UNIVERSITÉ DE MONTRÉAL

ON THE DETECTION OF LICENSES VIOLATIONS IN THE ANDROID ECOSYSTEM

ONS MLOUKI
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AVRIL 2016

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

ON THE DETECTION OF LICENSES VIOLATIONS IN THE ANDROID ECOSYSTEM

présenté par : MLOUKI Ons
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. MERLO Ettore, Ph. D., président
M. ANTONIOL Giuliano, Ph. D., membre et directeur de recherche
M. KHOMH Foutse, Ph. D., membre et codirecteur de recherche
M. ADAMS Bram, Doctorat, membre

DEDICATION

To my *family*

ACKNOWLEDGEMENTS

First, I want to express my deepest gratitude to my supervisors Dr. Giuliano Antoniol, for his faith in me, his support, and his good mood. And Dr. Foutse Khomh for his great guidance. They were really a continual source of learning during all our discussion and meeting.

Secondly, I would like to thank Dr. Bram Adams for his good suggestions.

In addition a particular acknowledgement for my committee members, Dr. Ettore Merlo, Dr. Giuliano Antoniol, Dr. Foutse Khomh and Dr. Bram Adams, for their valuable feedback on this thesis. For all my professors during these last two years. They were an inexhaustible source of learning.

Finally, I would like to thank all professors and students who work in SOCCER, SWAT, PTIDEJ and MCIS labs for the cordial reception they reserved to me. They provided me with a great comfort with their kindness, favour, generosity and humour.

My thoughts go to all those who have been a real help for me my parents and my husband for their continuous support. This work couldn't be achieved without their continuous encouragement.

ABSTRACT

Mobile applications (apps) developers often reuse code from existing libraries and frameworks in order to reduce development costs. However, these libraries and frameworks are governed by licenses to which developers must comply. A license governs the way in which a library or chunk of code can be reused, modified or redistributed. It can be seen as a list of rules that developers must respect before using the component. A failure to comply with a license is likely to result in penalties and fines.

In this thesis, we propose our approach for license identification in open source applications. By applying this approach, we conduct a case study to identify licenses in 857 mobile apps from the F-droid market with the aim to understand the types of licenses that are most used by developers and how these licenses evolve overtime. We conduct our study both at project level and file level. We also investigates licenses violations and the evolution of these violations overtime; we compare licenses declared at the project level, file level and those of the libraries used by a project to seek for licenses that are incompatible and used in the same project.

Results show that most used Licenses are GPL and Apache licenses both at the project level and file level. In many cases we noticed that developers didn't pay too much attention to license their source code. For 3,250 apps releases out of 8,938 releases, the apps were distributed without licenses information. Regarding license evolution, we noticed that the probability for a project to stay under the same license is very high (95% in average) and in case of change, changes are generally toward more permissive licenses. At the file level, we noticed that developers tend to delay their decision about license selection, also in 15% of license changes, developers removed licensed information. We identified 15 projects out of 857 projects, with a license violation; 7 projects had violations in their final release. To solve license violations, developers either changed the license of some of the apps' files or removed the contentious files from the apps. It took in average 19 releases to solve a license violation.

These findings suggest that developers of mobile apps may be having some difficulties in understanding the legal constraint of licenses' terms or it may be that the lack of consistency and standardization in license declarations fosters confusion among developers. Our license detection approach can be used by developers to track license violations in their projects.

RÉSUMÉ

Très souvent, les développeurs d'applications mobiles réutilisent les bibliothèques et les composants déjà existants dans le but de réduire les coûts de développements. Cependant, ces bibliothèques et composants sont régies par des licences auxquelles les développeurs doivent se soumettre. Une licence contrôle la manière dont une bibliothèque ou un bout de code pourraient être réutilisés, modifiés ou redistribués. Une licence peut être vu comme étant une liste de règles que les développeurs doivent respecter avant d'utiliser le composant. Le non-respect des termes d'une licence pourrait engendrer des pénalités et des amendes.
A travers ce mémoire de maîtrise, nous proposons une méthode d'identification des licences utilisées dans une application à code source ouvert. A l'aîde de cette méthode, nous menons une étude pour identifier les licences utilisées dans 857 applications mobiles, provenant du marché "F-Droid", dans le but de comprendre les types de licences les plus souvent utilisées par les développeurs ainsi que la manière avec laquelle ces licenses évoluent à travers le temps. Nous menons notre étude sur deux niveaux; le niveau du projet et celui du fichier. Nous investigons également les infractions portées aux licences et leursévolutions à travers le temps; Nous comparons les licences déclarées au niveau du project avec celles de ses fichiers, des fichiers entre eux et des projets et fichiers avec ceux des bibliothèques utilisées par le projet, afin d'identifier des licences incompatibles utilisées dans un même projet.
Les résultats montrent que les licences les plus utilisées sont les licences "GPL" et "Apache"; aussi bien au niveau du projet qu'au niveau fichiers. Nous remarquons que, dans plusieurs cas, les développeurs ne portent pas assez attention aux licences de leurs code source. Des 8 938 versions d'applications analysées, 3 250 versions ne sont pas accompagnées d'informations relatives aux licences. Concernant l'évolution des licences, nous remarquons que la probabilité pour un projet de demeurer sous une même licence est très élevée (95% en moyenne), et dans le cas d'un changement de license, le changement se fait généralement vers des licences plus permissives. Au niveau du fichier, nous avons remarqué que les développeurs ont tendance à retarder leur choix de licence. Dans 15% des changements de license, les développeurs retirent les informations relatives aux licences. Parmi les 857 projets analysés, nous avons identifier 15 projets contenant des infractions concernant les licences. 7 de ces projets contenaient encore des infractions dans leur version finale. Dans les autres cas, pour résoudre les infractions, les dévloppeurs ont changés les licences liés à quelques fichiers de l'application; ou ont retirés les fichiers problématiques des applications. En moyenne, 19 versions de l'application étaient nécessaires pour résoudre les infractions portées aux licences.
Ces résultats sont une indication que les développeurs ont de la difficulté à comprendre les con-

traintes légales des termes des licences. Une autre explication est que le manque de cohérence et d'uniformisation des déclarations des licences créent une confusion chez les développeurs. Notre méthode de détection des licences pourrait être appliqué par les développeurs afin de traquer les infractions portées aux licences dans leurs projets avant la mise en marché.

CO-AUTHORSHIP

Earlier study in the thesis was published as follows:

- On the Detection of Licenses Violations in the Android Ecosystem
  Ons Mlouki, Foutse Khomh and Giuliano Antoniol , in *Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER), Osaka, Japan, 16-18 March, 2016.*
  My contribution: methodology and analysis, paper writing, and presentation at the conference.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

When developing a new software, developers often reuse code chunks and components that have been made available under a variety of licenses (*e.g.*, Apache, BSD, GPL, or LGPL). Software licenses govern the way a software component or chunk of code can be reused or distributed. These software licenses describe the liabilities and responsibilities of parties interested in using, modifying, or redistributing software artifacts. Unlike proprietary licenses, open-source licenses allow access to the source code at any time and without any restriction. However, reuse and–or distribution are often limited by certain conditions [3]. For example, the Section 5 of the GPLv3.0 license [4] states the following about code modification: *"You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy"*. Hence, the use of components governed by different licenses into a same software system can generate a licence violation if the rules of the different licenses are not compatible. This is the case for GPLv2 and Apachev2. In fact, GPLv2 requires that any software system using a component licensed under its terms should be licensed under the GPLv2 license while the Apache Software Foundation requires that all Apache software must be distributed under the Apachev2 license. These two licenses are therefore incompatibles and any software system that contains components under these two licenses (at the same time) is exposed to penalties [5]

With the very high speed of apps development (on average 2,371 new apps are published in Google Play [6] every day), developers are increasingly inclined to reuse code from other open source projects. Vendome et al. [7] who investigated how developers of open-source projects adopt and change licenses found that these developers often experience difficulties understanding licensing terms. These factors combined with the large number of available licenses (*i.e.*, more than 70 open source licences exist today [3]) that developers could choose from, makes license violations very likely. Therefore, apps development teams should track and correct any license violation before releasing their software to the public. In this thesis, we aim to examine the state of license usages and violations in the Android Ecosystem.

1.1 Research Statement

Prior research have studied license usages in open-source software systems, but to the best of our knowledge, none of the studies have specifically targeted mobile apps. Mobile apps developers perform substantial software reuse [8], making licenses violations very likely. In this thesis, we investigate license violations on a set of applications from the F-droid market,

taking into consideration the different artifacts of the apps. We also investigate the evolution of licenses overtime as well as the evolution of license violations.

1.2 Thesis Overview

- *Can we accurately detect license information from mobile apps (Chapter 3)?*

  Ninka is considered to be the state-of-the-art tool for license identification. However, Ninka cannot detect the license information of a library used by an application. In this thesis, we propose a novel approach named LIT (License Identification Technique), that can detect license information from both the source code of an application and the libraries used by the apps. We evaluate and compare the performance of LIT with the performance of Ninka.

- *How do mobile apps licenses evolve overtime (Chapter 4)?*

  In this chapter, we examine the evolution of licenses overtime both at project and file levels, in order to understand the main patterns of licenses evolution in the Android Ecosystem.

- *Can we accurately detect and track licenses violations in mobile apps (Chapter 5)?*

  In this chapter, we study licenses violations in mobile apps and examine the evolution of license violations overtime, with the aim to understand if and–how developers address these violations.

1.3 Thesis Contribution

This thesis make the following contributions:

- We propose a new approach LIT to identify licenses in mobile apps, while taking into account the libraries used by the apps. We validate our proposed approach on a set of 857 Android apps and their releases (8,938 release in total). Results show that LIT outperforms the state-of-the-art approach Ninka when detecting license information from source code. In fact, LIT could recovered 16% of license information missed by Ninka. Thanks to LIT we conducted a study on the license usages, to figure out the preference of developers in license choices. We found that at project level, release level

and file level, the most used licenses are first GPL and then Apache.

- In Chapter 4, we observed that many apps change their licenses overtime. We conduct a study on licensing evolution over releases both at project and file levels, in order to identify the main licenses evolution patterns followed by developers of mobile apps. Our goal was to determine if overtime, developers tend to make their apps more–or–less open for reuse. Results show that developers often change the license of their mobile apps towards a less restrictive license.
- In Chapter 3, we proposed a methodology for license violation detection. We applied this methodology on our 857 apps and observed that 85% of apps contain files with potential licenses violations. We investigated the domains of the apps (*e.g.*, Games, Multimedia, office, etc...) involved in these potential violations and found that the similar files that are licensed under different terms mostly belong to apps from different domains. This suggests that developers tend to copy code from apps that are not in the same category as their app (*i.e.*, a Games app vs. a Multimedia app). We do not claim license violations for the 85% of apps that contain files with inconsistencies because although these apps contain files that share code with files licensed under a different (conflicting) license, it is possible that the two apps copied the code from a third app that released the code under a dual license, and the two apps simply picked different licenses. We investigated license violations in more details and analyzed their evolution overtime. Out of the 857 studied apps, we found 17 apps with clear license violations. These 17 apps have inconsistencies between their declared licenses and the licenses of their files and–or the licenses of their dependencies. For the 17 projects that clearly violate license terms, 10 of them corrected the issue after 19 releases in average. The remaining 7 projects still had a license violation at the time of this study.

