Le guide du chercheur

Créer des logiciels à l'UNamur

FSE

UNION EUROPEENNE

Wallonie

LE FONDS SOCIAL EUROPEEN ET LA WALLONIE INVESTISSENT DANS VOTRE AVENIR.

Version 1.4
juin 2012

Auteurs
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)

Introduction

Beaucoup de chercheurs universitaires développent leurs propres logiciels, généralement au départ pour répondre à leurs propres besoins, puis ces logiciels évoluent et commencent à intéresser d'autres personnes, chercheurs ou industriels.

Et c'est à ce moment que peuvent apparaître certaines difficultés :

- *problèmes pratiques pour développer du code à plusieurs institutions*, faute de documentation, d'outils de gestion des sources, et d'outils de communication (forum/wiki/mailing-list).

- *problèmes juridiques liés à la propriété du code* (le logiciel appartient à plusieurs donc on ne peut pas en faire ce qu'on veut) : ce genre de problème peut empêcher un laboratoire de négocier un contrat de recherche ou de commercialiser son logiciel

- *problèmes juridiques liés aux licences* : si on intègre dans son logiciel des bouts de code **open source** distribués sous des licences incompatibles (par exemple GPLv2 et Mozilla 1.1), on ne peut légalement pas distribuer le logiciel résultant sur un site web, même gratuitement, même avec le code source, et même juste pour une collaboration scientifique …

Or il y a moyen d'éviter une grande partie de ces problèmes en se posant quelques questions dès les premiers stades de développement du logiciel.

C'est dans cet esprit, et sur la base d'interviews avec quelques chercheurs, qu'est construit ce guide, qui a pour vocation de proposer quelques bonnes pratiques, afin de gagner beaucoup de temps si un jour le logiciel se développe.

Il s'agit de la première version de ce guide. Toutes les suggestions sont les bienvenues pour la prochaine version ! N'hésitez pas non plus à vous faire connaître si vous avez de l'expérience à partager dans le développement de projets informatiques, les licences open source, les outils de collaboration pour développeurs, la qualité logicielle…

Contact KTO (Knowledge Transfer Office)
Laurent Galas
ICT Knowledge Transfer Officer
laurent.galas@unamur.be
Tél. : 081 72 53 39

A qui est destiné ce guide ?

Ce guide est destiné à tous ceux qui à l'Université sont concernés par le développement de logiciels : étudiants, chercheurs, doctorants, mais aussi académiques et promoteurs qui encadrent parfois le développement de logiciels.

Il est évident que ce guide ne fait que survoler ces domaines qui pourraient faire l'objet d'un livre chacun. Les raccourcis sont donc inévitables, voire même voulus pour que le document reste lisible pour tous. Si vous souhaitez aller plus loin et pour toute question à ce sujet, vous trouverez des conseils avisés auprès de votre Service de Valorisation de la Recherche (KTO - Knowledge Transfer Office).

Nous n'avons pas pu éviter certains termes courants en informatique (open source, bibliothèque logicielle, code source…). Vous trouverez une définition de ces termes dans le LEXIQUE à la fin de ce guide.

Utiliser des logiciels

Il est très fréquent de commencer à développer un logiciel sur base d'un autre ou à l'aide d'autres ou d'utiliser un logiciel pour effectuer des tests, comparer des résultats, etc…

TOUT LOGICIEL DE SOURCE EXTERNE DOIT ETRE ACCOMPAGNE D'UNE LICENCE, qui définit les droits d'utilisation/modification d'un logiciel autorisés par le titulaire.

Si un logiciel de source externe n'est pas accompagné d'une licence, il faut partir du principe que ce logiciel ne peut être utilisé. Il est conseillé dans ce cas d'identifier le titulaire des droits (l'auteur, ou son employeur) et de lui demander une licence. Ceci peut être fait avec l'aide de votre KTO.

TOUT LOGICIEL DOIT ETRE ACCOMPAGNE D'UNE LICENCE.

Dans le cas contraire il faut partir du principe que le logiciel ne peut pas être utilisé.

