UNIVERSITÉ DE MONTRÉAL

# SOURCE CODE AND LICENSE STATEMENT CO-EVOLUTION

FERDAOUS BOUGHANMI
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 ET GÉNIE LOGICIEL)
NOVEMBRE 2012

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

SOURCE CODE AND LICENSE STATEMENT CO-EVOLUTION

présenté
par : Mme BOUGHANMI Ferdaous
en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées
a été dûment accepté par le jury constitué de :

M. ADAMS Bram, Doct., président.
M. ANTONIOL Giuliano, Ph.D., membre et directeur de recherche.
M. GUÉHÉNEUC Yann-Gaël, Ph.D., membre et directeur de recherche.
M. C. DESMARAIS Michel, Ph.D., membre.

# Résumé

Les logiciels libres reposent largement sur la réutilisation de composants logiciels disponibles sous une variété de licenses (*e.g.*, Apache, BSD, GPL, ou LGPL). Différentes licenses imposent des limitations et des conditions différentes sur la réutilisation d'un programme et sa redistribution ce qui rend difficile la compréhension des contraintes juridiques imposées au système final. La license d'un fichier est spécifié par une déclaration de license. Les déclarations de licence sont des snippets de texte insérées en haut du code source ou de tout autre fichier qui spécifie la license sous laquelle le fichier peut être réutilisé, ainsi que les contributeurs qui possédent des droits d'auteur sur le fichier. Les déclarations de license ne sont pas un concept statique, car les projets peuvent mettre à jour leur licenses (version ou type) ou ajouter des contributeurs. Comme ces changements peuvent avoir un impact majeur sur un système en terme de sa distribution et utilisation, il est important de comprendre quand ils se produisent au cours du développement relativement à l'évolution du source code (le changement des licenses peut être pendant d'importantes modifications ou indépendamment de l'évolution des modifications du système), combien de fois ils se produisent (rare vs.récurants), et qui les effectue (experts vs.développeurs réguliers). Nous proposons donc, un méta-modèle pour effectuer des analyses qui permettent la detection des problèmes de license et présente une source d'information structurée qui peut être utilisé dans les études reliées aux licenses. Ensuite, nous présentons une étude sur la co-évolution des déclarations de license et le code source dans sept systèmes OSS : JFreeChart, Jitsi, PHP, Rhino, Tomcat, XalanJ et XercesJ. Notre étude montre que ce n'est que dans quelques cas, dans PHP, que l'évolution des déclarations de licenses et celle du logiciel sont soigneusement planifiés et gérés ensemble juste avant les versions majeures. Dans tous les systèmes, les développeurs qui effectuent plus de changement de code source, sont aussi les plus actifs mainteneurs de license. Notre travail permet de comprendre quand les déclarations de license sont changées et permet d'identifier les développeurs qui effectuent ces changements. Notre travail est un travail préliminaire afin de mieux contrôler l'impact de ces changements sur le système, *i.e.*, éviter l'introduction des inconsistences en proposant une méthodologie pour la gestion des changement de licenses.

# Abstract

Open-source software (OSS) systems heavily rely on the reuse of software components made available under a variety of software licenses (*e.g.*, Apache, BSD, GPL, or LGPL). Different licenses impose different limitations and conditions on program reuse and redistribution thus making it difficult to understand the legal constraints for the final system. The file license is specified using a license statement. License statements are snippets of text near the top of a source code or other files that specify the software license under which the file can be used as well as which contributors own copyrights over the file. Such license statements are not static because projects might update the licenses (version or type) or add contributors. Such changes can have a major impact on a software system, so it is important to understand when they happen during development (with major souce code changes vs.independently), how often they happen (rare vs.recurring), and who performs them (experts vs.regular developers). In addition, a meta-model could help analyse to detect license issues in studies related to licenses. In this thesis, we propose a meta-model based on previous work and on information gathered from license text. Then, we perform a study on the co-evolution of license statements and source code in seven OSS systems: JFreeChart, Jitsi, PHP, Rhino, Tomcat, XalanJ, and XercesJ. Only in a few cases in PHP, license statement and software evolution are carefully planned and managed together just before major releases. In all systems, the developers performing most of the commits, are also the most active license maintainers. Thus, we are able to understand when license statements are changed and we identified the developers that perform these changes. Our finding are the result of preliminary work to permit better control on license change impact on the system, *i.e.*, avoid the risk of introducing inconsistencies by proposing a methodology to manage the process of license changes.

# Contents

remove italic

Need

Verifying License Evolution/changes

# Chapter 1

# INTRODUCTION

## 1.1 Context

A software license governs the legal use and redistribution of a system and its components by dictating what can and cannot be done with the system, *e.g.*, if the users can access the artifacts [1], if they can modify or enhance it and, more importantly, if they are allowed to re-distribute the original source code as well as any improvements. In open source software (OSS) systems, license information is included in each source code file as a textual license statement or as a notice file for the whole system or for each component. Such a statement also includes copyright information: the names of contributors to the source code file and the copyright owner. The copyright owner of a software system has exclusive rights to make copies of the system, prepare derivative works based on it, and distribute copies. She uses a license to grant permission to the licensees to use and exploit her intellectual property by granting rights. Each grant is given provided a set of conditions are satisfied (German et Hassan (2009)).

The availability of OSS systems mainly because of the advent of Free/Open Source Software (FOSS) and also proprietary systems with open APIs and the need for more rapid product development encourage creating systems through integration of pre-existing components. In fact, developers tend to assemble different components instead of writing all the system by themself. This practice leads to systems composed of heterogenously licensed components, *i.e.*, package, libraries, framework... where each component could have a different license and the whole system could be licensed differently from its components.

Yet, it could introduce another type of problem due to the various rights/obligations of each license, the large number of licenses, *i.e.*, more than 70 OSS licenses exist today, and their different versions. Thus, it becomes difficult to follow all the rights/obligation of each license and their combination thereof, which increases the

1. In this thesis, we are interested to source code

complexity to manage licenses. In addition, the kind of reuse could even add additional problems, because the reuse of existing components could lead to two types of works, *i.e.*, derivative works or collective works. A derivative work is a work based upon one or more preexisting works in which a work may be recast, transformed, or adapted[2], in contrast a collective work is an assembled independent work that could be distributed independently. In general, the case of the creation of derivative work poses more constraints in some case. Thus, it is important to know if the created work is derivative by determining the connectors used to connect each component, *e.g.*, when we connect to GPL component using by instanciating a class, is considered derivative work, thus it is required that the final work must be licensed under GPL. In fact, one of the major challenge is the reuse of software licensed under reciprocal licenses *e.g.*, GPL license, to create derivative work because requires that the whole work will be licensed under the same version of the reciprocal license.

The license of an OSS system could evolve like any other software artifact. Such, license evolution is driven by many factors, *e.g.*, to make the license more restrictive by the addition of new terms, or to allow derivative works by adding exceptions. In fact, a license can either be changed pervasively throughout a software system (*e.g.*, the switch GPLv2 to GPLv3), or only locally (*e.g.*, contributor name added to one file). Furthermore, a license statement evolution can be coarse-grained (switch to a different license), fine-grained (copyright year updated) or anything in between (clause added or removed) (Di Penta *et al.* (2010)).

This evolution introduces a risk of license terms violation. Many software systems are composed of different libraries and components; if one component changes its license, then it might no longer be possible to use it because of incompatibility. We provide now some examples that show the possible consequences of license evolution and how complicated it is to reuse GPL-licensed programs. The first example is the IPFilter[3] that is a component used by the OpenBSD[4] system, but the author of IPFilter added new terms to its license, which were not compatible with the existing license of OpenBSD. Thus, openBSD replaced IPFilter by its own OpenBSD-based implementation.

A second example is the "Java Classpath exception": the Java JDK was dis-

2. add ref
3. add ref to ipfilter
4. write something about openBSD

tributed until recently under the Common Development and Distribution License (CDDL). Sun then decided to change the license of the JDK to GPLv2 to encourage the use of Java. A problem related to license compatibility appeared: any program that runs under the JVM dynamically links to the runtime library that is a part of the JVM. Hence, this program is considered to be derivative work of the JVM, and hence should be licensed under the GPLv2. Consequently, Sun added the Classpath exception to the GPL2 to resolve this issue. This exception states that linking to the provided library is not considered a derivative work.

A third example is the case of MySQL client libraries, which were licensed under the terms of the LGPLv2. The LGPL license allows the reuse of a system licensed under its terms to create and distribute software under any license. In 2004, MySQL-AB changed the license of the MySQL client librairies to GPLv2 but they wanted to allow some Free/Libre and Open Source Software (FLOSS) systems to still use their libraries, even though their licenses are not compatible with GPLv2, *e.g.*, the PHP run-time engine. MySQL-AB resolved this issue by adding to its license the "MySQL FLOSS License Exception", which permits to create a derivative work based on MySQL client libraries to be licensed under any of 24 licenses, *e.g.*, BSD, MIT, Mozilla Public v1.0, PHP. The two previous examples show how to create non-GPL compatible programs based on GPL'ed programs, another solution is multi-licensing in which the user chooses the license from two or more licenses. An example of this practice is the Mozilla Foundation, which makes Mozilla, Firefox, and Thunderbird available under three different licenses: the Mozilla Public License version 1.1 (MPLv1.1), the GPLv2 or later, or the LGPL v2.1 or later.

Consequently, developers should be aware of license changes and their possible effects. Also, OSS systems are developped/maintained by many developers that could change a license of a file without being aware of the consequences of this evolution. Thus, to study license evolution, we will look at changes to license statements. These changes could produce some license incompatiblity in a system. Therefore, we must analyse the existing licenses of a software system before modifying it or adding more components under different licenses. However, developers are not specially trained to deal with licenses. In addition, manually detecting various licenses and their interaction is a laborious task. Thus, this problem raises the need for some sophisticated license evolution management techniques to assit developers to organise their software licenses in a better way.

Thesis:

> The license statements are changing frequently, but not necessarily coevolve with source code and managed by a minority of developers that are probably experts.

We will follow two research steps to confirm our thesis:

*Step*$_1$: First, we will study all entities/data involved in licenses and their evolution, as well as their relations, to design a system meta-model. Our meta-model provides informative data needed to study license evolution.

*Step*$_2$: Our meta-model indicates to us which data are related to license evolution. Then, this data will be extracted into a meta-model instance to be processed. After, we will analyze license statement and source-code co-evolution and license committers to answer our thesis and understand license evolution; if they evolve according source code evolution, or independently, or every project has his proper culture of evolution, and if they are modified by minority that could be probably group of experts. Our results could be used for future work to develop better licensing tools and techniques.

The data described by our meta-model might be also the support to develop a tool for license evolution management and the identification of different license evolution culture will be informative to define the requirements of this tool.

### 1.1.1 System Meta-model

To fully understand license evolution and all related entities, we first build a meta-model for license evolution. Such meta-models have been proposed to help in using OSS sysem to avoid license inconsistencies. These meta-models represent some license aspects, *e.g.*, grant and their conditions (German et Hassan (2009); Alspaugh *et al.* (2009)). Yet, the data presented in previous models is not sufficient to cover many entities that are important in resolve license issues. Hence, we expand previous meta-models and provide a complete meta-model. Using our meta-model, we will locate which aspects of licensing should be explored in detail in our work about license evolution. To build a complete license evolution meta-model, we first perform a literature review to find pertinent license related data to design our meta-model. Then, we extend this design by analysing additional elements that we found while studying license text of some popular licenses like GPL.

We think that there is need of a meta-model and tool to help maintaining system licenses and improving license changes process. Such help sustaining open source adoption in practice.

### 1.1.2 Co-evolution of License Statements and Source Code

Usually, when researchers study system evolution, they focus on the evolution of source code (different artifacts like class, method...) and of documentation, because they think that their evolution is important and could impact software quality. Thus, they try to find method to preserve this quality during the evolution of the system.

Since the license statements specify which license applies to which file and who owns copyrights, understanding the frequency and kinds of license statement changes and their risks is essential for a number of reasons. For one, license or copyright infringements can completely outweigh the financial gain of reusing OSS systems, which is why many companies are extremely cautious when reusing OSS components in their proprietary systems ????. For example, MySQL changed its license to the more restrictive license $GPLv2.0$ to prevent the commercial abuse. However, this change also prevented FOSS applications from including MySQL Client Libraries and could only be resolved by adding a license exception[5] for OSS systems. For another, license statement changes are not trivial because they are written in "legal" English, and do not necessarily follow strict formatting; the volunteers developing open-source systems may or may not be legal experts or have the proper training to fully understand the impact of a license statement change.

To confirm our thesis about co-evolution of software licenses and source code, we investigate the following research questions:

- **RQ1: Do licenses co-evolve with source code at the system level?**
  We want to relate license statement changes and software evolution to understand whether developers change license statements when they change the source code of systems, *i.e.*, whether the peaks of license statement changes are synchronized with peaks in source code changes or instead shifted in time. The distribution of license statement changes (dispersed or grouped by period) and their evolution relative to source code evolution will help us to (1) understand whether the process of license statement changes is a planned and organised activity relatively to SLOC changes, to (2) know how to design/develop a tool to improve the process of license management and avoid license inconsistencies, and (3) to decide if licenses should be managed together with source code or it can be an independent process.

5. http://www.mysql.com/about/legal/licensing/foss-exception/

Result:

License statement changes could occur as needed when a substantial contribution is made (addition of contributors) or whenever the legal team advises so (update of license version or type).

