UCL
Université catholique de Louvain

scouting

CREATING SOFTWARE AT THE UNIVERSITÉ CATHOLIQUE DE LOUVAIN

The Researcher's Guide

Louvain Technology Transfer Office

financing

scouting

maturation

commercialisation

L I E U
LIAISON
ENTREPRISES
UNIVERSITÉS

AUTHORS

Jérémie Fays (ULg)
Céline Thillou (UMONS)
Nathalie Poupaert (UCL)
Anne-Gaelle Peters (UCL)
Yves Laccroix (ADISIF)
Nathanael Ackerman (ULB)
Bernard Detrembleur (FUNDP)
Sébastien Adam (UCL)
Edgar Moya Alvarez (ULB)
Jonathan Pardo (UMONS)

CREATING SOFTWARE AT THE UNIVERSITÉ CATHOLIQUE DE LOUVAIN

The Researcher's Guide

V 1.4
June 2012

TABLE OF CONTENT

INTRODUCTION

We could not avoid using certain common terms from the IT field (open source*, software library*, source code*, etc.). A definition of these terms, followed by an asterisk in the text, is provided in the GLOSSARY at the end of the guide.

Many university researchers develop their own software, at first generally in order to meet their own needs, but then this software evolves and starts to draw the interest of other people, researchers or manufacturers.

This is when certain difficulties can arise:

- *practical problems for developing code in* **collaboration** *with several institutions*, lack of documentation, source management tools and communication tools (forums / wikis /mailing lists);
- *legal problems in connection with the* **ownership** *of the code* (the software belongs to several people and thus cannot be disposed of freely): this type of problem can prevent a laboratory from negotiating a research contract or marketing its software;
- *legal problems in connection with* **licenses***:* if snippets of open-source code distributed under incompatible licenses (for example GPLv2 and Mozilla 1.1) are integrated in the software, the resulting software cannot be distributed legally via a website, not even for free, not even with the source code*, and not even when only for scientific collaboration...

However, many of these problems can be overcome by addressing some questions during the early stages of software development.

This is the main purpose behind this guide, based on interviews with several researchers: to explain a series of best practices aimed at saving a lot of time if the software is ever developed.

Welcome to the first version of this guide. All suggestions are welcome for the next version! Also, please let us know if you have any experience to share concerning the development of IT projects, open-source licenses, collaboration tools for developers, software quality, etc.

Contact
LTTO - Louvain Technology Transfer Office
Sébastien Adam
sebastien.adam@uclouvain.be
+32.10.47.24.43

Who should read this guide ?

This guide is aimed at everyone in the University who is involved with software development: students, researchers, PhD students, as well as academics and promoters who sometimes supervise software development.

It is obvious that this guide only provides an overview of these fields, each of which could fill an entire volume. We were forced to cut out a lot of information, sometimes even intentionally in order to guarantee that the document was readable. If you would like to take this further or have any questions on the matter, please contact the LTTO for further advice.

USING SOFTWARE

It is quite common to start developing new software based on or using existing software, and to use existing software to run tests, compare results, etc.

ALL SOFTWARE FROM EXTERNAL SOURCES MUST HAVE A license, which defines the rights of the user to use/modify the software.

!

ALL SOFTWARE MUST HAVE A License. Otherwise, you must assume that you are not entitled to use it.

Before using the software, it is advisable to read the license and understand its effects. If software from an external source does not include a license, you must assume that you are not entitled to use it. It is advisable in this case to identify the copyright owners (the author, his/her employer) and ask them for a license. This can be done through your Knowledge Transfer Office (KTO).

In addition, commercial software available in student / academic / non-commercial versions (for free or at a discount price) generally includes limitations on the commercial uses to which it can be put.

If the software is produced by a partner in a project or by an identified third party, it is highly advisable to sign a license in order to define your user rights. Please contact your KTO for this.

If the license is a proprietary license and you have the option of choosing different competing software, it is advisable to choose software with an open-source license. Most software programs (editors, graphics, etc.) have open-source equivalents. You can find more information on this at Osalt website (http://www.osalt.com), among others.

NG YOUR
FTWARE:
RCIAL OR
E LICENSE

PROPRIETARY License
The source code is not distributed. This is the case with most commercial licenses.

For example: Windows

When developing software, it is advisable to **choose the license under which the software will be distributed as early on as possible** (see the next chapter: "Integrating open-source code"). This makes it possible, from the start, to identify what open-source modules can be iantegrated for easy development.

