Core, містить внутрішні компоненти, що реалізують алгоритм контролю ліцензій проєкту. Компонент Project Manager відповідає за відстеження стану проєкту та оновлення інформації про знайдені ліцензії компонентів проєкту. License Manager здійснює визначення сумісних ліцензій для кожного модуля проєкту, а також виявляє потенційні порушення ліцензій бібліотек та підмодулів кожного модуля проєкту. Компонент License Detectors містить детектори ліцензій, які визначають тип ліцензії з урахуванням її повного тексту. Виділення детекторів на окрему компоненту дозволило абстрагуватися від процесу визначення ліцензій під час реалізації алгоритму контролю ліцензій проєкту, а також спростило додавання нових детекторів. Package Search Client відповідає за взаємодію з Package Search, сервісом для пошуку Java бібліотек від компанії JetBrains. Компоненти Dependency Model та License Model містять моделі даних залежностей та ліцензій відповідно, що дозволяє уніфікувати роботу з різними залежностями та ліцензіями проєктів, а також знижують витрати на

розширення плагіна та підтримку нових ліцензій.

Другий модуль, Integration, відповідає за інтеграцію з IntelliJ IDEA та надає користувачеві графічний інтерфейс, який дозволяє працювати з інформацією про ліцензії у зручному вигляді. Компонент Tool Window відповідає за вікно, що згортається, з інформацією про ліцензії проєкту. License Editor Notification надає панель вгорі редактора файлу з ліцензією, а Inlay License Hints відповідає за підказки під час редагування скриптів систем для збирання проєктів.

Таким чином, розроблена архітектура дозволяє відокремити логіку роботи з ліцензіями проєкту від інтерфейсу користувача, а також спрощує додавання нових детекторів ліцензій та підтримку нових типів ліцензій у плагіні.

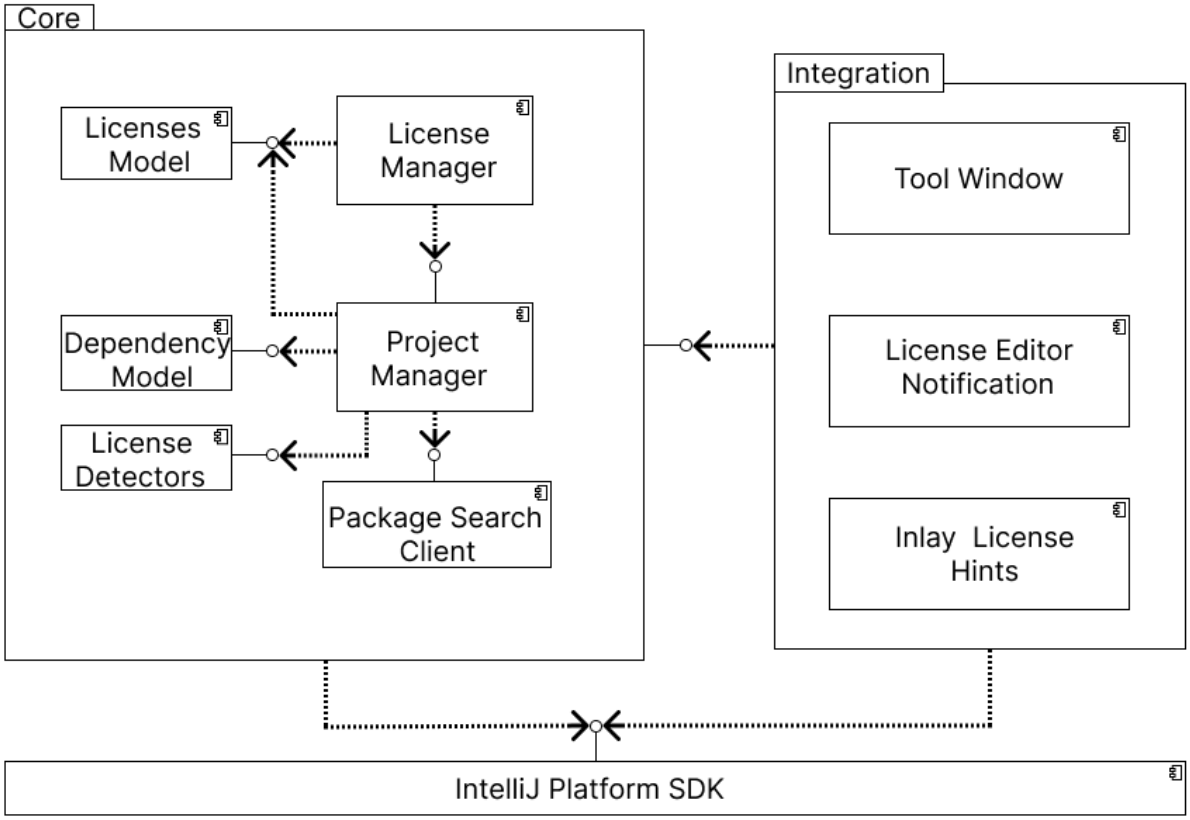

Рис.3.1 – Архітектура плагіну

РОЗДІЛ 4 АЛГОРИТМ КОНТРОЛЮ ЛІЦЕНЗІЙ

Алгоритм контролю ліцензій відстежує ліцензії компонентів проєкту та визначає допустимі ліцензії для кожного модуля. Допустимі ліцензії модуля — це ліцензії, які міг би мати модуль які не порушують ліцензії бібліотек та ліцензії інших модулів проєкту. На підставі допустимих ліцензій модулів алгоритм також знаходить потенційні порушення ліцензування у проєкті. Етапи алгоритму представлені рисунку 4.1.

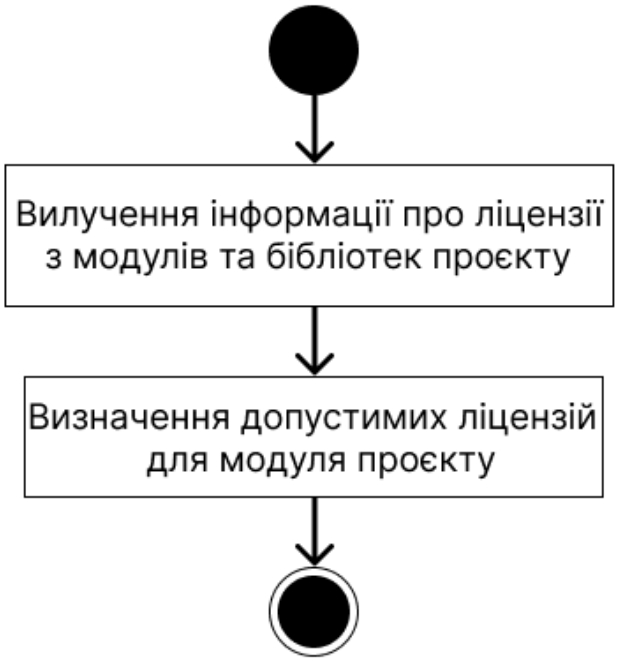

Рис. 4.1 – Алгоритм контролю ліцензій.

Першим етапом є отримання інформації про ліцензії компонент проєкту. На цьому етапі відбувається сканування модулів та бібліотек проєкту, а також пошук та вилучення інформації про їх ліцензії.

На другому етапі відбувається обробка отриманої інформації про ліцензіях модулів та бібліотек, визначаються допустимі ліцензії для кожного модуля проєкту та виявляються потенційні порушення ліцензій бібліотек та підмодулів.

Розглянемо докладніше процес отримання інформації про ліцензії компонент проєкту.

4.1. Вилучення інформації про ліцензії компонентів проєкту

Вилучення інформації про ліцензії походить з модулів та бібліотек проєкту. Отримана інформація використовується для визначення допустимих ліцензій для кожного модуля проєкту та пошуку потенційних порушень ліцензування. Процес отримання інформації про ліцензії компонент проєкту представлений на рисунку 4.2.

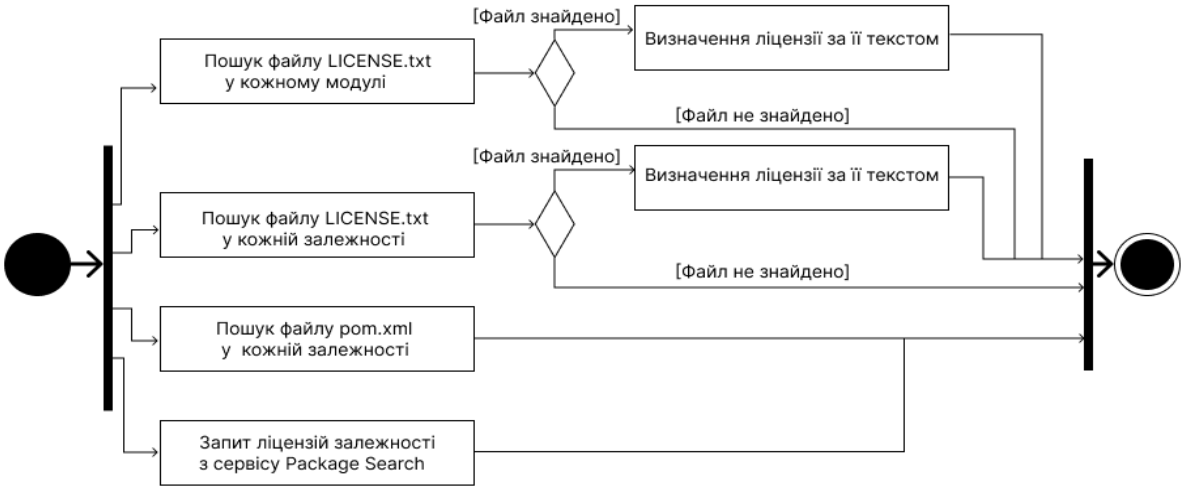

Рис. 4.2 – Процес отримання інформації про ліцензії компонент проєкту.

4.2 Вилучення ліцензій модулів

Вилучення інформації про ліцензії модулів проєкту відбувається наступним чином. З IntelliJ Platform витягується інформація провсіх модулях проєкту. Далі проглядаються кореневі директоріїкожного модуля на наявність файлу з назвою LICENSE.txt або подібним до нього. Знайдені файли повинні містити повний текст ліцензії, який використовується для визначення типу ліцензії. Процес визначення типу ліцензії за її повним текстом описано далі. В результаті ми отримуємо список модулів проєкту зі знайденими ліцензіями.

Необхідно відзначити, що використання IntelliJ Platform для отримання

даних про модулі проєкту дозволяє відмовитися від визначення ліцензій для кожної директорії проєкту та визначати ліцензії лише для модулів. Для цього необхідно переглядати лише кореневі директорії кожного модуля у пошуках ліцензії. У такий спосіб вдалося зробити процес визначення ліцензій більш оптимальним, ніж перегляд усіх директорій проєкту.

Крім цього, IntelliJ Platform використовується для отримання інформації про бібліотеки модуля, що дозволяє врахувати абсолютно всі бібліотеки модуля при визначенні допустимих ліцензій та пошуку потенційних порушень ліцензування. Без використання платформи отримання інформації про всі бібліотеки є нетривіальним завданням. Наприклад, якщо проєкт використовує систему складання Gradle, то аналіз Gradle скрипта не дозволить виявити всі бібліотеки, що підключаються за допомогою Gradle плагінів або завдань Gradle. IntelliJ Platform позбавлена таких недоліків. Тому її використання дає плагіну перевагу проти іншими інструментами, підвищує точність визначення потенційних порушень ліцензування і коректність рекомендацій ліцензій.

