

QualiPSo
Quality Platform for Open Source Software
IST- FP6-IP-034763



Deliverable A1.D2.1.4
Report on the proposed IPR tracking methodology

Luc Grateau (INRIA)
Magali Fitzgibbon (INRIA)
Guillaume Rousseau (INRIA)
Stéphane Dalmas (INRIA)

Due date of deliverable: 30/10/2009

Actual submission date: 16/12/2009

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 License.

To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

This work is partially funded by EU under the grant of IST-FP6-034763.

Change History

Version	Date	Author (Partner)	Description
0.1	01/10/2008	Guillaume Rousseau (INRIA)	Initial version, first version of the Palette annex 2 questionnaire
0.2	01/11/2008	Stéphane Dalmas (INRIA)	Palette annex 2 Questionnaire update
0.3	02/03/08	Guillaume Rousseau (INRIA)	Palette Questionnaire feed back and analysis. Contribution to Palette Open Source Strategy (including proposal for IPR Tracking set up)
0.4	24/06/2008	Magali Fitzgibbon (INRIA), Luc Grateau (INRIA), Guillaume Rousseau (INRIA)	Definition and integration of the Legal Situation
0.5	19/12/2008	Magali Fitzgibbon (INRIA), Luc Grateau (INRIA), Guillaume Rousseau (INRIA)	1 st Full Draft
1	26/01/2009	Magali Fitzgibbon (INRIA), Luc Grateau (INRIA), Guillaume Rousseau (INRIA)	1 st version of the deliverable
2	16/12/2009	Magali Fitzgibbon (INRIA), Luc Grateau (INRIA), Guillaume Rousseau (INRIA)	Final version of the deliverable

EXECUTIVE SUMMARY

Improving the legal quality and legal safety of components or component-based software released under open source licenses, is globally considered as a key issue for the software industry as well as for open source software communities, editors or/and users.

The QualiPSo project is addressing these questions in the “Legal Issues” Activity, contributing to enhance the awareness of the open source ecosystem to these issues. The present deliverable is aiming at providing a general, conceptual framework for Intellectual Property Rights Tracking (IPRT) management of Component-Based and Collaboratively Developed software (CBCDS). However, the methodology proposed in this document is as generic as possible and should apply not only to open source Software or components, but also to hybrid software (proprietary software containing components under permissive licenses) or even to fully proprietary software (with no non-permissive components such as “Copyleft” components) or to software exploited in a Software as a Service –SaaS model.

The underlying model in which the IPRT methodology is proposed consists in considering the coupling between a “licensing in” process – reuse of preexisting components - and an exploitation scheme of the CBCDS that is most often a “licensing out” – distribution - process.

The “legal roadmap” is the name given to this “licensing out” / “licensing in” coupling formalization.

The goal and objective of an IPRT methodology or framework are presented in this report as well as the main concept for IPRT, the Legal Status of Software. The report proposes an IPRT methodology and describes the key elements that have an impact on the legal status of the software. The developers have a key role to reach a Legal Status that is compliant with exploitation intentions or model of the editor, as the Legal Status is formed through their actions and with this respect, so called “good IPR development practices” are of tremendous importance.

IPRT is a tool to make sure this “legal roadmap” is followed during the development process and more generally to make sure that the Software’s legal status is compatible with the exploitation scheme.

A dedicated audit team is set up to plan and implement the six steps IPRT framework to a given development situation. Development team provides a detailed description of the software (1.) Goals and objectives of the audit are defined, as well as a “licensing in” and “licensing out policies” (2.) Legal status is determined by comparing “perceived” legal status – based on a questionnaire (Annex 1) – and “determined” legal status – based on a code mining tool such as FOSSology™ (3.) Problem Identification and Risk Evaluation is operated (4.) Critical problem solving is performed (5.) Residual Risk (if any) is covered by insurance before dissemination/distribution (6.)

A use case is given to illustrate the implementation of the methodology done at INRIA.

The associated tool box to establish the Legal status of CBCDS is provided.

Already existing tools are helping to gain productivity, such as the FOSSology open source license checker. However, strong standardisation needs were encountered (name of the licenses, structure and content of the headers, etc...) to reduce the time spent on and costs of CBS’s analysis of tools results. New tools are to be developed, to avoid bottleneck phases of the audit process (mapping license checker’s results to functional zones for example.)

Document Information

IST Project Number	FP6 – 034763	Acronym	QualiPSo
Full title	Quality Platform for Open Source Software		
Project URL	http://www.qualipso.org		
Document URL			
EU Project officer	Michel Lacroix		

Deliverable	Number	1	Title	Report on the proposed IPR tracking methodology
Work package	Number	1.4	Title	IPR Tracking
Activity	Number	1	Title	Legal Issues

Date of delivery	Contractual	30/10/2009	Actual	16/12/2009
Status	Version 2.0, dated 16/12/2009		final <input type="checkbox"/>	
Nature	Report <input type="checkbox"/> Demonstrator <input type="checkbox"/> Other <input checked="" type="checkbox"/>			
Dissemination Level	Public <input checked="" type="checkbox"/> Consortium <input type="checkbox"/>			
Abstract (for dissemination)	Methodology to track rights and obligations during the whole development cycle, basic issues of trust in contributors and contributions, integration in development environments, prototype software			
Keywords	IPR Tracking			

Authors (Partner)	Luc Gateau (INRIA) Magali Fitzgibbon (INRIA) Guillaume Rousseau (INRIA) Stéphane Dalmas (INRIA)			
Responsible Author	Luc Gateau		Email	luc.gateau@inria.fr
	Partner	INRIA	Phone	+33 1 39 63 54 59

List of contents

EXECUTIVE SUMMARY	3
1 Introduction	7
1.1 A need to improve software legal Quality	7
1.2 Free software development life cycle, legal maturity, and industry or business trends	9
1.3 Legal dimension of the software development process and good practices	10
1.4 IPR Tracking models	14
1.5 Exploitation intentions and IPR Tracking	15
1.6 IPR Tracking QualiPSo deliverable objective and structure of the report	18
2 Project management and Legal Status	20
2.1 Implementing the IPRT Methodology: defining tools and processes	20
2.2 Introducing the notion of <u>Legal Status</u>	21
A. Position in the chain of Rights	22
B. Owner of Intellectual Property Rights (IPR).....	23
C. Legal conditions of exploitation of the software.....	23
D. Other enforceable IPR against software.....	24
E. Representation of the Legal Status	24
2.3 Intellectual Property Rights Tracking methodology	24
2.3.1 General.....	24
2.3.2 Scope of the IPRT process: what is analyzed?	25
3 Key IPRT issues during the software life cycle analyzed by the IPRT Methodology	27
3.1 External or exogenous components	27
3.2 Authorship and ownership	27
3.3 Contracts' provision tracking	28
4 Presentation of the <u>Audit module</u>	30
4.0 Overview	30
4.1 Step 1 : High Level Description of the Software	31
4.1.1 General Description at largest scale:	32
4.1.2 Description of functional zones:.....	33
4.1.3 Description of each functional zone:	33
4.2 Step 2 : Defining intentions	34
4.3 Step 3 : Determination of the Legal Status	35
4.3.1 "Perceived Legal Status" by the development and audit team.	35
4.3.2 Determined Legal Status.....	37
4.4 Step 4 : Problem Identification and Risk Evaluation	38
4.5 Step 5 : Solve Blocking/Critical Problem	40
Problem solving by developers.....	40
Problem solving with legal action.....	41
4.6 Step 6 : Insurance, Dissemination and IPR tracking	42
5 Conclusion	43
Annex I: Toolbox for legal status drafting	45
Legal Status Template	45
FLOWCHART	46
<u>Guidelines – Legal Situation Questionnaire</u>	51
Position in the chain of rights :.....	51
Owner of intellectual property rights:	52
Legal conditions of exploitation:	53
Other enforceable IPR against component's:.....	54

ANNEX 2: EU PALETTE CASE STUDY.....	60
Introduction:	60
IPRT and EU PALETTE Project:	60
<u>Step 1 : High level description of the PALETTE services and audit objectives</u>	<u>60</u>
<u>Step 2: Preparation of a questionnaire for “perceived legal status determination”</u>	<u>60</u>
<u>Step 3 Determination of “Perceived legal Status” for any PALETTE software and additional “FOSSology Scanned” Legal Status for Bay Fac Software</u>	<u>61</u>
<u>Step 4: Problem identification and Risk evaluation.</u>	<u>62</u>
<u>Step 5 : Solve Blocking problems</u>	<u>62</u>
Next phase for Palette projects:.....	62
Palette use case conclusion:	63

List of figures:

Figure 1: Free software lifecycle	9
Figure 2 : The CBCDS model from the legal issues perspective.....	11
Figure 3: The freedom matrix	12
Figure 4: The technical vs legal maturity diagram	14
Figure 5: IPR Tracking models	15
Figure 6: The exploitation schemes matrix.....	17
Figure 7: Methodology overview.....	30
Figure 8: XtremOS overall architecture	32
Figure 9: XtremOS layer description	32
Figure 10: High level Description of DIET’s functional zones	33
Figure 11 DIET Detailed High Level Description Functional zones.....	34
Figure 12: Step 3. Legal Status determination.....	35
Figure 13: DIET perceived legal Status.....	37
Figure 14: Step 4. Problem identification and risk evaluation.....	38
Figure 15: Blocking problems identification for DIET	39
Figure 16: Step 5. Solve Blocking/Critical Problem.....	40
Figure 17: Final legal situation of DIET after problem solving.	42
Figure 18: Template used to determine perceived Legal Status	45
Figure 19: Flowchart used to complete the Legal Status of DIET (Part 1)	46
Figure 20: Flowchart used to complete the Legal Status of DIET (Part 2)	47
Figure 21: PALETTE IPRT Process.....	64

1 Introduction

1.1 A need to improve software legal Quality

Component-based software (CBS) is somehow a pleonasm. Indeed, most software is nowadays component-based software, and the proportion of components released under an open source license to be used in CBS is expected to grow¹. Despite the increasing complexity of CBS, it is commonly expected and understood today that the technical maturity of open source based CBS could reach excellence and confidence (as already further exemplified by the use of open source components in proprietary software). However, this state of fact is paradoxically related with a growing uncertainty regarding the use of open source containing CBS. The collaborative development of software and the reuse of pre-existing components have legal consequences developers should be aware of and that open source communities or editors should master. Legal status of CBS, and especially CBS containing open source license based components is a concern for most software editors.

This lack of trust and confidence is partly due to the extreme diversity of licenses and other Intellectual property rights (IPR) attached to the available components together with the difficulty to identify these components in a large project. A case of concern for proprietary software editors or for hybrid software editors (editors having proprietary licensing of software incorporating open source components under permissive license such as GNU LGPL) is the fact that they must avoid to use open source components licensed under non-permissive licenses that are not compatible with their exploitation or business model. Another case of concern for open source communities is the fact that they could integrate available open code components that are not licensed under open source license criteria (as defined by the Free Software Foundation² or by the Open Source Initiative³) but under licenses that turns out not to be compatible with a distribution under open source license (for example, some licenses give free access to source code for research use only, which does not fit with the open source license criteria). This uncertainty is reinforced for collaboratively developed software (CDS), for which large numbers of contributors, with various profiles, are jointly developing software, sometimes from different countries with different applicable laws.

However, improving the legal quality and legal safety of components or component -based software released under open source license, is globally considered as a key issue for the software industry as well as for open source software communities, editors or/and users.

The QualiPSo project is addressing these questions in the “Legal Issues” Activity, contributing to enhance the awareness of the open source ecosystem to these issues. The present deliverable is aiming at providing a general, conceptual framework for Intellectual Property Rights Tracking (IPRT) management of Component-Based and Collaboratively Developed software. However, the methodology proposed in this document is as generic as possible and should apply not only to open source distributed Software or components, but also to hybrid software (proprietary software containing permissive components) or even to fully proprietary software (with no non-

¹ Ref : Economic impact of FLOSS on innovation and competitiveness of the EU ICT sector
Contract ENTR/04/112 Final report 2006 Lead contractor UNU-MERIT, the Netherlands

² <http://www.fsf.org/licensing/>

³ <http://www.opensource.org/docs/osd>

permissive components such as "Copyleft" components) or to software exploited in a Software as a Service –SaaS model.

The underlying model in which the IPRT methodology is proposed consists in considering the coupling between a "licensing in" process – reuse of preexisting components - and an exploitation scheme of the CBCDS that is most often a "licensing out" – distribution - process.

This framework is presented as a methodology to track IPR along the software Product Life Cycle. It is a contribution to a difficult issue that do not pretend to be a universal solution, but a way to address the problem that is aiming to highlight some key dimensions of it and to show some standardization needs (such as license denomination). The methodology presented here has been designed to be used in diverse organizations each having its governance (management) structure and using different tools and development environments. An objective of the methodology is to promote a definition of the Legal Status - as defined below - of software. However, this proposal will have to be discussed and further enriched and could lead to an interesting standardization process to be set up.

By Intellectual Property Rights Tracking (IPRT), we refer in this document to a set of process and actions aimed at defining the legal status of Software and monitoring its evolution during the development life cycle.

The objective is to look at the software development process from the legal perspective, in order to achieve software development with a controlled legal status, which is compliant with distribution and exploitation models of the Software as defined by its editor(s)/owners. The compliance is checked during audit phases. It allows to determine risks associated to unclear chains of rights and obligations and to cover those risks with appropriate actions (technical, legal, insurance).

For the purpose of this report, the term "Software" means "Component-based and Collaboratively Developed software". When "software" is written without a capital letter "S", the word has the generic meaning.

1.2 Free software development life cycle, legal maturity, and industry or business trends.

According to SENYARD and MILCHMAYR⁴ free software development process could be modeled with a three phases lifecycle. A first “cathedral” phase is characterized by closed development performed by small group of developers, followed by a transition phase in which the development community is emerging together with its appropriate re-engineering of the code architecture, to facilitate collaborative development. This phase is leading to so called “bazaar phase”.

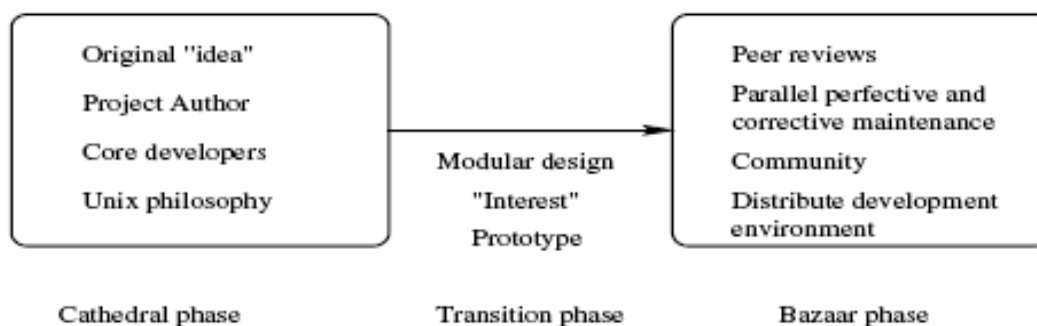


Figure 1: Free software lifecycle

The basic idea is that open source communities do not appear spontaneously, but through a continuous process in which the organizational structure of the development team is evolving from a “clan” based governance to a structured community having sometimes a rather sophisticated governance model.

In a sense, one might consider that the legal quality or maturity goes with the technical quality or maturity along this cycle.

In the first phase of the development, the developers are focused on the technical or functional proof of concept and they often consider legal issues as not essential. The awareness and organization of how to handle legal issues should be present at any stage of the development process, but is very likely to be implemented differently at each stage of the life cycle.

However, an important trend that appears recently, with the availability of license checker,⁵ is the external audit of the legal quality of existing open source projects by independent third parties. The available audits show contrasted situations with respect to the legal quality of open

⁴ How to Have a Successful Free Software Project, Anthony Senyard and Martin Michlmayr Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)

⁵ License checkers : <http://sourceforge.net/projects/oslc/> or <http://fossology.org/>

source projects releases⁶.

Another trend is the emergence of a new service industry that provides software intellectual property management solutions: code reuse analysis, license checking, components license compatibility for open source or proprietary software packages.⁷

The last trend we have observed, in relation with legal quality of open source software, is initiative aiming at sharing legal information related to open source components in shared databases.

1.3 Legal dimension of the software development process and good practices

Introducing preexisting code or components indubitably raises legal issues: from a legal point of view, this is indeed a “licensing in” activity that needs to be regulated by an appropriate “licensing in” policy.

Introducing a legal dimension into the software development process raises project management issues. It is true that some “Open Source” development communities have a technical governance structure (college of architects or similar structures), but they should also take into account legal governance issues at each maturity stage.

This approach (introducing legal dimension in Software development) must be discussed at the highest level of the governance structure in charge of the Software development and must require its involvement and formal policies statement. Taking into account legal issues, in a Software development process, raises a rather large set of questions that the project manager(s) should cope with.