We find that license statements are changing frequently and continuously, but not necessarily together with source code.

**RQ2: What types of license changes are performed?**

We want to refine the analysis of RQ1 and distinguish between different change types to link our analysis closer to practice. For this, we will first identify different types of license statement changes, then study the co-evolution of SLOC and the number of license statements per change type. The answer to this question will either show that certain license statement change types occur in a similar degree across all systems, or that their evolution depends on the specific licensing policies or guidelines of an open source project.

Result:

Different kinds of license statement changes can evolve differently. We identified three main types of license changes: license type change, license version change and contributor change. We found that license type and version changes co-occur more often with SLOC changes than other license change types do.

**RQ3: Who performs license changes?**

There are two major groups of stackeholders related to source code changes: authors and committers. The author of a change is the contributor who physically changes a set of files, whereas the committer is the gatekeeper who decides whether those changes will be made available to the whole project by committing them into the source control system. Applied to software licenses, the author of a change might propose a change in a license, however it is the committer who has the authority to accept or deny this proposal. License statement changes could introduce inconsistencies and cause legal violations, thus it is important to know who is responsible for this risky task. For this reason, we study the committers of the seven projects to understand which ones are responsible for accepting license statements, and what their role is in the project.

Result:

> License statement changes could be limited to a minority of specialised committers, because they require thorough knowledge of dealing with legal issues. Alternatively, it could be left up to individual committers because open source developpers are sufficiently familiar with licensing. We observe that the most active committers (in the CVS or SVN repository) performing license statement changes are also the project members with a leading role.

## 1.2 Background

In this section, we define and clarify some concepts that we will use in our thesis.

### 1.2.1 Open Source Software

Open source software (OSS) development has some typical characteristics, such as the widespread reuse of components and licenses. This widespread of various and different licenses increases the difficulty to understand their constraints. Consequently, new re-engineering tool must consider the licenses analysis. OSS development process outputs have been studied to study many aspects of programs, for example in Capiluppi *et al.* (2003), they analyzed a sample of around 400 projects from a popular OS project repository. Each project is characterized by a number of attributes. According this study, the most used languages were C, C++, Perl, and Java. Despite the large number of OSS projects, developments effort have focused on a few large projects such as Linux, Mozilla, and Apache. In Capiluppi *et al.* (2003), Capiluppi and al. confirmed that few projects are capable of attracting a meaningful community of developers. The majority of projects is made by few (in many cases one) person with a very slow pace of evolution. We think that the analysis of licenses will be more useful in project with great community and in constant evolution because the evolution of the systems increase the threat of license violation and the large number of components and licenses increases the constraints to respect inter-licenses compatibility.

### 1.2.2 Collective and derivative works

Distinguishing between collective work and derivative work is fundamental for analysis of legal issues of component based software system, because constraints

imposed by license could be different for collective and derivative work.

A collective work is:

*A work in which a number of contributions, constituting separate and independent works in themselves, are assembled into a collective whole. (17 U.S.C. Â§ 101.)*

And a derivative work is:

*A work based upon one or more preexisting works, such as a translation or any other form in which a work may be recast, transformed, or adapted. (17 U.S.C. Â§ 101.)*

### 1.2.3 Types of Licenses

Licenses can be categorised into four categories:

1. Academic Licenses: *"so named because such licenses were originally created by academic institutions to distribute their software to the public, allow the software to be used for any purpose whatsoever with no obligation on the part of the licensee to distribute the source code of derivative works. The Berkeley Software Distribution (BSD) license used by the University of California to distribute its software is the archetypal academic license. Academic licenses create a public commons of free software, and anyone can take such software for any purpose including for creating proprietary collective and derivative works without having to add anything back to that commons."* Rosen (2004)
2. Reciprocal Licenses: *"Allow software to be used for any purpose whatsoever, but they require the distributors of derivative works to distribute those works under the same license, including the requirement that the source code of those derivative works be published. The GPL license, written by Richard Stallman and Eben Moglen at the Free Software Foundation, is the archetypal reciprocal license. Anyone who creates and distributes a derivative work of a work licensed under a reciprocal license must, in turn, license that derivative work under the same license. Reciprocal licenses, like academic licenses, contribute software into a public commons of free software, but they mandate that derivative work also be placed in that same commons."* Rosen (2004)
3. Standards Licenses: *"are designed primarily for ensuring that industry standard software and documentation be available to all for implementation of standard*

*products. These licenses sometimes require that any differences from the industry standard be published as a reference implementation so that the standard may evolve if necessary."* Rosen (2004)

4. Content Licenses, *"ensure that copyrightable subject matter other than software, such as music, art, film, literary works, and the like, be available to all for any purpose whatsoever. These licenses are discussed more fully on the Creative Commons website at* `www.creativecommons.org`. *While the Creative Commons goals are not directly related to software freedom, there are many similarities of objective. A few of the software licenses discussed in this book, in particular the Academic Free License (AFL) and the Open Software License (OSL), are appropriate for use with content as well as software, as will be explained in due course."* Rosen (2004)

### 1.2.4 Example of licenses: GPL, BSD, and Apache

In this subection, we present the most used licenses[6] : GPL, BSD, and Apache.

1. BSD: Academic License,
   Contrary to the GPL License, Berkeley Software Distribution license (BSD) allows anyone to redistribute the work or any derivative work without any source. So BSD do not cause incompatibility problems: the reader of program under BSD license can use any license.
2. GPL: Reciprocal License.
   GNU Public License (GPL) is very common license for open-source packages. GPL is known for having strict reuse constraints. So it is important to focus on incompatibility issues involving GPL license. GPL is reciprocal license because any software that reuses code licensed under GPL should be licensed under the same version of the GPL. Here the GPL say it:
   *"You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this license."* (GPL, Section2)
   They are strong conditions on how a caller can use GPL package. The GPL requires to analyse the software based not just upon how it is linked but also

6. http://www.opensource.org/licenses/category

upon how it is distributed: *"These requirement apply to the modified work as whole. if identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this license, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the must be whole on the terms of this License, whose permissions for other licenses extend to the entire whole, and thus to each and every part regardless of who wrote it".* (GPL section 2)

According to the first sentences, the GPL is applied to "modified work as whole". A modified work is derivative work (17 U.S.C). There is no hint that linking makes a difference. The second sentences refers to portions of the work that are not derived from the program (have their own copyright owners and their own license). A work must be independent and separate works are linked in some way to the GPL program. Such works remain "independent and separate works," at least "When you distribute them as separate works," and the GPL cannot possibly apply to them without their copyright owner's consent.

In the GPL, we must analyse the software on how it is distributed. But, we could guess every connector type correspond to which type of work (derivative or collective). Here we give the list of connector type and the license that must have the caller of GPLed program:

- if the caller uses via fork/exec then the caller can have any license.
- if the caller uses called components as a plugin then the caller can have any license.
- if the caller uses linking as types of connexion so it must be licenses under the same version of GPL.

The program licensed under academic open source licenses can be incorporated into GPL-licensed software but the inverse is not always true.

There some licenses are not compatible at all with GPL, we will limit these list to the different versions of the different version of BSD, and Apache licenses:

- Apache License, version 1.1.
  This is 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.

– Apache License, version 1.0.
This is a simple, 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.

And the compatible version of our chosen licenses (BSD, Apache) are:

– Apache License, version 2.0.
This is a free software license, compatible with version 3 of the GPL. This license is not compatible with GPL version 2, because it has some requirements that are not in the older version.

– Modified BSD license.
This is the original BSD license, modified by removal of the advertising clause. It is a simple, permissive non copyleft free software license, compatible with the GNU GPL.

Case of plugin : The legality depends on how the program invokes its plugins. If the program uses only simple fork and exec to invoke plugins, then the plugins are considered separate programs, so the license of the plugin imposes n constraints on the whole program. If the program dynamically links plugins, and they make function calls to each other and share data structures, thus they form a single program. To use the GPL-covered plugins, the main program must be released under the GPL or a GPL-compatible license, and that the terms of the GPL must be followed when the main program is distributed for use with these plugins. If the program dynamically links plugins, but the communication between them is limited to invoking the main function of the plugin, that is a borderline case. Using shared memory to communicate with complex data structures is equivalent to dynamic linking.

3. Apache license, version 2.0: Academic license.
The Apache license is a free software license authored by the Apache Software Foundation (ASF). The Apache license requires preservation of the copyright notice and disclaimer, but it is not a copyleft license; it allows use of the source code for the development of proprietary software as well as free and open-source software.
All software produced by the ASF or any of its projects or subjects is licensed according to the terms of the Apache License. Some non-ASF software is licensed using the Apache License as well. As of July 2009, over 5,000 non-ASF

projects located at SourceForge.net are available under the terms of the Apache License. In a blog post from May 2008, Google mentioned that 25% of the 100,000 projects then hosted on Google Code were using the Apache license.

Like any free software license, the Apache license allows the user of the software the freedom to use the software for any purpose, to distribute it, to modify it, and to distribute modified versions of the software, under the terms of the license. The Apache license, like BSD licenses, does not require modified versions of the software to be distributed using the same license (in contrast to copyleft licenses). In every licensed file, any original copyright, patent, trademark, and attribution notices in redistributed code must be preserved (excluding notices that do not pertain to any part of the derivative works); and, in every licensed file changed, a notification must be added stating that changes have been made to that file.

### 1.2.5 License compatibility and constraints

The intellectual property (IP) are expressed in terms of the licenses, rights, and obligations. They include: the right to use, distribute, sublicense, interoperation of the system with specific IP regimes. This IP can have conflicts with other licenses' obligations. So, the combination of different licenses in a single system is not simple because each license introduces constraints on the way of use (distribution, copy,...) that can be incompatible and also how we can reuse a program by integrating it to another system or modifying it. We have to know the IP to be able to identify the possible legal combinations of licenses in one system.

For example, when programmers want to develop a system S under a license L that reuses an open-source component C, they must verify whether they respect the restrictions of the grant given by the license of C. In fact, a component can be reused to create from it a derivative work mainly by using white-box form that permits to use one or more files of C, either in the original or modified form. It can be also used as part of collective work that is usually realized via black-box form for example by calling components as executables. But determining whether a work is derivative or collective work for a black-box reuse is difficult because it depends on the nature of the use and the interconnection type.

Consider the following scenario: suppose we want to distribute a system S under a proprietary license P and one of the component $C_i$ of S is licensed under the terms

of $GPL_2$. C is interconnected to S via black-box linking, S is a derivative work of C. $GPL_2$ imposes that all derivative work S made from component under $GPL_2$ must be also licensed under $GPL_2$. In contrast, if we modify the interconnection type, and that black-box forking is used instead of black-box linking, then, according to the FSF, S is not a derivative work of C. In this case $GPL_2$ gives grant to distribute S under a proprietary license German et Hassan (2009)Rosen (2004). This example show us how can the interconnections type be a constraint to respect the IP and it depends on the licenses used and their versions and it is complicate to verify this respect of the IP of a large software.

## 1.3 Thesis Plan

This thesis is organised as follows: Chapter 2 summarises work related to license analysis, and the Chapter 3 presents a meta-model for license analysis. Chapter 4 presents our study setup, while Chapter 5 addresses our research questions and discusses our results. The Chapter 6 presents a preliminary step for a tool that helps to avoid license inconsistencies. Finally, Chapter **??** concludes the thesis and presents future work.

# Chapter 2

# STATE OF THE ART

Previous research mostly targets technical problems of software development and maintenance, without much attention for the legal complexity of software systems (German *et al.* (2010b)). We discuss related work on (1) license analysis, (2) license evolution, and (3) license identification tools. Overall, no previous work considered the relation, if any, between code change and license modification or between source code committers and developers performing license evolution, except for some work that analysed license statements independently of source code. Some work proposed a system meta-model that is concentrated on license modeling and did not consider other related data.

## 2.1 Meta-model and Software License Analysis

German *et al.* (German et Hassan (2009)) defined a license as a set of grants, each of which has a set of conditions necessary for the grant to be given. They analysed the interactions between pairs of licenses in the context of five types of component interconnections: linking, forking, subclassing, IPC, and plugins. German *et al.* also identified and discussed 12 patterns to avoid license mismatches caused by license changes, found in a large group of OSS systems. They described patterns commonly used to solve license mismatches in practice.

German *et al.* (German *et al.* (2010b)) proposed a method to understand several licensing incompatibility issues, concerning mismatches between the license of a system and that of its source code files, or its libraries that can arise from changing, combining, and re-distributing packages in open distribution. They carried a large empirical study aimed at analyzing licensing issues in the entire Linux-based Fedora-12 operating system. They considered constraints imposed by OSS licenses, relied on these constraints to mine inconsistencies, and identified the licenses and dependencies of all files using RPM package descriptions. They concluded that there exist many nuances in determining the license of a binary package from its source

code, for example, many packages could contain source code under different licenses. Moreover, they found many cases in which the license of a package changed, and this created problems, *e.g.*, the package still declared the old license, making the package use potentially incompatible. Such mismatches are common in modern open-source systems (German *et al.* (2010b)), which supports our claim that license maintenance must be carefully managed. Hence, we are looking when licenses evolve and who changes them.

Alspaugh *et al.* (Alspaugh *et al.* (2009)) used a semantic parameterisation of nine OSS licenses and the patterns and models established by German *et al.* (German et Hassan (2009)) to derive a meta-model for licenses shown in figure 2.1. This license model considers semantic connections between obligations and rights. The goal of this metamodel is to support analysis and management of the license constraints. They developed a tool that supports intellectual property requirements management.

