CULTURE DU WEB

Politique open source

Pourquoi les DSI doivent définir une politique open source, et comment appréhender la question

Patrice **BERTRAND**
Directeur général

Smile
OPEN SOURCE SOLUTIONS

www.smile.fr • +33 (0)1 41 40 11 00 • contact@smile.fr
www.smile-oss.com • blog.smile.fr • twitter: @GroupeSmile

[1] PRÉAMBULE

[1.1] Smile en bref

Smile est une société d'ingénieurs experts dans la mise en œuvre de solutions open source et l'intégration de systèmes appuyés sur l'open source. Smile est membre de l'APRIL, l'association pour la promotion et la défense du logiciel libre, de Alliance Libre, PLOSS, et PLOSS RA, des associations clusters régionaux d'entreprises du logiciel libre. Ces associations sont réunies au sein du Conseil National du Logiciel Libre, dont Patrice Bertrand est le porte-parole.

Smile compte 420 collaborateurs en France, 500 dans le monde, ce qui en fait la première société en France spécialisée dans l'open source.

Depuis 2000, environ, Smile mène une action active de veille technologique qui lui permet de découvrir les produits les plus prometteurs de l'open source, de les qualifier et de les évaluer, de manière à proposer à ses clients les produits les plus aboutis, les plus robustes et les plus pérennes.

Cette démarche a donné lieu à toute une gamme de livres blancs couvrant différents domaines d'application. La gestion de contenus (2004), les portails (2005), la business intelligence (2006), les frameworks PHP (2007), la virtualisation (2007), et la gestion électronique de documents (2008), ainsi que les PGIs/ERPs (2008), e-commerce open source et réseaux sociaux d'entreprise (2010). Parmi les derniers ouvrages publiés, citons également « Les VPN open source », et « Firewall est Contrôle de flux open source », et « Middleware », dans le cadre de la collection « Système et Infrastructure ». Chacun de ces ouvrages présente une sélection des meilleures solutions open source dans le domaine considéré, leurs qualités respectives, ainsi que des retours d'expérience opérationnels.

Au fur et à mesure que des solutions open source solides gagnent de nouveaux domaines, Smile sera présent pour proposer à ses clients d'en bénéficier sans risque. Smile apparaît dans le paysage informatique français comme le prestataire intégrateur de choix pour accompagner les plus grandes entreprises dans l'adoption des meilleures solutions open source.

Ces dernières années, Smile a également étendu la gamme des services proposés. Depuis 2005, un département consulting accompagne nos clients, tant dans les phases d'avant-projet, en recherche de solutions, qu'en accompagnement de projet. Depuis 2000, Smile dispose d'un studio graphique, devenu en 2007 Agence Interactive, proposant outre la création graphique, une expertise e-marketing, éditoriale, et interfaces riches. Smile dispose aussi d'une agence spécialisée dans la

Tierce Maintenance Applicative, le support et l'exploitation des applications. Et Smile a également une activité d'hébergement.

Enfin, Smile est implanté à Paris, Lyon, Nantes, Bordeaux, Aix, Montpellier, Grenoble et Lille. Et présent également en Espagne, en Suisse, au Benelux, en Ukraine et au Maroc.

[1.2] Ce livre blanc

Ce livre blanc vise à expliquer *ce qu'est une politique open source, pourquoi* les entreprises doivent entreprendre de définir leur politique open source, ce qu'elles y mettront, quelle sera la démarche d'élaboration et de communication.

Ce livre blanc n'est pas en lui-même une politique open source, ni même un modèle de politique open source, mais nous pensons qu'il pourra aider les DSI à définir la leur.

Comme on le soulignera plus loin, la politique open source ne vise pas nécessairement à amener davantage d'open source dans le système d'information, même si dans certains cas, ce peut être l'un des objectifs, pour le plus grand bénéfice de l'entreprise.

Bien entendu, les consultants de Smile sont à votre disposition pour vous accompagner dans votre démarche et dans vos choix de produits.

[1] http://creativecommons.org/licenses/by-nd/2.0/fr/.

Table des matières

[2] POURQUOI UNE POLITIQUE OPEN SOURCE ?

[2.1] Il y a du logiciel open source dans votre entreprise

Toutes les études le confirment, l'open source est présent dans toutes les entreprises, grandes et petites. Et ceci, qu'elles l'aient voulu ou non.

Une étude du Gartner de 2009 estimait que 85% des entreprises utilisent déjà des logiciels open source, tandis que les 15% restant envisageaient de le faire. Encore ne s'agit-il que d'un sondage, et il est vraisemblable qu'un audit sur le terrain aurait trouvé un pourcentage très supérieur.

Une autre étude, du même Gartner, prévoit que dès 2011, 80% des logiciels commerciaux contiendront des composants open source. En d'autres termes, si l'open source ne rentre pas en tant que tel, les logiciels propriétaires vous en serviront eux-mêmes.

De nombreuses entreprises mesurent les bénéfices qu'elles peuvent tirer de ces logiciels, non plus seulement en termes de budgets, mais aussi de robustesse, d'ouverture, de dynamique de développement et d'indépendance dans les choix.

Et souvent, cet open source officiel n'est que la partie émergée de l'iceberg.

Parce qu'il suffit qu'un développeur trouve une librairie sur Internet qui lui fait gagner du temps, qu'un administrateur trouve un utilitaire open source utile et performant, qu'un chef de service déploie un produit open source que ses équipes auront sélectionné.

Et, pour l'entreprise, c'est une bonne chose à beaucoup d'égards. C'est ainsi que les produits open source se font connaître: par le bas de l'échelle. Certes, il n'y a personne pour faire du lobbying en jouant au golf avec le Président, mais sur le terrain, ceux qui développement, administrent, exploitent ou architecturent les systèmes d'information, trouvent des produits de qualité, robustes et sûrs, et libres d'utilisation.

[2.2] Vous devriez savoir où

Le constat est le suivant: l'open source est entré dans le système d'information, dans certains cas par la porte de service, hors d'un schéma directeur, sans politique formelle. Et une majorité d'entreprises n'ont pas même un recensement complet de leurs logiciels open source, des produits utilisés et des composants intégrés à leurs applications.

Dans beaucoup de cas, les DSI comptent sur le processus d'achat comme ultime rempart pour faire appliquer leurs choix d'entreprise. Un chef de service peut avoir des souhaits, peut parler même à tel ou tel fournisseur, mais aucun produit n'entrera dans le système d'information sans un bon de commande signé, et c'est au pire à ce stade que la conformité aux orientations de la DSI sera assurée. L'open source se passe de bon de commande et franchit donc ce type de barrière. C'est d'ailleurs une des qualités souvent appréciées: la rapidité d'acquisition et de mise en œuvre.

Comme nous le détaillerons plus loin, il y a de nombreuses raisons de vouloir maîtriser l'insertion de l'open source dans le système d'information, pour en tirer le meilleur bénéfice. Et la première étape d'une politique maîtrisée est souvent de faire un état des lieux de ce qui est déjà déployé, où, pour quel usage, avec quels retours.

[2.3] Vous pourriez gagner à en avoir davantage

Si l'open source entre dans les entreprises, ce n'est pas juste l'affaire de programmeurs incontrôlables. C'est véritablement que les bénéfices sont énormes. Les administrateurs et exploitants le savent: rien n'est plus fiable et performant qu'un serveur Linux pour faire tourner un serveur d'application java, par exemple, ou encore une base de données.

Depuis longtemps déjà, il serait ridicule d'affirmer *« nous sommes une grande entreprise, avec un immense système d'information, critique pour la marche des affaires, l'open source n'est pas fait pour nous »*. Car les plus grandes plateformes au monde sont construites sur des socles open source. Les géants de l'Internet, Google, Facebook, Amazon et les autres, s'appuient massivement sur des composants et grands produits open source. Ils ont des exigences de qualité de service, de performance et de productivité qui valent bien celles des plus grandes entreprises, et ils ont choisi l'open source. Ils ont les avocats parmi les plus compétents, les plus rigoureux et les plus paranoïaques du monde. Et ça ne les a pas empêché de définir une politique open source volontariste.

Dans une étude de Forrester réalisée en 2008, puis en 2010, une des tendances qui se dégage est que les raisons citées pour adopter davantage d'open source dans les entreprises ont évolué, le moindre coût étant moins souvent cité parmi les premières motivations, tandis que l'indépendance et la liberté de choix, la robustesse, l'ouverture, .. sont plus souvent mentionnés.

Nous n'allons pas faire ici le recensement complet des bénéfices que peut trouver l'entreprise à déployer des produits open source, ce n'est pas le but. Nous voulons surtout démontrer que, *puisque toutes les entreprises déploient des produits open source*, il convient qu'elles le fassent de manière encadrée.

Parmi les bénéfices les plus importants des solutions open source :

- Pérennité – nous y revenons plus loin.
- Liberté de choix: une moindre dépendance (lock-in) vis à vis d'un petit nombre de fournisseurs en situation de monopole ou d'oligopole. Les marges que procure une position de quasi-monopole sont telles que le marché des éditeurs de logiciels se concentre très rapidement, laissant les clients dans une situation de dépendances préoccupante. Les produits open source de qualité leur rende un peu de cette liberté perdue.
- Respect des standards: les logiciels open source sont en général plus respectueux des standards, à la fois parce que c'est la condition pour s'appuyer eux-mêmes sur d'autres briques open source, et parce qu'ils ne sont pas dans une logique de protection.
- Dynamique d'évolution: les logiciels open source, du moins certains d'entre eux, ont un développement qui s'appuie en tout ou partie sur une large communauté de développeurs, ce qui permet un rythme d'évolution supérieur.
- Standard de fait: certains des grands logiciels open source sont devenus des standards de fait, de sorte qu'ils concentrent à la fois les efforts de développement et l'expertise disponible.

[2.4] Deux univers open source différents

Lorsqu'on parle d'open source, il faut distinguer deux univers assez différents: l'open source des fondations et des communautés d'une part, l'open source des éditeurs d'autre part.

Leur point commun est que leurs programmes sont distribués sous des licences open source, qui donc donnent le droit d'accéder au code source, de l'étudier, de le modifier et de redistribuer librement le programme. Pour plus d'information sur ces licences on se reportera à l'annexe 9.1 « Copyright et licences » . Mais pour le reste, leurs modalités d'utilisation pour l'entreprise sont assez différentes.

Dans la catégorie des programmes de fondations et de communautés, on trouve quelques monstres sacrés, des piliers de l'informatique moderne. Linux soi-même, bien sûr, ainsi que tous les produits de la fondation Apache (le serveur du même nom, mais aussi Tomcat, ActiveMQ, Lucene, SolR, et tellement d'autres qu'on ne peut les citer ici. Ceux des fondations telles que Eclipse, Mozilla ou la FSF. On trouve aussi des produits d'origine communautaire, sans fondation, tels que les CMS Typo3, ou encore Spip.

Dans la seconde catégorie figurent les programmes issus d'éditeurs open source. Un éditeur open source est une entreprise commerciale presque ordinaire, à but lucratif, qui a investi dans la réalisation d'un produit, et qui le distribue en tout ou partie sous une licence open source. Ses motivations pour choisir l'open source peuvent être diverses, mais en général l'une d'entre elles est d'aider à faire connaître et adopter rapidement le produit. Pour plus d'information sur les différents modèles

économiques des éditeurs open source, on se reportera à notre livre blanc « Comprendre l'open source et les logiciels libre » dont un chapitre entier y est consacré.

La *politique open source* de l'entreprise devra certainement considérer ces deux catégories de programmes open source de manière distincte.

Typiquement, en termes de support, les choses sont très différentes. Dans la catégorie *fondations et communautaire*, il n'y a pas d'offre de support « officielle » de la part de la fondation. Il y a en revanche une communauté de développeurs, des mailing lists et des forums, qui peuvent être extrêmement réactifs pour certains produits phares. Pour ces produits, l'entreprise peut s'appuyer sur un prestataire assurant le support en niveau 1 et 2, et prenant en charge la relation avec les communautés de développeurs pour le niveau 3. Des distributeurs tels que Redhat ont une offre de support couvrant l'ensemble de leur distribution.

Pour les programmes d'éditeurs open source, la situation est tout à fait différente: le support est à la base de leurs business models, et ils s'attachent donc à avoir une offre de support de qualité sur tous leurs produits.