4.3 Вилучення ліцензій бібліотек

Після вилучення ліцензій із модулів відбувається вилучення ліцензій бібліотек проєкту, які зберігаються у вигляді jar-архівів. Такий архів містить усі необхідні файли для роботи бібліотеки. Серед цих файлів може бути файл LICENSE.txt з повним текстом ліцензії бібліотеки, яким можна визначити тип ліцензії. Крім цього в jar-архіві повинен бути файл pom.xml, який містить у собі метаінформацію про бібліотеку, включаючи назву її ліцензії. Якщо такий файл в архіві є, то вилучення інформації про ліцензію залежності зводиться до визначення ліцензії її назвою. Це не завжди просто зробити. На даний момент немає строго формалізованих назв ліцензій, тому назва однієї і тієї ж ліцензії може різнитися в метаданих різних бібліотек. Для розпізнавання ліцензії за назвою в плагіні використовуються регулярні вирази.

Однак на практиці досить часто виявляється так, що бібліотеки постачаються без файлу з метаінформацією або у такому файлі відсутня інформація про ліцензію залежності. У такому разі дізнатися про ліцензію бібліотеки без запитів до сторонніх ресурсів практично неможливо.

На даний момент існує кілька репозиторіїв залежностей та сервісів для хостингів IT-проєктів, які можуть містити інформацію щодо ліцензії конкретної бібліотеки. Реалізація отримання інформації про ліцензії бібліотек з усіх цих джерел складна і вимагає багато ресурсів на розробку та налагодження.

Альтернативним рішенням є використання сервісу для пошуку Java бібліотек Package Search [10] для отримання інформації про ліцензії бібліотек. Сервіс індексує популярні репозиторії та хостинги Java бібліотек і надає API для отримання повної інформації про конкретну залежність, включаючи ліцензію бібліотеки і, наприклад, посилання на репозиторій GitHub. Package Search є продуктом компанії JetBrains, тому не є стороннім рішенням по відношенню до плагіна, що розробляється.

Для визначення ліцензії бібліотеки за допомогою Package Search був реалізований компонент Package Search Client, який надсилає запити сервісу та перетворює отриману інформацію у внутрішнє уявлення плагіна. Незважаючи на всі зручності роботи з Package Search, він має недоліки. Перший недолік у тому, що сервіс не індексує репозиторій bintray/jcenter, у якому опубліковано безліч Java-бібліотек, яких немає в інших репозиторіях. Цей недолік перестав бути актуальним у зв'язку з новиною про швидке закриття репозиторію bintray/jcenter [12]. Усі бібліотеки, які в ньому знаходилися, повинні незабаром мігрувати в інші репозиторії, наприклад заходів у Maven Central [8] або на GitHub.

Інший недолік полягає в тому, що Package Search надає інформацію про ліцензію бібліотеки в цілому, а не для кожної конкретної версії. Якщо різні версії бібліотеки знаходяться під різними ліцензіями, то сервіс повертає перелік ліцензій без уточнення, яка ліцензія якої версії відповідає.

Також необхідно зазначити, що алгоритм контролю ліцензій при пошуку потенційних порушень враховує всі надані сервісом ліцензії. Це дозволяє уникнути деяких ситуацій неправильного визначення ліцензії бібліотеки, які можуть наводити до знайдених реальних порушень ліцензій. Водночас, використання всіх ліцензій збільшує кількість хибних потенційних порушень, що знаходить плагін. Однак у цьому випадку не пропустити реальне порушення важливіше, ніж потенційно знизити кількість хибнопозитивних спрацьовувань.

4.4 Визначення ліцензії на повний текст

Окремо необхідно розглянути процес визначення ліцензії щодо її повного тексту. Цей процес представлений на рисунку 4.3.

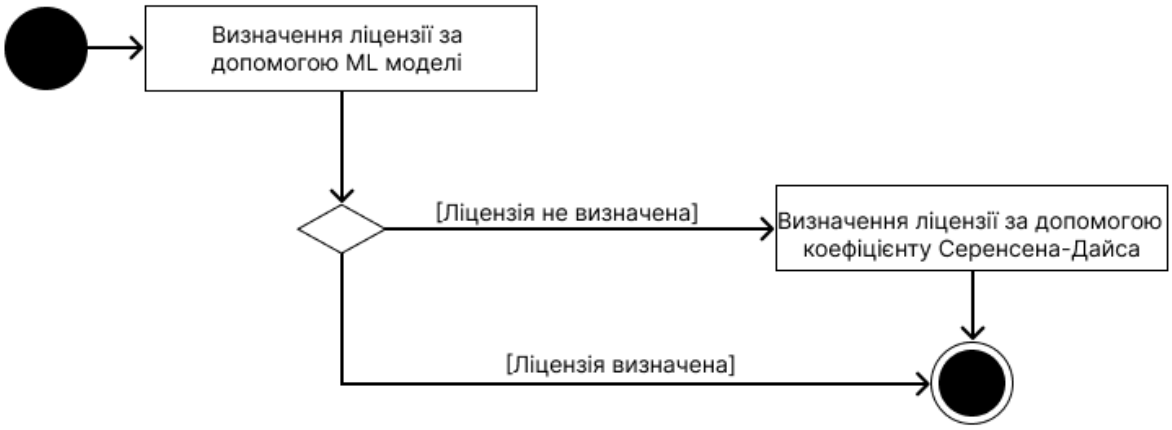

Рис. 4.3 – Процес визначення ліцензії за її повним текстом.

Для визначення ліцензії за її текстом використовують два детектори. Модель написана мовою Python з потужністю бібліотеки CatBoost. Для використання моделі в Kotlin плагіні, готова модель була експортована у форматі ONNX та інтегрована в плагін за допомогою бібліотеки KInference [6] від компанії JetBrains. Модель підтримує роботу із 12 ліцензіями. Це обмеження випливає з навчального набору даних, який був розміщено лише для 12 ліцензій. Для визначення інших ліцензій та випадків, коли ML детектор не зміг визначити тип ліцензії, використовується детектор на основі

коефіцієнта подібності Серенсен-Дайса (DSC). Коефіцієнт подібності для двох наборів елементів X та Y визначається як:

$$\mathrm{DSC} = \frac{2|\mathrm{X} \cap \mathrm{Y}|}{|X|+|Y|}, \tag{4.1}$$

де | X | та |Y | - Відповідні потужності наборів. Що ближче значення коефіцієнта до одиниці, то більше вписувалося набори елементів схожі.

Детектор на основі коефіцієнта подібності працює в такий спосіб. Кожна ліцензія, що підтримується, містить заздалегідь підготовлений набір слів оригінального тексту. Детектор приймає на вхід текст невідомої ліцензії та розбиває його на слова. Отриманий набір слів порівнюється за допомогою коефіцієнта Серенсен-Дайса з наборами слів оригінальних ліцензій. Збіг зараховується, коли коефіцієнт подібності більше порогового значення, яке дорівнює 0,95. Таким чином, детектор вимагає збігу з оригінальним текстом ліцензії на 95% та допускає невеликі зміни тексту.

Цей коефіцієнт подібності та значення порога подібності вибрано не випадково. Коефіцієнт Серенсен-Дайса з порогом 0.95 лежитьв основі інструмента для визначення ліцензій licensee, який використовується визначення ліцензій проєктів на популярному хостингу GitHub. Тому даний метод визначення ліцензій є перевіреним рішенням та показує хороший результат.

Обидва детектори ліцензій містяться в компоненті License Detectors модуля Core Завдяки їм розроблений плагін підтримує роботу із 16 ліцензіями. У таблиці 4.1 представлені підтримувані ліцензії та методи їх визначення у плагіні.

Ліцензія	Метод визначення
GNU Affero General Public License v3.0 only	ML + DSC

Apache License 2.0	ML + DSC
BSD 2-Clause “Simplified” License	ML + DSC
BSD 3-Clause “New” or “Revised” License	ML + DSC
Common Development and Distribution License 1.0	DSC
Eclipse Public License 1.0	DSC
GNU General Public License v2.0 only	ML + DSC
GNU General Public License v2.0 w/Classpath exception	DSC
GNU General Public License v3.0 only	ML + DSC
ISC License	ML + DSC
GNU Lesser General Public License v2.1 only	ML + DSC
GNU Lesser General Public License v3.0 only	ML + DSC
MIT License	ML + DSC
Mozilla Public License 1.1	DSC
Mozilla Public License 2.0	ML + DSC

Таблиця 4.1 – Підтримувані ліцензії та методи їх визначення у плагіні.

Ці ліцензії покривають приблизно 95% усіх Java-проєктів. Ліцензії були обрані на основі результатів дослідження, в якому міститься інформація про найпопулярніші ліцензії Javaпроєктів на GitHub. Незважаючи на те, що в дослідженні розглядаються популярні ліцензії сімейства GPL і LGPL з приставкою абозаписом, у плагіні підтримані суворіші версії цих ліцензій з приставкою тільки. Ліцензії з приставкою or-later, на відміну від ліцензій з приставкою only, дозволяють використовувати ліцензоване ПЗ під тією самою ліцензією більш старшої версії. Наприклад, бібліотеку з ліцензією

GPL-2.0-or-later можна використовувати для розробки продукту з ліцензією GPL-3.0-only, а ліцензія GPL-2.0-only не надає такої можливості. Тексти версій only і or-later дуже схожі і можуть відрізнятися всього в одному словосполученні, що погано позначається на точності детектування. Невеликі відмінності в таких текстах ліцензій тягнуть різні правила сумісності з старшими версіями ліцензій. Тому для уникнення некоректної роботи з ліцензіями сімейств GPL та LGPL плагін підтримує роботу з більш строгими версіями цих ліцензій.

4.5 Визначення допустимих ліцензій для модуля проєкту

На основі інформації про ліцензії компонент, отриманої напопередньому етапі відбувається процес визначення допустимих ліцензій для кожного модуля і пошук потенційних порушень ліцензування.

На даному етапі для кожного модуля визначено ліцензію самого модуля, а також ліцензії його підмодулів та бібліотек. Якщо у модуля немає ліцензії, використовується ліцензія модуля батька, якщо він існує. Також, якщо у підмодуля немає ліцензії, то всі його підмодулі та бібліотеки враховуються при обробці модуля батька.

Процес визначення допустимих ліцензій представлений рисунку 4.4.