We simply propose to consider legal issues related to CBCDS as way of coupling a “licensing in” process – related to code reuse – with an exploitation scheme that is most often a “licensing out” process.

Figure 2 represents the situation from this perspective: CBCDS is assembled for a given exploitation scheme. Most often CBCDS are aiming to be distributed under a given exploitation license (right part of the figure)⁸. This distribution license choice should be determined as soon as possible in the development process. This is the “licensing out” policy.

CBCDS is also assembled from preexisting components (left part of the figure), having their own distribution licenses. Such preexisting raw material should only be used (i.e. incorporated in the CBCDS) if the license attached to this particular preexisting component complies with a “licensing in” policy. The “licensing in” policy is defined at the governance level with the aim that any license attached to reused components complies with the “licensing out” policy. In other

⁶ <http://www.neolex.se/open-source-audit/database/index.php?p=contents> or

⁷ see <http://www.blackducksoftware.com/> or <http://www.protocode.com/> or <http://www.palamida.com/> or <http://www.nexb.com/corp/>

⁸ Except when the Software is exploited in a SaaS mode.

words, any license attached to the reused component is compatible with the license chosen to distribute the CBCDS.

The “licensing in” or code reuse rules should also give clear instructions on how to use the reused component or reused source code within the Software architecture. It is necessary that the use of each component within the Software architecture complies with the license attached to the component, as the scope of the license is a function of how the component is used in the Software.

Both “licensing in” policy and use rules should be formally defined by the governance structure. It is also important to do appropriate training of the developers. They must respect the “licensing in” policy and use rules of components and source code within the architecture when they used preexisting components or preexisting source code.

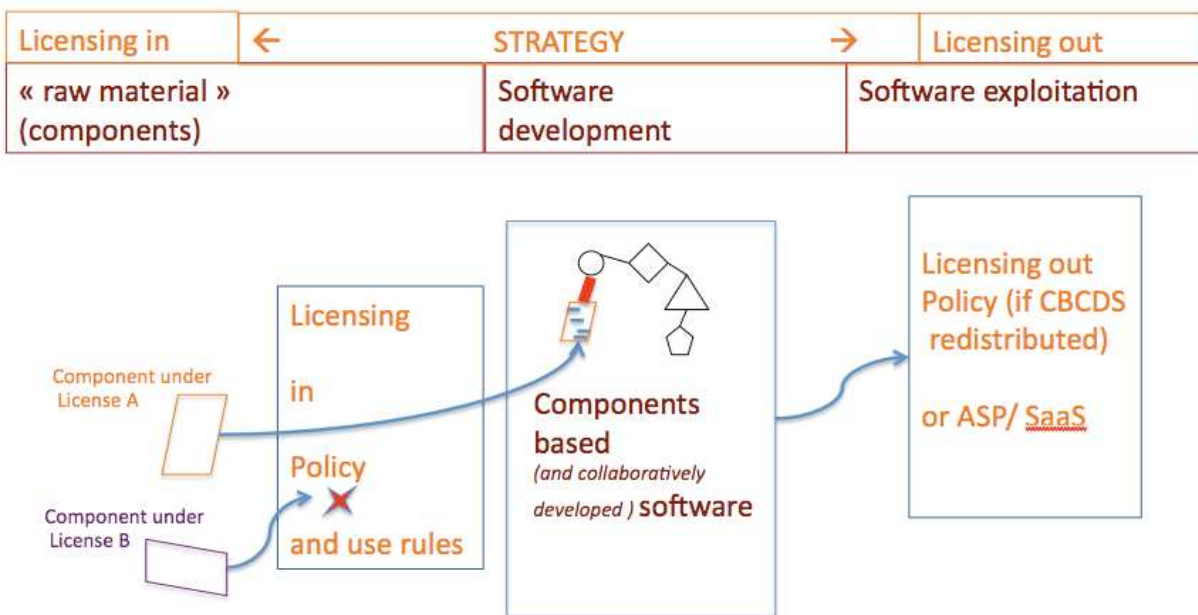


Figure 2 : The CBCDS model from the legal issues perspective

Part of the IPRT methodology proposed in this report is simply a way to check that the development process respects the “licensing in” policy and its rules of usage. These conditions are necessary but not sufficient to make the “licensing out” possible.

1.3.1 Freedom Matrix:

The above-mentioned legal questions are basically related to two major features of the development process: the fact that the development is collaborative (or not) and the fact that there is component reuse (or not). We propose the Freedom matrix (Figure 3) to summarize the different cases encountered.

Code reuse (open source or not)	yes	<p>CASE 2: Medium risk</p> <p><i>(licenses compatibility)</i></p> <p><i>Freedom to exploit ?</i> <i>Freedom to enforce ?</i></p>	<p>CASE 4: High risk</p> <p><i>(licenses compatibility contracts – choice of exploitation license)</i></p> <p><i>Freedom to exploit ?</i> <i>Freedom to enforce ?</i></p>
	no	<p>CASE 1: Low risk</p> <p><i>Freedom to exploit</i> <i>Freedom to enforce</i></p>	<p>CASE 3: Medium risk</p> <p><i>(contracts – choice of exploitation license)</i></p> <p><i>Freedom to exploit ?</i> <i>Freedom to enforce ?</i></p>
		No	yes
		Collaborative development	

Figure 3: The freedom matrix

The figure shows four development situations. The simplest development situation (CASE 1: low risk) is non-collaborative projects with no code reuse. This situation might occur at the early stage in the lifecycle and is a common situation in academic or research organism software developments. As there are only components developed from scratch, and only one owner, the freedom to exploit (use, distribute, etc ...) is acquired and so is the freedom to enforce the IPR associated to the software in case of counterfeiting acts by third parties. You should not have to define a “licensing in” policy, as you do not reuse component.

In a second situation (CASE 2: Medium risk), where a non-collaborative development project reuses pre-existing components, the IPR owner of the Software may not freely exploit the Software, as he should respect the provisions of the licenses attached to the reused components, or exclude them in your “licensing in” policy. Indeed, the freedom of exploitation is very often given by open source licenses attached to the components⁹, with some restrictions on the distribution license of derivative works (constraints for your “licensing out” policy if you use such components). Moreover, the upstream compatibility of the license attached to each reused component with the license of the Software must be checked¹⁰. This compatibility checking task is the cost driver of the legal analysis of Software. A common pitfall in this situation is to reuse non-open source components, having a license with no restriction for research applications or activities, but that would not allow commercial activities¹¹. This is a typical case of license

⁹ Except when the exploitation model is a Software As A Service and the reuse components are under a GNU Affero General Public License or license having similar legal effect. <http://www.fsf.org/licensing/licenses/agpl-3.0.html>

¹⁰ For compatibility refer to “Report on Study of the compatibility mechanism of the EUPL (European Union Public License v1.1.0)” 11th September 2006 Fabian BASTIN, Philippe LAURENT Advice Report <http://ec.europa.eu/idabc/servlets/Doc?id=27472>

¹¹ See for example the numerical recipes licenses (<http://www.nr.com/licenses/redistribute.html>)

incompatibility with an open source distribution scheme.

In this development model, the owner would have the opportunity to enforce the IPR of its Software, if counterfeiting acts are reported to him or observed by him.

A third situation (CASE 3: Medium risk) is collaborative development process with no code reuse. In this case, joint owners of the IPR on the Software should agree among themselves to decide on a common exploitation scheme or license (in French law, each owner, taken individually, does not have the freedom to take the initiative to exploit the Software without the express consent of all the other joint-owners).

The last situation (CASE 4: High risk) is the most complicated situation from the legal risk point of view, as the constraints of the two previously detailed cases (2&3) apply to it.

As in CASE 3, a joint owner does not have the freedom to enforce the IPR associated to the Software without the consent of all the other joint-owners.

1.3.2 Legal risk and legal Maturity

The legal risk associated to software development is not the same depending on its exploitation scheme.

For example, we can consider the extreme case of Software exploited in a Software as a Service (SaaS) model: this type of software is not “distributed”, which means that the license provisions of its components referring to distribution would not apply in most cases (except if the software contains GNU Affero GPL-like licensed components). However, it should be pointed out that the notion of “distribution” is not clearly defined from a legal point of view among the different national law systems and can lead to discussion as such or in actual court case¹².

On the contrary, legal risk increases in the case of distributed software: distribution constraints and incompatibilities issues between the components’ licenses have to be taken into account.

However, we believe that software legal maturity should improve along with its technical maturity. Indeed, the closer we get to software’s first release, the more important it becomes to make sure that its legal status is compatible with its exploitation scheme, in order to avoid or limit risks.

Figure 4 shows schematically, in the case of distributed software, that legal acceptable risk or legal maturity should be reached as soon as possible during the development process, and most preferably before the first release of the Software.

¹² Harald WELTE, Erik ANDERSEN and Rob LANDLEY, the FREE SOFTWARE FOUNDATION vs FREE

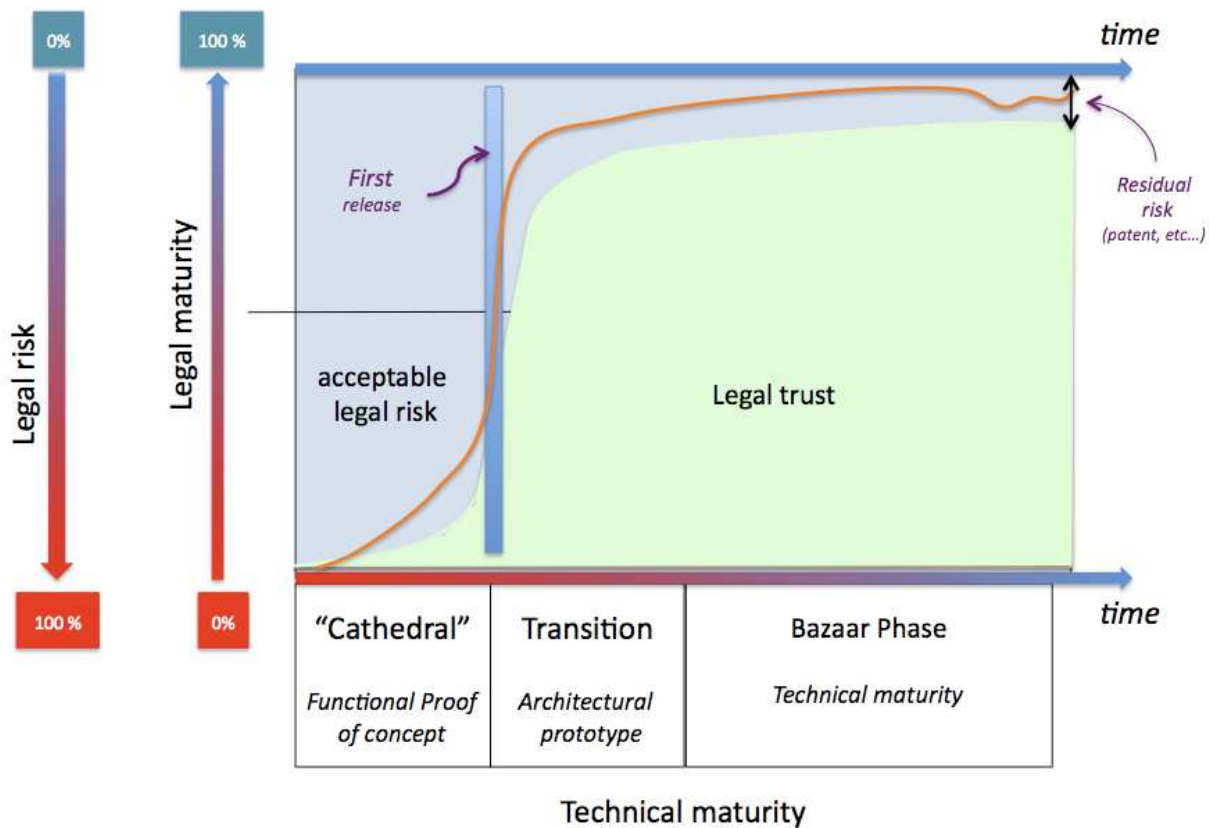


Figure 4: The technical vs legal maturity diagram

The practical consequence is that you should define both your “licensing out” and related “licensing in” policies preferably before the first release. We call “Legal roadmap” for a CBCDS the set of coherent and formalized “licensing in” policy, use rules, and “licensing out” policy.

1.4 IPR Tracking models

Legal quality of Software could be reached through different ways and processes, but having a Legal roadmap is a key success factor. This could be pragmatically done by limiting the choice of pre-existing components for the developers within a limited list of qualified components. This is an example of “licensing in” policy. Some companies are using such a licensing-in policy and process and they have set up a database of qualified (certified) usable open source components for software developments. If developers want to use a component that has not been listed in the database, a team of experts check it through a technical and legal validation process, and the database is enriched with the component if it is compliant with the company policy, or the component is black listed if the component does not pass the validation process.

Figure 5 shows a matrix describing four situations, depending if the IPR Tracking model is based on a content controlled process (selection of component from a certified component database) or if the IPR Tracking is based on a legal status checking process of Software (integration of an IPR tracking methodology in the development process, which can either be used continuously, as Software is developed and components are licensed-in, or used sequentially, only at various stages of the development process.)

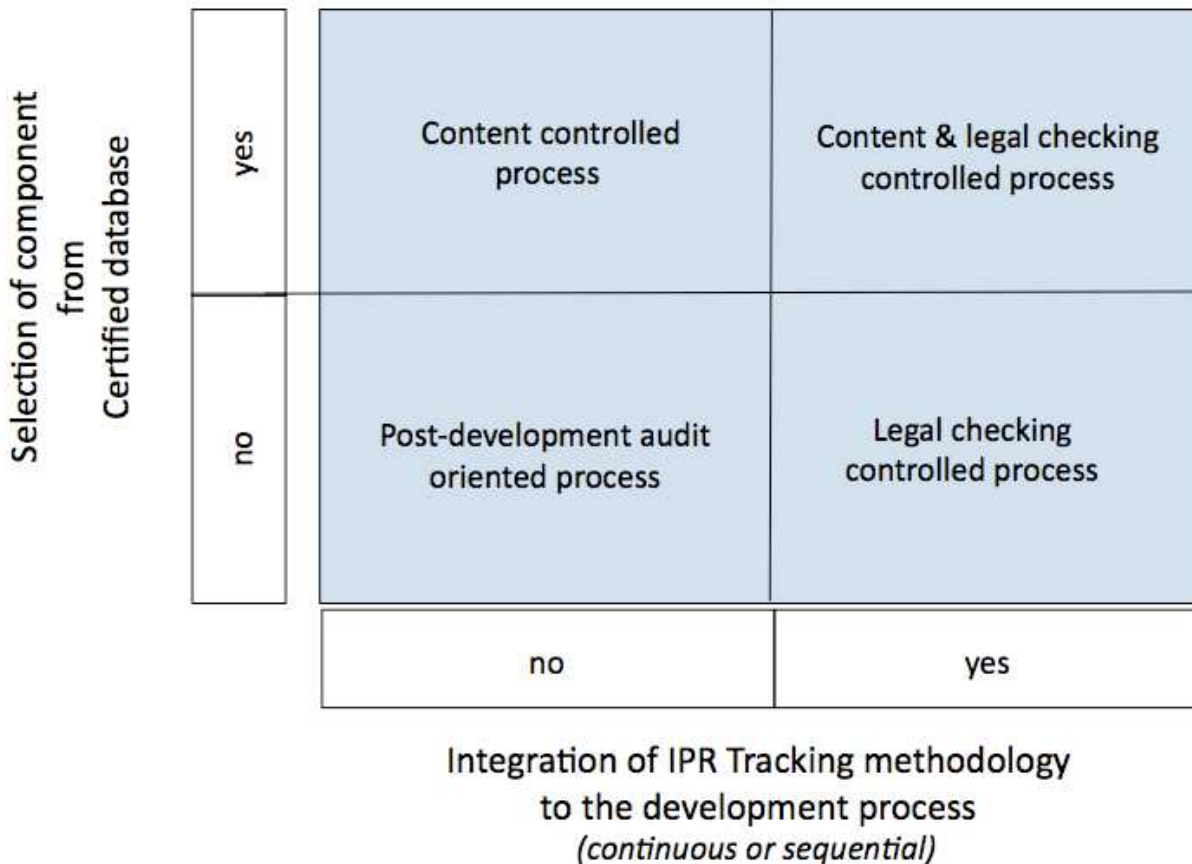


Figure 5: IPR Tracking models

The worst case situation is obviously when no legal status checking process occurs during the development and when components are not chosen from a qualified and sometimes certified components database. No “licensing in” policy, no control of use rules of reused components and even no “licensing out” policy (the choice of the distribution license will be made after the development.) If a post-development analysis or audit is not performed, the Software is very likely to be of low legal maturity and its distribution should represent a high legal risk for the Software editors/owners.