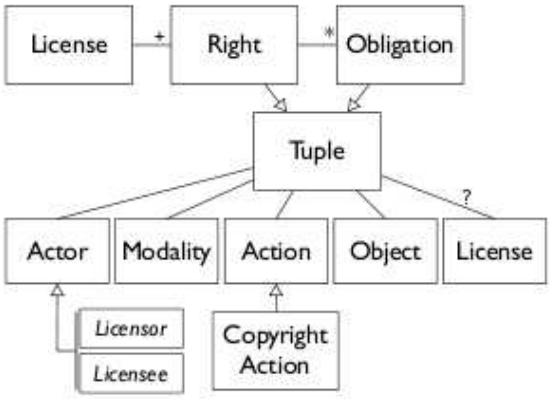



Figure 2.1 The meta-model for licenses

Tuunanen *et al.* (Tuunanen *et al.* (2009)) also tackled license mismatches in OSS systems. They implemented a tool, ASLA, to identify licenses in source code and to identify mismatches using compiling information from GCC, ar (an archive tool), and ld (a linker). They achieved license identification using templates and regular expressions. Their license identification does not work well with real source code files because of many reasons, *e.g.*, comments and various kinds of white space characters prevent an exact matching, many developers modify predefined licenses, there are different published versions of licenses.

Hemel *et al.* (Hemel *et al.* (2011)) focused on identifying license violation in third-party packages distributed in binary releases of several systems. They developed a tool, Binary Analysis Tool, that compares a given binary against a large repository of packages using clone detection and provides as output a list of third-party packages likely used in the binary; thus the compatibility of their licenses and the license of the whole system must be checked. They did not study whether license mismatches occurred between packages.

Similarly, Cordy *et al.* (Cordy et Roy (2011)) proposed DebCheck, a clone detection tool to perform cross-package clone detection. It is based on the NiCad clone detection tools developed by the same author and was used to verify whether GPL or other OSS-licensed code has been copied into other systems.

Di Penta *et al.* (Penta et German (2009)) studied changes of the names of copyright owners. They found that contributor names are added to a license statement upon changes that are significantly larger than usual (in terms of numbers of lines of code changed). They also found that the most frequent committers are not necessarily the copyright owners.

The above cited works focused on license modeling and license violation detection. In this paper, we want to investigate another direction in the same field: the evolution of license statements and its relation with source code changes.

## 2.2 License Change Analysis

Hindle *et al.* (Hindle *et al.* (2008)) studied large commits in OSS systems. Among other things, they identified license statement changes as one of the reasons for bulk file changes and large commits.

Di Penta *et al.* (Di Penta *et al.* (2010)) studied license evolution. They proposed

an approach to automatically track changes across the license statements of source code files. An empirical study on license evolution of six OSS systems showed that license statements change frequently and, thus, that is why it is necessary to study these changes in more detail. Furthermore, Di Penta et al. found that the changes occurring to the copyright year depend on the amount of changes made by developers during the year. However, they did not relate the license changes to system evolution or identify committers of license changes. In our thesis, we will propose a meta-model for license evolution. Then, we study the license statement evolution, in addition we relate them to software evolution, we will identify the license statement committers.

Manabe *et al.* (Manabe *et al.* (2010)) studied how and why ArgoUML, Eclipse, FreeBSD, and OpenBSD switched licenses. They found that: (1) the number of licenses used in operating systems are larger than those in other open source systems; (2) projects sometimes choose radically different licenses; and, (3) the usage of different licenses in the kernel files of operating systems is similar to each other. Their study did not consider software evolution. In contrast, in our work, we focus on license statement and source code co-evolution to understand if license statements evolve according software evolution and they have their proper evolution pattern.

## 2.3 License Identification Tools

A license statement is a comment block on top of a source code or other file that contains the terms under which the file is licensed. The elements of a license statement are the license or licenses that cover the file, a list of copyright owners, a list of contributors, warranty and liability statements. However, the format of license statements is not strict and can be customized. As such, detecting and identifying licenses is not trivial, and specialized tools are needed.

We focus on three tools: FOSSology (Gobeille (2008)), OSLC[1], and Ninka (German *et al.* (2010a)), these are the state-of-art. FOSSology automatically identifies licenses in a license statement using a Binary Symbolic Alignment Matrix pattern matching algorithm. Its negative points are the complexity of setup, the need of a running database, and its low speed. OSLC is more simple, because it uses regular expressions. However, it is prone to false positives. For example, a file is reported to be using the GPL when it finds: "*This file is not licensed under the GPL*". Compared

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

to the previous tools, Ninka is the most accurate one (German *et al.* (2010a)). Each license statement corresponds to a sequence of one or more sentence-tokens. Ninka extracts the license statements from files, splits them into textual sentences that are normalized, and tries to find a match for each of these sentences with the license sentence-tokens. The list of the matched sentences determines if a file contains one or more licenses. Due to its high accuracy, we used Ninka in the rest of this work.

# Chapter 3

# SYSTEM META-MODEL FOR LICENSE ANALYSIS

In this chapter, we propose a system meta-model for license evolution analysis. Then, we show an example of use of our meta-model combined with logical expressions to express constraints imposed by a license.

## 3.1 Meta-model Design

In this section, we present our meta-model. We combined different sources of information to model data that we include in our meta-model. We used previous work about license analysis, some of them (German et Hassan (2009); Alspaugh *et al.* (2009)) propose a meta-model that are in general limited, *i.e.*, the meta-model established in (German et Hassan (2009))) did not design well the system architecture, *e.g.*, interconnection between different component is not presented, which is important to find license inconsistencies, and in (Alspaugh *et al.* (2009)) Alspaugh derived a meta-model for licenses from the meta-model of German where he added a semantic connections between obligations and rights but he did not also consider in the meta-model at all the system architecture representation. But, we prefer to assemble all needed data: license meta-data and architecure in one meta-model with semantic links between them. In addition, we analysed the content of license terms to collect pertinent data.

We show our meta-model in Figure 3.1.

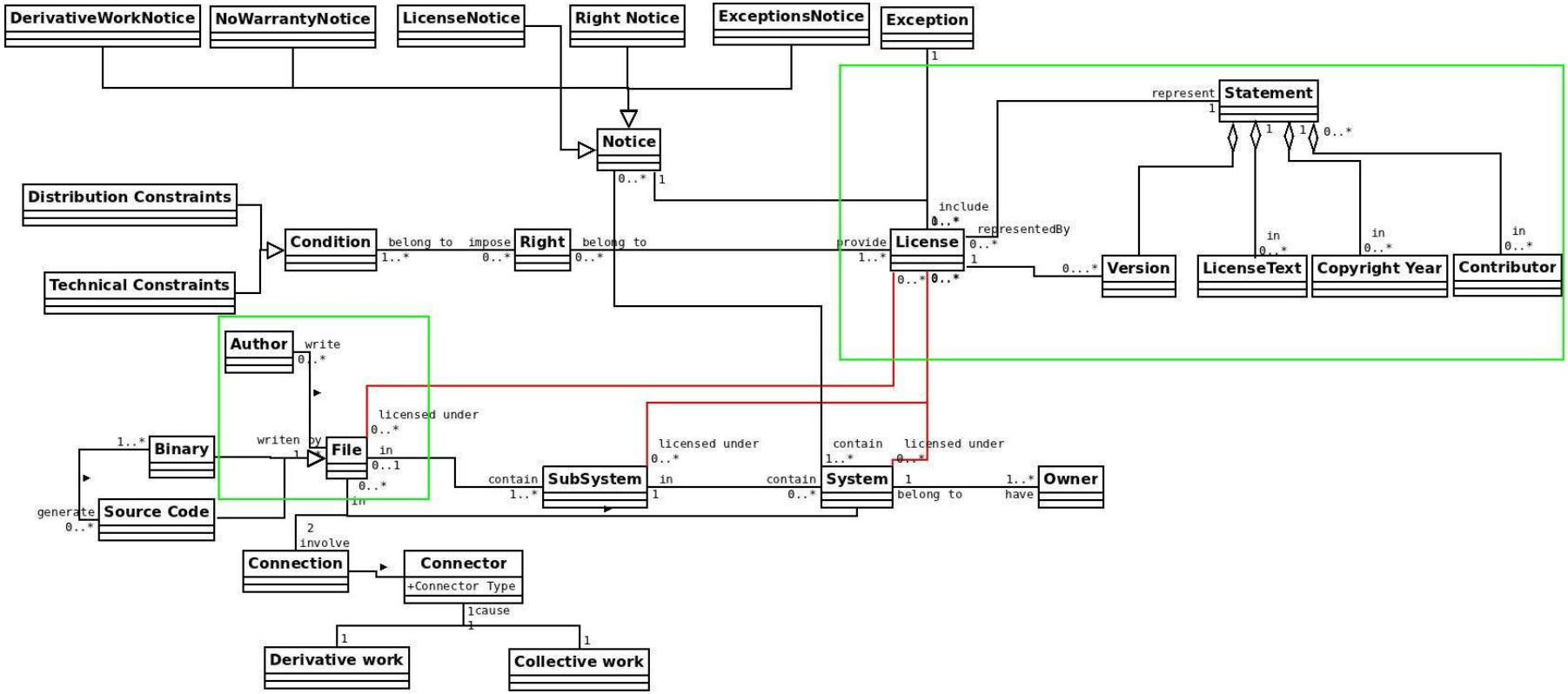



Figure 3.1 System Meta-Model.

As shown in Figure 3.1, a System can be composed of zero or many packages denoted Sub-System and Files. A Sub-System can be composed also of zero or many Sub-Systems and Files. The System, Sub-Systems and Files may have zero or many Licenses. The files of the same sub-system can have different licenses as well as the Sub-Systems.

To Ferdaous: ▶ *instanciate your metamodel* ◀ We present an example of two concrete systems using our meta-model.

- Case of **findUtils V4.4.2**.
  The Findutils[1] is package contains programs to find files under linux. The system S is "findUtils". findUtils contains XARGS, LIB, M4,... XARGS, lib, M4 are Sub-systems (S). findUtils at top level contains also: README, ChangeLog, AUTHORS,..., which are files (F) and they do not belong to XARGS or LIB or M4 subsystems. File may belong to a package or not. When we compile findUtils V4.4.2, the binary *find* is generated.
- Case of **fileUtils v3.16**. The fileutils package includes a number of GNU versions of common file management utilities. Fileutils includes the many tools: mv, chown, chmod, mv, du, old,... In the case of fileUtils, the system S is named "fileUtils" and it contains two SubSystem (first level) lib and M4 are Sub-Systems. fileUtils at top level contains files: README, ChangeLog and Config.in. If we compile fileutils v3.16, the binaries that we obtain are *chmod, chown, mv, rm, old, du* are generated.

## 3.2 Definitions of Meta-model Constituents

In our meta-model, we have a set of entities and relations between them. We define each entity as follows.

- System (S): system is the collection of all files and sub-systems also said software system or software.
- Sub-System (SS): a set of files with an organization such as to constitute an independent component that can be distributed separately and/or reused in other system.
- File (F): it is a collection of bytes stored in same format, it can be an ASCII or binary "file".

1. http://www.linuxfromscratch.org/lfs/view/development/chapter06/findutils.html

- Binary (B): an executable, library, stored object no in a plain ascii format aka (English/Italian/Deutsch Text).
- Source code (SC): is a text written using the format and syntax of the programming language.
- Software license/license (L): is a legal instrument (written into a text file) to govern the use and distribution of a software. It is a set of terms (explanations and conditions), exceptions, warranties, version, statements, notices.
- Version (V): is unique identifier attributed to unique states of the license, the version number is generally assigned in increasing order and correspond to new feature in the license. For example, GPLv2 (version 2 of GPL license), BSD-3 (version 3 of BSD license), etc...
- Statement (ST): for a given license, is a summary text of the license terms to be inserted at a beginning of a file to claim the file being licensed under the specified license. And It contains always the name of contributors and the copyright year and it indicates the version of the license.
- Term (T): it has two semantics, it could be 1) an explanation of a word used in the license, *e.g.*, "convey" : any kind of propagation that enables other parties to make or receive copies, or it could be 2) a right and its conditions that must be satisfied.
- Exception (E): a modification or addition to the standard license conditions.
- Notice (N): is a written information *i.e.*, license text, by which a party *i.e.*, the user of the program concerned by this notice, is made aware of a legal process affecting their different rights, obligations or duties [2] (creation of derivative work, warranties,...). It could also indicates an exception.
- NoWarranty Notice : it is a notice that make the user aware that there is no warranty given. A warranty is an assurance by the licensor to the other party that specific facts or conditions are true or will happen; it is an insurance of good quality and functioning; the other party relies on that assurance and seeks some type of remedy if it is not respected [3].
- Author (Auth): "the person who originates or gives existence to a file". Holding the title of "author" over a file gives rights to this person, the owner of the copyright, exclusive right to do or authorize any copy or distribution of this file.

2. http://en.wikipedia.org/wiki/Notice
3. `http://en.wikipedia.org/wiki/Warranty`

Any person or entity wishing to use intellectual property held under copyright must receive permission from the copyright holder to use this work.