En termes économiques aussi, les deux catégories sont à distinguer. Les programmes de la première catégorie sont totalement gratuits d'utilisation: ils n'ont pas de version autre que la version open source, et la fondation ne demande rien en échange de la libre utilisation de ses programmes, même si elle apprécie certainement les dons. Bien sûr, le coût total de possession est rarement nul: il faut bien déployer et configurer ces programmes, puis les exploiter, et dans beaucoup de cas prévoir une prestation de support.

Dans le cas des programmes d'éditeurs open source, il convient en général de financer le développement du produit conduit par l'éditeur. Dans certains cas, la version du produit dont l'usage est recommandé en entreprise, et sur laquelle le support est assuré, est une version qui n'est pas sous licence open source. On entend parfois des critiques envers ce modèle, parfois appelé *freemium*, où le logiciel open source est un produit d'appel pour le *vrai* logiciel. Mais au final, l'entreprise fait son choix sur le bilan économique du logiciel, son rapport service rendu / coût total de possession. Et les logiciels open source, quel que soit leur modèle, sont le plus souvent gagnants sur ce critère.

[2.5] Une *politique open source*

Qu'est-ce qu'une politique open source ?

La politique open source de l'entreprise est un document qui définit ce que l'entreprise décide en matière de déploiement de logiciel open source, quels sont les critères de sélection, les exigences en termes de support, les licences acceptées, les

consignes adressées aux développeurs, les processus d'acquisition et de mise en œuvre, les modalités de contribution, etc.

On rencontre parfois le terme de *gouvernance open source*, pour désigner la même chose. Le terme nous semble moins approprié, la gouvernance faisant davantage référence à des processus de décision. Certes, la *définition* de la politique open source suit son propre processus, mais elle est fondamentalement de la responsabilité de la DSI, et le plus important n'est pas de savoir qui en décide, mais quelle voie elle montre.

D'autres parlent aussi de *schéma directeur open source*, mais ici aussi, c'est un peu différent. Un schéma directeur open source pourrait définir une situation cible de système d'information appuyé sur l'open source, et la voie à suivre pour atteindre cette cible. Il nous semble que la politique open source peut s'intégrer au schéma directeur, mais l'un et l'autre ne se confondent pas.

L'open source cohabite avec le propriétaire

Peu d'entreprises sont parvenues à être 100% open source et, il faut l'avouer, pour une majorité d'entre elles ce ne serait pas aisé. Les systèmes d'information sont donc amenés à voir cohabiter des logiciels open source et des logiciels propriétaires.

Ce n'est pas un problème, mais c'est un phénomène qui s'est accentué ces dernières années, et qui demande à être bien compris et mieux maîtrisé.

Sur le plan technique, bien entendu, cette cohabitation ne présente pas de difficulté particulière. En règle générale, les solutions open source sont plus respectueuses des standards, et donc offrent de meilleures possibilités d'interopérabilité. Mais c'est bien sûr au cas par cas que cette question devra être étudiée. L'accès au code source contribue parfois à faciliter cette intégration.

Dans beaucoup de cas, l'open source intervient particulièrement sur les couches basses du système d'information: OS, virtualisation, serveur d'application, middleware, annuaire, utilitaires d'infrastructures. Et au dessus de cette pile se positionneront différents produits, propriétaires ou bien également open source.

Sur le plan juridique, les incidences de cette cohabitation sont bien ciblées, nous les détaillerons plus loin. Lorsque deux produits, l'un open source, l'autre propriétaire, sont installés côte à côte et interfacés par des protocoles et échanges standards, il n'y a pas la moindre considération particulière au plan juridique. C'est lorsque open source et propriétaire sont intégrés en un même programme que des impacts sont à considérer.

Mais, comme on le verra plus loin, open source et propriétaire doivent cohabiter dans d'autres domaines encore: dans le schéma directeur, dans la boîte à outil de l'architecte, dans les réflexes du développeur, mais aussi dans les processus de sélection et d'achats.

Pourquoi une politique open source ?

La politique open source est un document qui énonce la politique de l'entreprise vis à vis de l'open source.

Elle permet à une DSI de piloter, maîtriser, organiser et accompagner, le déploiement de composants et produits open source.

Elle est une condition nécessaire pour (1) tirer tous les bénéfices du déploiement de solutions open source, et (2) maîtriser les risques, éviter les pièges.

Elle établit des directions à suivre, des recommandations, des interdits, elle définit des process et désigne des responsables, mais elle vise aussi à former et accompagner tous les acteurs impliqués dans l'entreprise.

Une politique open source n'est pas une politique *en faveur* de l'open source

Notez bien que la question n'est pas de savoir si la DSI apprécie les bénéfices spécifiques de l'open source, ou si au contraire elle s'en méfie. Définir une politique open source ce n'est pas nécessairement définir une politique *en faveur* de l'open source. A la rigueur, la politique open source de la maison pourra se limiter à déclarer *« pas de ça chez nous ! »*. Mais même dans ce cas, il faudra encore expliquer ce qu'on fait avec tous les produits qui sont déjà là, comment on s'en passe ou par quoi on les remplace, quels sont les budgets alloués à cette migration hors de l'open source …

Donc encore une fois que l'on veuille *plus d'open source*, ou que l'on veuille *moins d'open source*, il faut dans tous les cas définir sa *politique open source*.

D'autant que, comme nous le verrons ici, la question a de multiples facettes, et ne se réduit évidemment pas à *« plus »* versus « *moins* ».

Notons enfin que l'étude Gartner citée plus haut évaluait à 31% le nombre d'entreprises qui avaient déjà édicté une *politique open source*. Mais c'était aux Etats-Unis, début 2009. Il est probable que ce pourcentage a déjà augmenté, mais il est vraisemblable aussi qu'il serait bien moindre en France.

[3] LA PROPRIÉTÉ INTELLECTUELLE

La politique open source a bien sûr des aspects juridiques importants, mais elle ne doit pas pour autant être laissée entre les mains des seuls avocats. Dans beaucoup de cas, il s'agit de définir, pour le contexte spécifique de l'entreprise, quel est le meilleur équilibre entre bénéfices et risques, et si les avocats sont utiles pour souligner les risques, ils percevront certainement très mal les bénéfices.

Au chapitre juridique, le plus important est d'apporter aux acteurs un minimum de formation, éteindre les fausses idées, et attirer l'attention sur les vrais impacts.

[3.1] A qui appartient le programme ?

Lorsqu'un développeur écrit du code pour son employeur, l'entreprise détient la propriété intellectuelle du programme. Elle est libre d'en faire l'usage qu'elle choisit, en particulier de l'utiliser comme bon lui semble, de le distribuer sous la licence qui lui semblera la plus appropriée, et également de revendre son droit d'auteur en tant que bien immatériel. Évidemment, pour un produit qui connait un certain succès, cette propriété intellectuelle peut avoir une valeur chiffrée en dizaines ou centaines de millions d'euros. Ce n'est donc pas anodin.

Et bien sûr, l'usage jugé le plus approprié peut changer dans le temps: un programme initialement prévu pour un usage interne peut en fin de compte être distribué, puis un jour l'entreprise peut être mise en vente, et le programme sera valorisé comme un actif.

Il est donc important pour l'entreprise de savoir quel est l'état des lieux de sa propriété intellectuelle sur les programmes qu'elle possède, qu'ils aient été développés en interne ou bien achetés. Car la question se pose de la même manière pour un programme développé par un prestataire pour l'entreprise. De manière un peu plus complexe seulement puisqu'il s'agit alors de savoir (a) quelle était la propriété du prestataire lui-même sur le code qu'il a livré, et (b) quel a été le transfert de propriété dans le cadre du contrat de fourniture.

Il est important de bien comprendre que la propriété intellectuelle se définit au niveau microscopique, sur quelques lignes de code seulement. Si un développeur trouve 50 lignes de code qui lui simplifient la vie, et les colle dans son programme, alors il est vraisemblable que son employeur n'a plus la propriété complète du programme ainsi réalisé. Il n'y a pas de seuil officiel du nombre de lignes qui puissent être sujette à un droit d'auteur.

Les éditeurs de programmes ont bien compris l'importance de la propriété intellectuelle, non pas juste d'une manière globale, mais au plus fin, jusqu'au plus

petit composant. Mais les autres entreprises doivent s'en préoccuper également car, encore une fois, on ne sait pas toujours quel sera le devenir d'un programme, en termes de propriété, surtout s'il est de grande qualité.

Soulignons que les considérations citées ici ne sont pas propres aux composants open source. Encore une fois, la seule particularité est que ceux-ci demandent moins de permission pour s'insérer dans un programme, ou dans un système d'information.

[3.2] Les licences

Principes élémentaires

Les programmes open source ne sont pas des programmes « *sans licences* » comme on l'entend parfois. C'est au contraire leur licence qui les fait *open source*. Ils ne sont pas non plus *dans le domaine public*, c'est à dire n'appartenant à personne en particulier, ou du moins exempts de droits patrimoniaux.

Lorsqu'un développeur écrit un programme, il en détient les droits d'auteur, le *copyright*. Dans certains cas, ce peut être l'entreprise qui l'emploie qui en détient les droits. Et ce copyright peut être vendu, comme bien immatériel, d'une entreprise à une autre.

Le détenteur du copyright est libre de définir l'utilisation qui peut être faite de son programme :

- Il peut le garder pour lui, en interdire l'utilisation à qui que ce soit.
- Il peut vendre ses droits à un tiers, personne physique ou morale.
- Il peut utiliser son droit d'auteur pour préciser les conditions qu'il pose à l'utilisation de son programme. Il écrit ces conditions dans les termes de la *licence d'utilisation*.

A noter qu'en droit français, il n'est pas aisé *d'abandonner ses droits* et de mettre son programme dans le domaine public de manière irréversible.

Il faut bien expliquer aussi que ce n'est pas la *diffusion des sources* qui fait qu'un programme est *open source*, c'est le droit, inscrit dans la licence, de les utiliser, de les modifier et de les redistribuer librement.

Il est donc important de bien assimiler la logique suivante : à la base de l'open source il y a la *licence*, et la licence n'existe qu'à partir du *droit d'auteur*.

Ainsi tous les logiciels open source ont un *propriétaire*, ils ne sont pas *« à personne »*, ni même *« à tout le monde »*. Dans certains cas, ce propriétaire peut être une fondation à but non lucratif, ou bien ce peut être une entreprise commerciale

ordinaire. Il peut s'agir aussi de *plusieurs coauteurs,* en particulier à la suite de contributions successives.

Le détenteur des droits est libre de fixer les conditions de licence, il est libre d'en changer même, et il est libre d'y faire des aménagements ou exceptions, ou de diffuser à certains selon une licence, à d'autres selon une autre licence.

Celui qui *reçoit* le programme, en revanche, n'est pas libre. Il est lié par les termes de la licence. Certes il n'a pas signé de contrat, mais la licence lui a été bien énoncée, et elle stipule qu'il n'a le droit d'utiliser le programme que sous telles et telles conditions. S'il refuse ces conditions, il n'a pas le droit d'utiliser le programme.

Analyse et impacts des licences

Nous reproduisons en annexe le chapitre « Licences » de notre livre blanc intitulé « Introduction à l'open source et au logiciel libre », qui est plus complet sur le sujet.

[3.3] Politique open source et licences

La *politique open source* devra donc définir ce que l'entreprise souhaite et permet en termes de licences.

On évoque les licences le plus souvent concernant un produit complet, et ses conditions de déploiement et d'utilisation. Et dans le cas d'un produit complet, les DSI ont généralement été sensibilisées, parfois sévèrement, par les éditeurs propriétaires. Nous verrons que c'est au niveau composant et librairies que l'affaire mérite plus d'attention.

Programme utilisé en l'état

Lorsqu'il s'agit d'utiliser un produit complet, en l'état, « *off the shelf* », les produits open source sont absolument limpides: ils ne présentent aucune restriction d'utilisation, ni en nombre d'instances, de serveurs, d'utilisateurs, ni en finalité d'utilisation. Et ceci quelles que soient leurs licences, du moment qu'elles sont de nature open source. Cela inclut également un produit configurable, que ce soit de manière interactive, ou par le moyen d'un fichier de configuration.

A cet égard, les logiciels open source sont infiniment plus simples, juridiquement, que n'importe quel produit propriétaire, c'est important à souligner. La définition même d'une licence open source implique qu'il n'y a aucune restriction ou condition à l'utilisation du programme. Et les clauses subtiles des licences open source concernent uniquement les œuvres *dérivées*, que l'on évoquera plus loin.

Donc, concernant des produits utilisés en l'état, la *politique open source* peut se préoccuper de support, de pérennité, de conformité aux standards et de bien d'autres choses, mais n'a du moins pas à se préoccuper de risques juridiques.