On the contrary, the best-case situation is when an IPR Tracking methodology is integrated to the development process, legal status regularly checked according to a defined Legal Roadmap, and when the pre-existing components are selected from a database of qualified components.

1.5 Exploitation intentions and IPR Tracking

However, in this context, we believe it is essential, for development managers or governance committee of the development community, to define at an early stage the economical model of diffusion or exploitation of the Software, most often the “licensing out” policy. This exploitation scheme is the “Business” model associated with the software. This is essential, as IPRT is

useless if you cannot compare Software's legal status with its exploitation scheme (licensing out policy).

There are many different exploitation models (business models) for Software. Although it is not the purpose of this report to exemplify any exploitation model, a topic that is addressed in depth in Qualipso Activity 2, it would be useful to skim through main exploitation models in respect with the IPRT issue and to illustrate it.

The formalisation of the exploitation model of the Software is a key task, as there is a relation between the exploitation scheme and how to adapt the proposed IPR Tracking framework to your own case.

"Service based exploitation", as such, is the first exploitation or business model to be considered here, as it is always possible to provide services whatever your licensing model is, including when you have chosen to distribute the Software under a non permissive GNU GPL-like license, which is the most common business model of open source companies. This exploitation mode is not related to the licensing scheme of the Software, except that the service provider may propose additional warranty provisions in the service contracts.

Figure 6 shows an exploitation schemes matrix that covers the other main licensing scheme based exploitation models encountered in software industry.

Exploitation of the Software under Commercial SaaS (Software as a Service) or PaaS (Platform as a Service), FreeSaaS, or open SaaS model (an Open Source Software to be used in a SaaS mode) is represented within the exploitation matrix.

Basically, your exploitation scheme (as a company, organization, consortium or community) could be based on proprietary licensing or double licensing if you want to get or capture some revenues from your licensing program. Double licensing may be your economical model, as a community, to get revenues to sustain the development of your open source project on the long term.

On the contrary, if you (as a company, organization, consortium or community) do not intend to capture revenues from your licensing or exploitation activities, you can distribute a freeware, provide Software exploitation under a Free SaaS mode, or license the Software under a permissive license.

The Software exploitation model could be distribution or software as a service. Distribution could be done under proprietary license, open source license, or dual licensing based. Combination of models or mix exploitation mode is also possible

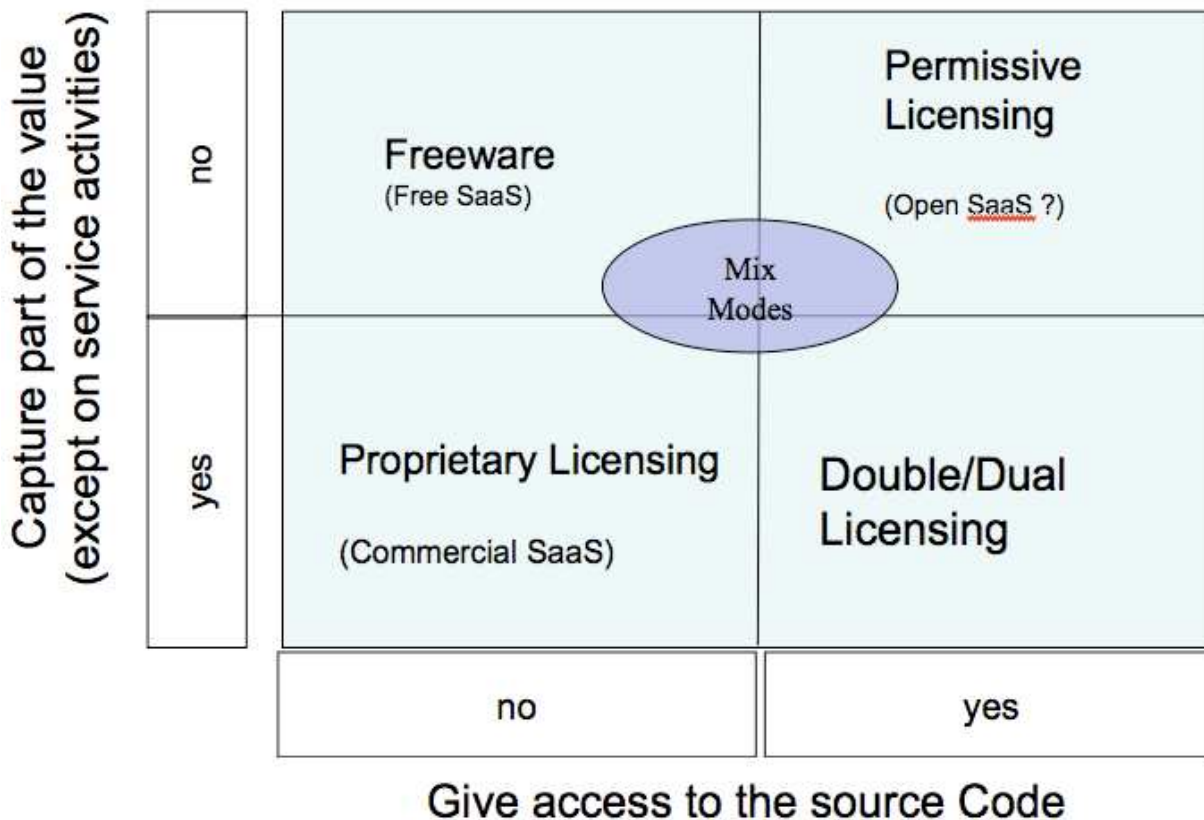


Figure 6: The exploitation schemes matrix

For example, if the exploitation model is based on a “Software as a Service” architecture and that the service provider doesn’t want to give access to the source code of its software, legal analysis could be simplified, as there might not be any code distribution (server side). Components under GNU Affero like license have to be identified in such case, as GNU Affero is aiming at obliging access to the source code of derivative works or composite works for third parties, even used through Software as a service architecture, with no physical distribution of binaries or source code. With such SaaS exploitation intentions, the “licensing in” policy must contain interdiction to use components under GNU Affero Public Licence.

Another example is the case of a software editor whose aim is to distribute component-based software under a proprietary license (licensing out policy) that integrates GNU LGPL or other permissive licensed based components (licensing in policy). He should avoid GNU GPL licensed components, or avoid integrating GNU LGPL code in Software’s proprietary files.

Sometimes, the strategic intention of the management concerning the exploitation model (licensing out) of the software might change during the course of the development process. Such changes might lead to high reengineering costs if components having non-compatible licenses with the new “licensing out” policy were used during the development.

One can also imagine that regulated business might introduce responsibilities constraints, such as Software for critical applications (example, software for medial application or used in domain of space and defense) and IPRT methodology should be tuned accordingly.

The governance committee in charge of the development strategy might also have diverse control strategies of the architecture, technical and sometimes scientific choices, or of technical quality and maturation process. It is often a concern to early developers to control the development to avoid “forks”.

For example, the QPL license is considered as a license tool which is adapted to Software maturation, and which allow avoiding premature “forks” for technical divergence by providing exploitation centralization. IPR centralization by assignment is another control strategy commonly used for the FSF projects, by the FSF.

Another issue is the enforcement of the license you have chosen to distribute the Software you have developed if a third party did not respect it. Assignments of IPR of contributors might be necessary if you intend to have reasonable transaction costs before legal action. In large “open source” projects with a high number of contributors/owners, the agreement of each owner is necessary for entering in a litigation process or trial, at least for counterfeiting actions. This might be almost impossible if IPR centralization with assignment of contributor’s IPR was not the model chosen and if joint ownership was preferred (where each contributor keeps its ownership and provide its contribution under a compatible license). IPR acquisition of contributors should be part of the development process if the Software editor wants to enforce the license against counterfeiting third party. FSF policy of some FSF projects is based on such IPR centralization model (i.e. GCC project for example and related key components – MPFR, etc...)

The choice of joint ownership versus IPR centralization is of importance for the governance committee as it strongly shapes the IPRT and management processes during the Software Life Cycle.

To sum up, the IPRT process you should have to design and implement from the QualiPSO IPRT framework could be function of when you intend to use it, of the complexity of the Software architecture, of its technology, of the exploitation plan and of the enforcement intentions you are considering. For example, if you are a company that plans to use open source components to develop a piece of software that you intend not to distribute, but to propose as a SaaS, your IPR Tracking process might be simpler than for CBCD Software developed by worldwide distributed teams or individual contributors to be distributed and defended by the editor.

1.6 IPR Tracking QualiPSO deliverable objective and structure of the report

It is not the purpose here to define a rigid framework or “THE” universal IPRT methodology adapted for any software development. On the contrary, the objective is to present a simple framework and a set of key issues, both technical and organizational, that one should adapt to its particular case or its own software development process.

In the next section, the goal and objective of an IPRT methodology are presented as well as the main concept for IPRT, the Legal Status of Software. The following section proposes an IPRT methodology and finally, the last section describes the key elements that have an impact on the

legal status of the software. The developers have a key role to reach a Legal Status that is compliant with exploitation intentions or model of the editor, as the Legal Status is formed through their actions and with this respect, so called “good (IPR) development practices” are of tremendous importance.

In a certain sense, Audit phase for Legal Status formalization is a measure of developer’s ability to take legal dimension into account during the development process.

The Audit Module described in this report is an example of IPRT methodology implementation. This Audit Module uses a questionnaire to define a “perceived” legal situation, as described in the corresponding section of this report. This questionnaire is presented in Annex 1.

It is specified that the Audit Module was used in the context of PALETTE, an FP6 EC project. Its results are presented in Annex 2 as a case study. Annex 3 briefly presents the Audit Module and IPRT methodology used at INRIA and a complete use case (DIET¹³) exemplifies the section 4 (Audit) of the report. The aim of the DIET project (with Frédéric Deprez, Eddy Caron, David Loureiro and Aurélien Cedeyn as core developers) is aiming at developing a set of tools for building computational servers.

¹³ <http://www.inria.fr/actualites/sc07/art1.en.html> or <http://graal.ens-lyon.fr/DIET/>

2 Project management and Legal Status

2.1 Implementing the IPRT Methodology: defining tools and processes

The aim of the IPRT methodology is to provide a clear picture (Legal Status) of the intellectual property rights associated to the Software and to any of its components as well as all other legal constraints that may exist.

From the point of view of project management, the management team is responsible for defining the process and tools used for IPRT. They have to:

- Set up appropriate training sessions for developers to foster “development good practices” and legal issues awareness within the organization, if necessary.
- Define a clear and formal (not implicit) legal development roadmap “Licensing out”, “licensing in” and related use rule (to make sure that the Software’s legal status is compatible with the exploitation scheme)
- Inform and train developers to be aware of this legal roadmap and especially of the “licensing in” policy and use rules of reused components and IPR Tracking process to follow
- Choose whether they empower the development team to follow the IPRT process on their own, or if they prefer to separate the development and the -legal- quality control, as we recommend hereafter (and as it is normally the case in most organizations for development and quality assurance).
- Choose if they would prefer internal audit structure or externalization of the legal Quality Control (outside the organization) or partnership with external service company. (Typical make, strategic partnership or buy decision).
- Define the IPRT methodology that will use to determine Legal Status of Software
- Define when such IPRT Methodology should be used (at different stages of Software development or exclusively at the end of it?)
- Check compliance with exploitation intentions at chosen points in time (as soon as possible at the beginning of the project, and usually anytime a new release is planned)

We wish to insist on the point that the IPRT Methodology can either be used as:

1/ a “support tool” during Software development:

IPRT Methodology is used at different stages of the development process, in order to be sure and confirm that Software’s legal maturity increases and improves along with its technical maturity. This should usually be the case if all the actors involved (developers, lawyers...) are kept informed of intentions and development strategy, and of any change occurring to it.

For example, depending on the strategy adopted (licensing out), developers may have to refrain from using pre-existing components submitted to a particular license (licensing in - exp. GNU GPL license if the objective is to distribute Software under a BSD license). Lawyers may have to be careful to avoid writing research contracts with IP clauses incompatible with intention of exploitations.

Taking into account the strategy and try to remain coherent with what we believe to be “good practices”. In the context of IPRT Methodology, we assume that people are respectful to such good practices. However, we also believe that it is important for a company, an institution or an organization to make sure that people involved in a project are aware of such good practices. If necessary, such people should be trained.

Consequently, this skilled and trained team can then be expected to perform a high level of self-controlled activities and to develop high legal quality Software, both due to their awareness of good practices and their compliance with the exploitation goal and the corresponding development strategy.

2/ a tool for “emergency” cases or a final validation:

This is the case when the IPRT Methodology is used for the first time at the end of a development process or when it is used when problems concerning the software arise (for example when software’s legal status is suspected by a third party to be incompatible with its current exploitation scheme).

The IPRT Methodology will usually be, in this context, more time consuming, as it may be more difficult to apply to Software, as it might be more difficult to go “back in time “ to gather the information which is necessary for the analysis.

In addition, if it turns out that no development strategy has been defined accordingly with Software’s exploitation scheme, the risk is that the analysis may reveal that Software’s legal maturity is far from being sufficient in order to limit legal risks.

2.2 Introducing the notion of Legal Status

Before entering the details of the proposed IPRT methodology, it is necessary to define its main feature and the main object it manipulates: the Legal Status to be tracked down.

Defining the global Legal Status of a Software package is not obvious and there is surprisingly very few academic works on this topic¹⁴. However, it is a key point to reach a standardized definition that would be used for IPRT. Although IPRT methodologies and tools to determine the Legal Situation of Software might be specific to a given organization, there is a need for a generic Legal Status definition that would allow exchange and communication on it. One could even imagine that the components themselves will be distributed with their Legal Status. What we propose hereafter is a rather abstract definition of the Legal Status of Software.

This definition specifies a minimum set of categories or elements, which have an impact on the way Software may be used. This allows to determine subsequently the possible ways of using and exploiting this Software, and to determine if they are compatible with the exploitation goals that have initially been chosen.

¹⁴ Open (Research) issue toward a legal framework for OSS, FOSDEM 2008 ROUSSEAU

We propose to base the definition of the Legal Status on a set of only four following elements:

- The position of the software in the chain of rights
- The ownership of intellectual Property Rights of the Software
- The legal conditions covering the exploitation of the software
- Other existing enforceable IPR against software

Moreover, we have to define a minimum set of characteristics for each software component. **We propose the following set, which has to be discussed and standardized for automated recursive components analysis (each combination of components can be seen itself as a component).**

- Component Name: For component identification
- Component Status: From the development of a CBS point of view, a given component could be an original work written for the purpose of the CBS, a pre-existing component used “as such”, without modification in the CBS (not modified anterior work), a pre-existing component modified to be used in the CBS (modified anterior work).
- Composition Rules: Information such as nature of links and dependencies between components should be defined here (categories to be standardised)
- Version: for exact component identification
- Functional zone: it is necessary to know, in the context of the IPRT methodology, in which functional zone the component is located (core, kernel...).
- Localization: It could be the localization in a repository, in an archive, in a distant server or others.
- License: The license attached to the component. The license denomination is also a characteristic that would need standardisation.

A. Position in the chain of Rights

The Software can either be:

- An initial software: refers to software which has not been developed on the basis or by integrating (or extracting from or translating) previous software.
- A derived software: refers to software for which at least one file containing source code

has been modified by applying sets of derivation rules.

- A component-based software: refers to software made of different components by applying sets of composition rules (given that a component can also be itself a component based software).

The “component” term itself could be subject to discussion. It should be considered in this methodology in the broader sense of software “brick”.

B. Owner of Intellectual Property Rights (IPR)

The ownership has to be identified for both:

- Moral rights: in most national laws, authors are the exclusive owners of their moral rights. In France, moral rights are indefeasible, inalienable, perpetual and can not be transmitted (notwithstanding some exceptions, such as for example Luxemburg Law, which allows an author to assign its moral rights). However, in the case of software, these moral rights are generally limited compared to those of authors of other pieces of copyrightable works.
- Patrimonial (economic) rights: they can either belong to the author, be automatically assigned (example: from an employee to an employer) or be contractually assigned.

C. Legal conditions of exploitation of the software

The exploitation (and use) of a component can be legally restricted by several means:

- Exploitation is restricted by a license: for example, the GNU GPL forbids distributing any modified software under another license.
- Exploitation is restricted by an agreement (other than a license): this can be, for example, a joint property agreement which forbids each party to exploit the software without the other party’s previous consent. Another example is the case of Software developed under EC funded shared costs actions, where the Legal Status is partly determined by the so-called general conditions of the model contract with the European Commission jointly with a specific consortium agreement, defining what are the so-called “*access rights for use*” (exploitation) granted to partners and/or third parties.
- Exploitation is restricted by Law: for example, in case of joint ownership and in the absence of a joint ownership agreement, Law can restrict exploitation. Another example may be export restrictions (forbidding distribution to some specific countries).
- Exploitation is restricted by another binding rule or legal provision (judgment)

Tracking down the legal conditions of exploitation is sometimes costly, time consuming and would require assistance by a legal counsel.

