ANR PROPICE

Propriété Intellectuelle, Communs et Exclusivité
Les nouvelles frontières de l'accès et de l'innovation partagés

WP 2012-20

Pierre-André Mangolte
Centre d'Economie de Paris Nord – CNRS

Les communs du logiciel libre :

Deuxième partie :

La vie du commun

Work package 3 : « *Commons based production* : commun et innovation partagée »

Document de travail [*version 1 – novembre 2012*]

Les communs du logiciel libre

Deuxième partie : La vie du commun

par Pierre-André Mangolte
CEPN-CNRS
p.a.mangolte@wanadoo.fr
perso.orange.fr/lepouillou

version 1 - novembre 2012

I. Transition

Dans la première partie de cette étude, nous avons retracé les origines du logiciel libre, né d'un refus de l'utopie propriétaire mettant sous copyright, dans un système de droits exclusifs et réservés, toutes les ressources logicielles. La copie, le partage et le ré-usage des codes-sources existants étaient alors déclarées illégales, et la circulation des logiciels, hors des formes autorisées par les titulaires des droits, strictement interdite. La possibilité d'une coopération entre programmeurs, dépassant les limites de leur firme ou organisation, disparaissait alors. Et c'est contre cette situation et cet enfermement que sont nés les logiciels libres, avec leur propres licences et le principe d'un renoncement à toute exclusivité, par transfert aux utilisateurs recevant ou récupérant le code-source d'une manière ou d'une autre de la totalité des droits d'usage sur celui-ci, sans aucune réserve.

Le logiciel, devenu ainsi une ressource commune, est alors comme un texte entré dès sa publication dans le domaine public, un texte que chacun peut ré-utiliser à sa guise, y compris pour des usages commerciaux, la liberté de ré-impression étant alors totale, et les éditions concurrentes un fait bien établi. Le logiciel libre est même un peu plus, car l'auteur d'origine autorise la ré-écriture de son texte, allant jusqu'à la suppression de celui-ci, par morceaux ou en totalité; une liberté accordée à l'utilisateur qui s'explique par la nature particulière du logiciel, par l'existence des bogues, et la nécessité d'une transformation du texte pour conserver et améliorer la valeur d'usage du programme. La raison d'être de l'existence du logiciel libre est d'ailleurs là : permettre le travail sur le code, permettre le développement du logiciel. Le code peut donc être ré-utilisé pour produire un autre programme en améliorant l'ancien ou créer tout-à-fait autre chose. Pour sécuriser à l'avance ce travail et le principe de mise en commun du code-source, pour empêcher que le nouveau code donne naissance à des logiciels sous licence exclusive, une clause *copyleft* a été inventée, qui oblige le programmeur retravaillant un code copylefté à transférer aux utilisateurs

successifs les mêmes libertés d'usage qu'il a lui-même obtenu avec le code d'origine.

On peut alors distinguer dans l'ensemble des logiciels libres libres ceux qui sont sous clause *copyleft* et dont le code transformé ne peut sortir du commun, et ceux qui, n'étant pas copyleftés, peuvent à l'inverse, dans une nouvelle version, échapper à celui-ci. Cette clause sécurise donc durablement le commun. Elle interdit de retrancher quoique ce soit aux ressources partagées, et conduit quasi-mécaniquement à une augmentation de celles-ci ; elle fait du partage une obligation, et sert aussi de garantie à tous les programmeurs engagés dans le développement prolongé des logiciels libres.

Etudier cet aspect des choses est l'objet de notre deuxième partie. En effet, le système des licences des logiciels libres a mis en place un système de règles et proposé une base constitutionnelle pour une économie différente de celle qu'engendre l'usage habituel, exclusif ou propriétaire, des droits de copyright. Il a proposé, mais n'a pas directement créé. Il a proposé, en la sécurisant à l'avance juridiquement, la construction d'un commun, c'est-à-dire d'un ensemble de ressources destinées au partage, mais les licences elles mêmes ne peuvent créer ce commun. Le mystère – s'il y en a un – est que cette innovation institutionnelle du logiciel libre ait effectivement donné naissance à un commun, avec son adoption par un grand nombre de programmeurs et la production de nombreux logiciels libres, avec l'apparition aussi de formes de production originales, comme les grands projets de développement logiciel communautaires que sont Linux, Debian, Gnome, KDE, Mozilla, etc. Un autre mystère – pour autant qu'il y ait mystère ici – est l'intégration du phénomène logiciel libre dans les activités les plus ordinaires de l'industrie informatique, une industrie pourtant dominée par des logiques capitalistes et financières affirmées, avec des entreprises plus habitées à défendre leurs droits de propriété intellectuelle qu'à les abolir. Un point intéressant est d'ailleurs l'impact que ce commun a eu et a toujours sur l'organisation générale de l'industrie, sur les rapports entre firmes, entre les organisations, les administrations, et entre les autres utilisateurs de logiciels, ainsi que sur le degré de concurrence et de monopolisation.

Quand on observe en longue période, depuis les années 1980, l'évolution de la production des logiciels libres et les entrées dans le commun, on voit se dessiner une sorte de séquence, formée d'une succession d'étapes. Ce sont tout d'abord les outils logiciels les plus nécessaires aux utilisateurs-programmeurs dans leur travail de programmation qui apparaissent : compilateur, éditeurs, etc. Puis sous l'impulsion du projet GNU de la *Free Software Foundation*, les éléments constitutifs d'un système d'exploitation entièrement composé de logiciels libres se mettent progressivement en place. Le noyau de ce système d'exploitation, indispensable au projet d'ensemble, surgit lui tout-à-fait indépendamment de la FSF[1], avec la formation et l'essor du

1 Pour le noyau du système d'exploitation, la *Free Software Foundation* avait choisi une architecture à micro-noyau, différentes des noyaux monolithiques traditionnels des systèmes Unix de l'époque, une conception modulaire, plus facile en principe à faire évoluer dans la durée que les architectures antérieures, mais bien compliquée à mettre au point. C'est le projet Hurd toujours en développement à ce jour et toujours imparfait. Linus Torvalds devait faire un

développement communautaire de Linux, lequel est cependant sous licence GNU-GPL. Au milieu des années 1990, les qualités propres de Linux sont reconnues non seulement par les communautés de programmeurs partisans du logiciel libre mais aussi par une partie de l'industrie, qui ignorait jusqu'alors le phénomène. Celle-ci découvre alors l'importance et les potentialités d'un développement communautaire distribué. Mais à l'époque, ce système GNU/Linux et tous les logiciels libres qui l'accompagnent ne peuvent être utilisés qu'en ligne de commande, au moyen du *shell*. Les différents composants nécessaires à la gestion de l'interface graphique utilisateur (*Graphical User Interface*) n'existe pas en effet alors comme logiciel libre, et les utilisateurs du commun ne sont pour l'essentiel que des utilisateurs-programmeurs.

La période suivante, un peu avant l'an 2000, voit l'adoption de Linux par une partie de l'industrie, en remplacement des Unix propriétaires qui sont abandonnés l'un après l'autre. Linux trouve aussi sa place, bien vite dominante, dans le code embarqué, les serveurs, la conduite de certains processus industriels, et l'ingénierie des systèmes informatiques, comme les supercalculateurs. L'activité du service logiciel découvre les avantages du libre, et les premières sociétés commerciales vendeuses de logiciels libres et de services associés apparaissent. Par ailleurs, les premières libération de code, avec *Netscape* et le projet Mozilla, ont lieu à la fin des années 1990.

