CULTURE DU WEB

Comprendre l'**open source** et les **logiciels libres**

Histoire, philosophie, licences, support, marché, modèles économiques, modèles de développement

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

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.

Smile compte 290 collaborateurs en France, 330 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). Parmi les ouvrages publiés en 2009, 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. Enfin, Smile est implanté à Paris, Lyon, Nantes, Bordeaux et Montpellier. Et présent également en Espagne, en Suisse, en Ukraine et au Maroc.

[1.2] Ce livre blanc

On parle beaucoup de logiciels libres et d'open source, mais en creusant un peu, il apparaît que nombreux sont ceux, même parmi les professionnels de l'informatique, qui ont une connaissance et une compréhension assez superficielles du phénomène.

D'un côté les passionnés, engagés, qui se régalent de la démarche communautaire, mais ne connaissent pas toujours les aspects économiques, de l'autre les décideurs du monde de l'entreprise, qui sont de plus en plus sensibles aux bénéfices des solutions open source, mais en connaissent mal la philosophie, l'histoire, ou même les questions de licences.

Ce livre blanc est une introduction au phénomène de l'open source, la plus grande révolution qui touche l'informatique depuis l'Internet. Comme on le verra, le mouvement est bien antérieur au web, néanmoins la puissante perturbation sur l'économie de l'informatique date de ces dernières années, et ne fait que commencer.

Cet ouvrage a une vocation de vulgarisation, s'efforçant surtout d'expliquer l'open source à ceux qui n'y sont pas impliqués, mais commencent à en sentir l'importance, et ont besoin de mieux connaître le phénomène.

Notons que le monde de l'open source est sujet à diverses « controverses », qui enflamment les esprits et scindent les communautés depuis de longues années. À commencer par l'appellation *logiciel libre* versus *logiciel open source* ou encore GNU/Linux versus Linux. Même s'il faut les mentionner, nous passerons rapidement sur ces disputes internes, pour mieux nous focaliser sur ce qui nous semble être plus fondamental.

Je remercie chaleureusement les personnes qui ont bien voulu me faire part de leurs remarques, corrections et enrichissements, en particulier Benoit Jacquemont, Frédéric Couché et Benjamin Jean.

Patrice Bertrand
Directeur Général

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

Table des matières

[2] INTRODUCTION

[2.1] Terminologie

Le code source est la version d'un programme qui est lisible et intelligible pour l'homme. C'est le code source qui est écrit par l'informaticien, le programmeur, et qui pourra être relu et modifié par d'autres. Les programmes peuvent ensuite être compilés, ce qui produit le code *objet*, ou *binaire*, ou encore *exécutable*, qui lui n'est pas compréhensible.

Un *logiciel libre*, ou *logiciel open source*, est un programme dont le code source est distribué et peut être utilisé, copié, étudié, modifié et redistribué sans restriction.

Notons qu'il existe des langages informatiques interprétés, tels le PHP, qui n'existent pas autrement que sous forme de code source. Mais même lorsque le code source est *disponible*, il n'est pas toujours *autorisé* de le modifier. Ce sont les termes de *la licence*, concédée par l'auteur ou le *détenteur des droits*, qui précisent s'il est permis ou non de modifier le code, de le réutiliser, de le redistribuer, et sous quelles conditions.

Logiciel libre est la juste traduction française de *free software*, l'appellation lancée par Richard Stallman et défendue par la *Free Software Foundation,* la FSF.

Open source est l'appellation de l'*Open Source Initiative*, qui édicte sur le site *opensource.org* les conditions que doit satisfaire une licence pour se dire *open source*.

Le *logiciel libre* est défini par quatre *libertés fondamentales* : exécuter le programme, l'étudier, l'adapter, le redistribuer. Il faut souligner que le libre accès au code source est simplement rendu nécessaire par ces libertés fondamentales, et non une fin en soi.

Le *logiciel open source* se définit par les 10 articles de l'*open source definition*, sur laquelle nous reviendrons plus loin.

Les deux appellations sont presque équivalentes, mais correspondent à des écoles de pensées différentes. Aucune n'acceptant d'être englobée par l'autre, les américains utilisent parfois le terme de *FOSS* pour *« Free and Open Source Software »*, ou encore *FLOSS* pour *« Free/Libre and Open Source Software »*.

Nous avons estimé qu'utiliser « FLOSS » dans tout le corps de ce livre blanc serait pesant pour le lecteur, et avons pris le parti d'utiliser le terme *open source*. Notons toutefois que FLOSS est le terme officiel adopté par la commission européenne.

[2.2] Philosophie de l'open source

Une liberté fondamentale

Pour Richard Matthew Stallman, le père de la Free Software Foundation (1985), « RMS » pour les intimes, le logiciel libre est avant tout affaire de *liberté*. La liberté que doit avoir chaque individu d'utiliser, modifier, et redistribuer *n'importe quel programme*. Une liberté aussi fondamentale que la liberté d'expression. Et indissociable d'autres valeurs, d'éthique et de responsabilité sociale.

Dans cette logique, un logiciel non libre, « propriétaire » donc, porte atteinte à cette liberté fondamentale. Le logiciel libre n'est donc pas une simple alternative, et encore moins le choix d'un business model parmi d'autres. Le logiciel

propriétaire est « privateur » dans le sens où il prive de liberté, et il est en ce sens intolérable. Il s'agit bel et bien une lutte du bien contre le mal.

On pourrait sourire de ce manichéisme, mais lorsque l'on voit l'extraordinaire patrimoine de logiciels que le mouvement initiée par Stallman a mis à disposition de tous, on se doit d'être surtout admiratif et reconnaissant. On ne fait pas une révolution avec des idées molles, et il fallait l'intransigeance de Stallman, pour créer une vraie rupture, et un mouvement de pensée profond, où la liberté va de pair avec des valeurs de solidarité sociale et d'entraide.

Un modèle de développement

Pour Eric Raymond, il ne s'agit guère d'éthique, ou même de philosophie, il est question avant tout de démontrer la *supériorité* des logiciels réalisés selon un modèle de développement open source communautaire, et de les faire *entrer dans la sphère économique*.

Pour Eric Raymond, le dogmatisme de la FSF ne joue pas en faveur du mouvement, et ce sont des logiciels de qualité supérieure, plus que les valeurs éthiques, qui imposeront l'open source.

Avec Bruce Perens, il fonde l'Open Source Initiative en 1998, pour promouvoir l'open source (cf. « Eric S. Raymond et l'OSI », page 15). Le mouvement « open source » apparaît à certain comme une *opération de marketing* en faveur du logiciel libre. Mais pour Richard Stallman, il n'est pas permis de jeter au passage les valeurs fondatrices, en particulier de liberté.

Dix ans plus tard, la cicatrice de cette scission n'est pas refermée entre logiciel libre et open source, et l'on ne peut choisir une appellation plutôt qu'une autre sans s'attirer les foudres de l'un des camps. Dans la pratique, Stallman convient que *« les deux termes décrivent pratiquement la même catégorie de logiciel. Mais ils représentent des vues basées sur des valeurs fondamentalement différentes. »*

Un patrimoine de l'humanité

Enfin, nous proposons ici notre propre vision de l'open source, non pas tant affaire de liberté, mais de progrès et de patrimoine. Voici le texte d'un article qui présente ce point de vue.

"Nous sommes des nains sur les épaules de géants". C'est dans le domaine des sciences que l'on entend cette pensée. Et en effet, les savants d'aujourd'hui ne sont pas plus intelligents que ceux d'hier, mais ils bénéficient, dès leur formation, de siècles de science accumulée et c'est sur ce socle immense construit par Newton, Einstein et les autres, qu'ils apportent leurs petites pierres.

L'informatique n'est pas exactement une science. Mais doit-elle pour autant tout reconstruire à chaque génération ? Si c'était le cas, elle serait condamnée à toucher rapidement ses limites. Les informaticiens d'aujourd'hui sont-ils plus doués que ceux d'hier ? Certainement pas. Ont-ils appris plus de choses en cours ? Un peu sans doute. Mais cela ne suffirait pas à s'élancer plus loin.

Car si, en sciences, le patrimoine est entièrement dans le savoir, en informatique, il y a deux patrimoines : la connaissance d'une part, le code source d'autre part. La connaissance progresse lentement et il y a peu de savoirs fondamentaux pour bâtir, disons, Firefox ou bien Eclipse, qui étaient inconnus il y a 15 ans. Si l'informatique progresse, c'est plus par le patrimoine de code source que par la connaissance, c'est-à-dire que l'on peut s'appuyer aujourd'hui sur un immense socle de code source.

Dans les premiers temps, les informaticiens devaient tout créer, pratiquement pour chaque programme. Puis, les systèmes d'exploitation ont

amené un premier niveau de socle, qui est devenu plus sophistiqué au fil des années, et les langages de haut niveau ont amené des bibliothèques de plus en plus riches.

Sur ce socle élémentaire, nous avons ajouté différents socles de développement, des *frameworks*, qui constituent une seconde couche. Et ce n'est pas tout : nous disposons aussi d'une quantité de composants de haut niveau, que nous pouvons assembler pour construire des applications nouvelles. Au total, 90% du code déroulé dans ces applications sera issu, soit du système d'exploitation, soit des *frameworks*, soit des composants. Et nous n'aurons réellement développé que les 10% de valeur ajoutée spécifique.

C'est un constat important : l'informatique progresse essentiellement parce que le socle de code qui constitue notre patrimoine s'agrandit.

Si, dans un effort gigantesque, je réalise un programme nouveau, représentant disons un million de lignes de code originales, que ce programme répond à un besoin et qu'il est un succès commercial, c'est certes une belle aventure, qui m'enrichira peut-être et sera utile à mes clients.

Mais je n'aurai pas réellement fait progresser l'informatique d'un pouce, car trois ans après moi, si un autre veut aller plus loin dans cette voie, pour faire un meilleur programme sans disposer du mien, il lui faudra repartir d'où j'étais parti, ré-écrire mon premier million de lignes de code, pour enfin y ajouter 200 000 lignes qui l'amèneront un peu plus loin. Ne pouvant grimper sur mes épaules, il a les deux pieds dans la même boue que moi, et n'a d'autre choix que d'être géant lui-même.

C'est la dimension humaniste de l'open source que de considérer que nous apportons chacun notre pierre, ajoutant à ce patrimoine commun, qui nous permettra d'aller plus loin. »

[2.3] Bière gratuite ?!

« *Free* » voulant dire à la fois « gratuit » et « libre », les tenants du *free software* s'évertuent à faire comprendre qu'il s'agit bien de liberté et non de gratuité, selon la formule *« free as in 'free speech' and 'free market', not as in 'free beer' »*.

En français, nous n'avons pas cette ambiguïté, mais nous avons gardé la formule « logiciel libre ne signifie pas gratuit ». Et de cette formule, certains comprennent qu'un logiciel libre peut être payant. Ce n'est pas strictement faux, mais presque. Expliquons.

Rien en effet, dans les licences open source, n'interdit de faire payer la distribution du logiciel. Mais celui à qui vous le distribuez sera autorisé à le dupliquer et le redistribuer gratuitement s'il le souhaite. On voit qu'il est bien difficile de vendre quelque chose que d'autres peuvent donner !

Comme nous le verrons en expliquant les modèles économiques, on ne peut faire payer un droit d'utilisation d'un logiciel open source. On peut faire payer des prestations associées (intégration, support, formation, etc), et/ou un droit d'utilisation associé à une licence non open source du logiciel.

Donc dans la pratique, il faut retenir que *un logiciel open source est bel et bien gratuit*, d'acquisition comme d'utilisation, du point de vue de sa licence.

Comme nous le verrons, cela n'empêche pas qu'il soit accompagné d'une offre de services payants : intégration, support, formations, développements complémentaires, voire même assurance juridique. De sorte que son « *coût total de possession* » est rarement nul, même s'il est presque toujours inférieur à celui d'une solution propriétaire équivalente.

[2.4] Les bénéfices de l'open source pour le client

Pas seulement moins cher…

Bien sûr, les bénéfices économiques sont parmi les premières raisons dans le choix de solutions open source. Même si « libre ne signifie pas gratuit », ces solutions ont toujours un coût de possession sensiblement moins élevé que leurs équivalents propriétaires.

D'autant que les prix de prestations tendent aussi à être moins élevés, car l'ouverture du produit facilite la diffusion de la connaissance.

Mais au fur et à mesure que ces solutions arrivent à maturité, le moindre coût n'est plus le premier critère de choix.

Les principaux arguments sont alors :

- La non-dépendance, ou moindre dépendance, par rapport à un éditeur. On sait que changer d'outil peut coûter très cher, et les éditeurs peuvent être tentés de profiter de la vache à lait que constituent ces clients devenus captifs. En anglais, on parle de *vendor lock-in*, le verrouillage par le fournisseur.
- L'ouverture est également un argument de poids. Les solutions open source sont en général plus respectueuses des standards, et plus ouvertes vers l'ajout de modules d'extension.
- La pérennité est un autre critère de choix fort, nous y revenons plus loin.
- Et la qualité finalement, car dans beaucoup de domaines les solutions open source sont réellement, objectivement, supérieures. Le très grand nombre de déploiements et donc de retours d'expérience, mais aussi leur modèle de développement et leur intégration de composants de haut niveau, permet à beaucoup de surclasser les produits propriétaires souvent vieillissants.

A quoi on peut ajouter le plaisir, pour les informaticiens, d'utiliser des programmes dont ils peuvent acquérir une totale maîtrise, sans barrière ni technique ni juridique.

La pérennité

En matière de pérennité, les solutions open source n'ont pas une garantie d'éternelle jouvence. Elles peuvent mourir, aussi, mais de mort lente !

Le pire qu'il puisse arriver pour une solution open source est une désaffection progressive de la part des communautés, généralement au profit d'une solution plus prometteuse. Ainsi, il est possible qu'il faille un jour changer de produit. Mais du moins le phénomène est toujours lent, et le client a le temps d'organiser la migration.

Il faut souligner aussi que, même si l'éditeur original était un jour défaillant, il resterait toujours possible pour une communauté de reprendre en main le produit et ses évolutions, c'est le principe des licences open source.

Le notoriété, l'envergure des déploiements, la dynamique du développement et de la communauté, ces critères de pérennité sont relativement facile à évaluer, et une solution open source leader offre une garantie de pérennité supérieure à la majorité des solutions propriétaires.

L'ouverture

Un mot également sur la question de l'ouverture. La possibilité de faire des modifications dans les sources est fondamentale sur le plan théorique, mais

parfois risquée sur le plan pratique (cf. « Maîtriser les sources : un droit, non un devoir », page 10). Ce n'est donc pas en ces termes qu'il faut apprécier l'ouverture, mais plutôt dans la capacité à accepter des extensions, ou à s'interfacer à d'autres applications.

Sur le fond, il faut comprendre qu'un éditeur à vocation commerciale n'a pas que des intérêts convergents avec ceux de ses clients. Certes, il évolue dans un marché concurrentiel, et son produit doit être au niveau de ses concurrents. Mais une fois sa position bien assise, l'éditeur peut faire l'analyse que :

- Son produit doit être performant, mais pas trop, car s'il faut plus de serveurs, ce sera davantage de licences vendues.
- Son produit doit être robuste, mais pas trop, car il faut continuer à vendre du support.
- Son produit doit être ouvert, mais pas trop, pour garder la maîtrise du client.