De plus, les logiciels commerciaux disponibles en version student/académique/non commerciale (gratuite ou à prix réduit) comportent en général des limitations sur les usages commerciaux qui pourraient en être faits.

Si le logiciel est issu d'un partenaire dans un projet ou d'un tiers identifié, il est fortement conseillé de signer une licence pour définir les droits d'utilisation. Pour cela, veuillez contacter votre KTO.

Si la licence est une licence propriétaire et que vous avez la possibilité de choisir un autre logiciel concurrent, il est conseillé de favoriser un logiciel sous licence open source. Bon nombre de logiciels (édition, graphique, etc…) ont un équivalent sous licence open source. Pour cela, vous pouvez consulter, entre autre, le site Osalt (http://www.osalt.com).

Diffuser son logiciel : licence commerciale ou open source ?

Lors du développement d'un logiciel, il convient de **choisir le plus tôt possible la licence sous laquelle le logiciel sera distribué** (voir chapitre suivant : "intégrer du code open source"). Ainsi, on peut dès le départ identifier quels sont les modules open source que l'on pourra intégrer pour faciliter le développement.

Et on peut surtout éviter de se retrouver après des années de développement avec un logiciel invendable, voire même qu'on ne puisse pas distribuer gratuitement à cause de **licences incompatibles** !

ATTENTION !
Certaines licences open source sont incompatibles avec une licence commerciale.

Egalement ceraines licences open source sont incompatibles entre elles !

Veut-on garder le code source secret ?

On parle dans ce cas de distribution sous licence "**propriétaire**" (le code source n'est pas distribué) par opposition à la distribution sous licence "open source" (le code source est accessible).

Il y a de nombreux **avantages** à ne pas rendre le code source accessible :

- les concurrents ne peuvent pas facilement copier la technologie;
- on est incontournable pour le développement de nouvelles fonctionnalités;
- les faiblesses du logiciel sont moins facilement visibles;
- on peut vendre le logiciel.

Parfois également on n'a pas le choix : si le logiciel est développé pour un partenaire industriel qui souhaite le garder secret, ou si on est en train de déposer un brevet sur la technologie, une licence propriétaire s'impose.

En résumé, une distribution propriétaire permet de **garder la maîtrise** et permet plus facilement de **vendre le logiciel**.

LICENCE PROPRIETAIRE
On ne distribue pas le code source. C'est le cas de la plupart des licences commerciales.

Exemple : Windows

Veut-on partager le code source ?

On choisira alors une distribution sous licence "**open source**" ou "**libre**" (voir les licences conseillées au chapitre suivant).

Il y a de nombreux **avantages** à distribuer en open source :

- le logiciel est généralement gratuit, ce qui permet d'avoir plus facilement des utilisateurs (l'open source facilite le rayonnement du projet);
- on peut intégrer de nombreux modules open source existants, ce qui accélère le développement du logiciel;

LICENCE OPEN SOURCE
On met le code source à disposition, en général sur un site web.

Exemple: Linux

- on peut plus facilement attirer d'autres développeurs qui vont contribuer au développement de nouvelles fonctionnalités;
- les bugs et faiblesses du logiciel peuvent être facilement corrigés par tout le monde.

Parfois également on n'a pas le choix : si par exemple on contribue à un projet open source existant, ou si le bailleur de fonds / partenaire industriel l'impose.

En résumé, distribuer en open source permet d'**accélérer le développement** du logiciel et son **rayonnement**.

Intégrer du code open source

Il existe de nombreuses licences open source qui sont parfois incompatibles entre elles. La Free Software Foundation tient une liste à jour des licences compatibles avec la famille des licences GPL (GPL, LGPL, AGPL…), qui sont les licences open source les plus fréquemment utilisées.
http://www.gnu.org/licenses/license-list.html#GPLCompatibleLicenses

LICENCES INCOMPATIBLES
Toutes les licences open source ne sont pas compatibles entre elles.

→ uniquement important si on souhaite **distribuer** le logiciel.