1.4 Organisation of the Thesis

The rest of this thesis is organised as follows:

- Chapter 2 outlines literature review in the areas of license identification, license evolution, and license violations.
- Chapter 3 describes our approach for license identification.
- Chapter 4 presents our empirical study on license evolution.
- Chapter 5 presents our empirical study on license violation detection.
- Chapter 6 summarises and conclude the thesis and discuss future work.

CHAPTER 2 LITERATURE REVIEW

In this chapter, we present our critical review of the literature related to license detection, license evolution, and license violation.

2.1 License identification

A license statement is a textual information included in the top of each source code file or in a separate textual file often named (COPYING, LICENSE, README or POM), and stored under the main repository of the project. It includes copyright information, such as the names of contributors to a source code file, the copyright owner, warranty, and liability statements. Many tools have been proposed to identify License statements in source files or license files (*e.g.*, README files): Ninka [1], FOSSology [9] and OSLC[1]. In the following, we discuss each of these tools in more details.

FOSSology, a tool proposed by Gobeille [9], uses a binary symbolic alignment matrix pattern matching algorithm (bSAM) [10] to compare two sequences of symbols against each other and determine whether they are similar. To identify the license statement of a file, FOSSology matches the text of the file with templates of license contained in a database. If a match is found, the matching license is reported as the license of the file. FOSSology is reported to have a low speed and precision [1].

OSLC, is a tool based on regular expression, that finds exact or partial matches against licenses stored in a database. The project was started by five developers and released on sourceforge on December 2005. The main disadvantage of OSLC is its low precision [1].

Ninka, state-of-the-art tool for license identification, proposed by German et al. [1], identifies the license of a file by matching sentences in the license statement with sentences stored in Ninka databases. Once it extracts license statements, Ninka performs a text segmentation to extract sentences from these statements. The sentences are then normalizes using known sentences of licenses. This step is primordial to remove any syntax mistakes that might have been into the license statement. Relevant sentences are retained and a sequence of tokens is generated from them. Finally, Ninka matches the tokens with a set of predefined tokens from existing licenses. Although Ninka needs predefined rules (license sentences) to perform license identification, at the time of this writing, Ninka can identify 112 most known OS licenses and it can be extended. Unlike OSLC, whenever license informations are missing,

[1] http://oslc.sourceforge.net/

Ninka explicitly reports that it can not recognize a license. For license analyses studies, it is important to distinguish between a file without license information and a file with an unknown license.

Table 2.1 Evaluation of license identification tools [1]

Tools	**Recall**	**Precision**	**Execution Time**
Ninka	82.3%	96.6%	22s
FOSSology	99.2%	55.0%	923s
OSLC	100%	29.5%	372s

The aforementioned tools can identify license information from source code files, however, binary files are often the only data available for a project. To address this issue, researchers have also developed approaches to identify license informations from binary files.

Di Penta et al. [11] proposed an approach to identify the license of Jar archives using a code search engine. Their proposed approach consists in querying a code search engine for matching and mining the textual files in jars using the FOSSology tool. Although the approach was validated on a sample of 37 jars and reported a 95% precision, it is based on Google Code Search[2] for finding matching. Having qualified package and class name found by ASM bytecode analysis library[3] into jars, the search engine returns only the license's family name of the class if found. Also the approach is not capable of matching the binary jar to a precise version.

Davies et al [12] downloaded libraries from the Maven2 repository, computed signatures (classes and method signatures) for these libraries and stored them in a database. They also developed the tool Joa that can compute the signature of any new library and match it against the database (both source code and binaries) to identify the license of the new library and its specific version. They were able to get provenance information for over 95% of the subject archives (84 open source binary archives found in a proprietary e-commerce application). German et al. [13] used the Joa tool to track the provenance of 57 jars that were used in a commercial product. Knowing the identity of the libraries used in the product and since its source code is available in Maven2, they perform license identification from the source code using the Ninka tool.

Among the three tools presented above, we selected Ninka for our study, since it is reported to have a high accuracy and performance (see Table 2.1). To identify library provenance for license identification, we use Joa tool.

[2]https://code.google.com/p/chromium/codesearch

[3]http://asm.ow2.org/

2.2 Licensing evolution

When the license of a component is modified, applications making use of the component have to be updated to adjust to the change. License changes can have harmful consequences on software reuse. This was the case when the license of the security package IPFilter added a condition incompatible with the license of OpenBSD to its license terms [14]. OpenBSD had to replace the IPFilter package by an OpenBSD based implementation of the same features.

Also, we can cite the case of MySQL client libraries, which changed their license from LGPLv2 to the GPLv2 license. This change was made to prevent the usage of MySQL within proprietary products. However, this change had an impact on PHP systems since they can no longer connect to MySQL, because the PHP license is incompatible with the GPLv2 license. Eventually, this problem was solved by adding the MySQL FLOSS Licence Exception [15]. Licenses are also changed to facilitate software reuse. This was the case for Java JDK which was distributed under the Common Development and Distribution Licence (CDDL) until novembre 2006. Because CDDL is incompatible with GPL licenses. Sun Microsystems collaborated with the FSF (Free Software Foundation) to change the license of Java to GPLv2 (in order to encourage its usage). However, any system that runs under the JVM is considered as a part of it and needed to be licensed under the GPLv2 license. To solve this issued Sun added the Classpath exception to the license in Java 5.0. Enabling Java programs to be released under any license as long as it satisfy the Classpath requirements [16].

Another example of license change toward a more permissive license is Mono[4], which is an open-source framework proposed by Novell to support .NET software systems under other operating systems than Microsoft Windows. Mono class libraries were originally released under LGPL, this created a problem since running .Net systems could be considered as a derivative work of Mono [17]. To solve the issue, Mono's developers decided to change the license to MITX11 license, which allows developers to create free open source or proprietary systems with it.

The license of a software can also be changed to impede the development of a competing product. In 20XX KDE, the desktop suite for Unix systems was attacked for using the Qt cross-platform GUI toolkit [18]. The problem was due to the fact that many applications in KDE were under GPL and Qt was under Q Public License, which is incompatible with GPL. To solve the issue, the Harmony project was started, to create a Qt replacement that would be licensed under GPL. To stop this competing project, Trolltech the company behind Qt decided to change the licence of QTv3 to GPL.

[4] http://mono-project.com

Researchers have investigated this important phenomenon, *i.e.*, license changes, with the goal to understand the context of licensing changes and identify the most relevant patterns for license changes.

Di Penta et al. [19] proposed a four steps method to track files licensing evolution across releases. First, they extracted licensing statement using a comment extractor. Second, They identified changes in licensing statements by comparing licensing statements of file revisions. The third step consisted in classifying licenses. They used the FoSSology tool for license identification and experienced its hight execution time. In the final step of their study, they performed the identification of changes in copyright years. The proposed method was applied on six well known open source systems: ArgoUML, Eclipse-JDT, the FreeBSD and OpenBSD kernels, the Mozilla Suite, and Samba to better understand when and how developers change licensing statements in source code files and the impact of these license changes. Based on the analysis of those six open source systems, they observed that many files experienced changes in their licensing statements. Many files started without license information and in later releases the license was added. Other files moved toward less restrictive licensed. Those finding needs to be validated by further studies since the low precision of the tool used for license identification (55%) is a threat to validity. Also, despite the variety of systems that were analyzed (belonging to different domains and developed under different programming languages), more studies are required to make their findings generic.

Vendom et al. [20] conducted a large study on license usage and changes in a set of 16,221 java projects from Github, analyzing source code files, tracing commit notes and discussions related to license changes and identifying the most common license change patterns. They extracted commit information (*e.g.*,commit message, file paths, changes applied to files, license changes occurrences, etc...) using The Markos code analyzer [21], which uses Ninka for license identification. In their study, they were interested in atomic license changes without considering the occurrence of license changes within the same project. They found that 1,833 projects among 16,221 projects experienced an atomic license change. Regarding discussions related to license changes, they found that only 0.9% of commit messages and 0% of issue discussion reports were related to license issues. They concluded that developers are very shy in documenting license changes. They also observed a trend towards the usage of permissive licenses like Apachev2 and MIT; licensing change patterns were toward less restrictive licenses. The authors didn't investigated the reasons for the observed license evolutions. Also, their analysis were limited to java projects from Github. In a follow up study [7] the authors conducted a survey to understand when and why developers adopt and change licenses. They found that developers have some difficulties in dealing with license terms *e.g.*,with incompatible licenses, and that they change licenses toward more permissive licenses to facilitate the

reuse of their product in commercial systems.

In our study, we analyse a sample of applications (857) from an Android Market and study license evolution across their releases. We consider also hybrid applications with c and cpp source code for diversity. We aim to validate previous results on Android Market and study in depth the causes of license evolution.

2.3 License violations

The reuse of different pre-existent components to build a new application can lead to licenses violations. In March 2011, OpenLogic, a company that advocates for (and helps ensure) the proper use of open source software ran their private OSS Deep Discovery license compliance scanner against 635 apps from both the Android Market and the Apple App Store, and found that 71% of the apps that were using open source code failed to comply with the terms of the open source licenses. The apps did not license their code properly. In the case of Apache licenses, information about the notice/attribution of the licenses were missing. For GPL licenses, developers of the apps failed to provide information about how to access the code. The detection of license violations in this study was done manually. Although these developers were not pursued in court for their infringements, penalties for licensing violations can be severe. In August 2012, Samsung was found guilty of violating Apple's iPhone and iPad technologies licenses, and condemned to pay to Apple a billion dollars in damages. In march 2015, VMware was also sued by Christoph Hellwig, a Linux developer and the nonprofit organisation SFC (Software Freedom Conservancy) over an improper use of the Linux kernel [5]. Christoph and SFC claimed that VMware violated the terms of the copyright license of the Linux kernel. Sojer et al [22] [23] conducted a survey with 869 developers and found that reusing open source code is so common but developers have limited knowledge on licenses terms and thus they don't understand the associated legal risks and this is due to the fact that industry and academic institutions do not give importance and training regarding licenses and their impact on code reuse. In march 2016, Oracle was asking for more than 9 billions dollars (yes, billions!) to Google for copyright infringement of the Java APIs claiming that it used 37 Java API packages to create its Android mobile operating system [24].

These evidences highlight the importance of tracking and correcting eventual licenses violations early on before the distribution of a software system. Two main categories of open source licenses exist: restrictive licenses (also known as copyleft or reciprocal), and permissive licenses. Restrictive licenses require that developers use them to distribute their software if they happen to use a component under such license, *e.g.*,GPL license; "You must license the

entire work, as a whole, under this License to anyone who comes into possession of a copy" [4]. However, permissive licenses allow the distribution under a different license, *e.g.*,BSD, MIT licenses [25]. License violations concern restrictive licenses. Researchers has investigated licenses violations with the goal to understand the context in which they occur and how developers address them.

German et al. [3] manually examined 124 OSS packages to understand the way in which developers solve license incompatibilities. They built a model to document integration patterns that are used to help developers to solve license inconsistency issues and highlighted the characteristics of certain licenses and their applicability. In [26], authors propose a semi-automatic method to detect licenses incompatibilities in software packages. They compared the declared licenses of binary packages with the licenses of their source files identified by the Ninka tool and license of dependent packages documented in the dependency graph identified by *rpmquery*, to identify possible inconsistencies. They validated their approach on 3,874 binary packages from the Fedora-12 GNU/Linux distribution.