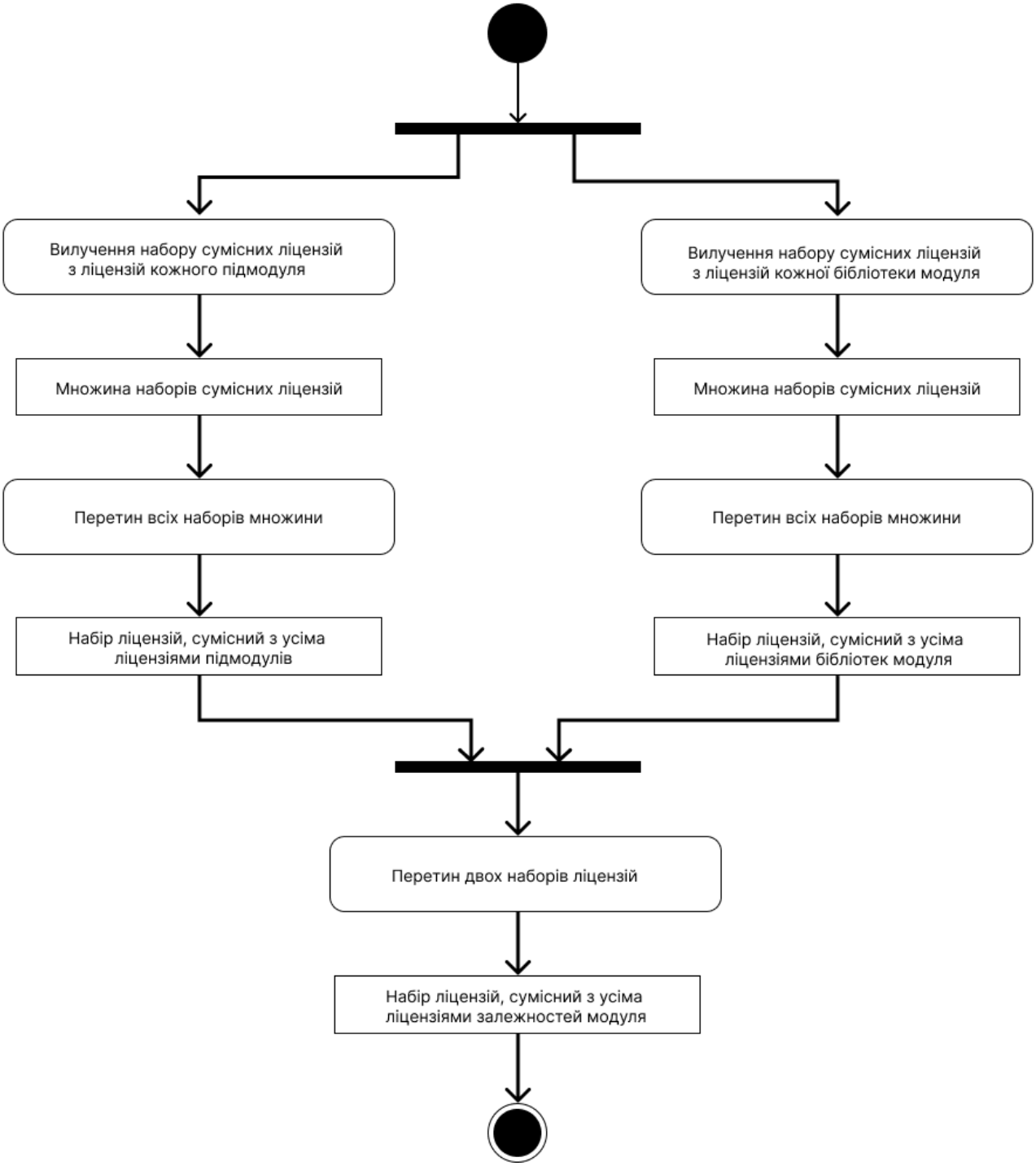

Рис. 4.4 – Визначення допустимих ліцензій модуля.

Набір допустимих ліцензій модулів визначається так. Для кожної підтриманої в плагіні ліцензії встановлено два набори сумісних із нею ліцензій. Один із них призначений для ліцензій бібліотек модуля. Він містить ліцензії модуля, які не порушуватимуть цю ліцензію, якщо вона є ліцензією бібліотеки. Такий набір отримується з кожної ліцензії бібліотек модуля. В

результаті одержуємо безліч наборів допустимих ліцензій. Перетинаючи всі набори сумісних ліцензій отримуємо один набір з ліцензій, які не порушують жодної ліцензії бібліотеки цього модуля.

Детальна процедура відбувається для ліцензій підмодулів. Для використовується інший набір сумісних ліцензій. Він містить допустимі ліцензії модуля, які не порушуватимуть цю ліцензію, якщо вона є ліцензією підмодуля. Такий набір витягується з кожної ліцензії підмодулів. В результаті одержуємо безліч наборів сумісних ліцензій. Перетинаючи всі набори сумісних ліцензій отримуємо один набір із ліцензій, які не порушують жодної ліцензії підмодуля цього модуля.

Отримані два набори ліцензій знову перетинаються. В результаті отримуємо набір із допустимих ліцензій модуля, які не порушують ні ліцензії бібліотек, ні ліцензії підмодулів. Знайдені допустимі ліцензії модуля дозволяють надати рекомендацію щодо вибору ліцензії та допомогти користувачеві вибрати ліцензію, не замислюючись про порушення ліцензування.

4.6 Пошук потенційних порушень

Окремо необхідно розглянути пошук потенційних порушень ліцензування. Процес пошуку представлений рисунку 4.5.

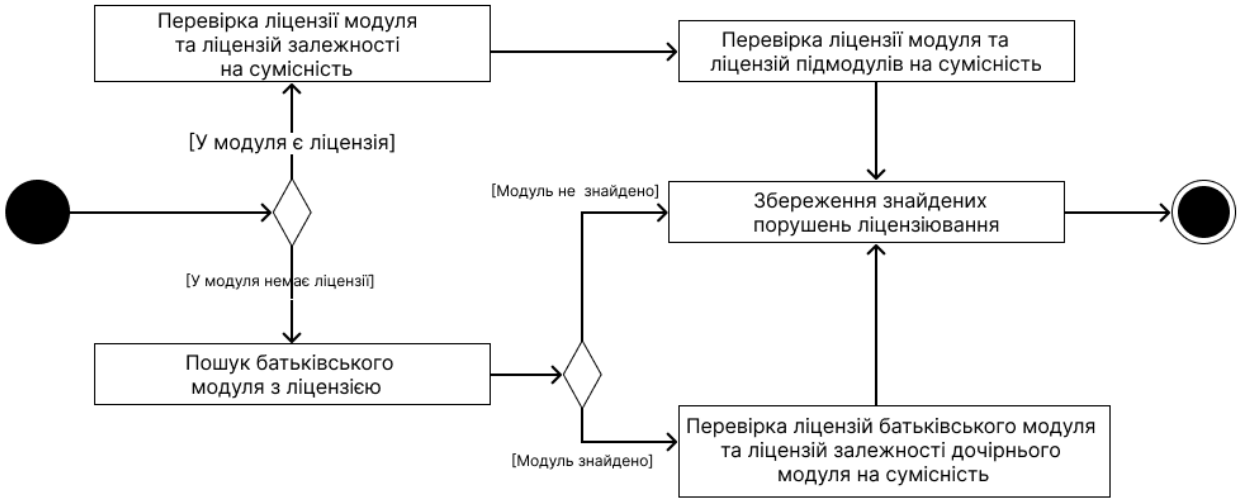

Рис. 4.5 – Процес пошуку потенційних порушень ліцензування.

Потенційні порушення ліцензій визначаються як між ліцензією модуля та ліцензією бібліотеки, так і між ліцензією модуля та ліцензією підмодуля. Процес пошуку потенційних порушень багато в чому схожий на визначення допустимих ліцензій модуля. Для визначення порушення між ліцензією модуля та ліцензією бібліотеки з ліцензії бібліотеки витягується набір допустимих ліцензій модуля. Цей набір містить усі ліцензії, які може мати модуль та які не порушуватимуть ліцензію бібліотеки. Якщо поточна ліцензія модуля входить до цього набору, то порушення ліцензії немає. Якщо поточна ліцензія в набір не входить, то плагін реєструє потенційне порушення ліцензії та виводить розробнику відповідну інформацію. Подібна процедура проводиться для кожної бібліотеки кожного модуля. Таким чином плагін визначає всі потенційні порушення, пов'язані з порушенням ліцензій бібліотек.

Розглянемо приклад визначення порушення ліцензії бібліотеки.

Приклад представлений рисунку 4.6.

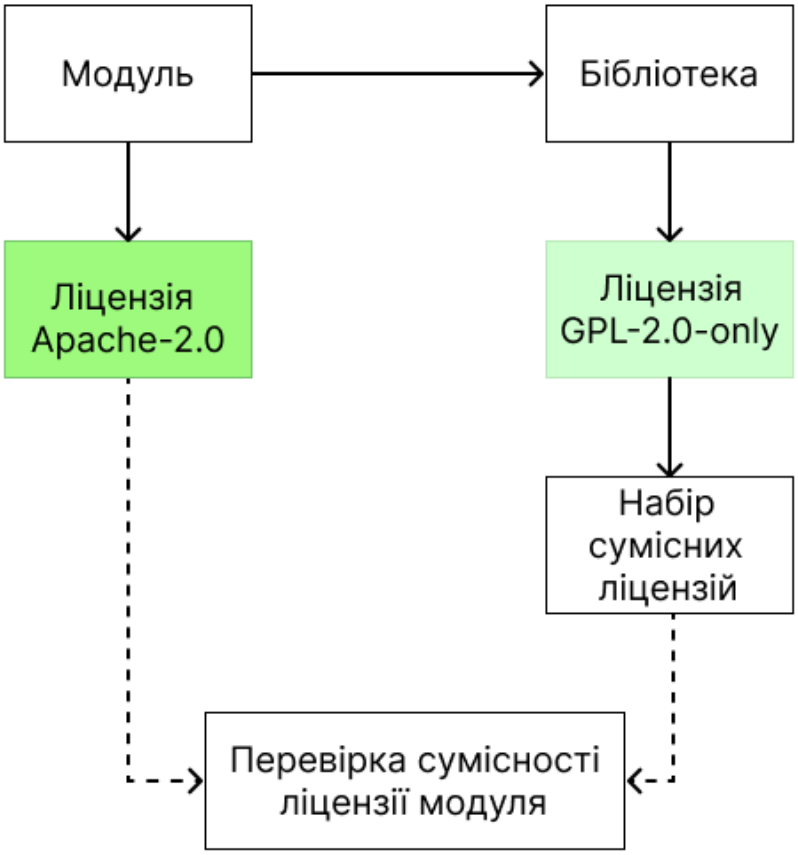

Рис. 4.6 – Приклад визначення порушення ліцензії бібліотеки.

Нехай модуль із ліцензією Apache-2.0 використовує бібліотеку з ліцензією GPL-2.0-only. З ліцензії GPL-2.0-only виходить набір можливих ліцензій модуля, які не порушують ліцензію бібліотеки. Плагін виявляє, що ліцензія Apache-2.0 не входить до цього набору ліцензій та реєструє відповідне порушення ліцензії.

Аналогічно відбувається визначення потенційного порушення між ліцензій модуля та ліцензією підмодуля. Для цього з ліцензії підмодуля виймається набір допустимих ліцензій. Цей набір містить усі ліцензії, які може мати модуль та які не порушуватимуть ліцензію підмодуля. Якщо поточна ліцензія модуля входить до цього набору, порушення немає. Якщо поточна ліцензія в набір не входить, то плагін реєструє потенційне порушення ліцензії та виводить розробнику відповідну інформацію. Ця процедура проводиться для кожного підмодуля кожного модуля. У результаті плагін визначає всі потенційні порушення пов'язані із порушенням ліцензій

підмодулів.