→ Si on ne fait que l'utiliser en interne, il n'y a normalement pas de problème de compatibilité

Quand on développe un logiciel open source

De nombreuses licences open source sont incompatibles entre elles. Il est donc important de se fixer une licence de distribution en début de projet et de n'intégrer que des bibliothèques (logicielles) avec des licences qui sont compatibles. Parmi la pléthore de licences open source disponibles, voici les 4 licences que nous conseillons pour distribuer votre logiciel :

GPLv3 (ou v2)
GNU Public Licence

Licence **GPL version 3**
→ à conseiller par défaut

C'est une licence qui ne permet pas de mélange avec du logiciel propriétaire. Elle impose la mise à disposition du code source en cas de *distribution* du logiciel. C'est la **licence à conseiller par défaut pour les projets open source**.

Il existe deux versions : la version 2 (v2) est la plus courante (plus de 50% des projets open source !) mais elle est incomplète et pose des soucis légaux en droit européen. La v3 est beaucoup plus complète et compatible avec le droit international. **C'est donc la GPLv3 que nous conseillons.**

La GPLv3 est par contre malheureusement incompatible avec la GPLv2…

Si vous avez absolument besoin d'intégrer une bibliothèque sous licence GPLv2, dans ce cas vous n'avez pas le choix : vous devrez également distribuer votre logiciel sous GPLv2.

AGPLv3
Affero GNO Public Licence

Licence **AGPL version 3**
→ à conseiller pour interdire la récupération de son code dans des services en ligne propriétaires.

Basée sur la GPLv3, cette licence impose également la distribution du code source en cas de *services en ligne*. Elle permet donc d'éviter l'appropriation de code open source par des gens qui proposent des services en ligne mais qui ne diffusent pas leur code (ex : Google docs...). **A conseiller pour les projets avec un potentiel web, si on veut interdire l'usage par des prestataires de services web « propriétaires »**.

Elle est également incompatible avec la GPLv2…

EUPL
European Union Public Licence

Cette licence est l'équivalent européen de l'AGPL. Elle présente l'avantage de disposer de traductions en 22 langues. Elle permet d'intégrer des bibliothèque GPLv2, mais alors elle se transforme en GPLv2, avec les conséquences qui s'en suivent.

Cette licence est à conseiller si vous souhaitez participer à des marchés publics visant des logiciels open source.
Compatible avec : GPLv2, OSLv2.1/v3.0, CPLv1.0, CeCILLv2.0

Licence **EUPL**
→ à conseiller dans le cadre de certains marchés publics qui imposent que la licence soit rédigée dans la langue du pays.

LGPLv3
Lesser GNU Public Licence

Permet l'intégration dans des développements propriétaires. Cette licence est surtout utile quand on développe une bibliothèque et qu'il est stratégiquement intéressant de pouvoir intégrer cette bibliothèque dans du logiciel propriétaire, par exemple pour concurrencer un standard propriétaire existant.

Licence **LGPL version 3**
→ à conseiller pour des bibliothèques qui doivent concurrencer un standard propriétaire.

Quand on développe un logiciel propriétaire

Certaines licences open source permettent l'intégration du code concerné dans du logiciel propriétaire, moyennant le respect de quelques conditions (notamment : mentionner les copyright).

Si vous développez du logiciel propriétaire, vous pourrez intégrer des bibliothèques distribuées sous licences : **LGPL, Apache, MIT et BSD**. D'autres licences permettent également une intégration dans un logiciel propriétaire : pour tout conseil, contactez votre KTO.

Vous pouvez normalement aussi intégrer du code propriétaire développé en interne, ou provenant d'un autre laboratoire de l'institution, avec l'accord des auteurs.

Comment protéger ses droits ?

Garder la propriété

Que ce soit dans le cadre d'une collaboration scientifique ou industrielle, il est important de conserver une **propriété unifiée** de tout le code source. Dans le cas contraire, le copropriétaire peut potentiellement bloquer les futurs développements, que ce soit des projets de recherche ou une commercialisation.

Un copropriétaire peut bloquer le développement d'un logiciel, que l'on envisage un projet de recherche ou une commercialisation.

C'est particulièrement vrai pour les logiciels propriétaires (propriétaire : on distribue juste l'exécutable, sans le code source), mais parfois également pour les projets open source.