D Other enforceable IPR against software

Some intellectual property rights (that may belong to third parties) may conflict with the exploitation of the component:

- Patent: it is however limited to the territories where it has been filed and registered.
- Trademark: it is usually the name of the software and is limited, like patents, to the territories where it has been filed and registered as a trademark. Trademarks issue may limit your ability to exploit Software under a given name. Trademarks are often used as a mean to control forks, or to make them explicit. To illustrate this point you could refer to the interdiction to redistribute modified versions of the official Mozilla™ web browser release under the Mozilla™ name and by using Mozilla™ logo (visual trademark). This obliged the Debian™ community to distribute its Mozilla™ modified web browser under the **Iceweasel** name.
- Copyright (name of the software, images, icons, ... that may be contained in the component): unlike patents or trademark, it automatically benefits of a worldwide protection.
- Others elements used with, or integrated into the Software: it may be for example data base subject to “sui generis” rights in European Law, identified know-how protected by confidence...

E Representation of the Legal Status

This set of four elements enables, in our view, to have an automated process for the recursive determination of the Legal Status of complex and large software: indeed, both software and each of its component's Legal Status can and must be determined this way.

A representation of this Legal Status is suitable in order to be implemented as a software component or module for Legal Status determination that would be used by software development managers as a core service in the context of various organisations or software development environments, such as the QualiPSo “Next Generation Forge”.

2.3 Intellectual Property Rights Tracking methodology

2.3.1 General

An "IPR tracking methodology" provides a method which determines the legal status (LS in the sequel, including rights and obligations, copyrights and authorship when available, origins of the work for dealing with export control rules,...) of a software at any stage of its development. It has to be applicable at least in the various development situations considered by QualiPSo (in the development or collaborative tools).

IPRT is a rather complex activity, that would involve both a dedicated team having various skills (technical, legal, managerial, quality control), and dedicated software tools to improve productivity, such as software tools providing automatic code mining (license checking and

compliance, source code comparison, statistical reporting tools¹⁵, etc).

Such tools make it possible to perform the IPRT at reasonable costs, quality and duration. Moreover, this automation of IPRT process is a necessity for large CBS, with large number of components (the largest CBS could reach more than ten millions lines of code and more than one hundred thousand components, typically CBS routinely use more than a dozen components.)

Each entity or company has its own governance, organization, rules, development environment, processes, and regulation constraint for software development. Consequently, the methodology presented hereafter is designed at a level of abstraction that aims to be generic enough to be tailored for any given organization.

2.3.2 Scope of the IPRT process: what is analyzed?

In order for the IPRT process to be effective, it is necessary to define its scope, by specifying the elements that should be considered as part of software to be analyzed.

Indeed, should an element be forgotten, the analysis would turned out to be incomplete. This would lead to the interpretation of an erroneous Legal Status and subsequently, to an exploitation of a software in good faith but nevertheless unsecure from a legal point of view.

We believe this scope should be defined at two different levels: a legal one and a more practical or technical one.

a) Scope from a legal point of view:

The different elements which should be considered as being part of a piece of software are defined by law and/or case law. This would therefore usually include in some national laws (in particular in French Law) source code and object code, obviously (including pre-existing external components' source code), but also the specification of the software and preparatory documents.

Elements identified by law as being part of Software should therefore fall in the scope of what should be analyzed.

b) Scope from a technical point of view:

Software is often made of various and numerous components of different nature (core's components, libraries...), sometimes in a complex architecture. It is therefore important, for the purpose of the audit, to define precisely what components are part of Software and should be analyzed. It is the reason why we believe that a detailed description of Software is necessary (see point 4.1 hereafter), in order to know what is actually going to be analyzed.

¹⁵ such as StatSVN <http://www.statsvn.org/>

Audit team

The audit team is the first and the most obvious interlocutor in the context of IPRT process. For the purpose of the present IPRT process, we believe the audit team should include developers, but also IPR lawyers/legal counsels, the transfer/licensing manager, the product manager and any other person involved in software development's life cycle: both legal and technical skills are needed to use the IPRT Methodology.

Indeed, although IPRT Methodology is applied to software, there are numerous legal issues, such as license compatibility, IPR clauses in contracts, identification of authors' status, on which it is highly recommended to work with a lawyer. This is also recommended for responsibility reasons, as lawyers are the authorized persons to give counsel or an advice on legal matters.

The development team and contributors should therefore have an awareness of the legal issues associated with Software. This implies that the development team should be provided with appropriate "good practices", training and quality control support. For the purpose of the present IPRT process, we assume particularly that a good level of "-legal- development good practices" is implemented within the development team. This means, for example, that developers do not delete existing headers or do not modify license attached to external components, without formal authorization of the IPR owners of the external components. We also assume that the developers do not try to hide the origin of external code, by reengineering it, changing the names of variables or doing other non authorized practices. We assume "development in good faith" when it comes to use of pre-existing components

Nevertheless, developers should be aware and informed that advanced code reuse detection technologies can detect unfair practices or counterfeiting of that kind; "development good practices" must be the rule and other practices should be strictly prohibited.

3 Key IPRT issues during the software life cycle analyzed by the IPRT Methodology

The IPRT activities, occurring during the development process, are aiming at identifying and mastering elements that have an impact over the legal status of the software.

The objective in this section is not to present an exhaustive list of such elements, but rather to concentrate on the elements which we believe have the strongest impact on the legal status.

3.1 External or exogenous components

A pre-existing external component (sometimes referred to as “Pre-existing know-how” in EC contract definition) has its own legal status, which determines the ways it can be used (part of it is of course its license). Consequently, this component integration or combination in software may impact the latter, if the component legal status is not compatible with software exploitation goals.

For example, if your aim to release a Software under a GNU GPL V2 license (“licensing out” policy), then you should make sure that you do not integrate a component with a license which is incompatible with the GNU GPL V2 – “licensing in” policy – (such as a routine coming from the Numerical Recipes book, which is not compliant with the open source definition and which is not compatible with the GNU GPL, because the Numerical Recipes license does not allow commercial exploitation).

Another consequence is that using an external component or pre-existing code may prevent the freedom to choose a license for Software. For example, if GNU GPL components are integrated in a newly developed software, there will be not other choice but to distribute Software under a GNU GPL license.

Identifying and auditing the legal status of external components, to be combined or integrated into the software, should be clearly done before the project starts and during the development process, each time an external (exogenous) code is integrated.

3.2 Authorship and ownership

It is obviously superfluous to explain here how IPR ownership strongly impacts software legal status. However, it is important to remind that authorship and ownership are closely related but different notions, as the first one usually determines the second one.

Indeed, in some cases, the author of software will also be the owner of related IPR according to Law. In other cases, although the author is not the owner of patrimonial rights, its position as an author and its status allows to determine who actually is the owner of software’s patrimonial rights (for example, according to French Law, when the author is an employee, its employer automatically becomes the owner of the patrimonial rights).

It is therefore essential to identify all authors at any development phase, especially in the context of collaborative development, which logically involves a greater number of actors with various statuses. Depending on the goals of exploitation, on who are the authors and the IPR owners, IPR assignments may be necessary to secure the chain of rights or to facilitate the enforcement of IPR.

For example;

- When part of software development is subcontracted, it may be appropriate to specify in the corresponding contract that the service provider assigns its IPR.
- The Free Software Foundation (FSF) centralizes IPR of contributions to some key FSF projects – i.e. GCC, by assignment of IPR from contributors to the FSF, although GCC is released under GNU GLP license, to facilitate enforcement of IPR

This leads us to the next point: it is also important, in this context, to be able to determine wherever a person is an author or not. Indeed, numerous people can step in the development process of software, yet not all of them may be considered as authors, from a legal point of view. In order to determine if a person actually is an author, the project manager or its IPR lawyer/legal counsel must refer to Law and case law's criteria.

For example, according to French law and case law, a person is an author when it directly contributes to an original piece of intellectual work. This means that a person who merely gives instructions or advices may not be considered as an author. In the same way, people performing tests and other evaluations are not usually considered as authors, unless their activity brings them to propose original source code modifications, to solve critical problems that occurred during these tests or evaluations.

On the contrary, people writing specification, preparatory documents and generating source code are definitely considered as authors.

3.3 Contracts' provision tracking

Software may be developed over many years in the context of various, successive and numerous different contracts (EU projects, bilateral agreements, development consortium agreements...), with different partners. Each contract may have its own specific IPR clauses.

If a development strategy has been previously defined (for example: IPR centralization) and this strategy has been taken into account, when negotiating and writing the contracts, we can reasonably expect all the different IPR provisions to be coherent, as well as compatible with Software's exploitation scheme. However, if such a strategy has not been defined, contracts' IPR provisions may be very different one from another, incoherent and incompatible with Software's exploitation scheme.

For example, if an organization wishes to exploit Software in a dual licensing scheme, with an exclusive proprietary license, this may be impossible if it turns out that:

- Software was used and developed in different contracts, with IPR provisions according to which all results are the joint property of the parties;
- The joint ownership agreement provides that each joint owner may only give non-

exclusive sublicense on Software, with the other joint(s) owners(s)'s prior approval.

For this reason, we believe that this part of the analysis should not be neglected.

4 Presentation of the Audit module

4.0 Overview

The IPR tracking methodology implies to proceed with several audit phases, to make sure the development process has produced a legal status which is compliant with the exploitation strategy of the Software, or which allows to perform corrective action(s) to modify the legal status accordingly.

The Audit module is based on six steps or phases:

A dedicated Audit team is set up (see 2.3.2.1). Development team provides a detailed description of the software (1.) Goals and objectives of the audit are defined (2.) Legal status is determined by comparing “perceived” legal status – based on a questionnaire (Annex 1) – and “determined” legal status – based on a code mining tool such as FOSSology™ (3.) Problem Identification and Risk Evaluation is operated (4.) Critical problem solving is performed (5.) Residual Risk (if any) is covered (or at least evaluated) by insurance before dissemination/distribution (6.)

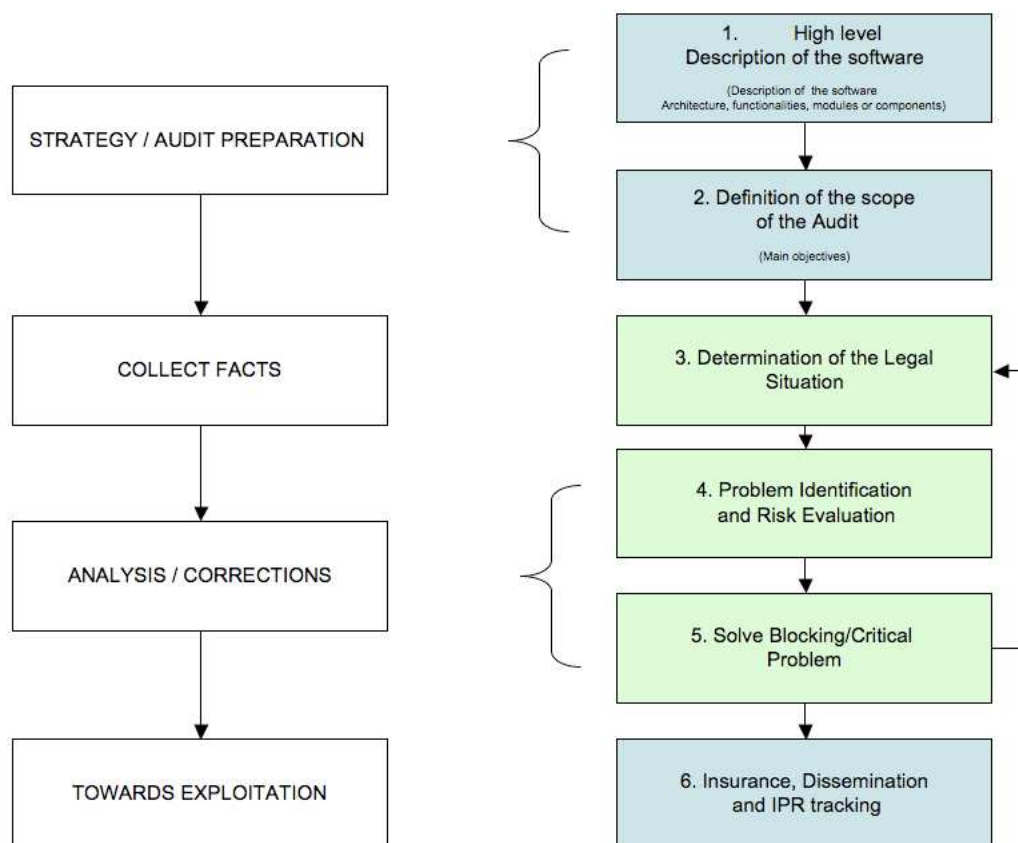


Figure 7: Methodology overview

The draft version of the methodology has been tested to audit software in PALETTE EC FP6 ICT project and to prepare its exploitation plan (annex 2). It also has been tested at INRIA, to analyze the DIET software, in order to validate its exploitation scheme.

The DIET case will be largely used hereafter to illustrate the IPRT Methodology different steps.

4.1. Step 1 : High Level Description of the Software

A high level description of the software is mandatory to allow actors to have a shared functional representation of the software, which is necessary to discuss the audit or analysis' objectives.

This description also enables the Audit Team to establish a relation between information gathered during the audit (for example a research agreement), the component(s) concerned by this information and its (their) location in Software. Of course, this high level description will be regularly updated with the technical evolution of the software.

We believe that a high level description is based on three different levels (two mandatory and an optional one):

- A general description of Software at largest scale (optional and only if pertinent)
- A description of Software's different functional zones and their interactions between them (mandatory).
- A description of each functional zone detailing the different components that are part of it, as well as their interaction and dependencies between them (mandatory)

Such description should help the audit team to determine the perceived legal situation, as explained in step 3 hereafter.

4.1.1 General Description at largest scale:

This representation or architectural pattern of the software is at the largest scale.

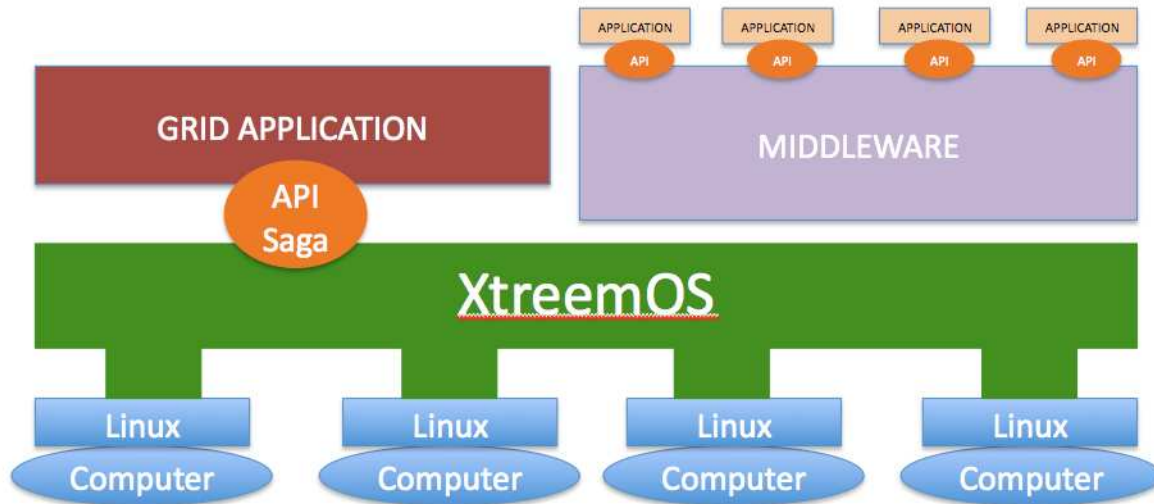


Figure 8: XtreamOS overall architecture

Figure 8 represents an example of such a description for the XtreamOS EC funded software. The overall objective of the XtreamOS project is the design, implementation, evaluation and distribution of an open source Grid operating system (called XtreamOS) with native support for virtual organizations (VO) and capable of running on a wide range of underlying platforms, from clusters to mobiles. On top of the XtreamOS operating system layer you can plug applications or a middleware. On top of this middleware are plugged specific applications.