Companies, like Black Duck[5] and HewlettParckard[6] propose their own infrastructure to help users avoid license incompatibility issues. They build their own databases and validate that it does not contain any license violation. The databases are then provided to users who can reuse code from them without any fear of license violation. Although this technique solves license incompatibility issues and makes developers feel safe, it is quite limited by the scope of the databases that are built.

Hemel et al. [27] introduced BAT, a tool that detects code cloning in binaries. They implemented three binary clone detection techniques. The tool helps users to detect clones between a subject binary and the binaries from their repository to identify license violations *e.g.*,using third-party software packages licensed under GPL in a binary that could be used in a proprietary product. Although in theory BAT could track license violations, in practice, its accuracy is poor.

German et al. [13] proposed *Kenen*, a semi-automatic approach to verify license compliance in java applications. Kenen identifies the license of souce code using the Ninka tool. It also identifies dependancies' license by combining both Joa (to identify jar provenance in order to obtain the source code) and Ninka (to identify license information from the obtained source code). With the given license informations, authors manually identified license violations. Kenen has been applied to one proprietary application (an editor of music files).

In our study we analyse a sample of Android applications (8,938) from Github to identify

[5]http://blackducksoftware.com
[6]http://www.hp.com

license violations automatically, using a table specifying licenses that cannot co-exist together.

2.4 Chapter Summary

In this chapter we have reviewed the relevant literature on license identification, license evolution and license violation. For license identification, we have in one hand license identification techniques that can be applied to source code, in this category Ninka performs better than other existing tools, however in some cases it fails to identify licenses existing into source files as well as the licenses of source files that are not explicitly declared. On the other hand we have license identification techniques for binaries. This second category is mostly composed of derivatives of the Joa tool. Most license evolution studies are performed using a license identification tool with low precision and/or a small number of applications and are often limited to projects written in specific programming languages. Concerning license violations, existing approaches considers source code and binaries separately. Also, companies, like Black Duck have proposed the use of proprietary databases of code that can be safely reused in order to prevent license violations. In this thesis we will propose a methodology for license identification that improves over the limitations of Ninka. We will study license evolution in Android applications with the aim the identify the most common license evolution patterns. We will also study license violations and the evolution of license violations in our subject apps, considering both the source code of the apps and the libraries used by these apps.

In the following chapter, we describe our approach for license identification and the different observations we made related to the licensing practice in the Android Ecosystem.

CHAPTER 3 LICENSE IDENTIFICATION

In this thesis, our goal is to examine licenses usages and violations in the Android Ecosystem. To achieve this goal, we first need to identify license informations from mobile apps. License information can be found in source file as a license statement or in a separate text document generally included in the main directory of the project. Moreover, this textual information can be inconsistent across the files of a system because of developers failing to update them properly. Android apps as many projects may use some external libraries. Often, those libraries are in the binary format, making it impossible to retrieve their license information through a textual analysis. In this chapter we introduce our license identification technique named LIT (License Identification Technique) which leverages the state-of-the-art tool Ninka to identify license information from textual files, the clone detection technique CCFinderX, to track missing license informations using similarity between files, and Joa to identify the license of libraries used by apps. Figure 3.1 presents an overview of LIT. We conduct a case study with 857 mobile apps to assess the effectiveness of LIT, answering the following two research questions:

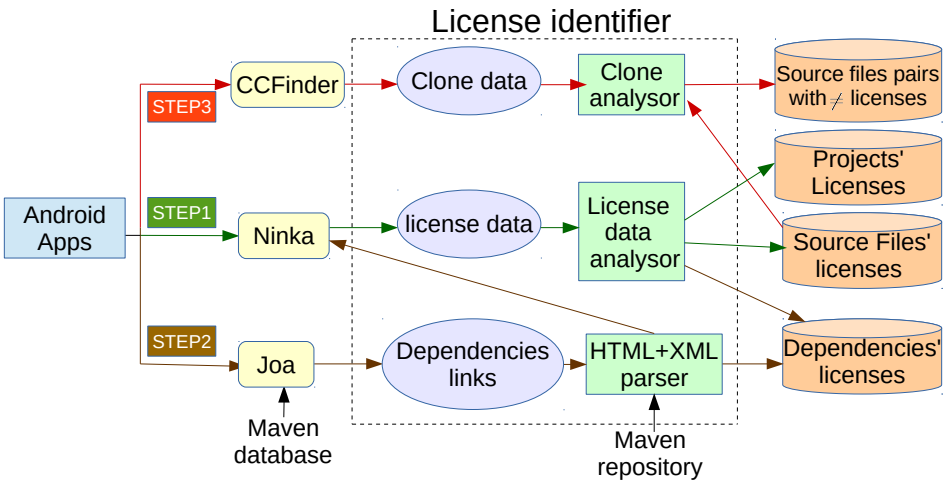

Figure 3.1 Identification of license information

RQ1: *Could LIT outperform the state-of-the-art approach Ninka when identifying the license of mobile apps?*

License statements are textual informations that can be changed during the evolution of a project, which often results in inconsistent informations or files with missing license statements. In these cases, tools like Ninka are unable to accurately identify the license

information of the projects. Our proposed approach LIT relies on clone detection techniques to identify similar code files and transitively recover missing license informations. This research question aims to evaluate the effectiveness of this approach.

RQ2: *Could LIT identify the license of libraries used by a mobile app?*

Libraries are generally composed of binary files. Hence, it is difficult to identify library's license informations since this information is absent from binaries. Traditional approaches like Ninka cannot be applied to binary files. LIT makes use of the API provenance tracking approach Joa proposed by Davies et al. [12] to identify the location of the source code of libraries in order to detect license informations. This research question assesses the effectiveness of this approach.

3.1 LIT: A clone detection based approach to identify the license of mobile apps

In this section, we explain the steps of our approach LIT, for license identification. Figure 3.1 summarizes the steps of this approach.

3.1.1 Step 1: Identification of license statements

A license statement is a textual information about license, generally found on the top of a source code file (in this case it indicates the file's license) or within a text file, so often named (COPYING, LICENSE, README or POM) and located in the main directory of the project (in this case it indicates the project's license). Since Ninka is reported to have a high accuracy and performance when identifying license information from text files, we selected it for the identification of license information from source code files. To identify license information contained in a file, Ninka splits statements contains in the file into textual sentences, normalizes them, and matches them against known licences tokens. Whenever, it cannot find any match or the license information is missing, Ninka explicitly reports that it can not recognize a license, outputting the keyword "UNKNOWN" and "NONE" respectively. In such case, we move to step 3.

The license information identified by Ninka both form source code files and textual files found in the main directories of the projects are stored into a database.

3.1.2 Step 2: Identification of the licenses of libraries used by Apps

Mobile apps frequently use externally developed libraries to provide their services. However, these libraries are often deployed in the form of binaries, making it impossible for Ninka, or

any of the tools mentioned in Chapter 2, to extract licenses information from these libraries. To address this issue, we make use of the Joa tool proposed by Davies et al. [12]. Joa allows to track the provenance of software artifacts (both source code and binaries). In fact, Davies et al. [12] downloaded libraries from the Maven repository, computed signatures for these libraries and stored them in a database. They also developed the tool Joa that can compute the signature of any new library and match it against the database to identify the identity of the new library. Knowing the identity of the libraries used by our studied mobile apps allows us to extract their license information by parsing the POM files available in the Maven repository. In POM files, license informations are stored under a tag named license or as a license statement in the top of the file. We use Ninka to extract license information from license statements and for license informations stored under the license tag, we use an xml parser. Figure 3.2 presents an example of POM file in which we have both the information in the statement and under the tag. In such case, we consider only the information under the license tag. In this study we use the same database as Davies et al [12], which contains all libraries that were available in the Maven repository in April 2013.

```
          <?xml version="1.0" encoding="UTF-8"?>
<!--
  ~ Copyright (c) 2011 Roberto Tyley
  ~
  ~ This file is part of 'Agit' - an Android Git client.
  ~
  ~ Agit is free software: you can redistribute it and/or modify
  ~ it under the terms of the GNU General Public License as published by
  ~ the Free Software Foundation, either version 3 of the License, or
  ~ (at your option) any later version.
  ~
  ~ Agit is distributed in the hope that it will be useful,
  ~ but WITHOUT ANY WARRANTY; without even the implied warranty of
  ~ MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
  ~ GNU General Public License for more details.
  ~
  ~ You should have received a copy of the GNU General Public License
  ~ along with this program. If not, see <http://www.gnu.org/licenses/>.
  -->

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
       <modelVersion>4.0.0</modelVersion>
       <version>1.0-SNAPSHOT</version>
       <groupId>com.madgag</groupId>
       <artifactId>agit-parent</artifactId>
       <packaging>pom</packaging>
       <name>Agit parent POM project</name>
                              ---
                              ---
                              ---
   <licenses>
       <license>
           <name>GPL v3</name>
           <url>http://www.gnu.org/licenses/gpl-3.0.html</url>
           <distribution>repo</distribution>
       </license>
   </licenses>
                              ---
                              ---
                              ---
</project>
```

Figure 3.2 An Example of POM file from Agit-an Android Git client

3.1.3 Step 3: Identification of similar source files

To identify the license of files with missing license statements or files for which Ninka could not recognize the license, we use clone detection techniques to identify source files pairs that

share similar code.

Among multiple clone detection tools that exist, we choose to work with CCFinderX [28] since it is known to scale well. CCFinderX represents the content of source code files as sequences of tokens and uses a suffix-tree matching algorithm to compute matchings. Clone location information is represented as a tree with sharing nodes for leading identical subsequence. It gives as a result a list of clone pairs. We consider that two files should have the same license if the portion of cloned code that is shared between the files is more than 50 tokens (which represents about 20 lines of code). We consider only Type 1 and Type 2 clones in our study.

3.2 Study Design

In this section, we perform a case study to assess the effectiveness of LIT at identifying the licenses of mobile apps. We aims to answer the following two research questions:

RQ1: Could LIT outperform the state-of-the-art approach Ninka when identifying the license of mobile apps?

RQ2: Could LIT identify the license of libraries used by a mobile app?

3.2.1 Data Collection

We crawled the F-droid Website and downloaded 857 mobile apps from the Android Market F-Droid. We chose these apps because their source code is available on Github[1]. Next, for each app, we downloaded all its releases (8,938 releases in total) from the Github repository. Figure 3.3 shows the distribution of our data across the categories of apps.

3.3 Results

This section presents and discusses the results of our two research questions.

RQ1: Could LIT outperform the state-of-the-art approach Ninka when identifying the license of mobile apps?

By using CCFinderX to track similar files that should be under the same license, LIT could identify the licenses of many files that Ninka failed to identify. Figures 3.4 and 3.5 present the gain achieved in license information, taking into account different ranges of clone sizes (number of tokens). Overall, LIT could recover 16% of the information missed by Ninka.