La propriété du logiciel dépendra généralement du règlement de propriété intellectuelle applicable dans chaque université. La plupart des règlements de propriété intellectuelle reprennent la présomption prévue par la loi de 1994 relative à la protection juridique des programmes d'ordinateur et prévoient donc que la propriété des logiciels développés par les chercheurs revient à l'université.

Bonne pratique : faire signer un accord de cession de droit à toute personne qui travaille sur le code source du logiciel.

ATTENTION : cet accord doit reprendre des mentions obligatoires.

➔ Contactez votre KTO pour un document type.

Cette préoccupation concerne également les collaborations avec les doctorants, post-docs, boursiers, stagiaires, ou des travaux étudiants : si la personne doit développer du logiciel, il convient alors de lui faire signer un accord de cession de droits avant le début de son travail, afin d'éviter toute contestation par la suite.

Egalement, quand il commence à y avoir des enjeux, cette propriété peut être renforcée par des moyens juridiques, ce qui est indispensable avant une commercialisation, mais qui est aussi fortement conseillé avant toute collaboration. Par défaut, le logiciel est protégé par le droit d'auteur, mais d'autres systèmes de protection peuvent être utilisés en complément.

Le droit d'auteur

La protection par le droit d'auteur existe d'emblée dès la création de l'œuvre sans qu'aucune formalité ne soit nécessaire. Cela étant, en cas de litige, il faudra apporter la preuve de la paternité et de l'antériorité. Pratiquement, il faudra prouver qu'on a bien créé le code source à un moment donné. Le contenu des cahiers de laboratoire, les notes et toute autre trace écrite qui permet d'attester que le logiciel a bien été développé par telle personne et à tel moment sont toujours les bienvenus pour permettre d'apporter cette preuve. Mais la seule façon d'obtenir date auprès d'un tribunal est de recourir à un service reconnu de dépôt : : iDepot, IDDN, APP (en France)…

Le droit d'auteur protège la forme : le code source et le logiciel exécutable.

Par contre il ne protège pas les fonctionnalités du logiciel.

Ce genre de démarche est bon marché (iDepot = 45€ pour 5 ans), mais le droit d'auteur n'offre qu'une protection limitée. En effet, il est facile de prouver une copie à l'identique (copies illégales de programmes par exemple, ou bouts de code repris tels quels), mais il peut être délicat de prouver une contrefaçon pour un code source « inspiré » d'un autre…

Néanmoins, procéder à un dépôt est une bonne pratique avant d'entamer une collaboration, car ça permet de lever l'équivoque sur ce qui était en possession de chacun avant le début de la collaboration, tout en gardant le secret.

Les accords de confidentialité

Dans le cadre d'une collaboration sur du code propriétaire, il est conseillé de prévoir dans le contrat de recherche une clause de confidentialité. Cette clause permettra de disposer d'une base juridique solide, éventuellement assortie d'astreintes financières, particulièrement dissuasives.

Les brevets

Il est possible de déposer un brevet (Europe y compris) sur des développements logiciels, pour autant que l'on puisse mettre en évidence un effet technique (par exemple, un logiciel embarqué de traitement de vidéo en temps réel a un effet technique sur le monde qui l'entoure, alors qu'un logiciel comptable n'en a pas). Contrairement au droit d'auteur, le brevet ne protège pas le code source, mais bien la fonctionnalité, quelle que soit la manière de l'écrire. Cette protection dure maximum 20 ans.

Contrairement au droit d'auteur, le brevet permet de protéger une fonctionnalité, quelle que soit la manière de l'écrire

Les inconvénients sont la lourdeur de la procédure (rédaction du brevet puis plusieurs années de suivi) et son coût (entre 20 et 100 k€ suivant la liste de pays dans lesquels on veut la protection). Il faut donc de sérieuses opportunités de marché pour se lancer dans cette démarche.

Egalement, un brevet est public et décrit avec détail l'algorithme: les concurrents ont donc tout ce qu'il faut pour s'en inspirer ! En conséquence, il faut veiller à ce que le brevet soit suffisamennt large pour ne pas être contourné.