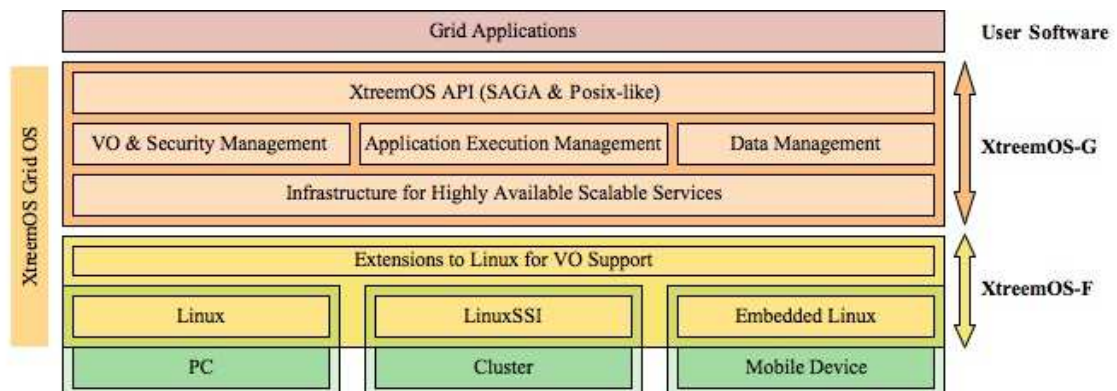


Figure 9: XtreamOS layer description

Such representation is only pertinent when Software is part of a larger and complex structure made of several Softwares. It should therefore be completed with a more precise functional description of the Software, as indicated hereafter.

4.1.2 Description of functional zones:

The first mandatory level of description required is that of Software itself, by detailing the different functional zones and their interactions.

The DIET case:

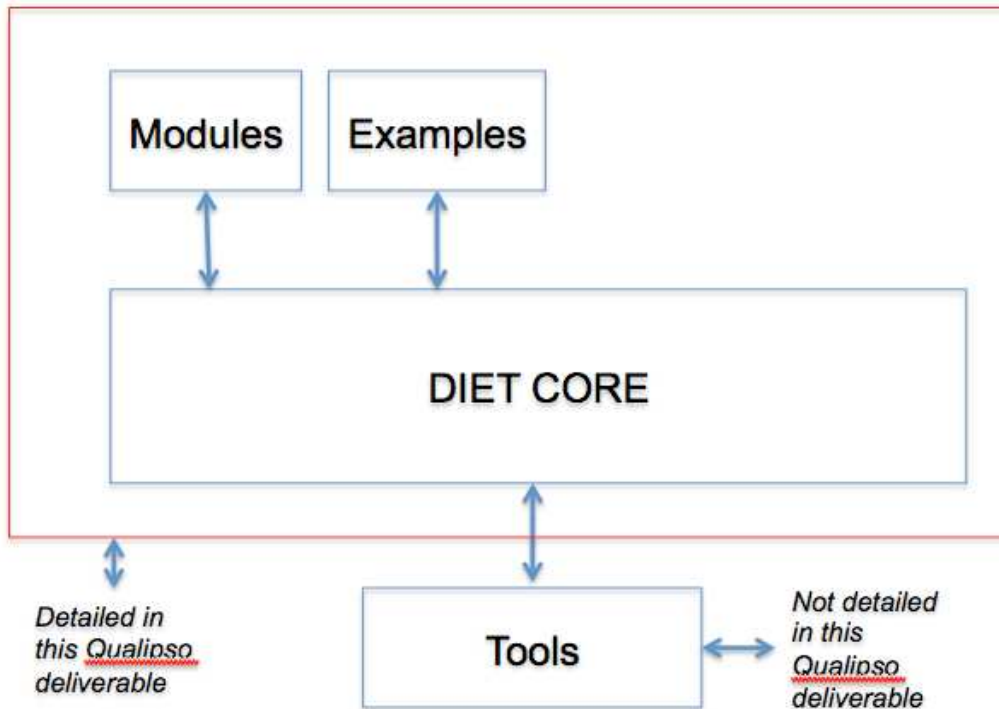


Figure 10: High level Description of DIET's functional zones

4.1.3 Description of each functional zone:

Such description should specify the components each functional zone is made of. It should also indicate, for each component:

- its license, if any;
- if the component was developed specifically for the Software or if it is a pre-existing component (and if it has been modified for the purpose of the project);

The DIET case:

The Software is described as a set of functional areas:

Figure 11 is a more detailed description of DIET Core and DIET Modules functional areas. In this report, we will focus on these two functional areas, but similar treatment was done on DIET

Tools functional domain.

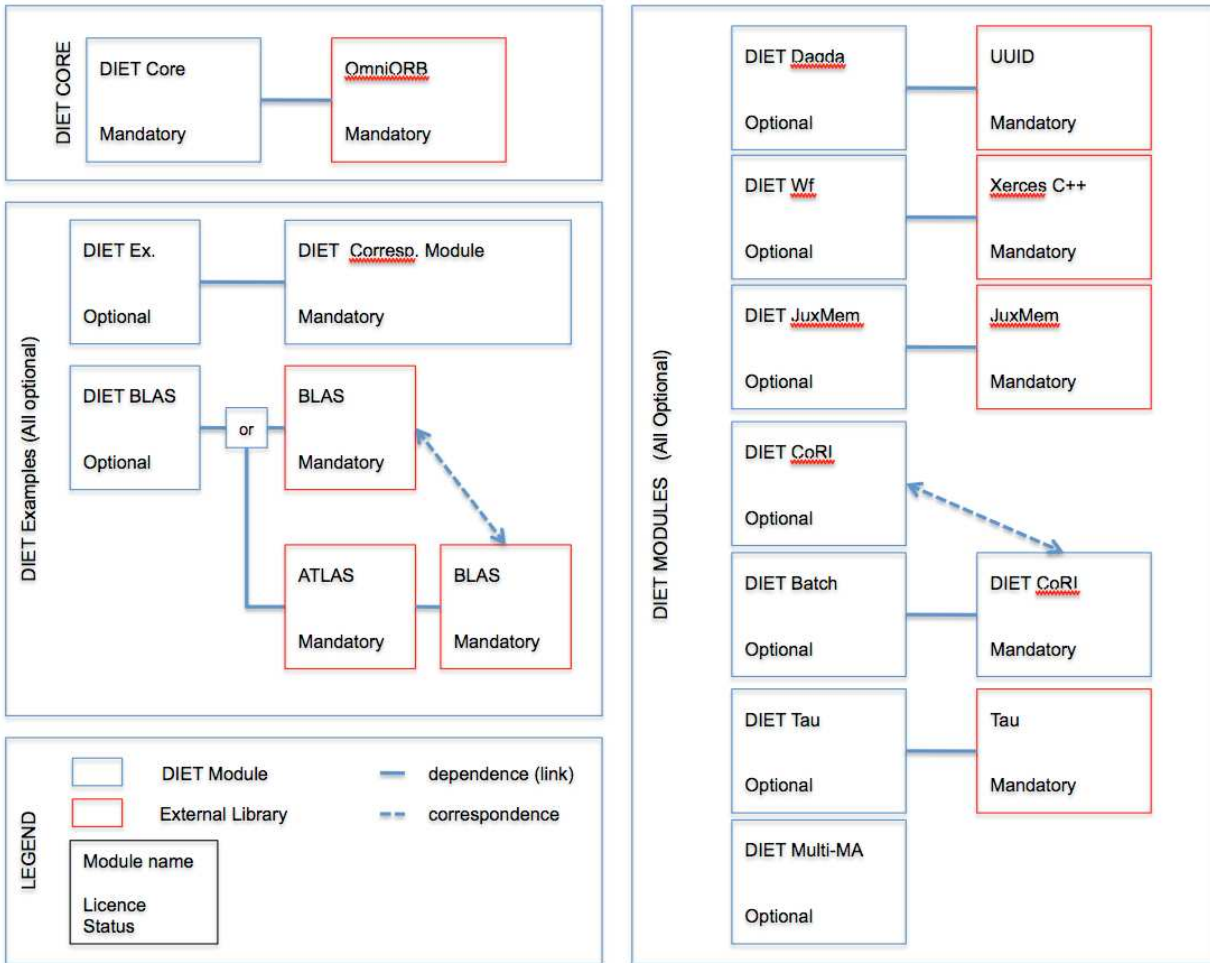


Figure 11 DIET Detailed High Level Description Functional zones

4.2 Step 2 : Defining intentions

If this has not been done previously by the governance committee of Software, or by the core developer’s team at earlier phase in the lifecycle of the project, a key preliminary action of the Audit team is to formalize the exploitation plan, and to have it validated by the governance committee.

If these goals and objectives are not defined correctly, the audit’s relevance will not be optimized. Indeed, once the audit is done and the actual Software’s Legal Status established, goals and objectives will be used together with this Legal Status during the analysis phase, in order to determine if both are compliant (as explained in step 4). If it is not the case, some corrective actions may have to be taken to solve identified problems (as explained in step 5) before preparing the final release (step 6).

The DIET case:

Although the DIET development project has started long before the Qualipso project, developers had an implicit “Legal roadmap”:

In the DIET CASE, the exploitation intention was a double licensing scheme : an open source licensing under the CeCILL license and to keep preserved the possibility to offer proprietary licensing (licensing out policy). For these reasons, the development strategy that has been adopted was clearly non collaborative¹⁶ during the “cathedral phase” and development was done by reusing only external libraries distributed under permissive licenses (licensing in policy). Although enforcement has not been clearly discussed, it was however obtained due to the resulting “de facto” IPR centralization scheme.

4.3 Step 3 : Determination of the Legal Status

4.3.1 “Perceived Legal Status” by the development and audit team.

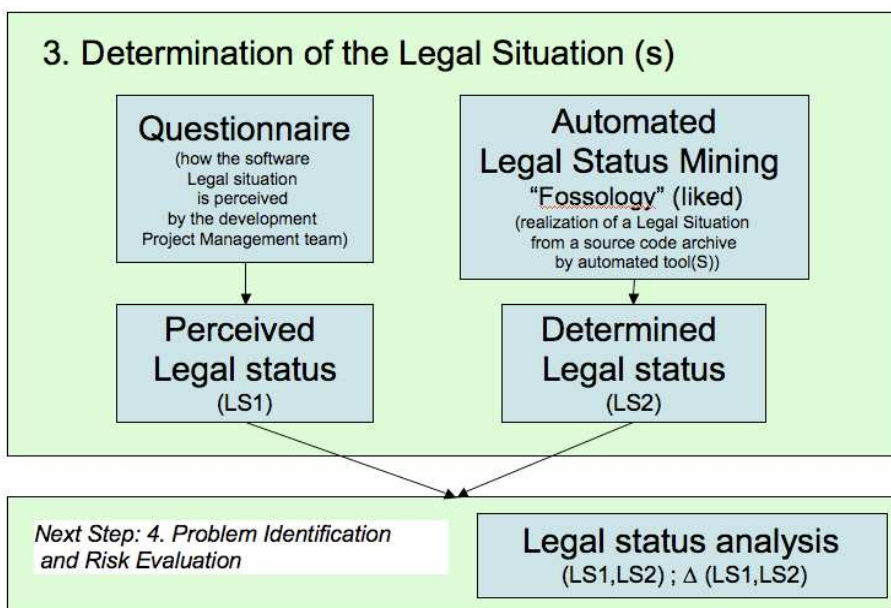


Figure 12: Step 3. Legal Status determination.

As said in a previous section, Software’s description is a great help to define the perceived legal status LS1, as it enables to collect a minimum of information that should be analyzed by the Audit team.

A questionnaire is then given to the Audit team to establish a “perceived” legal status LS1. Annex 1 presents a proposal for such legal status template, as well as a flowchart-questionnaire,

¹⁶ The double licensing scheme supposes an IPR centralization (in fact a partnership among French Academics or French research entities.)

and guidelines and illustration to help the Audit team to fill the template. The aim is to gather the necessary information to establish the “perceived” legal status.

The template is aiming at qualifying the Software and gathering information about the software content which is related to legal status’ elements presented in section 2.2. Such information concerns, for example, modules that compose Software or its different functional zones, the authors of the modules or components, the legal information concerning the legal conditions of production of the components (such as research contracts), the licenses attached to them. The objective is to complete and fill in the four categories that compose and allow defining the Legal Status.

This “perceived” LS1 allows to confirm if Software has been developed accordingly with the exploitation, control or enforcement intentions, or to identify one or several LS1’s elements which are not compatible with these intentions.

However, it possible that this “perceived” LS1 is incomplete or inaccurate for various reasons. For example, the notification of an external component may have been forgotten.

The DIET case:

Figure 13 shows the perceived legal situation by the development team. It consists of :

- The high level functional areas description with a description of the links among the components, and the perceived license attached to each component.
- The owners and authors of each component
- 3 research contracts under which some modules were produced

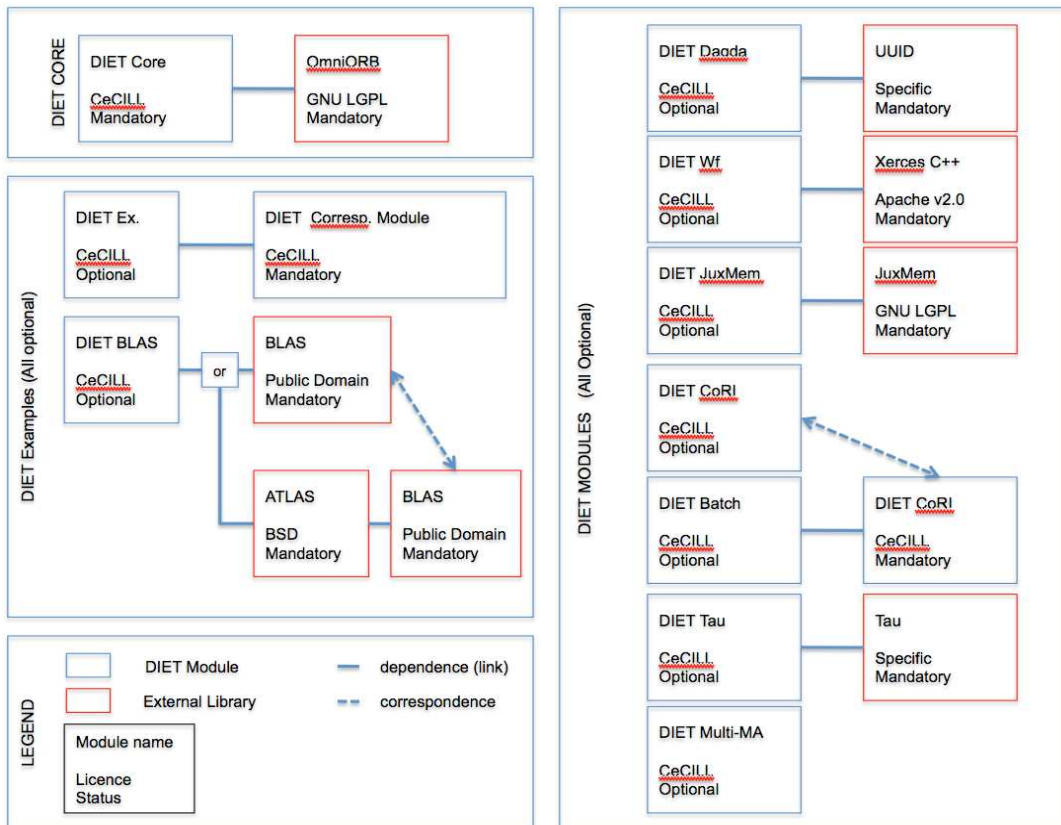


Figure 13: DIET perceived legal Status

The developers provided additional information such as the list of authors and related information (position, employee or interns, etc...)

4.3.2 Determined Legal Status

In order to determine the Software legal status, it is necessary for the audit team to further analyze all information gathered on Software. It is also highly recommended, in order to establish a determined legal status, to “scan” Software using mining tools such as Fossology. Such tools allow to discover the existence of unidentified licenses.

Each “perceived” or determined” LS can then be compared and discussed. The possible differences between LS1 and LS2 are also studied. The LS resulting from this analysis becomes the base of problem identification.