- Contributor (): is a person that contributed in writing/modification of a file
- Owner (): "The programmer who writes software or the company that hires that person to write software is deemed to be the first owner of intellectual property embodied in that software. That owner may exercise dominion over that intellectual property. He can give it away, sell it, or license others to use it. That owner has the prerogative to create copies of the intellectual property, and he or she may prevent others from making, using, or selling those copies."[4]
- Right (): an open software license provides its licensee with a grant to one or more of the exclusive rights owned by the copyright owner of that component. Each grant is given provided a set of conditions are satisfied.
- Condition: A future and uncertain event upon the happening of which certain rights or obligations will be either enlarged, created, or destroyed.[5]
- Technical constraints or distribution constraints: the conditions that must be satisfied to have a right could be technical constraints, *e.g.*, architecture style, or distribution constraints, *e.g.*, notice of no warranty must be distributed with source code.
- Collective work : a work in which a number of contributions, constituting separate and independent works in themselves are assembled into a collective work as whole.
- Derivative work: "a work based upon one or more preexisting works in which a work may be recast, transformed, or adapted"
- Interconnection (): between two entities (file, susbsystem, system) in any use of an entity by the other so I(e1, e2) means e1 uses some data, services, functionality provided by e2. The interconnection need connector to realize it.
- Connector (): is glue that links several files, is the required physical linking between several entities, files, to realize an interconnection.
- Connector Type (): we can classify the connectors in to four types: Link, fork/exec, IPC, Plugin:
  - Link (LK): any kind of function call, global data usage, method call made to statically or dynamically linked artifact. Example: if we have an OO

4. `http://rosenlaw.com`
5. http://legal-dictionary.thefreedictionary.com/condition

framework and we extend a class or call a method, it is considered a Link connector.

- fork/exec (FE): a child process is created and a new executable loaded and run.
- IPC: any kind of Inter Process Communication such as pipe, shared memory, queue, socket,...
- Plugin (PE): dynamically loaded component adding/extending specific functionality via an API.

As, we explained before, it is important to determine if the work is derivative or collective work to be able to judge if any sub-system or file can be used in the final system. The connector used in the program are the key to decide if the system is derivative work of one of its sub-system or file. Reciprocal licenses (GPL for example) consider the system that use LK and some PL connector to use a component (sub-system or file) c as derivative work, thus the system must be distributed under the same reciprocal license of the component c. The FE and IPC are not problematic for reciprocal license, the system that use a component (sub-system or file) licensed under reciprocal license using these type of connectors can be distributed under any license.

To automate the process of deciding if the system is derivative of one of its component (sub-system or file), we need a function *Derivative* that takes as parameter two systems and a connector type and returns True or False.

Let $S_N$ be the whole system.

Let $S_w$ be the set of sub-systems/files used by $S_N$.

for each $s \in S_w$

print $Derivative(s, ConnType(S_N, s))$

$Derivative(s, ConnType(S_N, s)) \in \{0, 1\}$

if $S_N$ is derivative work of $s$ then $Derivative(s, ConnType(S_N, s)) = 1$ else $Derivative(s, ConnType(S_N, s)) = 0$. The fact that $S_N$ is derivative work of $s$ or not depends on $I(S_N, s)$ and $L(s)$.

For example, if $S_N$ contains a Sub-System $s$, $L(s) = GPLv2$ and $ConnType(S_N, s) = LK$, thus $S_N$ is considered a derivative work of $s$ and $Derivative(s, ConnType(S_N, s)) = 1$

# Chapter 4

# STUDY SETUP

We performed an empirical study to answer our three research questions presented in the chapter 1. In this chapter, we define our study, then we present the context of the study by giving the objects that we considered. Next, we describe the steps of our approach and we explain how we used the proposed meta-model. Finally, for each research question we explain the analysis method that we will use to analyse our data and interpret the result.

## 4.1 Definition of Our Study

Following GQM Basili et Weiss (1984), our *goal* is to perform an exploratory analysis of the co-evolution of license statements and source code, to observe license statements evolution and to analyze who performs license statement changes. Our *purpose* is to better understand when developers change license statements, who performs such changes, and how license statements are changed. Such an understanding could help improve license change management. The *quality focus* is the consistency of license changes. The *perspective* is of both researchers and practitioners who are interested in understanding license statement change activities in software projects. The *context* of our study are the CVS/SVN repositories of seven OSS: JFreeChart, Jitsi, PHP, Rhino, Tomcat, XalanJ, and XercesJ.

## 4.2 Context

The *objects* of our study consist of seven OSS systems, i.e, JFreeChart, Jitsi, PHP, Rhino, Tomcat, XalanJ, and XercesJ [8]. Table 4.1 presents some descriptive statistics

8. http://www.jfree.org/jfreechart/, http://jitsi.org/, http://www.php.net/, http://www.mozilla.org/rhino/, http://tomcat.apache.org/, http://xml.apache.org/xalan-j/, http://xerces.apache.org/

| Object Systems | #Files | #Releases | License of last release | Considered History |
|---|---|---|---|---|
| JFreeChart | 1,335 - 9,105 | 51 | LGPLV2.1+ [1] | 25/11/2000 - 20/04/2009 |
| PHP | 2,615 - 15,021 | 63 | PHP License v3.01 [2] | 12/07/1999 - 18/05/2011 |
| XercesJ | 5,100 - 12,585 | 39 | Apache License v2.1 [3] | 05/11/1999 - 01/01/2010 |
| Rhino | 104 - 695 | 17 | MPL 1.1/GPL 2.0 [4] | 19/04/1999 - 16/09/2010 |
| Tomcat | 2,565 - 7,426 | 70 | Apache License v2 [5] | 08/10/1999 - 14/09/2011 |
| Jitsi | 5,653 - 15,954 | 8 | LGPL [6] | 21/07/2005 - 12/09/2011 |
| XalanJ | 832 - 1,433 | 14 | Apache License v2.0 [7] | 09/11/1999 - 11/12/2009 |

Table 4.1 Statistics of our seven subject systems.

of these systems. JFreeChart is a free Java chart library to display professional quality charts. Jitsi (previously SIP Communicator) is an audio/video and chat communicator. PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. Rhino is an open-source implementation of a JavaScript interpreter in Java. Tomcat is an open-source software implementation of the Java Servlet and JavaServer Pages technologies. Xalan-J is an XSLT processor for transforming XML documents. XercesJ is an open-source family of packages for parsing and manipulating XML. We chose also these systems because they are medium-sized OSS, yet small enough to manually verify our observations on license statement and source-code co-evolution using external information, such as bug reports. We chose these systems also because their evolution history is long enough to contain substantial license statement evolution.

## 4.3 Setup of the Study

Our approach is illustrated in Figure 4.1 and consists of 5 steps.

*Step 0:* Using our meta-model, we determined which entities must be considered in our study to track the evolution of license and source code. According to our meta-model, a license of file is indicated in the license statement which is composed of license text (version, terms,...), copyright year, contributor list. Thus to find license changes we have to find change in license text, copyright year, and contributor list.

*Step 1:* First, to improve performance, a local copy of the CVS/SVN repository of each studied system is downloaded.

*Step 2:* We then use Ibdoos, our group's framework for the analysis of source control systems. Ibdoos parses change-log files (both CVS/SVN) to extract the following change facts: commit date, revision number, author, filename and log comment. This

, which implements our meta-model and provide a database to store instances of this meta-model

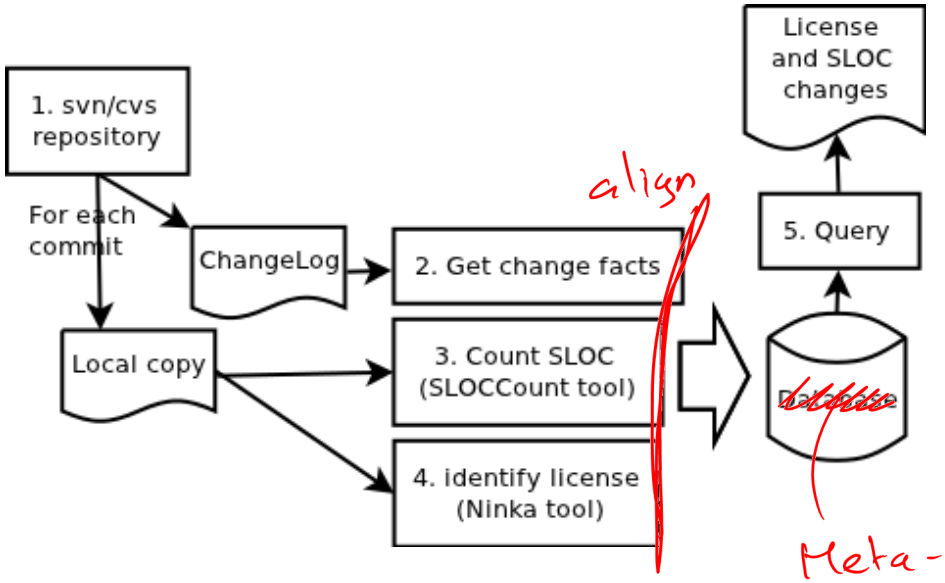



Figure 4.1 Approach overview.

information is stored in a relational database for later processing and computation. As we are interested in the source code and license evolution, we only analyzed source code files , *i.e.*, *.java* files for Java systems, *.c* for C systems, and *.c* and *.cpp* files for C++ systems. Note that other files such as READMEs, configure scripts or Makefiles can be analyzed as well, but fell outside the scope of this paper.

*Step 3:* Once all revisions of all the files are available, we compute the Source Lines of Code (SLOC) count of each file at each revision using the SLOCCount tool[9]. SLOCCount counts just source code lines and excludes whitespace and comments (and hence license statements). As we want to relate maintenance effort evolution to license statement evolution, we decided to use the evolution of SLOC because it is correlated to maintenance effort Hayes *et al.* (2004, 2003). Alternatively, one could use churn as a measure of effort.

*Step 4:* At this step, our goal is to extract the value of license statement that we identified in the step 0 which is composed of license text (version, terms,...), copyright year, contributor list. Thus, we invoke Ninka German *et al.* (2010b) to identify the licenses of each file. Ninka provides the license of the file, the license version (*e.g.*, GPLv3) and the list of file contributors, all of which are fed into the Ibdoos databases. Ninka also generates a list of so-called "unmatched sentences".

9. `http://www.dwheeler.com/sloccount/`

Indeed, it may happen that a file contains one or more licenses that have not been identified by Ninka or extra text such as comments about the code. In this case, Ninka will report the list of sentences that it was not able to match with any sentences of a known license. To reduce the risk of missing important license information, we decided to also look inside the unmatched sentences for license information. We did this by manually scanning the unmatched sentences for license information, then using regular expression patterns to mine this information in an automated way. Once licenses have been identified for a file, its licenses are compared for each pair of consecutive revisions. If the comparison detects a textual difference, we consider this to be a license statement change. License statement changes and all related data, once available, are then stored in Ibdoos' databases stored according our meta-model (The part of the meta-model framed in green, see Figure 3.1).

*Step 5:* Finally, we query the Ibdoos databases to analyse the co-evolution of license statements and source code. The next subsection explains the analyses we had to perform.

## 4.4 Analysis Methods

### 4.4.1 RQ1: Do licenses co-evolve with source code at the system level?

Using the data in the Ibdoos databases, we compute the number of license statement changes performed in different periods of time—discretised on a 15-day basis. We do this analysis twice, once with and once without the initial introduction of a license. This allows us to isolate of the effect of the initial introduction of a license. We also compute the difference in SLOC between successive versions in each object system—again discretised on a 15-day basis. Note that we discretised the collected data because the data would be too sparse otherwise and hard to compare. We adopt a sampling granularity of 15 days as a compromise, as argued by Kenmei *et al.* (2008): fine-grained data such as a daily-based discretisation is likely to be too detailed (many events at which no license statement change happens), while 2 week-or longer discretisation may average out interesting facts. In Eshkevari *et al.* (2011), our colleagues confirmed that 15-days is a sufficient granularity to track changes.

On this data, we perform both a quantitative and a qualitative study.

**Quantitative study.** We compute the cross-correlation between two time series, i.e., the time series describing the number of all license statement changes and the time series describing the evolution of SLOC for all the files in a system. We also compute the cross-correlation between two other time series, *i.e.*, the time series describing the number of all license statement changes excluding the initial addition of a license and the time series describing the number SLOC changes for all the files in a system. Cross-correlations are computed automatically for different lags between the two series. The maximum lag is $10 \times \log 10(N/m)$ where $N$ is the number of observations and $m$ the number of series. These cross-correlations will permit to check whether the license statement changes are correlated with major events in the evolution of a software system. Cross-correlation r can take on any value in between the following extreme values: perfect positive correlation ($r = +1$), where, as the number of SLOC changes increases, the number of license changes are predicted to increase at a similar rate; zero ($r = 0$) or no correlation; and, perfect negative correlation ($r = -1$), where, as the number of SLOC changes increases, the number of license statement changes decreases. We note that the r value takes into account lags. We assume that a positive or negative correlation indicates that the license and source code co-evolve. The case of zero correlation indicates that the license statement changes are not planned together with source code changes.

**Qualitative Study.** The cross-correlation will reflect whether there is a general tendency of co-evolution of license and source code, but this general trend could hide some particular cases. The complementary qualitative study will focus on such particular cases where there is some correlation between the evolution of SLOC and license statement changes. We start the analysis by plotting the three time series, i.e, (1) the number of license statement changes performed in different periods excluding the initial addition of a license, (2) including all license changes, and (3) the number of added/removed lines of code. We analyse these curves to assess whether there is a relation between license changes and the evolution of SLOC. We locate the peaks in the license statement changes relatively to peaks in SLOC changes to understand whether the license changes are planned relatively to the maintenance cycle or major events during development, whether it is a continuous process, or whether it has no

special distribution throughout time. We use external sources of information like mailing lists, change logs and release notes to interpret our observations.