Parallèlement et indépendamment, les communautés du libre s'attaquent au problème de l'interface graphique, afin de doter le système d'exploitation GNU/Linux de différents environnements de bureau (ce sont les projets KDE (1996), Xfce (1996), Gnome (1997), etc.), et d'atteindre ainsi le grand public peu habitué à piloter son ordinateur en ligne de commande. Les premières distributions GNU/Linux apparaissent sous forme de projets communautaires : Slackware (1992), Debian (lancé en 1993), Gentoo, suivies par des distributions commerciales pilotées par des firmes : RedHat (1994), Mandriva Linux (1998), Suse (1994, appartenant aujourd'hui à *Novell*), etc. C'est une nouvelle sorte de production. Il s'agit en effet de constituer une collection de logiciels, tous libres (cas de Debian par exemple) ou en majorité libres mais comportant certains logiciels propriétaires (cas de la plupart des distributions commerciales), qui est assemblée en un tout cohérent autour d'un noyau Linux, pour éviter aux utilisateurs de ce système d'exploitation d'avoir à composer eux-mêmes cette collection en réunissant tous les logiciels dont ils ont besoin, avec tous les problèmes que pose ce genre d'intégration. Ces distributions sont destinées tant aux utilisateurs-programmeurs qu'aux utilisateurs ordinaires, avec des choix techniques, des modèles économiques (commercial ou non), et des philosophies différentes. Elles s'adressent aussi à toutes les catégories d'usages, et sont de fait très nombreuses

choix inverse, en préférant produire le Linux *kernel* comme un gros bloc de code, un noyau monolithique donc, une solution traditionnelle déjà bien connue. IL défendit d'ailleurs cette approche pragmatique et prudente (et conservatrice du point de vue de l'innovation), contre toutes les critiques (voir ses échanges avec Andrew Tannenbaum, le spécialiste des système d'exploitation dans Behlendorf et alii. 1999, p 237 et suivantes.

(plus de 500 aujourd'hui)[2]. L'économie non marchande du logiciel libre cohabite alors avec des entreprises cotées en bourse, commercialisant leur propre distribution GNU/Linux, accompagnée de services d'aide et de support.

Depuis, la production et la diffusion des logiciels libres s'est encore diversifiée, envahissant progressivement les différents domaines de l'économie et de la société où des logiciels sont présents, en s'installant sur toutes les plates-formes, y compris les plus récentes, comme les smarphones et tablettes (avec par exemple Android), les systèmes de navigation GPS (avec TomTom), et dans toutes les niches en matière d'usages.

Dans cette deuxième partie, nous allons revenir sur ces différents points. Au préalable, il nous faudra traiter une question qui, analytiquement, n'a rien à voir avec le logiciel libre, rien à voir avec le commun, mais qui, concernant toute l'économie des logiciels, libres ou propriétaires, a aussi son importance pour le commun. C'est le phénomène des gratuiciels (ou *freewares*) (Point II). Nous analyserons ensuite l'importance du logiciel libre pour l'innovation (Point III), et l'économie des logiciels libres (Point IV), avec au centre les développements pris en charge par des réseaux d'utilisateurs (ou communautés). Nous verrons enfin de l'impact du commun sur l'industrie et des entreprises face au commun (Point V).

II. Préalable : la place du gratuit dans l'économie des logiciels

Dans l'économie actuelle des logiciels, à la différence de la plupart des autres secteurs de l'économie, une partie importante des produits est fourni et circule sans contre-partie monétaire. On parle alors de *freewares* ou de gratuiciels, un phénomène différent du logiciel libre. La fourniture gratuite de code source, voire de versions compilées, adaptées à telle ou telle machine (binaires), est une réalité ancienne de l'industrie informatique remise en cause au cours des années 1980 par l'essor des progiciels commerciaux et les stratégies propriétaires en matière de code. La part du logiciel fourni gratuitement avait alors fortement régressé. Cette tendance aurait du normalement se prolonger, et la gratuité ne plus représenter aujourd'hui qu'une part marginale dans l'ensemble de la distribution des programmes. Et pourtant, cela n'a été le cas, car il y a toujours des logiciels qui sont échangés et circulent sans contrepartie monétaire, qu'il s'agisse d'un code source en cours de développement (comme pour les logiciels libre), ou de versions binaires, stables et bien déboguées, immédiatement installables dans un ordinateur, et prêtes à l'emploi. Aujourd'hui, avec la généralisation de l'accès à l'internet et la connexion des ordinateurs à ce réseau de transport, la part du gratuit a d'ailleurs plutôt tendance à s'étendre qu'à régresser.

2 Cf. < http://lwn.net/Distributions/ >. Cette diversité contraste avec l'unicité des univers de Microsoft et d'Apple, et a d'ailleurs été critiquée, mais Linus Torvalds la défend au contraire avec vigueur. A côté de quelques distributions orientées « grand public », il est normal que se multiplient des distributions spécialisées chacune sur un créneau particulier, dans une logique où ce sont les usages et les utilisateurs qui imposent leur loi.

Un grand nombre de programmes sont fournis gratuitement par les entreprises de l'industrie informatique, et même parfois uniquement sous cette forme, la vente ayant totalement disparu. C'est le cas par exemple des producteurs d'imprimantes et consommables (encres), lesquels fournissent gratuitement leurs pilotes dédiés, mais aussi le cas des éditeurs commerciaux de logiciels qui mettent une partie de leur offre en gratuiciels sur l'internet, en liaison ou non avec une autre offre purement commerciale (logiciel de lecture d'un certain format de document, l'éditeur du format étant lui-même vendu, par exemple), et d'autres qui l'utilisent comme une forme temporaire de distribution (programme d'essai ou de démonstration, bridé ou non), etc. Certaines applications complexes et importantes, comme les navigateurs, ne sont même distribuées que sous cette forme, le « marché » ayant été ici en quelque sorte « tué » par la décision de Microsoft de fournir gratuitement *Internet Explorer* avec son propre système d'exploitation. S'ajoutent ici tous les programmeurs isolés qui choisissent cette forme de distribution avec des motivations souvent proches de celles de l'*open source* (apprentissage personnel, volonté de partage, priorité donnée à la publication et à la diffusion). Dans ces gratuiciels, on trouve donc de tout : des applications complètement fonctionnelles comme des versions bridées, des logiciels libres comme des logiciels propriétaires.

Dans l'ensemble des gratuiciels, il y a aussi des logiciels libres, mais il n'y a rien dans la définition du *free software* ou du logiciel *open source* qui impose une quelconque gratuité en matière de distribution ou redistribution, qu'il s'agisse du code source ou des versions binaires. Le terme *free* en anglais peut cependant prêter à confusion, signifiant tout-à-la-fois « libre » et « gratuit » ; mais comme le rappelle régulièrement la *Free Software Foundation* : « *Free software is free speech and not free beer* ». Le logiciel libre est en effet « libre » au sens de la liberté ; ce qui signifie aussi que l'utilisateur qui détient le code et l'a reçu d'une manière ou d'une autre, doit rester totalement libre de le distribuer gratuitement ou moyennant paiement. Il n'existe donc aucune clause dans les différentes licences qui impose la gratuité, et ceci même dans le cas où on a soi-même reçu le logiciel gratuitement. Il n'existe aucune clause qui interdit de choisir entre la vente ou la gratuité, aucune clause qui encadre ou limite le prix que l'utilisateur pourrait exiger. Un logiciel accompagné d'une licence comportant de telles clauses ne serait d'ailleurs ni un logiciel libre (pour la *Free Software Foundation*), ni un logiciel *open source* (pour l'*Open Source Initiative*). Imposer la gratuité serait d'ailleurs absurde, car un tel principe interdirait toute vente éventuelle du code et toute distribution commerciale, ce qui limiterait fortement les possibilités de diffusion. Il empêcherait aussi de payer éventuellement les producteurs du code. La vente (et revente) du code-source ou d'une version binaire (avec accès au code-source) n'est donc aucunement interdite par les règles du commun.