Розглянемо приклад визначення порушення ліцензії підмодуля. Приклад представлений рисунку 4.7.

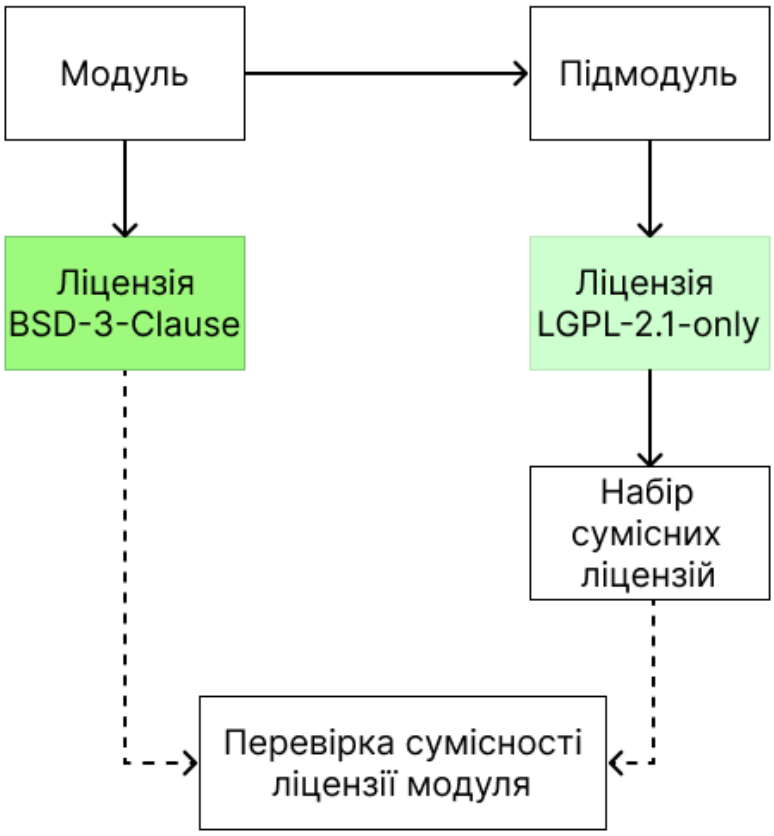

Рис. 4.7 – Приклад визначення порушення ліцензії підмодуля.

Нехай модуль із ліцензією BSD-3-Clause має підмодуль із ліцензією LGPL-2.1-only. З ліцензії LGPL-2.1-only виходить набір можливих ліцензій модуля, які не порушують ліцензію підмодуля. Плагін виявляє, що ліцензія BSD-3-Clause не входить до цього набору ліцензій та реєструє порушення ліцензії підмодуля.

РОЗДІЛ 5 ІНТЕРФЕЙС КОРИСТУВАЧА ПЛАГІНА

Розроблений плагін надає користувачеві графічний інтерфейс для зручності роботи з ліцензіями проєкту. Для реалізації інтерфейсу використано бібліотеку Swing [13], яка є стандартною бібліотекою для розробки графічного інтерфейсу в плагінах до IntelliJ IDEA і входить до IntelliJ Platform SDK.

Розглянемо докладніше реалізовані компоненти графічного інтерфейсу.

5.1 Tool Window

Основним графічним інтерфейсом плагіна є вікно, що згортається Tool Window. Вікно містить дві вкладки та надає основну інформацію про ліцензії проєкту.

Перша вкладка, Project License, містить інформацію про ліцензію кореневого модуля проєкту з докладним описом дозволів та обмежень ліцензії. Крім того, вкладка містить потенційний список порушень ліцензування у проєкті, які плагін зміг виявити. Project License також надає можливість експортувати список порушень ліцензування у форматі json та перезапустити алгоритм контролю ліцензій проєкту. Ця вкладка представлена на рисунку 5.1.

Друга вкладка, Package License, проставляє інформацію про знайдені бібліотеки проєкту та їх ліцензії. Також на цій вкладці є можливість фільтрації залежностей за приналежністю до модуля та пошуковим запитом. Вкладка Package License представлена рисунку 5.2.

5.2 License Editor Notification

Крім вікна Tool Window, плагін надає панель License Editor Notification. Дана панель знаходиться вгорі редактора файлів з ліцензією. Вона дозволяє подивитися, які ліцензії сумісні з усіма ліцензіями залежностей модуля або проєкту та вибрати найбільш підходящу. Крім того, панель надає можливість

порівняти поточний текст ліцензійного файлу з оригінальним текстом ліцензії. Панель License Editor Notification представлена рисунку 5.3.

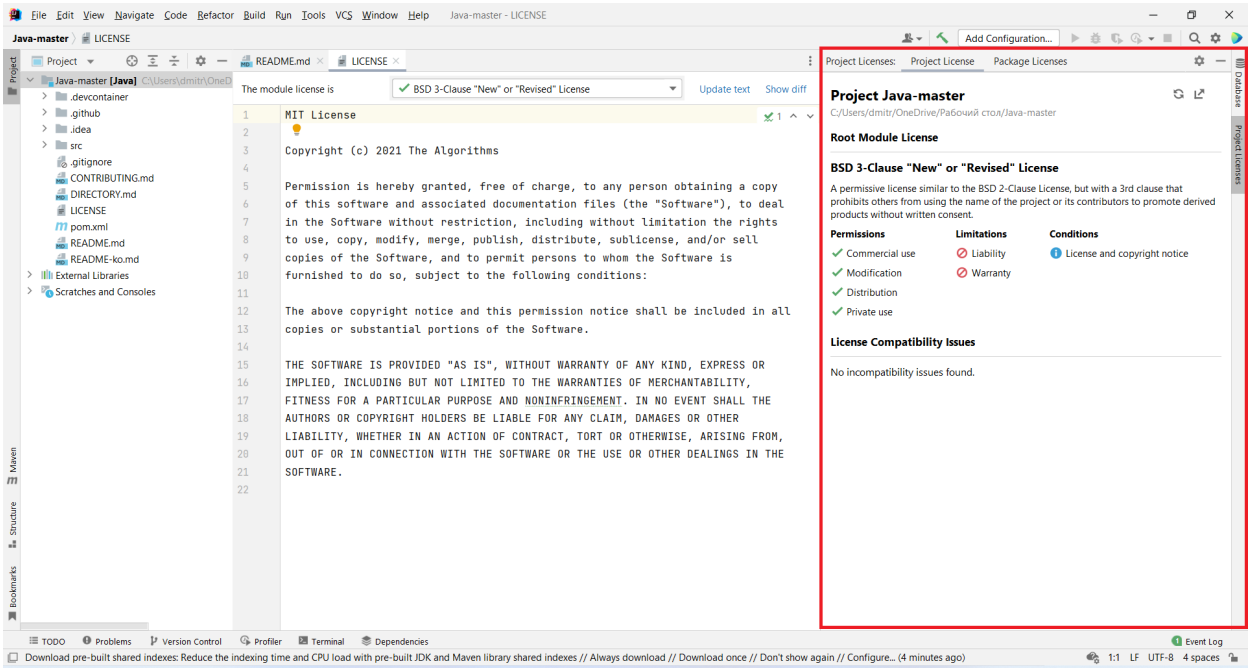

Рис. 5.1 – Вкладка Project License.

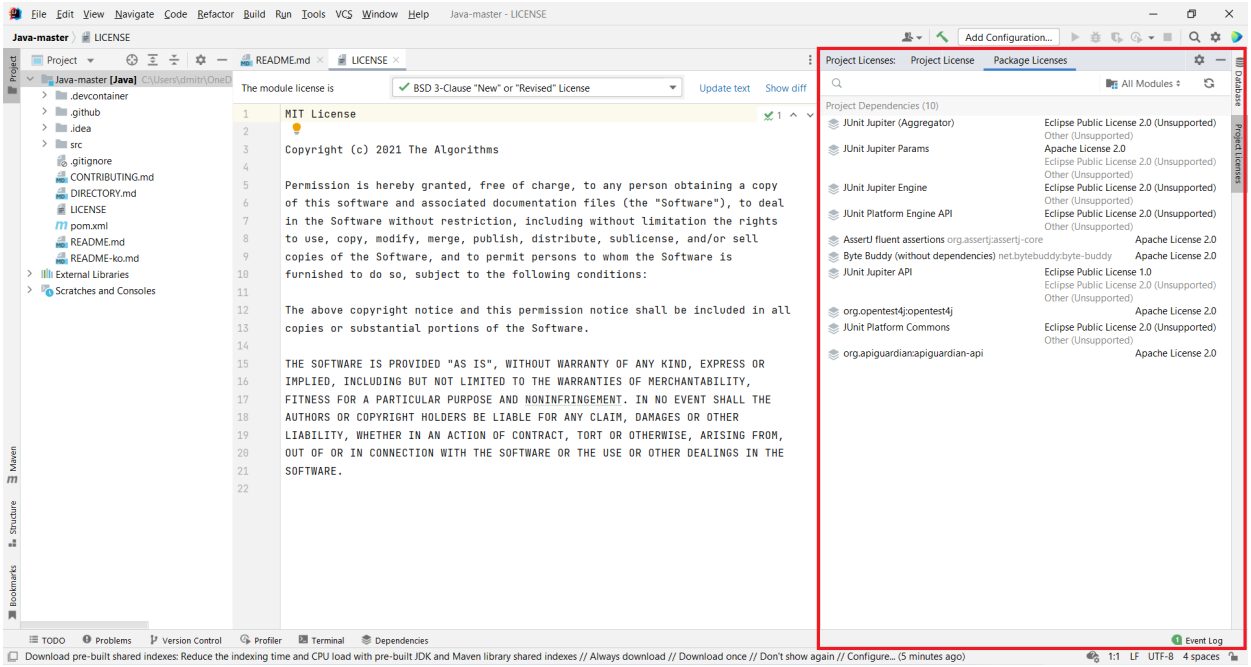

Рис. 5.2 – Вкладка Package License.