### 4.4.2 RQ2: What types of license changes are performed?

Previous studies have suggested that there are different kinds of license statement changes, a finding that can be used to refine the result of RQ1. Hence, we analyzed Ninka's output to distinguish different types of changes. Ninka reports data about four elements: license name, license version, unmatched sentences, and the number of contributors (in some systems), because of project-specific coding conventions, it could not identify all the elements for all the systems. For example, in some cases the license name is not identified. For that reason, we used the information in the unmatched sentences. We parsed Ninka's output to compute the occurrences of each type of license statement change.

Using a histogram, we get information about how changes are distributed different types of changes. Once these types are identified, we compute the cross-correlation for each type of license statement change between two time series, i.e, the number of license statement changes discretised on a 15-days basis and the evolution of SLOC. As in RQ1, the cross-correlation ranges from perfect negative correlation, over no correlation to perfect positive correlation results. The cross-correlation results of RQ2 are more refined than the ones of RQ1, because we are considering each type of license statement changes seperately instead of aggregating all types of changes together. Hence, the correlation could be positive/negative/zero for specific types of license statement change and not for others.

### 4.4.3 RQ3: Who performs license changes?

We compute the number of commits performed by each developer in the three systems using the Ibdoos databases. Then, we identify the top seven committers that changed license statements. We select the top seven, since that number covers the most active committers in most of analysed systems Eshkevari *et al.* (2011). We ranked the committers using their total number of performed SLOC changes to measure their activities. This data allows to find how many committers modify licenses and the relatin between license statement change activity and developement activity. If the committers changing the licenses are a minority and their activities

are mainly changing licenses, we can say that there is a core of license experts in the project.

# Chapter 5

# RESULTS AND DISCUSSION

This chapter is composed of two sections. First, we answer the three research questions established in the chapter ??. Then, we discuss our results and we present the threats of validity.

## 5.1 Study Results

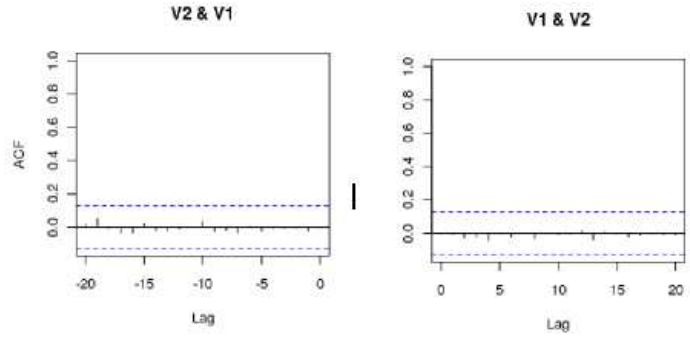
This section presents the results of the three RQs.

### 5.1.1 *RQ1: Do licenses co-evolve with source code at the system level?*

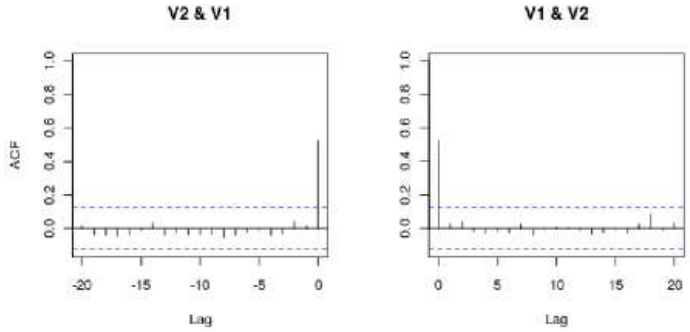
**Quantitative Study.** Figure 5.1 plots the results of the cross-correlations of three systems.

From the macro-level study, we cannot observe systematic large-scale license changes accompanying large restructurings of the system, Except for Tomcat, where cross-correlation reaches 80% (discussed later). The cross-correlation values[1] are almost zero for the non-zero lags between the time series. For example, PHP cross-correlation values vary between -5% and +5%, while those for XalanJ vary between -10% and 50%, and those for Tomcat vary between -40% and 80%. Other projects have similar ranges.

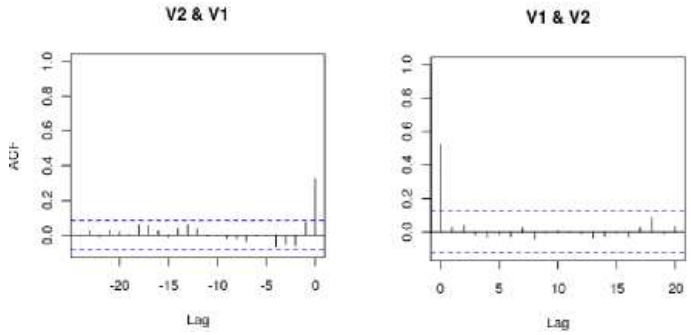
1. Detailed results are available in the annexe




(a) PHP.




(b) XalanJ.



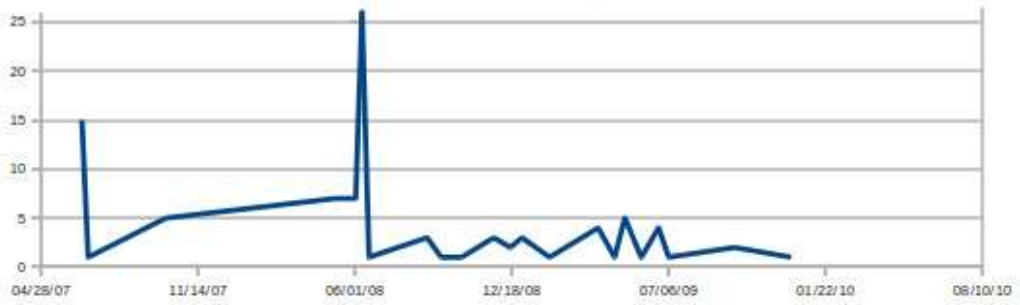

(c) XercesJ.

Figure 5.1 Cross-correlation values between license and SLOC changes in all files.

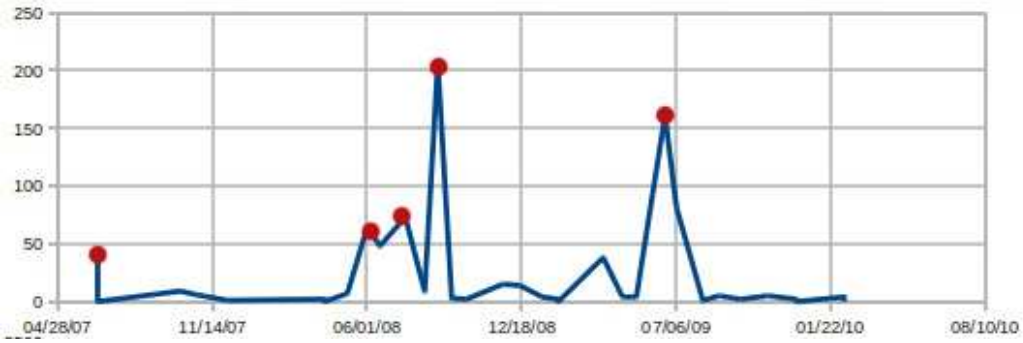
However, since the cross-correlations value are different from zero and reach up to 80% in some cases, it is possible that the license changes are performed during intensive maintenance periods. To understand this phenomenon in more detail, we conduct the qualitative study.

**Qualitative Study** We performed our qualitative study on three systems out of the seven analysed systems, *i.e.*, JFreeChart, PHP, and XercesJ, we chose these three systems because they have different licenses (LGPLv2.1+, PHP, Apache) and sizes. Figures 5.2, 5.3, and 5.4 plot the corresponding evolution of the number of SLOC and license changes performed. Figures 5.2(a), 5.3(a), and 5.4(a) show the number of license changes excluding the initial addition of a license to new files, while Figures 5.2(b), 5.3(b), and 5.4(b) show the number of all license statement changes. Figures 5.2(c), 5.3(c), and 5.4(c) show the evolution of the SLOC. The red dots are the peaks in the number of license statements that correspond to peaks in SLOC evolution. We observe that license statement changes are relatively frequent, for example PHP reaches an average of 14 changes per two weeks. This observation is not surprising and confirms previous observations by Manabe *et al.* Manabe *et al.* (2010) and Di Penta *et al.* Di Penta *et al.* (2010). We also observe that license statement changes are in general dispersed over time with only some specific limited time frames in which license statement changes are concentrated (red dots). In the following, we will give more details about such changes.

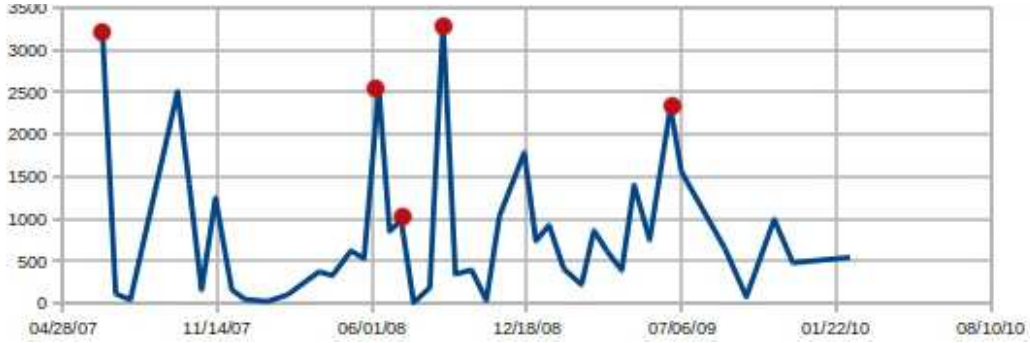
**JFreeChart:** We can see several red-dotted peaks for license statement changes (see Figure 5.2(b)), for example September $1^{st}, 2008$ (206 changes), June $22^{nd}, 2009$ (161 changes) and July $7^{th}, 2009$ (81 changes). These peaks correspond exactly to three peaks in SLOC evolution (see Figure 5.2(c)), *i.e.*, September $1^{st}, 2008$ (3319), June $22^{nd}, 2009$ (2323) and June $7^{th}, 2009$ (1556). The most frequent license statement changes on these dates are: (1) adding new contributor(s) to the license statements and (2) adding a license to a newly created file. We looked manually to changes corresponding to these peaks, and also checked the comments corresponding to the commits on these dates. We found that the majority of the red-dotted peaks indeed can be explained by developers updating the names of contributors during large source code modifications. These findings confirm earlier findings of Di Penta *et al.* Penta *et al.* (2009).




(a) License changes excluding the introduction of licenses to newly created files.




(b) License changes including the introduction of licenses to newly created files.




(c) SLOC evolution.

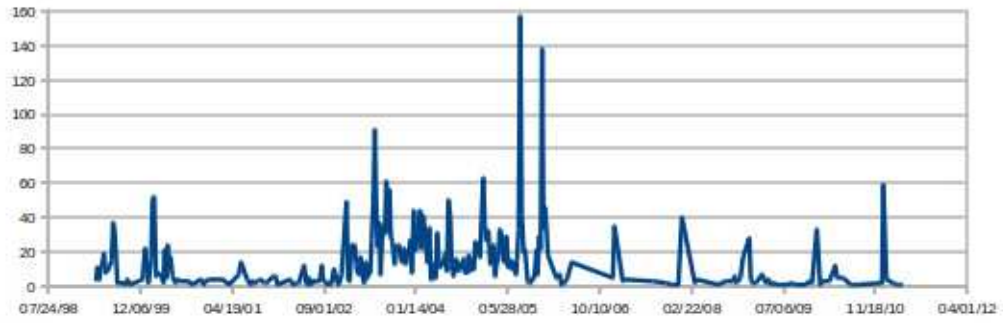
Figure 5.2 Evolution of SLOC and license statement changes over time in JFreeChart.

**In PHP.** The licenses are generally changed to upgrade their version number, for example from PHP license $v2.02$ to PHP license $v3.0$. We can see several peaks in license statement changes that correspond to the release dates[2] of PHP (see Figure 5.3(b)), for example:

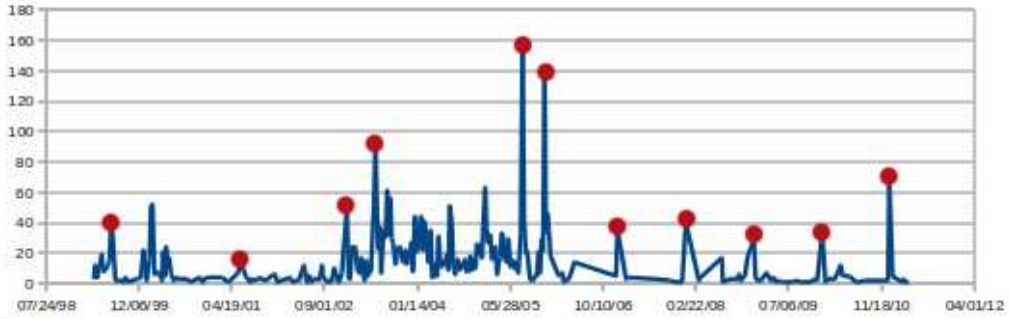
1. On May $22^{nd}, 2000$: PHP $v4.0.0$ is released. We observe that, just before this date, there are many license changes in the "Zend" package. On May $18^{th}, 2000$, the committers updated the PHP license $v2.01$ to PHP license $v2.02$ by adding the new clause 6 (Revision 24539). On May $19^{th}, 2000$, committer "Zeev" corrected the URL in the license of the "Zend" package three times. This was not straightforward, since each time he made a change, he introduced another error, for example he did not mention the URL in the correct place in the license statement. Finally, on May $22^{sd}, 2000$ he logged his final change with "*Sigh, that should be the last one*". Even though this license statement change problem was harmless, it shows how committers can easily make errors while changing a license statement.
2. On July $22^{nd}, 2002$, PHP $v4.2.2$ is released. We see that, just before this date, two major license statement changes were performed. On July $21^{st}, 2002$: the committers removed the clause and the license of all the files in the "Zend" package and they replaced them by a notice at the end of the license file. On the same day, they updated the PHP license $v2.02$ to PHP license $v3.0a1$.
3. On August $25^{th}, 2003$, PHP $v4.3.3$ is released. The committers updated the PHP license $v2.02$ to PHP license $v3.0$ just before this date.