Du moins, pour être précis, il faut ajouter: pour autant que le programme en question *puisse réellement être sous la licence open source indiquée*. Car on peut imaginer qu'un développeur assemble des composants de diverses origines pour constituer un programme, et déclare ce programme disponible sous une licence open source. Il est reçu comme tel par l'entreprise, mais en réalité, l'auteur n'était pas en droit de le diffuser sous cette licence. Soulignons ici qu'une telle situation n'est pas propre aux logiciels open source, elle peut se produire à l'inverse pour des logiciels propriétaires utilisant des composants GPL sans le dire. Ainsi en novembre 2009, Microsoft a du reconnaître qu'un programme distribué, le Windows 7 USB/DVD Download Tool, utilisait du code GPL et n'en respectait pas les exigences.

Composants logiciels, librairies et morceaux de code source

La question des licences est plus délicate et plus complexe concernant de petits composants, intégrés à des programmes. Copier quelques dizaines de lignes d'un programme A pour les coller dans un programme B suffit à faire du programme B une œuvre dérivée du programme A. Et selon les conditions de licence du programme A, cela peut entraîner des exigences particulières quant au programme B, particulièrement dans le cas où il serait distribué.

En général, les juristes des entreprises préfèrent les licences déjà bien connues et bien analysées, sur lesquelles on trouve une large littérature, et une certaine jurisprudence. Et a contrario, ils se méfient des licences trop particulières, qui demanderaient à être étudiées.

Les licences dites *non-copyleft*, telles que BSD, Apache APL, MIT, et quelques autres, ne présentent pas de risque juridique particulier, et doivent être acceptées sans attention particulière.

La licence GPL, en V2 ou en V3, est une licence dite *copyleft*, et à ce titre présente une exigence particulière : si une *œuvre dérivée* est *distribuée*, alors elle doit l'être sous la même licence. On se reportera à l'annexe 9.1 pour plus de précisions sur ce que signifient *œuvre dérivée* et *distribuer*. Une chose essentielle à retenir est que toute forme d'usage interne à l'entreprise n'est pas une *distribution* et n'est donc pas concernée par cette exigence. Elle est à considérer donc lorsqu'il y a une perspective de *distribution,* et particulièrement de distribution payante, du programme qui aurait été construit en intégrant un composant sous licence GPL.

Pour une entreprise qui opère un service en ligne – et elles sont de plus en plus nombreuses – la licence AGPL, ou Affero GPL, présente une exigence semblable. A la manière de la licence GPL, une œuvre dérivée, c'est à dire intégrant un composant AGPL, si elle est rendue disponible en tant que service en ligne, doit alors donner accès au code source de l'œuvre.

Enfin, entre *copyleft* et *non copyleft,* les licences de type *weak copyleft* telles que MPL, CDDL ou EPL, permettent une utilisation commerciale, mais portent néanmoins quelques exigences importantes.

Synthèse juridique

Ainsi, on pourrait dire en synthèse:

Licences non open source	En général très restrictives quant aux œuvres dérivées, parfois totalement interdites, parfois autorisées contre paiement d'un droit d'utilisation et avec limitations.
Licences non-copyleft connues (BSD, Apache APL, MIT, etc.)	Peu de contraintes au plan juridique.
Licences weak copyleft connues (Eclipse, Mozilla MPL, CDDL etc.)	Intégration possible à des œuvres dérivées non open source, mais des exigences particulières à considérer.
Licence GPL	Des contraintes en cas de distribution d'une œuvre dérivée. N'est pas adaptée pour les éditeurs de logiciels non open source.
Licence AGPL	Des contraintes en cas d'offre de service en ligne sur la base de l'œuvre dérivée. N'est pas adaptée pour les éditeurs de services en ligne qui estiment que leur code dérivé constitue un atout concurrentiel.
Licence LGPL	Elle permet qu'un programme appelle les fonctions d'une librairie LGPL sans qu'il soit requis que le programme soit distribué sous une licence particulière. En revanche, modifier la librairie elle-même implique de la redistribuer sous la même licence LGPL, si du moins on choisit de la redistribuer, ou de redistribuer le programme qui en fait usage.
Autres licences, moins connues	A étudier au cas par cas, en général moins appréciées.

Enfin, il faut une mention particulière pour les programmes dont une version existe sous une licence open source, mais qui sont le plus souvent utilisés sous une version non open source. Dans ce cas, on est ramené en pratique au cas des licences commerciales ordinaires, c'est à dire qu'il y a en général des restrictions d'utilisation, que ce soit en termes de serveurs, d'instances, de processeurs ou d'utilisateurs.

Ce que l'on peut figurer comme ceci:

	Typologie d'entreprise			
	Etablissement public	Entreprise commerciale	Editeur de logiciel	Editeur d'un service en ligne
Licences non open source	?	?	?	?
Licences non-copyleft connues (BSD, Apache APL, MIT)				
Licences weak-copyleft (MPL, CDDL, Eclipse)		?	?	?
Licence LGPL		?	?	?
Licence GPL		?	?	?
Licence AGPL		?	?	?
Autres licences open source , moins connues	?	?	?	?

Nous n'avons pas voulu mettre ici des marques de type « Oui / Non », car ce n'est pas approprié. Nous ne mettons que des « ? », pour indiquer des situations qui méritent plus ou moins d'attention.

Nous avons distingué les établissements publics, dans la mesure où la valorisation de leur propriété intellectuelle sur le logiciel n'est le plus souvent pas dans leurs perspectives.

Ce tableau, avec toutes ses imperfections, illustre aussi une chose: les licences propriétaires, de leur côté, présentent *toujours* d'importantes restrictions juridiques.

[3.4] Le développement

Comme on l'a vu au chapitre précédent, c'est dans le processus de développement que l'on est susceptible de créer une œuvre dérivée, avec les implications que l'on a évoquées.

La cible de la politique open source

La cible de la *politique open source* inclut tous les acteurs du développement, en particulier chefs de projets et développeurs. Il est important qu'ils soient informés, d'une part des règles générales régissant le copyright et les licences, telles que présentées plus haut, et d'autre part des directives particulières données par la société dans le cadre de leur développement.

Il faut bien mesurer que la plupart en sont très mal informés.

Ne pas réinventer la roue

De tout temps, on a demandé aux développeurs de ne pas réinventer la roue, de concentrer leurs efforts sur ce qui est spécifique, de réutiliser au maximum. Et aujourd'hui, le développeur applique ces consignes en s'alimentant sur le web, où il trouve rapidement une librairie ou un morceau de code qui répond à son problème.

C'est grâce à ce recours généralisé aux composants open source que le développement moderne est d'une productivité décuplée, et l'entreprise qui voudrait s'en passer aurait un prix élevé à payer.

Il ne s'agit pas nécessairement de faire la police, de contrôler ce que chaque développeur utilise. Il s'agit en premier lieu d'expliquer aux développeurs que ces petits composants, ces quelques lignes de code, ne sont pas sans conséquences. Et d'autre part de leur expliquer la politique open source, leur expliquer ce que l'entreprise leur demande de faire ou leur interdit de faire. Car une majorité de développeurs, voire de chefs de projets, est tout simplement ignorante sur la question.

[3.5] Contrats de prestation

Lorsque des fournisseurs livrent un logiciel réalisé dans le cadre d'un contrat de prestation, les entreprises savent qu'il convient de préciser qu'il y a transfert de priorité du fournisseur vers le client. Mais cela suppose que le fournisseur ait lui même la pleine propriété de la totalité de ce qu'il livre.

Il serait assez rétrograde et fort coûteux d'interdire l'intégration de composants open source à une application que l'on fait réaliser par un prestataire.

En fait, il convient de préciser dans le contrat :

- Soit que le fournisseur ne peut utiliser que des composants sous telles licences, ou bien telles catégories de licences, qui conviennent au client
- Soit que le fournisseur doit faire valider par le client les composants qu'il intègre au logiciel.

Et il faut qu'il soit bien entendu pour chacune des parties que « composant » couvre également des portions de code source recopiées.

Mais ces dispositions contractuelles ne suffisent pas. Il faut encore s'assurer que le prestataire prend les mesures nécessaires pour que ses équipes comprennent ces exigences, leur motivation et leurs implications.

Soulignons encore une fois que ces disposition ne sont en rien spécifiques à l'open source, elles s'appliquent à tout fournisseur.

[4] CHOIX ET SÉLECTION DE PRODUITS

[4.1] Une offre pléthorique, quelques pépites

Il existe 230 000 logiciels disponibles sur le site SourceForge.net, l'une des plateformes de référence des projets open source, et à peu près autant sur Google Code. Et beaucoup des plus grands projets ne sont pas sur des plateformes globales de ce genre.

A l'évidence, un grand nombre de ces projets sont encore en cours de développement, ou sont de piètre qualité. Beaucoup n'ont en fait qu'un seul développeur, et ne seront jamais vraiment achevés.

Alors comment faire son choix ?

[4.2] La recherche et l'identification des produits

La recherche d'une solution open source répondant à ses besoins suit une démarche profondément différente de celle habituelle pour les produits propriétaires:

- On ne peut guère compter sur les salons commerciaux, et encore moins sur des encarts publicitaires, pour repérer les meilleurs produits open source. Leurs éditeurs concentrent généralement leurs ressources sur le développement, et n'ont guère de budget marketing.
- On ne doit pas non plus compter sur les recommandations du Gartner ou d'autres, qui méconnaissent profondément l'open source.
- C'est souvent par une simple recherche Google qu'on pourra établir une première liste, au moins sur un critère de notoriété. La notoriété n'est pas tout, mais elle est un bon critère pour un premier filtre, car un produit médiocre fera généralement peu parler de lui.
- Une fois identifiés quelques produits, on pourra chercher encore des avis et retours d'expérience sur le web.
- On trouvera souvent différents sites dressant des recensements d'outils dans une catégorie donnée. Il est rare qu'ils soient très détaillés dans leur analyse, mais c'est une source complémentaire d'information.
- Bien entendu, les livres blancs de Smile peuvent être de précieuses aides au choix. Ils constituent des études plus complètes que les simples sites comparateurs.

- Enfin, une caractéristique importante des produits open source est qu'il est aisé de les installer et de les évaluer, que ce soit avec des ressources internes, ou en s'appuyant sur des prestataires spécialisés.

Au travers de cette recherche, on s'attachera bien sûr à évaluer chacun des critères listés plus haut, que l'on aura pondéré à sa guise. Dans beaucoup de cas, on mettra en compétition produits open source et produits propriétaires sans a priori, en laissant l'évaluation les départager au final.

[4.3] Des critères habituels, toujours d'actualité

L'un des chapitres les plus importants de la *politique open source* porte sur les critères de choix et de sélection de produits et composants.

Les critères fondamentaux sont bien sûr identiques à ceux qui concernent les logiciels propriétaires, typiquement:

- La réponse fonctionnelle au besoin identifié
- La capacité à évoluer ou accueillir des extensions pour répondre à un besoin plus large
- La capacité à accueillir une volumétrie plus importante
- Le respect des standards
- La robustesse et la qualité technique du produit
- L'ouverture et la capacité à s'interfacer à d'autres composants du système d'information
- La performance, la tenue en charge, les temps de réponse
- L'utilisabilité, l'ergonomie
- La disponibilité d'une documentation
- La qualité du support
- La pérennité
- La diversité des fournisseurs et prestataires, la concurrence et la liberté de choix
- Le coût initial et le coût récurrent.
- L'existence d'une feuille de route d'évolutions pour les années à venir
- Et bien sûr, les modalités de licence et leurs impacts

Ce ne sont là que des critères très habituels. Nous disons que sur bon nombre de ces critères, les logiciels open source auront des atouts spécifiques. Mais c'est bien sûr à démontrer au cas par cas.

Ce que *doit* faire la politique open source, c'est au minimum *affirmer l'éligibilité et l'égalité* des produits open source dans cette recherche, et de s'assurer également qu'ils ne sont pas pénalisés par des croyances erronées.

L'évaluation et la pondération de chacun de ces critères dépendra bien entendu du contexte particulier de l'entreprise et du projet.

Il y a toutefois quelques petites spécificités des logiciels open source.

[4.4] La pérennité

Les produits open source offrent souvent des perspectives supérieures en termes de pérennité. Pour autant, elles n'ont pas une garantie d'éternelle jouvence.