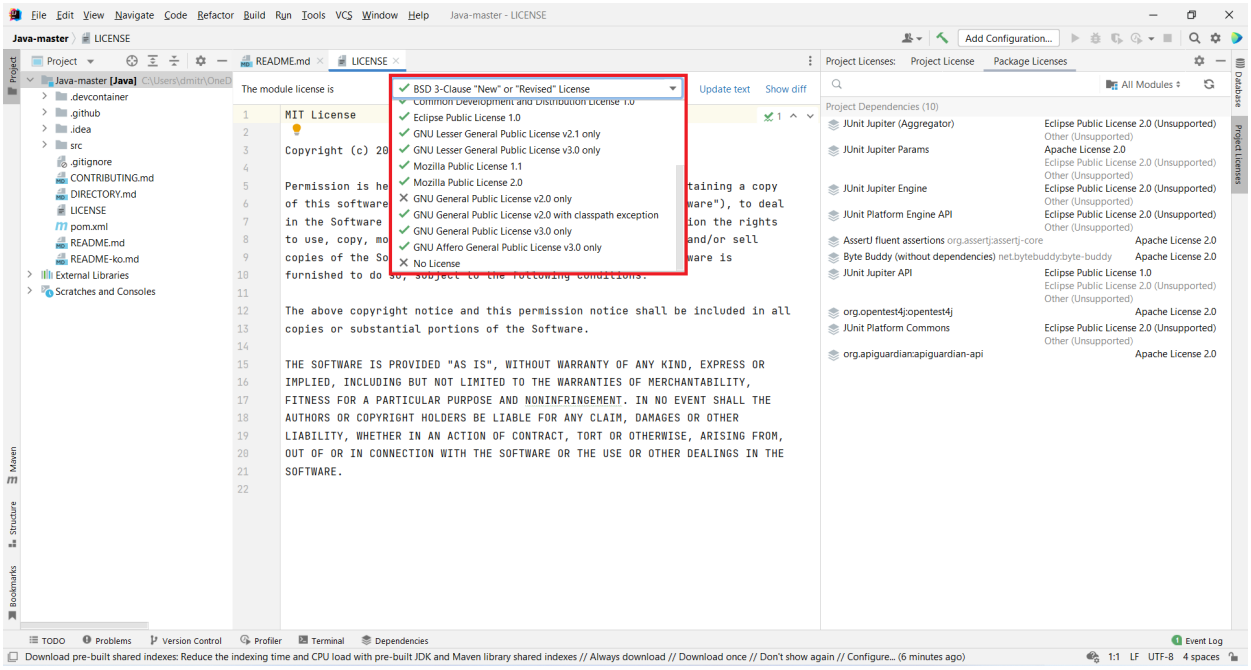

Рис. 5.3 – Панель редактора для роботи з файлом основної ліцензії проєкту.

У цьому прикладі проєкт має залежності майже з усіма ліцензіями, і є сумісними окрім GPL-2.0-only. Всі сумісні ліцензії позначаються галочкою, інші – хрестиком. Для того щоб дізнатися, які порушення виникають при використанні несумісних ліцензій, необхідно вибрати таку ліцензію зі списку ліцензій. Після цього у вікні ToolWIndow з'явиться відповідний перелік потенційних порушень. Приклад на рисунку 5.4.

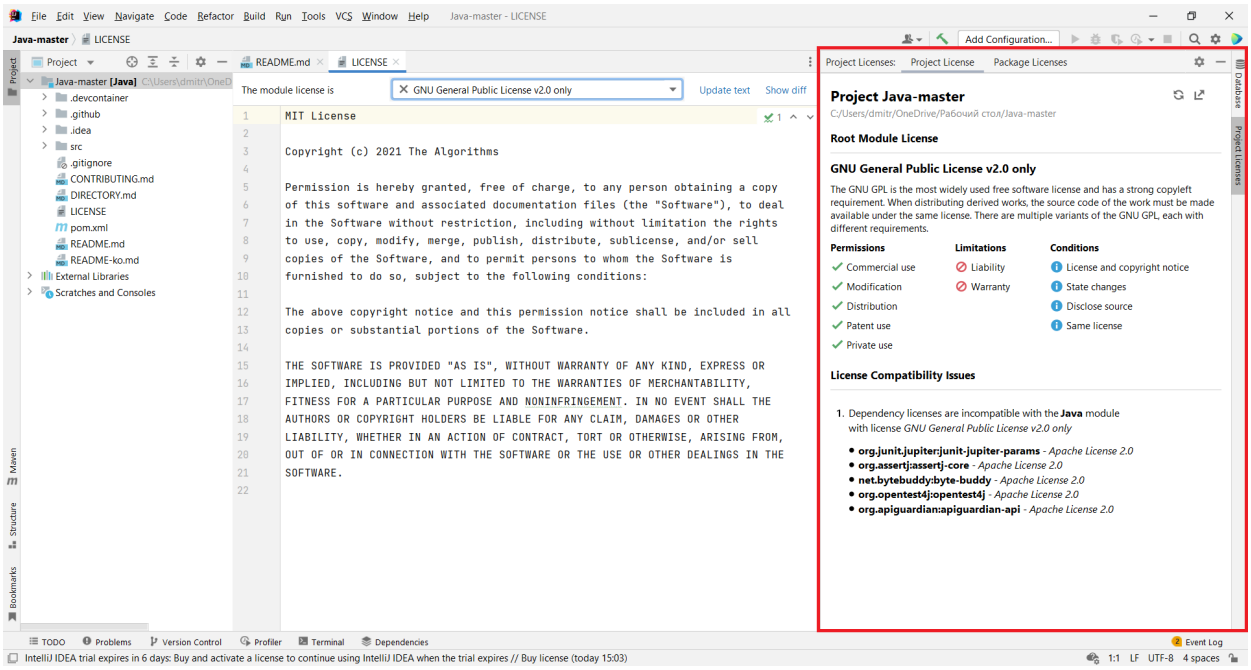

Рис 5.4 – Перелік порушень ліцензії у проєкті

Крім цього плагін надає можливість створити файл з найбільш допустимою ліцензією модуля. Завдяки цим можливостям користувач, який практично не має знань про ліцензії, зможе вибрати коректну ліцензію для свого проєкту і не припуститися помилки.

5.3 Inlay License Hints

Розроблений плагін також надає підказки з ліцензіями бібліотек під час редагування скрипту системи збирання проєкту. Напроти кожної команди з підключенням бібліотеки з'являється підказка під назвою ліцензії цієї бібліотеки. Такі підказки дозволяють користувачеві відразу дізнатися ліцензію залежності прямо у редакторі коду. Дані підказки реалізовані для Maven, Groovy Gradle та Kotlin Gradle скриптів. Приклад роботи підказок у Kotlin Gradle скрипт представлений на рисунку 5.5.

Рис. 5.5 – Підказки ліцензій бібліотек у Kotlin Gradle скрипті.

РОЗДІЛ 6 АПРОБАЦІЯ ПЛАГІНУ

В рамках апробації плагіна було вивчено потенційні порушення ліцензування, які знаходить плагін, а також проаналізовано найчастіші знайдені порушення між ліцензіями.

Апробація плагіна проводилася на 100 Java-проєктах GitHub з найбільшою кількістю зірок. Для запуску плагіна на безлічі проєктів було написано спеціальний скрипт, який запускає IntelliJ IDEA у headless режимі (без графічного інтерфейсу) та збирає інформацію про порушення ліцензій з плагіна. Однак у процесі апробації виявилося, що частина проєктів некоректно імпортується за допомогою фреймворку для тестування IntelliJ Platform. Тому такі проєкти були оброблені вручну та відкриті в IDE у стандартному режимі.

За результатами апробації у 100 проєктах плагін виявив 762 потенційних порушення між ліцензією бібліотеки та ліцензією модуля. Десять найпопулярніших пар ліцензій у потенційних порушеннях ліцензування представлені рисунку 6.1.

Серед знайдених порушень ліцензій виділяється група із 314 випадки використання бібліотек з дозвільними ліцензіями (Apache-2.0, MIT, BSD-3-CLause) у модулях без ліцензії (No license). Дані потенційні порушення пов'язані з нестандартними або модифікованими текстами ліцензій модулів, що дозволяє програмно визначити тип ліцензії. Тим не менш, дані порушення вказують розробнику на можливі проблеми з ліцензією модуля і можуть мотивувати його змінити ліцензію однією з загальноприйнятих.

Крім цього, серед виявлених порушень 384 потенційних порушення пов'язані з використанням бібліотек під ліцензією GPL2.0-only у модулях з дозвільними ліцензіями (Apache-2.0, MIT та BSD-3-CLause). Причиною цих потенційних порушень могла стати сувора робота з модифікаціями ліцензії GPL-2.0. Бібліотеки з ліцензією GPL-2.0-with-classpath-exception можуть

використовуватись в проєкті з дозвільною ліцензією, наприклад з Apache-2.0, на відміну від бібліотек з ліцензією GPL-2.0-only.

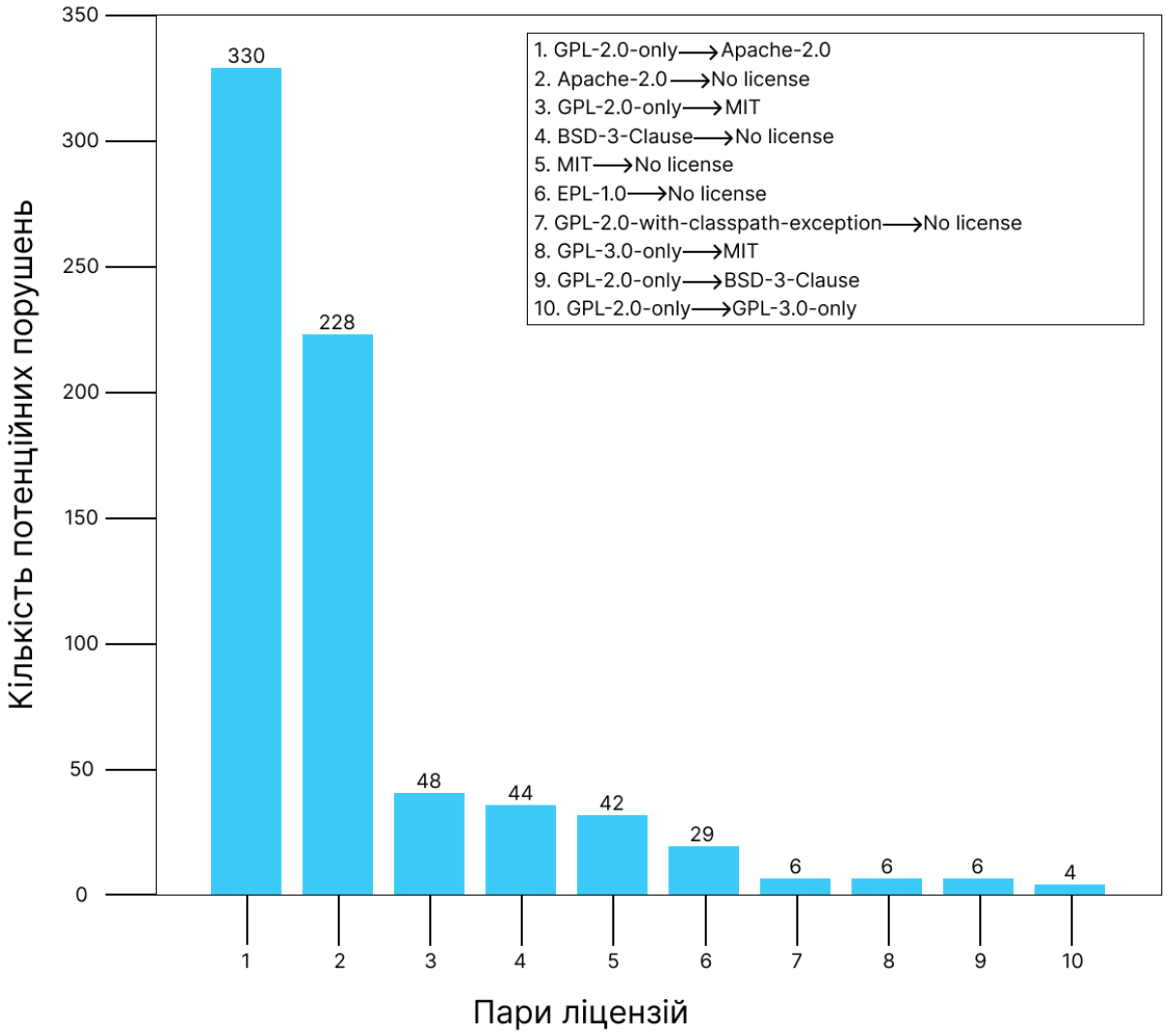

Рис. 6.1 — Десять найпопулярніших пар ліцензій (бібліотека → модуль) у потенційних випадках порушення правил ліцензування.