It will also save you from suddenly discovering, after years of development, that your software cannot be sold, or even distributed for free because it uses **incompatible licenses!**

!

WARNING!
Certain open-source licenses are incompatible with commercial licenses.

Likewise, certain open-source licenses are mutually incompatible!

Do you want to keep the source code secret?

In this case you need a **proprietary** license (the source code is not distributed) as opposed to distribution under an "open source" license (the source code is accessible).

Many **advantages** can be achieved by not granting access to your source code:

- it makes it more difficult for competitors to copy your technology;
- you are essential when it comes to developing new functionalities;
- the software's weaknesses are less visible;
- the software can be sold.

Sometimes you have no other choice: if the software is developed for an industrial partner who wants to keep the source code, or if the technology is being patented, a proprietary license is required.

In short, proprietary distribution allows you to **retain control** and makes it easier to **sell the software.**

OPEN-SOURCE License
The source code is made available, generally via a website.

For example: Linux

Do you want to share the source code?

Then you should choose distribution with an **open source** or **free** license (see the recommended licenses in the following chapter).

There are many **advantages** to open-source distribution:

- the software is generally free, which makes it easier to gainusers (open source ensures easier dissemination of the project);
- you can integrate many existing open source modules, which speeds up the development of the project;
- you can easily attract other developers who would like to contribute to the development of new functions;
- all users can easily correct bugs and weaknesses in the software.

Sometimes you have no other choice: for example, when contributing to an existing open-source project, or complying with the requirements of a financial backer or industrial partner.

In short, open-source distribution helps speed up the development of the software and increases its **dissemination**.

GRATING

RCE CODE

!

INCOMPATIBLE LicenseS

Not all open-source licenses are mutually compatible.

- Only important when you want to distribute the software.
- If you only want the software for internal use, compatibility is rarely an issue.

There are many types of open-source licenses, some of which are mutually incompatible. The Free Software Foundation keeps an up-to-date list of licenses that are compatible with the GPL license family (GPL, LGPL, AGPL, etc.) which are the most commonly used open-source licenses.

http://www.gnu.org/licenses/licenselist.html#GPLCompatibleLicenses

When developing open-source software

Many open-source licenses are mutually incompatible. For this reason, it is important to decide on a specific distribution license at the start of the project and only to integrate libraries that have compatible licenses. Among the plethora of available open-source licenses, these are the four licenses we recommend for distributing your software:

GPLv3 (or v2)

GNU Public License

This license does not support mixing with proprietary software. It requires the source code to be made available in the event of distributing the software. This is the default recommended license for open-source projects.

GPL version 3 license

Recommended by default

There are two versions: version 2 (v2) is the most common (more than half of all open-source projects) but it is incomplete and gives rise to legal issues under European law. On the other hand, v3 is much more complete and compatible with international law. For this reason, we recommend using GPLv3.

Unfortunately though, GPLv3 is incompatible with GPLv2.

If you absolutely need to integrate a library with the GPLv2 license, then you have no other choice: you must also distribute your software under GPLv2.

AGPLv3

Affero GNU Public License

Based on GPLv3, this license also requires the source code to be distributed for online services. Thus, it protects the open-source code from being appropriated by people who offer online services but do not distribute their code (for example: Google Docs). It is recommended for projects with web potential, if you want to prevent it from being used by proprietary web service providers.

AGPL version 3 license

Recommended to protect your code from being appropriated by proprietary online service providers.

It is also incompatible with GPLv2…

EUPL

European Union Public License

This license is the European equivalent of the AGPL. It has the advantage of being translated into 22 languages. It supports the integration of GPLv2 libraries, but then it becomes GPLv2, with the resulting consequences.

This license is recommended if you want to participate in public procurement contracts aimed at open-source software. Compatible with: GPLv2, OSLv2.1/v3.0, CPLv1.0, CeCILLv2.0.

EUPL license

Recommended for certain procurement contracts which require the license to be written in the language of the country.

LGPL version 3 license

Recommended for libraries that have to compete with a proprietary standard.

LGPLv3

Lesser GNU Public License

Supports integration in proprietary developments. This license is especially useful when you are developing a library and it is of strategic interest to be able to integrate this library in proprietary software, for example in order to compete with an existing proprietary standard.

When developing proprietary software

Some open-source licenses allow the code they protect to be integrated in proprietary software, as long as certain conditions are observed (in particular: mentioning the copyright).