Il faut distinguer en fait différents cas de figure:

- Les produits phares de fondations et communautés. Lorsqu'il sont devenus des standards de fait, à l'image de Apache, Tomcat, ou encore OpenSSL, mais des dizaines d'autres encore, leur pérennité est assurée pour une bonne dizaine d'années au moins.
- Les produits communautaires moins en vue. Il n'est pas impossible que leur activité vienne à décroître progressivement, généralement au profit d'une autre solution, qui aura des atouts particuliers et aura réussi à gagner une meilleure dynamique de développement. Mais ce mouvement prend en général quelques années, et l'on a tout le temps pour organiser une migration.

Dans tous les cas, les produits de fondation ou bien communautaires ne répondent pas à une logique commerciale, ce qui les rend moins fragiles. Ils dépendent moins de leurs clients que de leur communauté de développeurs, qui sont moins volatiles.

- Les produits d'éditeurs obéissent à une logique différente, plus ordinaire. Un éditeur open source peut faire faillite, ou bien être l'objet d'une acquisition, comme il y en a eu bon nombre ces dernières années. Et l'acquéreur peut avoir une politique différente vis à vis du produit. La pérennité de l'éditeur open source s'évaluera sur des critères habituels: années d'expérience, base installée, fonds levés, etc.

Mais une chose fondamentale en matière de pérennité est que, pour un logiciel sous licence open source, vous avez dans tous les cas le droit d'utiliser (et même de modifier et de distribuer) le produit sans limitation, ni de périmètre ni de durée.

Un autre point essentiel est la possibilité d'un *fork* , c'est à dire de la poursuite du développement par une autre entreprise, ou par une communauté de développeurs, à

partir d'une certaine version open source. C'est en phénomène relativement rare, mais qui peut se produire en particulier si un acteur modifie sa politique commerciale unilatéralement.

[4.5] Communauté

Qu'est-ce qu'une communauté ?

La communauté est une notion très spécifique au logiciel open source.

Le communauté, c'est tout simplement l'ensemble des personnes qui participent à la vie du logiciel. Non pas uniquement les développeurs, mais tous ceux qui sont disposés à faire quelque chose pour aider à l'épanouissement du logiciel. Celui qui répond à une question posée sur un forum participe déjà à la communauté. Celui qui signale ce qui lui semble être une anomalie, contribue à la validation et donc à la meilleure robustesse du logiciel. D'autres traduisent des interfaces ou bien de la documentation. A sa manière, le simple utilisateur fait partie de la communauté aussi; en parlant à un ami, ou à un collègue, il va contribuer à faire connaître le produit, et à lui donner une plus large diffusion.

Ainsi, la « communauté » est une notion dont les contours ne sont pas définis avec précision. Et pourtant, tout le monde s'accorde à penser qu'elle joue un rôle de première importance pour un logiciel open source, étant à la fois l'indicateur et le facteur de sa vitalité, donc de sa pérennité.

Noyau et extensions

Dans le cas des produits d'éditeur open source, même si l'essentiel du produit, le noyau, est entre les mains des équipes de l'éditeur, la communauté conserve une grande importance. Les produits modernes ont souvent adopté une architecture de type noyau-extensions, parfois appelée *« hub and spoke »*, où le cœur du produit est complété par des extensions développées par d'autres, entreprises ou particuliers. Lorsque le déploiement du produit dans un contexte particulier requiert des fonctionnalités supplémentaires, on ne modifie pas directement le code source du produit, on ajoute une extension, qui apporte la fonctionnalité manquante. Cette extension peut facilement être rendue disponible pour d'autres projets. Certains projets open source mettent à disposition plusieurs centaines d'extensions de cette nature, qui non seulement complètent le produit, mais même lui donne une toute autre dimension.

Mesurer la dynamique communautaire

La force de la communauté est donc un des critères de sélection d'un produit open source.

Pour l'évaluer, on peut s'appuyer sur:

- Le Page Rank de Google, ou encore des outils tels que Google Trends, ou encore Alexa, évaluant la popularité du produit et de son site, et permettant de comparer les produits entre eux.
- Les forums, le nombre de *posts*, et plus encore les réponses aux questions posées, nombre et délai moyen.
- Le nombre de développeurs et de *commiters*.
- Le nombre de téléchargement du produit (mais certains produits gonflent cet indicateur de manière artificielle).
- Pour les produits d'éditeurs, le nombre de développeurs qui n'appartiennent pas à la société éditrice.
- Le nombre d'extensions disponibles et l'existence d'un référentiel répertoriant ces extensions.

[4.6] Le processus de sélection: qui cherche, qui choisit ?

Selon les cas, la politique open source peut soit:

1. Centraliser la sélection de solutions, entre les mains d'une petite équipe, qui sélectionne et valide les produits, dans une version donnée, et élaborent une sorte de catalogue open source de l'entreprise. Les architectes, chefs de projets et développeurs ne sont autorisés qu'à choisir dans le catalogue.
2. Centraliser la sélection des produits majeurs, structurants, tels que base de données, middleware, frameworks, outils de développement. Mais laisser une certaine liberté aux développeurs et chefs de projet pour trouver des outils de niche en complément des premiers.
3. Poser des règles générales de sélection de produits et laisser une large autonomie dans leur mise en œuvre.

Il nous semble que la voie (2) est un bon compromis, certains composants devant impérativement être unifiés au sein du système d'information, tandis que pour des petits outils, il peut être utile de conserver une certaine agilité.

Encore faut-il bien définir ces règles de choix, sur les critères cités plus haut, incluant en particulier les questions de licences ou de support, et assurer la formation des équipes sur ces sujets.

[4.7] Intégrer systématiquement les solutions open source à la sélection

Quelle est la place *souhaitée* des solutions open source dans les achats logiciels de l'entreprise ?

De nombreux états et organismes publics ont édicté une politique open source qui énonce que, dans tout processus d'acquisition, les solutions open source doivent être considérées. Cela ne signifie pas qu'elles doivent être retenues, mais que le processus d'achat doit les mettre dans la course, et qu'en conséquence bien sûr, si elles ne sont pas retenues, les raisons doivent en être bien identifiées.

Une telle politique de *considération* de l'open source est un pas important, et nous semble être le minimum. Se priver délibérément des outils qui constituent le socle de l'informatique moderne, des outils sur lesquels s'appuient toutes les grandes plateformes de l'Internet, serait une aberration.

Obliger à *considérer* les offres open source revient à affirmer que d'une part il n'y a pas de problème particulier à choisir ces solutions, et d'autre part que, si leurs qualités intrinsèques sont égales, elles peuvent avoir des bénéfices spécifiques.

Bien sûr, certains vont plus loin, et énoncent qu'il convient de *préférer* des solutions open source. C'est précisément l'un des rôles de la *politique open source*, que de se positionner entre le minimum, *mettre en lice l'offre open source,* et une ligne plus volontariste.

[5] LA DÉMARCHE

[5.1] Formation & Communication

En octobre 2009, le CIO « Chief Information Officer », c'est à dire le DSI, du Department of Defense (DoD) américain, adressait à l'ensemble de ses services un mémo qui est une ébauche de politique open source. Nous reproduisons en annexe [9.3] « Mémorandum du CIO Department of Defense », le texte de ce court mémo, qui est particulièrement instructif.

L'un des points soulignés par le CIO est le suivant:

« Il y a une mauvaise interprétation qui voudrait que le Gouvernement soit obligé de distribuer au public le code source de n'importe quel logiciel open source, et que, en conséquence, le logiciel open source ne devrait pas être intégré ou modifié pour un usage dans des systèmes classifiés ou bien sensibles du DoD. ».

Ça n'est qu'un point parmi d'autres. Ce qu'il illustre, c'est que le CIO a estimé nécessaire de combattre une croyance erronée, et de s'assurer que ses services ont une vision correcte du logiciel open source. *C'est absolument fondamental.*

Dans le même mémo, le CIO précise:

« Même si ces considérations (les bénéfices cités des logiciels open source) sont importantes, elles ne peuvent pas être les aspects prééminents de toute décision concernant le logiciel. En fin de compte, c'est le logiciel qui répond le mieux aux besoins et à la mission du Département qui doivent être utilisés, qu'ils soient ou non open source »

On est donc typiquement dans une politique open source, qui ne dit pas « il faut un maximum d'open source », mais qui dit *« vous devez bien mesurer les bénéfices de l'open source, vous ne devez pas laisser une méconnaissance du sujet influer sur vos décisions, mais in fine, vous devez prendre le meilleur produit. »*

Un tel mémo, signé du plus haut responsable informatique, est bien évidemment la première étape dans la communication sur la politique open source de l'entreprise.

Mais il ne suffit pas, bien entendu, et doit être complété par:

- Le texte intégral de la *politique open source*, accessible à tous et régulièrement mis à jour, avec les différents chapitres évoqués ici.
- Un site interne de référence, qui fournisse en particulier
    - L'information de référence en matière d'open source dans l'entreprise

  - Une FAQ, enrichie par des questions/réponses en continu
  - Un ensemble de bonnes pratiques
  - Des retours d'expérience de projets ayant déployé des outils et composants open source
  - Des liens vers différentes ressources externes
- Un référentiel des solutions et composants dont l'usage est recommandé dans l'entreprise
- Un cursus de formation à destination des chefs de projets et chefs de service.

La *politique open source de l'entreprise* devra comporter un volet communication, à destination de tous les acteurs impliqués dans ce déploiement, en particulier :

- Développeurs
- Chefs de projets
- Architectes
- Responsables de départements
- Maîtrises d'Ouvrage
- Achats.

Bien qu'on en parle beaucoup, l'open source est très souvent mal connu, de nombreux clichés subsistent et des croyances erronées sont à combattre.

[5.2] Une forge logicielle

L'open source, c'est aussi une manière différente d'aborder le développement, en particulier dans le contexte de grands projets où interviennent de très nombreux développeurs. Il y a assez peu d'entreprises qui ont, en interne, des projets de cette nature. Néanmoins, de nombreux aspects des approches modernes du développement ont pris naissance dans ces projets open source, intégration continue, par exemple, mais aussi méthodologies agiles. C'est pourquoi, mieux connaître l'open source, y compris dans ses méthodes, peut apporter beaucoup aux entreprises.

Dans certaines organisation, il existe des projets de cette nature, qu'ils soient open source ou non, qui font intervenir des développeurs de différentes équipes, dans différents pays, travaillant ensemble hors de la hiérarchie structurée habituelle. Il existe aussi des projets qui font intervenir des sociétés partenaires, au sein d'un consortium, par exemple des projets de recherche.

Pour ces projets, il est tout à fait pertinent de mettre en place une forge logicielle, réunissant un ensemble d'outils d'aide au développement collaboratif.

Et bien sûr, c'est d'autant plus pertinent pour un projet open source auquel participe l'entreprise. D'une manière générale, la bonne pratique n'est pas de développer *puis de poser le projet sous Sourceforge, GoogleCode* ou ailleurs, il est toujours préférable qu'un projet open source soit mis sous une forge le plus en amont possible.

[5.3] Un référentiel

Pour une entreprise qui mesure les bénéfices qu'elle peut tirer des produits open source, et qui souhaite en promouvoir l'utilisation, il ne suffit pas de dire à ses équipes: *vive l'open source, allez-y, foncez !* Il faut les accompagner et les aider en particulier à faire les bons choix, et des choix cohérents au sein de l'entreprise. Si les RH ont fait un site en Drupal tandis que la communication a utilisé eZ Publish, ce n'est pas une bonne chose. Si la filiale allemande déploie du PostgreSql tandis que les anglais préfèrent utiliser MySql, ce n'est pas bon non plus. Même chose si tel département développe en framework Zend et tel autre en Symfony.

En fait, ces quelques exemples relèvent d'un objectif d'harmonisation qui n'est pas propre à l'open source: pour les choix majeurs de plateformes et de frameworks, les entreprises cherchent déjà à définir des préconisations groupe contraignantes, qu'il s'agisse ou non de composants open source. Mais rappelons nous encore une fois que, comme il n'y a pas d'achat, il n'y a pas de contrôle, et la connaissance des recommandations est donc plus encore importante.

Bien sûr, la recommandation groupe ne porte pas uniquement sur des produits, mais bien sur des versions de produit, constituant des plateformes logicielles complètes, cohérentes et validées.

[5.4] Une cellule d'expertise et de conseil

Les composants open source sont disponibles par centaines de milliers. Et il n'est pas possible de les référencer tous, évidemment. Pourtant, il serait dommage d'établir un catalogue de quelques dizaines de produits, et de se priver du reste. On ne peut pas tirer pleinement parti des ressources de l'open source avec un simple catalogue, même mis à jour régulièrement. C'est pourquoi, dans une entreprise d'une certaine dimension, il faut qu'un développeur puisse s'adresser à une cellule qui le conseillera.