[1]http://github.com

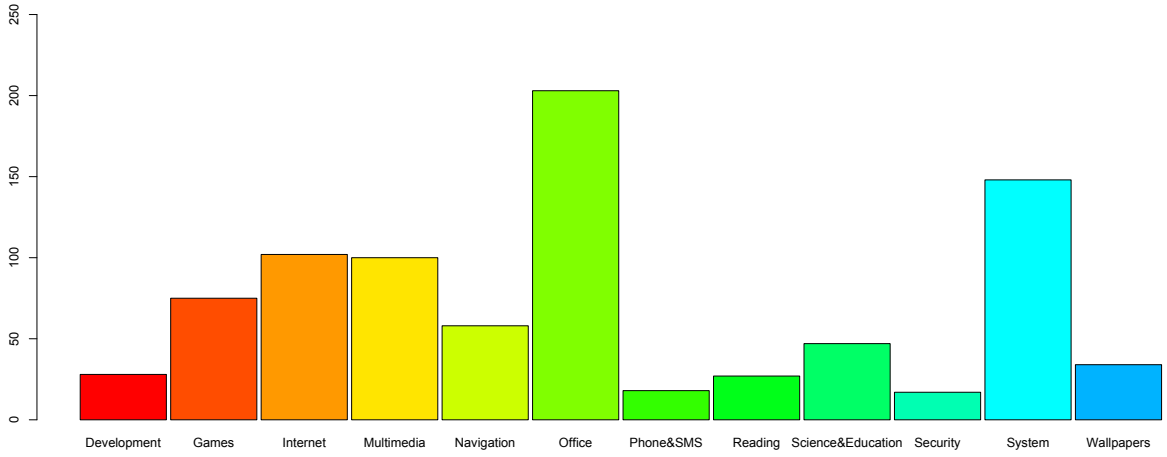

Figure 3.3 F-Droid categrories

The longest clone fragment found in our data set is composed of 54,540 tokens. The average number of tokens in the clones found is 127 tokens, *i.e.*, 55 lines. As observed in Figures 3.4 and 3.5, there is still a considerable number of file for which we could not identify the license information.

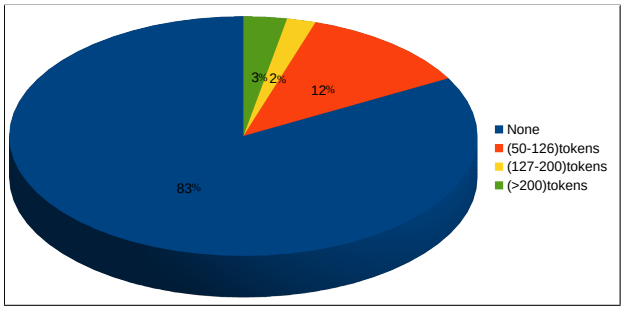

Figure 3.4 Identification of files' license reported as NONE by Ninka

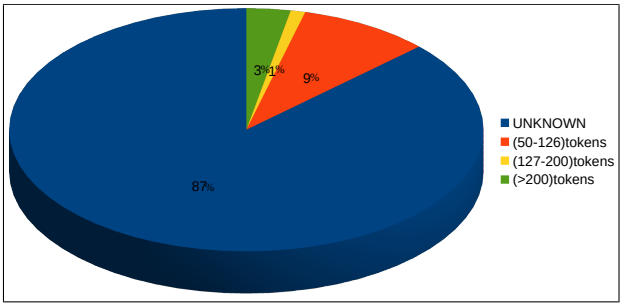

Figure 3.5 Identification of files' license reported as UNKNOWN by Ninka

Moreover, Table 3.1 summarizes inconsistencies found among the licenses statements of our studied apps. In this table, we highlight two harmful cases of inconsistencies that include incompatible licenses "GPLv2-Apachev2" and "GPL-Other".

We investigated the domains of apps involved in these inconsistencies and found that the similar files that are licensed under different terms mostly belong to apps from different domains (as presented in Table 3.2). This result suggests that developers tend to copy code

Table 3.1 Inconsistencies found among the license statements of the studied apps

Kind of disagreement	**#Files**	**#Releases**	**#Apps**
GPLv2-Apachev2	54 335	1 067	70
GPL-OTHER	18 853 940	7 141	685
Others	18 284 150	-	-
Apps with license inconsistency	-	-	731

from apps that are not in the same category as their application. This was the case for 60,72% of the inconsistencies found.

Table 3.2 License inconsistencies by categories

Category	**Part of desagreement (%)**
Inconsistent files from different domains	60,72
InternetApps	16,62
OfficeApps	14,69
ScienceAndEducationApps	3,65
ReadingApps	1,75
SystemApps	1,09
GamesApps	0,08
MultimediaApps	0,29
DevelopmentApps	0,09
NavigationApps	0,73
PhoneAndSMSApps	0,28
SecurityApps	0,01

RQ2: Could LIT identify the license of libraries used by a mobile app?

Figure 3.6 presents the licenses of the libraries that LIT could recover from our studied apps. On this Figure (*i.e.*, Figure 3.6) we report only libraries that perfectly matched a known library from the Maven repository; *i.e.*, the proportion of files in the library that matched the files contained in a library from the Maven repository is 100%. We opted for this conservative approach because if a library used by an app is licensed under Apachev2 and only matches 50% of the files in a library from the Maven repository that is licensed under GPLv3, we cannot know for sure if the app is using the part of the features provided by the files under the GPLv3 license, and hence, we cannot conclude on a potential license violation for example. In fact, a GPLv3 library can contain files licensed under Apachev2 (since a project containing Apachev2 sources code and GPLv3 source code must be licensed under GPLv3) and the app may be using only the Apachev2 files.

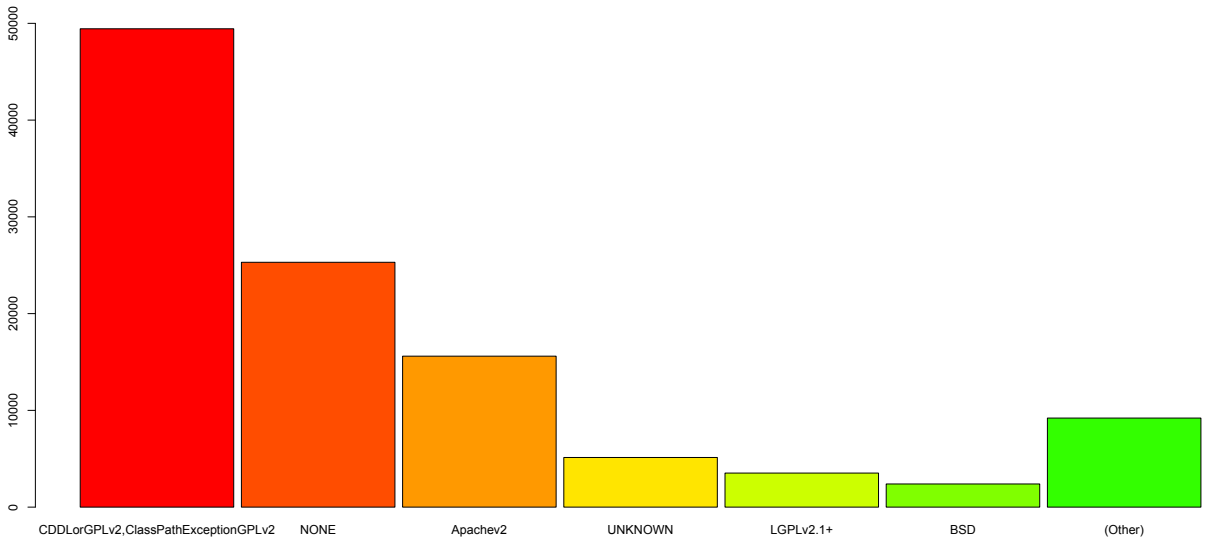

Figure 3.6 Licenses of libraries used by the studied apps

3.4 Discussion

Even though Ninka does a great job at identifying license identification from text files [1], it relies blindly on the textual information declared in the files and in some cases fail to recognize the license of the files. More over in case where the license information is missing, Ninka reports NONE. LIT combines Ninka with a clone detection technique to overcome this limitation and results show that this combination could recover 16% of the information missed by Ninka; improving recall. By tracking similarities between files, LIT also found license inconsistencies in 85% of the studied apps, *i.e.*, 85% of the apps contain files that share code with files licensed under a different (conflicting) license. Although these inconsistencies could be symptoms of license violations, it is possible that the two apps copied the code from a third app that released the code under a dual license, and the two apps simply picked different licenses. For this reason, we can not confirm that the inconsistencies found are violations of licenses terms unless we dispose of information about the origin of the cloned code.

By leveraging the provenance analysis approach proposed by Davies et al [12], LIT could identify the license of 75% of the libraries used by the studied apps. We expect this percentage to increase if we can access a recent copy of the Maven repository.

3.5 Threats to Validity

This section discusses the threats to validity of our study following the guidelines for case study research [30].

Construct validity threats concern the relation between theory and observation. In this study, the construct validity threats are mainly due to measurement errors. We rely on Ninka, a state-of-the-art tool for license identification from textual files. It has 97% precision and 82% recall. We rely on CCFinderX to track similarities between code files. Although, it doesn't have a 100% precision and recall, CCFinderX has the advantage that it can process a large amount of data in a reasonable time [31]. It has been used successfully in previous studies on clone detection [32]. We randomly selected 100 clone pairs from our obtained results (which corresponds to a confidence level of 95% and a confidence interval of 10) and manually examined them, to assess the precision of CCFinderX. Out of the 100 clone pairs, 57 were true clone pairs. This result is in line with the performance of CCFinderX reported by [31]. The precision of the Joa tool has a limited impact on our results since we considered only libraries that were perfectly matched. To the best of our knowledge, Joa is the only tool that can detect the provenance of software artifacts. Our study relies on a copy of the Maven repository that was obtained in 2013, therefore, it is possible that some libraries are missing. However, it is no more possible to obtain a full copy of the Maven repository. Nevertheless, it is possible that the license of some of the libraries from our Maven repository copy were changed after 2013.

Internal validity threats concern our choice of tools and apps; i.e., Ninka for license identification, Joa for software provenance analysis, CCFinderX for clone detection and the 857 studied apps. Different tools and apps could yield different results. However, our approach is generic and can be easily adapted for other tools and applied on other apps easily.

Conclusion validity threats concern the relation between the treatment and the outcome. We used a set of 857 mobile apps to evaluate LIT. A different corpus could give different results. We share all the data of our study at: `http://swat.polymtl.ca/data/SANER16/AndroidAppsDataONF-DroidJanv2015.7z`. Further studies with different sets of mobile apps (including close source apps) from different markets are required to verify our results and make our findings more general.

External validity threats concern the possibility to generalise our results. Although we only conducted our evaluation of LIT on 857 mobile apps from the F-droid market, our finding on the precision of CCFinderX is consistent with previous studies (*e.g.*, [31]).

3.6 Chapter Summary

To conduct any study on licenses, *e.g.*, license evolution, licensing usages, it is primordial to be able to identify licenses informations accurately. In this chapter we have proposed LIT, a license identification technique that can detect the license of libraries used by a mobile app. Results show that LIT also outperforms the state-of-the-art approach Ninka when detecting license informations from source code files. In the following chapter, we will leverage LIT to examine license evolution, license violations and the evolution of these violations overtime.

CHAPTER 4 LICENSE EVOLUTION

Licenses as source code informations are subject to changes. Developers can change the license of their software for multiple reasons, *e.g.*, they could update the license of a component because the license of a dependent component was changed by the component owner. Studying license changes is important to understand developers practice regarding license usages, *e.g.*, whether they follow specific patterns in licensing updates. In the previous chapter, while evaluating LIT, we noticed license changes in many files across versions. In this chapter we study license usages and evolutions in order to identify the most used licenses in the Android Ecosystem, both at file level and project level, across releases. We address the following two research questions:

RQ1: *What are the most common licenses used in open source mobile apps?*

This research question aims to identify licenses that are frequently used by developers of Android apps. The results of this research question will provide insights about the preferences of mobile apps developers among the more than 70 open source licenses that are available.