We mined the change log of PHP to find information about these license changes. We noticed that the copyright year changed periodically at the end or the beginning of the year (January $1^{st}, 2009$, January $1^{st}, 2007$, January $1^{st}, 2006$ and December $31^{st}, 2002$). This type of change is not detected by Ninka, but instead we found it by mining the change log file of PHP using `grep` for specific expressions like: "Bump year", "update year", "year++", "update copyright year", "copyright year", and others.

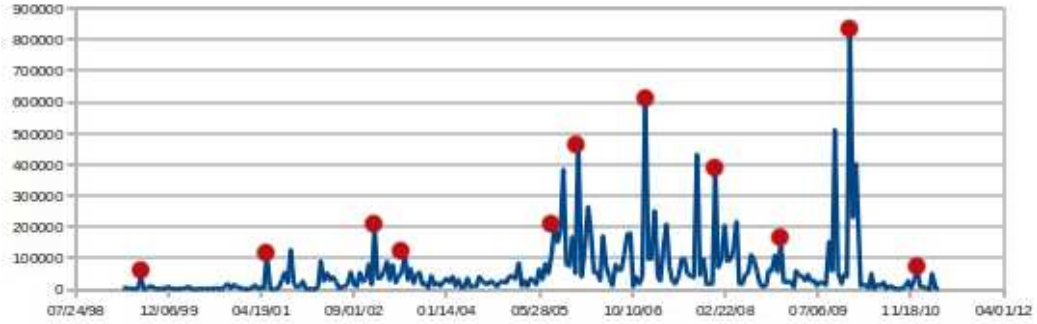
2. `http://php.net/releases/index.php`




(a) Evolution of the number of license changes excluding the introduction of license statements to newly created files.




(b) Evolution of the number of license changes including the introduction of license statement to newly created files.




(c) SLOC evolution.

Figure 5.3 Evolution of the SLOC and license changes over time in PHP.

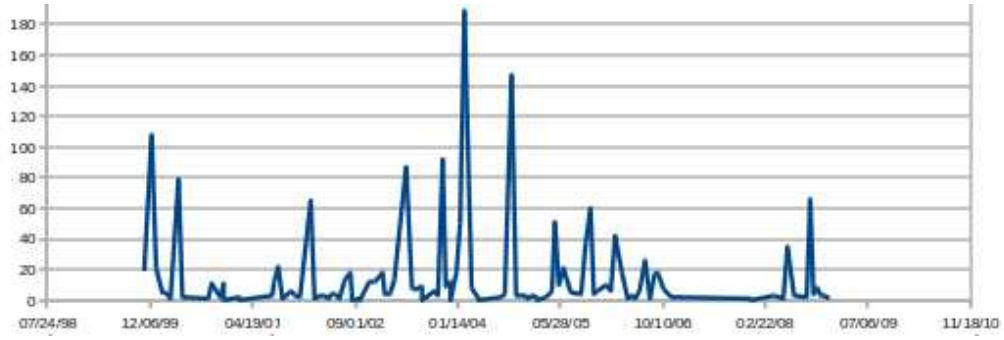
**In XercesJ,** We can see several red-dotted peaks in license statement changes (see Figure 5.4(b)), for example for October 2006, for which we analysed the change log comments and find that there wasa major *: "Update to the latest ASF license header*[3] (ASF stands for the Apache Software Foundation). We also find some comments in the mailing lists that illustrate this change[4, 5, 6], which seems to be an organized change of license statements.

These peaks do not have corresponding peaks in SLOC (see Figures 5.4(b) and 5.4(c)), since they only involve changes to license statement (SLOC does not count license statement). Instead, the changes are performed in a calm period without regular code changes by one committer ("mrglavas"). In fact, this committer only becomes active around the period of the license changes (period 2). Before this period (period 1), many small license statement changes were performed by different developers. For example, on 2001-09-12, "sandygao" changed a license statement by adding missing terms and the log message: "*Forgot to put license information in.*".

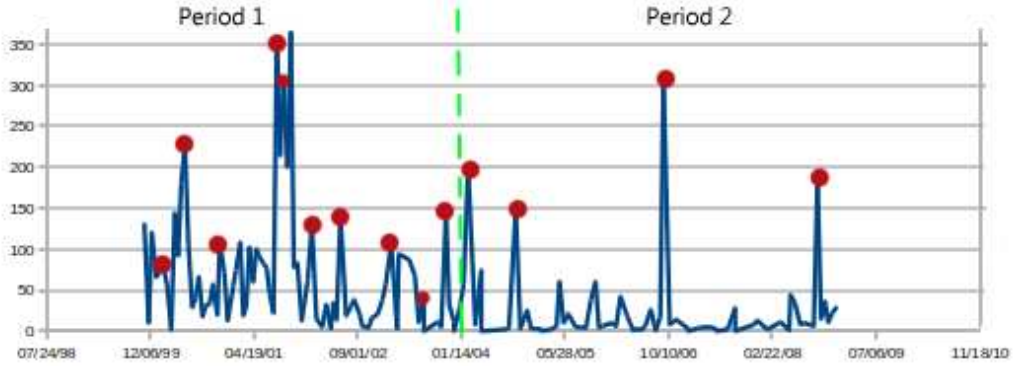
We observe some red-dotted peaks in Figure 5.4(a) corresponding to red-dotted peaks in Figure 5.4(b)). These peaks also correspond to peaks in SLOC evolution (Figure 5.4(c)). We can explain these by two type of license statement changes: (1) the introduction of licenses to existing files due to a missing license and, (2) the addition of new contributors while implementing new functionality. The peaks that exist only on Figure 5.4(b) are explained by the addition of licenses to newly created files.

all URL should be (or not) in textit

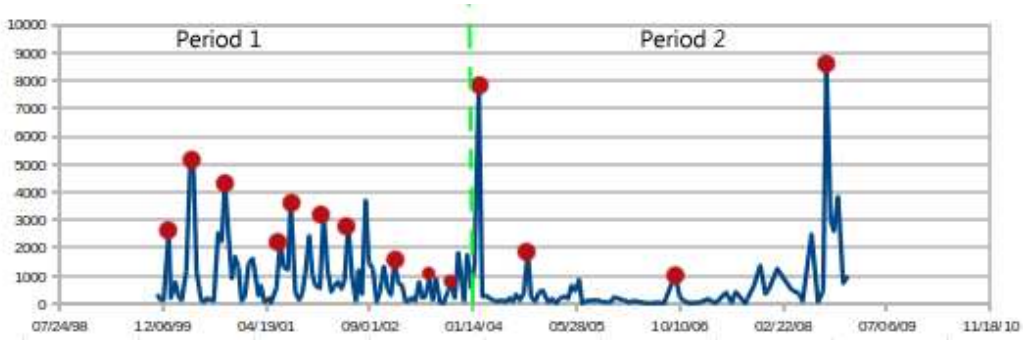
3. http://www.apache.org/legal/src-headers.html
4. http://goo.gl/UPbVc
5. http://goo.gl/Bb7qh
6. http://goo.gl/yTJUP




(a) Evolution of the number of license changes excluding the introduction of licenses to newly created files.




(b) Evolution of the number of license changes including the introduction of license statement to newly created files.




(c) SLOC evolution.

Figure 5.4 Evolution of the SLOC and license statement changes over time in XercesJ. (Red dots represent peaks, where as the green seperate two periods)

### 5.1.2 *RQ2: What types of license changes are performed?*

We found three main types of license statement changes: license type change, license version change and contributor addition. Their popularity is different from one project to the other. It seems to depend on each project's guidelines or culture towards software licenses. We found also that the cross-correlation between license type change or license version change with SLOC evolution is higher that one found in RQ1 when all type of license changes are mixed together.

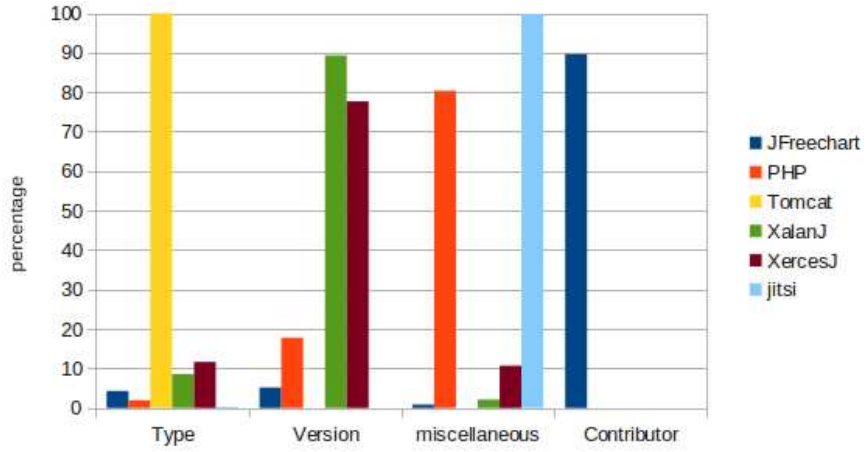



Figure 5.5 Number of license statement changes per type.

The qualitative study of RQ1 allowed us to identify the most popular types of license statement changes:

**Addition of contributors:** The license statement contains a list of names of all contributors who have developed the file. This list is updated by adding the name of a new contributor if (s)he helped to add a functionality or fix a bug. For example, in Nov $13^{rd}$ 2003 Tim Bardzil is added as a contributor in the file org/jfree/chart/renderer/category/BoxAndWhiskerRenderer.java because he added drawHorizontalItem() method.

**Updating the version of the license:** The version number of a license is the unique identifier attributed to a particular version of a license. A license version number is generally assigned in increasing order and corresponds to new features in the license. For example, PHP updated from PHP license $v2.01$ to PHP license $v2.02$ on May $18^{th}$, 2000.

**Change of the license type:** A project switches from a license to another for some reason, such as to be compatible with other software. For example, PHP changed the license of php4/main/output.c from php License V3.01 to LGPLv2+.

**Miscellaneous changes:** These are the remaining changes, which are smaller in nature and hence harder to identify automatically. Most of them are buried inside unmatched sentence changes, *i.e.*, those sentences that Ninka cannot match with the sentences of a known license, because they typically are due to customization of license text.

The histogram in Figure 5.5 shows the distribution of license statement change types per system. The cross-correlation between license statement changes and SLOC changes per type of license statement change are available in the annexe. We find the following:

**JFreeChart:** Almost all license statement change types in JFreechart are contributor changes. This confirms what we observed manually in the qualitative study of RQ1. The cross-correlation value of RQ1 is dominated by this kind of change.

**PHP:** The most popular kind of change are by far the miscellaneous changes, followed by license version changes and the license type changes. The cross-correlation is high for miscellaneous sentences (close to 1), while the cross-correlation of license type change and license version change is near 60%.

The majority of changes belong to the miscellaneous category, because licenses in PHP files do not include the full license text. Instead, they only contain a short summary for the full license (to avoid cloning the full license everywhere) and refer to the file $php/php - src/trunk/LICENSE$. Hence, Ninka is not able to detect the exact name of the license. To refine our analysis, we mined to the unmatched sentences for more detailed information. We found that the unmatched sentence tokens include the actual name of the licenses and their version number in the url to the license text. By parsing these links, we found out that all changes classified as miscellanous either correspond to license version changes or license type changes.

**Tomcat:** Although all Tomcat's license statement changes are classified as "type change", these changes mainly correspond to the addition of the apache clause [7] and a link to the integral apache license text, and hence are not really license type changes. The cross-correlation increases until 55% if all change kinds are seperated contrary to RQ1 (license type change and initial addition of license to a file –this type of change is not considered here).

**XalanJ:** About 90% of the license statement changes are license version changes and 9% are license type changes seperately changes. We computed the cross-correlation for these types of changes. We found that the cross-correlation between either license type or version changes with SLOC evolution is almost 1, which is much higher than the global cross-correlation from RQ1.

**XercesJ:** The license type and version changes are the most frequent changes. The cross-correlation between license type changes and SLOC evolution (reaching 70%) is much higher than the one between all license statement changes and SLOC evolution of RQ1 (reaching 20%), The same is found for version license version change. Thus, version and type changes co-occur often with large code changes.