If you develop proprietary software, you can integrate libraries distributed under the following licenses: **LGPL, Apache, MIT and BSD**. Other licenses also support integration in proprietary software: for further advice, please contact your Knowledge Transfer Office (KTO).

You can normally also integrate proprietary code developed internally or by another laboratory in the institution, with the agreement of the authors.

HOW TO PROTECT YOUR RIGHTS?

Best practice: ask every person working on the source code to sign an agreement foreseeing the transfer of rights.

WARNING: this agreement must contain certain compulsory information.

Contact your Knowledge Transfer Office for a document template.

Protecting ownership

Whether in the context of scientific or industrial collaboration, it is important to maintain **unified ownership** of the entire source code. Otherwise, a co-owner could potentially block future developments, whether in research projects or for commercial purposes.

This is especially true in the case of proprietary software (proprietary: only the executable file is distributed, without the source code), but also sometimes for open-source projects.

A co-owner can block the development of software, whether in the context of a research project or for commercial purposes.

Software ownership generally depends on the intellectual property regulations that apply to each university. Most intellectual property regulations include the presumption provided for under the Act of 1994 relating to the legal protection of computer programs, and thus provide for the ownership of software developed by researchers to be granted to the university.

This concern also affects collaborations with PhD students, postdocs, fellows, interns and student projects: if a person will be developing software, it is advisable that he or she is asked to sign an agreement foreseeing the transfer of rights before starting to work, in order to avoid any subsequent disputes.

Likewise, whenever any issues begin to emerge, this ownership can be backed up by legal instruments, which is essential before the software can be marketed as well as recommended before undertaking any collaboration. Software is protected by copyright by default, but additional protection systems may also be used.

Copyright

Copyright protection applies from the outset as soon as the work is created, without requiring any further formalities. Therefore, in the event of a dispute, it would be necessary to provide proof of authorship and priority. In practical terms, it would be necessary to prove that one actually wrote the source code at a given moment in time. The contents of laboratory logs, notes and any other written traces that can provide proof that the software was actually developed by such a person at a given time are always useful for providing this proof. However, the only way to establish the date before a court is to use a recognised depositing service: iDepot, IDDN, APP (in France), etc.

Copyright protects the form: the source code and the executable software.

However, it does not protect the functionalities of the software.

This type of process is inexpensive (iDepot = €45 for five years), but copyright only offers limited protection. Thus, it is easy to prove for identical copies (illegal copies of programs for example, or code snippets extracted verbatim), but it can be complicated to show infringement when source code is "inspired" by other source code…

Nevertheless, depositing source code is a good practice before undertaking any collaboration, since it effectively dispels any confusion about what each party owned before starting the collaboration, while also maintaining secrecy.

Confidentiality agreements

When collaborating on proprietary code, it is advisable to include a confidentiality clause in the research contract. This clause will provide a solid legal basis, and may indicate financial penalties, in particular as deterrents.

Unlike copyright, patents protect functionality, regardless of how it is written.

Patents

Patents can be filed (even in Europe) for software developments, as long as a technical effect can be proven (for example, on-board real-time video-processing software has a technical effect on the world around it, whereas an accounting software does not). Unlike copyright, patents do not protect the source code, only the functionality, regardless of how it is written. This protection has a maximum duration of 20 years.

Disadvantages include the complexity of the procedures (drawing up the patent plus several years of follow-up) and their cost (€20k to €100k according to the countries in which protection is sought).

Serious market opportunities must therefore be foreseeable before this process can be considered.

In addition, patents are public and describe the algorithm in detail: this gives your competitors everything they need to draw inspiration from your code! Therefore, it is important to make sure that patents are broad enough to prevent infringement.

Trademarks and domain names

Trademarks can be useful for differentiating quality products or services from the competition.

Trademarks entitle the holder to prevent a similar or identical name from being used for certain types of product or services. This makes them useful for protecting a reputation — a brand image — if, for example, your software begins to gain recognition, or if you want to give it broader dissemination (for example when developing an open-source community). Trademarks are relatively inexpensive (from €500 to several thousand euros for 10 years when using an agent, according to the type of trademark and the territory covered). They last for as long as you pay to maintain them and for as long as they remain in use.

Domain names for internet activity can be reserved for a very low price (under €20/year): this is a very easy step if you want to build up a reputation around a name.