The DIET case:

In the DIET case, the determined legal status did turn out to be different from the perceived legal status:

- a component license had not be fully analyzed;
- questions were raised concerning one of the authors’ status and affiliation;

- the three research contracts that supported the development of DIET were gathered;
- an availability search on the name DIET Solve was made by trademark attorneys, in order to check for any prior trademark (or another prior distinctive sign);
- FOSSology was used to look for any unidentified licenses (for INRIA's components as well as pre-existing components);

4.4 Step 4 : Problem Identification and Risk Evaluation

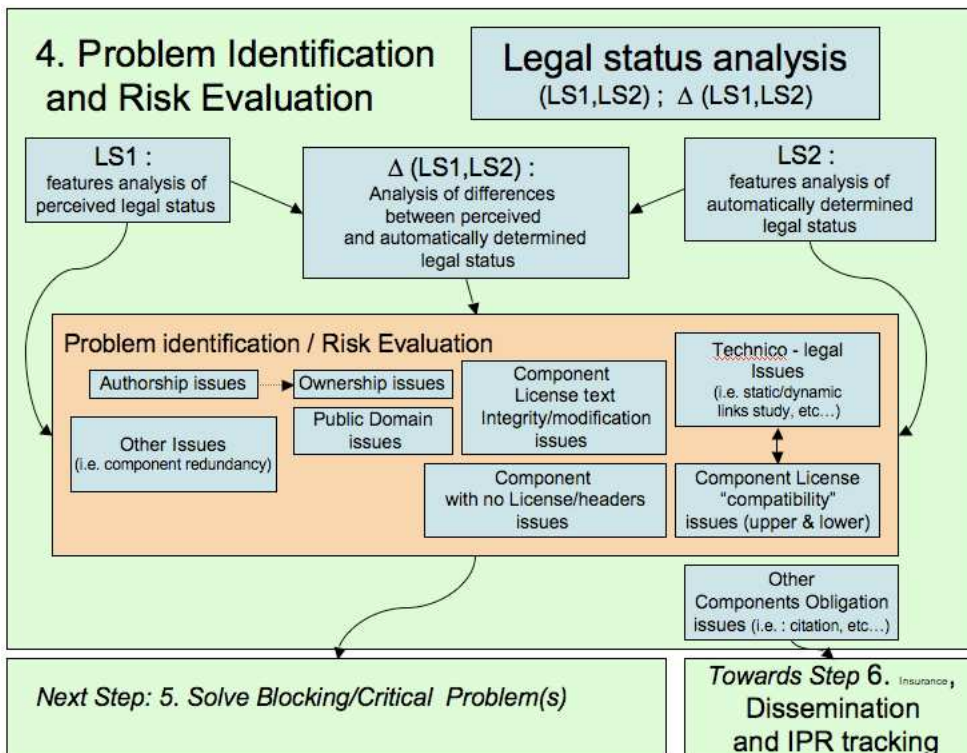


Figure 14: Step 4. Problem identification and risk evaluation

As mentioned in step 2, the comparison between the exploitation, control and enforcement goals and the LS enable to determine if Software actually is compliant with these goals. In case one or several elements of the LS are not, this comparison allows identifying what are the problems and obstacles.

Legal status allows to identify the following problems, in particular:

- Authorship problems when author are not identified.
- Ownership problems when relationships (if any) between author and employers are unclear.
- Component License text problems: component without license or with unclear or incomplete legal notice (notice such as “all rights reserved”).
- Component with unclear or fragmentary license identification or name.
- Undefined version of the license.
- Incompleteness of the license text (text modification - lack of integrity)

- Incompatibility problems between the licenses attached to different components.
- Technical-legal problem
- Other problems such as packaging problems: component redundancy, component present in the packaging, but unused in the software, etc...

Once these problems are identified, the Audit team tries to solve the problems when this is possible, or identifies otherwise those remaining critical, as explained hereafter.

In the context of DIET Case, the analysis showed:

- License incompatibility between components under CeCILL v2 license and components under GNU LGPL licenses;
- Component TAU's license was unclear;
- One of the authors's status is unclear and prevents having a reliable list of patrimonial rights owners on Software;
- The trademark scanning search revealed that DIET was not use in the exploitation domain of DIET software.

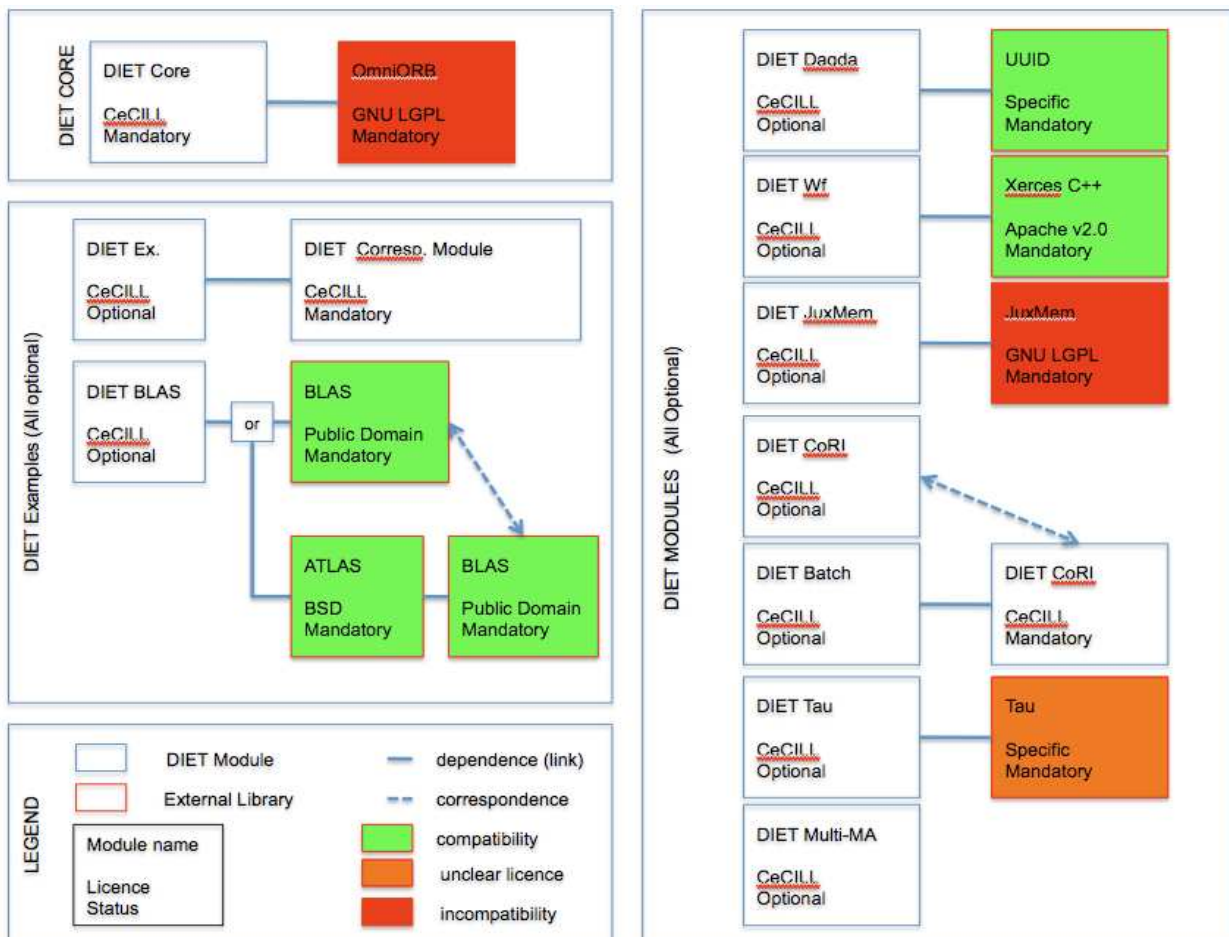


Figure 15: Blocking problems identification for DIET

4.5 Step 5 : Solve Blocking/Critical Problem

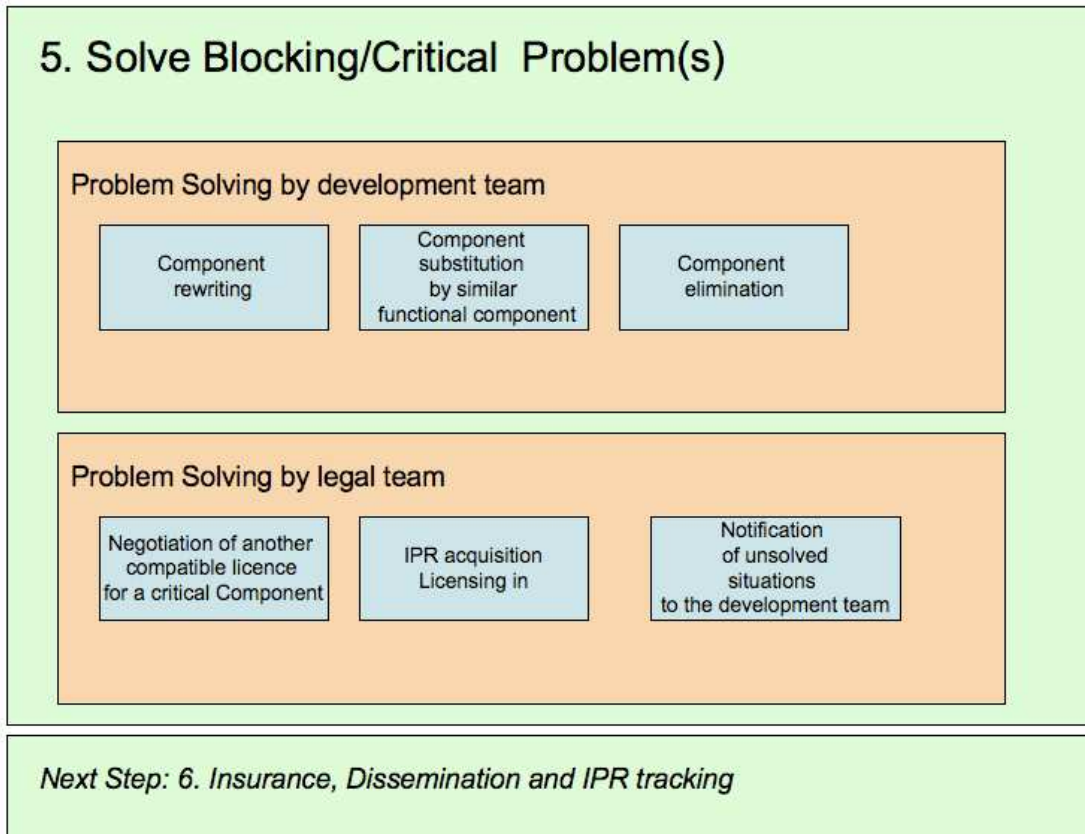


Figure 16: Step 5. Solve Blocking/Critical Problem.

A critical problem is a problem that is blocking the exploitation of the software. It can be, for example, the existence of a component developed by an author under a proprietary license, which would therefore be incompatible with the distribution of the Software in which it is integrated under a GNU GPL license.

Critical problems are solved by the development team by taking appropriate technical or legal actions, whenever possible.

Problem solving by developers

The developers can solve problems, each time it is possible for the team to take an appropriate action on a “critical” component (which license is incompatible with license scheme). This can be done by:

- component elimination (when such component is not technically necessary);
- component rewriting;
- component substitution by other existing component having similar functionalities but different legal status (compatible with the software exploitation plan);
- distributing the component separately from Software.

Problem solving with legal action

Problems can also be solved with appropriate legal actions, when they cannot be solved by developers or when it seems more appropriate (for example, regularizing the situation between joint owners of a critical component can sometimes be less time consuming than rewriting the component). For example, this can be done by:

- Negotiating another license on a critical component with its owner, which is compatible with Software's exploitation plan;
- Negotiating an assignment of IPR on the critical component;
- Provide the component's license with an exception.

Unsolved situation might however occur and lead to project stopping, unless it is decided that the risk should be taken, which remains both the Software owner's choice and responsibility.

The DIET case:

In the context of DIET, the following solutions were found:

- Incompatibility between CeCILL v2 license and GNU LGPL license did not turned to be a real problem:

The development team was aware of the license incompatibility but knew that it could be avoided by distributing the GNU LGPL components separately.

Another solution (a legal solution) was identified: it also turned possible to add an exception to the CeCILL v2 license to allow linking with the GNU LGPL components in DIET, as INRIA has the required IPR to do so. This solution turned out to be the simplest.

- TAU's license:

INRIA's lawyers contacted TAU's owners in order to have confirmation that TAU's license is a BSD-like license. This was confirmed.

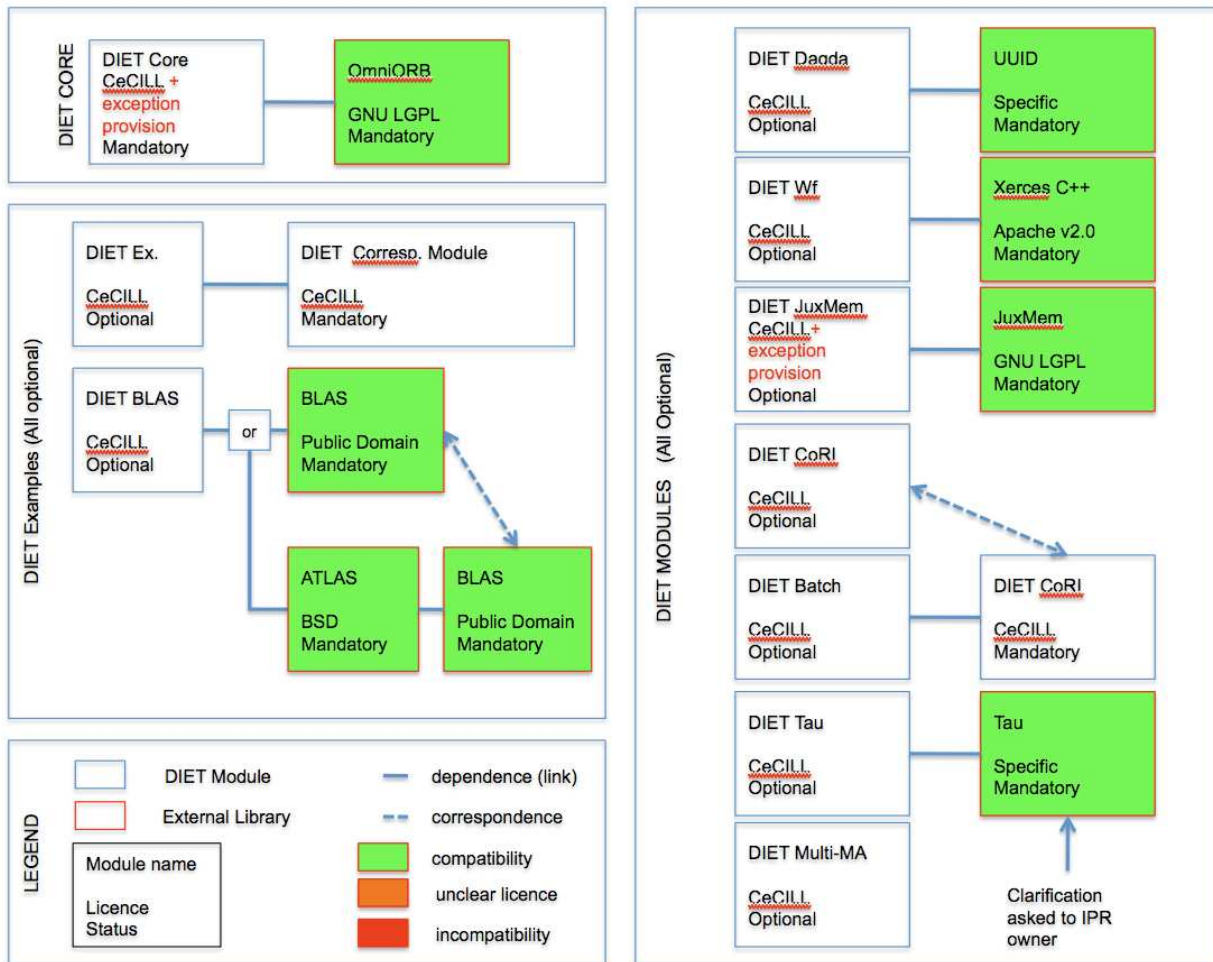


Figure 17: Final legal situation of DIET after problem solving.

4.6 Step 6 : Insurance, Dissemination and IPR tracking

It is sometimes not possible to identify the origin of a component or its authors. Residual risk can then be evaluated and covered by insurance.

Other obligations should be respected (citation obligations, etc...) and more generally obligations that have an impact on the release or on the documentation that would create conflicts if not respected. For example, some licenses impose citation obligations of different nature that must be tracked and respected.

Moreover, these issues should also be considered as good practices with respect to IPRT issues. Good packaging practices are also to be formalized with respect to IPR Tracking issues, for example:

- Packaging: clear identification of third parties components, filing by type of licenses, etc... For example, some headers may not clearly indicate the component's license or may differ from what is required by a particular license. Such requirements can be part of good practices (for

example, respecting headers' integrity) and others may be specific obligations required by some licenses, which can become difficult to respect in the case of Software made of numerous preexisting components.

- Documentation: appropriate treatment of the Software documentation is also part of the IPRT, as the Legal Status of the Documentation attached to components is to be studied and respected (for example, the fact that a component documentation is released under a creative commons license).

5 Conclusion

A general framework to tackle the issue of the legal quality of software is proposed. This framework is based on a rather simple model of coupling a “licensing out” process – exploitation intention – with a “licensing in” process – pre-existing code reuse.

However, component-based and collaboratively developed software are rather complex object to describe from the legal point of view. We propose in this report a definition of what we call the “legal status”, a concept we introduced, in order to determine this complexity.