Les licences mettent cependant tous les utilisateurs du logiciel sur le même plan. Ils ont les mêmes droits sur le logiciel, et l'auteur d'origine, celui qui possède formellement le copyright, n'a

pas plus de droits que les autres. Tous peuvent donc copier et redistribuer le logiciel autant de fois qu'ils le souhaitent; et ils peuvent le vendre ou le donner, sans pouvoir interdire aux autres de faire de même. Les licences de logiciels libres mettent donc tous les utilisateurs, tous ceux qui d'une manière ou une autre ont reçu une copie, sur un pied d'égalité et en concurrence les uns avec les autres, et cette situation empêche à moyen ou long terme tout prélèvement de rente. Le prix réellement exigible ne peut alors guère dépasser, sauf cas particuliers, le coût de la distribution (bandes magnétiques, disquettes, disques, frais de commercialisation éventuels d'un *package*, etc.), ou le paiement des services accompagnant le programme (aide à l'installation, formation, maintenance, etc.). Dans certains cas cependant, le logiciel peut être vendu et cette vente se maintenir dans la durée, dans le cas par exemple d'un paiement direct à l'auteur du code, ou à l'équipe, à l'organisation ou à l'entreprise qui a produit le programme[3]. On peut avoir aussi deux distributions qui cohabitent, une marchande, avec ou sans services d'accompagnement, et une gratuite. Aujourd'hui cependant, comme la distribution d'un grand nombre de logiciels libres utilise l'internet, en particulier pour les programmes destinés à un grand nombre d'utilisateurs, une forme très usitée est celle du gratuiciel (ou *freeware*).

Cette situation ne découle pas directement du système des logiciels libres. C'est plutôt l'existence de l'internet, une infrastructure de transport qui réduit à presque rien le coût de déplacement des fichiers numériques, et la nature matérielle des ressources mises en commun (des fichiers numériques) qui est à l'origine des gratuiciels. Les libertés de copier et de redistribuer jouent cependant leur rôle en plaçant toutes les activités de distribution (et de vente) sous la menace permanente des nouveaux entrants (et donc du gratuit). La situation est d'ailleurs telle aujourd'hui qu'on peut facilement trouver un logiciel gratuit pour la plupart des tâches confiées à un ordinateur dans un cadre privé ou domestique, avec ici des logiciels propriétaires comme des logiciels libres. On a alors une sorte de commun différent du commun des logiciels libres, un ensemble de ressources gratuites et accessibles où toute une population d'internautes peut puiser, et qu'ils peuvent utiliser légalement comme exécutables, dans les limites fixées par les différentes licences. Deux sphères cohabitent alors dans l'économie des logiciels, une sphère marchande et une sphère non marchande, avec parfois, pour les mêmes produits ou pour des produits analogues, coexistence de la vente et de la gratuité. On peut s'interroger sur les conditions de cette situation, difficile à imaginer dans d'autres industries et pour tout autre type de produits. Tout repose économiquement sur le fait que les logiciels sont des fichiers numériques, qui peuvent être dupliqués très facilement et quasiment sans frais. Nous avons vu dans notre première partie (voir l'Annexe A en particulier) qu'il y avait ici deux processus de production différents, séparés techniquement l'un de l'autre, et accompagnés de surcroît de leurs propres coûts de production : (1) le processus de développement du logiciel et (2) la production de copies de ce même logiciel à

3 Cf. le témoignage de Stallman, dans notre première partie, note 18.

un moment donné du temps, par duplication du code source, ou duplication d'une version compilée de celui-ci. La « copie » ici est bien une duplication, donnant un produit totalement identique au fichier d'origine. Cette opération de plus ne détruit rien. Elle ne modifie ni le produit copié ni le logiciel en développement, et ceci même quand la copie est répétée plusieurs fois.

A la différence du développement qui est toujours coûteux en temps et en temps de travail, représentant inévitablement un certain nombre d'heures de travail pour un certain nombre de programmeurs (et souvent pour d'autres personnes), le coût des opérations de duplication est toujours faible. Quand l'internet n'existait pas encore ou n'était pas totalement déployé, il fallait pour distribuer un programme le copier sur des bandes magnétiques ou sur des disques, et ajouter la documentation sous forme papier. La copie sur bandes ne consommait que du temps machine, fonction cependant du nombre d'exemplaires à produire, mais les bandes étaient coûteuses en elles-mêmes, sans même parler de la documentation papier et du prix de transport. Aujourd'hui, ces coûts ont été réduits à presque rien, car tout peut prendre la forme de fichiers numériques, y compris la documentation, et le transport de ses fichiers est fait par l'internet. Le coût de distribution d'un logiciel déjà développé est alors limité au coût de production d'une copie (une seule), auquel s'ajoute éventuellement la fabrication (par compilation, etc.) de différentes versions binaires et la mise en ligne de l'ensemble. Le reste est pris en charge par ceux qui viennent télécharger le programme. Le coût des téléchargements, c'est-à-dire la duplication de la copie mise en ligne et son transport jusqu'au disque dur des utilisateurs, est supporté par ceux-ci, qui de plus installent eux-mêmes le programme sur leur machine. Ce coût est de toute manière très faible et semble même n'avoir aucune existence, les téléchargements n'étant pas payés à l'unité, mais intégrés dans un abonnement qu'on paye de toute manière, qu'elle que soit l'utilisation de la bande passante.

Quand un programme est déjà développé et déjà financé, le distribuer comme gratuiciel est non seulement possible, mais représente la solution la moins coûteuse à mettre en place. Le choix d'une distribution commerciale impose par contre des dépenses répétées qui peuvent s'avérer fort lourdes : publicité, investissements marketing, coûts de commercialisation pour une vente éventuelle de *packages* sur linéaires, etc.[4], tous ces investissements étant nécessaires pour transformer un programme déjà développé et pleinement fonctionnel en une marchandise, pour défendre une part de marché, ou rendre obsolètes des produits plus anciens. Les profits peuvent être cependant considérables si les ventes sont suffisamment importantes, mais ce n'est pas toujours le cas, et rarement le cas pour tous les concurrents (et logiciels) en présence. Si les recettes commerciales obtenues sont faibles, il est plus rationnel de distribuer le logiciel gratuitement sur

4 Ces coûts commerciaux (ramenés à presque rien pour une distribution en gratuiciel) représentent la plus grande partie du « coût total » des progiciels commerciaux. Campbell-Kelly (2003) donne ainsi un ordre de grandeur : 35 à 50 % pour le marketing contre 10 à 15 % pour le développement proprement dit, le reste (40 à 50 %) étant absorbé par les frais de production et de distribution physique des *packages* logiciels.

l'internet, car cette solution peu coûteuse assure toujours une certaine diffusion, l'autre alternative étant le renoncement à toute distribution.

La forme numérique des logiciels et l'existence de l'internet ouvre la possibilité d'une diffusion à très large échelle des logiciels, et plus généralement de tout ce qui peut prendre la forme du fichier numérique. S'ouvre alors la possibilité d'une multiplication à l'infini des copies, une multiplication générée directement par la demande et les besoins des utilisateurs, ce qui représente une sorte d'économie d'abondance[5], dont l'existence réactive les questionnements des débuts de l'industrie informatique sur la valeur marchande (et le prix) des logiciels. Les utilisateurs peuvent télécharger un logiciel, l'installer sur leur machine, le tester, l'adopter ou l'envoyer à la poubelle sans regret, et quasiment sans frais. Ils peuvent tout aussi bien aller en chercher un autre, plus performant et plus attractif à leurs yeux, n'étant limité ici que par l'offre des gratuiciels disponibles à un moment donné sur l'internet.