Industrial designs

The graphic interface and the design, if they are of major importance for the appeal of the software, can be protected as “Industrial Designs”. The cost is similar to that of filing a trademark, but the duration is limited to 25 years at most.

Sui generis law on database contents

Sui generis law offers 15 years of protection for the producers of databases, in other words the legal or natural persons who take the initiative and assume the risk of making the investments required to compile the database.

This sui generis law provides certain protection of the contents of the database, granting the producer two prerogatives: the right to oppose the extraction (provisional or final transfer to another medium) of all or a substantial part of the contents of the database, and the right to oppose the reuse (making available to the public) of all or a substantial part of the contents of the database.

i

***Sui generis* law helps protect the considerable investments required in order to obtain a validated, high-quality database.**

Copyright on databases (as a container)

If it is original enough, the database as a container (in other words, the structure with which the elements are selected and arranged, also referred to as the architecture or the backbone of the database) is protected by copyright, from the time that said container is created, without any further formality being required. Therefore, as in the case of software, it is important in the event of a dispute to be able to offer proof of priority and thus to be able to establish a certain date for the creation of the database. This can be done by filing the creation with a recognised service (cf. above).

Invention Announcement and Invention Disclosure

They are not means for protection as much as communication tools. These forms allow you to interact with the KTO of your university in order to initiate the protection or commercialisation processes:

Invention Announcement

Short form (2 pages) containing the main information on the software and providing the basis for an initial meeting with an adviser.

Invention Disclosure

More complete form, to be filled out when protection or commercialisation are foreseeable.

BEST PRACTICES FOR DEVELOPING CODE

As soon as two people are working together on the software, it is essential to start thinking about collaboration tools. These tools take some time to set up and program, but the longer you wait, the more work will need to be done.

Here are some useful tips:

Coding rules

These writing rules help ensure uniformity of the code and make it easier to understand. They are used in most open-source projects. Although these rules vary for each project, a number of major guidelines can be found, and inspiration can be drawn from existing rules to help you get started (one example for PHP language: http://pear.php.net/manual/en/standards.php)

CODING RULES are aimed at standardising the writing of source code in order to make it easier to understand.

A part of these rules relating to page layout can also be automated in your favourite code editor.

Coding rules generally cover:

- rules for naming files, classes, variables and functions;
- code indentation (to highlight the structure);
- code organisation (for example: separating the graphic interface from the model);
- code documentation.

Documenting code

Although this aspect is generally included in the coding rules, we still felt that we should emphasise this point: not only does documenting code make it possible to collaborate with other developers, it also allows you to save enormous amounts of time when you need to look up certain parts of the code after a few months. Every function/method must describe at least its input and output variables.

Note : certain languages (PERL, Python) have their own documentation managers. Certain software applications can also be used to partially automate code documentation, such as:

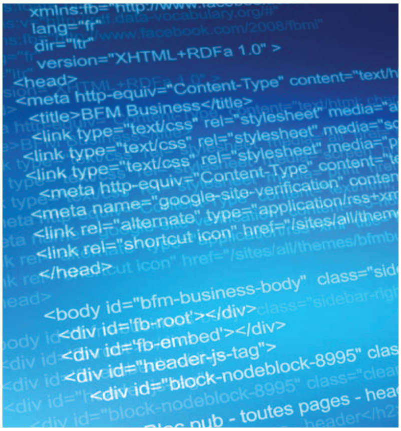
Doxygen (www.doxygen.org) or Javadoc (http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html).

Version management systems / forges

These systems are used to centralise the code on a server and manage all updates (versions) made to the various files. This makes sure that all developers have permanent access to the latest version of the code for their developments.

These systems often make it possible to manage different development branches: for example continuing to release bug corrections and other incremental updates (v1.0, v1.1, v1.2, etc.) while at the same time working on version 2.0, which includes completely new functionalities. Some examples of version management systems include: SVN and Git.

Forges (for example code.google.com or www.sourceforge.org) offer a version management system as well as additional tools: mailing lists, forums, documentation wikis, bug trackers, etc.



```
xmlns:fb="http://www.facebo
lang="fr"
dir="ltr"
version="XHTML+RDFa 1.0" >
<head>
<meta http-equiv="Content-Type" content="text/h
<title>BFM Business</title>
<link type="text/css" rel="stylesheet" media="a
<link type="text/css" rel="stylesheet" media="sc
<link type="text/css" rel="stylesheet" media="p
<meta http-equiv="Content-Type" content="te
<meta name="google-site-verification" conten
<link rel="alternate" type="application/rss+xm
<link rel="shortcut icon" href="/sites/all/them
</head>
<body id="bfm-business-body" class="sid
<div id='fb-root'></div>
<div id='fb-embed'></div>
<div id="header-js-tag">
<div id="block-nodeblock-8995" clas
pub - toutes pages - hea
```

Communication tools

The larger a project gets, the more important it becomes to structure communication among developers, as well as with users. Mailing lists, forums, wikis and bug trackers offer a solution to this need.