The idea is to reach a legal status that is compliant with the exploitation intentions of the component-based and collaboratively developed software.

This legal status is the output of an intellectual property rights tracking process.

The Intellectual Property Rights Tracking framework and audit methodology is proposed to design such process.

This approach was applied to several cases and has already provided interesting results.

The methodology we applied at INRIA is a FOSSology assisted process, as license checking of reused components and component license integrity verification duration and costs are dramatically improved with this open source tool.

This contribution to the software legal quality debate has to be discussed and further enriched and could lead to an interesting standardization process to be set up. We think it opens space for automatically handling or tool assisted treatment of legal quality monitoring of component-based and collaboratively developed software.

However, a strong need of standardization is shown.

This standardization need is a growing concern of open source communities, see for example Debian or eclipse initiatives in related fields¹⁷.

¹⁷ Debian <http://dep.debian.net/deps/dep5/>
Eclipse http://www.eclipse.org/projects/dev_process/project-log.php

Up to now, the following feed-back, based on the cases the IPRT Methodology was applied to, can be provided:

- Although it is a key issue, especially in the context of collaboratively developed software, analyzing the various contracts in the context of which software was developed can turned out to be a delicate task, especially when software development last for decades: the chain of contracts may not be always very clear and an effort (which can be time consuming) has to be done to identify which part of software has been developed in the context of each contract (and therefore to which part of software should IPR clauses eventually apply). This is especially true for academic institutions, which are very familiar with bilateral and collaborative research contracts.
- Although we believe FOSSology to be today the most or one of the most efficient license checker, the analysis of results can be time consuming, especially for software with numerous files: some files may be identified twice in Fossology's results (for example when a file contains source code submitted to two different licenses), which may be misleading or confusing if the audit team is not aware of it. Moreover, FOSSology does not make an automatic link between a file, the component which uses the file and its location in software. This has to be made by hand by the audit team and can therefore take a long time. This is not du to FOSSology intrinsic drawbacks, but du to the poor quality of the header or legal notice or information encountered in source code files.
- The standardization priority should to focus on headers information content and structure.

We believe however, that if the IPRT analysis would be much efficient and cheaper if clear legal development roadmap (licensing out and licensing in policies) is formalized and implemented as soon as possible during the development lifecycle and if legal data are gathered or generated along the development process along with the software's documentation, reducing the audit phase to a simple verification phase.

Annex I: Toolbox for legal status drafting

Legal Status Template

Figures 18 to 20 illustrate the form to complete the Legal Status, as well as the reasoning the audit team has to follow when analyzing the different software components. Figure 18 is the form which is currently used at INRIA (in particular for the DIET case). Finally, the last document of this annex is a guideline to help the audit team fill out the form.

Document 1: LS form

1. Position in the chain of rights: initial software derived software components-based software

For each functional zone

High level description functional zone											
Functional zone's licensing scheme											

For each component in a specific zone

name	Version #	location	nature	licence	Integrity (Licence)	Composition rule (nature of the link)	Compatibility	Packaging constraint	comments
					0/1				

2. Owner of intellectual Property Rights On Software

Moral rights

Author's name	Affiliation (Employer, ...)	Affiliation link (work contract,...)	Contribution nature	comments
			spec. code doc	

Patrimonial rights

Organisation	Licensing Contact name	comments

3. Legal conditions of exploitation

Restricting Licenses	
Other restricting Agreements	
Restricting laws	
Other binding rule or legal provision	

4. Other enforceable IPR against software

Patent	
Trademark	
Copyright	
Database	

Figure 18: Template used to determine perceived Legal Status

FLOWCHART

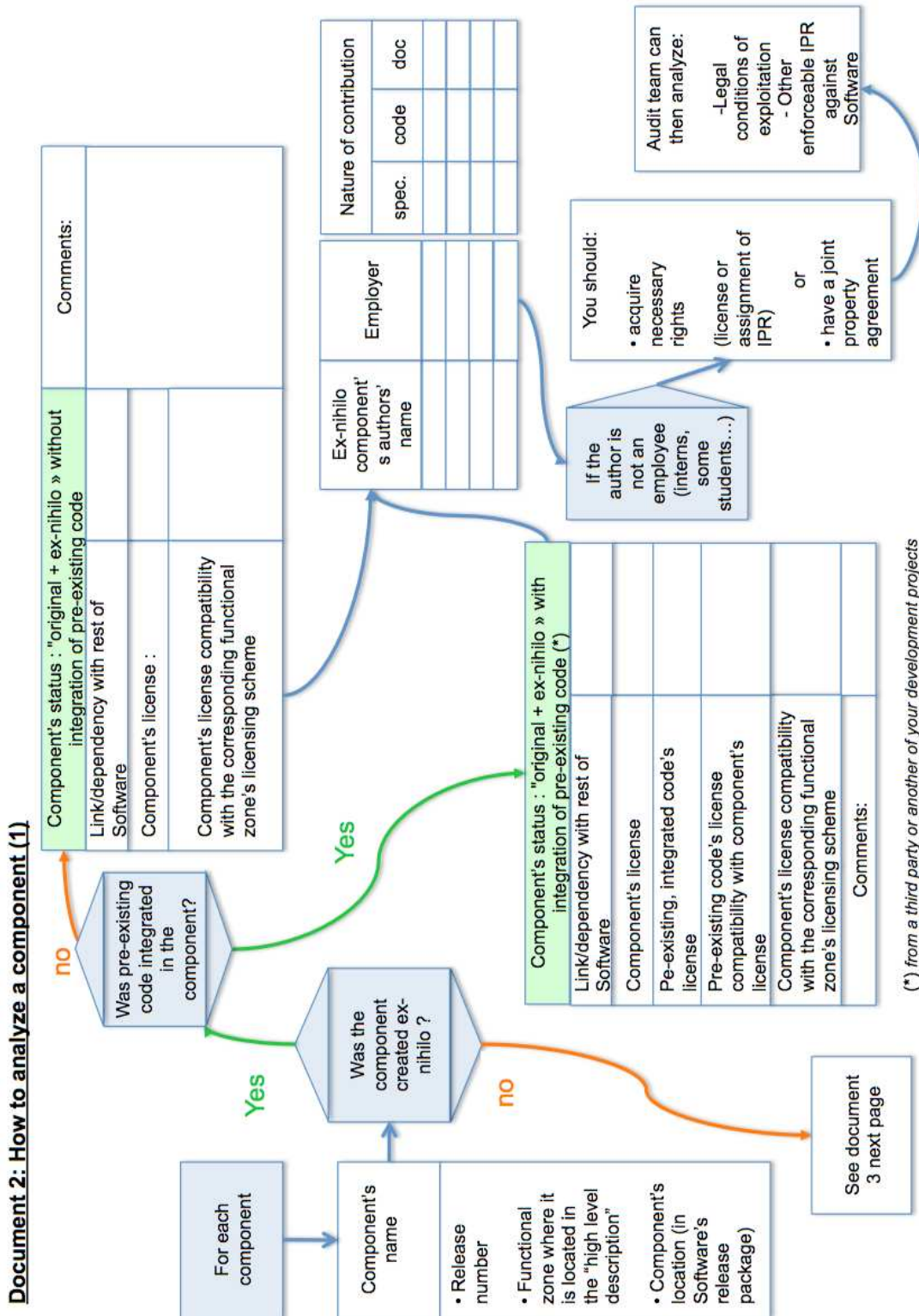


Figure 19: Flowchart used to complete the Legal Status of DIET (Part 1)



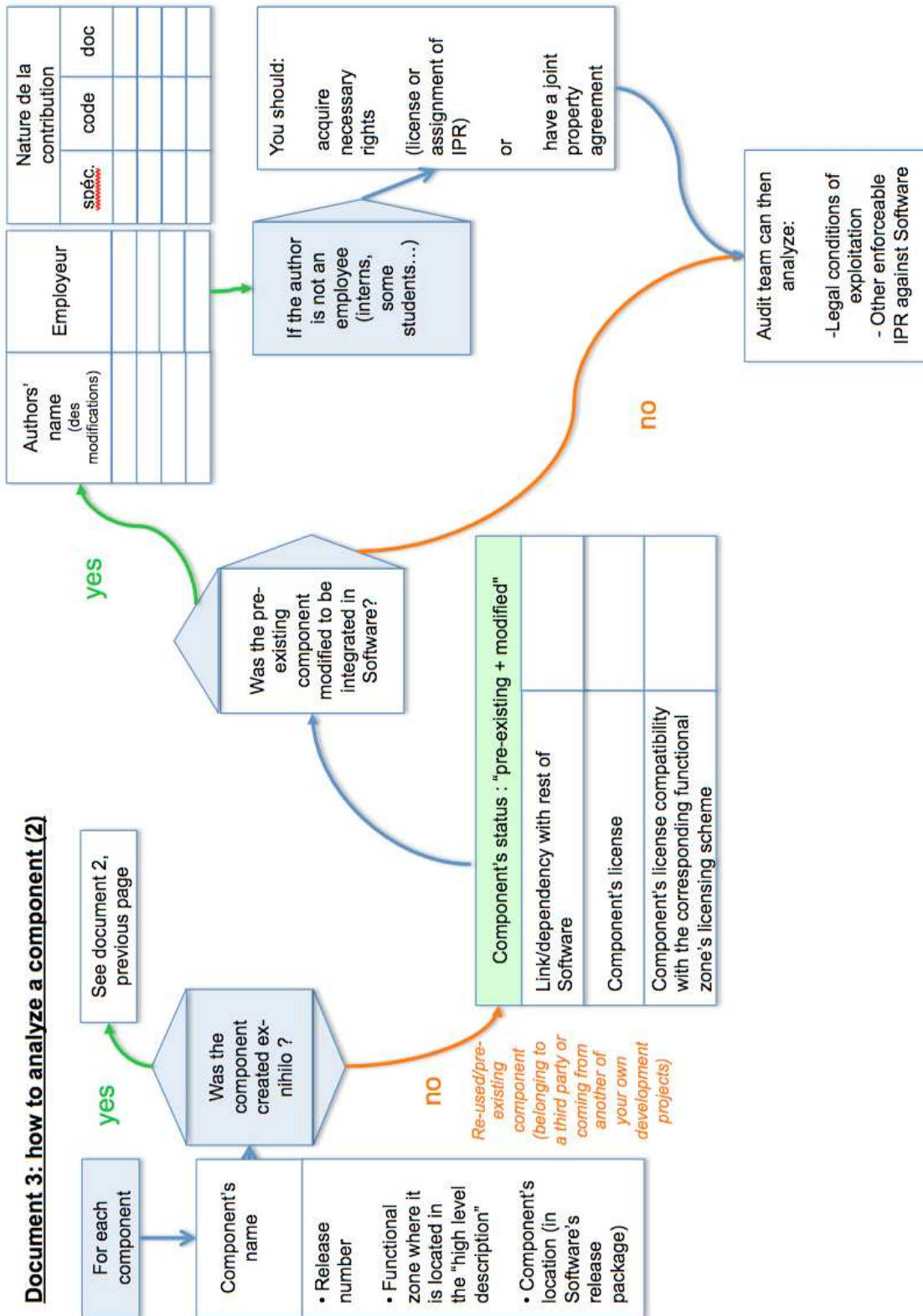


Figure 20: Flowchart used to complete the Legal Status of DIET (Part 2)

Structured template (text format) of the Legal Status

X Software analysis

1. Position in chain of rights

Initial software	
derived software	
components-based software	

High level description functional zone:

High Level description functional zone:	
Functional zone licensing scheme:	

[Name of functional zone]'s components:

name	version	location	nature	License of the component <i>(Licensing in)</i>	License Text integrity	composition rule	Compatibility <i>(with proposed Licensing out)</i>	packaging	comments

2. Owners of IPR on Software:

Moral rights:

Author's name	Affiliation	Affiliation link	Contribution			Comments
			spec	code	doc	

Patrimonial Rights

Organisation	Licensing contact name	Comments

3. Legal Conditions of exploitation:

		comments
Restricting Licenses		
Other restricting agreements		
Restricting laws		
Other binding rule or provision		

4. Other enforceable IPR:

Patent	
Trademark	
Copyright	
Icons/Images	
Fonts	
Database	

Guidelines – Legal Situation Questionnaire

The Legal Situation (LS) form (document 1) is a resume of all the information which enables to determine Software's LS, as well as the adequacy of this LS to Software's licensing scheme(s).

To fill out this form and determine the LS, the audit team needs, in particular, to examine each component part of software, individually.

In order to help the team examining each component, and find the appropriate information to complete the LS form, it may use document 2 and document 3: they indicate the reasoning that should be followed and the questions the audit team should ask itself, when examining a component part of Software.

Position in the chain of rights :

This section enables to identify, from a legal point a view, if Software is related with any previous software.

Initial software: software developed without being based or integrating previous software.

Derived software: software which original source code has been modified.

Component-based software: software made with or of pre-existing components.

Your software may be both derived and component-based software.

Information required hereafter, in section 1, should provide the audit team with what we call a "high level description" (a detailed description) of Software.

For each functional zone:

A functional zone corresponds to a specific and identifiable part of the software (for example: kernel, core, tools, GUI, ...).

High level description functional zone: describe the functional zone. Refer to a document representing and describing the functional zone if any.

Licensing scheme: indicate what kind of exploitation/distribution you wish to have for the components part of this functional zone (depending on your intention, it may be the same scheme for all software's functional zones or there may be a specific scheme for each, one or several functional zones. For example: I want to distribute my software's core under a GNU LGPL license but I want to distribute the plug-ins under a proprietary license).

For each component of a specific functional zone:

Name: indicate the component name

Version: indicate the component version number

Location: indicate where the component is available in the release package (path) or its url if it is located on a remote server, or equivalent information in case of other technical situation.

Nature:

Indicate if:

- the component was created ex-nihilo, without integrating pre-existing code (pre-existing code belonging to a third party or belonging to you, but coming from a different software)
- the component was created ex-nihilo and integrates pre-existing code.
- the component is a pre-existing component, which has not been modified by Software's authors.
- the component is a pre-existing component which has been modified by Software's authors.

License: indicate wherever the component is submitted to a particular license.

Integrity: indicate wherever the license's text has been modified (example: the license is called GNU GPL, but the text of the license is modified or the owner of the component added an exception).

Composition rule: indicate the type of link/dependency existing between the component and the rest of software (example: dynamic linking).

Compatibility: compatibility between the component's license and software's licensing scheme.

Packaging: indicate wherever the component's license has obligation in relation with software's packaging, you should comply with or if you need to change the component license (when it is allowed by it) to comply with the exploitation scheme of the Software.

Comments: you may wish to indicate additional information, such as a problem you identified (example: name of component's owner if it is a pre-existing component) or that you may need for further action related to the component.

Owner of intellectual property rights:

We are talking here of IPR ownership on Software as a whole, and not on the components taken individually.

Moral Rights:

Author's name: indicate the author's full name. For the purpose of this form, an author is a person who contributed to the Software by creating new code, documentation and/or specification. Authors of pre-existing code/components do not need to be mentioned.

Each time you identify an author of “new code”, while using document 1 and document 2’s framework to analyse Software components, you should add him/her in this list.

Affiliation: indicate if the author is affiliated to a particular organisation (example: your company, a third party’s company, a university...).

Affiliation link: indicate how the author is affiliated (work contract, as a student, an intern...).

Contribution’s nature: indicate wherever the author contributed by developing code and/or writing specifications and/or writing documentation.

Comments: you may wish to indicate additional information (for example: if you need to acquire exploitation rights on the component or if the same component is used in different functional zone, you may mention it).

Patrimonial rights:

Organisation: indicate which organisation is the owner of patrimonial rights on software.

Licensing contact name: indicate the name of the authorised person to be contacted in this organisation, preferably in a licensing department.

Legal conditions of exploitation:

Restricting licenses: indicate the different licenses (FLOSS, proprietary license...) which are restricting.

Other restricting agreements: indicate if an agreement, such as a research contract, consortium agreement, a joint property agreement, a confidentiality agreement, or any contract in relation with Software exists. If such an agreement exists, indicate it has clauses (in particular IPR clauses) which could restrict Software’s use.

Depending on these other restricting agreement, you may identify new patrimonial rights’ owners you were not aware of (for example: a research contract according to which the parties are joint owners on all results, wherever their employees are authors or not of such results) and therefore complete the list in the previous section.

Restricting laws: indicate if there are any laws existing which may restrict Software’s use or creating obligations you should comply with (example: if Software is to be used in France, and that this Software collects IP addresses when it is used: you are required by French Law to be previously granted permission to collect IP addresses by a public authority called CNIL).

Other binding rule or legal provision: you may be aware of another binding rule or legal provision, such as jurisprudence, court decision or ruling, regulation, certification or standards to comply with, for example.