RQ2: *How do mobile apps licenses evolve overtime?* In this research question, we analyse the evolution of licenses overtime both at project and file levels, in order to understand the main patterns of licenses evolution in the Android Ecosystem.

Chapter Overview

Section 4.1 describes the design of our case study. Section 4.2 describes and discusses the results of our two research questions. Section 4.3 discusses the results of our study in the context of previous work. Section 4.4 discloses the threats to validity of our study. Section 4.5 summarizes this chapter.

4.1 Study Design

This section presents the design of our case study, which aims to address the following two research questions:

RQ1: What are the most common licenses used in open source mobile apps?

RQ2: How do mobile apps licenses evolve overtime?

4.1.1 Data Collection

We apply LIT to the 857 mobile apps downloaded from the Android Market F-Droid, to identify their licenses and the licenses of each of their source files. Next, we build four databases containing respectively, Android apps licenses, source files licenses, dependencies (*i.e.*, libraries) licenses and pairs of cloned files found with different licenses.

4.1.2 Data Processing

We analyse the evolution of licenses overtime both at project and file levels, in order to understand the main patterns of licenses evolution in the Android Ecosystem. For each file in each app, we track the evolution history of the file and build a genealogy. We identify files across the releases using their absolute paths. To handled cases of renaming, we apply clone detection to track files with similar contents that were renamed. Next, using licenses information collected using LIT, we map licenses to the different versions of each file and compute all licenses evolution patterns. Finally, we build state transition models capturing the license evolution patterns at file and project levels, respectively. For each transition in our state-transition diagrams, we compute the probability of the transition following Equation 4.1. To focus our interest, we narrow the data analyzed to only entities that experienced at least one change in their life-cycle. Thus, to calculate the probability of a transition from License A to License B, we calculate the occurrence of License A → License B divided by the occurrence of License A in our reduced data set.

$$\mathbf{OccurrenceOf(A{\rightarrow}B)} \Big/ \mathbf{OccurrenceOf(A)} \tag{4.1}$$

4.2 Results

This section presents and discusses the results of our two research questions.

RQ1: What are the most common licenses used in open source mobile apps?

When considering only the information provided by F-Droid, **we obtain that 35% of apps are licensed under GPLv3, 24% under Apachev2 and 12% under MIT License.** Figure 4.1 presents the licenses distribution of the final releases of all the apps in our data sets.

When considering all the released versions of each app from our data set, the picture is a

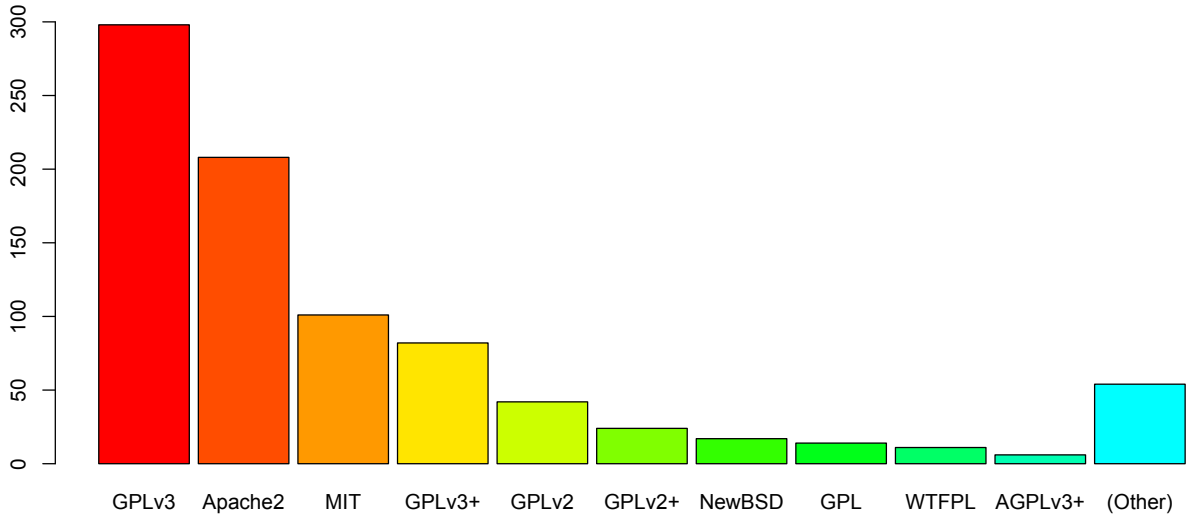

Figure 4.1 Projects licenses when considering only the latest release of each app

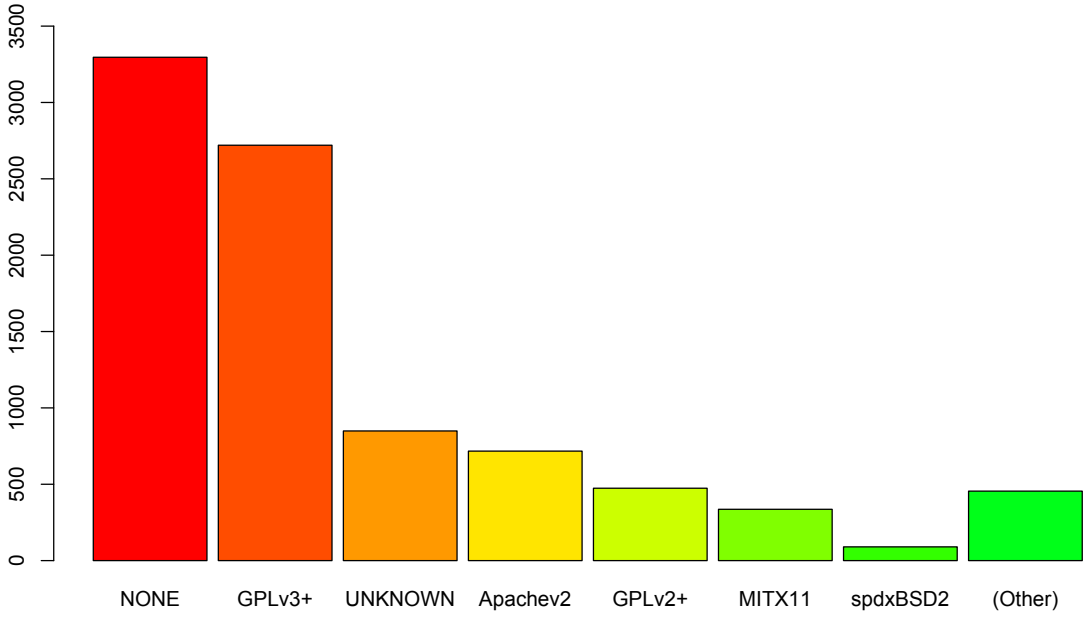

Figure 4.2 Projects licenses when considering all the releases of the apps

bit different. **We obtain that 37% of releases are licensed under GPLv3, 8% under Apachev2 and 4% under MIT License.** This difference is due to the fact that many apps change their licenses overtime. Figure 4.2 presents the distribution of licenses for all the releases of all the apps in our data set. We can observe that more than 3,250 apps releases out of 8,938 apps releases are unlicensed or their license information is not declared in any of our analysed files (described in Section 3.1.1).

At the file level, GPL and Apache are still the most used license; representing 47% and 12% of files respectively. However, there are more files licensed under the BSD license than the MIT license. Table 4.1 summarizes results obtained at the file level. Our set of apps contained in total 1,429,330 source code files (1,168,899 written in Java; 81,378 written in C; and 138,396 C++ files). In Table 4.2 the keywords “SeeFile", “UNKNOWN" and “ERROR" are used by Ninka and LIT (which is built on top of Ninka) to indicate that the analyzed file points to another file that may contain the license, that the found license is not recognized, and that Ninka (and LIT) encountered an error.

RQ2: How do mobile apps licenses evolve overtime?

We now report about licenses' evolution in Android applications and identify the most common patterns of licenses change.

4.2.0.1 License changes at the file level

Among the 857 apps from our data set, 128 apps experienced a license change. These 128 apps contain 2,062 Java files(0,1% of the total number of java files in our data set) that experienced at least a license change during their app's life-cycle. Figure 4.3 summarizes the transition patterns found in our data set and their probabilities. As one can see, the probability for a file to stay under the same license is very high for almost all the licenses (0.9 in average). We classify our patterns into two categories: general patterns (summarized in Table 4.3), which contains patterns that include no license or unknown license; and specific patterns (summarized in Table 4.4), which contains only patterns where we have changes between two known licenses. In the following, we discuss each of these categories in more details.

General patterns: files in this category generally start with no License (NONE) in their first release. This finding is consistent with observations made by Vendome et al. [7] that developers tend to delay their decision about license selection. Regarding transitions from a known license to NONE or UNKNOWN license,

Table 4.1 Licenses detected at file level

License	**Version**	**# Occurrence**	**%**
GPL	noVersion	32	**45,68**
	v2	37 388	
	v2+	395 198	
	v3	19 125	
	v3+	201 171	
Apache	Apache	2	**12,39**
	v1.0	52	
	v1.1	6	
	v2	184 765	
LGPL	v2	84	1,13
	v2.1	2 962	
	v2.1+	7 634	
	v2+	95	
	v3	2 097	
	v3+	3 284	
BSD	3NoWaranty	2 279	1,07
	BSD3	6 759	
	BSD4	168	
	spdxBSD2	2 281	
	spdxBSD3	3 655	
	spdxBSD4	130	
MIT	oldwithoutSelland	5	0,87
	oldwithoutSelland-NoDocumentationRequi	147	
	MITVariant	1	
	MITX11	11 965	
	X11BSDvar	4	
	X11noNotice	343	
PublicDomain		2 957	0,21
artifex		136	0,01
BeerWareVer42		6	0
CDDLorGPLv2		475	0,03
CPLv1		20	0
DoWTFYWv2		48	0
EPLv1		13	0
FreeType		1 714	0,12
MPLv1_1		4	0
ZLIBref		137	0,01
SunSimpleLic		555	0,04
SimpleLicence1		18	0
orGPLv2+orLGPLv2.1+		234	0,02
Total number of source code files analysed		**1 429 330**	

Table 4.2 Proportion of files with missing license

License	**# Occurrence**	**%**
None	392 314	**27,45**
SeeFile	8 053	0,56
UNKNOWN	139 484	9,76
ERROR	930	0,07