Dans le modèle propriétaire du progiciel commercial, on fait dépendre le financement du développement des recettes tirées de la vente des copies de versions successives du même logiciel en développement. Ce modèle obéit à la formule générale du capital : A – M – A' (avec A' > A), où les *avances* monétaires (A) doivent, après la production et la vente des marchandises (M), revenir sous forme monétaire augmentées d'un excédent (A')[6]. Mais rien n'impose pour financer un développement logiciel de retenir cette solution, rien n'impose en réalité de lier et de faire dépendre le développement logiciel de la vente de copies. L'idée, systématiquement ressassée pour défendre cette solution selon laquelle le « piratage », c'est-à-dire une certaine liberté de copie et de partage, conduirait inévitablement par manque de financement à la ruine de tout développement logiciel et à la paralysie de l'innovation, est d'ailleurs manifestement fausse, invalidée par l'essor et le déploiement d'une économie du logiciel libre. L'erreur consiste ici à confondre deux processus de production distincts, développement et duplication, et à les fusionner en un seul, mais les deux processus sont en réalité séparés, et peuvent être financés indépendamment l'un de l'autre. C'est pourtant sur cette confusion, sur cette idée fausse de l'existence d'une liaison impérative entre les recettes tirées de la vente de copies binaires (avec protection par secret de fabrication, copyright, etc.) et le financement du développement logiciel que repose le modèle commercial du progiciel propriétaire. Mais il n'y a aucune nécessité, ni technique ni économique, à attribuer aux copies le

5 Le gratuiciel n'est d'ailleurs qu'un exemple particulier dans l'ensemble des produits et contenus mis en ligne et offerts gratuitement aux autres par certains internautes (information et aide en ligne, blogs, pages persos, photos, images, enregistrement sonores ou musicaux, vidéos, etc.).

6 Cf. Marx, 1872-1875, Livre I, 1, p. 151 et suivantes. Comparant la formule de la circulation M – A – M et la formule générale du capital A – M – A, Marx, citant James Steuart (1805), distingue entre les *avances* (les investissements) et les *dépenses*, le dernier terme de la formule de la circulation, « *où l'argent est enfin converti en marchandise qui sert de valeur d'usage; il est donc définitivement - dépensé. Dans la forme inverse A – M – A , l'acheteur donne son argent pour le reprendre comme vendeur. Par l'achat de la marchandise, il jette dans la circulation de l'argent qu'il en retire ensuite par la vente de la même marchandise. S'il le laisse partir, c'est seulement avec l'arrière-pensée perfide de le rattraper. Cet argent est donc simplement - avancé* » (p. 153).

coût du développement logiciel.

En matière de duplication, les coûts étant quasiment nuls, la diffusion comme gratuiciel est la solution la moins coûteuse, et partant la plus rationnelle économiquement. En matière de développement, la situation est complètement différente, car les coûts peuvent être considérables pour un processus de production qui reste en principe unique et s'inscrit inévitablement dans la durée. On a alors plusieurs modes de financement possibles, lesquels peuvent d'ailleurs être combinés entre eux, soit comme *avance*, selon la formule générale du capital, soit comme *dépense*, une dépense motivée par la volonté d'obtenir une certaine valeur d'usage, un logiciel donc, sans retour sous forme monétaire des sommes d'argent engagées dans la production, sans volonté non plus de couvrir les coûts ou d'obtenir un excédent.

(1) *Avance* en capital (ou investissement)	**A – DVT – A'**	vente du code source (dont logiciels libres) et vente de copies binaires (progiciels commerciaux propriétaires)	pas de gratuiciels
(2) Investissement joint	**A – M – A'** \ **DVT**	financement par les profits tirés d'une autre activité commerciale, liée ou non au développement logiciel	existence possible de gratuiciels
(3) *Dépense* « à fonds perdus »	**A – DVT**	financement public ou privé, collectif ou individuel, dont la seule finalité est le logiciel lui-même, l'innovation, la recherche	existence possible de gratuiciels
(4) Financement « passé »	**DVT1 → DVT2 →**	financement par réutilisation d'un code existant, produit d'un développement passé	existence possible de gratuiciels
Note : **DVT** = « développement logiciel »			

Tableau 1 : Les modes de financement du développement logiciel

Dans le premier cas, on est bien dans la formule générale du capital, avec par exemple la vente d'un logiciel entièrement développé ou du travail sur le code; c'est le cas assez général des *SSII* (*Sociétés de Services en Ingénierie Informatique*), spécialisées en ingénierie, en production de logiciels sur commande, en opérations de maintenance. Il n'y a ici en règle générale ni vente de binaires, ni fourniture de gratuiciels, il y a par contre vente directe du logiciel ou d'un service logiciel, la plupart du temps avec fourniture du code source, les recettes obtenues payant le travail du développement. Certains résultats peuvent d'ailleurs être réutilisés pour répondre à des contrats ultérieurs sur le mode de l'investissement passé (cas n° 4); et le logiciel peut être un logiciel libre,

ou du code accompagné d'une licence restrictive. Toutes les situations sont possibles.

Dans le deuxième cas, l'investissement croisé ou joint, une activité commerciale rentable finance le développement d'un programme (sur le mode de la dépense)[7]. On peut distinguer deux cas extrêmes : (a) La situation où la production du logiciel n'a qu'un rapport lointain avec l'activité générant les recettes, et alors il peut y avoir ici un projet de développement logiciel totalement autonome, à l'abri de toute pression commerciale, le développement étant uniquement orienté vers la production et l'amélioration du logiciel comme valeur d'usage. Cette situation est presque une règle générale – comme nous le verrons par la suite – pour les projets de développement *open source*. (b) La situation où existe un lien entre l'activité générant les recettes et le travail de développement financé par celles-ci. Le logiciel développé transforme, améliore ou renforce l'efficacité de l'organisation productive de la firme, ou son offre commerciale, sans être pourtant directement vendu.

Dans le troisième cas, les « dépenses à fonds perdus », on a une rupture bien plus nette avec la formule générale du capital, le but de ce financement, qu'elle qu'en soit la source (contributions d'entreprises, fondations, finances publiques), étant d'entretenir un projet de développement particulier, ou d'une ensemble de développements logiciels, dans l'espérance d'obtenir des logiciels différents par leurs qualités techniques ou leur statut juridique (logiciel libre par exemple) de ceux qui existent déjà.

Le dernier cas, ce que nous appelons le « financement passé », existe dès que l'on ré-utilise du code issu d'un développement logiciel antérieur. Dans une situation d'utopie propriétaire, où le code source est enfermé dans le système des droits exclusifs du copyright, ce mode de financement est réservé en quelque sorte au logiciel libre, avec la liberté inscrite dans les licences de ré-utilisation du code, ou aux entreprises et organisations qui en interne peuvent ré-utiliser leur propre code[8].

Dans les cas (2), (3), (4), on peut avoir fourniture de gratuiciels. Le financement du travail de développement n'étant pas directement lié et totalement dépendant des recettes éventuelles obtenues dans la distribution du programme. Comme ces différents manières de financer une activité économique existe déjà, en matière de production de logiciel comme dans beaucoup d'autres activités, et que le coût de la diffusion des copies éventuelles est pratiquement nul, on a ici la possibilité économique de voir émerger – ce qui est aujourd'hui une réalité – une économie

7 On peut donner comme exemple le navigateur *Opera*, dont les binaires sous licence propriétaire sont disponibles gratuitement pour un grand nombre de systèmes d'exploitation, la société *Opera Software* se finançant en vendant des licences d'utilisation sur son moteur de rendu *Presto* (le moteur qui assure l'affichage des pages web), lequel est intégré aujourd'hui dans des produits comme *Adobe Dreamweaver*, *Adobe Creative Suite*, et certaines consoles de jeu de la firme *Nintendo* (*Nintendo DS* et *WII*).

8 La règle du *work-for-hire* et les différents contrats salariaux ou de sous-traitance transfèrent en effet les droits de copyright des programmeurs individuels à une entité collective, l'entreprise (ou l'organisation), ce qui en interne, dans les limites de cette entité, rend possible le partage du code et le travail en coopération.

stable des gratuiciels, incluant même la production de ceux-ci.