Marques et noms de domaine

Les marques donnent le droit d'interdire l'utilisation d'une dénomination similaire ou identique pour certains types de produits ou services. Elles sont donc utiles pour protéger une réputation, une « image de marque », si par exemple un logiciel commence à être reconnu, ou si on souhaite lui donner le rayonnement le plus large (par exemple si on développe une communauté open source). Les marques sont relativement bon marché (environ entre 500€ et quelques milliers d'euros pour 10 ans si on passe par un mandataire, en fonction du type de marque et du territoire concerné). Elles durent aussi longtemps qu'on paye pour les maintenir, et qu'on les utilise.

La marque peut être utile pour distinguer de la concurrence des produits ou services de qualité.

Le nom de domaine peut être réservé afin d'être actif sur internet, pour un prix minime (moins de 20€/an) : si on souhaite bâtir une réputation sur un nom, c'est une des étapes les plus faciles.

Les dessins et modèles

L'interface graphique et le design, si ils ont une importance significative dans l'attrait du logiciel, peuvent être protégés par « Dessins et Modèles ». Les coûts sont semblables aux dépôts de marques, mais la durée est limitée à 25 ans au plus.

Le droit *sui generis* sur le contenu des bases de données

Le droit *sui generis* protège pendant 15 ans les producteurs de bases de données, c'est à dire les personnes physiques ou morales qui prennent l'initiative et assument le risque des investissements qui sont à l'origine de la base de données.

Le droit *sui generis* permet de protéger les investissements importants nécessaires pour obtenir une base de données validée et de qualité.

Ce droit *sui generis* permet, d'une certaine manière, de protéger le contenu de la base de données en offrant au producteur deux prérogatives : le droit de s'opposer à l'extraction (transfert temporaire ou définitif sur un autre support) de la totalité ou d'une partie substantielle du contenu de la base de données, et le droit de s'opposer à la réutilisation (la mise à disposition du public) de la totalité ou d'une partie substantielle du contenu de la base de données.

Le droit d'auteur sur les bases de données (contenant)

Si elle est suffisamment originale, la base de données en tant que contenant (il s'agit de la structure en fonction de laquelle les éléments sont choisis et agencés, en d'autres termes l'architecture ou encore le squelette de la base de données) est protégée par le droit d'auteur, et ce d'emblée dès la création dudit contenant, sans qu'aucune formalité ne doive être accomplie. Cela étant, comme pour les logiciels, il est important en cas de litige de pouvoir prouver l'antériorité et donc de pouvoir donner date certaine à la création de la base de données. Cela peut se faire via un dépôt auprès d'un service reconnu (cf. ci-dessus).

L'Annonce d'Invention et la Déclaration d'Invention

Il ne s'agit pas de moyens de protection, mais plutôt de communication : ces formulaires vous permettent d'interagir avec le KTO de votre université pour initier toute démarche de protection ou de commercialisation.

Annonce d'Invention
Formulaire court (2 pages) qui reprend les principales informations sur le logiciel et sert de base à un premier rendez-vous avec un conseiller

Déclaration d'invention
Formulaire plus complet qui sera rempli dans le cas où on envisage une protection ou une commercialisation.

Quelles bonnes pratiques pour développer du code ?

Dès que deux personnes travaillent sur le logiciel, il devient indispensable de penser à des outils de collaboration. Ces outils demandent un investissement temps pour la mise en place et l'apprentissage, mais plus on le fait tard, plus le boulot est important. Voici donc quelques conseils utiles…

Coding rules

Ces règles d'écriture permettent d'uniformiser le code et de s'y retrouver plus facilement. Elles sont présentes dans la plupart des projets open source. Si ces règles varient pour chaque projet, il existe cependant des grandes lignes et on peut s'inspirer de règles existantes pour commencer (un exemple pour le langage PHP : http://pear.php.net/manual/en/standards.php)

LES CODING RULES visent à standardiser l'écriture du code source pour s'y retrouver plus facilement.

Une partie de ces règles concernant la mise en page peuvent également être automatisées dans votre éditeur de code favori.