Nous ne disons pas que les éditeurs propriétaires seraient machiavéliques au point de dégrader ces qualités dans leur produit, nous disons seulement que la priorité stratégique n'est pas nécessairement mise sur ces qualités.

En matière d'ouverture, enfin, il faut souligner que le logiciel propriétaire n'est pas la seule manière d'enfermer un client. Les formats de documents sont aussi une arme puissante pour parvenir au *verrouillage du client*. Ces dernières années, on a pu voir une forte prise de conscience de l'importance des *formats ouverts*, c'est-à-dire à la fois documentés, et d'utilisation libre. Ils sont à la fois la condition de l'indépendance, mais aussi de la pérennité des documents, et de l'interopérabilité des applications partageant ces documents[2].

[2]Pour en savoir plus: http://blog.smile.fr/documents-ouverts-un-pont-entre-bureautique-et-gestion-de-contenus

La sécurité

Le domaine de la sécurité mérite une mention spéciale. Car en matière de sécurité, l'accès aux sources est quasiment une obligation. On ne concevrait pas que l'armée française utilise pour ses communications un VPN reçu sous forme d'exécutable d'un éditeur américain ou chinois.

En matière de sécurité, il est absolument obligatoire de pouvoir auditer ce qu'un programme fait *vraiment*, et cela ne peut se faire qu'en analysant ses sources.

Pour autant, cela n'implique pas que le programme soit open source. Certains éditeurs non open source acceptent de livrer les sources à leurs clients, après signature d'un accord de non divulgation.

Mais il est un autre argument qui rend l'open source indispensable ici : le *peer review*, la validation des pairs, c'est à dire d'autres experts, et du plus grand nombre possible d'autres experts.

Faisons un petit parallèle. Dans ses débuts, la cryptographie utilisait majoritairement des algorithmes secrets. On considérait alors que la protection de l'algorithme contribuait à la sécurité. Dans l'après-guerre, une révolution s'est amorcée : on a finalement conclu qu'un algorithme secret, dont la qualité n'est affirmée que par la petite équipe qui l'a créé, avait de fortes chances d'être défaillant, si ce n'est aujourd'hui alors sans doutes dans quelques années. Et a contrario, les algorithmes qui sont exposés sur la place publique, sont analysés par des centaines d'experts dans le monde. S'ils ont une faille, elle est rapidement identifiée et connue. On peut donc en dire autant des programmes qui exécutent ces algorithmes : le meilleur moyen d'être assuré de leur perfection est de les exposer à l'audit de milliers d'experts.

Enfin ajoutons un dernier argument : en matière de sécurité, on préfère

généralement les vieux algorithmes, qui ont fait leurs preuves, et on se méfie des dernières innovations. L'algorithme RSA date de 1977 ! Il est naturel que depuis ce temps, les programmes qui implémentent ces algorithmes soient parvenus dans le patrimoine commun, sinon dans le domaine public.

Il est donc naturel que le portail du gouvernement consacré à la sécurité informatique[3] accorde une place importante au logiciel libre.

Maîtriser les sources : un droit, non un devoir

Il faut souligner, car c'est souvent mal compris, qu'il n'est nullement nécessaire de maîtriser les sources d'un produit open source, pour le déployer, l'utiliser et en tirer bénéfice. Ni de les maîtriser, ni de les regarder, ni même de les télécharger.

Il y a quelques années encore, certains produits open source s'attachaient à ne diffuser *que* les sources, obligeant l'utilisateur à recompiler et générer son programme. Cette démarche un peu extrémiste est aujourd'hui abandonnée car elle nuit à la diffusion de l'open source.

Prendre connaissance des sources est un droit et non un devoir.

De même, modifier les sources est un droit fondamental, mais dans beaucoup de cas c'est une chose qui n'est pas recommandée. Cela pour plusieurs raisons :

- Il y a un risque important de fragiliser le produit, parce que votre code sera moins bien testé que le reste, et que vous l'aurez écrit avec une moindre maîtrise de l'ensemble.
- Sur des produits d'envergure, apporter des modifications demande un grand investissement, une grande implication, et donc un budget sérieux.
- Votre version modifiée est donc un *fork*, une version alternative, du produit. Elle ne bénéficiera pas, ou plus difficilement, du support tant éditeur que communautaire, et il faudra réintroduire vos modifications dans les nouvelles versions pour pouvoir en bénéficier.

Dans une majorité de cas, ces raisons l'emportent. Pourtant, si elles devaient bloquer toute forme de contribution, la vitalité de l'open source serait compromise.

Ce qu'il faut, c'est que chacun, développeur indépendant ou organisation, mesure l'investissement qu'il peut faire sur un projet, s'y implique en échangeant abondamment avec les autres développeurs du projet, et effectue ses modifications non pas dans son coin, mais dans le référentiel commun.

C'est à dire qu'il est tout à fait souhaitable d'enrichir le produit *au sein de la communauté* ou en liaison avec l'éditeur, mais qu'il n'est en général pas souhaitable de le faire autrement.

Pseudo open-source

En théorie, il suffit de proposer ses sources sous une licence agréée pour se revendiquer logiciel open source. Pour les utilisateurs toutefois, il faut se défier des produits pseudo-open-source.

Il arrive couramment que des éditeurs de solutions propriétaires en échec sur le marché, particulièrement face à la montée en puissance de solutions concurrentes open source, prennent un ultime revirement stratégique avant de disparaître, en déclarant que leur produit devient open source. Ils diffusent les sources avec plus ou moins de bonne volonté, et envoient leurs commerciaux clamer sur le marché qu'ils sont désormais aussi ouverts que leurs

[3] http://www.securite-informatique.gouv.fr/

concurrents open source. Mais le cœur n'y est pas, et ils ont la ferme résolution de ne laisser personne prendre la maîtrise de leur code, et de garder la mainmise sur la totalité des déploiements.

Pour les clients, ces solutions sont la pire des voies car ils n'auront au bout du compte aucun des bénéfices de l'open source, et en particulier subiront le même verrouillage, le *vendor lock-in*, qu'avec une solution propriétaire. Mais plus grave que ça : l'expérience montre que ces solutions disparaissent presque toujours dans l'année qui suit.

[3] LE MARCHÉ DE L'OPEN SOURCE

[3.1] Quelques études

Tous les analystes, tant en France qu'aux États-Unis, s'accordent à percevoir l'extraordinaire percée des solutions open source dans la sphère économique ces dernières années, et à la prolonger sur les années à venir.

Il est toujours difficile de mesurer la pénétration de l'open source en termes de milliards d'euros. Dans la mesure où une part importante des produits sont utilisés gratuitement, la part de marché en termes de déploiement est immensément plus grande que la part de marché en termes de chiffre d'affaire. Il conviendrait de mesurer le marché de l'open source en termes de *valeur de remplacement propriétaire*, c'est à dire valeur marchande d'un produit propriétaire équivalent.

En France, le Syntec estimait pour sa part, dans une étude de 2007, que le marché des logiciels et services open source représente 450 M€ sur un marché total des logiciels et services de plus de 30 Md€, soit une part de marché de 1,4%. Ce marché devrait croître de 50% par an, sur un marché en croissance de 6,5%, ce qui amène la part de marché du logiciel open source à doubler en deux ans. Le Syntec estime que la tendance sera à la mise en place de systèmes d'information mêlant open source et propriétaire sans préjugés.

Selon une étude 2007 de Pierre Audoin Consultants, le marché de l'open source en France progresse d'environ 70% par an. En France, le secteur public tient une place particulière, et l'étude Markess, de 2007, estime que le secteur public consacre en moyenne 11% de ses budgets informatiques aux technologies libres, contre 7% en 2006, et 14% en 2009. Les raisons invoquées par les décideurs sont autant les contraintes budgétaires que le besoin d'indépendance et d'interopérabilité.

La France apparaît comme un précurseur dans ce domaine, mais de son coté IDC estime que le marché mondial de l'open source passera de 2 Md$ à 6 Md$ en 2011.

Une étude citée par « Le Nouvel Economiste » en mars 2010 donnait les chiffres suivants pour le marché de l'open source:

	2008	2010	2012	CAG
UE	3,5	7,4	12	25%
France	1,1	1,9	2,9	13%
Allemagne	0,7	1,5	2,5	27%
Royaume-Uni	0,7	1,5	2,5	27%

La colonne « CAG » à droite est le taux de croissance moyen annuel.

Enfin, citons également l'étude américaine de Saugatuck Technologies, qui évalue à 10% la part des logiciels utilisés en entreprise aux U.S. qui sont des logiciels open source, et estime qu'elle passera à 15-20% d'ici à 2010.

Mais au delà des parts de marché, tous les analystes s'accordent à penser que la

pénétration de l'open source dans un nombre croissant de domaines est un des facteurs les plus importants de réduction des coûts informatiques dans les années à venir.

[3.2] Une vague puissante

Comme on le verra plus loin, l'open source est loin d'être un phénomène nouveau. Dans certains domaines, cette ancienneté fait partie de ses atouts : les logiciels open source se sont bonifiés avec les années, sont devenus toujours plus robustes et ont vu asseoir également leur part de marché, en particulier dans les couches d'infrastructure et les outils de développement.

Mais ces dernières années ont vu une sensible accélération de deux phénomènes plus nouveaux.

Le premier est que les entreprises, y compris les plus grandes d'entre elles, n'ont plus aucune réticence vis à vis de l'open source. Les grandes DSI et les Directions des Achats ont compris qu'elles pouvaient y trouver à la fois des produits particulièrement solides et de vrais bénéfices économiques. On constate que de plus en plus d'appels d'offres mentionnent, et parfois exigent, des solutions open source.

Le second est l'apparition d'acteurs nouveaux, les éditeurs de solutions open source commerciales. A la manière des compagnies aériennes low-cost, ces nouveaux entrants s'appuient sur un business model différent pour apporter une dynamique nouvelle dans un paysage informatique souvent sclérosé. Base de données, gestion de contenus, CRM, ERP, Décisionnel, … dans un nombre toujours croissant de domaines, ces acteurs nouveaux révolutionnent le marché avec un rapport service/prix inégalé.

La rencontre des entreprises ouvertes à l'open source, et de ces solutions toujours plus riches, est rendue possible par des prestataires informatiques spécialisés, qui investissent dans la construction d'une forte expertise, et sont capables d'offrir un support de qualité.

[3.3] Une analyse économique

Il nous semble intéressant de citer ici une analyse de Tim O'Reilly, l'éditeur de la collection du même nom, et l'un des penseurs de l'open source.

Dans un article de 2003[4], il revient sur la rupture provoquée par la banalisation du matériel, entamée en 1981, lorsque IBM crée un marché du PC compatible en ouvrant son architecture. Il utilise le terme de « *commoditization* », qui fait référence aux « *commodities* », les biens ordinaires tels que le blé ou le pétrole, des biens dont le prix peut fluctuer, mais où il n'y a plus guère de valeur ajoutée spécifique, qui sont interchangeables, banalisés.

La banalisation du matériel va donner naissance à une immense industrie du logiciel, dominée par Microsoft. Et donner naissance également à Dell, qui comprendra le premier que le matériel est devenu une simple denrée industrielle.

Vingt ans plus tard, l'open source apporte une rupture comparable, un *changement de paradigme*, la banalisation du logiciel, voire sa démonétisation. Système d'exploitation, serveurs, bases de données, ces composants logiciels ont perdu l'essentiel de la valeur marchande qu'ils portaient.

[4] http://www.oreillynet.com/pub/a/oreilly/tim/articles/paradigmshift_0504.html, Tim O'Reilly s'appuie également sur une analyse antérieure de Ian Murdock : http://ianmurdock.com/open-source-and-the-commoditization-of-software/

Et cette banalisation a donné naissance à une nouvelle industrie, dont les tenants sont Google, Amazon, eBay, ou Facebook. Les nouveaux géants du web, qui utilisent des centaines de milliers de serveurs, ont besoin de logiciels démonétisés.

Certains ont dénoncé une destruction de valeur, lorsque les éditeurs traditionnels perdent des parts de marché face à la concurrence de solutions open source, ou bien sont contraints de baisser leurs prix de manière drastique. Mais c'est le propre de tout progrès, quel que soit le domaine, que d'apporter une telle perturbation, une « destruction créative », selon l'expression consacrée. Au final, le moindre coût des programmes apporte un gain de compétitivité pour toutes les industries qui consomment du logiciel, et donc un gain de niveau de vie pour chacun.

Aux États-Unis, des personnes en apparence intelligentes comme Steve Ballmer ont comparé l'open source au communisme, injure suprême !

D'une certaine manière, on pourrait dire : c'est tout le contraire, l'open source est un pur produit du capitalisme. L'une des lois du capitalisme n'est-elle pas que dès lors qu'un acteur tire un profit exagéré de sa position sur le marché, il apparaît des acteurs concurrents pour ramener un niveau de profit raisonnable ? Mais depuis longtemps déjà cette loi élémentaire de la concurrence ne semblait plus pouvoir jouer dans l'édition logicielle. C'est finalement l'open source qui ramènera un niveau de profit raisonnable dans l'industrie du logiciel. Si 100 millions de personnes sur terre ont besoin d'une suite bureautique, alors il suffit qu'ils dépensent chacun 0,1 € par an pour financer un effort de développement satisfaisant. D'une manière indirecte, c'est ce juste prix qu'apporte l'open source.

Pour autant, il existe aussi des domaines où la banalisation n'est pas à l'ordre du jour, et où au contraire c'est l'open source qui apporte une nouvelle dynamique de progrès dans des marchés sclérosés. Mais toujours en réhabilitant la concurrence, et donc l'innovation, ainsi que le retour à un juste prix de marché.

D'une manière générale, avec l'open source c'est l'expertise, c'est à dire la *connaissance* qui prend toute sa valeur, au détriment de la simple *propriété* ou antériorité. La monétisation de la connaissance est simplement proportionnelle à la rareté de l'expertise au regard de la demande, selon des lois de marché ordinaires.

[4] HISTOIRE ET GRANDES FIGURES

[4.1] Les hackers

D'une certaine manière, l'open source est aussi ancien que l'informatique, il est important de le souligner.

Lorsque, dans les années 60, les premiers ordinateurs arrivent dans les universités, l'accès libre aux programmes est la norme. Lorsqu'un universitaire trouve une nouvelle molécule, il montre le procédé à ses collègues, lorsqu'il écrit un programme intéressant, il le montre à ses collègues. C'est la démarche normale du progrès scientifique.

Les années 60 et 70 sont sous le signe des *hackers,* le plus souvent des étudiants brillants, des meilleures universités américaines, qui se jettent avec passion dans les premiers balbutiements de l'informatique. Ils passent des nuits sur leurs programmes, attendant de pouvoir accéder quelques heures à un peu de temps-machine, qui est une denrée rare. Ils partagent leurs astuces et leurs programmes, au sein de différents clubs.

En 1962, Spacewar, un programme réalisé au MIT, est parfois cité comme le premier projet open source, en même temps que le premier jeu vidéo. Créé par une petite équipe, il s'enrichit ensuite pendant plusieurs années, grâce aux multiples contributions rendues possibles par le libre accès au code source.

Le terme *hacker* de cette époque n'a pas la connotation sulfureuse d'aujourd'hui : un *hacker* est alors un programmeur à la fois passionné et surdoué. Pas très éloignés des *nerds, « polards »* en français, ce sont eux qui posent les fondations de l'informatique moderne et beaucoup créeront les entreprises leaders d'aujourd'hui.