III . Les utilisateurs, l'innovation, et les logiciels libres

Le principe du commun des logiciels libres est le partage de différentes ressources logicielles avec les mêmes droits d'usage pour tous les utilisateurs. Les licences ne font d'ailleurs aucune différence entre les utilisateurs. Elles n'évoquent en effet qu'un utilisateur générique et abstrait, lequel obtient le droit d'exécuter, copier, distribuer, modifier et redistribuer comme il l'entend le logiciel, c'est-à-dire le code source et la documentation qui l'accompagne, et les fichiers binaires compilés à partir de ce code. Mais la « ressource logicielle » est une valeur d'usage complexe qui admet des usages différents, et la « ressource logiciel libre », c'est-à-dire le code plus les libertés fondamentales du logiciel libre (avec ou sans clause *copyleft*), est une valeur d'usage encore plus complexe. Dans cette partie, nous ne nous intéresserons qu'à deux catégories d'utilisateurs, les utilisateurs ordinaires (ou utilisateurs finaux), dépourvus pour la plupart de toute capacité en matière de programmation, et les utilisateurs-programmeurs qui à l'inverse sont suffisamment compétents pour, s'ils le souhaitent, pouvoir retravailler le code source. Nous laisserons donc de côté provisoirement les entreprises[9].

Usages et types d'utilisateurs

L'utilisateur ordinaire (ou final) d'un logiciel est essentiellement intéressé par ses fonctionnalités, ce pourquoi le logiciel a été conçu et développé. Cet utilisateur veut un exécutable pour réaliser par exemple des tâches bureautiques, modifier une image numérique, produire un dessin, gérer les réservations d'une compagnie aérienne, ou contrôler la marche d'une ligne de métro, d'un robot de soudage, etc. La variété des valeurs d'usage demandées par les utilisateurs finaux, qu'il s'agisse de simples particuliers, d'entreprises, d'organisations, est d'ailleurs aujourd'hui extrême, en relation directe avec la numérisation d'un grand nombre de produits culturels, l'importance acquise par l'internet et la pénétration croissante des logiciels dans toutes les machines que nous utilisons.

Pour ce type d'utilisateur, une forme fixée (suffisamment stable) d'un logiciel en développement convient généralement. Ce qui l'intéresse en effet est un outil bien défini, avec ce que cet outil est capable de faire. Si l'utilisateur n'est pas pas lui-même programmeur ou n'envisage pas de modifier le programme mais veut plutôt l'utiliser tel quel, l'accès au code source ne lui sert à rien, et sa recherche de valeur d'usage est parfaitement satisfaite par une version binaire déjà

9 Le logiciel libre présente en effet d'autres « valeurs d'usage ». On peut utiliser le commun pour ses règles ou pour les conséquences que ces règles produisent dans l'environnement économique et les rapports entre partenaires commerciaux. C'est alors un « usage stratégique » du commun et des ressources logicielles libres que nous aborderons dans notre point V).

compilée pour son propre système d'exploitation, ou par une version stable qu'il peut compiler lui-même. Ici la différence entre logiciel propriétaire et logiciel libre reste limitée, dans la gratuité éventuelle, dans la possibilité de copier légalement et d'installer sur des machines différentes, dans la facilité du téléchargement, dans la garantie de disposer de mises à jour et de voir le programme s'améliorer éventuellement, etc. Pour ce type de valeur d'usage, les différences les plus importantes sont celles qui peuvent séparer des produits aux fonctionnalités comparables sans être totalement identiques.

Mais il existe une autre catégorie d'utilisateur, celle des utilisateurs-programmeurs (ou développeurs). Leur demande est complètement différente, car ce dont ils ont besoin, ce qu'ils recherchent, c'est du code source, avec la possibilité de le modifier, de le retravailler, et d'en faire autre chose. La valeur d'usage recherchée n'est donc pas un produit fini, un programme tout fait auquel on ne touche pas, mais un certain code source utilisable comme matériau pour un travail de développement. On trouve ici potentiellement tous ceux qui, maîtrisant les langages de programmation, effectuent des opérations de maintenance sur des logiciels existants ou développent à partir d'un code existant un nouveau logiciel, des tâches qui impliquent de pouvoir légalement accéder au code avec le droit de le modifier. Le code libre est en effet un point de départ possible pour l'innovation, soit pour améliorer le comportement d'un programme utilisé, soit pour lancer un tout autre développement logiciel.

Dans le système des logiciels propriétaires à code fermé, seuls certains programmeurs sont autorisés à accéder au code source. Les licences de logiciels libres ont justement été inventées pour changer cette situation, pour instituer une circulation libre du code en permettant son ré-emploi, et c'est sans doute la raison principale de leur succès. Elles sont faites pour les utilisateurs-programmeurs et non pour les utilisateurs ordinaires. Elles ont été conçues pour permettre l'innovation logicielle. Il suffit de reprendre les différentes règles, droits ou libertés, figurant dans les licences pour le constater. Les libertés de modifier, combiner, redistribuer le code transformé ne peuvent guère intéresser l'utilisateur ordinaire dénué de toute compétence en matière de programmation, à l'inverse de celui qui a la capacité de transformer le code pour en faire tout autre chose.

Droits inscrits dans les licences					
	Exécuter le programme	**Copier et distribuer (en l'état)**	**Modifier, pour son propre usage**	**Modifier, combiner avec un autre code, et redistribuer**	**Clause *copyleft***
Usages, domaines d'utilisation	usage final (tel que)	partage et circulation sans modification	correction de bogues, nouvelles fonctionnalités	développement logiciel	développement logiciel
Forme du logiciel	code compilé (binaire)	code source et binaires	code source	codes sources d'origines différentes	logiciel (code source +...)
Utilisateur non programmeur	oui	oui	sans objet	sans objet	sans objet
Utilisateur-programmeur	oui	oui	oui	oui	oui

Tableau 2 : Les droits (ou libertés) et les différents utilisateurs

Ces deux populations utilisatrices du commun l'utilisent de manière différente. L'une téléchargera les fichiers binaires des logiciels libres, l'autre téléchargera du code source en développement et souvent instable. L'une veut un produit fini, et éventuellement des mises à jour; l'autre veut un code à transformer, un matériau pour l'innovation logicielle. Leurs rôles respectifs dans la vie du commun et l'évolution des ressources partagées diffèrent de la même manière. La population des utilisateurs finaux est fondamentalement passive; elle « consomme » du logiciel libre en le prenant tels qu'il est, dans l'incapacité de le transformer et d'accroître ainsi les ressources du commun. Mais la population des utilisateurs-programmeurs est bien plus active, et c'est cette population qui est à l'origine de l'entrée dans le commun de logiciels libres améliorés, transformés, ou même totalement nouveaux, des logiciels libres produits par des individus isolés ou travaillant en coopération[10]. C'est aussi dans cette population que se développe le sentiment d'appartenir à une communauté spécifique, avec sa propre identité et ses propres valeurs à défendre, celles du partage des ressources logicielles.

L'importance des licences de logiciel libre réside en effet dans le fait qu'elles donnent naissance à ces comportements de partage et de mise en commun du code. Ce code peut ensuite circuler et être ré-utilisé, combiné, et intégré à d'autres travaux de programmation, sans que le titulaire d'origine du copyright puisse intervenir et l'empêcher. On a donc, par cette innovation

10 Il faudrait ajouter ici tous ceux qui interviennent activement dans la production d'un logiciel libre sans travailler le code source; ceux qui s'occupent de la documentation, des fichiers d'aide accompagnant le programme, ceux qui traduisent, etc. Pour donner un exemple, le projet de développement KDE (environnement de bureau utilisé essentiellement dans les systèmes d'exploitation GNU Linux et BSD) a environ 800 contributeurs, dont 300 pour l'équipe de traduction (cf. < http://fr.wikipedia.org/wiki/KDE >).

institutionnelle du *free software*, créé (recréé plutôt) les conditions sociales et juridiques de l'invention collective au sens de Robert Allen.