Проте тексти ліцензій GPL-2.0-with-classpath-exception і GPL-2.0-тільки можуть відрізнятися всього в одному реченні. У випадках, коли немає можливості визначити точну модифікацію ліцензії, плагін вибирає найбільше обмежуючу GPL-2.0-лише, що призводить до появи потенційних порушень. Такий процес роботи з модифікаціями ліцензій GPL-2.0 дозволяє не пропустити реальні порушення ліцензії GPL-2.0-only, увагу розробника на підозрілі бібліотеки та спонукає перевірити коректність виявлених порушень

вручну.

Необхідно відзначити, що деякі знайдені потенційні порушення після ретельної ручної перевірки можуть виявитися хибними і не будуть порушеннями як такими. Однак однією з задач плагіна є завдання відсіювання свідомо коректних поєднань ліцензій та надання розробнику інформації про підозрілі випадки поєднання ліцензій. Розробник, отримавши інформацію про потенційні порушення, може вручну перевіритиїх на коректність або усунути порушення відразу, відмовившись від бібліотеки з несумісною ліцензією або змінивши ліцензію відповідного модуля.

Також слід зазначити, що плагін не виявив потенційних порушень ліцензій між модулями самого проєкту. Ймовірно це пов'язано з тим, що розробники зазвичай ліцензують весь проєкт повністю, а чи не кожен модуль окремо. Тому порушень ліцензування між модулями проєкту немає.

ВИСНОВОК

У ході цієї роботи було отримано такі результати.

1. Зроблено огляд предметної області. Розглянуто різні типи відкритих ліцензій та правила сумісності між ними. Також зроблено огляд відомих досліджень про порушення ліцензування під час копіювання коду відкритих проєктів.

2. Проведено аналіз популярних інструментів для роботи з ліцензіями проєктів, який показав, що більшість інструментів не опрацьовують ліцензії залежностей проєкту. Безкоштовні інструменти не надають користувачеві рекомендації щодо вибору основної ліцензії проєкту та не повідомляють про несумісні ліцензії. Крім того, жоден з інструментів не вбудований одну з найпопулярніших інтегрованих середовищ розробки.

3. Спроєктовано архітектуру плагіна. Вона дозволила відокремити логіку роботи з ліцензіями від інтерфейсу користувача та спростити підтримку нових детекторів та ліцензій у плагіні.

4. Реалізовано алгоритм контролю ліцензій проєкту. Реалізовано механізм вилучення інформації про ліцензії модулів та залежностей проєкту, а також реалізовано механізм визначення сумісних ліцензій для кожного модуля проєкту на основі ліцензій залежностей та підмодулів.

5. Реалізовано інтерфейс плагіна, який містить докладну інформацію про ліцензії модулів та залежностей, а також список потенційних порушень ліцензування Крім того, інтерфейс містить рекомендації щодо вибору ліцензії, сумісної з ліцензіями всіх модулів та залежностей проєкту.

6. Проведено апробацію розробленого плагіна на 100 найпопулярніших проєктах на GitHub, що містять код на Java. Плагін виявив 762 потенційні порушення ліцензування, з яких 384 потенційні порушення пов'язані з використанням бібліотек з модифікаціями ліцензії GPL-2.0 у модулях із

дозвільною ліцензією.

В результаті даної роботи було реалізовано та випущено пілотну версію плагіна для роботи з ліцензіями Java-проєктів, який може успішно застосовуватись для пошуку потенційних порушень ліцензування. Плагін може бути покращений головним чином за рахунок розширення списку підтримуваних ліцензій та більш точних інструментів для визначення типу ліцензії за її повним текстом.

СПИСОК ЛІТЕРАТУРИ

1. Gaudeul Alex. Do Open Source Developers Respond to Competition? The LATEX Case Study // Review of Network Economics. — 01 Jun. 2007. — Vol. 6, no. 2. — URL: https://www.degruyter.com/ view/journals/rne/6/2/article-rne.2007.6.2.1119.xml.xml

2. German Daniel M., Di Penta Massimiliano, Davies Julius. Understanding and Auditing the Licensing of Open Source Software Distributions // Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension. — ICPC ’10. — USA : IEEE Computer Society, 2010. — P. 84–93. — URL: https://doi.org/ 10.1109/ICPC.2010.48

3. Intellij Platform GitHub repository. — 2021. — URL: https://github.com/JetBrains/intellij-community.

4. Intellij Platform SDK Docs. — 2021. — URL: https://plugins.jetbrains.com/docs/intellij/welcome.html.

5. Java Research License. — 2020. — URL: https: //www.oracle.com/technetwork/java/javase/jrl-5-150091.txt

6. KInference. — 2021. — URL: https://github.com/ JetBrains-Research/kinference.

7. Markovtsev Vadim, Long Waren. Public git archive // Proceedings of the 15th International Conference on Mining Software Repositories - MSR ’18. — 2018. — URL: http://dx.doi.org/10.1145/ 3196398.3196464

8. Maven Central dependency repository. — 2021. — URL: https://mvnrepository.com/repos/central/

9. The Open Source Definition. — 2007. — URL: https:// opensource.org/osd

10. Package Search Official Website. — 2021. — URL: https: //package-search.jetbrains.com/

11. Sourcerer’s Apprentice and the study of code snippet migration / S. Romansky, C. Chen, B. Malhotra, A. Hindle // ArXiv. — 2018. — Vol. abs/1808.00106.

12. Sunset Bintray JCenter Official Blog Post. — 2021. — URL: https://jfrog.com/blog/ into-the-sunset-bintray-jcenter-gocenter-and-chartcenter/

13. Swing — GUI widget toolkit for Java. — 2021. — URL: https://docs.oracle.com/javase/8/docs/technotes/guides/ swing/index.html

14. ootpa Troan Larry. Open Source from a Proprietary Perspective. — 2006. — URL: https://docplayer.net/ 5605674-Open-source-from-a-larry-ootpa-troan.html