Les *hackers* ont une philosophie, la *hacker ethic*, qui prône le libre accès aux ordinateurs et aux programmes, et d'une manière plus large la gratuité de l'information. Ils sont globalement méfiants de l'autorité – en phase avec les mouvements étudiants des années 60 – mais surtout, ils sont convaincus qu'il y a de la beauté, de l'art, dans un programme, et que l'informatique peut amener un monde meilleur.

C'est dans les années 70 que la pratique de *ne pas* diffuser les codes source des programmes s'est répandue, et que le business model de l'éditeur de logiciel propriétaire est apparu.

On pourrait retenir comme date marquante de la scission entre le logiciel libre et le logiciel propriétaire la réunion du *Homebrew Computer Club*, en 1976. Lors de cette réunion, Bill Gates et Paul Allen présentent un programme interpréteur de langage Basic, qu'ils ont écrit pour le Altair 100, un des premiers ordinateurs à microprocesseur. Les membres du club prennent la bande perforée représentant le programme, la dupliquent et la diffusent. Bill Gates, furieux, écrira une lettre devenue fameuse, intitulée *Lettre Ouverte aux Hobbyist*[5], dans laquelle il explique que le travail des développeurs doit pouvoir être justement rémunéré, et que s'il ne l'est pas, c'est l'innovation qui sera étouffée. Le raisonnement est juste, et pourtant l'avenir montrera qu'il est également possible de réaliser de grands programmes en open source.

[4.2] Richard M. Stallman et la FSF

On peut considérer Richard Matthew Stallman comme le père fondateur du logiciel libre en tant que courant de pensée, et il est décrit parfois comme le dernier des vrais hackers. Dès 1983, il souhaite un système d'exploitation et des outils libres d'utilisation, en lançant le projet GNU, qui vise à créer le premier système d'exploitation libre, inspiré de Unix. En 1990, le projet est bien avancé, avec en particulier un excellent compilateur C (GCC), un éditeur réputé (Emacs), et une grande panoplie d'utilitaires. Mais le noyau (GNU Hurd) est à peine commencé lorsque Linus Torvalds sort son noyau Linux.

En 1985, Stallman fonde la Free Software Foundation (FSF), qui est à la fois l'entité en charge du projet GNU, un lieu de réflexion et un vecteur de promotion et de défense du logiciel libre. La FSF a créé la licence GNU GPL, et son évolution récente en v3 (cf. « La licence GNU GPL », page 19).

Richard Stallman est une personnalité atypique, penseur et activiste, en même temps que hacker. Il continue de sillonner le monde aujourd'hui, pour faire la promotion du logiciel libre, et ne permet pas que l'on oublie les valeurs fondatrices du mouvement.

[5] http://en.wikipedia.org/wiki/Open_Letter_to_Hobbyists

[4.3] Linus Torvalds

En 1991, Linus Torvalds, étudiant finlandais âgé de 21 ans, travaille à développer un noyau de système d'exploitation. Il s'inspire en partie de Minix, un noyau expérimental qui accompagne le livre de Andrew Tanenbaum, ouvrage de référence depuis 1987 : *« Operating Systems : design and implementation »*. En quelques mois de travail, il sort la version 0.01. Fin 1991, Linux passe sous licence GPL, ce qui contribue à lancer une forte dynamique de développement communautaire, qui conduira à la version 1.0 de Linux en 1994.

Linus Torvalds est plus un architecte et développeur qu'un penseur ou un militant de l'open source ; il est respecté par tous, mais prend rarement part aux débats enflammés qui secouent les communautés. Aujourd'hui encore, c'est lui qui arbitre les orientations importantes du noyau Linux.

Soulignons que le système d'exploitation est constitué du noyau et d'un grand nombre de composants utilitaires sans lesquels on ne saurait utiliser le noyau. Une majorité des composants entourant le noyau Linux étant issus du projet GNU, Richard Stallman estime qu'il convient de toujours appeler le système GNU/Linux, en reconnaissance des apports du projet GNU.

[4.4] Eric S. Raymond et l'OSI

Eric S. Raymond est l'un des avocats célèbres de l'open source, dans la fin des années 90. Il a écrit différents ouvrages dont 'La Cathédrale et le Bazar', un des textes fondateurs du mouvement[6].

Il défend principalement la supériorité du modèle de développement, donc de la qualité des applications, davantage que les questions morales et humanistes.

Contrairement à Stallman, Raymond n'est pas lui-même un hacker de haut vol, il est davantage un penseur de l'open source. Il s'est opposé à Stallman dans différents articles, estimant que les positions intégristes de ce dernier pouvaient desservir le mouvement.

Eric Raymond est, avec Bruce Perens, l'un des fondateurs de l'*Open Source Initiative (OSI)*, qu'il crée en 1998, l'année où la mise en open source du navigateur Mozilla marquera une victoire symbolique du mouvement. Encore aujourd'hui, l'OSI est un peu le gardien du temple de l'open source au travers de son site opensource.org, qui porte la *définition officielle d'une licence open source*, publiée sur le site, et qui fait consensus.

Le site opensource.org publie également un recensement des licences open source agréées, une soixantaine de licences.

[4.5] Les grandes dates de l'open source

60-70	■ Les années hacker – voir plus haut
1983	■ Année du "GNU Manifesto" de Richard Stallman.
1984	■ Début du développement du projet GNU, premier système d'exploitation libre ; le noyau Hurd ne sera démarré qu'en 1990.
1985	■ La Free Software Foundation ■ Distribution de la couche graphique X Window en open source par le MIT

[6] http://catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

1989
- Création de la licence GNU GPL

1991
- Linus Torvalds diffuse la première version de Linux

1993
- Création de la distribution Linux Debian
- FreeBSD 1.0

1994
- Première distribution RedHat

1995
- Première version du serveur Apache Httpd

1997
- « La Cathédrale et le Bazar » de Eric S. Raymond

1998
- Netscape livre Mozilla en open source
- Debian 2.0
- IBM choisit le serveur Http Apache pour son offre web
- Les « Halloween Documents » notes internes de Microsoft sont mis sur la place publique (cf « Erreur : source de la référence non trouvée », page Erreur : source de la référence non trouvée)

1998
- Première version de Typo3

1999
- Introduction en bourse de Redhat

2000
- SUN ouvre la suite Open Office en open source.
- Première version de eZ Publish
- Smile déploie Cofax, CMS open source pour le CEA et Egide.

2001
- Introduction en bourse de Mandriva

2003
- SCO, avec l'aide de Microsoft, attaque IBM et quelques autres, invoquant des droits sur Linux.

2005
- Création de Alfresco, éditeur open source d'une solution de GED

2006
- Redhat acquiert JBoss, pour 350 millions de dollars.
- SUN annonce le passage de Java sous GPL

2007
- L'Assemblé Nationale Française adopte Linux pour les postes de travail des députés.

2008
- SUN rachète MySql pour 1 milliard de dollars.
- Google lance Android, un OS open source pour smartphones

2009
- Oracle rachète SUN pour 7 milliards de dollars
- Nokia ouvre sa plateforme Symbian en open source

2010
- Les entreprises françaises du logiciel libre, organisées en associations régionales, se réunissent au sein du Conseil National du Logiciel Libre.

[5] COPYRIGHT ET LICENCES

[5.1] 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 :

1. Libre redistribution : la licence ne doit pas interdire à qui que ce soit de vendre ou donner le programme.
2. 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.
3. 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, *non-copyleft* et *copyleft*.

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.

Enfin, nous verrons plus loin quelques licences dites *weak-copyleft,* qui ont des exigences intermédiaires.

[5.2] 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.

[5.3] La licence 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*.

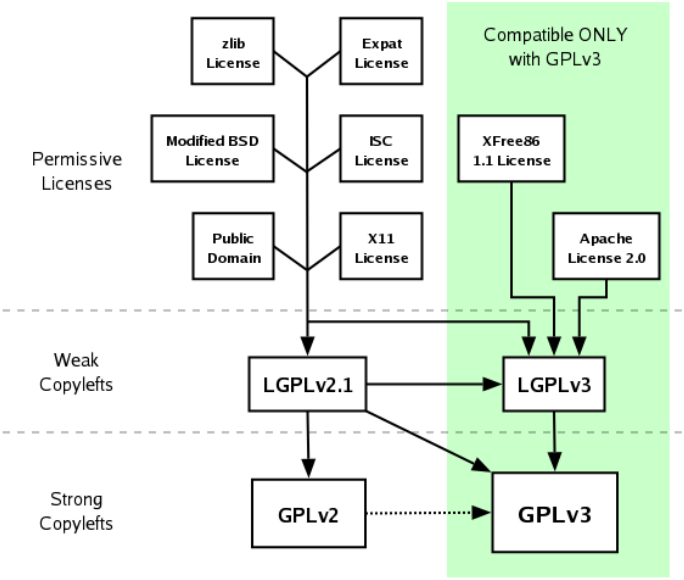
Compatibilité des licences

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[7] 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 librairies sous

[7] Image © 2007 Free Software Foundation Inc.

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.

[5.4] Propriété Intellectuelle et brevets

Le terme général de propriété intellectuelle fait référence à tous les aspects juridiques relatifs à la propriété sur les biens immatériels créés par l'intellect.

Le copyright sur un programme est une notion assez claire. Même si l'on a évoqué plus haut les imprécisions possibles dans la notion d'œuvre dérivée, une chose est sûre : si un programmeur se met à son clavier et écrit du code que son esprit conçoit, il n'est pas en train de violer un quelconque copyright. Lui-même, ou son employeur, est titulaire des droits d'auteur sur son code.

En d'autres mots : on sait quand on enfreint un droit d'auteur. Ce n'est pas le cas pour les brevets, et aucun développeur ne pourrait travailler s'il devait se demander à chaque ligne est-ce qu'il est en train de violer un brevet parmi les millions déposés.

Il faut bien comprendre la distinction entre le copyright, le droit d'auteur et le brevet. Le copyright – qui est à le fondement des licences, qu'elles soient libres ou non – porte sur un programme, tandis que le brevet logiciel peut porter sur un procédé, dans le sens le plus vague qui soit. Le double clic sur un bouton de souris, par exemple (brevet 6,727,830) , ou bien encore le fait d'établir une liste de tâches pour les programmeurs (brevet 6,748,582) et bien d'autres aberrations. La plupart de ces brevets sont sans valeur, mais on ne le saurait qu'à l'issue d'un long et coûteux procès. Leur valeur est dans la perspective du procès, et non dans l'issue du procès.

En 1991, Bill Gates déclarait « *Si, au moment où la plupart des idées d'aujourd'hui ont été inventées, les gens avaient compris comment les brevets seraient accordés avaient déposé des*

brevets, notre industrie serait totalement paralysée aujourd'hui ». Et effectivement, si Dan Bricklin, l'inventeur du tableur avec Visicalc avait déposé un brevet en 1979, Microsoft n'aurait pu sortir Excel avant 1999.

Aujourd'hui, les grands éditeurs déposent des brevets aux Etats-Unis, à tour de bras, par milliers. Ils ne s'attaquent pas entre eux, mais ils utilisent la menace d'actions invoquant les brevets comme arme atomique à l'égard des plus petits éditeurs, et du monde open source. De manière totalement organisée, presque avouée, ils utilisent les brevets pour verrouiller leur oligopole, en sclérosant l'innovation.

On considère parfois que les logiciels open source ont plus à craindre des brevets logiciels, simplement parce que leur code est ouvert. S'ils font usage d'algorithmes couverts par brevet, alors il est facile de le découvrir. A l'inverse, si Microsoft ou Oracle utilise du code qui enfreint un brevet, ou même un copyright, il serait extrêmement compliqué de le démontrer.

Fort heureusement, les brevets logiciels n'ont pas cours en Europe, mais le danger est grand et ses partisans ne désarment pas.

[6] SUPPORT

[6.1] Open source et support

Le support des programmes est une question clé. Dans l'informatique en général, et plus encore dans l'open source.

Qu'entend-on par support ? La capacité à apporter de l'aide dans l'utilisation du programme et à corriger le programme le cas échéant.

Le support peut s'adresser aux utilisateurs finaux, comme aux exploitants du programme, ou encore aux programmeurs travaillant sur le programme.

Le déploiement de programmes pour des tâches critiques, en particulier dans des entreprises, requiert absolument un support, car le risque d'une situation de blocage est trop important, cela que ce blocage soit dû à une anomalie ou à un mauvais usage, mauvaise configuration, incompatibilité, etc.

La question du support est un sujet sensible en matière de logiciel open source. En premier lieu parce qu'il y a une différence de fond, pour les produits d'origine communautaire, entre un support de communautés et un support d'éditeur. En second lieu parce que les éditeurs de produits propriétaires voudraient faire croire que le support est un point faible des logiciels open source.

Il est vrai que la licence open source porte en général en gros caractères la mention : *ce logiciel est fourni en l'état, sans garanties, etc…* Et de fait, il serait extraordinaire de réclamer quoi que ce soit à l'auteur qui vous a permis d'utiliser son œuvre. Mais on oublie parfois que l'absence de garantie est le plus souvent invoquée également par les licences propriétaires.

[6.2] Support communautaire et support d'éditeurs

En matière de support open source, il faut bien séparer deux mondes : les produits communautaires d'une part, les produits d'éditeurs commerciaux d'autre part.

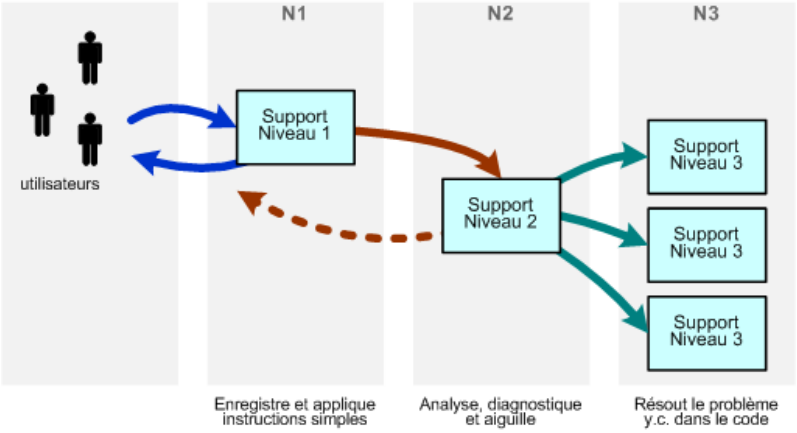
Les produits communautaires (Linux, Apache, PHP, ...) bénéficient avant tout d'un support communautaire. C'est à dire basé sur le volontariat de développeurs impliqués, qui répondent aux questions des utilisateurs sur les mailing-lists et forums. Et basé également sur le suivi et la prise en charge des anomalies sur les plateformes de développement communautaires.

Lorsque la communauté est active, comme c'est le cas autour des grands produits, ce support communautaire peut être d'une très grande efficacité, d'une très grande réactivité, très supérieur à un support commercial. Mais certains utilisateurs resteront chagrinés qu'il soit *sans garantie*, que l'on ne puisse attaquer personne si un problème n'était pas résolu. En réalité, la plupart des supports d'éditeurs commerciaux sont également sans garantie de résultat.

Outre l'aspect communautaire, se pose aussi le problème de la diversité des produits. Un système d'information peut inclure couramment plus de 10 produits différents dans la 'pile' logicielle : Linux, Apache, Tomcat, MySql, Hibernate, ... Lorsqu'un problème survient, à qui s'adressera-t-on ? Les clients professionnels demandent un interlocuteur unique, pour prendre en charge les premiers niveaux de support.