Rappelons que dans son article de 1983, Allen distingue quatre institutions en matière d'innovation, c'est-à-dire quatre schèmes de comportement récurrent (« *recurent pattern of behavior* ») : (1) Les institutions « *no-profit* », où la recherche est pilotée par les politiques publiques, les objectifs militaires, ou pour elle-même, et où les patents n'ont qu'une importance secondaire, la plupart des résultats entrant directement dans le domaine public; (2) Les firmes avec deux possibilités ici, celle où l'innovation est liée à l'activité de la firme et réutilisée en interne, et celle où la firme dispose d'un service de R&D doté d'un budget spécifique, ce service produisant des patents et vendant les résultats de l'activité de recherche; (3) Les inventeurs indépendants, pour lesquels les patents ont été conçus à l'origine, mais dont le rôle est aujourd'hui complètement marginal; (4) L'invention collective enfin qui suppose plusieurs inventeurs (individus ou firmes) en relation de fait les uns avec les autres, avec deux pré-conditions : un échange libre d'informations sur les nouvelles techniques, ce qui signifie évidemment ni patents, ni secret de fabrication, soit par impossibilité de breveter et de garder secrète l'information, soit par une volonté de partage.

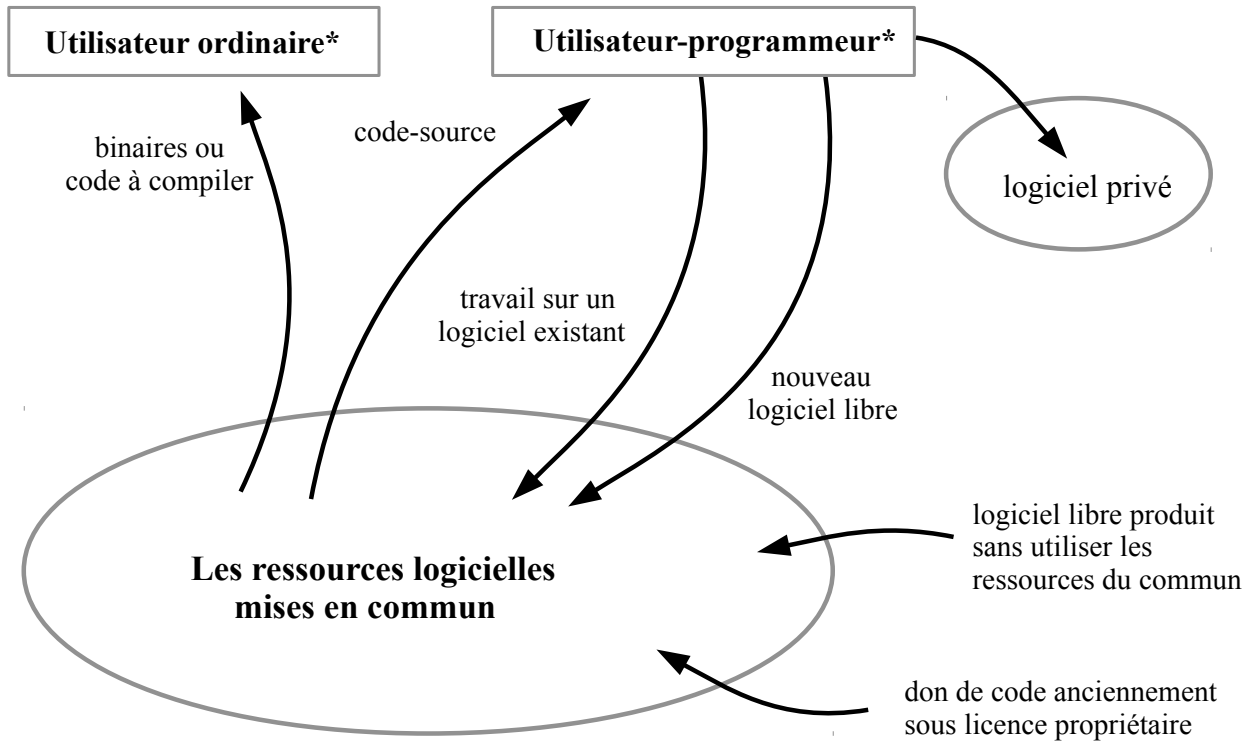
C'est ainsi qu'Allen explique la transformation des hauts-fournaux dans la sidérurgie américaine du XIX^{ème} siècle. L'augmentation de la taille et de la forme (et donc de la capacité) du haut-fourneau permettait en effet d'accroître les performances de celui-ci. Mais cela ne pouvait être ni breveté, ni gardé secret, et il en était de même des résultats obtenus, en termes de productivité, coût, qualité des produits, etc. A chaque construction d'un nouveau haut-fourneau, les sidérurgistes pouvaient donc intégrer le meilleur des inventions des autres, avec comme résultat final une augmentation régulière de la taille et des performances, l'ensemble représentant une trajectoire d'invention collective.

Un commun fait pour l'innovation et l'invention collective

On peut rattacher le commun des logiciels libres à ce concept d'invention collective. Un certain nombre de ressources sont en effet rendues disponibles, et chaque utilisateur-programmeur peut les utiliser pour produire un nouveau logiciel. Si le nouveau logiciel n'est pas distribué, les règles du commun n'imposent pas la publication du nouveau code (même avec une clause *copyleft*)[11]. On a affaire alors à ce que la *Free Software Foundation* appelle du « *logiciel privé* », c'est-à-dire « *un logiciel développé par un utilisateur (individu, organisation ou entreprise) qui le garde pour lui et ne publie ni les fichiers sources, ni les fichiers binaires* ». C'est, dit-elle, une sorte

11 C'est la situation le plus fréquente en matière de production de code. Un très grand nombre de programmeurs travaillent en effet dans des organisations ou des entreprises qui ne distribuent et ne vendent aucun logiciel, comme par exemple les banques, les compagnies d'assurances, les compagnies aériennes, le métro, les universités, etc. On a alors affaire à de l'auto-production dans le périmètre de l'organisation. Le programmeur travaille à maintenir le fonctionnement du système informatique utilisé par l'organisation ou développe un nouveau programme qui ne sera utilisé que par celle-ci, pour les besoins de sa propre activité (et ne sera donc jamais ou très rarement re-distribué).

de logiciel libre « *au sens le plus trivial* », car l'unique utilisateur a tous les droits dessus[12]. Dans cette situation, il n'y a aucun apport au commun, mais rien n'est retranché à celui-ci.



[* individu, organisation, entreprise, etc.]

Schéma 3 : Utilisation et production des ressources, la place des utilisateurs

Si le nouveau logiciel est distribué, c'est-à-dire transmis à d'autres utilisateurs d'une manière ou d'une autre[13], et si la licence d'origine comporte une clause *copyleft*, son code-source doit être fourni au destinataire de la distribution, et celui-ci peut à son tour le fournir à d'autres. Le code-source peut donc circuler, d'un utilisateur-programmeur à l'autre, en fonction évidemment de l'intérêt qu'il présente pour les uns et les autres. Cette circulation informelle entre utilisateurs-programmeurs est portée à un niveau supérieur par la mise en ligne; le code étant posté sur un site web, afin que ceux qui le souhaitent puissent venir eux-mêmes le chercher par téléchargement. Et cette mise en ligne peut être permanente, ce qui est quasiment la règle pour tous les grands projets

12 Cf. « logiciel privé » in < http://www.gnu.org/philosophy/categories.fr.html >.

13 On peut d'ailleurs lire ceci sur le site de la Free Software Foundation : « *En général, nous ne pensons pas qu'il soit mauvais de développer un programme et de ne pas le publier. Il y a des circonstances où un programme est si utile que le garder pour soi est mauvais pour l'humanité. Toutefois, la plupart des programmes ne sont pas si merveilleux que ça, et les garder pour soi n'est pas particulièrement dommageable. Par conséquent, il n'y a pas de conflit entre le développement de logiciel pour un usage privé ou personnalisé et les principes du mouvement du logiciel libre* » (« logiciel privé » in < http://www.gnu.org/philosophy/categories.fr.html >).

de développement. Il existe par ailleurs de grands dépôts (ou forges) qui rassemblent le code-source (versions stables ou en développement) de très nombreux logiciels libres, comme SourceForge, GNU Savannah, Freshmeat, etc. Ces dépôts rendent le code-source accessible à toute la population des programmeurs internautes, quelque soit l'endroit où ceux-ci se situent. En règle générale, ils stockent aussi l'historique des projets et maintiennent des liens avec les équipes de développeurs; ils permettent même de retrouver le code-source de logiciels dont les développeurs ont disparu ou ont abandonné pour une raison ou pour une autre le développement du programme.