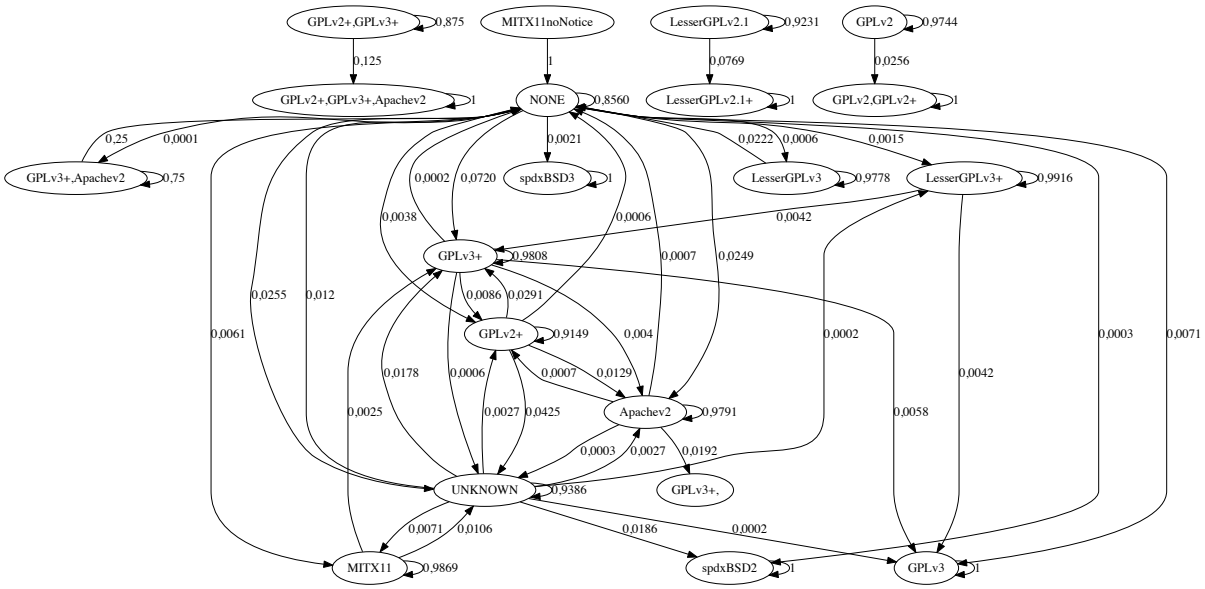

Figure 4.3 Evolution of licenses at the file level

Table 4.3 General patterns

Pattern	#Occurence
UNKNOWNorNONE → GPL3+	820
UNKNOWNorNONE → Apachev2	269
UNKNOWNorNONE → MITX11	92
UNKNOWNorNONE →spdxBSD2	79
GPL2+ → UNKNOWNorNONE	77
UNKNOWNorNONE → GPL3	75
UNKNOWNorNONE → GPL2+	50
NONE → spdxBSD3	22
UNKNOWNorNONE → LGPLv3+	17
MITX11 → UNKNOWN	17
GPL3+ → UNKNOWNorNONE	10
NONE → LGPLv3	6
Apachev2 → UNKNOWNorNONE	3
LGPLv3 → NONE	1
MITX11noNotice → NONE	1
GPLv3+,Apachev2 → NONE	1
NONE → GPLv3+,Apachev2	1

we could not find a plausible explanation, after manually examining all these patterns. It is unclear why some developers remove the license of their app. Below are some examples of general patterns found:

The app *PageTurner* started with no license. Later on developers licensed it to GPLv3+ and changed all its Apachev2 files to GPLv3+.
Wifi-Fixer also didn't have any license initially but later on was licensed under GPLv3+ and all its Apachev2 files were licensed under GPLv3+.
AndroidCaldavSyncAdapater was initially under no license, developers later on changed all its Apachev2 files to GPLv3+ and licensed the app under GPLv3+. All other source files in that app are under BSD3.
Specific patterns: We examined transitions between known licenses and noticed cases of license upgrade and downgrade. We discuss some of these cases in the coming paragraphs.

The app *keepassdroid* was initially released under the GPLv2 license. However it contained files licensed under Apachev2 and GPLv2 licenses which are incompatible. Later on, the project was changed to the dual license "GPLv2, GPLv2+" in order to solve its license violation.

The app *open-keychain* was initially licensed under Apachev2, then, developers changed all

Table 4.4 Specific patterns

pattern	#Occurrence
GPLv3+→GPLv2+	109
GPLv3+→GPLv3	73
Apachev2→GPLv3+	56
GPLv3+→Apachev2	50
GPLv2+→GPLv3+	52
GPLv2+→Apachev2	23
GPLv2→GPLv2,GPLv2+	5
MITX11→GPLv3+	4
Apachev2→GPLv2+	2
LGPLv2.1→LGPLv2.1+	2
LGPLv3+→GPLv3+	1
LGPLv3+→GPLv3	1
GPLv2+,GPLv3+→GPLv2+,GPLv3+,Apachev2	1

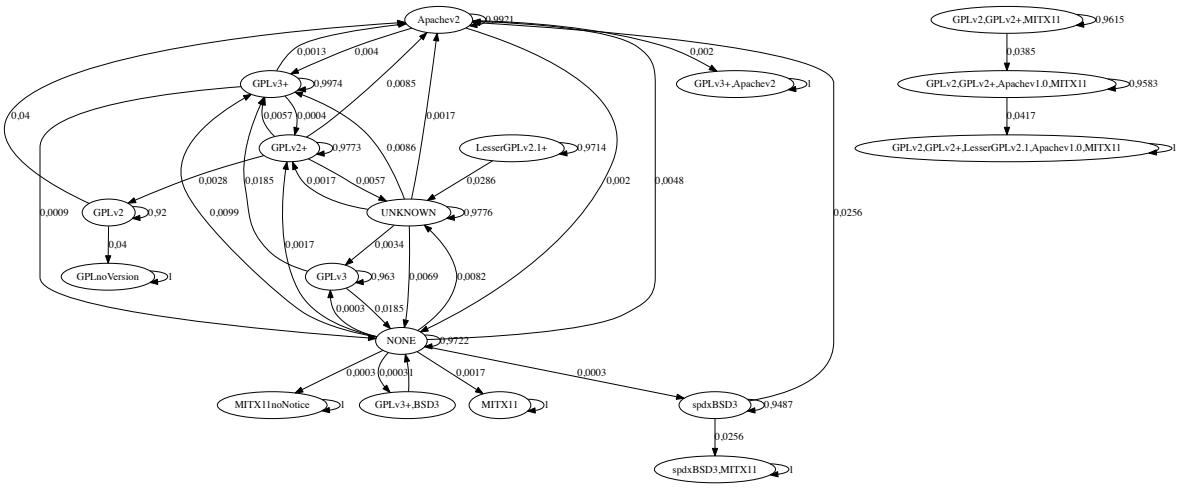

Figure 4.4 Evolution of licenses at the projects level

Apachev2 Java files to the GPLv3+ license and updated the license of the app to GPLv3+. The app *geopaparazzi* is now under GPLv3+. It started with no license, then, developers changed some Apachev2 files to the GPLv3+ license. The app also contain files under LesserGPLv2.1+ and GPLv3 licenses. The last 15 releases of the app (out of a total of 79 releases) are under GPLv3+.
The *connectbot* app started under the GPLv3+ license. It maintained this license through 11 releases. It contains files under BSD3, BSD, GPL2+, MITX11, GPLv3+ and apachev2 licenses. The license of *connectbot* was changed to Apachev2 in the last 22 releases (out of a total of 33 releases). During this transition, developers changed the license of all GPLv3+ files to Apachev2.
gnucash-android was first released under GPLv2+, files were under GPLv2+ and Apachev2 licenses, then, the license of the app was changed to Apachev2 and all GPLv2+ files were licensed under Apachev2.

4.2.0.2 Licenses changes at the project level

We noticed that the probability for a project to stay under the same license is very high (see Figure 4.4). Only 122 apps(14%) out of 857 apps changed their license at least once during their lifetime. Apps license changes are generally toward more permissive licenses (see Table 4.5)

4.3 Discussion

This section discusses some of the key findings of this chapter. By tacking licenses changes across releases, we observed that license changes are toward less restrictive licenses, this finding confirms the observation of Vendome et al. [20] that open source developers tend to make their projects more open for reuse. However, a deeper look at license changes at file level reveals a strange phenomenon; the licenses of all files are often changed at the same time, and usually after the project has experienced some license inconsistencies. This observation lead us to suspect potential licenses infringement issues. In fact, studies made by Vendome and al. [7] have shown that developers have a limited knowledge on licensing issues. In the following chapter we will investigate license violations in more details.

4.4 Threats to Validity

This section discusses the threats to validity of our study following the guidelines for case study research [30].

Table 4.5 License change patterns at project level

Patern	Occurence
General Patern	
UNKNOWNorNONE→GPLv3+	40
UNKNOWNorNONE→Apachev2	16
GPLv3+→UNKNOWNorNONE 7	7
UNKNOWNorNONE→GPLv2+	6
NONE→MITX11	6
UNKNOWNorNONE→GPLv3	4
GPLv2+→UNKNOWN	2
UNKNOWNorNONE→spdxBSD2	2
NONE→MITX11noNotice	2
NONE→spdxBSD3	2
NONE→GPLv3+,BSD3	1
Apachev2→NONE	1
GPLv3→NONE	1
GPLv3+,BSD3→NONE	1
LesserGPLv2.1→UNKNOWN	1
Specific Pattern	
GPLv3+→Apachev2	3
GPLv2+→Apachev2	3
Apachev2→GPLv3+	2
GPLv2+→GPLv3+	2
Apachev2→GPLv3+,Apachev2	1
GPLv3→GPLv3+	1
GPLv3+→GPLv2+	1
GPLv2+→GPLv2	1
GPLv2→Apachev2	1
spdxBSD3→spdxBSD3,MITX11	1
spdxBSD3→Apachev2	1
GPLv2→GPLnoVersion	1
GPLv2,GPLv2+,MITX11→GPLv2,GPLv2+,Apachev1.0,MITX11	1
GPLv2,GPLv2+,Apachev1.0,MITX11 →GPLv2,GPLv2+,LesserGPLv2.1,Apachev1.0,MITX11	1

Construct validity threats concern the relation between theory and observation. In this study, the construct validity threats are mainly due to measurement errors. The precision of our detection approach LIT is likely to affect our findings. However, as shown in Chapter 3, LIT outperforms state-of-the-art approaches like Ninka.

Internal validity do not affect this particular study, being an exploratory study [30]. Thus, we cannot claim causation.

Conclusion validity threats concern the relation between the treatment and the outcome. To track file's license changes, we relied only on the absolute path of the file. However, we required that the files in the subsequent releases share a significantly large similarity (measured using CCFinderX).

External validity threats concern the possibility to generalise our results. Although we only conduct our case study with 857 mobile apps from the F-droid market, most of our findings are consistent with previous studies (*e.g.*, [7]).

We share our data at:
`http://swat.polymtl.ca/data/SANER16/AndroidAppsDataONF-DroidJanv2015.7z`.
Further studies with different sets of mobile apps (including close source apps) from different markets are required to verify our results and make our findings more general.

4.5 Chapter Summary

In this chapter we conduct a study on licensing evolution, our findings corroborate those obtained by Vendome et al. [20] on open-source projects. In the following chapter we will study license violations and their evolution.

CHAPTER 5 LICENSE VIOLATIONS DETECTION

With the very high speed of apps development, developers are increasingly inclined to copy code from pre-existing open source applications. Open source application are made available under specific license terms that developers should follow. A failure to satisfy license terms can lead to serious penalties. With the large number of licenses available (more than 70 OSS license and their different versions), it is becoming more and more complex to decipher all the different rights and obligations of licenses; making it difficult for developers to understand all the legal constrains of their software. In this chapter we conduct a study on license violation using LIT and try to answer the following research question:

RQ1: *Can we identify license violations using LIT?*

In Chapter 3, we propose LIT, our three steps technique for license identification. Using LIT, we identified 85% of applications with license inconsistencies. However, not all of these inconsistencies are likely to be license violations. In fact, since it is possible to release code under a dual license (*e.g.*, Apachev2 and GPLv2), two apps reusing a code licensed under a dual license could simply chose to pick different licenses (*e.g.*, the first app picking Apachev2 and the second app GPLv2). Our clone analysis (with LIT) would report this as a case of license inconsistency. However, there is no license violation in this case. In this research question, we aim to identify clear cases of licenses violations in the Android free software apps ecosystem.