Très tôt dans le développement de l'open source, des acteurs commerciaux ont répondu à cette demande de support. C'est le positionnement des 'distributeurs' tels que Redhat ou Mandriva, mais aussi des premières SSLL en France, telles que Alcôve, Linagora ou OpenWide.

Du côté des éditeurs open source (MySql, eZ Publish, OpenERP et bien d'autres), la question est différente : l'éditeur est une société commerciale et son business model est essentiellement basé sur son offre de support. Ici donc, le dispositif de support est très proche de celui des produits propriétaires. Pas identique toutefois car *en parallèle, en complément* au support payant de l'éditeur, il existe souvent un support communautaire, plus ou moins vivace selon les produits.



Mais le plus souvent, les corrections touchant au code ne sont assurées que par l'éditeur.

Pour les nouveaux éditeurs de l'open source commercial, le support produit est le fondement du business model, il est leur raison de vivre, leur unique source de revenus. On peut donc s'attendre à un support de grande qualité, incluant une variété d'options en termes de réactivité.

[6.3] 3 niveaux de support

Rappelons tout d'abord la définition usuelle des niveaux de support :

- Niveau 1 : un opérateur non-expert prend note de la demande, la saisit dans l'outil de suivi, et consulte des

instructions simples pour tenter un dépannage.

- Niveau 2 : un intervenant expert sur une variété de domaines analyse la demande, fait un premier diagnostic du problème. Il résout le problème ou trouve un contournement dans la mesure de ses compétences, ou sinon détermine l'aiguillage approprié vers un spécialiste.
- Niveau 3 : un intervenant expert spécialisé apporte la correction définitive.

Une correction portant sur le code source d'un programme ne peut se faire qu'au niveau 3.

Bien entendu, une règle générale en matière de support est qu'il faut limiter les intervenants aux premiers niveaux, si possible n'en avoir qu'un seul jusqu'au niveau 2, qui est en charge de l'aiguillage.

C'est ce qui est représenté sur la figure suivante :

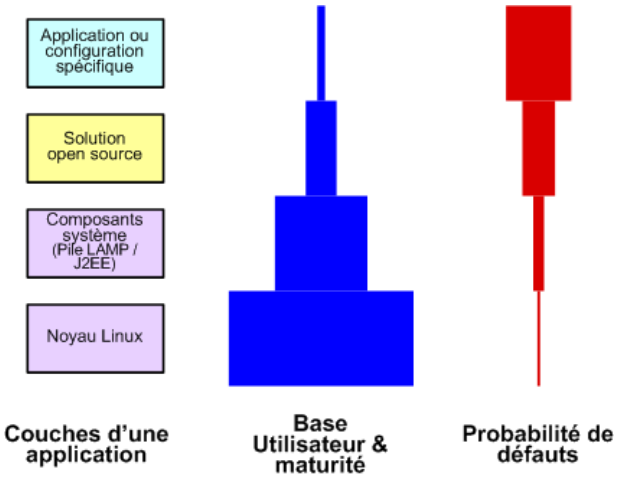
[6.4] Couches logicielles

Dans la suite, nous distinguons 4 couches d'une plateforme open source typique :

- Le système d'exploitation GNU/Linux. Le niveau 3 ne peut être assuré que par des communautés (Debian), ou des distributeurs spécialisés (Redhat, Mandriva, Canonical).
- Les composants système divers, généralement inclus dans la distribution, typiquement Apache ou Tomcat. Ceux-là également sont généralement supportés au niveau 3 par les communautés, par exemple celle de l'Apache Software Foundation.
- Les solutions de haut-niveau d'éditeurs open source, typiquement eZ Publish, OpenERP ou bien Nuxeo. Elles ne peuvent être supportées au niveau 3 que par leur éditeur.
- Enfin, les modules applicatifs spécifiques, ou bien configurations complexes de ces applications. Ce peut être par exemple une application métier entièrement spécifique, construite sur un framework open source, ou bien des jeux de gabarits ou extensions d'un outil de gestion de contenus. Elles sont réalisées le plus souvent par un intégrateur de solutions open source, mais éventuellement par les équipes du client final. Et bien sûr, c'est celui qui les a réalisées qui est le mieux placé pour en assurer le niveau 3.

Il est clair que la stabilité va croissante entre ces 4 couches : les bugs dans Apache sont rares, mais ceux de Linux sont plus rares encore. Les solutions de haut niveau sont moins robustes que Apache, mais la probabilité de bugs est la plus forte dans les développements spécifiques, tout simplement parce qu'ils n'ont qu'un nombre limité d'utilisateurs et une moindre ancienneté.

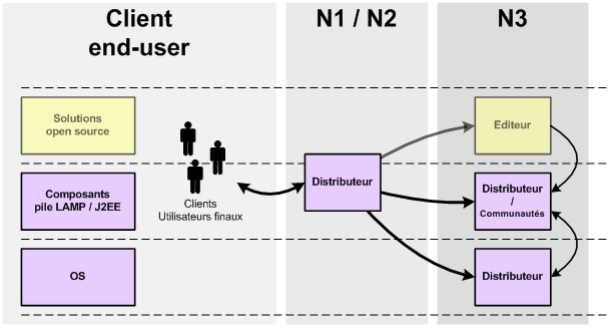
C'est ce qui est représenté sur la figure suivante :



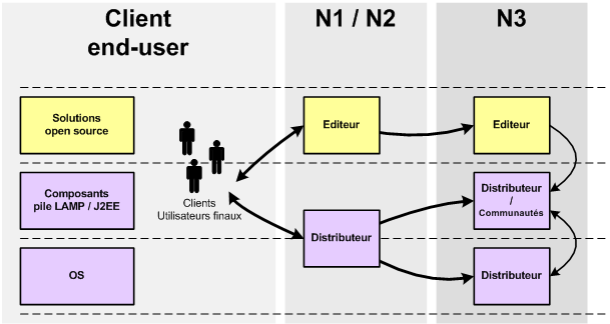
Ainsi, il est clair que le *besoin* de support est plus fort pour les couches supérieures. S'il y a interlocuteur unique, alors ce sera naturellement celui qui peut intervenir sur ces couches.

Support en l'absence d'applicatif spécifique

La figure suivante représente le cas où les produits open source sont utilisés en l'état, avec aucune ou très peu de configuration. C'est typiquement le cas d'un déploiement de la suite OpenOffice.



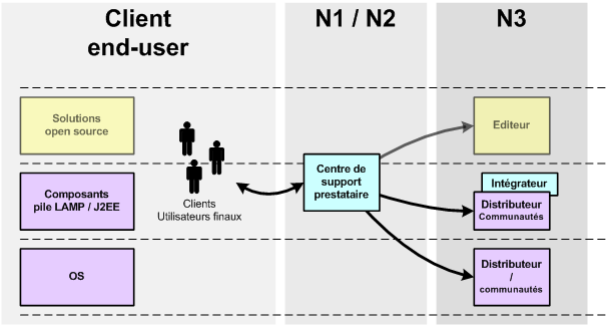
Si la configuration inclut un produit d'éditeur en plus des produits de communautés, et toujours en l'absence de configuration spécifique élaborée, alors le client peut s'adresser directement à l'éditeur pour les niveaux 1, 2 et 3 sur son produit, et au distributeur sur les couches inférieures. Il est rare que les éditeurs souhaitent assurer du niveau 1 et 2 au delà de leur produit, sur l'ensemble de la configuration.



Centre de support intégrateur

Enfin, certains prestataires proposent d'assurer un support global multi-produits en niveaux 1 et 2. C'est le cœur de métier de certaines SSLL, mais quelques SSII généralistes s'y sont mises également.

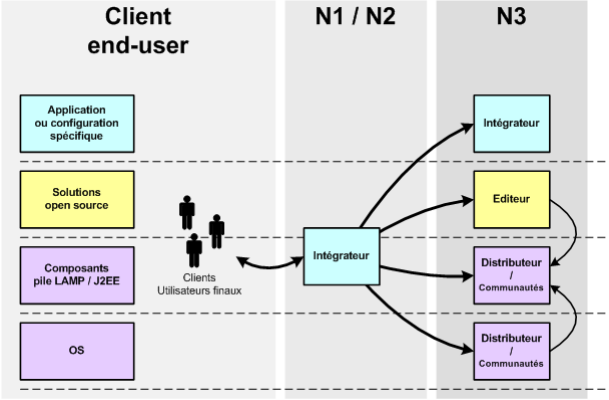
Il est rare qu'elles aient l'expertise requise pour assurer le niveau 3 sur toute la palette des composants, mais elles peuvent construire une expertise suffisante sur quelques-uns d'entre eux.



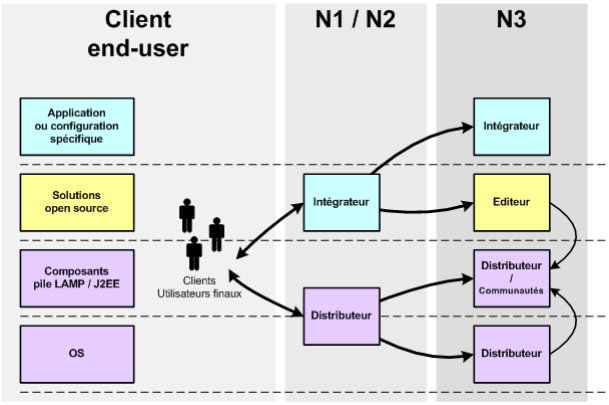
Cas d'une application ou configuration spécifique

La figure suivante représente le cas où le client utilise une application spécifique, ou bien une configuration complexe d'une solution open source d'éditeur, qui a été élaborée pour lui par un intégrateur de solutions open source.

Dans ce cas de figure, l'intégrateur est le plus à même d'assurer le support de niveau 1 et 2 sur l'ensemble de la pile, en se tournant vers les expertises appropriées pour le niveau 3.



Il peut arriver toutefois que le client ait plusieurs configurations différentes, reposant sur les mêmes couches basses, et aura confié à un distributeur le support global sur ces couches.



Cette seconde figure représente une alternative, dans laquelle le client final s'adresse à un distributeur pour le support du bas de la pile : système d'exploitation et composants système, et à l'intégrateur pour les composants supérieurs.

[7] MODELES ECONOMIQUES

[7.1] Principes

Ce qu'on appelle *modèle éconmique,* ou *« business model »*, ce sont les principes de fonctionnement qui assurent la rentabilité d'une société. On peut étendre l'analyse aux organismes à but non lucratif en s'intéressant du moins à leurs revenus et charges.

Et de fait, tout n'est pas gratuit dans l'open source, et il y a une vraie *économie de l'open source*, qui a ses particularités.

On distingue trois grandes familles de logiciels open source : (1) les produits de fondations (Apache, Eclipse, Linux, …), (2) les produits communautaires et (3) les produits d'éditeurs. La question des modèles économiques se pose essentiellement pour ces derniers, dont l'offre constitue une part croissante du patrimoine open source.

On sait bien que les logiciels open source ne sont pas tous écrits par des bénévoles, le soir après leur travail, pour le plaisir ou pour la gloire. Ils existent, ces développeurs passionnés qui prennent sur leur temps libre pour faire avancer des projets. Ils existent, et on peut les remercier, mais ils n'écrivent au final qu'une toute petite partie des programmes open source que nous utilisons.

La question est souvent posée par les néophytes incrédules : *Mais si c'est gratuit, alors comment ça peut marcher, il faut bien que quelqu'un paye à un moment ?*

Nous étudions ici les 4 typologies d'acteurs de l'open source :

Fondations

A l'image de la fondation Apache, ou Eclipse, ce sont des organismes à but non lucratif, qui stimulent et pilotent le développement de grands produits open source.

Distributeurs

A la manière de Redhat, Canonical (Ubuntu) ou Mandriva, ils sélectionnent des outils et composants autour d'un noyau Linux, en assurent le packaging, la

distribution et le support. Ils ont souvent aussi une activité d'éditeurs.

Éditeurs

Ils créent un produit logiciel, qu'ils diffusent sous licence open source, en tout ou partie. Ils assurent la promotion de leur produit, et proposent des offres de support.

Prestataires

Les prestataires de l'open source vendent des services, que ce soit dans un mode de régie ou de forfait. La prestation peut porter sur du conseil, de l'intégration, du support, de la formation, de l'hébergement, etc.

[7.2] Les fondations

Les fondations et autres organismes à but non lucratif tiennent une place très importante dans l'écosystème de l'open source.

Les plus grands produits open source, et ceux qui ont la plus large diffusion, sont issus de ces fondations, ou bien repris en charge par celles-ci.

La Free Software Foundation, déjà évoquée plus haut pour sa définition du logiciel libre et des licences GNU GPL et ses missions de défense et de promotion du logiciel libre, continue de jouer un rôle clé dans le développement des composants du GNU Project qui sont associés au noyau Linux.

Voyons quelques unes des autres grandes fondations.

Apache

Le serveur Http Apache est le produit fondateur de la fondation éponyme. Il remonte aux tous débuts du web, soit 1995, où quelques développeurs réunis au sein de l'Apache Group entreprennent d'améliorer le premier Httpd du NCSA, comme alternative aux outils de Sun et Netscape. A partir de 1996 et jusqu'à aujourd'hui, le serveur Apache est le plus utilisé sur le web.

La *Apache Software Foundation* (ASF) est une association à but non lucratif de droit américain, et l'un des temples de l'open source. Dans le paysage de l'open source, c'est la seule entité qui ait à la fois les moyens de pousser des projets nombreux et d'envergure, et qui ne soit pas à la recherche de profits.

Du fait de cette vocation non commerciale, l'ASF est motivée à donner naissance à des projets de qualité, qui puissent être utilisés librement par le plus grand nombre. C'est aussi cette caractéristique qui amène des entreprises ou développeurs à donner des programmes à l'ASF.

Les programmes des projets Apache appartiennent à l'ASF, qui les diffuse sous licence APL, une licence non-copyleft.

La fondation Apache est financée par quelques sponsors, et tire de petits revenus de l'organisation de séminaires, vente de goodies et dons en ligne. Mais en fait, la fondation a surtout un tout petit budget.

Sur son bilan 2005-2006, qui reste en 2010 le dernier publié, elle déclarait des recettes de 150 K$, dont 95 K$ de dons et 50 K$ de revenus de ses services, répartis entre recettes des conférences Apache (environ 30 K$) et des Codes Awards (20 K$).

Les dépenses, sur la même période, ne sont que de 33 K$, dont 28 consacrés à la mission principale de la fondation de *diffuser ses logiciels open source au public gratuitement,* c'est à dire principalement des coûts d'hébergement et d'exploitation.

On constate donc que les flux financiers sont minuscules, comparés à la puissance

effective de la fondation dans sa mission de promotion et de développement de grandes applications open source.

L'avancement des projets est surtout basé sur le volontariat, mais également sur les dons en nature que peuvent faire des entreprises en autorisant certains de leurs développeurs à travailler sur des projets Apache sur leur temps de travail, pour une période convenue. Des accords spécifiques permettent d'assurer que le fruit de ce travail est propriété de l'ASF.

Il y a une bonne cinquantaine de projets Apache, dont la notoriété, la diffusion et la qualité sont variées. Dans l'ensemble, une caractéristique commune est d'avoir une architecture logicielle solide, basée sur des standards.