Les coding rules couvrent en général :

- les règles de nommage des fichiers, classes, variables et fonctions;
- l'indentation du code (pour mettre en évidence la structure);
- l'organisation du code (par exemple : séparer interface graphique du modèle);
- la documentation du code.

Documentation du code

Cet aspect fait en général partie des coding rules, mais on ne peut qu'insister sur ce point : documenter son code permet non seulement de collaborer avec d'autres développeurs, mais permet également de gagner énormément de temps lorsqu'on doit se replonger dans certaines parties de code après quelques mois. Chaque fonction/méthode doit décrire au minimum ses variables d'entrée et de sortie.

Note : certains langages (PERL, Python) disposent de leur propre gestionnaire de documentation. Il existe également certains logiciels qui permettent d'automatiser en partie la documentation du code, par exemple : Doxygen (www.doxygen.org) ou Javadoc (http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html).

Système de gestion de version / forges

Ces systèmes permettent de centraliser le code sur un serveur et de gérer les modifications (versions) apportées aux différents fichiers. Tous les développeurs ont donc accès en permanence à la dernière version du code pour leurs développements.

Ces systèmes permettent souvent de gérer des branches de développement différentes : par exemple continuer à sortir des corrections de bugs et autres modifications incrémentales (v1.0, v1.1, v1.2…) en même temps qu'on travaille sur la version 2.0 qui apporte des fonctionnalités complètement nouvelles. Quelques exemples de systèmes de gestion de version : SVN ou Git.

Les forges (par ex. code.google.com ou www.sourceforge.org) proposent un système de gestion de version, accompagné d'outils complémentaires : mailing-lists, forum, wiki pour la documentation, bug tracker…

Outils de communication

Plus un projet grossit, plus il devient important de structurer la communication entre développeurs, mais aussi avec les utilisateurs. Les mailing-lists, forum, wiki et bug trackers permettent de répondre à ce besoin.

Il peut être également intéressant d'utliser un gestionnaire de tâches qui pourra intégrer le bug tracking, le ticketing, les demandes d'améliorations, et la gestion de projet (décomposition des tâches complexes en tâches simples, répartition des tâches entre développeurs).
Exemple : JIRA - http://www.atlassian.com/software/jira/overview

Design patterns

Les design patterns sont une manière de concevoir un logiciel en utilisant des designs qui ont déjà prouvé leur efficacité.

De cette manière, on peut plus facilement adapter le logiciel quand dans le futur, on doit par exemple gérer un nouveau type de base de données, ou bien quand on souhaite redévelopper l'interface utilisateur à l'aide de nouveaux outils plus performants.

Par exemple, le plus connu est le "**MVC**" (Model-View-Controler). Ce design pattern propose de séparer la vue (View = interface utilisateur), le traitement des données (Controler = traitement des données, vérification des saisies utilisateur) et l'interaction sur les données (Model = gestion de la base de données, gestion des fichiers).

Lectures conseillées

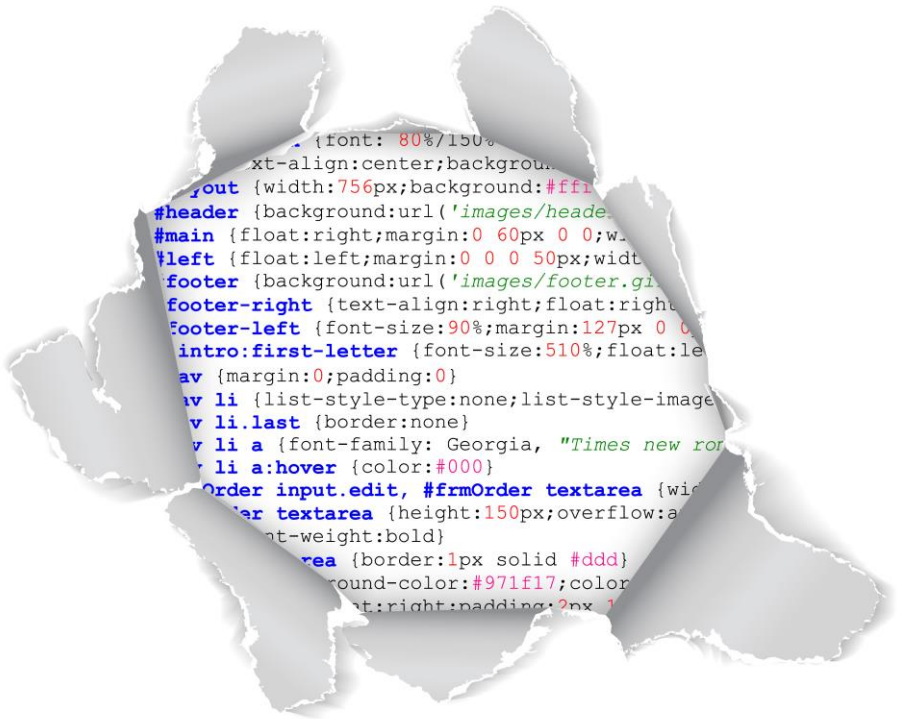
Aspects juridiques des logiciels libres / open source (http://www.crid.be/pdf/public/6566.pdf)