[5.5] Les outils d'audit de propriété intellectuelle

Il existe différents prestataires et outils permettant des audits de propriété intellectuelle, tels que en particulier Black Duck Software, Protecode, ou encore OpenLogic. Ce sont des outils qui scannent les référentiels de code, et analysent le code, source ou compilé, afin de reconnaître des composants présents, et bien sûr de pointer les licences associées à ces composants et d'analyser les compatibilités et impacts de ces licences.

Ces produits sont souvent utilisés dans des contextes de fusions et acquisitions, afin de valider la valorisation de propriété intellectuelle revendiquée par le vendeur.

Pour une entreprise qui, justement, a manqué de *politique open source*, ce peut être un bon moyen de commencer par faire un état des lieux.

Les vendeurs de ces outils d'audit ont parfois été critiqués pour un discours commercial qui veut faire peur, et qui donc fait du tort à l'open source. On peut imaginer que, sur le terrain, les commerciaux se laissent aller, mais sur le fond, il est exact qu'une entreprise doit se préoccuper de la propriété intellectuelle de ses programmes.

Du moins s'il s'agit de *valoriser* le programme, c'est à dire de le distribuer, de le vendre, ou de vendre l'entreprise qui le possède, alors l'analyse fine de la propriété intellectuelle est importante, car celle-ci peut restreindre les possibilités d'exploitation commerciale du programme, au travers des licences.

[6] LA CONTRIBUTION

La politique open source définira aussi, et c'est important, ce que souhaite l'entreprise ou l'organisation en termes de contributions aux produits open source.

Que doit faire un développeur qui a rencontré une anomalie et qui l'a corrigée ? Est-il encouragé à communiquer sa correction ?

Cela semble évidemment un minimum, mais encore faut-il que le développeur ait des instructions précises à ce sujet. L'entreprise lui accorde-t-elle même un petit quota de jours afin de mieux packager et documenter ses contributions ? En retour, elle peut s'attendre à bénéficier des contributions de qualité réalisées par d'autres entreprises.

Et s'il commite un peu de code, doit-il le faire de manière anonyme, sous son nom propre, ou bien au nom de l'entreprise ? S'il le fait au nom de l'entreprise, ne serait-ce qu'en indiquant son adresse mail professionnelle, il peut – dans des cas exceptionnels – fournir une information confidentielle à la concurrence, à savoir que l'entreprise utilise tel produit.

Mais à l'inverse, associer le nom de l'entreprise à des contributions de qualité, peut participer favorablement à sa réputation, démontrer son engagement pour un développement durable des produits.

Beaucoup de projets open source demanderont à un responsable de l'entreprise la signature d'un accord, un *contributor agreement,* spécifique qui cède les droits de l'employeur vis à vis de la contribution. En effet les fondations qui pilotent les grands projets open source ne veulent pas gérer une propriété intellectuelle dispersée, qui leur interdirait en particulier de changer des clauses de licence le cas échéant.

Nous avons évoqué la contribution de quelques lignes de code, mais l'entreprise peut également reverser des développements plus importants, par exemple des extensions réalisées autour d'un produit qu'elle utilise.

Ou, mieux encore, elle peut choisir de mettre à disposition un produit complet, qu'elle aurait développé en interne.

La contribution peut être motivée par un sens du devoir, c'est à dire une manière non monétaire de « payer » ce dont on a bénéficié gratuitement. Mais elle peut aussi résulter d'un calcul rationnel: si je contribue, je peux m'attendre à bénéficier des nouvelles contributions apportées par les autres.

Il est très important que cette logique soit expliquée par la *politique open source*, car elle ne sera pas spontanément comprise par un chef de service, qui percevra l'intérêt

court terme (ne rien faire, et attendre que les autres contribuent), mais aura du mal à mesurer l'intérêt de moyen terme: je contribue et bénéficie des contributions des autres.

Il faut mesurer toutefois que, pour des développements d'une certaine envergure, poser les sources sur Sourceforge ou équivalent ne sert pas à grand chose, si l'on n'a pas une démarche plus large de constitution et d'animation d'une communauté.

[7] LE SUPPORT

Il existe une diversité d'offres de support des produits et composants open source.

Pour une entreprise, ou un établissement public, le support est en général indispensable, sauf pour certains composants véritablement non critiques.

Ici aussi, on pourra se référer au chapitre « Support » de notre livre blanc « Comprendre l'open source et le logiciel libre », plus complet sur ce sujet.

[7.1] Produits d'éditeurs, produits communautaires

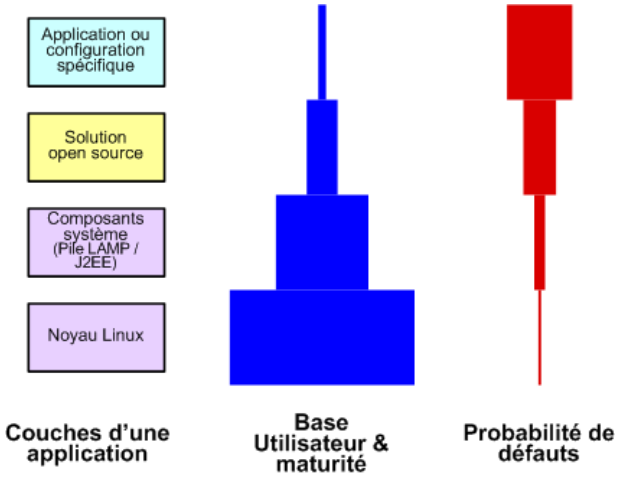
D'une manière synthétique, on peut distinguer les cas de figure suivants:

- Le produit provient d'un éditeur open source. Dans ce cas, il propose certainement une offre de support associée, qui constitue sa principale source de revenus. En fait, seul l'éditeur peut assurer, de manière crédible, un support de niveau 3, c'est à dire avec correction d'une anomalie dans le code. Néanmoins, un prestataire qui a assuré le déploiement du produit peut assurer le support aux niveaux 1 et 2, ce qui est en général une bonne chose car il connait les spécificités de l'environnement.
- Le produit est d'origine communautaire ou de fondation, par exemple le serveur Tomcat. Dans ce cas, il n'y a pas d'offre de support « officielle ». Le support peut être assuré: par un distributeur tel que Redhat, ou bien par un prestataire ayant une offre de support transverse et se chargeant de gérer la relation avec la communauté des développeurs.

Pour une entreprise intégrant un grand nombre de composants open source de différentes natures dans leurs développements internes, une offre de support transverse peut être une bonne solution. Mais les entreprises qui ont une forte expertise interne choisissent parfois de s'interfacer elles-mêmes avec les communautés. Il est vrai que les produits phares des fondations sont extrêmement robustes.

Pour ce qui est de la *politique open source*, il conviendra de définir justement *quelle est l'exigence de support requise pour l'entreprise*. Ceci en distinguant, bien sûr, différents cas de figure en termes de criticité dans le système d'information, et de typologie de produit.

[7.2] Le support sur les différentes couches



La figure précédente, extraite de notre livre blanc, fait apparaître les 4 couches d'une application typique: (1) le noyau Linux, (2) les serveurs d'application, interpréteurs et bases de données, (3) un produits open source de haut niveau, de type CMS, ERP, e-Commerce, par exemple, et (4) des développements spécifiques construits en complément du produit. La robustesse est en proportion du nombre de déploiement: les anomalies dans le noyau Linux sont extrêmement rares, et celles dans les couches telles que Apache ou PHP sont très rares également. De sorte que certaines entreprises considèrent que, sur ces couches inférieures, on peut vivre sans support. Il est impossible de dire si c'est un bon calcul ou non: tout dépend de la conséquence d'une anomalie.

[7.3] Politique open source et support

C'est pourquoi la politique open source doit définir les différents périmètres de criticité au sein du système d'information, en distinguant par exemple trois périmètres: vital, critique et utile. Chaque application, et donc chaque plateforme en production, doit être clairement identifiée comme relevant de l'un de ces périmètres.

Pour chaque périmètre, on indiquera l'exigence de support voulue par l'entreprise, qui pourra être déclinée également selon les typologies de produits et leur niveau de qualité connu. Ainsi, comme on l'a évoqué, dans un contexte qui ne serait pas réputé *vital*, on peut envisager de ne pas avoir un support contractualisé sur le noyau Linux ou le serveur Http Apache, mais il serait impensable, même dans un périmètre *critique*, de ne pas avoir de support sur un développement applicatif spécifique.

Ce que l'on peut figurer comme ceci:

	Périmètre		
	Utile	Critique	Vital
Configuration ou développement spécifique	☑	☑	☑
Progiciel applicatif		☑	☑
Stack (LAMP, JEE)			☑
Noyau			☑

Où la marque ☑ indique l'obligation d'un support contractualisé. *Soulignons qu'il ne s'agit ici que d'un exemple, et non d'une recommandation.*

Il faut garder à l'esprit toutefois que le support produit, quelle que soit sa qualité, et sa réactivité, *ne permet pas d'assurer la continuité de service* d'une plateforme. Aucun problème sérieux ne pourra être résolu en moins de 24 heures. Du côté des plateformes en production, la voie de la haute disponibilité est plutôt dans la politique de validation, de secours, de gestion des configurations et de retour arrière.

C'est principalement dans les processus de développement et d'intégration que le support sera important.

[8] LA DIMENSION COOPÉRATIVE DE L'OPEN SOURCE

Sony, Nokia, Samsung, Volkswagen, … de très grandes entreprises contribuent activement au développement du noyau Linux. Pourquoi Volkswagen payerait ses salariés à améliorer le noyau Linux ?

Pour ces entreprises, Linux fonctionne sur une logique de coopérative agricole, une logique de mutualisation des moyens donc de division des coûts. Une variante de ce qu'on appelle parfois « coopétition », c'est à dire que l'intérêt bien réfléchi de chacun conduit à identifier un terrain de coopération dans un contexte général de compétition.

Créer seul un système d'exploitation solide et complet serait hors de portée, même pour un grand constructeur automobile. Utiliser un système propriétaire serait trop coûteux et pas toujours adapté à ses besoins spécifiques. Reste la mutualisation: chacun "met au pot" quelques développeurs, et ensemble ils disposent d'un OS de qualité exceptionnelle, sur lequel ils ont une certaine maîtrise. Bien sûr, cela signifie qu'aucun d'entre eux ne dispose d'un OS meilleur que celui de son concurrent: cet aspect là de la concurrence a été pratiquement neutralisé. Mais il leur reste encore bien assez de domaines dans lesquels ils peuvent porter le combat, être meilleur, écraser les concurrents. Au global, ce domaine de coopération apporte bien sûr de moindres coûts à chacun, et donc une meilleure compétitivité.

Les quelques lignes qui précèdent sont extraites d'un article intitulé « l'open source dans sa logique mutualiste, un vecteur de compétitivité », que nous reproduisons en Annexe, et auquel nous renvoyons le lecteur qui voudra en savoir davantage.

Pour une entreprise qui aurait analysé les bénéfices d'une telle démarche coopérative, ce sera bien évidemment un chapitre essentiel du document définissant sa *politique open source*.

Sony, Nokia, Samsung, Volkswagen, IBM, Oracle, … On pourrait citer en exemple également le projet Genivi, qui vise à développer une plateforme open source pour ce qui est appelé le « In-Vehicle Infotainment », c'est à dire tout ce qui est information, amusement et loisirs à l'intérieur des véhicules. Participent à ce projet: BMW, Delphi, General Motors, Continental, Nokia, Renault, Valeo, … et bien d'autres.

Pas vraiment de doux rêveurs, épris seulement d'amour et d'humanisme. On peut compter que c'est pas un intérêt bien compris que ces entreprises s'engagent dans une démarche coopérative de cette nature. Et cela devrait donner des idées à bien d'autres entreprises, car les champs de possibilités sont immenses.

Ce peut être l'un des objets de la politique open source d'énoncer la volonté de l'entreprise de s'engager dans une telle démarche coopérative. Mais il est vrai qu'ici on est davantage dans des actions stratégiques que dans de la politique.