On a bien une situation où le code peut circuler librement, les licences donnant de surcroît tous les droits d'usage sur celui-ci. Les conditions de l'invention collective sont donc réunies, donnant naissance à une dynamique particulière de création, d'accumulation, et de partage des savoirs dans le domaine des logiciels, avec ici trois grandes formes d'innovation, trois modes d'évolution des ressources du commun.

(1) Le mode cumulatif, par transformation progressive du même logiciel

A la différence d'un logiciel dont le code-source est caché et inaccessible, le code d'un logiciel libre peut être modifié par les utilisateurs-programmeurs qui le souhaitent, et une fois modifié remis éventuellement en circulation par eux. Cela donne naissance à des ré-écritures successives, par ajout ou suppression de portions du code, dans le but souvent de corriger une erreur (bogue), ou d'améliorer la performance, ou d'introduire de nouvelles fonctionnalités, ou d'adapter le logiciel existant à une transformation de son environnement; le but visé étant toujours ici un changement dans la valeur d'usage. Le logiciel est ainsi modifié de manière incrémentale. Avec un minimum d'organisation de la circulation du code transformé (voir le point suivant, IV), le développement se structure, pris en charge de fait par des utilisateurs-programmeurs différents, parfois nombreux, lesquels travaillent en parallèle ou successivement à l'évolution cumulative du programme. Le logiciel est alors le résultat d'une production collective, avec comme origine différents innovateurs individuels.

(2) Ré-utilisation et combinaison de codes d'origines différentes

Les licences de logiciels libres permettent de ré-utiliser le code-source produit par d'autres. On peut même combiner des codes-sources d'origines différentes. Cette liberté permet la ré-utilisation de composants logiciels existants, lesquels sont déjà développés et souvent bien débogués. La production par combinaison, avec l'ajout d'un nouveau code écrit pour l'occasion, est techniquement facilité aujourd'hui par le caractère systématiquement modulaire de la plupart des logiciels libres, qui fait que l'ensemble du code-source est découpé en sous-ensembles ayant chacun leur fonction, développés et débogués séparément[14]. Ces modules peuvent alors être ré-employés

14 Sur le concept de modularité et les logiciels libres, voir Narduzzo et Rossi (2003). L'approche modulaire a remplacé en génie logiciel une conception plus ancienne, où un logiciel signifiait un seul bloc de code. Le découpage en modules, en blocs indépendants, communiquant par des interfaces standardisées et fixées, a été facilité par

ailleurs, pour la tâche spécifique qu'ils sont capables de remplir. Si les différentes licences sont juridiquement compatibles entre elles, toutes les combinaisons sont possibles[15]. Ce ré-emploi est économiquement une source d'économies non négligeables pour l'innovation logicielle (sur le mode du « financement passé »), car s'il fallait ré-écrire tous les composants nécessaires, il faudrait du temps et un nombre d'heures de travail parfois considérable, sans même évoquer la capacité des développeurs à traiter l'ensemble des problèmes techniques que pose une complète ré-écriture.

On peut donner un exemple de cette possibilité de ré-emploi en listant tous les programmes produits à partir des « moteurs de rendu html », un composant important des navigateurs web qui permet d'afficher correctement tout ce qui est contenu dans le code d'une page web. A la fin des années 1990, la firme *Microsoft* avait obtenu un quasi-monopole en matière d'accès à l'internet avec son navigateur Internet Explorer (plus de 95 % des utilisateurs internautes de l'époque). Aujourd'hui, la situation est complètement différente et le logiciel libre n'y est pas étranger. Il s'est avéré en effet être un puissant vecteur de création de variétés et d'alternatives technologiques, et par là de dé-monopolisation, plus efficace que toutes les procédures antitrust. Il existe aujourd'hui en effet de nombreux navigateurs différents par leur interface, leur ergonomie et les choix techniques des développeurs et leur disponibilité, certains pouvant être installés sur tous les plates-formes et systèmes d'exploitation, d'autres réservés à MSWindows ou à Mac OS X, ou aux systèmes Unix, ou à certains environnements de bureau pour GNU/Linux, comme Gnome, KDE, Xfce, etc.[16]. Derrière cette variété, il y a cependant des éléments communs, dont les moteurs de rendus html, avec actuellement cinq moteurs de rendus, deux propriétaires et trois *open source*[17].

Les moteurs libres sont *Gecko*, *KHTML* et *Webkit*. Le moteur *Gecko* est un produit de la fondation *Mozilla,* publié sous trois licences : la MPL (*Mozilla Public License*), la GNU-GPL et la GNU-LGPL, trois licences qui comportent à des degrés divers une clause *copyleft*, ce qui permet à une grande palette d'utilisateurs d'intégrer Gecko dans des produits libres ou propriétaires, mais dans ce cas la fermeture du code n'affecte pas le moteur, qui avec ces trois licences et son

---

l'évolution des langages, avec l'apparition en particulier du langage C, qui produit systématiquement un code organisé de cette manière. Chaque module peut être développé séparément, sans que ses développeurs aient à se soucier de l'évolution des autres modules, ce qui explique sa quasi-généralisation dans les projets de développement en logiciel libre (voir le point IV).

15 Le code peut aussi intégrer des blocs écrits dans des langages de programmation différents, un compilateur comme le *GNU-Compiler Collection* (en abrégé GCC) par exemple, pouvant compiler ensemble des langages très différents (C, C++, Objective-C, Java, Ada, Fortran).

16 On peut évaluer l'adoption de ces différents navigateurs, en utilisant des logiciels qui relèvent le navigateur utilisé lors d'une connexion à l'internet. On parle alors d'une « part de marché », ce qui ne veut évidemment rien dire, car rien n'est vendu et aucun marché n'existe ici. Il s'agit plutôt de mesurer l'utilisation à un moment donné des navigateurs, beaucoup d'internautes ayant d'ailleurs sur leur machine plusieurs navigateurs qu'ils utilisent alternativement. Les mesures disponibles diffèrent d'une source à l'autre, avec des écarts importants d'une source à l'autre. Elles témoignent cependant d'une situation différente de l'époque où Internet Explorer dominait totalement l'accès à l'internet, son seul rival Netscape étant en voie de disparition. Selon *StatCounter* (août 2012), les parts mondiales étaient respectivement : Internet Explorer : 32,8 %, Chrome : 33,6 %, Firefox : 22,8 %, Safari : 7,3 %, Opera : 1,6 %, autres : 1,7 %.