ДОДАТОК А

```
package com.jetbrains.plugin.actions

import com.intellij.openapi.actionSystem.AnAction
import com.intellij.openapi.actionSystem.AnActionEvent
import com.intellij.openapi.actionSystem.PlatformDataKeys
import com.intellij.openapi.application.ApplicationManager
import com.intellij.openapi.application.WriteAction
import com.intellij.openapi.application.WriteActionAware
import com.intellij.openapi.command.CommandProcessor
import com.intellij.openapi.fileEditor.OpenFileDescriptor
import com.intellij.openapi.module.ModuleUtilCore.findModuleForFile
import com.intellij.openapi.project.Project
import com.intellij.openapi.project.guessModuleDir
import com.intellij.openapi.util.ThrowableComputable
import com.intellij.psi.PsiDocumentManager
import com.intellij.psi.PsiFile
import com.intellij.psi.PsiManager
import com.intellij.util.IncorrectOperationException
import
com.jetbrains.sorrel.plugin.detection.DetectorManager.licenseFileNamePattern
import com.jetbrains.sorrel.plugin.licenses.NoLicense
import com.jetbrains.sorrel.plugin.utils.licenseDetectorModel
import com.jetbrains.sorrel.plugin.utils.logDebug
import java.io.File

class CreateProjectLicenseFile : AnAction(), WriteActionAware {

    companion object {
        const val LICENSE_FILE_NAME: String = "LICENSE.txt"
    }

    private val actionName = "CreateProjectLicenseFile"

    override fun actionPerformed(e: AnActionEvent) {
        val application = ApplicationManager.getApplication()
        val commandProcessor = CommandProcessor.getInstance()

        val project: Project = e.project!!

        val model = project.licenseDetectorModel()

        val module =
findModuleForFile(e.getRequiredData(PlatformDataKeys.VIRTUAL_FILE),
```

```
project)!!
        val moduleDir =
PsiManager.getInstance(project).findDirectory(module.guessModuleDir()!!)!!

        val createProjectLicenseFile = {
            try {
                val licenseFile: PsiFile = WriteAction.compute(
                    ThrowableComputable<PsiFile, IncorrectOperationException> {
                        moduleDir.createFile(LICENSE_FILE_NAME)
                    }
                )

                val licenseDocument =
PsiDocumentManager.getInstance(project).getDocument(licenseFile)!!
                val curProjectModule = model.projectModules.value.find {
it.nativeModule == module }!!
                val compatibleLicenses =
model.licenseManager.modulesCompatibleLicenses
                    .value[curProjectModule]!!

                //TODO: Mb must be done in full order in licenses. Now the order is
partial
                if (compatibleLicenses.any()) {
                    val recommendedLicense = compatibleLicenses[0]
                    application.runWriteAction {
                        licenseDocument.setText(recommendedLicense.fullText)
                    }
                    val newModulesLicenseMap =
model.licenseManager.modulesLicenses.value.toMutableMap()
                    newModulesLicenseMap[curProjectModule] = recommendedLicense
                    model.licenseManager.modulesLicenses.set(newModulesLicenseMap)
                } else {
                    application.runWriteAction {
                        licenseDocument.setText(NoLicense.fullText)
                    }
                    val newModulesLicenseMap =
model.licenseManager.modulesLicenses.value.toMutableMap()
                    newModulesLicenseMap[curProjectModule] = NoLicense
                    model.licenseManager.modulesLicenses.set(newModulesLicenseMap)
                }

                val openFileDescriptor = OpenFileDescriptor(project,
licenseFile.virtualFile)
                openFileDescriptor.navigate(true)
```

```
        } catch (e: IncorrectOperationException) {
            logDebug("Failed to create license file.", e)
        }
    }

    commandProcessor.executeCommand(
        project,
        createProjectLicenseFile, actionName, null
    )
}

override fun update(e: AnActionEvent) {
    val project = e.project

    if (project == null) {
        e.presentation.isEnabledAndVisible = false
        return
    }

    val virtualFile = e.getData(PlatformDataKeys.VIRTUAL_FILE)
    if (virtualFile == null) {
        e.presentation.isEnabledAndVisible = false
        return
    }

    val module = findModuleForFile(virtualFile, project)
    if (module == null) {
        e.presentation.isEnabledAndVisible = false
        return
    }

    val moduleDirPath = module.guessModuleDir()
    if (moduleDirPath == null) {
        e.presentation.isEnabled = false
        return
    }

    val moduleDir = File(moduleDirPath.path)
    if (!moduleDir.exists() || !moduleDir.isDirectory || !moduleDir.canRead()) {
        e.presentation.isEnabled = false
        return
    }
```

```
        if (moduleDir.listFiles()!!.any { licenseFileNamePattern.matches(it.name) }) {
            e.presentation.isEnabledAndVisible = false
            return
        }

        e.presentation.isEnabledAndVisible = true
    }
}
package com.jetbrains.plugin.detection

import com.intellij.openapi.application.ReadAction
import com.intellij.openapi.editor.Document
import com.intellij.openapi.fileEditor.FileDocumentManager
import com.intellij.openapi.project.Project
import com.intellij.openapi.roots.OrderRootType
import com.intellij.openapi.roots.libraries.Library
import com.intellij.openapi.vfs.VirtualFile
import com.intellij.psi.PsiManager
import com.jetbrains.sorrel.plugin.licenses.*
import kotlinx.coroutines.yield
import org.jetbrains.kotlin.backend.common.pop

/**
 * This class provide all logics for License detection.
 */
object DetectorManager {

    private val mlDetector = MLDetector()
    private val sorensenDiesDetector = SorensenDiesDetector()

    val licenseFileNamePattern: Regex = Regex(

"(LICENSE.*|LEGAL.*|COPYING.*|COPYLEFT.*|COPYRIGHT.*|UNLICENS
E.*|" +
                "MIT.*|BSD.*|GPL.*|LGPL.*|APACHE.*)(\\.txt|\\.md|\\.html)?",
RegexOption.IGNORE_CASE
    )

    private const val metaInfoFolderName = "META-INF"
    private const val pomXmlFileName = "pom.xml"

    suspend fun getPackageLicensesFromJar(library: Library, project: Project):
Set<License> {
        val result: MutableSet<License> = mutableSetOf()
```

```
        val libraryRootFiles = library.getFiles(OrderRootType.CLASSES)

        for (rootFile in libraryRootFiles) {
            val rootFileChildren = rootFile.children

            yield()

            for (childrenFile in rootFileChildren) {
                yield()

                if (!childrenFile.isDirectory &&
licenseFileNamePattern.matches(childrenFile.name)) {
                    val fileText = ReadAction.compute<Document?, Throwable> {
                        FileDocumentManager.getInstance().getDocument(childrenFile)
                    }?.text ?: continue
                    result.add(getLicenseByFullText(fileText))
                }
            }

            val metaInfFolder = rootFile.findChild(metaInfoFolderName) ?: continue
            if (metaInfFolder.isDirectory) {
                val childListToProcess =
mutableListOf<VirtualFile>(*metaInfFolder.children)
                while (childListToProcess.isNotEmpty()) {
                    yield()
                    val child = childListToProcess.pop()
                    if (child.isValid) {
                        if (!child.isDirectory) {
                            if (licenseFileNamePattern.matches(child.name)) {
                                val fileText = ReadAction.compute<Document, Throwable> {
                                    FileDocumentManager.getInstance().getDocument(child)
                                }?.text
                                if (fileText != null) {
                                    result.add(getLicenseByFullText(fileText))
                                }
                            }

                            yield()

                            if (child.name == pomXmlFileName) {

PsiManager.getInstance(project).findFile(child)?.accept(PomXmlPsiElementVisito
r(result))
```

```
                    }
                }
            } else {
                childListToProcess.addAll(child.children)
            }
        }
    }
    return result.filter { it != NoLicense }.toSet()
}

fun getLicenseByNameOrSpdx(name: String): License {
    val detectedLicenseByModel =
mlDetector.detectLicenseByShortName(name)
    if (detectedLicenseByModel != NoLicense && detectedLicenseByModel is
SupportedLicense) {
        return detectedLicenseByModel
    }

    val detectedLicenseByRegex = ALL_SUPPORTED_LICENSE.firstOrNull {
it.nameSpdxRegex.matches(name) }
    if (detectedLicenseByRegex != null) {
        return detectedLicenseByRegex
    }

    return detectedLicenseByModel
}

fun getLicenseByFullText(text: String): SupportedLicense {
    val detectedLicenseByModel = mlDetector.detectLicenseByFullText(text)
    val detectedLicenseBySorensenDiesCoefficient =
sorensenDiesDetector.detectLicenseByFullText(text)

    // Handling the case when the original license is GPL-2.0-with-classpath-
exception
    // and the ml model recognizes it as GPL-2.0-only.
    // In such a situation, priority should be given to the Sorensen Dies detector.
    if (detectedLicenseByModel == GPL_2_0_only &&
detectedLicenseBySorensenDiesCoefficient != null) {
        return detectedLicenseBySorensenDiesCoefficient
    }

    if (detectedLicenseByModel != NoLicense) {
        return detectedLicenseByModel
```

```
        }

        if (detectedLicenseBySorensenDiesCoefficient != null) {
            return detectedLicenseBySorensenDiesCoefficient
        }

        return NoLicense
    }
}
package com.jetbrains.plugin.detection

import com.jetbrains.sorrel.plugin.licenses.*
import io.kinference.data.map.ONNXMap
import io.kinference.data.seq.ONNXSequence
import io.kinference.data.tensors.Tensor
import io.kinference.data.tensors.asTensor
import io.kinference.model.Model
import io.kinference.ndarray.arrays.FloatNDArray
import io.kinference.ndarray.arrays.LongNDArray
import java.io.File
import java.nio.file.Path

class MLDetector {
    //Classes for decode numeric predictions
    private val classes: List<String> =
MLDetector::class.java.getResourceAsStream(
        "/detection/license_level_classes_v2.txt"
    ).reader().readLines()

    // Detected License (string) to License class mapping
    private val licenseToClass = mapOf(
        "AGPL-3.0-only" to AGPL_3_0_only,
        "Apache-2.0" to Apache_2_0,
        "BSD-2-Clause" to BSD_2_Clause,
        "BSD-3-Clause" to BSD_3_Clause,
        "EPL-1.0" to EPL_1_0,
        "GPL-2.0-only" to GPL_2_0_only,
        "GPL-2.0-or-later" to GPL_2_0_only,
        "GPL-3.0-only" to GPL_3_0_only,
        "GPL-3.0-or-later" to GPL_3_0_only,
        "ISC" to ISC,
        "LGPL-2.1-only" to LGPL_2_1_only,
        "LGPL-2.1-or-later" to LGPL_2_1_only,
        "LGPL-3.0-only" to LGPL_3_0_only,
```

```
        "MIT" to MIT,
        "MPL-2.0" to MPL_2_0
    )

    // Model & vectorizer for detection licenses on project level initializiation
    private val mlModel: Model = Model.load(

MLDetector::class.java.getResource("/detection/license_level_model_v2.onnx").re
adBytes()
    )
    private val vectorizer: Vectorizer = Vectorizer(
        MLDetector::class.java.getResourceAsStream(
            "/detection/license_level_model_words_v2.txt"
        ).reader().readLines()
    )

    // Number of features that model accepts
    private val numFeatures = vectorizer.vector_dim + 1
    private val THRESHOLD = 0.8

    // Shape of input data
    private val inputShape = listOf(numFeatures).toIntArray()

    /**
     * Given a path to some directory. Extracts all files from this directory.
     * @param path path to directory from which to extract files.
     * @return list of Files detected objects.
     */
    private fun extractFiles(path: String): List<File> {
        val files = mutableListOf<File>()

        for (file in File(path).walk()) {
            if (file.isFile) {
                files.add(file)
            }
        }

        return files
    }

    /**
     * Removes all non-alphanumeric characters and extra non-printable symbols
from text.
     * Also transforms characters to lowercase.
```

```
     * @param text the text to be filtered.
     * @return filtered text.
     */
    private fun filterText(text: String): String {
        val re = Regex("[^A-Za-z0-9 ]")
        val cleanText = text.replace("\\s+".toRegex(), " ")

        return re.replace(cleanText.lowercase(), "")
    }

    /**
     * Removes all non-alphanumeric characters and extra non-printable symbols
from text of file.
     * Also transforms characters to lowercase.
     * @param file file from which to filter text.
     * @return filtered text.
     */
    private fun filterText(file: File): String {
        return filterText(file.readText())
    }

    /**
     * Removes all non-alphanumeric characters and extra non-printable symbols
     * from text of file located at path. Also transforms characters to lowercase.
     * @param path path to file from which to filter text.
     * @return filtered text.
     */
    private fun filterText(path: Path): String {
        return filterText(path.toFile())
    }

    /**
     * Join iterable object of Strings into one string and removes from it
     * all non-alphanumeric characters and extra non-printable symbols.
     * Also transforms characters to lowercase.
     * @param strings strings to be joined and filtered
     * @return filtered text in one string
     */
    private fun filterText(strings: Iterable<String>): String {
        return filterText(strings.joinToString(separator = " "))
    }


    /**
```

```
     * Find all files which names are license-like in given path. Traverse path in-
depth.
     * @param path path from which to start searching licenses files.
     * @return list of all found files which name are license-like.
     */
    private fun findLicensesFiles(path: String): List<File> {
        val files = mutableListOf<File>()
        for (file in extractFiles(path)) {
            if (DetectorManager.licenseFileNamePattern.matches(file.name)) {
                files.add(file)
            }
        }
        return files
    }

    /**
     * Find license-like file which is closest to path in terms of depth.