[9] ANNEXES

[9.1] Copyright et licences

[Reproduction d'un chapitre du livre blanc de Smile « Comprendre l'open source et le logiciel libre » - Pour plus d'information, ce référer au livre blanc complet]

Principes élémentaires

Les programmes open source ne sont pas des programmes « *sans licences* » comme on l'entend parfois. C'est au contraire leur licence qui les fait *open source.* Ils ne sont pas non plus *dans le domaine public*, c'est à dire n'appartenant à personne en particulier, ou du moins exempts de droits patrimoniaux.

Lorsqu'un développeur écrit un programme, il en détient les droits d'auteur, le *copyright*. Dans certains cas, ce peut être l'entreprise qui l'emploie qui en détient les droits. Et ce copyright peut être vendu, comme bien immatériel, d'une entreprise à une autre.

Le détenteur du copyright est libre de définir l'utilisation qui peut être faite de son programme :

- Il peut le garder pour lui, en interdire l'utilisation à qui que ce soit.
- Il peut vendre ses droits à un tiers, personne physique ou morale.
- Il peut utiliser son droit d'auteur pour préciser les conditions qu'il pose à l'utilisation de son programme. Il écrit ces conditions dans les termes de la *licence d'utilisation*.

A noter qu'en droit français, il n'est pas aisé *d'abandonner ses droits* et de mettre son programme dans le domaine public de manière irréversible.

Il faut bien expliquer aussi que ce n'est pas la *diffusion des sources* qui fait qu'un programme est *open source*, c'est le droit, inscrit dans la licence, de les utiliser, de les modifier et de les redistribuer librement.

Il est donc important de bien assimiler la logique suivante : à la base de l'open source il y a la *licence*, et la licence n'existe qu'à partir du *droit d'auteur*.

Ainsi tous les logiciels open source ont un *propriétaire*, ils ne sont pas *« à personne »*, ni même *« à tout le monde »*. Dans certains cas, ce propriétaire peut être une fondation à but non lucratif, ou bien ce peut être une entreprise commerciale

ordinaire. Il peut s'agir aussi de *plusieurs coauteurs,* en particulier à la suite de contributions successives.

Le détenteur des droits est libre de fixer les conditions de licence, il est libre d'en changer même, et il est libre d'y faire des aménagements ou exceptions, ou de diffuser à certains selon une licence, à d'autres selon une autre licence.

Celui qui *reçoit* le programme, en revanche, n'est pas libre. Il est lié par les termes de la licence. Certes il n'a pas signé de contrat, mais la licence lui a été bien énoncée, et elle stipule qu'il n'a le droit d'utiliser le programme que sous telles et telles conditions. S'il refuse ces conditions, il n'a pas le droit d'utiliser le programme.

Mentions élémentaires des licences

Toutes les licences open source ont en commun quelques clauses de bon sens :

- L'identification claire du propriétaire du copyright, y compris au travers des copies ou travaux dérivés.
- L'obligation de conserver la notice de licence en l'état, sur le programme et les travaux dérivés. C'est bien sûr une nécessité technique : inutile de définir des termes de licence s'ils sont évacués dès la première copie.
- La protection de l'auteur vis à vis des utilisateurs de son programme, ses éventuels défauts et les conséquences de ces défauts : *« ce programme est fourni 'en l'état' (« as is »)… »*. C'est bien le moins qui puisse être exigé : l'auteur vous laisse utiliser librement son travail, vous n'allez pas quand même lui réclamer des dommages et intérêts.

A noter que dans certains pays, la distribution payante d'un programme entraîne des droits inaliénables. D'une manière générale, la licence ne peut être contraire au droit national. C'est pourquoi elle dit *"Si vous ne pouvez pas distribuer le programme en satisfaisant à la fois vos obligations liées à licence et d'autres obligations applicables, alors vous ne pouvez pas distribuer le programme du tout ».*

C'est à dire que soit l'on peut respecter les lois nationales et la licence à la fois, soit on est dans l'interdiction de distribuer le programme sous ladite licence.

Définition d'un logiciel libre

Comme évoqué plus haut, le logiciel libre se définit par le respect de quatre libertés fondamentales :

- exécuter le programme,
- étudier le programme et l'adapter selon son besoin (ce qui implique bien sûr l'accès au code source),

- redistribuer le programme pour aider son prochain,
- et enfin améliorer le programme et distribuer ces améliorations au public (ce qui de même implique le libre l'accès aux sources).

Comme on l'a évoqué déjà, la finalité première est la liberté, l'accès au source n'est qu'un pré requis pour respecter cette liberté.

Définition d'une licence open source

L'OSI, *Open Source Initiative,* a édicté une définition précise de ce que signifie *open source*, une définition qui est aujourd'hui reconnue de manière à peu près universelle.

Avoir une définition officielle précise est très important, une licence ne doit pas pouvoir être *plus ou moins open source* : elle l'est ou ne l'est pas, les choses doivent être claires.

Et le site de l'OSI, opensource.org, indique aussi quelles sont les principales licences qui se conforment à cette définition. On y retrouve bien entendu les licences bien connues, à commencer par la GPL, que nous détaillons plus loin.

La définition comporte dix points, dont les trois premiers sont les principaux :

4. Libre redistribution : la licence ne doit pas interdire à qui que ce soit de vendre ou donner le programme.
5. Code source : la licence doit permettre la distribution sous forme de code source, et si le code source n'accompagne pas le programme il doit être disponible de manière facile et pratiquement gratuite.
6. Travaux dérivés : la licence doit permettre des modifications et des travaux dérivés, et doit permettre que ces travaux soient distribués sous les mêmes termes de licence.

Revenons sur ce point 3 : la licence doit au minimum *permettre* de redistribuer les travaux dérivés sous la même licence. Elle ne doit pas nécessairement l'obliger. On verra que cette nuance est à la base de la distinction entre la famille BSD et la famille GNU.

Parmi les autres articles de cette définition figurent différentes clauses de *non-discrimination* : la licence ne doit pas exclure tel groupe d'utilisateurs, ni tel domaine d'application, ni tel environnement technique. Par exemple, l'auteur du programme ne peut pas, en pacifiste militant, préciser que son programme ne doit pas être utilisé pour guider des missiles. Du moins s'il ajoute cette clause la licence ne sera plus open source.

Licences GNU et BSD

Il y a deux grandes familles de licences open source : la famille BSD et la famille GNU GPL. On parle parfois de licences *copyleft* pour les secondes et de licences *non copyleft* pour les premières. *« Copyleft »* est bien sûr un jeu de mot en référence au *« copyright »*, jeu de mot traduit parfois par *« gauche d'auteur », vs. « droit d'auteur ».* Mais pour autant le *copyleft* n'est pas un abandon de droit.

Pour qu'il n'y ait pas de confusion, précisons que si la FSF et le mouvement du logiciel libre *préfère* les licences *copyleft*, à commencer par la GPL, il n'y a pas correspondance entre *logiciel libre* et *copyleft* : les licences BSD sont aussi du logiciel libre.

La famille BSD

La licence BSD (Berkeley Software Distribution) autorise n'importe quelle utilisation du programme, de son code source et de travaux dérivés. Le code sous licence BSD peut en particulier être utilisé intégré à des logiciels sous licence non open source. On sait que Microsoft a repris du code TCP-IP sous licence BSD dans Windows, et que MacOSX est basé sur FreeBSD.

La seule contrainte spécifique est l'interdiction de chercher à tirer avantage de la dénomination de l'auteur, ici l'Université de Berkeley.

C'est donc la licence la plus libérale, qui entraîne le moins de contraintes : les programmes sous licence BSD sont quasiment dans le domaine public. C'est aussi peut-être la plus ancienne, puisqu'elle remonte à 1980. Il n'est pas interdit de modifier le texte de la licence, de sorte que l'on rencontre une multitude de versions dérivées, à quelques mots près. C'est un handicap pour la clarté et la lisibilité de la licence.

Dans la famille BSD, on trouve aussi la licence MIT, et la licence Apache. Cette dernière est d'une grande importance puisque utilisée déjà par la cinquantaine de projets de la fondation Apache. Les différences entre ces différentes licences sont de l'ordre du détail.

Licences Copyleft et GNU GPL

La licence GNU GPL

La licence GNU GPL est utilisée par 70% des programmes open source. Mais ce pourcentage en nombre n'est pas le plus important puisque certains logiciels phares de l'open source sont sous d'autres licences.

La licence GNU GPL, « GNU General Public Licence », se caractérise principalement par son article 2, qui énonce le droit de modifier le programme et de redistribuer ces modifications, qui constituent des *œuvres dérivées*, à la condition que ce soit *sous la même licence GPL*.

C'est ce que certains appellent le caractère *viral* de la licence : elle se communique aux travaux dérivés. Mais il est plus correct de parler de réciprocité, ou de donnant-donnant.

Bien sûr, toute la question est alors de savoir qu'est-ce exactement qu'une *œuvre dérivée* et *qu'entend-on par distribuer* ? Il existe une vaste littérature sur le sujet, et pourtant les zones d'ombre subsistent. Certains estiment même qu'il n'est pas mauvais de laisser quelques doutes.

Voyons déjà ce qui est clair.

Que signifie « Œuvre dérivée » ?

À coup sûr, si vous prenez un morceau de code source du programme A, que vous modifiez des lignes ou ajoutez des lignes pour obtenir un programme B, c'est une *œuvre dérivée*.

De manière certaine également, si vous appelez des fonctions du programme A depuis un programme B, en liant les deux programmes (*« link »*), alors ici aussi, le programme B est une *œuvre dérivée*. Cette liaison entre les programmes peut être statique ou bien dynamique, c'est à dire résolue à l'exécution seulement. Il existe un débat quant à savoir si une liaison dynamique donne une œuvre dérivée.

Dans les environnements techniques modernes, il existe en fait une diversité de moyens d'invoquer les services d'un programme autrement qu'en appelant une fonction. L'appel des services d'un programme A au moyen de protocoles d'échange réseau standards n'implique pas que le programme B soit une *œuvre dérivée*. Si c'était le cas, alors un navigateur adressant une requête à un site dont les programmes sont sous GPL, se trouverait lui-même obligé d'être GPL.

En fait, il est souvent admis qu'un programme B est considéré *œuvre dérivée* du programme A, si *B ne peut pas fonctionner de manière utile sans A*, ceci indépendamment des modalités techniques de la liaison.

Qu'en est-il d'un programme qui utilise par exemple une base de données MySql, sous licence GPL ? Si ce programme n'utilise pas, pour appeler la base, de librairies sous licence GPL, alors il n'invoque les services de MySql que au moyen de protocoles standards, ce qui n'implique pas qu'il soit GPL lui-même. Mais si le programme ne peut fonctionner autrement qu'avec une base MySql, alors on pourra considérer qu'il est œuvre dérivée malgré tout. A noter que la FAQ de MySql sur le sujet des licences a été critiquée pour laisser entendre que toute forme d'utilisation commerciale devait être sous licence commerciale, ce qui est erroné.

Que signifie « Distribuer » ?

Ici encore, certaines choses sont claires. À coup sûr, si vous commercialisez votre programme en tant que progiciel, cela s'appelle distribuer.

A l'inverse, utiliser et déployer un programme *au sein d'une même organisation*, n'est pas *distribuer*. Ce qui signifie qu'une entreprise peut construire une œuvre dérivée, et l'utiliser en interne sur autant de postes ou serveurs qu'elle juge utile, sans être tenue de diffuser les sources de l'œuvre. C'est un point essentiel dans la sphère économique.

Une autre question importante est celle de la relation client-fournisseur dans les métiers de l'informatique. Lorsqu'un prestataire tel que Smile construit une application utilisant des composants sous licence GPL, et livre cette application à son client, le prestataire doit livrer l'ensemble des sources, y compris ajoutés. Cette obligation de distribution des sources ne concerne QUE les personnes qui reçoivent le programme, ici donc le client. Il n'est pas requis de les mettre sur la place publique.

Par ailleurs, le client peut soit garder pour lui le programme (c'est-à-dire au sein de son organisation), soit le distribuer, mais alors obligatoirement sous licence GPL.

Notons aussi que utiliser ou proposer l'œuvre dérivée sous forme de service en ligne (*software as a service*), même commercial, n'est pas *distribuer*. C'est ce que fait Google par exemple. Sur ce point, voir plus loin la licence AGPL.

L'esprit de la GPL

Au delà des mots, *l'esprit* de la licence GPL est que, en tant qu'auteur ou propriétaire d'un programme, je vous donne le droit de l'utiliser et d'utiliser ses sources *à condition que vous en fassiez autant*. En somme, c'est donnant-donnant.