Osalt : alternatives open source aux programmes commerciaux connus (http://www.osalt.com)

Liste des licences compatibles avec la licence GPL
http://www.gnu.org/licenses/license-list.html#GPLCompatibleLicenses

Annonce d'Invention (http://www.interface.ulg.ac.be/docs/Software_disclosure.doc)

Prince2 : une méthode de gestion de projet informatique (http://www.prince2.com). Cette méthode présente l'avantage d'être flexible et de s'adapter aux projets en fonction de leur taille et de la finesse de gestion qu'on souhaite.



Lexique

Code source
Code écrit par un programmeur dans un langage de programmation (Fortran, C++, Pascal…) et lisible par un être humain. Ce code va ensuite généralement être compilé pour générer un fichier **exécutable**, lisible par la machine (par exemple les fichiers .exe sous Windows). Ce fichier exécutable n'est pas lisible par un être humain, et on ne peut donc pas facilement le modifier.

Logiciel open source / logiciel libre
Logiciels distribués sous une licence définie comme "open source" ou "libre". Il existe des dizaines de licences de ce type, avec comme point commun qu'on doit garantir l'accès au code source du logiciel si on le distribue.

On parlera de licence "**libre**" si la licence garantit les 4 libertés définies par la Free Software Foundation (http://www.fsf.org/) et on parlera de licence "**open source**" si la licence respecte les 10 critères définis par l'Open Source Initiative (http://www.opensource.org/). En pratique, ces critères sont proches, et beaucoup de licences sont en même temps "libres" et "open source".

Note : il arrive que des éditeurs logiciels donnent accès à une partie de leur code source (par exemple pour permettre au client de vérifier les aspects sécurité). Cela n'en fait pas des logiciels open source, car le code n'est disponible qu'en lecture. On ne peut en aucun cas modifier soi-même le logiciel, ou réutiliser des bouts de code.

Logiciel propriétaire
Logiciel vendu (ou distribué gratuitement) sans accès au code source. On repose donc sur l'éditeur pour toute correction de bug ou pour de nouvelles fonctionnalités. La plupart des logiciels commerciaux (Windows, Photoshop…) sont propriétaires, mais c'est le cas également des freeware et shareware.

Freeware
Logiciel propriétaire distribué gratuitement. On n'a donc pas accès au code source.

Shareware
Logiciel propriétaire distribué gratuitement en version démo. Ces logiciels ont soit une durée d'utilisation limitée dans le temps, soit ne disposent que d'une partie des fonctionnalités du logiciel commercial payant.

Distribuer (un logiciel)
Le terme "distribuer" couvre aussi bien la vente, que l'installation sur la machine de quelqu'un d'autre, ou que la mise à disposition en téléchargement sur un site web.

Bibliothèque logicielle ("Library")
Une bibliothèque regroupe une série de fonctions utiles dans un domaine, comme par exemple les fonctions mathématiques, la gestion de bases de données, la réalisation de graphiques… De nombreuses bibliothèques existent et peuvent être intégrées dans des logiciels, ce qui permet de ne pas devoir réécrire ces fonctionnalités.