Citons quelques-uns de ces produits : Apache Httpd, Perl, Lucene, Tomcat, Ant, Cocoon, Lenya, OfBiz, Struts, et bien d'autres…

Eclipse

Eclipse est une initiative qui réunit de grandes sociétés informatiques, à l'initiative d'IBM, pour développer initialement une plateforme de développement intégrée (IDE, *Integrated Development Environment*), du même nom.

Le mouvement change de statuts en 2004 pour devenir fondation Eclipse, association à but non lucratif *(non profit organization*).

La mission de l'organisation est de créer et promouvoir un ensemble d'outils de conception, développement et gestion de programmes, ainsi que des composants et frameworks.

L'intérêt stratégique, pour IBM, est de contrer la plateforme Microsoft dans les entreprises. En effet, la qualité des outils de développement tient une part importante dans l'adoption d'une plateforme par les développeurs, et les outils de développement Java étaient souvent jugés moins bien intégrés et moins ergonomiques que ceux de Microsoft.

La fondation est financée par ses membres, de grandes sociétés proches de l'informatiques (IBM, Intel, BEA, Motorola, Nokia, Oracle, SAP, Zend, …). Elle dispose de salariés pour des missions administratives, mais les développeurs sont des programmeurs indépendants, ou bien travaillant pour des entreprises qui leur accordent du temps pour participer à ces projets.

La fondation fournit 4 services à la communauté Eclipse :

- une infrastructure matérielle qui héberge les travaux,
- un cadre juridique pour les questions de propriété intellectuelle,
- des processus pour le développement communautaire,
- et la promotion et facilitation des projets au sein de l'écosystème.

[7.3] Les Distributeurs

Les distributeurs sont des sociétés comme *Redhat, Ubuntu, Mandriva, Suse* et quelques autres, dont l'activité est de :

- *Sélectionner* des produits et versions, entourant le noyau Linux
- *Valider* la maturité et la robustesse de ces produits
- *Distribuer* ces produits et leurs mises à jour, c'est à dire assurer leur acheminement jusqu'aux utilisateurs-clients
- Assurer le *support* de ces produits : hot-line, traitement des demandes, conseil, formation

Initialement, dans les années 90, le moyen de distribution privilégié était la

disquette puis le CD-ROM, et la principale activité des distributeurs était le gravage et la distribution des CDs. Aujourd'hui, la diffusion est majoritairement online, et le cœur de métier des distributeurs s'est déplacé pour se centrer sur le support.

Les distributeurs distribuent essentiellement des produits dont ils ne sont pas détenteurs des droits. Ils n'ont donc pas le choix de proposer telle ou telle licence, ou bien une licence GPL et une licence commerciale comme le font certains éditeurs : c'est le détenteur des droits qui décide de la licence. Les distributeurs open source diffusent une majorité de produits sous GPL, et quelques produits sous BSD ou d'autres licences.

Certains sont aussi éditeurs de quelques produits de leur distribution.

Redhat

Fondé en 1994, Redhat domine de très loin ce marché, avec près 2800 employés dans le monde en 2009. Pour beaucoup d'entreprises, en particulier aux États-Unis, Redhat a donné sa crédibilité à l'open source.

Redhat est aussi un des plus importants contributeurs du noyau Linux, et est également éditeur de produits open source, au premier rang desquels figure le serveur d'application JBoss, acquis en 2006, ou des outils tels que Hibernate.

Le support est disponible par abonnement (*subscription*), payé annuellement. Le prix dépend du périmètre de produits concernés, et du niveau de service. Au catalogue Redhat, ce support peut aller de $350 à $2500, par an et par serveur. Le modèle est donc par construction très récurrent. A noter que le contrat de support inclut une clause de « Intellectual Property protection », une assurance juridique qui protège le client d'éventuelles actions d'éventuels détenteurs de brevets. Une clause très prisée aux États-Unis.

Les souscriptions représentent 82% des revenus de Redhat, le reste provenant des prestations de formations et conseil. Du coté des coûts on peut identifier, sans connaître la part de chacun :

- Les coûts habituels d'une société commerciale : services généraux, RH, services commerciaux, marketing. A noter que les coûts de commerce et marketing représentent 36% du chiffre d'affaire, et les autres coûts administratifs, 20%.
- Les coûts associés aux prestations de support : hot-line, experts, consultants
- Les coûts des développeurs contribuant aux produits open source distribués, ou *recherche et développement :* 18% environ.

Au total, le résultat net de Redhat pour l'année 2008-2009 est d'environ 79 M$, sur un chiffre d'affaire de 653 M$.

Mandriva

Si Redhat affiche une santé éclatante, les distributeurs Linux français ont connu au contraire des années difficiles. Anciennement Mandrakesoft, la société diffuse et supporte depuis 1998, la distribution Mandriva Linux, qui a accédé au 'top 10' des distributions au plan mondial.

Introduite en bourse en 2001, la société a un parcours mouvementé, traversant un redressement judiciaire en 2003, parvenant à quelques résultats positifs en 2004 avant de replonger dans le rouge. Sur l'exercice 2006-2007, le chiffre d'affaire est de 4,2 M€ et la perte d'exploitation de 2,3 M€. Les derniers chiffres disponibles à mars 2010 sont ceux du troisième trimestre 2008, avec un CA de 0.83 ME, et une perte d'exploitation de 0.64 ME. Il semble que

Mandriva souffre particulièrement de la montée en puissance de la distribution Ubuntu.

Mandriva est éditeur de produits en propre : la solution de gestion de parc informatique Pulse 2.0, et le serveur Ldap Mandriva Directory Server. En 2007, Mandriva a également fait l'acquisition de Linbox.

Debian

Même si elle ne vise pas un semblable business model, il faut ici une mention spéciale pour Debian, une distribution Linux non-commerciale, la plus ancienne, la plus communautaire et finalement la plus proche des valeurs fondatrices de l'open source. C'est aussi la seconde distribution Linux la plus utilisée.

Projet fondé par Ian Murdock (aujourd'hui chez SUN), en 1993, il se caractérise par les Debian Free Software Guidelines, énoncées en 96 (qui inspireront l'Open Source Definition), et son système de gestion des packages.

[7.4] Les éditeurs open source

Éditeur open source

L'éditeur, c'est celui qui détient les droits du produit, en assure le développement, la promotion, la diffusion et le support.

Dans un premier temps, les seuls acteurs commerciaux de l'open source étaient des distributeurs plus que des éditeurs, l'acteur emblématique étant Redhat.

C'est MySql qui a ouvert la voie de la logique de l'éditeur open source, et depuis quelques années, ce modèle a donné naissance à de nombreux nouveaux acteurs, particulièrement dynamiques.

Les éditeurs open source sont des sociétés commerciales ordinaires, c'est à dire à but lucratif. Comme un éditeur ordinaire, elles investissent massivement dans le développement de leur produit, et parfois également dans sa promotion, son marketing. La seule différence est que le produit est diffusé sous licence open source, ou parfois sous double licence.

Pourquoi choisissent-ils ce modèle ? Au delà de l'adhésion aux valeurs de l'open source, ils font sans doute l'analyse que l'open source est devenu le seul moyen de percer dans un marché prisonnier de quelques oligopoles. Un peu à la manière des compagnies aériennes low-cost, ces nouveaux acteurs amènent un business model légèrement différent, de nature à casser les positions acquises.

Si la finalité économique est très semblable, ces nouveaux acteurs ont néanmoins des caractéristiques spécifiques par rapport aux éditeurs classiques. En premier lieu, ce sont de petites structures, de très petites structures en comparaison des éditeurs en place. MySql comptait 360 employés au moment où SUN en a fait l'acquisition. Et leurs forces sont majoritairement tournées vers le développement et le support du produit. Elles font peu de marketing, peu de commerce. Comme les compagnies low-cost, elles ont structurellement des coûts très inférieurs, ce qui leur permet de vivre avec de faibles revenus.

Malgré tout, développer un programme de qualité coûte cher, même s'il est open source. Un peu moins cher peut-être que son équivalent propriétaire, parce que (a) il peut s'appuyer sur d'autres briques open source, pour autant que leur licence le permette, (b) il peut bénéficier de contributions communautaires, et (c) il a probablement plus de développeurs passionnés. Mais tout cela ne fait pas du logiciel à moitié prix.

Puisque développer un programme coûte cher, il faut bien que l'éditeur trouve un

retour sur son investissement. Et même une certaine prime de risque, puisque son risque est grand de ne pas rencontrer le succès espéré.

Ce qui pose la question du modèle économique des éditeurs open source, souvent indissociable du choix de licences. Voyons quels sont ces modèles, et les grandes tendances du moment.

Vraiment open source ?

Certains éditeurs pourraient être tentés d'être « plus ou moins open source », diffuser les sources sans les droits d'utilisation, ou avec différentes conditions restrictives, bref de se proclamer open source sans l'être vraiment.

Heureusement, ce n'est pas possible, et c'est l'apport fondamental d'institutions comme la FSF et l'OSI, que de définir sans ambigüité ce qui est, et ce qui n'est pas, une licence open source.

Les éditeurs open source ne peuvent donc pas tricher, ils doivent diffuser leurs produits sous une licence agréée. Ce qui signifie que les utilisateurs peuvent librement et gratuitement redistribuer lesdits produits, c'est la définition même.

La plupart des éditeurs choisissent la licence GPL, qui présente pour eux deux grands avantages : (1) elle est connue et donc parfaitement lisible, elle est perçue par le marché comme un gage de transparence, (2) elle interdit à d'autres d'intégrer le produit dans un développement propriétaire, et donc de se faire de l'argent sur le dos de l'auteur ou détenteur du droit d'auteur.

Business model de l'éditeur open source

Les éditeurs open source ont trois types de revenus :

- Ventes de licences
- Vente de support
- Vente de prestations

Auxquels s'ajoutent dans certains cas des revenus issus de leurs intégrateurs partenaires : partenariat payant pour certains, ou commission sur l'apport d'affaires, lorsque des prospects sont dirigés vers des partenaires.

Et bien sûr, au chapitre des dépenses, on trouve :

- Le développement produit
- Le support
- Le commerce et le marketing

Les éditeurs bénéficient aussi de l'open source au niveau des *coûts* : en pouvant s'appuyer sur l'extraordinaire patrimoine de code déjà disponible sous licence open source, ils font d'importantes économies de développement.

Vente de licence

« Vendre des licences » et « open source » semble une contradiction. Effectivement, même s'il n'est pas interdit de faire payer la distribution, il n'est pas possible de faire payer un droit d'utilisation sur un logiciel open source (cf. « Bière gratuite ?! », page 7).

Mais il y a plusieurs cas de figure de *double licences* possibles.

1) Sortir de la GPL

Le premier est une licence non open source, propriétaire donc, qui permet au client de ne pas être tenu par les obligations de la licence GPL. En particulier si le client veut distribuer une œuvre dérivée utilisant le programme, et ne souhaite pas diffuser ses sources, il lui faudra acquérir une licence commerciale. Des éditeurs comme eZ Systems ou MySql proposent cette option.

Pour des applicatifs de haut niveau, par exemple le CMS eZ Publish, ce n'est en général pas une source de revenus importante car la finalité est de *faire un site* et rarement de *faire un produit*. Mais dans le cas de MySql, la vente de licences représente plus de la moitié du CA.

2) Modules complémentaires payants

Le second cas de figure est celui où l'éditeur propose des modules complémentaires à l'application principale, ces modules étant exclusivement sous licence commerciale. Selon les cas, la partie open source pourra être plus ou moins complète. Mais si elle est trop légère, et donne l'impression d'être un simple appât pour ferrer le pigeon, elle sera rejetée. Si l'application open source est de qualité, et que les modules payants sont optionnels, le modèle peut tenir la route. On peut citer Talend et Pentaho, comme éditeurs ayant choisi ce modèle.

3) Dépendance entre le support et la licence

Le troisième cas, le plus courant, est celui où l'éditeur crée une dépendance entre son offre de support et la licence commerciale. Dans ce cas l'éditeur ne propose aucun support, même payant, sur la version open source. Pour avoir du support, il est nécessaire de choisir la licence commerciale. C'est le cas de l'éditeur Alfresco par exemple.

Quels revenus ?

Pour un éditeur open source, les sources de revenus possibles sont les suivantes :

- le support et la maintenance, toujours dans le cadre de contrats récurrents
- les prestations associées: audit, conseil, formations, voire intégration
- les droits d'utilisation associés à la licence, à une licence non libre, donc.
- les reversements de prestataires habilités

Dans une logique d'expansion globale et rapide, les éditeurs ne peuvent pas miser beaucoup sur les prestations. Faire l'intégration de son propre produit est tentant au début, mais cela ne permet pas de construire un réseau de partenaires intégrateurs.

Mais surtout, un modèle économique basé sur la prestation est perçu par les éditeurs, et ceux qui les financent, comme un modèle « non scalable », c'est-à-dire qui ne permettra pas d'augmenter les revenus sans augmenter les effectifs, et donc les coûts. En somme un modèle qui ne permettra pas les taux de marge des éditeurs propriétaires.

Modèle 1 : uniquement le support

Certains éditeurs ont un business model fondé presque uniquement sur l'offre de support. Dans cette catégorie on trouve eZ Systems, Nuxeo, Tiny (OpenERP), Spago.

Il n'y a qu'une version du produit, et une licence unique - open source donc - un seul référentiel de sources, et des correctifs disponibles à tous. Le produit peut être librement téléchargé et déployé, et les « clients » *ont le droit d'utiliser le produit sans payer le support.*

Le support étant clairement facultatif, il faut savoir démontrer aux clients les bénéfices qu'ils peuvent en attendre. Auprès des DSI de grands comptes, ce n'est généralement pas trop difficile. En cas de plantage d'une application critique, personne n'imagine d'aller expliquer au président qu'il était possible de souscrire un support mais qu'on avait préféré s'en passer ! Et par ailleurs, les grandes DSI perçoivent bien que l'économie est déjà très grande par rapport aux logiciels qu'elles utilisaient précédemment. Mais auprès de plus petites entreprises, la démonstration est parfois plus difficile.

L'une des difficultés du pur support annuel et que s'il n'a pas servi pendant une année, certains pourront hésiter à le renouveler. Bien sûr, on leur dira : il y a une logique d'assurance dans une prestation de support, ce n'est pas parce que vous n'avez pas été cambriolé cette année que vous allez résilier votre assurance ! Mais de leur côté ils répondront : si cette appli a tourné sans problème pendant un an, elle pourra bien continuer de tourner un an de plus !

C'est le problème et le paradoxe du support : plus le produit est de qualité, moins le support est facile à vendre.

Mais en fait, une majorité de produits sont en permanente évolution, sous la pression concurrentielle. Et c'est la force des produits open source que d'évoluer souvent plus vite que les autres. Le produit n'est jamais totalement stabilisé, toujours en mouvement, et le support toujours précieux pour une utilisation professionnelle.

Modèle 2 : stabilité et support

D'autres éditeurs choisissent un modèle qui fait payer la stabilité. Ils distribuent leur produit sous deux licences, l'une libre, l'autre non-libre. Mais ce n'est pas tout à fait le même produit.

Dans cette catégorie, on peut citer Alfresco, Jahia, Pentaho, eXo, Liferay.

Avec la licence non-libre, on a une version « *enterprise-ready* », « *production grade* », « *fully tested* », etc. tandis que la version libre, intitulée « *community* », ou encore « *labs* », est présentée comme pas stable, pas testée, vraiment le truc pour les *geeks*, limite dangereux, à ne surtout pas déployer en production.