La licence GPL a pour effet de diviser le monde en deux « camps » : le GPL et le reste du monde. Si vous êtes du coté GPL, alors tout le patrimoine open source sous GPL vous est accessible sans restriction. Si vous êtes dans l'autre camp, c'est à dire que vous ne voulez pas distribuer votre code en donnant aux autres la même liberté qui vous était donnée, alors vous ne pouvez pas en profiter.

C'est ce qu'on pourrait appeler du *donnant-donnant*, que les critiques de cette licences appellent son aspect *viral*.

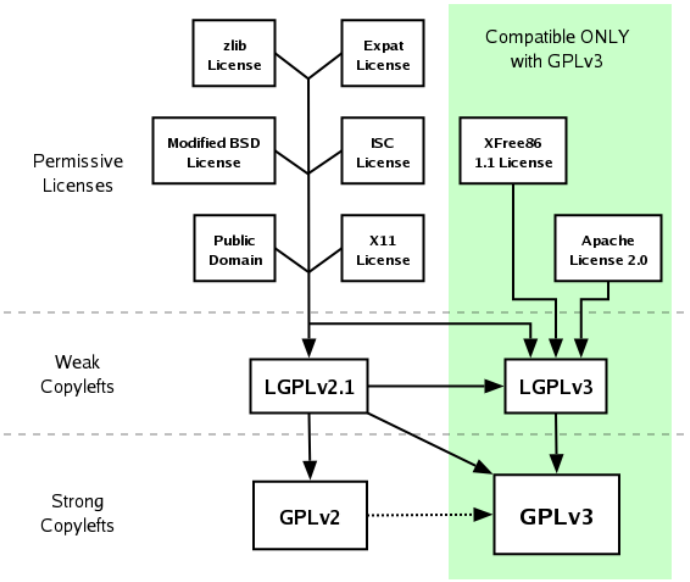
Compatibilité des licences

La question de compatibilité des licences est primordiale. Si un programme A est sous licence L_A et un programme B est sous licence L_B, alors est-il possible de construire un programme C utilisant à la fois A et B ? Le programme C héritera des exigences de L_A *et* de celles de L_B, et s'il y a des contradictions entre ces exigences, s'il est impossible de respecter les unes et les autres, alors il faudra renoncer à utiliser A et B.

Etant donné la domination de la licence GPL dans l'open source, la question principale est la compatibilité avec la licence GPL. Un programme open source, qui aurait une licence incompatible avec la GPL, aurait une utilisation plus réduite.

Parmi les licences compatibles on peut citer les licences BSD, MIT, ou la licence Apache (compatible GPL-v3). Parmi les non-compatibles, citons les licences SUN CDDL, Eclipse, Mozilla.

La figure suivante[2] est issue du site gnu.org. Les flèches indiquent la compatibilité des licences.



Notons qu'une licence L_A qui serait de type *copyleft*, et *pratiquement identique à la GPL*, mais porterait un autre nom, ne serait pas *compatible GPL*, puisque l'œuvre dérivée ne pourrait pas être *à la fois GPL et* L_A. C'est le cas par exemple de la licence « réciproque » introduite récemment par Microsoft, la MsRL.

LGPL

La licence LGPL est très proche de la GPL, mais autorise à appeler des fonctions du programme à partir d'un autre programme, sans exigence vis à vis de ces programmes, qui peuvent ne pas être sous licence open source eux-mêmes. Cette licence est donc particulièrement appropriée pour des librairies de fonctions destinées à être appelées par différents programmes, sans poser de conditions trop fortes sur ces programmes.

LGPL signifiait initialement *Library GPL*, mais l'appellation a été changée en *Lesser GPL* (Moindre GPL), car Richard Stallman souhaitait minimiser la correspondance « librairie = LGPL » et permettre d'envisager aussi bien des

[2] Image © 2007 Free Software Foundation Inc.

librairies sous GPL. La LGPL est un compromis entre la volonté forte de promouvoir l'open source, et éviter sa récupération au service de logiciels propriétaires, et d'autre part la volonté de rendre le plus grand service par la plus large utilisation.

A noter qu'au delà de la permission de *linker*, la LGPL introduit quelques conditions un peu subtiles sur le programme linké. Certains programmeurs ont préféré diffuser sous GPL avec en addendum une *special linking exception*, autorisant l'appel de fonction.

La GPL *with special linking exception* est plus lisible et plus permissive, ce qui l'a fait choisir par SUN pour le JDK.

GPL-v3

La version 3 de la licence GPL a été achevée courant 2007, et s'est déployée progressivement.

Elle vise à améliorer la v2, et l'adapter à un contexte qui a évolué, sur les points suivants :

- Une plus stricte définition juridique des termes, prêtant moins à interprétation ;
- L'interdiction d'empêcher, au moyen de dispositifs physique ou en ne fournissant pas l'information requise, la *mise en œuvre* du logiciel modifié sur son hardware cible. C'est ce qui avait été appelé *tivoisation*, du nom de l'entreprise Tivo, fabriquant de magnétoscope, qui y avait recours.
- Le cas de l'interdiction faite, dans de nombreux pays, de contourner les DRM. Une œuvre dérivée d'un logiciel GPL-v3 ne peut invoquer cette interdiction. C'est à dire qu'il n'est pas interdit d'écrire un programme de DRM en utilisant des composants GPL-v3, mais il est interdit d'interdire de le contourner.
- Une protection contre les revendications de brevets logiciels : celui qui diffuse son code sous licence GPL-v3 donne tous les droits d'utilisation permis par la licence et s'interdit de poursuivre les utilisateurs au nom des brevets logiciels.
- La possibilité d'ajouter certaines restrictions particulières à la licence, parmi un nombre limité de possibilités, ce qui donne un peu plus de flexibilité dans les problèmes de compatibilité de licences.

AGPL (Affero)

Comme on l'a vu plus haut, il n'est pas interdit de prendre un programme sous licence GPL, construire sur cette base un programme qui soit œuvre dérivée, et utiliser ce programme pour son propre besoin, y compris en le déployant au sein de son organisation, sans pour autant en diffuser les sources. De la même manière, il n'est pas interdit d'offrir un service accessible sur l'Internet, qui soit construit avec

cette œuvre dérivée, sans pour autant en diffuser les sources, car cet usage n'est pas une distribution.

Avec la montée en puissance des offres de services hébergés, de type Software as a Service (SaaS), ce type d'usage s'est étendu fortement. Or à bien y réfléchir, rendre le programme accessible directement à ses utilisateurs finaux au travers de l'Internet, est bel et bien une manière d'interdire l'accès aux sources, tout en faisant une exploitation le plus souvent commerciale, qui s'apparente à une distribution.

C'est pour répondre à ce risque de contournement que la société Affero a créé, en coordination avec la FSF, la licence AGPL ou Affero GPL. Elle est identique à la GPL, mais ajoute un article qui dit que si le programme initial permettait un accès par le réseau et diffusait ses sources par le réseau, alors le programme dérivé doit en faire de même.

L'article est le suivant:

« Si le programme tel que vous l'avez reçu est prévu pour interagir avec les utilisateurs au travers d'un réseau, et si, dans la version que vous avez reçue, un utilisateur interagissant avec le programme avait la possibilité de demander la transmission du code source intégral du programme, vous ne devez pas retirer cette possibilité pour la version modifiée u programme ou une œuvre dérivée du programme (...) »

C'est une mesure fondamentale, et il nous semble qu'elle tendra à se généraliser à l'avenir.

Les « weak copyleft »

La distinction entre copyleft et non-copyleft n'est en fait pas binaire. Entre la BSD, qui n'a presque aucune exigence quant aux œuvres dérivées, et la GPL qui exige la même licence, il peut y avoir des exigences intermédiaires. Ce sont les licences de type « weak copyleft », copyleft faible, en particulier la MPL, Mozilla Public Licence et la CDDL de SUN. A la manière de la GPL, la licence MPL oblige le code source copié ou modifié, s'il est redistribué, à l'être sous les termes de la même licence MPL. Mais cette exigence ne va pas au delà de la frontière du fichier de code source, de sorte que l'on peut associer des fichiers source sous MPL à d'autres fichiers pour former un programme sans conditions particulières sur cette œuvre dérivée. MPL et CDDL ne sont pas compatibles avec la GPL

De même la licence Eclipse, EPL, autorise des œuvres dérivées commerciales, sous licences non open source donc, mais a quelques exigences, entre autres d'identifier les portions d'origine sous EPL et d'indiquer la manière d'obtenir leur source.

[9.2] L'open source dans sa logique mutualiste, un vecteur de compétitivité

[Reproduction d'une tribune libre, signée par Patrice Bertrand, publiée dans Progilibre.com en 2009]

Lorsqu'on parle des business models de l'open source, on pense le plus souvent à ses acteurs économiques, éditeurs de logiciel, intégrateurs, distributeurs, prestataires de support, tous entreprises à but lucratif si l'on peut dire.

On imagine que pour le reste, l'open source est affaire de bénévoles, travaillant soirs et week-ends sur des projets communautaires, pour la gloire ou pour le bien de l'humanité.

Et on oublie bien souvent le troisième modèle de l'open source, celui qu'on pourrait appeler "coopératif", ou encore "mutualiste". Il a pourtant une importance extraordinaire, puisqu'on y trouve Linux soi-même, les projets de la fondation Apache ou de la fondation Eclipse. Du beau monde...

De quoi vivent ces projets ? Principalement de dons en nature, soit de code source directement, soit de temps de développeurs, payés par leur employeur, et travaillant sur les projets de la fondation. Pourquoi les entreprises font-elles ces contributions ? Pour l'essentiel, ce n'est pas affaire de marketing, il ne s'agit pas d'avoir son nom sur le spi d'un voilier géant. Elles sont au contraire relativement discrètes sur leur implication. Ce n'est certainement pas non plus par philanthropie ou dans une logique de mécénat. Leurs motivations sont de différentes natures. Participer à la gouvernance des projets, c'est à dire avoir son mot à dire dans les orientations qui sont prises, pour les diriger dans le sens de ses besoins. Construire une expertise interne sur ces projets, qui permettra de bien les intégrer à ses propres développements. Acquérir une légitimité sur le marché. Mais surtout, au final: disposer pour soi même d'un outil de qualité en partageant les coûts.

Il est très instructif de regarder la liste des plus grands contributeurs du noyau Linux.

Sony, Nokia, Samsung, Volkswagen, sont au nombre des contributeurs actifs. Pourquoi Volkswagen payerait ses salariés à améliorer le noyau Linux ?

Pour ces entreprises, Linux fonctionne sur une logique de coopérative agricole, une logique de mutualisation des moyens donc de division des coûts. Une variante de ce qu'on appelle parfois « coopétition », c'est à dire que l'intérêt bien réfléchi de chacun conduit à identifier un terrain de coopération dans un contexte général de compétition.

Créer seul un système d'exploitation solide et complet serait hors de portée, même pour un grand constructeur automobile. Utiliser un système propriétaire serait trop coûteux et pas toujours adapté à ses besoins spécifiques. Reste la mutualisation:

chacun "met au pot" quelques développeurs, et ensemble ils disposent d'un OS de qualité exceptionnelle, sur lequel ils ont une certaine maîtrise. Bien sûr, cela signifie qu'aucun d'entre eux ne dispose d'un OS meilleur que celui de son concurrent: cet aspect là de la concurrence a été pratiquement neutralisé. Mais il leur reste encore bien assez de domaines dans lesquels ils peuvent porter le combat, être meilleur, écraser les concurrents. Au global, ce domaine de coopération apporte bien sûr de moindres coûts à chacun, et donc une meilleure compétitivité.

On retrouve cette logique coopérative sur la plupart des projets de fondations. Le moteur de recherche Lucene en est un bel exemple. Depuis 2002, Yahoo a été un important sponsor des travaux de son créateur, Doug Cutting, qui a ensuite été salarié de Yahoo, de 2006 à 2008, tout en continuant ses travaux sur ce projet de la fondation Apache. En 2006, le groupe de média CNET donne à la fondation Apache le source de SolR, un outil de recherche initialement à usage interne, qui rejoint le projet Lucene. Et Yahoo continuera de financer les projets qui gravitent autour de Lucene, en particulier Hadoop, devenu l'outil de référence pour la répartition de tâches à très grande échelle. On est bien ici dans une logique coopérative: CNET bénéficie de produits très supérieur à ce qu'il aurait pu financer seul, et de même, pour chaque jour financé par Yahoo, le bénéfice est décuplé puisqu'une vingtaine de commiters entourent Doug Cutting, payés par différents employeurs.