Other enforceable IPR against component's:

Patent: indicate if software uses any identified patent (wherever it is your patent or a third party's patent) and indicate wherever you have the necessary rights to use it (if the patent belong to a third party, you may need a license, and if the patent belongs to you and a third party, you may need the joint owners approval, or if you are the only owner of the patent, you should check, for example, if you have not granted an exclusive license on the patent).

Trademark: if you wish to distribute and/or use the software under a particular name, indicate if this name is not a registered trademark or not.

Copyright: indicate if any copyright can be enforced against software (example: parameters files, copyrights on icons, pictures in documentation) and if you have the necessary exploitation rights.

Database: indicate if a database is included in or distributed with software and if you have the necessary rights to do so.

The DIET case: detailed legal Situation :

Annex 1 THE DIET CASE: This part of annex 1 presents a detailed legal status of DIET

Audit juridique du logiciel DIET:									
1. Position in the chain of rights:									
Initial software									
derived software									
components-based software	X								
High level description functional zones:									
High Level description functional zone:	DIET core								
Functional zone licensing scheme:	double licence: sous CeCILL v2 et licence propriétaire								
DIET CORE's components:									
name	version	location	nature	licence (licensing in)	integrity	composition rule	compatibility	packaging	comments
Diet core		Noyau	Composant développé ex-nihilo sans intégration de code préexistant	CeCILL		Dépendance avec OMniORB	Oui	licence CeCILL: accompagner la distribution d'une copie de la licence + avertissement de l'absence de garanties	
OmniORB		Noyau	librairie externe préexistante d'un tiers	GNU LGPL v2		Dépendance avec Diet Core	Oui avec le composant DIET Core sous CeCILL Modifiée Incompatible avec une distribution propriétaire de ce composant	indiquer que DIET est en partie basé sur OmniORB + copie de la GNU LGPL	Incompatible avec la licence CeCILL non modifiée. <u>Solution technique</u> : distribuer OMniORB séparément. <u>Solution juridique</u> : dans la mesure où l'INRIA est titulaire des droits sur DIET, on peut prévoir une exception à la CeCILL, permettant le lien avec un composant sous LGPL. Licence propriétaire: exclure le composant OmniORB de la liste des composants objet de la licence

Audit juridique du logiciel DIET:

1. Position in the chain of rights:

Initial software	
derived software	
components-based software	X

High level description functional zones:

High Level description functional zone:	DIET modules
Functional zone licensing scheme:	double licence: sous CeCILL v2 et licence propriétaire

DIET Modules' components:

name	version	location	nature	licence	integrity	composition rule	compatibility	packaging	comments
DIET DADGA			Composant développé ex-nihilo sans intégration de code préexistant	CeCILL v2		dépendance avec UUID	Oui	licence CeCILL: accompagner la distribution d'une copie de la licence + avertissement de l'absence de garanties	
DIET Wf			Composant développé ex-nihilo sans intégration de code préexistant	CeCILL v2		dépendance avec Xercès C++	Oui	licence CeCILL: accompagner la distribution d'une copie de la licence + avertissement de l'absence de garanties	
DIET JuxMem			Composant développé ex-nihilo sans intégration de code préexistant	CeCILL v2		dépendance avec Juxmem	Oui	licence CeCILL: accompagner la distribution d'une copie de la licence + avertissement de l'absence de garanties	
DIET CoRI			Composant développé ex-nihilo sans intégration de code préexistant	CeCILL v2		Composant essentiel à DIET Batch	Oui	licence CeCILL: accompagner la distribution d'une copie de la licence + avertissement de l'absence de garanties	
DIET Batch			Composant développé ex-nihilo sans intégration de code préexistant	CeCILL v2		dépendance avec Diet Cori	Oui	licence CeCILL: accompagner la distribution d'une copie de la licence + avertissement de l'absence de garanties	
Diet Tau			Composant développé ex-nihilo sans intégration de code préexistant	CeCILL v2		dépendance avec Tau	Oui	licence CeCILL: accompagner la distribution d'une copie de la licence + avertissement de l'absence de garanties	
Diet Multi-MA			Composant développé ex-nihilo sans intégration de code préexistant	CeCILL v2			Oui	licence CeCILL: accompagner la distribution d'une copie de la licence + avertissement de l'absence de garanties	
UUID			composant pré-existant d'un tiers non modifié	Spécifique = il s'agit en réalité d'une licence MIT		librairie externe essentielle à DIET Dagda	Oui	En-têtes à préserver en l'état	
Xercès C++			composant pré-existant d'un tiers non modifié	Apache 2.0		librairie externe essentielle à DIET wf	Oui		
JuxMem			composant pré-existant d'un tiers non modifié	GNU LGPL v2		librairie externe essentielle à DIET JuxMem	Oui avec le composant DIET JuxMem sous CeCILL Modifiée	En-têtes à préserver en l'état + indiquer que DIET est en partie basé sur JuxMem + copie de la GNU LGPL	Incompatible avec la licence CeCILL non modifiée. Solution technique: distribuer Juxmem séparément. Solution juridique: dans la mesure où l'INRIA est titulaire des droits sur DIET, on peut prévoir une exception à la CeCILL, permettant le lien avec un composant sous LGPL. Licence propriétaire: exclure le composant JuxMem de la liste des composants objet de la licence
TAU			composant pré-existant d'un tiers non modifié	Spécifique		composant essentiel à DIET TAU	BSD-like	En-têtes à préserver en l'état = les mentions de la licence de TAU doivent apparaître dans la documentation, le cas échéant.	Confirmation demandée au titulaire de TAU: la licence est bien une BSD-like - Pour le contrat de licence propriétaire, exclure l'exclusivité pour ce composant.

	A	B	C	D	E	F	G	H	I	J
1										
2	Audit juridique du logiciel DIET:									
3										
4										
5	1. Position in the chain of rights:									
6										
7	Initial software									
8	derived software									
9	components-based software	X								
10										
11	High level description functional zones:									
12										
13	High Level description functional zone:	DIET Examples								
14	Functional zone licensing scheme:	double licence: sous CeCILL v2 et licence propriétaire								
15										
16										
17	DIET Examples's components:									
18										
19										
20										
21	name	version	location	nature	licence	integrity	composition rule	compatibility	packaging	comments
22	DIET Ex			Composant développé ex-nihilo sans intégration de code préexistant	CeCILL v2		Dépendance avec DIET Corresp module	Oui	licence CeCILL: accompagner la distribution d'une copie de la licence + avertissement de l'absence de garanties	
23	DIET BLAS			Composant développé ex-nihilo sans intégration de code préexistant	CeCILL v2		Dépendance avec BLAS et ATLAS	oui	licence CeCILL: accompagner la distribution d'une copie de la licence + avertissement de l'absence de garanties	
24	DIET Corresp module			Composant développé ex-nihilo sans intégration de code préexistant	Non indiqué: CeCILL v2?		Elément essentiel de DIET Ex			
25	BLAS			composant pré-existant non modifié	Domaine Public		Elément essentiel de DIET BLAS et d'ATLAS	Oui	Préserver le nom des auteurs dans les en-têtes s'ils sont présents.	
26	ATLAS			composant pré-existant non modifié	BSD		Elément essentiel de DIET BLAS + dépendance avec BLAS	Oui	Préserver les en-têtes et la liste des conditions	
27										

Audit juridique du logiciel DIET:

2. Owner of IPR on Software:

Moral rights

Author's name	Affiliation	Affiliation link	Contribution			Comments
			spec	code	doc	
x	INRIA	postdoc	X	X	X	
y	Co-owner 2	ingénieur	X	X	X	
z	Co-owner 2	doctorant	X	X	X	
t	Co-owner 3	Post-doc		X	X	
r	Co-owner 4	Assistant professor	X	X	X	
s	Co-owner 3	Assistant professor	X	X	X	
q	INRIA	ingénieur	X	X	X	
p	INRIA	DR	X		X	
u	Co-owner 2 or co-owner 5	stagiaire	X	X	X	Droit du stagiaire acquis? Quel statut chez le Co-owner 2
v	INRIA	ingénieur	X	X	X	
w	Co-owner 6	doctorant	X	X	X	
o	Co-owner 7	doctorant	X	X	X	
n	Co-owner 7	Assistant professor	X	X	X	
m	Co-owner 7	Professeur	X	X	X	

Patrimonial Rights

Organisation	Licensing contact name	Comments
INRIA		
Co-owner 2		
Co-owner 3		
Co-owner 4		
Co-owner 5		Dépend du statut du stagiaire +Vérifier les accords avec le Co-owner 2 sur la propriété des travaux
Co-owner 6		
Co-owner 7		

DIET IPR owners (moral and patrimonial rights)

Audit juridique du logiciel DIET:					
3. Legal Conditions of exploitation:					
		comments			
Restricting Licenses	GNU LGPL (incompatibilité avec CeCILL) plusieurs solutions possibles	Modules sous GNU LGPL: problème à régler soit par une solution technique (distribution séparé des composants gênants) soit par une solution juridique (en prévoyant une exception à la CeCILL).			
Other restricting agreements	Contrat 1 + Contrat 2 + Contrat 3	les dispositions du CPI sont à chaque fois appliquées: les titulaires sont donc bien les employeurs des auteurs à chaque fois, sans qu'un droit ne soit concédé aux autres partenaires, si ce n'est ceux de la CeCILL	Contrat 1 : (pas d'accord de consortium): on applique par défaut le CPI	Contrat 2 : accord de subvention uniquement: à défaut on applique le CPI	Contrat 3 : on applique le CPI, car rien de particulier de prévu
Restricting laws	Non				
Other binding rule or provision	Non				

Audit juridique du logiciel DIET:	
4. Other enforceable IPR:	
Patent	non
Trademark	recherche d'antériorités sur DIET: antériorités connexes relevées ==> risque à évaluer lié au domaine d'utilisation de ce nom
Copyright	non
Database	non

ANNEX 2: EU PALETTE CASE STUDY

Introduction:

QualiPSo IPR tracking methodology was used by “ Pedagogically sustained Adaptive Learning Through the exploitation of Tacit and Explicit knowledge ” (PALETTE) European Project steering committee to determine the better open source licensing strategy for the six software services developed within PALETTE. This was the first case on which the IPRT Methodology has been testes and applied on.

Further to this first case, an attempt and efforts were made to improve the documents used in the context of the IPRT. The methodology itself has of course evolved since this first version of this QualiPSo D1.4.1 report on the proposed IPR tracking methodology.

IPRT and EU PALETTE Project:

Step 1 : High level description of the PALETTE services and audit objectives

PALETTE is coordinated by ERCIM. EPFL a PALETTE partner prepares the Exploitation Plan of PALETTE services. A first “draft Exploitation plan” was released.

PALETTE steering committee had a demand to work on “legal quality” of the software services developed within PALETTE and got in touch with QualiPSo people involved in the “Legal Issues” activity for that purpose. The “Final Open Source Strategy” deliverable of PALETTE aims at providing indicators in order to ensure the feasibility of the Exploitation Plan/ Business Plan for PALETTE services.

The objective of the collaboration with the QualiPSo project is to easily describe and audit the different services developed within PALETTE project, to know if their licensing constraints fit with the Exploitation Plan defined by the Consortium.

The results of the last QualiPSo methodology based analysis will be integrated in the “Final Exploitation Plan” deliverable due at the end of the PALETTE project.

Step 2: Preparation of a questionnaire for “perceived legal status determination”

In order to handle this Audit, the INRIA team involved in QualiPSo project built a questionnaire to collect related information.

The first part of annex 2 presents the analysis of QualiPSo questionnaire

This questionnaire has been sent to the “development managers” of the PALETTE services. The services which were analysed are the following:

Collaboration Services: CoPe_it! and eLogbook

Knowledge Management Services: SweetWiki, ECCO, Generis,

Information Services: Amaya, LimSee3, DocReuse, Palette Web Portal

Step 3 Determination of “Perceived legal Status” for any PALETTE software and additional “FOSSology Scanned” Legal Status for Bay Fac Software

A first analysis of the questionnaires was provided end of December 2007. This questionnaire is available in the previous version of this deliverable. However, we would consider annex 1 of this report to be an updated and revised version of the Questionnaire.

The questionnaire is composed of two parts: “Code and components” and “Contractual context and Peripheral IPR”.

First part: Code and Components

- Almost all components are based on a large number of external components
each external component must be clearly identified
- This is not necessarily the case for all tools and services
Some of them induce potentially strong constraints on possible exploitation scheme.
- Identify carefully if the link between the external component and the tool induces contamination from GPL-like components
Some of the components have been modified.
- Open question about integration of modified components into release of PALETTE tools or contribution to original community (with leading question about maintenance of modified/forked version)

Second part: Contractual context and peripheral IPR

- Contractual context
Pre-existing know-how (PKH) has been identified
- Check if the initial version of the PKH (before modification by PALETTE) is identified .
- Check if all IPR owners are members of the PALETTE consortium
- Check how PKH must be interpreted according the consortium agreement
-
- Global exploitation / dissemination scheme
Some of the PALETTE services already appear to have different dissemination strategy from GPL based to more permissive one based on MIT or LGPL licenses
PALETTE project aims to provide an interoperable and extensible set of innovative services. However, the questionnaire answers show or helped to :
- Open question about heterogeneity of licensing scheme for different services
- Lack of description on high level architecture implementing “interoperability” (ie what will be the nature of the link between the services)
Global vs. local dissemination scheme from the point of view of PALETTE objectives,
- Define objectives and priority in term of dissemination
- Improve understanding at services level of the possible compatibility problem with global

dissemination strategy

- Identify blocking/critical point
- According to resources available, roadmap, and priority, consider to solve some of the problems identified (for instance substituting a component under contaminating license), modifying the global interoperability scheme at the technical, legal or dissemination level

After this first analysis, the Bay Fac software development manager had decided to do an “automated legal status mining of Bay Fac” software.

This work was done using FOSSology license checker and lead to the Legal Status LS₂^{1 (Bay Fac)}.

Step 4: Problem identification and Risk evaluation.

This “automated legal status mining of Bay Fac” was compare to the LS₁^{1 (bay Fac)} “perceived” legal status of the development team and discussed with it.

No critical problem was identified for Bay Fac software and risk to distribute it under a GNU GPL license was considered to be very low.

An example of non critical problem detected by using FOSSology is the presence of the Bay Fac software archive used for the audit of a component not mentioned by the development team. After discussion it appears that this component was not used anymore...

Step 5 : Solve Blocking problems

Detailed in the previous report (minor points).

Next phase for Palette projects:

The process used for Bay Fac Software has been extended to the other PALETTE software. This work is in progress. However, although this preliminary work is not completed yet, we may draw observations at this stage.

The answers to the QualiPSO questionnaire show rather important variations

- from a PALETTE service to another one, on the one hand
- in the policies of the different partners in favor of the Open Source Strategy, on the other hand.

In order to determine the perimeter of the audit and the methodology of traceability to be implemented, it is important to identify (via the Exploitation Plan Questionnaire for example) the characteristics of the communities of users to be set up.

According to their composition and to what is awaited from their members, it will be possible to identify the best adapted Open source license’s scheme (from the most permissive to the most contaminant).

It is also necessary to have one or more “global” architectural visions for the various services (representing the interoperability of services), which will have sense for the actors (users, service companies, software publishers, researchers, etc).

It will then be possible:

- to determine if the exploitation schemas expected for the PALETTE services are compatible (legal analysis to be done...)
- to identify if the “legal status” of the components allows these exploitation schemas.

For this, we need a description of this legal status on one hand (on the basis of answer to the questionnaires), and an analysis of the contents via dedicated tools for source code analysis (i.e. to extend FOSSology audit test to other PALETTE services). If problems are identified, we will be able to consider corrective measurements, or to adapt the exploitation schema.

In term of traceability, an infrastructure allowing the development activity is strongly advised (for example via the portal gforge.inria.fr). The rules of governorship of these communities will have to be the subject of a specific attention. If necessary, the IPR could be centralized, to simplify the evolution of the licensing schema.

Palette use case conclusion:

PALETTE consortium still needs to work in collaboration with QualiPSo project (people of WP 1.4), to go further in the audit of the source code of PALETTE services, once "packages" of the different services will be available and the desired license defined.

The full 6 steps IPRT methodology has been used on the BayFac software service (see figure xx). It leads to a clearly defined Legal Status for this software service in a quick process of only two iterations with the development team (10 work days). Components redundancy as well as an unused component presence in the release package were identified, leading to corrective actions and improvement of the release package quality.

In parallel, PALETTE consortium should decide on:

- a global interoperability scheme of the PALETTE services
- an exploitation scheme for each service
- the OSS community and governorship
- an infrastructure for traceability

At last, we will give indications on (i) the activities to follow (implementation of corrective actions or not) in term OS license choice, (ii) the strategy to follow in term of services exploitation/valorisation.

Figure 21: PALETTE IPRT Process