RQ2: *How do mobile apps licenses violations evolve overtime?*

This research question aims to track license violations across the apps' releases in order to examine if and–how developers address these violations.

Chapter Overview

Section 5.1 describes the design of our case study. Section 5.2 describes and discusses the results of our two research questions. Section 5.3 discusses the results of our study contrasting it with previous works. Section 5.4 discloses the threats to validity of our study. Section 5.5 summarizes this chapter.

5.1 Study Design

This section presents the design of our case study, which aims to address the following two research questions:

RQ1: Can we identify license violations using LIT?

RQ2: How do mobile apps licenses violations evolve overtime?

5.1.1 Data Collection

The data set used in this chapter is composed of the same 857 mobile apps downloaded from the Android Market F-Droid. We studied the documentations about open-source licenses and identified 24 types of license incompatibilities (*i.e.*, licenses that should not coexist in a same project) that we summarize in Table 5.1 and Table 5.2.

5.1.2 Data Processing

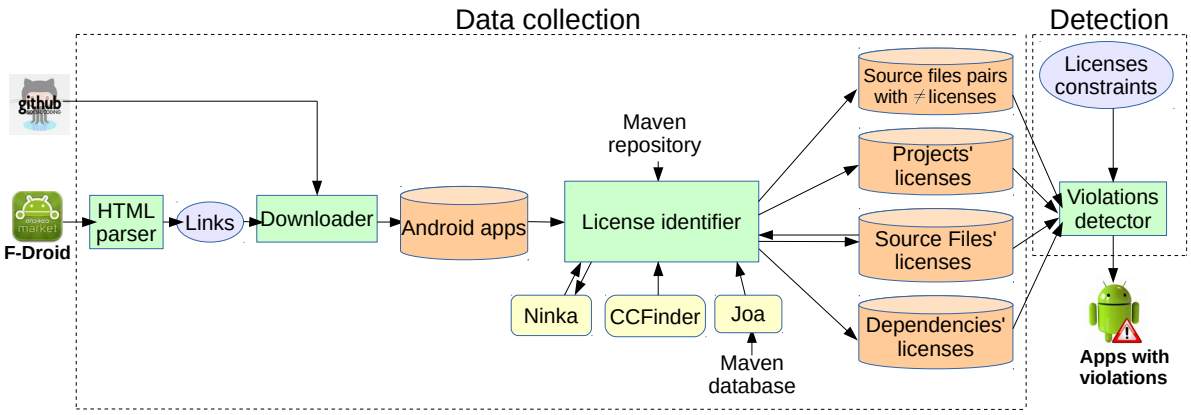

Figure 5.1 Overview of our approach to identify applications with license violation in an Android Market

Figure 5.1 shows an overview of our approach to identify license violations. First, we used our license identification approach LIT, described in Chapter 3 to extract licenses information for all the 857 apps. Next, we build four databases containing respectively, Android apps licenses, source files licenses, dependencies (*i.e.*, libraries) licenses and pairs of cloned files found with different licenses.

To track licenses violations, we implemented a script that checks for licenses inconsistencies using information from Table 5.1 and Table 5.2. Then, we ran this script against our three databases to obtain information about licenses violations. The source code of our script is available online at `http://swat.polymtl.ca/data/SANER16/Curseur1.sql`.

To track license violations, we adopt a conservative approach. We consider that an app violates some license terms only when there are inconsistencies between the declared license

Table 5.1 Licensing Rules [2]

Library License	**Project include**	**Combination must be under**
GPLv2	LGPLv2.1	GPLv2 only
GPLv2	LGPLv2.1+	GPLv2 only
GPLv2+	LGPLv2.1	GPLv2+
GPLv2+	LGPLv2.1+	GPLv2+
GPLv2+	LGPLv3+	GPLv3
GPLv3	GPLv2+	GPLv3
GPLv3	LGPLv2.1	GPLv3
GPLv3	LGPLv2.1+	GPLv3
GPLv3	LGPLv3+	GPLv3
LGPLv3	GPLv2+	GPLv3

Table 5.2 License incompatibilities

Incompatible license	Reason for the incompatibility
Apachev2 vs. GPLv2	Copyleft licenses are “reciprocal”, “share-alike”, or “viral” license each of them says, “If you include code under this license in a larger program, the larger program must be under this license too.”[33]
Apachev2 vs. GPL (except GPLv3)	restrictions of copyleft licenses; each of them says, “If you include code under this license ... the larger program must be under this license too.” “Apachev2 software can ... be included in GPLv3 projects"
Apachev2 vs. GPLv3	Copyleft licenses But one way is permitted; project under GPL3 source may have Apachev2[33]
Apachev2 vs. LGPL 2, 2.1, 3	“The LGPL is ineligible primarily due to the restrictions it places on larger works, violating the third license criterion. Therefore, LGPL-licensed works must not be included in Apache products”[34]
GPLv2 vs. GPLv3+	restrictions of copyleft licenses; each of them says, “If you include code under this license ... the larger program must be under this license too.”
GPLv2 vs. GPLv3	restrictions of copyleft licenses; each of them says, “If you include code under this license ... the larger program must be under this license too.”
LGPLv3+ vs. GPLv2	[35]
GPL vs Apachev1.0	“a lax, permissive non-copyleft free software license with an advertising clause. This creates practical problems like those of the original BSD license, including incompatibility with the GNU GPL” [35]
GPL vs Apachev1.1	“a permissive non-copyleft free software license. It has a few requirements that render it incompatible with the GNU GPL, such as strong prohibitions on the use of Apache-related names” [35]
GPL vs CDDL	http://www.gnu.org/licenses/license-list.en.html “It has a weak per-file copyleft. This means a module covered by the GPL and a module covered by the CDDL can not legally be linked together.” [36]
GPL vs CPLv1	“its weak copyleft and choice of law clause make it incompatible with the GPL” [35]
GPL vs EPLv1	“its weak copyleft and choice of law clause make it incompatible with the GPL” [35]
GPL vs MPLv1.1	“a free software license which is not a strong copyleft; unlike the X11 license, it has some complex restrictions that make it incompatible with the GNU GPL” [35]
GPL vs BSD4	“a lax, permissive non-copyleft free software license with a serious flaw: the “obnoxious BSD advertising clause”.” [35]

of the app and the licenses of its files, among its files and–or the license of its dependencies.

5.2 Results

This section presents and discusses the results of our two research questions.

RQ1: Can we identify license violations using LIT?

Table 5.3 shows the proportion of license violations found in our data set. Among the 857 studied apps, we found 17 apps (totaling 229 releases) that clearly violate the terms of some open-source licenses. Apps from eight categories are involved in license infringements. Most of the applications that violate a license term are under “development" category.

RQ2: How do mobile apps licenses violations evolve overtime?

From the 17 apps that contain a license violation, only 10 apps solved the license violations after some releases. The remaining 7 projects still had a license violation at the time of this study. To solve license violations, developers either changed the licenses of some of the app’s files or removed the contentious files from the apps. For the 10 apps that solved licenses violations. We noticed that it took them on average 19 releases to correct the violation. This means that 19 releases of some apps were distributed into the Android Market with license violations, which is quite troubling.

5.3 Discussion

This section discusses some of the key aspects of our study. We define a methodology for the detection of license violations within open source software systems. Our proposed methodology takes into consideration different artifacts of an open source software; its source code, its libraries and its declared licenses. Using our methodology we were able to detect license violations within a set of applications from an Android Market. We observed that licenses violations persist through multiple releases of the apps before they are eventually resolved. This finding suggests that developers seem to lack proper knowledge about licenses management.

Table 5.3 Projects with license violation

Category	Name	License	Nb releases	releases with Violation	Kind of violation
Security	afwall	UNKNOWN	31	5 - 19	GPLv2 vs Apachev2 GPLv2 vs GPLv3+
PhoneAndSMS	batphone	GPLv3+	37	16 - 23	GPLv2 vs Apachev2 GPLv2 vs GPLv3+
Navigation	MozStumbler	UNKNOWN	86	58 - 86	Apachev2 vs GPLnoVersion
Development	Vimtouch	Apachev2	22	20 - 22	(proj)Apachev2 vs GPLv3+(source)
	Travis-Jr	Apachev2	2	1 - 2	(proj) Apachev2 vs LGPL (lib)
	servdroid	Apachev2	5	1 - 5	(proj) Apachev2 vs LGPL (lib)
	andlytics	Apachev2	26	24 - 26	(proj) Apachev2 vs LGPL (lib)
	BalandruinoAndroidApp	GPLv2	10	1 - 4	(proj) GPLv2 vs Apachev2(source)
Games	ppsspp	UNKNOWN	24	1 - 13	Apachev1.0 vs GPLv3+ Apachev1.0 vs GPLv2+
	dolphin	NONE	9	1,3,5 - 7 1 - 7	Apachev1.0 vs GPLv3+ Apachev1.0 vs GPLv2+
Internet	open-keychain	Apachev2	30	1 - 4	(proj)Apachev2 vs GPLv3+(source)
	frostwire-android	GPLv3+	64	1 - 32 1 - 24 13	GPLv2 vs Apachev2 GPLv2 vs GPLv3+ (source)GPLv2 vs Apachev2(lib) GPLv2 vs LGPLv3+
	android	GPLv2	38	15 - 31	(proj)GPLv2 vs Apachev2(source)
Office	Keepassdroid	GPLv2, GPLv2+, LGPLv2.1, Apachev1.0, MITX11	144	1 - 90 36 - 90	GPLv2 vs GPLv3+ GPLv2 vs Apachev2
System	host-editor-android	Apachev2	6	2	(proj) Apachev2 vs LGPL (lib)
	gogodroid	GPLv2	7	1 - 7	Apachev1.1 vs GPLv2+ (proj) GPLv2 vs Apache
	920-Text-Editor	GPLv3+	2	1 - 2	GPLv2 vs GPLv3+ GPLv2 vs Apachev2

5.4 Threats to Validity

This section discusses the threats to validity of our study following the guidelines for case study research [30].

Construct validity threats concern the relation between theory and observation. In this study, the construct validity threats are mainly due to measurement errors. The precision of our detection approach LIT is likely to affect our findings. However, as shown in Chapter 3, LIT outperforms state-of-the-art approaches like Ninka.

Internal validity do not affect this particular study, being an exploratory study [30]. We do not claim causation. *Conclusion validity* threats concern the relation between the treatment and the outcome. License terms could be interpreted in deferent ways. There are ongoing discussions among developers about the interpretations of the legal terms of some open-source licenses. In our study, we relied only on the information explicitly written in the official websites of the licenses. This conservative approach may limit us from detecting more incompatible licenses. However, our proposed approach to detect license violations can be easily extended to include more license rules.