C'est un exemple particulièrement intéressant, qui pourrait être pris en modèle dans bien d'autres domaines. Dans un nombre extraordinaire de secteurs, des fédérations interprofessionnelles pourraient définir des terrains de coopération logicielle, stimuler des projets mutualistes...

D'une certaine manière, le modèle économique de l'éditeur logiciel traditionnel peut aussi s'assimiler à un financement coopératif: différents clients de l'éditeur payent un droit d'usage, et/ou un support, et la somme de ces contributions individuelles finance le développement du produit. La différence entre ce modèle éditeur standard et le véritable modèle coopératif est dans la prise de risque et dans le partage des bénéfices. L'éditeur prend des risques en investissant pour construire son produit, bien avant d'avoir des clients. En contrepartie de cette prise de risques, il escompte un profit important si son produit est un succès. Et en particulier, il espère vendre de nombreuses licences aussi cher que le permettra le marché, sans laisser croître ses coûts de développement en proportion. Dans un développement coopératif, les clients partagent le risque, et sont assurés en retour de coûts au plus bas et surtout, de tirer eux-mêmes les bénéfices d'échelle.

Pour autant, l'approche coopérative du développement n'est pas indissociable de l'open source. Dix entreprises peuvent s'associer et co-financer le développement d'une application, qui servira à chacune d'elles. Elles peuvent créer une petite structure pour en gérer les développements et les évolutions. Le domaine des ERP "verticalisés", des applications qui s'appuient sur un socle ERP pour en adapter les fonctionnalités aux besoins d'un secteur d'activité particulier, serait typiquement le domaine de prédilection de cette approche coopérative de l'open source.

En principe, il n'est pas requis que l'application résultant de cet effort coopératif soit open source. Et même, certains pourraient craindre qu'elle le soit, car elle pourrait alors être utile à d'autres qui n'auraient pas contribué à son financement.

L'open source n'accepte pas de frontière, pas de restriction dans la diffusion: si elle est open source, l'application ne pourra pas être réservée à ses seuls sponsors initiaux. Il faut avoir les nerfs solides, pour voir un concurrent non coopératif reprendre l'application sur laquelle on a investi.

Mais dans la pratique, on sait bien qu'il n'est pas si facile de "prendre" une application. Ayant démonétisé le copyright, l'open source a mis la connaissance, l'expertise, la maîtrise, au premier plan. L'entreprise qui n'aura pas participé aux travaux aura, dans la pratique, beaucoup de mal à profiter de l'application sans avoir construit une certaine maîtrise, et sans avoir son mot à dire quant aux orientations. En ne contribuant pas, elle se rendrait dépendante vis à vis de ses propres concurrents, qui seuls assureraient la gouvernance, définirait la roadmap du produit.

C'est pourquoi, pour un groupement d'entreprises dans une logique coopérative, la crainte de voire certains profiter sans avoir contribué, peut passer au second plan. Ce qui prime, c'est la croyance que d'autres au contraire viendront rejoindre les rangs des sponsors, apporter leurs propres idées, des moyens renforcés, des coûts encore divisés, pour viser une application encore meilleure. Bien sûr, il faut une certaine foi, mais quelques belles réussites ont ouvert la voie.

[9.3] Mémorandum du CIO Department of Defense

[Note interne adressée par le DSI du Département de la Défense américain à ses services, en octobre 2009. Il ne constitue pas une politique open source à lui seul, mais peut en être le point de départ. La traduction, non-officielle, est de Smile]

Department of Defense
6000 Defense Pentagon
Washington, DC 20301-6000

Le 16 octobre 2009

Recommandations concernant le logiciel open source

Références: voir annexe I

Pour accomplir sa mission avec efficacité, le Département de la Défense doit développer et faire évoluer ses capacités en matière de logiciel plus rapidement que jamais, pour se préparer à de nouvelles menaces et faire face à des exigences en permanente évolution. L'utilisation de logiciel open source (OSS) peut avoir des avantages dans cette perspective. Ce mémo apporte des clarifications quant à l'usage de l'open source et remplace le précédent mémo du DoD en date du 28 mai 2003.

Le logiciel open source est du logiciel dont le code source, lisible par un humain, est disponible pour être utilisé, étudié, réutilisé, modifié, amélioré, et redistribué par ses utilisateurs. En d'autres mots, l'OSS est du logiciel dont le code est « ouvert ».

Il y a beaucoup de programmes OSS utilisés de manière opérationnelle par le Département aujourd'hui, tant dans des environnements classés, que non-classés. Malheureusement il y a eu des incompréhensions et de mauvaises interprétations des lois, politiques et règlements en vigueur en matière de logiciel, appliquées à l'OSS, qui ont freiné l'utilisation et le développement effectifs de l'OSS par le DoD. L'annexe 2 apporte des clarifications afin de répondre à ces difficultés.

J'ai demandé au Directeur *Enterprise Services & Integration,* de travailler avec vos équipes afin d'identifier de nouvelles barrières à l'utilisation effective[3] de logiciel

[3]L'anglais « effective » peut avoir deux sens, soit « efficace », soit « effectif » dans le sens de « dans la réalité », ou encore « sur le terrain ». Nous opté pour ce second sens, qui semble logique après le mot

open source au sein du Département, de sorte que nous puissions continuer à augmenter les bénéfices que nous tirons de l'usage d'OSS. Plus d'information sur la manière dont les politiques du DoD en vigueur s'appliquent au logiciel open source seront publiées sur http://www.defenselink.mil/cio-nii/cio/oss/, Les questions relatives à ce mémo doivent être adressées à Daniel Risacher, Enterprise Services & Integration, au *[téléphone]* ou par email *[adresse email]*.

[Signature]
David M. Wennergren
ASD (NII)/DoD CIO

Annexe II

Recommandations concernant le logiciel open source

1. Généralités

Cette annexe fournit des éclaircissements et des recommandations complémentaires concernant l'utilisation et le développement de logiciel open source. Il ne modifie pas ou ne crée pas une nouvelle politique, mais vise à expliquer les implications et la signification des lois, politiques et réglements existants.

2. Recommandations

a) Dans pratiquement tous les cas, l'OSS entre dans la définition de « logiciel commercial » et se verra accorder une préférence statutaire conformément à *[un ensemble de références de textes cités]*.

b) Les établissements relevant de l'exécutif, incluant le Département de la Défense, ont l'obligation de mener une étude de marché lorsqu'ils préparent l'acquisition de biens ou de services, selon [référence aux textes de règlements]. Les études de marché concernant le logiciel doivent inclure le logiciel open source lorsqu'il peut satisfaire aux besoins de la mission.

(1) Il y a des aspects positifs à l'OSS qui doivent être pris en compte lorsque l'on mène une étude de marché portant sur le logiciel à l'usage du DoD, par exemple:

barrière, même si l'efficacité est citée par ailleurs.

(a) Le processus le revue par les pairs, large et permanent, qui est rendu possible par la disponibilité publique du code source, renforce la fiabilité et les efforts en matière de sécurité, au travers de l'identification et de l'élimination des défauts qui pourraient ne pas être détectés dans le cas d'une équipe de développement réduite.

(b) La possibilité de modifier le code source du logiciel sans restriction permet au Département de répondre plus rapidement à des situations et des missions changeantes et aux menaces futures.

(c) La dépendance vis à vis d'un développeur ou d'un vendeur particulier, liée aux restrictions propriétaires, peut être réduite par l'utilisation de l'OSS, qui peut être opéré et maintenu par de multiples vendeurs, ce qui réduit les barrières à l'entrée et à la sortie.

(d) Les licences open source ne présentent pas de restriction quant aux utilisateurs du logiciel ou les champs d'application possibles. En conséquence, l'OSS apporte un modèle de licences « net-centric », qui permet un processus d'acquisition couvrant à la fois les utilisateurs identifiés et ceux non-identifiés.

(e) Du fait que l'OSS n'a pas, le plus souvent, un modèle de licence par utilisateur, il peut apporter un avantage en matière de coût dans les cas où de nombreuses copies du logiciel peuvent être nécessaires, et peut amoindrir le risque d'augmentation des coûts lié aux licences, dans des situations où le nombre total d'utilisateurs peut ne pas être connu à l'avance.

(f) En partageant la responsabilité de la maintenance de l'OSS avec d'autres utilisateurs, le Département peut tirer un bénéfice, en réduisant le coût total de possession du logiciel, tout particulièrement en comparaison de logiciels dont le Département porte seul la responsabilité de la maintenance.

(g) L'OSS convient particulièrement au prototypage rapide et à l'expérimentation, où la possibilité de faire des essais du logiciel à des coûts minimaux et des délais administratifs réduits, peut être importante.

(2) Ces considérations sont pertinentes, mais elles ne peuvent pas être les facteurs prééminents d'une décision concernant le logiciel. Au fin de compte, c'est le logiciel qui satisfait le mieux les besoins et les missions du Département qui doit être utilisé, qu'il soit ou non open source.

c) L'instruction 8500.2 du DoD intitulée « Information Assurance (IA) Implementation » comporte un contrôle (…) qui limite l'utilisation des *« exécutables du domaine public ou autres logiciels avec une garantie limitée ou inexistante »* du fait que ces composants sont difficiles ou impossibles à vérifier, réparer ou enrichir, étant donné que le Gouvernement n'a pas accès au code source d'origine et qu'il n'y a pas de propriétaire qui puisse effectuer de telles modifications au nom du gouvernement. Ce contrôle ne doit pas être interprété comme interdisant l'utilisation de l'OSS, puisque son code source est disponible à des fins de vérification, réparation ou enrichissement, par le gouvernement

ou ses prestataires.

d) L'utilisation de n'importe quel logiciel sans maintenance et support présente un risque en termes de sécurité de l'information. Avant d'approuver l'utilisation du logiciel (y compris OSS), les chefs de programmes et chefs de systèmes, et au final les Autorités d'Approbation Désignées, doivent s'assurer que le plan concernant le support du logiciel (…) est approprié pour les besoins de la mission.

e) Il y a une mauvaise interprétation qui voudrait que le Gouvernement soit obligé de distribuer au public le code source de n'importe quel OSS, et que, en conséquence, l'OSS ne devrait pas être intégré ou modifié pour un usage dans des systèmes classifiés ou bien sensibles du DoD. Au contraire, de nombreuses licences open source autorisent l'utilisateur à modifier l'OSS pour un usage interne sans être obligé de distribuer le code source au public. Toutefois, si l'utilisateur choisit de distribuer l'OSS modifié en dehors de son organisation (par exemple un utilisateur du Gouvernement distribue le code en dehors du Gouvernement), alors certaines licences OSS (telles que GNU General Public Licence) imposent effectivement la distribution du code source correspondant, au destinataire du programme. Pour cette raison, il est important de comprendre à la fois les particularités des licences open source en question, et comment le Département envisage d'utiliser et de redistribuer tout OSS qui serait modifié par le DoD.

f) Le code source du logiciel et les documents de conception associés sont des « données », selon la définition du *[Référence]* et en conséquence seront partagés au sein du DoD le plus largement possible pour supporter les besoins de la mission. Les licences open source autorisent une dissémination large du logiciel concerné, ce qui permet à l'OSS d'être partagé à travers tout le Département. Une des manières de rendre le code source disponible dans le Département est d'utiliser l'environnement de développement logiciel collaboratif à l'adresse *[lien http]*, mise en place par le Defense Information Systems Agency.

g) Les composants logiciels, y compris les correctifs et améliorations, développés pour le Gouvernement doivent être rendus disponibles au public (par exemple sous une licence open source) lorsque toutes les conditions suivantes sont remplies:

(1) Le chef de projet ou chef de programme, ou un officiel semblable, estime qu'il est de l'intérêt du Gouvernement de le faire, par exemple parce qu'il s'attend à bénéficier des améliorations apportées par d'autres.

(2) Le Gouvernement a les droits relatifs à la reproduction et à la diffusion des composants, et peut permettre à d'autres d'en faire autant. Par exemple, le Gouvernement a les droits relatifs à la diffusion publique lorsque le logiciel est développé par du personnel du Gouvernement, lorsque le Gouvernement reçoit des « droits illimités » sur un logiciel développé par un prestataire financé par le Gouvernement, ou quand un logiciel OSS pré-existant est modifié par ou pour le Gouvernement.

(3) La diffusion publique n'est pas restreinte par d'autres lois ou réglements, tels que le *Export Administration Regulations* ou le *International trafic in Arms Regulation*, et le composant est éligible pour le *Distribution Statement A* selon

[référence interne].