Le vocabulaire est celui-ci, mais dans les faits les différences ne sont pas toujours bien identifiées. L'éditeur ne peut pas faire vivre deux référentiels de code différents sur le long terme, il faut donc au minimum resynchroniser *community* et *enterprise* de temps en temps. En général, la version *community* est la version en cours de développement, sur laquelle les développeurs interviennent en continu, tandis que la version *enterprise* est celle qui a été gelée puis validée en profondeur, a reçu différents correctifs et en reçoit en continu, distribués dans le cadre du support.

Ainsi, paradoxalement, la version *community* est en avance de phase, elle dispose des dernières fonctionnalités, mais elle est moins stable, soit du fait d'une avance de phase dans les développements, soit du fait d'un retard dans les correctifs. Dans un bon gestionnaire de sources, on pourrait espérer pouvoir extraire la dernière version stable, voir même y réappliquer les patchs, mais ici ce n'est pas le but.

Le support vient en *bundle* avec la licence entreprise, dont il est indissociable, tandis qu'au contraire on ne peut espérer aucun support, même payant, sur la version libre.

Pour l'éditeur, les avantages sont nombreux.

La version *community* aide à diffuser le produit, à le faire connaître largement au

niveau mondial, tant par les clients potentiels que par les intégrateurs potentiels.

Et par ailleurs, une fois parti sur la version *enterprise* et son support annualisé, il est délicat pour le client de revenir à une version *community*. Le *downgrading* serait à gérer comme une véritable migration. Il y a donc une forme de fidélisation que ne permet pas le seul support payant.

A l'inverse, l'inconvénient est que si la version community est trop instable, elle ne contribue pas à la bonne image du produit et à sa promotion. Et si au contraire elle est d'une bonne qualité, elle risque d'être utilisée malgré les mises en garde alarmantes.

Laisser les utilisateurs expérimenter une version instable n'est pas toujours suffisant. Le processus complet d'adoption est bien souvent : recueil d'information, téléchargement, expérimentation, projet réel non critique, et enfin projet stratégique. Pour beaucoup d'entreprise, la version *community* doit permettre d'aller jusqu'au projet réel non critique, qui permettra de faire un choix stratégique. C'est en cela que le degré de stabilité doit être étalonné avec soin.

Modèle 3 : fonctionnalités avancées et support

Enfin, le troisième modèle est celui d'une licence double définie selon le niveau de fonctionnalités : une version GPL avec des fonctionnalités réduites, une version *enterprise* non libre avec des fonctionnalités avancées.

Dans ce dernier modèle, on trouve par exemple Talend, Jasper, ou encore MySql.

Dans ce cas, il n'y a pas de différence de qualité, de stabilité. Les programmes correspondant au premier niveau de fonctionnalités sont les mêmes. Comme pour n'importe quel programme, le gestionnaire de sources identifie les dernières versions stables, et les derniers *builds*, mais n'importe qui peut aussi télécharger la dernière version stable. En revanche, les sources de la version *enterprise* ne sont parfois pas diffusées du tout.

Ne pas définir la différence en termes de qualité offre d'une certaine manière une meilleure image de l'éditeur, de sa capacité à produire un logiciel solide. Elle évite l'association d'idées « libre = instable », qui est assez désagréable et par ailleurs infondée.

Mais ici aussi, toute la difficulté réside dans le tracé de la frontière : il faut en mettre suffisamment dans la version libre pour qu'elle soit largement diffusée et donne l'impression d'un produit riche, et suffisamment limitée ne pas permettre des utilisations à forte valeur ajoutée.

Le support

Comme on l'a vu, le support est la principale source de revenus pour la majorité des éditeurs open source. Les offres de support sont le plus souvent sur la base d'une souscription annuelle, par instance du produit, par serveur ou par processeur.

Le support inclut en général :

- Un accès privilégié aux correctifs, et à des ressources spécifiques.
- La prise en charge des problèmes, que ce soit anomalies ou problèmes d'utilisation ou de mise en œuvre.
- Éventuellement des prestations d'audit, de certification, ou de prise de contrôle à distance, surveillance proactive et corrections.

Correctifs

Les éditeurs peuvent être tentés de maintenir un écart entre le niveau de

correctif des clients sous support et le référentiel de sources public. Mais ils n'en abusent pas, car c'est aussi la réputation de leur produit qui se joue sur la version open source. En revanche, les clients sous support reçoivent les correctifs en *'push'*, sans les avoir demandés.

Le contrat de support peut inclure également l'accès à certaines ressources privilégiées : forums, base de connaissances, mailing lists, voire même documentation. Mais là aussi, ne pas diffuser librement de documentation est en général plutôt pénalisant. Il faut se souvenir que les libres utilisations sont aussi des produits d'appel pour l'utilisation en conditions critiques, qui demandera du support.

Résolution de problèmes

Beaucoup de contrats de support distinguent différents niveaux, en termes de

- Nombre de problèmes sur l'année
- Temps de réaction sur problème
- Horaires d'ouverture de la hot-line
- Garantie de correction
- Prise de contrôle

La relation avec le support est en général de type web ou mail pour les contrats basiques, et téléphonique pour les contrats haut de gamme.

Trois éditeurs open source

MySql

MySql A.B. est une entreprise suédoise, éditeur de la base de données du même nom.

En 1994, il n'existait pas de base de données relationnelle légère, et encore moins open source. Depuis 1990, il existait déjà Postgres, créé par Michael Stonebraker, déjà fondateur de Ingres. Mais à cette époque, Postgres n'utilisait pas SQL mais QUEL comme langage de requête.

Dans l'année 1994, Hughes, un étudiant australien, réalise pour Postgres un traducteur de requête de SQL vers QUEL, puis il finit par réécrire la couche de stockage, en simplifiant au maximum les fonctionnalités. Son projet s'appelle mSql, et il acquiert rapidement une belle notoriété.

En 1995, Michael Widenius, de la société suédoise TcX ajoute une interface SQL compatible avec mSql sur le moteur de base de données maison Unireg. Dès le début, la société adopte un modèle d'éditeur open source, ce qui crée rapidement une vague d'adhésion et de contributions.

Ce n'est toutefois qu'en 2000 que MySql passera sous licence GPL. Jusqu'à cette date, une licence spécifique excluait les plateformes Windows et interdisait à qui que ce soit de proposer un support payant. En fait une telle licence ne passe pas les critères de l'*open source definition*.

MySql AB comptait 360 employés, pour un chiffre d'affaire de 40M$ en 2006, derniers chiffres connus lors du rachat par SUN. Il faut bien mesurer à quel point ce sont des chiffres minuscules au regard des grands éditeurs traditionnels. Oracle comptait le même année 68 000 employés et un chiffre d'affaires de 17 Md$. C'est à dire 400 fois plus.

En 2004, avec la version 4, MySql a modifié ses conditions de licence, les connecteurs permettant à des programmes d'accéder à la base, sont passés de la licence LGPL à la GPL. Les implications sont très fortes. En effet, le fait qu'un programme appelle une base de données n'implique pas qu'il soit « construit sur » la base de données, au sens de la licence GPL. A priori donc, un programme peut utiliser en tant que

serveur une base MySql sous GPL, sans tomber sous les obligations de la licence GPL. Mais pour réaliser cet appel, les applications utiliseront le plus souvent le connecteur fourni par MySql. Ainsi, en mettant les connecteurs sous licence GPL et non LGPL, MySql vise à propager aux applications utilisant son serveur les implications de la GPL, et de cette manière pousser davantage de clients vers ses licences non open source.

C'est pourquoi ce changement de politique commerciale a été mal vécu dans les communautés open source. Les développeurs de PHP en particulier ont invoqué l'incompatibilité entre la licence utilisée pour PHP. Pour y répondre, MySql a ajouté une clause particulier, la « FLOSS exception[8] », et l'armistice a été conclue avec la communauté PHP.

Enfin, en janvier 2008, SUN a acheté MySql pour environ 1 milliard de dollars, de l'ordre de 20 années de chiffre d'affaire. Le deal a été vu comme une prise de conscience, de la part des acteurs traditionnels, de la montée en puissance des éditeurs open source.

MySql vend son produit *MySQL Enterprise,* qui est essentiellement une offre de *service*. Les services relèvent principalement du support, de l'audit et du conseil.

L'offre de service est *dissociée* de la nature de la licence : elle concerne par défaut les programmes diffusés sous licence GPL, mais le client peut à son choix utiliser la licence commerciale, au même prix.

Sur son site, MySql énonce le principe général que l'utilisation de la base dans le contexte d'une organisation commerciale devrait être sous licence commerciale. L'autorisation de diffuser *au sein* d'une même organisation sans nécessité de rendre publiques les sources, n'est pas évoquée, mais elle découle de la licence GPL.

[8] http://www.mysql.com/about/legal/licensing/foss-exception.html

eZ Systems

eZ Publish est un outil de gestion de contenus (CMS) écrit par Bård Farstad en 1999 et diffusé sous GPL à partir de 2000. Le produit est d'entrée de jeu mieux conçu qu'une majorité des applications PHP, surtout de cette époque. Il adopte rapidement une modélisation objet, et une couche d'abstraction permettant de travailler avec n'importe quelle base de données.

Comme pour MySql, les revenus des premières années viennent surtout de l'intégration du produit dans ses propres projets, sur son marché local, la Norvège. En 2002, le produit a déjà une reconnaissance mondiale, et l'on parle déjà de « killer application du PHP » !

A partir de 2004, eZ Systems s'implante dans différents pays européens, puis aux États-Unis.

En 2006, eZ Systems met en open source les quelques composants associés au CMS, qui ne l'étaient pas. En avril 2007, eZ Systems lève 5 M$ de capitaux auprès d'investisseurs norvégiens, ce qui lui permettra certainement un élan nouveau vers son but de devenir le système de gestion de contenus de référence, y compris pour les plus grandes entreprises. En 2009, une nouvelle augmentation de capital est réalisée.

Le business model de eZ Systems est fondamentalement basé sur le support, avec des offres silver, gold, et platinum, qui se distinguent par le nombre de tickets d'incidents inclus, et le temps de prise en compte, pour des prix allant de $6990 à $13990.

Alfresco

Alfresco est fondé en 2005 par des anciens dirigeants de Documentum et de

Business Objects, qui amènent avec eux des architectes de grands éditeurs, et des capitaux importants. Alfresco est donc le parfait exemple de la nouvelle génération d'éditeurs open source.

La première génération d'éditeurs open source avaient en général commencé à l'université ou dans leur garage, et s'étaient imposés petit à petit en construisant une communauté. Au contraire, Alfresco a directement les moyens de financer une équipe de haut vol, tant en développement qu'en marketing, et se fait reconnaître en à peine deux ans, comme un grand acteur sur son marché, celui de la Gestion Electronique de Documents (GED), typiquement un marché sclérosé entre les mains de quelques grands acteurs vieillissants.

Alfresco diffuse son produit de GED sous deux licences :

- Une version *Community*, sous licence GPL
- Une version *Enterprise*, sous licence commerciale

Alfresco ne propose pas de support, même payant, pour la version *community*, tandis que la version Enterprise au contraire, est indissociable de son support, sur un mode de souscription annuelle, par serveur.

En termes de licences, il est intéressant de noter que Alfresco a commencé par utiliser une licence issue de la MPL (Mozilla), avec une clause spéciale obligeant à présenter un message d'avertissement sur toutes les pages de l'application. Début 2007, Alfresco est passé sous licence GPL pour la version *community*, ici aussi pour une meilleure lisibilité de la politique open source.

Alfresco précise clairement dans sa FAQ qu'un déploiement *au sein de son organisation*, n'est pas considérée comme une *distribution*, ce que tous les éditeurs ne disent pas aussi clairement.

La version community est modifiée au fil de l'eau, tandis que la version enterprise est l'objet de releases trimestrielles validées.

A noter également que le contrat de partenariat que Alfresco signe avec ses intégrateurs leur fait interdiction d'intégrer la version *community*, de sorte qu'il est difficile pour les utilisateurs de cette version d'obtenir un support professionnel.

Loi des grands nombres

Pour prospérer, les éditeurs open source doivent être dans une logique de grands nombres, à la manière de MySql : le fait que 10 millions d'utilisateurs ne payent rien à MySql AB n'est pas problématique si sur ce nombre, il en reste 10 000 qui ont des utilisations suffisamment stratégiques, et souhaiteront disposer d'un support d'éditeur, avec délai d'intervention garanti.

Le pourcentage de clients qui feront appel au support éditeur dépend de la typologie de produit. Pour un produit grand public, par exemple un antivirus open source, il sera bien difficile de faire payer du support à beaucoup, ou de vendre sa version payante. Pour un produit dont la vocation est fortement B2B, par exemple une Gestion Electronique de Documents, la part des clients demandeurs de support sera naturellement plus grande.

Contributions communautaires

Les éditeurs open source comptent en général assez peu sur les apports communautaires, du moins sur le cœur de leur produit. Ils les acceptent car c'est dans la logique de l'open source, mais ne les encouragent guère et l'on peut penser qu'il ne leur déplait pas de garder la maîtrise de leur produit.

A noter que si son morceau de code est accepté, le contributeur devra généralement signer un accord spécifique qui permet à l'éditeur de disposer librement de son code. C'est assez naturel, car si chaque contributeur pouvait spécifier ses propres conditions de licence, le produit final serait un enchevêtrement de licences indémêlables.

Afin de bénéficier d'une dynamique communautaire, tout en conservant la maîtrise du noyau de leur produit, certains éditeurs mettent en place un dispositif d'extensions, qui permet d'apporter des enrichissements au produit, de manière propre et indépendante du noyau, en assurant la compatibilité avec les versions futures.

Noyau, extensions et écosystème

Le modèle qui semble le plus efficace, et le meilleur compromis, est celui qui distingue le noyau du produit, sous la responsabilité de l'éditeur, et les extensions, réalisées par des contributeurs externes.

Les principes de cette séparation sont les suivants :

- Le noyau doit être d'une grande robustesse, il est certifié par l'éditeur ; les contributions externes y sont rares.
- L'interface entre le noyau et les extensions est bien documentée et stable, c'est à dire qu'un changement de version du noyau n'implique pas, du moins le plus souvent, un changement de version des extensions.
- L'éditeur stimule la réalisation d'extensions, car elles donnent de la valeur à son produit et témoignent aussi de l'existence d'une communauté, en soi un gage de pérennité. L'éditeur offre en général une plateforme de mise à disposition des extensions. Il peut le cas échéant mettre en place un dispositif d'évaluation ou de certification des extensions.

Ce modèle noyau/extensions est celui qui réalise le meilleur point d'équilibre entre les rôles respectifs de l'éditeur et de la communauté, réunissant la garantie et l'engagement de l'éditeur, avec le dynamisme et le l'énorme capacité de développement de la communauté.

Les « Forks »

On appelle « *fork* » une scission dans un projet de développement, dans laquelle une nouvelle équipe de développement part de la même base logicielle pour faire évoluer le produit à sa manière.

Les licences open source autorisent toutes les *forks*, c'est dans la définition même de l'open source.

Il peut y avoir en gros deux raisons pour un fork : un désaccord quant aux orientations technologiques, ou bien un désaccord quant à la politique commerciale et de licences.

Ainsi, si un éditeur prend une orientation qui déplait à la communauté, il s'expose à un *fork*. Par exemple en 2006, l'ERP Adempiere, naît d'un fork de Compiere résultant d'une politique commerciale tournant le dos aux communautés, suite à l'entrée d'investisseurs dans Compiere Inc.