7. Right and its conditions

**Jitsi:** There is just one license type change from GPLv2 to LGPL. The remaining changes are miscellaneous changes. Hence, we did not obtain a higher cross-correlation than the cross-correlation in RQ1, because Ninka did not provide an accurate classification of change. The cross-correlation is near to zero but reaches 65% for one lag of time.

We mined the unmatched sentences of Ninka output to improve the classification. Contrary to PHP, this mining did not provide license-related data, but rather license-unrelated code comments (*i.e.*, false positives of Ninka).

We did not present the result of Rhino in this RQ due to the low number of changes per type. So, the cross-correlation is not significant in this case.

### 5.1.3 *RQ3: Who performs license changes?*

Table 5.1 presents the number of committers involved in license statement changes. We see that 24 committers out of 28 (86%) for XercesJ and 2 out of 2 (100%) for JFreeChart are involved in license statement changes. In contrast to XercesJ, only 10 committers out of 222 (4.50%) of PHP are involved in license changes.

| | XercesJ | JFreeChart | PHP |
|---|---|---|---|
| Total # of found license statement changes | 3116 | 162774 | 27 |
| # (percentage) of committers involved | 24 (86%) | 100 (%) | 10 (4.50%) |

Table 5.1 Overview of the license statement changes and the committers involved.

| XercesJ | | JFreeChart | | PHP | |
|---|---|---|---|---|---|
| ID | # of license statement changes | ID | # of license statement changes | ID | # of license statement changes |
| mrglavas | 1536 (49%) | mungaly | 849 (99.53%) | zeev | 8 (29.62%) |
| lehors | 275 (9%) | taqua | 4 (0.47%) | ssb | 5 (18.51%) |
| elena | 247 (8%) | - | - | andi | 5 (18.51%) |
| no author | 188 (6%) | - | - | - | - |
| andyc | 178 (6%) | - | - | - | - |
| sandygao | 178 (6%) | - | - | - | - |
| arkin | 110 (4%) | - | - | - | - |
| Total top 7 | 2,712 | Total top 7 | 853 | Total top 7 | 18 |
| Total license statement changes | 3,116 | Total license statement changes | 853 | Total license statement changes | 27 |
| % license statement changes top 7 | 87% | % license statement changes top 7 | 100% | % license statement changes top 47 | 66.66 |

Table 5.2 Top seven committers involved in license statement changes. in parentheses we show the % of licenses changed per committer.

| Jitsi | | Tomcat | |
|---|---|---|---|
| ID | # of license changes | ID | # of license changes |
| yanas | 822 (25.60%) | markt | 741 (31.89%) |
| lubomir_m | 820 (25.54%) | mturk | 571 (24.58%) |
| damencho | 506 (15.76%) | kkolinko | 406 (17.47%) |
| s_vincent | 442 (13.76%) | remm | 404 (17.39%) |
| emcho | 339 (10.56%) | fhanik | 144 (6.19%) |
| wernerd | 143 (4.45%) | rjung | 38 (1.6%) |
| ibauersachs | 38 (1.18%) | kfujino | 7 (0.3%) |
| Total top 7 | 3.110 | Total top 7 | 2311 |
| Total license changes | 3.210 | Total License changes | 2323 |
| % license changes top 7 | 96.88% | % License changes top 7 | 99.48% |

Table 5.3 Top seven committers involved in license changes. Values in parentheses indicate the percentages of licenses changed per committer.

| XalanJ | | Rhino | |
|---|---|---|---|
| ID | # of license changes | ID | # of license changes |
| minchau | 1593 (50.14%) | nboyd | 326 (27.76%) |
| mkwan | 488 (15.36%) | szegedia | 269 (22.91%) |
| jycli | 320 (10.07%) | igor | 205 (17.46%) |
| sboag | 192 (6.04%) | gerv | 126 (10.73%) |
| zongaro | 154 (4.84%) | inonit | 100 (8.51%) |
| mcnamara | 148 (4.65%) | noris | 86 (7.32%) |
| santiagopg | 61 (1.92%) | hannes | 34 (2.89%) |
| Total top 7 | 2956 | Total top 7 | 1146 |
| Total License changes | 3177 | Total License changes | 1174 |
| % License changes top 7 | 93.04 | % License changes top 7 | 97.61 |

Table 5.4 Top seven committers involved in license changes. Values in parentheses indicate the percentages of licenses changed per committer.

| XercesJ | | JFreeChart | | PHP | |
|---|---|---|---|---|---|
| ID | # of changes | ID | # of changes | ID | # of changes |
| mrglavas | 4070 (29.62%) | mungaby | 3446 (99.94%) | zeev | 4655 (9.19%) |
| elena | 2253 (16.39%) | taqua | 2 (0.058%) | helly | 3502 (6.91%) |
| no author | 1841 (13.40%) | - | - | iliaa | 2999 (5.92%) |
| lehors | 1583 (11.52%) | - | - | dmitry | 2799 (5.53%) |
| neilg | 1234 (8.98%) | - | - | andi | 2792 (5.51%) |
| jeffreyr | 503 (3.66%) | - | - | sebastian | 2752 (5.43%) |
| andyc | 425 (3.09%) | - | - | sniper | 2145 (5.23%) |
| Total top 7 | 11909 | Total top 7 | 3448 | Total top 7 | 18 |
| % changes top 7 | 86.68% | % changes top 2 | 100% | % changes top 7 | 42.76 |

Table 5.5 The most active committers. Values in parentheses indicate the percentages of files changed per committer.

| Jitsi | | Tomcat | |
|---|---|---|---|
| ID | # of changes | ID | # of changes |
| yanas | 4992 (36.01%) | markt | 1629 (46.51%) |
| lubomir_m | 2753 (19.86%) | kkolinko | 582 (16.61%) |
| emcho | 2385 (17.20%) | remm | 566 (5.92%) |
| s_vincent | 1945 (14.03%) | fhanik | 389 (11.10%) |
| damencho | 772 (5.56%) | mturk | 122 (3.48%) |
| wernerd | 358 (2.58%) | rjung | 92 (2.62%) |
| sympho | 156 (1.12%) | pero | 28 (0.79%) |
| Total top 7 | 13361 | Total top 7 | 3408 |
| % changes top 7 | 96.38% | % changes top 7 | 97.3% |

Table 5.6 The most active committers. Values in parentheses indicate the percentages of files changed per committer.

| XalanJ | | Rhino | |
|---|---|---|---|
| ID | # of changes | ID | # of changes |
| sboag | 1738 (26.56%) | igor | 2009 (45.85%) |
| mkwan | 967 (14.77%) | nboyd | 1164 (26.56%) |
| norten | 796 (12.16%) | norris | 286 (6.52%) |
| minchau | 512 (7.82%) | gerv | 181 (4.13%) |
| santiagopg | 383 (5.85%) | nboyd | 168 (3.83%) |
| mmidy | 367 (5.60%) | inonit | 110 (2.51%) |
| minchau | 343 (5.24%) | szegedia | 34 (2.89%) |
| Total top 7 | 5106 | Total top 7 | 4028 |
| % changes top 7 | 78.03 | % changes top 7 | 91.94 |

Table 5.7 The most active committers. Values in parentheses indicate the percentages of files changed per committer.

Table 5.2 shows the list of the top seven committers involved in license statement changes. In XercesJ, 7 committers out of 28 performed 87% of the license statement changes while in JFreechart, 2 out of 2 committers performed 100% of all license statement changes.

Especially for XercesJ, most of the license statement changes have been performed by a small subset of the committers. As can be seen in the tables, the percentages of commits related to license statement changes is more or less similar for all XercesJ committers in the top seven, *i.e.*, ranging between 4% and 9%. One committer has a higher percentage of changes (mrglavas), with 49% of commits involving a license statement change. In JFreeChart, 1 committer performed 99.53% of license statement changes, while the other one hardly made any change.

In PHP, to extract the committers who changed licenses, we counted just the number of changes in the file $php/php-src/trunk/LICENSE$ and not the numbers of source code files for which licenses were changed, given PHP's specific license convention. Thus, the number of license statement changes in PHP is much lower than the one in JFreeChart and XercesJ. However, the results show the same trend as for JFreechart and XercesJ: a minority of committers performed the majority of license changes. Three committers performed 66.66% of all license statement changes.

In order to better understand the role of license statement change committers, Table 5.5 identifies the most active committers based on the number of commits (any commit that involves SLOC change) for JFreeChart, PHP, and XercesJ. We find that many committers in the top seven for license statement changes are also active committers. In XercesJ, the top seven active committers who also perform license statement changes are: "mrglavas", "lehors", '"elena", "no author", "andyc" (5 out of 7). In JFreeChart, the committer who commits the majority of license statement changes (99.43%) is also the most active one (99.94%). In PHP, 2 top committers out of the 3 that commit license statement changes are also the most active.

We found similar results in the remaining systems as shown in the Table 5.3, *i.e.*, Jitsi, Rhino, Tomcat, and XalanJ, where the top seven committers for license statement changes performs respectively 96.88%, 99.48%, 93.04%, and 97.61% of the source code changes. Thus, a minority of committers perform the majority of license statement changes. Moreover, these committers are the most active developers.

To summarize, the most active developers accepting changes to license statement are the main contributors to software projects. This seems reasonable, since they (1)

often are amongst the leaders of a project, having the actual power to decide about license changes, and (2) presumably have a very good insight into and experience with the software system, being able to clearly understand the repercussions of software license changes. For example, "mrglavas" in XercesJ is the primary contributor to the Apache Xerces2 project since 2003. "Zeev" in PHP is a PHP developer and co-founder of Zend Technologies. Together with a fellow student "andi" (also an important committer), he created PHP3 in 1997.

## 5.2 Discussions and Threats to validity

In previous work, researchers studied license statement changes independently from software maintenance tasks. In our work, we study license statement evolution in the context of source code evolution. Based on our findings in RQ1 (no systematic large-scale license changes and dispersed license statements), we can suggest improvements to the license statement change process. First, there is a need for tools that help track licenses and license statement changes to ensure systematic changes of all the licenses of files consistently to the wanted license if the team decided so. For example, this tool should allow visualising licenses at different levels of granularity, from files to systems (some package has different license of the system license like zend package in PHP). Moreover, during a change period, it could be used to automatically update files to their "future license". After the change is performed, this tool should check that the license statement changes are propagated throughout the system (consistency check), the current licenses are not violated in any way and if the right persons are changing the licenses (we observed some errors in license statement changes like the one zend package). There are quite some challenges involved with developing such a tool, in particular the textual nature of license statements, which encourages customizations. Furthermore, the fact that different change types do not have the same popularity or even formatting style across all projects, suggests that this tool must be adapted to the specific culture of license statement changes in a particular project.

Second, instead of tool support, one could change the concept of "license statement" to be more effective. This is basically what we saw in PHP, where instead of having license statements that are (possibly customized) clones of the original license text, the base license text is centralized. Less license statement changes occurred in

PHP compared to the other projects, yet more research on systems with a similar mechanism is needed to determine whether the low number of changes is really due to the centralized concept of license statements or due to some other factor.

For the two alternative, we need necessarily a meta-model that presents entities required for analysis. Previous work established models that are centralised on license type, right, condition, yet they did not consider other that we needed for more effective analysis as autor, system architecture. Our study shows the importance to include other information in the models, for example it is important to know who is the committer that changed the license and the contributor of the file covered by a license. We already designed an initial model that could be refined to include possibly more informations and add layer to help in license evolution managment.

Our study has some threats to validity, which we now discuss in more detail (Wohlin *et al.* (2000)).

*Construct validity:*

Construct validity concerns the relation between theory and observations. The later can be due to our measurements, *i.e.*, the way we extracted licenses and identified their changes. We extracted licenses using an existing license identification tool, Ninka German *et al.* (2010b). Although Ninka has a high accuracy, it also outputs unmatched sentences in licenses (i.e. sentences that it cannot parse. Although we manually scanned these sentences for patterns, there is a risk that the unmatched sentences might change some of the results. Moreover, Ninka does not detect the copyright year. Thus, to answer our qualitative study, we mined change logs using `grep` for specific expressions like: "Bump year", "update year", "year++", "update copyright year", "copyright year", and others. Consequently, there is a risk that we did not detect all copyright year changes.

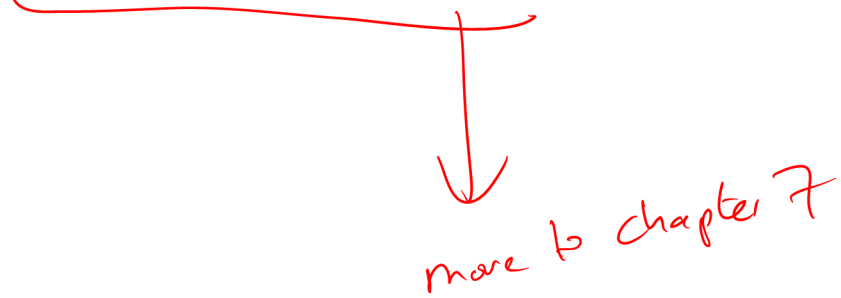
*Internal and Conclusion Validity:*

The internal validity of a study is the extent to which a treatment impacts the dependent variable. Conclusion validity threats concern the relation between the treatment and the outcome. Threats to internal validity do not affect this study, being an exploratory study Yin (2002). Conclusion validity is not threatened because we used cross-correlations and made sure that the conditions for their application held.