     * @param path path to which closest file to found.
     * @return file which is closest to path.
     */
    private fun getMainLicenseFile(path: String): File {
        val files = extractFiles(path)
        return files[0]
    }

    /**
     * From given text detects license class.
     * @param text text from which license to be detected.
     * @return object of detected License.
     */
    private fun detectLicense(text: String): SupportedLicense? {
        // Convert text into vector
        val filteredText = filterText(text)
        val vector = vectorizer.vectorizeWithLength(filteredText)
        val tensor = FloatNDArray(inputShape) { vector[it].toFloat()
}.asTensor("features")

        // Prediction
        val prediction = mlModel.predict(listOf(tensor))

        // Data transformation
        val predTensor = prediction[0] as Tensor
        val data = predTensor.data as LongNDArray
        val array = data.array.blocks
```

```
        val classIndex = array[0][0].toInt()
        val license = classes[classIndex]


        val unpack1 = ((prediction[1] as ONNXSequence).data as
ArrayList<ONNXMap>)[0]
        val unpack2 = (unpack1.data as HashMap<Long,
Tensor>)[classIndex.toLong()] as Tensor
        val probability = (unpack2.data as FloatNDArray).array.blocks[0][0]

        if (probability < THRESHOLD) {
            return null
        }
        return licenseToClass[license]
    }

    fun detectLicenseByFullText(fullText: String): SupportedLicense {
        return detectLicense(fullText) ?: NoLicense
    }

    fun detectLicenseByShortName(shortName: String): License {
        return detectLicense(shortName) ?: UnsupportedLicense(shortName, null,
null, null)
    }

    /**
     * From given path detects license class.
     * @param path path to a file from which license to be detected.
     * @return object of detected License.
     */
    fun detectLicense(path: Path): SupportedLicense? {
        return detectLicense(path.toFile().readText())
    }

    /**
     * From given file detects license class.
     * @param file file from which license to be detected.
     * @return object of detected License.
     */
    fun detectLicense(file: File): SupportedLicense? {
        return detectLicense(file.readText())
    }
```

```
    /**
     * Detect license for all files which name is license-like inside project path.
     * Detection is running in-depth.
     * @param path path to the project.
     * @return mapping file path to license detected in file at path.
     */
    fun detectLicensesInProject(path: String): Map<String, SupportedLicense?> {
        val fileToLicense = mutableMapOf<String, SupportedLicense?>()

        for (file in findLicensesFiles(path)) {
            fileToLicense[file.absolutePath] = detectLicense(file)
        }

        return fileToLicense
    }

    /**
     * From given path to project find the file closest to root level of project.
     * And detects license in this file. (Project license)
     * @param path path to project where to detect Project license.
     * @return  object of detected License.
     */
    fun detectProjectLicense(path: String): SupportedLicense? {
        val mainLicenseFile = getMainLicenseFile(path)
        return detectLicense(mainLicenseFile)
    }
}
package com.jetbrains.plugin.detection

import java.io.File
import kotlin.math.max

/**
 * Supportive class for Detector.
 * Transform text into vector of dimension = size of vocabulary size.
 * @property vocabulary vocabulary which use to vectorize text.
 */
class Vectorizer(val vocabulary: List<String>){
    val vector_dim = vocabulary.size

    /**
     * Vectorize (counting vectorization) given text into vector_dim size vector +
     * add length of text as last component of vector.
 *      * @param text text to be vectorized.
```

```
 * @return IntArray with vector_dim + 1 size (vector).
 */
fun vectorizeWithLength(text: String): IntArray {
    val text_vector = vectorize(text).toMutableList()
    text_vector.add(text.length)
    return text_vector.toIntArray()
}


/**
 * Vectorize (counting vectorization) given text into vector  dim size vector.
 * @param text text to be vectorized.
 * @return IntArray with vector_dim size (vector).
 */
fun vectorize(text: String): IntArray {
    // Initialize empty vector for text
    val text_vector = IntArray(vector_dim)
    val feature_count = HashMap<List<String>, Int>()
    val textList = text.split(" ")
    var max_feature_length = 0

    // Preprocessing: initialize empty mapping
    for (index in 0 until vector_dim) {
        val feature = vocabulary.get(index).split(" ")
        feature_count[feature] = 0
        max(max_feature_length, feature.size).also { max_feature_length = it }
    }

    // Going through all possible lengths and count features
    for (length in 1..max_feature_length - 1) {
        for (window in textList.windowed(length)) {
            if (window in feature_count.keys) {
                feature_count[window] = feature_count.get(window)!! + 1
            }
        }
    }

    // Convert mapping count into vector
    for (index in 0 until vector_dim) {
        val feature = vocabulary.get(index).split(" ")
        text_vector[index] = feature_count.get(feature)!!
    }

    return text_vector
```

```
    }

    /**
     * Vectorize (counting vectorization) text from file into vector_dim size vector.
     * @param file file from which text to be vectorized.
     * @return IntArray with vector_dim size (vector).
     */
    fun vectorize(file: File): IntArray {
        return vectorize(file.readText())
    }

    /**
     * Detects index of given feature in vocabulary.
     * @param feature feature which index to be found.
     * @return index inside
     */
    fun getFeatureIndex(feature: String): Int {
        for (i in 1..vector_dim) {
            if (vocabulary[i] == feature) {
                return i
            }
        }
        return -1
    }

    /**
     * Finds which feature is located in given index inside vocabulary.
     * @param index index of feature to be extracted.
     * @return feature which is located in given index.
     */
    fun getFeature(index: Int): String {
        return vocabulary[index]
    }
}
package com.jetbrains.plugin.editor

import com.intellij.diff.DiffContentFactory
import com.intellij.diff.DiffDialogHints
import com.intellij.diff.DiffManager
import com.intellij.diff.actions.impl.MutableDiffRequestChain
import com.intellij.diff.contents.DocumentContent
import com.intellij.diff.util.DiffUserDataKeys
import com.intellij.diff.util.DiffUserDataKeysEx
import com.intellij.diff.util.Side
```

```
import com.intellij.openapi.application.ApplicationManager
import com.intellij.openapi.application.ReadAction
import com.intellij.openapi.command.CommandProcessor
import com.intellij.openapi.editor.Document
import com.intellij.openapi.editor.EditorFactory
import com.intellij.openapi.editor.event.BulkAwareDocumentListener
import com.intellij.openapi.fileEditor.FileDocumentManager
import com.intellij.openapi.fileTypes.PlainTextFileType
import com.intellij.openapi.module.ModuleUtilCore
import com.intellij.openapi.project.Project
import com.intellij.openapi.ui.ComboBox
import com.intellij.openapi.util.Pair
import com.intellij.openapi.vfs.VirtualFile
import com.intellij.ui.EditorNotificationPanel
import com.jetbrains.sorrel.plugin.SorrelUtilUI
import com.jetbrains.sorrel.plugin.SorrelUtilUI.Companion.comboBox
import com.jetbrains.sorrel.plugin.detection.DetectorManager
import com.jetbrains.sorrel.plugin.diff.WordsFirstDiffComputer
import com.jetbrains.sorrel.plugin.licenses.ALL_SUPPORTED_LICENSE
import com.jetbrains.sorrel.plugin.licenses.SupportedLicense
import com.jetbrains.sorrel.plugin.model.ProjectModule
import com.jetbrains.sorrel.plugin.updateAndRepaint
import com.jetbrains.sorrel.plugin.utils.licenseDetectorModel
import com.jetbrains.sorrel.plugin.utils.logDebug

class LicenseFileEditorNotificationPanel(
    val project: Project,
    val licenseFile: VirtualFile
) : EditorNotificationPanel() {

    private val model = project.licenseDetectorModel()
    private val actionName = "Change project license file"

    init {
        text =
com.jetbrains.sorrel.plugin.SorrelBundle.message("sorrel.ui.editor.notification.lice
nse.file.title")
        myBackgroundColor = SorrelUtilUI.HeaderBackgroundColor

        val module = ModuleUtilCore.findModuleForFile(licenseFile, project)!!
        val projectModule = model.projectModules.value.find { it.nativeModule ==
module }!!
        val comboBoxCompatibleLicenses =
createComboBoxWithLicenses(projectModule)
```

```
        myLinksPanel.add(comboBoxCompatibleLicenses)

        val licenseDocument: Document = ReadAction.compute<Document,
Throwable> {
            FileDocumentManager.getInstance().getDocument(licenseFile)!!
        }

        if (licenseDocument.isWritable) {
            createUpdateLicenseFileTextActionLabel(comboBoxCompatibleLicenses,
licenseDocument)
        }
        addUpdateOnLicenseFileText(comboBoxCompatibleLicenses,
licenseDocument)

        createShowDiffLicenseFileActionLabel(comboBoxCompatibleLicenses,
licenseFile)

        model.licenseManager.modulesCompatibleLicenses.advise(model.lifetime) {
            updateAndRepaint()
            comboBoxCompatibleLicenses.updateAndRepaint()
        }
        model.licenseManager.modulesLicenses.advise(model.lifetime) {
            val moduleLicense: SupportedLicense = it[projectModule] ?:
return@advise
            comboBoxCompatibleLicenses.selectedItem = moduleLicense
        }
    }

    private fun createComboBoxWithLicenses(projectModule: ProjectModule):
ComboBox<SupportedLicense> {
        val moduleProjectLicense =
model.licenseManager.modulesLicenses.value[projectModule]!!
        val comboBox = comboBox(ALL_SUPPORTED_LICENSE)
        comboBox.isSwingPopup = false
        comboBox.renderer = LicenseListCellRenderer(model, projectModule)
        comboBox.selectedItem = moduleProjectLicense
        addUpdateLicenseFileActions(comboBox)
        return comboBox
    }

    private fun createUpdateLicenseFileTextActionLabel(
        comboBox: ComboBox<SupportedLicense>,
        licenseDocument: Document
```

```
    ) {
        val application = ApplicationManager.getApplication()
        val commandProcessor = CommandProcessor.getInstance()


createActionLabel(com.jetbrains.sorrel.plugin.SorrelBundle.message("sorrel.ui.edi
tor.notification.license.file.action.updateLicenseFileText")) {
            val selectedLicense = (comboBox.selectedItem as SupportedLicense)

            if (licenseDocument.isWritable) {
                commandProcessor.executeCommand(project, {
                    application.runWriteAction {
                        licenseDocument.setText(selectedLicense.fullText)
                    }
                }, actionName, null)
            }
        }
    }

    private fun createShowDiffLicenseFileActionLabel(
        comboBox: ComboBox<SupportedLicense>,
        licenseFile: VirtualFile
    ) {

createActionLabel(com.jetbrains.sorrel.plugin.SorrelBundle.message("sorrel.ui.edi
tor.notification.license.file.action.showDiffLicenseFile.label")) {
            val diffContentFactory = DiffContentFactory.getInstance()

            val selectedLicense = comboBox.selectedItem as SupportedLicense
            val selectedLicenseDocument =
EditorFactory.getInstance().createDocument(selectedLicense.fullText)
            selectedLicenseDocument.setReadOnly(true)
            val referenceLicenseContent = diffContentFactory.create(
                project,
                selectedLicenseDocument,
                PlainTextFileType.INSTANCE
            )
            val currentLicenseFileContent = diffContentFactory.create(project,
licenseFile)

            val chain = MutableDiffRequestChain(currentLicenseFileContent,
referenceLicenseContent)

            if (currentLicenseFileContent is DocumentContent) {
```

```
            val editors =
EditorFactory.getInstance().getEditors(currentLicenseFileContent.document)
            if (editors.isNotEmpty()) {
                val currentLine = editors[0].caretModel.logicalPosition.line
                chain.putRequestUserData(DiffUserDataKeys.SCROLL_TO_LINE,
Pair.create(Side.LEFT, currentLine))
            }
        }


chain.putRequestUserData(DiffUserDataKeys.DO_NOT_IGNORE_WHITESPAC
ES, true)

chain.putRequestUserData(DiffUserDataKeysEx.CUSTOM_DIFF_COMPUTER,
WordsFirstDiffComputer())


        chain.windowTitle = licenseFile.name +

com.jetbrains.sorrel.plugin.SorrelBundle.message("sorrel.ui.editor.notification.lice
nse.file.action.showDiffLicenseFile.vs") +

com.jetbrains.sorrel.plugin.SorrelBundle.message("sorrel.ui.editor.notification.lice
nse.file.action.showDiffLicenseFile.referenceLicenseFile")
        chain.title1 = licenseFile.name
        chain.title2 =

com.jetbrains.sorrel.plugin.SorrelBundle.message("sorrel.ui.editor.notification.lice
nse.file.action.showDiffLicenseFile.referenceLicenseFile")

        DiffManager.getInstance().showDiff(project, chain,
DiffDialogHints.DEFAULT)
    }
  }

  private fun addUpdateLicenseFileActions(comboBox:
ComboBox<SupportedLicense>) {
    comboBox.addActionListener {
        val module = ModuleUtilCore.findModuleForFile(licenseFile, project) ?:
return@addActionListener
        val projectModule = model.projectModules.value.find {
            it.nativeModule == module
        } ?: return@addActionListener
        val selectedLicense = (comboBox.selectedItem as SupportedLicense)
```

```
            val newModulesLicenses =
model.licenseManager.modulesLicenses.value.toMutableMap()
            newModulesLicenses[projectModule] = selectedLicense
            logDebug("Notification panel: Updating license of ${module.name}")
            model.licenseManager.modulesLicenses.set(newModulesLicenses)
        }
    }

    private fun addUpdateOnLicenseFileText(comboBox:
ComboBox<SupportedLicense>, licenseDocument: Document) {
        licenseDocument.addDocumentListener(object :
BulkAwareDocumentListener.Simple {
            override fun bulkUpdateFinished(document: Document) {
                val licenseDocumentText = document.text
                val license =
DetectorManager.getLicenseByFullText(licenseDocumentText)
                comboBox.selectedItem = license
```