Autre exemple fameux, le *fork* donnant naissance à Joomla en 2005, à partir de

Mambo, un outil de gestion de contenus très populaire.

Selon les cas, le *fork* peut l'emporter ou bien vivoter, selon le dynamisme de la communauté. Il y a aussi des *forks* non pas communautaires mais d'entreprises commerciales, par exemple l'ERP OpenBravo, s'appuyant également sur Compiere comme socle de son développement.

Le *fork* est une épée de Damoclès au dessus de la tête de l'éditeur, qui l'oblige à rester fidèle à ses valeurs et à sa communauté. Mais à contrario, certains éditeurs peuvent aussi en conclure qu'il vaut mieux éviter qu'il existe une réelle maîtrise de leur noyau dans les communautés.

Propriété Intellectuelle & Brevets

Aux États-Unis en particulier, les éditeurs proposent, en association avec leurs offres de support, une protection légale vis à vis de possibles violations de brevets logiciels. Dans ce pays, une entreprise qui utiliserait des programmes violant des brevets pourrait se voire attaquer et réclamer des indemnisations potentiellement énormes si la compagnie est riche.

C'est devenu, ces dernières années, l'un des axes d'attaque les plus virulents des concurrents de l'open source, à commencer par Microsoft, qui a utilisé son accord avec Novell pour entretenir l'idée qu'il existe un risque à cet égard. Et paradoxalement, ces intimidations servent aussi les éditeurs open source commerciaux, qui vendent finalement de la protection légale autant que du support produit.

En Europe toutefois, les programmes informatiques sont explicitement exclus de la Convention sur le Brevet Européen, et d'une manière générale le recours au judiciaire est moins dans les mœurs, de sorte qu'il n'y a pas de crainte semblable.

Éditeur-intégrateur

Le marché informatique se divise depuis longtemps entre éditeurs et intégrateurs. Les éditeurs développent des produits, susceptibles de satisfaire un nombre important de clients. Les intégrateurs s'appuient sur ces produits pour construire des systèmes d'information répondant au besoin spécifique d'un de leurs clients.

De tout temps, open source ou pas, il s'est trouvé des prestataires tentés par le double jeu : éditeur et intégrateur à la fois, éditeur qui intègre lui-même son produit, intégrateur qui n'a qu'un seul produit à son catalogue. Et de tout temps, cela n'a pas marché. Parce que pour gagner des marchés, l'éditeur doit construire un réseau d'intégrateurs partenaires, et il ne peut y parvenir s'il est lui même le premier concurrent de ses propres intégrateurs.

Sur le marché des solutions open source, on rencontre fréquemment de tels éditeurs-intégrateurs, mais ils ne donnent jamais naissance à des solutions leaders. La tentation est grande, pour l'éditeur qui a du mal à trouver son business model, et du mal aussi à convaincre des prestataires d'utiliser son produit, de le faire lui-même. Mais le marché sanctionne le plus souvent ces combinaisons.

[7.5] Les prestataires

Pour les prestataires IT, intégrateurs de solutions open source, le business model est pratiquement inchangé, basé sur la vente de prestations et d'expertise autour des produits open source, ceci sous la forme :

- De support
- De projets clés en main,
- De prestations de conseil ou d'expertise en assistance technique

Nous distinguons ici deux types d'activités, auxquelles correspondent des prestataires souvent différents : le support open source et l'intégration open source.

Les prestataires de support open source

Nous avons distingué ici distributeurs et prestataires, même si les distributeurs sont prestataires, à leur manière. Mais d'un point de vue historique, la frontière reste marquée. Les distributeurs n'offrent aucune autre prestation que la diffusion, le support et la formation, autour des logiciels inclus dans leur 'distribution'. Sur ces composants, sur le noyau Linux en particulier, ils ont une expertise pratiquement irremplaçable.

Mais il existe une telle diversité de composants open source, que le besoin est apparu très tôt d'avoir un support global, concentré entre les mains d'un prestataire unique. C'est sur ce métier qu'est né en France le concept de SSLL, Société de Service en Logiciel Libre : une société qui se propose d'assurer le déploiement et le support de configurations multi-produits à base de logiciels open source. Elles ont été rejointes plus tard par les SSII généralistes, ouvrant des centres de support open source.

Les prestataires de support open source, tels que Linagora par exemple, peuvent également construire des applications spécifiques, mais leur cœur de métier est dans le support, par exemple de produits prêts à l'emploi tels que la suite Open Office.

L'intégration de solutions open source

Les prestataires intégrateurs de solutions open source – tels que Smile – construisent des applications globales, des systèmes d'information, à base de logiciel open source.

Bien sûr, ils assurent également le support de l'ensemble de leurs créations, incluant les développements et configurations spécifiques, et produits sous-jacents, mais leur cœur de métier est dans l'intégration, la construction d'applications.

La valeur ajoutée de l'intégrateur open source commence dans le choix de solutions. L'open source amène une immense profusion de solutions, dont certaines immatures, ou au contraire obsolètes. Cette richesse est aussi un handicap : beaucoup de clients craignent de faire un mauvais choix. L'intégrateur open source ne peut pas attendre que le Gartner ait sélectionné les heureux élus, il doit mener une action permanente de veille et d'évaluation, afin de déceler les produits prometteurs, et les produits les plus solides.

Après cela, un projet d'intégration de systèmes à base d'open source ressemble à un projet d'intégration en général, et demande la même expertise méthodologique, tant en développement qu'en conduite de projets.

Au chapitre consacré au *modèle de développement*, nous verrons que l'open source a aussi beaucoup fait progresser le développement, et les prestataires intégrateurs sont les premiers à faire usage des bonnes pratiques, mais aussi des bons outils, issus de l'open source : IDE, gestion des sources, outils de tests et d'intégration continue, suivi des anomalies, etc. Les intégrateurs open source ont, en la matière, une longueur d'avance.

Prestation open source et prestation traditionnelle

Les produits open source communautaires n'ont pas d'éditeur susceptible d'apporter une aide

commerciale ou marketing, un support à l'avant-vente, ou un support en phase de projet. Même pour les produits open source commerciaux, les éditeurs sont souvent de petites structures, aux ressources limitées, et plutôt tournées vers le développement produit.

Le prestataire prend donc souvent à sa charge une partie de l'investissement amont qui incombait traditionnellement à l'éditeur : il sélectionne les produits les plus solides et pérennes, en assure la promotion et la vente, voire une partie du support.

Devant l'extraordinaire montée en puissance des solutions open source, tous les prestataires IT espèrent une part du gâteau ; c'est ainsi que les grandes SSII généralistes ont fini par s'y intéresser.

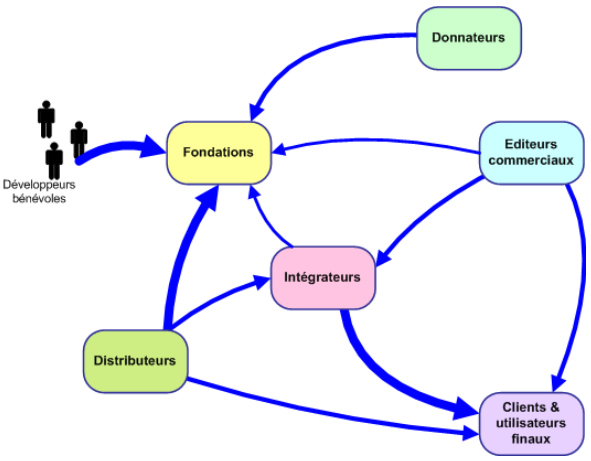
Mais à chaque révolution, qu'elle porte sur la technologie ou le modèle économique, les acteurs en place ne peuvent attraper la nouvelle vague. Parce qu'ils sont devant un dilemme que l'on peut résumer à *« peut-on se permettre de tuer la vache à lait historique ? »*.

Le phénomène est connu, et a été parfaitement analysé par C. Christensen dans *Tne Innovator's Dilemma (1997)*. Compagnies aériennes *majors* contre compagnies *low-cost*, opérateurs téléphoniques historiques contre nouveaux entrants, mais aussi IBM contre Microsoft, et maintenant Microsoft contre Google. L'histoire se répète et l'opposition libre vs propriétaire est du même acabit : un intégrateur traditionnel de peut pas renoncer à la manne qu'il tire du logiciel propriétaire et de ses prix élevés. A moins que l'open source ne soit exigé au cahier des charges, la SSII traditionnelle proposera un produit propriétaire. D'autant que, n'investissant pas en amont dans la veille technologique et la relation avec les communautés ou éditeurs open source, elles manquent de légitimité sur ces territoires nouveaux.

[7.6] Synthèse

En guise de synthèse, nous analysons les relations entre ces différents acteurs de l'open source, en considérant trois types d'interactions, en forme de *flux* :

- Les *prestations*, y compris écriture de programmes, support, conseil, formation, intégration.
- Le *code source*, c'est à dire les logiciels.
- *L'argent*, enfin, qui fait de tout cela un business-model !



Les flux de prestations

Sur la figure suivante, nous représentons les flux de prestations intellectuelle entre les différents acteurs identifiés. Cette prestation peut être du développement de programme, ou bien de l'intégration, du conseil, du support, de la formation.

On distingue les principaux flux de service :

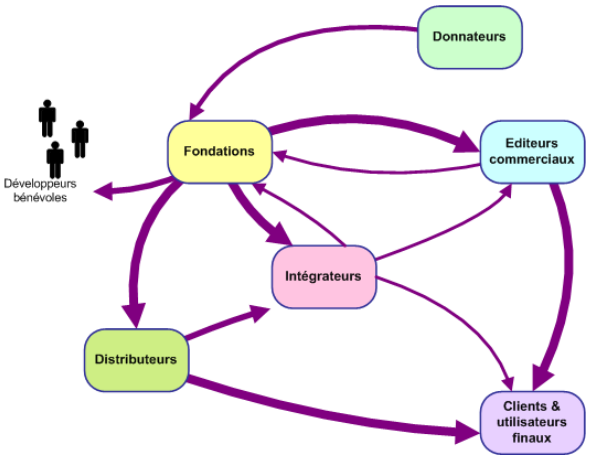
- Contributions en prestations de développeurs salariés, de la part des distributeurs tels que Redhat, de donateurs tels que IBM ou Google, et dans une moindre mesure, d'éditeurs commerciaux et d'intégrateurs, au bénéfice des fondations tels que Apache qui ont

en charge de grands projets open source.

- Offre de prestations d'intégration, développement et support des intégrateurs et des éditeurs commerciaux, vers les clients et utilisateurs finaux.

Les flux de code source

Sur la figure suivante, nous avons fait apparaître les flux de code source. Pour le distinguer de la *prestation de développement,* qui relevait des flux précédents, on ne considère ici que la livraison de programmes déjà écrits.

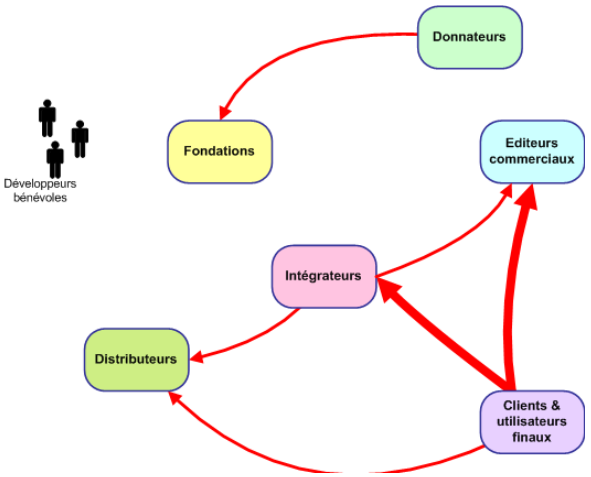


Ici, les principaux flux sont :

- Le code source constituant les grands logiciels open source, diffusés par les fondations et utilisés par les éditeurs commerciaux, et par les intégrateurs.
- Les programmes distribués par les distributeurs, à destination des clients finaux.
- Les programmes des éditeurs commerciaux, à destination des clients.

Les flux d'argent

Enfin, cette dernière figure représente les flux d'argent entre ces différents acteurs.



On y distingue les principaux flux suivants :

- Le paiement par les clients finaux des prestations de support aux éditeurs commerciaux et aux distributeurs
- Le paiement par les clients finaux des prestations d'intégration et de support aux intégrateurs.

[8] MODÈLE DE DÉVELOPPEMENT

[8.1] Introduction

Halloween document

En 1998, un mémo interne a filtré de chez Microsoft. Mis sur la place publique, il est connu depuis sous le nom de *Halloween Document I*[9].

Dans ce document, que Bill Gates en personne transmet à son *board*, un

[9] http://catb.org/~esr/halloween/

analyste étudie le mouvement open source et les dangers qu'il présente pour l'entreprise. Mais surtout, il reconnaît que :

- Les projets open source ont atteint ou dépassé la qualité des offres propriétaires
- Les projets open source sont maintenant des projets de grande échelle et de grande complexité
- Les projets open source ont des atouts spécifiques impossibles à reproduire en termes de motivation et de nombre de participants.

Au delà des considérations éthiques, au delà de la guerre commerciale, c'est un choc pour Microsoft que de réaliser qu'un mode de développement radicalement différent peut marcher aussi bien, et parfois mieux.

C'est ainsi que le modèle de développement des grandes applications est l'un des aspects les plus intéressant du mouvement open source.

Cathedral and the Bazaar

Ce que Microsoft découvre en 1998 a déjà été analysé, et théorisé par Eric S. Raymond dans un essai qui reste une référence : *The Cathedral and the Bazaar*.

Il y oppose le modèle de développement traditionnel, le modèle *cathédrale,* et le modèle de développement initié avec Linux, le modèle *bazar.*

Dans son analyse, le modèle cathédrale n'est pas uniquement celui des logiciels propriétaires, il est également celui de grands projets open source, tels que GCC. Ce modèle s'appuie sur une équipe de développeurs compacte, travaillant sur des cycles de développement relativement longs, et diffusant les sources à l'issue de chaque phase.

Le modèle *bazar*, au contraire, fonctionne dans un apparent désordre, où interviennent un très grand nombre de développeurs dans un cycle de production et tests en continu.

C'est ce modèle *bazar*, si représentatif de l'open source, que nous explorons plus avant.

Les grands projets communautaires

Quand on parle de modèle de développement open source, on parle uniquement des projets communautaires, et généralement des plus grands d'entre eux, tels que Gnome, Mozilla, Apache Httpd, Eclipse, Linux. Les éditeurs qui diffusent leurs produits sous licence open source ont en général des modèles de développements traditionnels, et n'ont pas une volonté particulière d'étendre leur communauté de développeurs.

Les études montrent que le plus souvent un petit nombre de programmeurs réalisent la plus grosse part des développements. Sur un projet où 200 programmeurs auront participé, on trouvera typiquement que 10 d'entre eux ont écrit 50% du code.

Comme tous les projets informatiques, les projets open source ont besoin de quelques leaders visionnaires et architectes de haut vol, pour à la fois montrer le chemin et définir le découpage en modules.

Sur les plus grands projets, on constate souvent que les principaux développeurs ne sont pas des bénévoles, mais sont salariés d'entreprises IT. Leurs employeurs ont différentes raisons pour les laisser travailler sur ces projets, des raisons qui peuvent être :

- De marketing : pouvoir faire état dans sa communication d'avoir un développeur *« commiter »,* sur un

projet phare équivaut à un important budget publicitaires.