*External Validity:*

The external validity of a study is the extent to which we can generalise its re-

sults. The main threat to the external validity of our study relates to the analysed systems (i.e., four medium-sized systems (JFreeChart, Rhino, XalanJ, and, XercesJ), and three large system (PHP, Tomcat and, Jitsi). All of these are open source, but from different domains and with four different licenses: Apache, LGPL, MPL/GPL, and PHP. Future work includes replicating our study on more systems, licensed under other licenses to confirm our results.



# Chapter 6

# PRELIMINARY FOR LICENSE VIOLATION DETECTION TOOL

In this chapter, we present a preliminary step for a tool that helps to avoid license inconsistencies in a system.

## 6.1 Tool Architecture Overview

The result of the license statement evolution study presented in the chapter shows there is need of tool to manage license statement changes. This tool must ensure a systematic changes of all the licenses of files consistently to the wanted license, and also permit to make developers aware of the constraints imposed by the used licenses. The meta-model proposed in the chapter could be extended by adding another layer to represent license constraints in order to be able to check license constraints for a given instance. In fact, the tool will process in two main steps, first step is to extract all the required system data according our meta-model, the second step is to transform the constraints and license terms to rules using a formal language using the meta-model entities, finally check if the rules are respected on the system meta-model instance (see Figure 6.1).

## 6.2 Example of GPLv3 License Rules

In this section, we present some example of GPLv3[1] terms, that we tried to formalize using logic expression using the entities that we defined in our meta-model. We extracted the terms of GPLv3 license. Then, we transformed them into rules using the entities defined in our meta-model.

1. http://www.gnu.org/copyleft/gpl.html

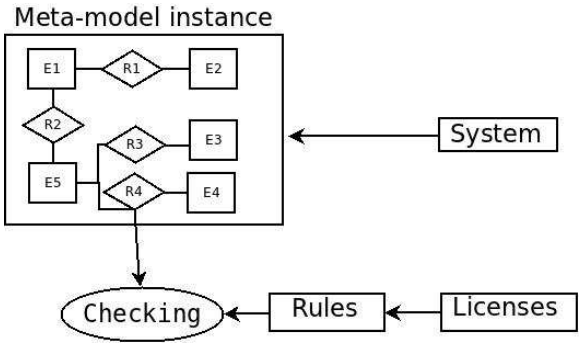



Figure 6.1 License constraints checking.

## Rule 1

"If you distribute copies of a program licensed under GPLv3, you must pass to the recipients the same freedom that you received. You must be sure that they receive or can get the source code. And you must show them this terms."

$if L(S) = GPLv3 \wedge distribute(S) \Rightarrow show(S, T(L(S))) \wedge accessible(Source(S))$

List of fact used :

- $distribute$ : distribute a copies of a system S
- $show(S, T(L(S))$ : show the terms of the system license
- $accessible(Source(S))$ : make the source code of S accessible

## Rule 2

"The GPL requires that modified versions be marked as changed (so that their problems will not be attributed erroneously to the author)"

$if derivative(P, ConnType(S_N, s)) \wedge L(P) = GPLv3 \Rightarrow L(S) = GPLv3 \wedge contain(S, N(Modif))$

List of fact used :

- $contain(S, N(Modif))$ : S contain a Notice of modification

## Rule 3

"If you convey a program under GPLv3, an interactive users interface must show to the user: 1) displays an appropriate copyright notice, and 2)tells the user that there is no warranty for the work, that licensees may convey the work under this License, and how to view a copy of this License."

$if L(S) = GPLv3 \wedge convey(S)$
$\Rightarrow show(S, N(L)) \wedge show(S, N(NW)) \wedge show(S, N(R(L(S), Convey))) \wedge show(S, N(L(S)))$
List of fact used :
– $N(NW)$ : Notice of no warranty

## Rule 4

"The output from running a covered work is covered by this license only if the output, given its content, constitutes a covered work. (example of exception is the output of gcc, compiled source code, is not covered by GPL)"

$if L(S) = GPLv3 \Rightarrow L(Output(S)) = GPLv3$

List of fact used :
– $Output(S)$ : output from running a system S

## Rule 5

"You may convey verbatim copies of the program's source code as you receive it, in any medium provided that you publish in each copy an appropriate copyright notice; keep intact all notices stating that this license and any non permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this license along with the Program."

$if L(S) = GPLv3 \wedge convey(S) \Rightarrow W(S) = W(copy(S)) \wedge contain(copy(S), Notice(L(S))) \wedge NW(S) = NW(copy(S)) \wedge Exception(W) \wedge Exception(PreservationSpecNotice) \wedge Exception(ProhibitMisRepresentOrigin) \wedge Exception(LimitPub) \wedge Exception(Decline) \wedge Exception(requireIndeminification)$

List of fact used :
– $Exception(W)$ : exception of the warranty.
– $Exception(PreservationSpecNotice)$ : exception of requiring preservation of specified reasonable legal notices or author attributions.
– $Exception(ProhibitMisRepresentOrigin)$ : exception of prohibiting misrepresentation of the origin of that material.
– $Exception(LimitPub)$ : Limiting the use for publicity purposes of names of licensors or authors of the material.
– $Exception(Decline)$ : exception of declining to grant rights under trademark law for use of some trade names, trademarks, or service marks.

– $Exception(requireIndeminification)$ exception of requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it.

## Rule 6

"You may convey a work based on the Program or a modification of the Program in the form of source code under the terms of rule 4 and under these conditions: a) contains notice that states that you modified it and indicates a relevant dates, b) the work must contain notice stating that is released under This license (GPLv3) and any conditions added under section 7. This requirement modifies the requirement in Rule 5 to keep intact all the notices.c) You must license the work as whole under this License to anyone comes into possession. d)If P contains user interface $\Rightarrow$ the user interface of the program must display Appropriate Legal Notice."

$if L(S) = GPLv3 \wedge Derivative(P, S, I(P, S)) \wedge convey(S) \Rightarrow contain(copy(P), N(Modif)) \wedge contain(copy(P), N(L(S))) \wedge (copy(P).contain(UI) \Rightarrow show(copy(P), N(L))) \wedge Exception(W) \wedge Exception(PreservationSpecNotice) \wedge Exception(ProhibitMisRepresentOrigin) \wedge Exception(LimitPub) \wedge Exception(Decline) \wedge Exception(requireIndeminification)$

List of fact used :

– $N(Modif)$ : Notice which indicates that the program is modified version of the original one

## Rule 7

"The combination of a covered work in a compilation of independent work doesn't cause this license to apply to the other parts of the aggregate."

$if L(S) = GPLv3 \wedge !Derivative(P, S, ConnType(P, S)) \Rightarrow \forall f \in S, L(f) = anyLicense$

# Chapter 7

# CONCLUSION

Several issues of license due to license evolution suggest that license changes could have negative impacts. Thus, we think that license evolution worth to be studied to help in automatic license changes tracking, due to the size of systems that prevent manual checking. Existing approaches for license statement change analysis do not focus on the relation between license statement changes and the software development cycle, *i.e.*, the co-evolution between licenses and source code. We think it is important to relate source code evolution and license evolution to find the license evolution is a systematic changes or depends on software evolution and also to find which type of developers are changing licenses, *i.e.*, the same modifying the source code. We think that it is important as first step, to propose a system meta-model that assemble pertinent informations. As a second step, we want to understand if license management is correlated with source code changes. Knowing how and when licenses change, we could propose a methodology to improve the process of license management to help developers in changing licenses without introducing inconsistencies and violating licenses using the outcome of our study and information from the meta-model.

We began by doing a litterature review about previous system meta-model for license analysis, studies around license analysis, and license terms, ...etc to gather license data that must be presented in a meta-model. Then, we identified relation between them and defined each element in the meta-model. After that, to study source code and license co-evolution, we used our system meta-model to identify which data we must track and stored them in Ibdoos database using within meta-model. Using this data, we performed a quantitative and a qualitative study on seven systems, and we found that licenses are changing frequently as other software artefacts are changing. However, these changes to a large degree seem independent from source code changes, *i.e.*, they are not necessarily aligned with massive code changes. Furthermore, we distinguished three main types of license statement changes: license

type change, license version change and contributor addition. The popularity of these change types is not uniform across all projects, but seems to depend on each project's guidelines or culture towards software licenses. Hence, different strategies are required to manage license evolution. Finally, we found that the committers that change the licenses are also the most active committers to the projects and the main contributors in some projects. This means that they have a leadership role in the project, as well as a good insight into the system.

Based on our findings, we believe that to improve the license statement change process, practitioners either need a dedicated methodology and tools to support them, or need to rethink the concept of license statements. This should help ensure that license statement changes do not introduce inconsistencies, and hence prevent legal or commercial damage to the organization.

As future work, we propose to extend our automatic approach to track license evolution by adding license compatiblity checking. As we did in our preliminary study in Chapter 6, we could formalize rules of each license. Then, we could check their rules are verified in the respected in the concerned system.

reuse the bullet points and the boxes p. 11-13.

# Références

ALSPAUGH, T. A., ASUNCION, H. U. et SCACCHI, W. (2009). Intellectual property rights requirements for heterogeneously-licensed systems. *RE '09: Proceedings of the 2009 17th IEEE International Requirements Engineering Conference, RE*. IEEE Computer Society, Washington, DC, USA, 24–33.

BASILI, V. R. et WEISS, D. M. (1984). A methodology for collecting valid software engineering data. *IEEE Trans. Software Eng.*, 10, 728–738.

CAPILUPPI, A., LAGO, P. et MORISIO, M. (2003). Characteristics of open source projects. *CSMR '03: Proceedings of the Seventh European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, Washington, DC, USA, 317.

CORDY, J. R. et ROY, C. K. (2011). Debcheck: Efficeient checking for open source code clones in software systems. *Proceedings of the International Conference on Program Comprehension, ICPC 2011*. IEEE Computer Society.

DA CRUZ, D. C. (2008). *Methods and techniques to analyze multi-level code to explore software components*. Thèse de doctorat, Universidade do minho.

DI PENTA, M., GERMAN, D. M., GUÉHÉNEUC, Y.-G. et ANTONIOL, G. (2010). An exploratory study of the evolution of software licensing. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*. ACM, New York, NY, USA, ICSE '10, 145–154.

ESHKEVARI, L. M., ARNAOUDOVA, V., PENTA, M. D., OLIVETO, R., GUÉHÉNEUC, Y.-G. et ANTONIOL, G. (2011). An exploratory study of identifier renamings. *MSR*. 33–42.

GERMAN, D. M. et HASSAN, A. E. (2009). License integration patterns: Addressing license mismatches in component-based development. *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, Washington, DC, USA, 188–198.

GERMAN, D. M., MANABE, Y. et INOUE, K. (2010a). A sentence-matching method for automatic license identification of source code files. *Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM, New York, NY, USA, ASE '10, 437–446.

GERMAN, D. M., PENTA, M. D. et DAVIES, J. (2010b). Understanding and auditing the licensing of open source software distributions. *ICPC '10: Proceedings of the 18th International Conference on Program Comprehension*. IEEE Computer Society, Los Alamitos, CA, USA, vol. 0, 84–93.

GOBEILLE, R. (2008). The fossology project. *Proceedings of the 2008 international working conference on Mining software repositories*. ACM, New York, NY, USA, MSR '08, 47–50.

HAYES, J. H., MOHAMED, N. et GAO, T. H. (2003). Observe-mine-adopt (oma): an agile way to enhance software maintainability. *Journal of Software Maintenance*, 15, 297–323.

HAYES, J. H., PATEL, S. C. et ZHAO, L. (2004). A metrics-based software maintenance effort model. *Software Maintenance and Reengineering, European Conference on*, 0, 254.

HEMEL, A., KALLEBERG, K. T., VERMAAS, R. et DOLSTRA, E. (2011). Finding software license violations through binary code clone detection. *Proceedings of the 8th international conference on Mining software repositories*. ACM, MSR '11, 63–72.

HINDLE, A., GERMAN, D. M. et HOLT, R. (2008). What do large commits tell us?: a taxonomical study of large commits. *Proceedings of the 2008 international working conference on Mining software repositories*. ACM, New York, NY, USA, MSR '08, 99–108.

KENMEI, B., ANTONIOL, G. et DI PENTA, M. (2008). Trend analysis and issue prediction in large-scale open source systems. *Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, Washington, DC, USA, 73–82.

KULLBACH, B., WINTER, A., DAHM, P. et EBERT, J. (1998). Program comprehension in multi-language systems. *WCRE '98: Proceedings of the Working Conference on Reverse Engineering (WCRE'98)*. IEEE Computer Society, Washington, DC, USA, 135.

MANABE, Y., HAYASE, Y. et INOUE, K. (2010). Evolutional analysis of licenses in foss. *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*. ACM, New York, NY, USA, IWPSE-EVOL '10, 83–87.

PENTA, M. D. et GERMAN, D. M. (2009). Who are source code contributors and how do they change. *Proceedings of the 16th Working Conference on Reverse Engineering, WCRE 2009*. IEEE Computer Society, 13–16.

ROSEN, L. (2004). *Open Source Licensing Software Freedom and Intellectual Property Law*. Prentice Hall.

TUUNANEN, T., KOSKINEN, J. et KÄRKKÄINEN, T. (2009). Automated software license analysis. *Automated Software Eng.*, 16, 455–490.

WOHLIN, C., RUNESON, P., HÖST, M., OHLSSON, M. C., REGNELL, B. et WESSLÉN, A. (2000). *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA.

YIN, R. K. (2002). *Case Study Research: Design and Methods*. Sage Publications, Inc, third edition édition.