It may also be useful to use a task administrator which can include bug tracking, ticketing, requests for improvements and project management (breaking complex tasks down into simple tasks, distributing tasks among developers).

E.g. JIRA: http://atlassian.com/software/jira/overview

Design patterns

Design patterns are one way to develop software by using designs that have already proven to be effective.

This makes it easier to adapt the software when, in the future, you need to manage a new type of database, or when you want to redevelop the user interface with new, more-efficient tools.

For example, the best known design pattern is MVC (Model-View-Controller). This design pattern establishes a separation between the view (view = user interface), data processing (controller = data processing, verification of user inputs) and data interaction (model = database management, file management).

RECOMMENDED READINGS

Legal aspects of free / open-source software (in French)

http://www.crid.be/pdf/public/6566.pdf

Osalt : Open-source alternatives to well-known commercial software

http://www.osalt.com

List of GPL-compatible free software licenses

http://www.gnu.org/licenses/license-list.html#GPLCompatibleLicenses

Software Invention Announcement (contact your KTO)

Prince2 : a method for managing IT projects. This method has the advantage of being flexible and adapting to projects according to their size and management needs.

http://www.prince2.com

```
   . {font: 80%/150
    xt-align:center;backgrou
  yout {width:756px;background:#ff
#header {background:url('images/heade
#main {float:right;margin:0 60px 0 0;wi
#left {float:left;margin:0 0 0 50px;widt
 footer {background:url('images/footer.gi
 footer-right {text-align:right;float:righ
 footer-left {font-size:90%;margin:127px 0
  intro:first-letter {font-size:510%;float:le
  av {margin:0;padding:0}
  av li {list-style-type:none;list-style-image
   v li.last {border:none}
   v li a {font-family: Georgia, "Times new ror
   li a:hover {color:#000}
   Order input.edit, #frmOrder textarea {wi
    er textarea {height:150px;overflow:a
       nt-weight:bold}
       rea {border:1px solid #ddd}
        ound-color:#971f17;color
```

GLOSSARY

Source code

Code written by a programmer in a human-readable computer language (Fortran, C++, Pascal, etc.). This code is then generally compiled in order to generate a machine-readable executable file (for example, .exe files in Windows). These executable files are not human-readable, and therefore cannot be easily modified.

Open-source software / Free software

Software distributed with an "open source" or "free" license. Dozens of types of licenses can be classified as "open source" or "free" and they have the common feature of allowing access to the source code of the software if it is distributed.

A license is said to be "free" when the license guarantees the four freedoms defined by the Free Software Foundation (http://www.fsf.org) and it is said to be "open source" if the license meets the 10 criteria defined by the Open Source Initiative (http://www.opensource.org/). In practical terms, these criteria are very similar, and many licenses are both "free" and "open source".

Note: some software publishers allow access to a part of their source code (for example to allow the customer to verify security-related aspects). This does not make them open-source software, since the code is only available in read-only mode. They never allow users to modify the software by themselves, or to reuse code snippets.

Proprietary software

Software sold (or distributed free of charge) without access to the source code. The user then depends on the publisher for any bug corrections or new functionalities. Most commercial software (Windows, Photoshop, etc.) is proprietary, but this is also the case with freeware and shareware applications.

Freeware

Proprietary software distributed free of charge. Therefore, no access to the source code is provided.

Shareware

Proprietary software distributed free of charge in demo mode. This software can only be used for a limited period of time, or has limited functionality compared with the full commercial version.

Distribution (of software)

The term “distribution” can refer to selling software, installing it on a computer or offering it as a download from a website.

Software library

A library contains a series of functions that are used in a domain, such as mathematical functions, database management, creation of graphics, etc. Many libraries are available and can be integrated in software, which saves you from having to rewrite these functions.

UCL
Université catholique de Louvain

Louvain Technology Transfer Office

www.LTTO.com
LTTO@uclouvain.be
✆ 010 47 25 47