- De gouvernance : c'est le moyen d'avoir son mot à dire sur les orientations stratégiques du produit.
- De socle technologique : ils font avancer plus dynamiquement un socle de produits dont ils sont directement utilisateurs, et dont dépend tout ou partie de leur business.
- De maîtrise : la société sera compétente et légitime pour proposer du support sur le produit.
- Voire également de motivation des collaborateurs, tant ceux qui participent que ceux qui pourraient le faire.

Une année de développement Linux

Le site LWN.net a publié une analyse très intéressante[10] des contributions au noyau Linux, d'où il ressort que sur une année de développement (2.6.16 vers 2.6.20) :

- 28 000 changements ajoutés
- Par 1 961 développeurs différents
- Remplaçant 1,26 millions de lignes par 2,01 millions de lignes de code nouveau.
- Le noyau a augmenté de 754 000 lignes.

Au total:

- Le développement est effectivement parallélisé à très large échelle
- Linus Torvalds n'est plus que l'auteur d'une très petite partie du code
- Une majorité des développeurs sont payés par leur employeur (Red Hat, IBM, Qlogic, Novell, Intel, ...), qui fournissent environ les 2/3 du code.

[8.2] Organisation, instances

Développeurs, commiters

Même si toute méthodologie cherche à rendre la qualité moins dépendante de la valeur individuelle des développeurs, il n'en demeure pas moins que l'expérience et le talent de chacun, mais aussi la motivation sont des paramètres fondamentaux.

Les projets open source, du moins les plus prestigieux d'entre eux, ont en général un avantage à cet égard. Ils attirent les meilleurs et les plus motivés des programmeurs, parce que faire partie des *commiters* Linux est la consécration suprême pour un développeur.

Les *commiters* sont les personnes autorisées à soumettre directement leurs contributions dans le référentiel des sources. Pour accéder au statut de *commiter*, il faut avoir proposé des contributions de qualité, et avoir gagné le respect de ses pairs. On est donc dans une logique de récompense du mérite et d'évaluation par ses pairs.

Gouvernance

Sur un projet, des choix sont à faire, des décisions sont à prendre. Qui décide et selon quel processus ?

D'une manière générale, les projets open source ont un fonctionnement relativement démocratique, dans le périmètre des commiters, avec tout de même une instance d'arbitrage, qui se réduit parfois au 'gourou' du projet.

Les fondations ont des institutions plus formalisées avec un *board of director* d'une

[10] http://lwn.net/Articles/222773/

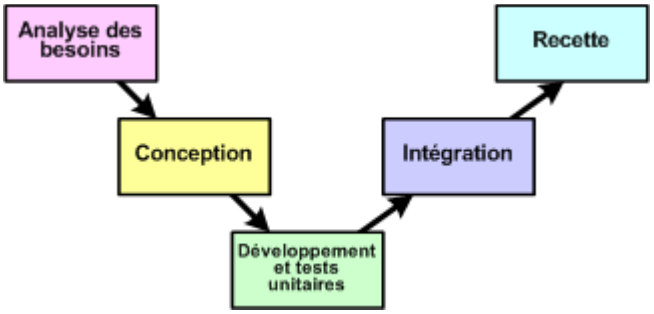
dizaine de membres, élus annuellement par les commiters. Le *board* se réunit périodiquement (virtuellement), de l'ordre d'une fois par mois, et prend ses décisions, qui sont publiées dans un compte rendu public.

Certains projets, Gnome par exemple, ont formalisé les règles de cette démocratie. Dans d'autres, un noyau plus réduit, voire une personne unique, rend les arbitrages ultimes. Pour le noyau Linux, typiquement, c'est encore le rôle qui revient à Linus Torvalds. Mais dans tous les cas, la hiérarchie au sein du projet n'est fondée que sur la valeur et la reconnaissance des pairs.

[8.3] Modèle de développement

Modèle de développement en cascade

Les modèles de développement traditionnels, que ce soit en cascade ou « cycle en V » ne conviennent pas aux projets communautaires.

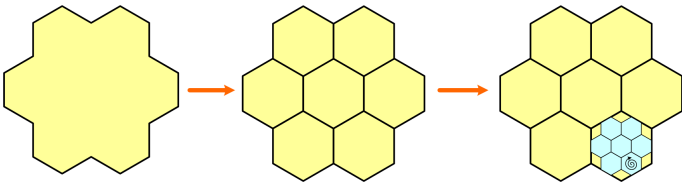


L'enchaînement linéaire de phases distinctes suppose une planification globale et une affectation des tâches centralisée. Par ailleurs, indépendamment même du contexte communautaire, ces modèles se prêtent mal aux très grands projets : ils ne permettent pas de bien gérer des besoins qui évoluent, et ils n'apportent pas suffisamment de retours d'expérience, de feedback, d'une étape vers la précédente.

Modularité impérative

Pour que quelques centaines de développeurs puissent travailler sans se marcher sur les pieds, il faut tracer des frontières propres et identifier des modules de dimension gérable par un développeur.

Si la bonne modularité, c'est à dire le découpage d'un grand projet en petites entités élémentaires, est un des principes élémentaires du génie logiciel, dans les grands projets open source cela devient une exigence vitale. La logique du « diviser pour régner » est incontournable.



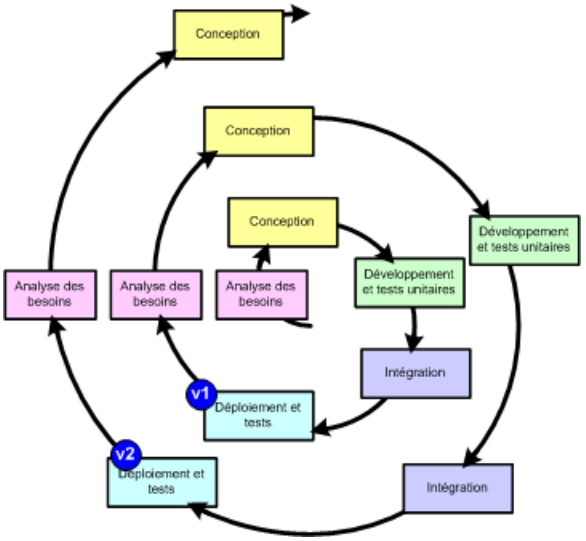
Le principe est donc de découper le système dans son ensemble en sous-systèmes dont les interfaces sont parfaitement définies, de sorte que chaque sous-système peut évoluer dans son développement indépendamment des autres, du moment qu'il respecte les interfaces convenues.

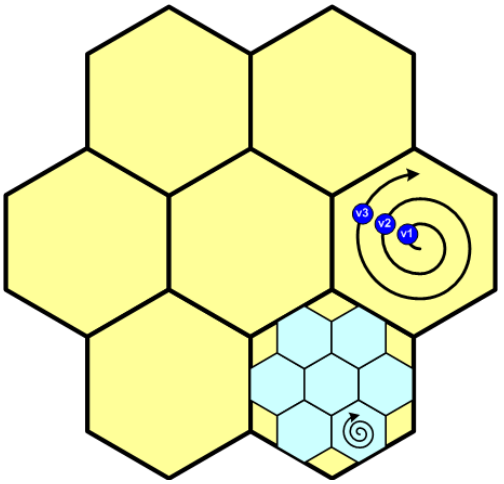
A l'intérieur d'un même sous-système, le découpage se poursuit, au niveau des classes, objets, fonctions.

Développement itératif

Le modèle de développement communautaire prédominant est appelé itératif, ou encore « en spirale ». Ses principes fondamentaux sont :

- d'une part un découpage en modules qui suivent chacun leur propre cycle de développement
- d'autre part l'itération de cycles courts répétés (spécifications, développement, intégration) de manière indépendante sur chacun des modules.





Et le tout dans le contexte général de l'intégration continue, qui permet de mesurer l'avancement, et de maîtriser les régressions, tout particulièrement dans les incompatibilités qui pourraient apparaître entre les modules.

Le modèle en spirale a également des aspects qu'on pourrait appeler « darwiniens », c'est à dire qu'à tout instant un développeur peut donner naissance à une version nouvelle d'un module, qui pourra ou non remplacer la précédente selon qu'elle est considérée supérieure.

On obtient donc le fonctionnement général représenté sur la figure suivante, où chaque module suit son propre cycle en spirale.

Il faut souligner que ces modèles de développement ne sont pas propres à l'open source communautaire. Ils peuvent convenir à une variété de grands projets. Mais ils doivent beaucoup à l'expérience des projets open source.

Test et accès au code

La phase finale d'un cycle de développement est la stabilisation et les tests. On parle de version béta lorsque le programme est entre les mains d'un sous-ensemble d'utilisateurs finaux volontaires.

Les projets open source ont une claire supériorité dans la phase de tests, à deux égards. Tout d'abord par le nombre plus important de personnes intervenant dans cette phase, et leur plus grande motivation. Et deuxièmement par le libre accès aux sources, qui permettra une meilleure qualification des anomalies.

[8.4] Les outils

Les grands projets open source ont fait progressé la méthodologie, mais ils ont aussi apporté énormément aux outils qui l'accompagnent.

Environnement intégré

Pendant longtemps, les outils de développement de l'open source étaient un peu rustiques. Sophistiqués au plan technique, mais peu travaillés au plan ergonomique.

Les choses ont changé toutefois, et comme on l'a vu plus haut, la plateforme Eclipse en est la meilleure illustration. Initialement plutôt orientée vers le développement Java, elle se prête aujourd'hui à tous les environnements, et son exceptionnelle modularité lui permet d'accueillir un nombre extraordinaire d'extensions. Même les équipes de développement qui ne sont pas spécifiquement orientées vers l'open source ont aujourd'hui adopté cette plateforme.

Gestion des sources

Les outils de gestion des sources tels que CVS ou SVN sont le fondement de tout développement communautaire. Ils permettent de gérer les ajouts simultanés de centaines de développeurs, en identifiant précisément chaque modification, son auteur, sa date et sa finalité, et permettent de revenir sur une modification.

L'usage de ces outils s'est aujourd'hui généralisé, mais les grands projets open source ne seraient tout simplement pas possibles autrement.

Outils de génération

Une autre famille d'outils, où excelle l'open source, est celle des outils de génération, qui permettent d'automatiser les opérations de génération d'un programme, en gérant les dépendances entre composants. Depuis le rustique *make* Unix, jusqu'à *Ant*, et plus récemment, la Rolls de la catégorie, *Apache Maven*.

Intégration continue

Les projets open source ont généralisé la pratique de l'intégration continue.

L'intégration continue consiste à générer et contrôler de manière quotidienne l'ensemble de l'application, afin d'identifier le plus en amont possible d'éventuelles régressions, erreurs ou incompatibilités entre modules.

On sait depuis longtemps que le coût de correction d'une anomalie croît de manière très forte avec le temps qui sépare son apparition dans le code de sa détection. L'intégration continue vise donc tout simplement à réduire ce temps au maximum : si au jour J un programmeur commite un changement comportant un bug, son erreur lui est signalée à J+1, et le coût de correction sera extrêmement faible.

L'intégration continue entre dans le cadre plus général du *test-driven development*, développement piloté par le test. Cette approche consiste à écrire les scénarios de test et mettre en place les outils associés, *avant* d'écrire les programmes. Les tests vont du niveau unitaire, jusqu'au niveau interfaces.

Les principaux outils d'intégration continue sont CruiseControl, et Continuum, qui est intégré à Maven.

Suivi des demandes et bugs

Moins sophistiqués, mais importants néanmoins, sont les outils de suivi des demandes et des anomalies, les *« issues »*, avec en particulier le célèbre *Mantis*, mais aussi Bugzilla, associé au projet Mozilla.

Outils d'échanges

Enfin, les développeurs utilisent pratiquement tous les outils d'échange communautaires existants, au développement desquels ils ont souvent participé :

- Mailing-lists automatique
- Forums
- Wiki, en particulier pour spécifications et documentations
- Messagerie instantanée

[9] CONCLUSION

J'espère que ce petit ouvrage vous aura permis de mieux comprendre l'open source, et que nous aurons su aussi vous communiquer un peu de notre enthousiasme.

L'open source est un mouvement merveilleux, à la fois par les valeurs qu'il porte, de liberté, de solidarité, d'ouverture, et par les bénéfices qu'il apporte, tant aux citoyens qu'aux entreprises.

Même si l'open source a ses racines bien antérieures au web, on peut affirmer qu'il porte aujourd'hui *la plus grande révolution de l'informatique depuis l'Internet.* Et le mot de révolution n'est pas excessif tant les positions en place sont bouleversées, et le modèle économique de nombreuses entreprises reconsidéré.

Pour aller plus loin, et apprécier les apports des solutions open source pour votre activité, pour votre entreprise, nous vous conseillons les autres livres blancs de Smile : gestion de contenus, portails, business intelligence, gestion de documents, ERP, … dans tous ces domaines, Smile a évalué les meilleures solutions open source du marché, et vous apporte ses retours d'expériences.

Expertises – Les livres blancs de Smile

Les livres blancs Smile sont téléchargeables gratuitement sur www.smile.fr

- **Introduction à l'open source et au Logiciel Libre**

  Son histoire, sa philosophie, ses grandes figures, son marché, ses modèles économiques, ses modèles de support et modèles de développement. [75 pages]

- **Gestion de contenus : solutions open source**

  Dans la gestion de contenus, les meilleures solutions sont open source. Du simple site à la solution entreprise, découvrez l'offre des CMS open source. [58 pages]

- **Portails : les solutions open source**

  Pour les portails aussi, l'open source est riche en solutions solides et complètes. Après les CMS, Smile vous propose une étude complète des meilleures solutions portails. [50 pages]

- **200 questions pour choisir un CMS**

  Toutes les questions qu'il faut se poser pour choisir l'outil de gestion de contenu répondra le mieux à vos besoins. [46 pages]

- **Conception d'applications web**

  Synthèse des bonnes pratiques pour l'utilisabilité et l'efficacité des applications métier construites en technologie web. [61 pages]

- **Les frameworks PHP**

  Une présentation complète des frameworks et composants qui permettent de réduire les temps de développement des applications, tout en améliorant leur qualité. [77 pages]

- **Les 100 bonnes pratiques du web**

  Cent et quelques « bonnes pratiques du web », usages et astuces, incontournables ou tout simplement utiles et qui vous aideront à construire un site de qualité. [26 pages]

- **ERP/PGI: les solutions open source**

  Des solutions open source en matière d'ERP sont tout à fait matures et gagnent des parts de marché dans les entreprises, apportant flexibilité et coûts réduits. [121 pages]

- **GED : les solutions open source**

  Les vraies solutions de GED sont des outils tout à fait spécifiques ; l'open source représente une alternative solide, une large couverture fonctionnelle et une forte dynamique. [77 pages]

- **Référencement : ce qu'il faut savoir**

  Grâce à ce livre blanc, découvrez comment optimiser la "référençabilité" et le positionnement de votre site lors de sa conception. [45 pages]

- **Décisionnel : les solutions open source**

  Découvrez les meilleurs outils et suites de la business intelligence open source. [74 pages]

- **Collection Système et Infrastructure :**
  - Virtualisation open source [41 pages]
  - Architectures Web open source [177 pages]
  - Firewalls open source [58 pages]
  - VPN open source [31 pages]
  - Cloud Computing [42 pages]
  - Middlewares [91 pages]

Contactez-nous, nous serons heureux de vous présenter nos réalisations de manière plus approfondie !
+33 1 41 40 11 00 / sdc@smile.fr

Smile
OPEN SOURCE SOLUTIONS