External validity threats concern the possibility to generalise our results. Although we only conduct our case study with 857 mobile apps from the F-droid market, most of our findings are consistent with previous studies (*e.g.*, [7]). We share our data and scripts at: `http://swat.polymtl.ca/data/SANER16/AndroidAppsDataONF-DroidJanv2015.7z`. Further studies with different sets of mobile apps (including close source apps) from different markets are required to verify our results and make our findings more general.

5.5 Chapter Summary

In this chapter we propose a methodology for the detection of license violations in open source software systems. This methodology is based on LIT, our three steps approach for license identification. Results shows that our methodology can help detect license violations within a set of apps from an Android Market. In the following chapter we will summarize our thesis, discuss the limitations of our work and outline some avenues for future work.

CHAPTER 6 CONCLUSION

This chapter summarizes the findings of this thesis, discusses the limitations of our proposed approaches, and outlines some avenues for future work.

6.1 Summary

With the very high speed of apps development; on average 2,371 new apps are published in Google Play every day; developers are increasingly inclined to reuse code from other open source systems. Those open source systems are made available under certain licenses. So the reuse of existing code from different systems can produce a system with heterogeneous components with different licenses. Although licenses clearly describe the legal constraints of individual components, *i.e.*, the various rights/obligations of each license, the large number of existing licenses (more than 70 OSS licenses) and their different versions makes license violations very likely. Many license violation issues are raised in courts and lead to big fines. Researchers and practitioners have been looking for solutions, to detect and prevent license violations early before the release of software products to the public. Some studies focused on the detection; They tried to detect license violations by analyzing the source code and its declared license or the byte code and its declared license. Some studies went on the preventive way, proposing their own database with the validation that it does not contain any violation. Open source projects are generally a mixture of source code and binaries, which makes it impossible to use one of the solutions proposed in literature, thus, in this thesis, we proposed a more complete solution for identifying license violation.

Our proposed methodology is based on LIT; a three steps approach for license identification. First, we identify the licenses of source files and the project license using Ninka; a state-of-the-art tool for license identification; Second we identify the license of libraries used by the projects with the help of the Joa tool and finally we use clone detection techniques to identify license inconsistencies and to improve Ninka. Having license informations, we could detect license violations using a set of rules derived from the rights and obligations of 8 open-source licenses and their versions. We applied our methodology on a set of 857 Android applications from an Android open source Market, *i.e.*, F-Droid, and their 8,938 releases, downloaded from Github. Results shows that LIT performs better than Ninka when identifying license informations; we observed a gain of 16% of the information missed by Ninka. LIT could also identify the license information of more than 80,000 libraries used by the studied Android apps. We examined licenses evolution and observed that developers

tend to change the license of their app toward less restrictive licenses. We found 17 apps out of the 857 apps with clear violations of license terms. We also observed that licenses violations persist through multiple releases of the apps before they are eventually resolved.

6.2 Limitations of the proposed approaches

Our methodology rely on many pre-existing tools and its precision is directly impacted by the precision of those tools. Although we have selected state of the art tools that are known to achieve the best performances to date, if better tools are developed, our methodology can be easily adapted for these tools. License terms could be interpreted in different ways. There are ongoing discussions among developers about the interpretations of the legal terms of some open-source licenses. In our study, we relied only on the information explicitly written in the official Websites of the licenses. This conservative approach may limit us from detecting more incompatible licenses. However, our proposed approach to detect license violations can be easily extended to include more license rules. Although we only conduct our case study with 857 mobile apps from the F-droid market, most of our findings are consistent with previous studies (*e.g.*, [7]). We share our data and scripts at: `http://swat.polymtl.ca/data/SANER16/AndroidAppsDataONF-DroidJanv2015.7z`.
Further studies with different sets of mobile apps (including close source apps) from different markets are required to verify our results and make our findings more general.

6.3 Future work

This thesis reports the results of a large empirical study aimed at understanding licenses usages and violations in the Android Ecosystem. The results of the study suggests that developers of mobile apps frequently fail to comply with the terms of licenses. Even more problematic is the fact that licenses violations persist through multiple releases of the apps before they are eventually resolved. Developers seem to lack proper knowledge about licenses management or it may be that the lack of consistency and standardization in license declarations fosters confusion among developers (as suggested by Vendome et al. [7]). Indeed, the fact that some licenses are contained in source code heading comments, while others are put in separate license files or README documents may cause developers to miss some license information. To help address this issue, we advocate for the development of tools that can assist developers in the management of licenses throughout the lifecycle of their apps. In the future, we plan to investigate further our fining that app developers tend to copy the code of apps from different domains, in order to understand the root causes of this behaviour.

REFEFENCES

[1] D. M. German, Y. Menabe, and K. Inoue, "A sentence-matching method for automatic license identification of source code files," in *Proceedings of the IEEE/ACM international conference on Automated software engineering.* ACM, 2010, pp. 437–446.

[2] "GNU General Public License Legal," http://www.gnu.org/licenses/gpl-faq.html#AllCompatibility, 2015, online; accessed September 09th, 2015.

[3] D. M. German and A. E. Hassan, "License integration patterns: Addressing license mismatches in component-based development," in *ICSE -09: Proceedings of the 31st International Conference on Software Engineering.* IEEE, 2009, pp. 188–198.

[4] "GNU General Public License," http://www.gnu.org/licenses/gpl.html, 2015, online; accessed October 05th, 2015.

[5] "VMware lawsuit. GPL violation case," http://www.infoworld.com/article/2893695/open-source-software/vmware-heading-to-court-over-gpl-violations.html, 2015, online; accessed April 15th, 2015.

[6] J. Koetsier. (2013, August) 700k of the 1.2m apps available for iphone, android, and windows are zombies: Last accessed: November 14, 2015. [Online]. Available: http://venturebeat.com/2013/08/26/700k-of-the-1-2m-apps-available-for-iphone-android-and-windows-are-zombies/

[7] C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. German, and D. Poshyvanyk, "When and why developers adopt and change software licenses," in *Proceedings of the 31st IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015.

[8] I. J. Mojica, B. Adams, M. Nagappan, S. Dienst, T. Berger, and A. E. Hassan, "A large-scale empirical study on software reuse in mobile apps," *IEEE Software*, vol. 31, no. 2, pp. 78–86, Mar 2014.

[9] R. Gobeille, "The fossology project," in *Proceedings of the 2008 international working conference on Mining software repositories.* ACM, 2008, pp. 47–50.

[10] N. Krawetz, "Symbolic alignment matrix," 2008, http://www.fossology.org/projects/fo-,ssology/wiki/Symbolic_Alignment_Matrix.

[11] M. Di Penta, D. M. German, and G. Antoniol, "Identifying licensing of jar archives using a code-search approach," in *Mining Software Repositories (MSR), 2010 7th IEEE*

Working Conference on. IEEE, 2010, pp. 151–160.

[12] J. Davies, D. M. German, M. W. G. Godfrey, and A. Hindle, "Software bertillonage: Finding the provenance of software development artifacts," *Empirical Software Engineering*, vol. 18, no. 6, pp. 1195–1237, 2013.

[13] D. German and M. Di Penta, "A method for open source license compliance of java applications," *IEEE software*, no. 3, pp. 58–63, 2012.

[14] "CNET News: Open-source spat spurs software change," http://www.cnet.com/news/open-source-spat-spurs-software-change/, 2002, online; accessed October 13th, 2014.

[15] "Mysql. foss license exception," http://www.mysql.com/about/legal/licensing/foss-exception/, 2012, online; accessed january 17th, 2016.

[16] D. M. German and J. M. González-Barahona, "An empirical study of the reuse of software licensed under the gnu general public license," in *Open Source Ecosystems: Diverse Communities Interacting*. Springer, 2009, pp. 185–198.

[17] "Novell inc. faq: Licensing," http://www.mono-project.com/docs/faq/licensing/, 2016, online; accessed january 12th, 2016.

[18] "KDE. Qt issue: Licensing," https://www.kde.org/community/history/qtissue.php, online; accessed january 14th, 2016.

[19] M. Di Penta, D. M. German, Y.-G. Guéhéneuc, and G. Antoniol, "An exploratory study of the evolution of software licensing," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 2010, pp. 145–154.

[20] C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. German, and D. Poshyvanyk, "License usage and changes: A largescale study of java projects on github," in *23rd IEEE International Conference on Program Comprehension, ICPC, Florence, Italy*. IEEE, 2015, pp. 18–19.

[21] G. Bavota, A. Ciemniewska, I. Chulani, A. De Nigro, M. Di Penta, D. Galletti, R. Galoppini, T. F. Gordon, P. Kedziora, I. Lener *et al.*, "The market for open source: An intelligent virtual open source marketplace," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. IEEE, 2014, pp. 399–402.

[22] M. Sojer and J. Henkel, "Code reuse in open source software development: Quantitative evidence, drivers, and impediments," *Journal of the Association for Information Systems*, vol. 11, no. 12, pp. 868–901, 2010.

[23] M. Sojer, O. Alexy, S. Kleinknecht, and J. Henkel, "Understanding the drivers of unethical programming behavior: The inappropriate reuse of internet-accessible code," *Journal of Management Information Systems*, vol. 31, no. 3, pp. 287–325, 2014.

[24] "arstechnica: Oracle will seek a staggering $9.3 billion in 2nd trial against Google," http://arstechnica.com/tech-policy/2016/03/oracle-will-seek-a-staggering-9-3-billion-in-2nd-trial-against-google/, 2016, online; accessed March 30th, 2016.

[25] P. V. Singh and C. Phelps, "Networks, social influence, and the choice among competing innovations: Insights from open source software licenses," *Information Systems Research*, vol. 24, no. 3, pp. 539–560, 2012.

[26] D. M. German, M. Di Penta, and J. Davies, "Understanding and auditing the licensing of open source software distributions," in *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*. IEEE, 2010, pp. 84–93.

[27] A. Hemel, K. T. Kalleberg, R. Vermaas, and E. Dolstra, "Finding software license violations through binary code clone detection," in *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 2011, pp. 63–72.

[28] T. Kamiya, S. Kusumoto, and K. Inoue, "Ccfinder: a multilinguistic token-based code clone detection system for large scale source code," *Software Engineering, IEEE Transactions on*, vol. 28, no. 7, pp. 654–670, 2002.

[29] J. Svajlenko and C. K. Roy, "Evaluating modern clone detection tools," in *2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2014, pp. 321–330.

[30] R. K. Yin, *Case Study Research: Design and Methods - Third Edition*, 3rd ed. SAGE Publications, 2002.

[31] S. Dang and S. A. Wani, "Performance evaluation of clone detection tools," *INTERNATIONAL JOURNAL OF SCIENCE AND RESEARCH (IJSR)*, pp. 1903–1906, 2015.

[32] L. Barbour, F. Khomh, and Y. Zou, "Late propagation in software clones," in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, Sept 2011, pp. 273–282.

[33] "Apache License vs GPLv3," http://www.apache.org/licenses/GPL-compatibility.html, 2015, online; accessed September 17th, 2015.

[34] "Apache License Legal," http://www.apache.org/legal/resolved.html, 2015, online; accessed September 17th, 2015.

[35] "GNU General Public License Legal," http://www.gnu.org/licenses/license-list.en.html, 2015, online; accessed September 09th, 2015.

[36] "CDDL License. 10 questions-answered," http://www.whitesourcesoftware.com/top-10-cddl-license-questions-answered/, 2015, online; accessed September 29th, 2015.