17 Voir l'Annexe B de cette deuxième partie : tableaux 1 et 2 (moteurs libres), tableau 3 (moteurs propriétaires).

développement *open source*, appartient toujours au commun. On trouve Gecko dans plusieurs navigateurs comme Firefox, Camino, K-Meleon, Epiphany et Kazehakase, et dans d'autres produits, logiciels de messagerie, éditeurs de code html, etc.[18] . Le moteur *KHTML*, sous GNU-GPL, est un développement logiciel lié au projet d'environnement de bureau KDE pour les distributions GNU/Linux. Il est apparu en l'an 2000 dans Konqueror, le navigateur et gestionnaire de fichier de KDE (version 2.0), et ses qualités lui valurent d'être retenu par *Apple* pour son futur navigateur Safari. Les programmeurs de la firme retravaillèrent le code de KHTML (et de KJS) afin d'améliorer l'ensemble et de l'adapter au langage utilisé par l'interface des macs[19] . Les deux développements de KHTML par la communauté KDE et de Webkit par les programmeurs d'Apple, étaient alors complètement séparés (ce qui représente ce qu'on appelle un *fork*, voir ci-après). Du côté de Apple, le développement est donc essentiellement privé, le code ne circulant que dans les limites des frontières de la firme sans publication ni communication à l'extérieur. Cependant après la sortie du navigateur Safari, le code de Webkit a été publié et communiqué à la communauté (sous licence GNU LGPL et BSD), et certaines modifications introduites par des développeurs d'Apple ont été intégrées dans KHTM, améliorant son fonctionnement. Depuis, des variantes de Webkit sont apparues, comme WebkitGTK+ et QtWebkit, ce qui rend plus facile l'intégration de ce moteur dans d'autres environnements de bureau que Mac OS X[20]. Un grand nombre de programmes ont été créés intégrant Webkit, qui tend d'ailleurs aujourd'hui à supplanter Gecko (et KHTML) dans les productions du logiciel libre. On trouve ici des navigateurs, Safari, OmniWeb, Midori, Epiphany, Konqueror (en option), Arora, et GoogleChrome (et sa version *open source* Chromium), et d'autres applications (voir les deux premiers tableaux de l'Annexe B).

(3) Les fourches, les écritures parallèles, et l'exploration tous azimuts

Comme le logiciel libre rompt avec la logique des droits exclusifs, n'importe quel utilisateur (seul ou avec d'autres) peut lancer son propre projet logiciel en reprenant le code-source d'un logiciel en développement. Les développeurs de ce programme ne peuvent s'y opposer. La liberté d'entreprendre est donc totale. Les licences mettent en effet tout le monde, dans ce domaine comme ailleurs, sur un pied d'égalité, économiquement en concurrence, et ceci en permanence. Cette situation offre la possibilité à des programmeurs motivés et compétents de fonder leur propre projet de développement à partir d'un autre. Elle permet de relancer un développement logiciel

18 Voir les détails dans l'Annexe B de ce livrable.

19 Une des limites de KHTML à cette époque était le faible support du javascript, un langage de scripts de plus en plus présent dans les pages web. KHTML avait aussi été conçu pour l'environnement KDE qui utilise le langage et les bibliothèques Qt. Il fallait donc modifier certaines parties du code pour adapter celui-ci au langage de l'interface de Mac OS X, l'objective-C.

20 GTK+ (licence GNU-LGPL) a été développé originellement pour les besoins du logiciel de traitement d'images GIMP (GNU Image Manipulation Program). C'est un ensemble de bibliothèques logicielles permettant de réaliser des interfaces graphiques. Aujourd'hui, ces bibliothèques sont utilisées dans de nombreux projets, dont les environnements de bureau GNOME et Xfce. Le projet KDE utilise la bibliothèque Qt, Mac OS X a son propre langage (l'Objective-C).

abandonné, ou insuffisamment dynamique, ou mal géré par ceux qui en ont la charge. C'est un phénomène nommé « embranchement » (ou « fourche » en anglais *fork*). Le développement du logiciel se scinde alors en deux, avec deux projets parallèles et concurrents, à partir du même code source, avec éventuellement par la suite des emprunts de code d'un projet à l'autre. Ces développements peuvent ensuite diverger progressivement et se séparer définitivement, ou à l'inverse fusionner. Plus fréquemment, l'un des développements finit par supplanter l'autre, qui disparaît faute de contributeurs.

Les raisons des scissions et divisions peuvent être des plus diverses : conflits individuels, divergences de point de vue en matière de choix techniques, insatisfaction devant la conduite d'un projet en cours, ou un enjeu plus stratégique, comme un changement de licence ou d'orientation du développement. Cette grande liberté d'entreprendre à partir d'un code existant et plus généralement de toutes les ressources présentes dans le commun fait que la formation de tels embranchements est fréquente. Donnons quelques exemples récents :

a) En septembre 2010, c'est la création de Mageia, suite de la liquidation judiciaire de *Edge-IT*, une filiale de la société *Mandriva Linux*. Les licenciés économiques ont alors créé une association pour lancer leur propre projet de distribution Linux, en partant du code, sous licence libre, de *Mandriva*.

b) Après le rachat de *SUN Microsystems* par *Oracle,* un projet communautaire ayant comme objectif de poursuivre le développement d'*OpenSolaris*, est apparu. *Oracle* a en effet décidé d'abandonner ce développement *open source,* et en réaction le projet *Illumos* se donne comme objectif de construire un système d'exploitation entièrement libre, compatible à 100% avec tous les logiciels produits pour OpenSolaris.

c) En rachetant *SUN*, *Oracle* a aussi racheté MySQL, un système de gestion de base de données (SGBD) passé sous contrôle de SUN en 2008 avec le rachat de MySQL AB. Ce système, distribué sous une double licence GNU-GPL et propriétaire, est le plus utilisé au monde[21]. En mai 2009, le fondateur de MySQL (Michael Widenius) quitte alors la société pour lancer sa propre version concurrente de MySQL. Cela donne un nouvel embranchement (sous licence GNU-GPL).

d) En septembre 2010, des contributeurs de la suite bureautique OpenOffice.org créent de leur côté *The Document Foundation* afin de poursuivre le développement du programme, rebaptisée alors LibreOffice (sous licence GNU-LGPL). Ils obtiennent le soutien de la *Free Software Foundation*, de *Newell*, *RedHat*, *Canonical* (distribution Ubuntu), *Google*, *Gnome*, *NeoOffice,* et d'un grand nombre de communautés formées autour d'OpenOffice.org. Un autre projet rival est apparu peu après sous le nom d'Apache OpenOffice, abrité par la fondation *Apache* et soutenu par *IBM*. Il existe donc à l'heure actuelle deux projets de développement parallèles et

21 Ce rachat mettait dans le même panier les deux produits concurrents que sont Oracle Database et MySQL. Il a néanmoins été autorisé par la Commission européenne le 21 janvier 2010, et par les autorités antitrust américaines.

concurrents qui partent du même code-source (voir l'Annexe D de ce livrable).

Pour le commun, les fourches sont d'abord des mécanismes de sauvegarde et un moyen puissant de maintenir les règles et les droits existants (en dehors de toute clause *copyleft* et de toute action judiciaire). Quand un développement logiciel pris en charge par une équipe ou une communauté d'utilisateurs-programmeurs subit une dérive qui, de proche en proche, pourrait progressivement transformer le logiciel libre en un logiciel moins libre, voire non libre (par fermeture du code ou difficulté d'accès à celui-ci, changement de licence, etc.), la formation d'une fourche et d'un nouveau développement logiciel clairement ancré du côté du logiciel libre est toujours possible. C'est le sens de la formation de LibreOffice. Dans le commun, il n'y a plus de droits exclusifs sur la technique, plus de monopole à base technique et juridique. Mais le commun est bien plus que cela dans sa dynamique, car les monopoles de fait ou droits particuliers qu'un tel ou un tel pourrait revendiquer sur tel ou tel projet de développement logiciel sont toujours menacés par la formation d'un *fork*.

Mais plus profondément, c'est aussi une grande liberté en matière d'innovation et de développement de nouveaux logiciels, d'expérimentation et d'exploration de différentes alternatives technologiques, quand ces alternatives existent. L'existence des fourches, des développements parallèles et concurrents, permet aux innovateurs d'explorer en parallèle les voies les plus divergentes, et d'expérimenter tous azimuths. Quand par exemple, dans le développement d'un logiciel libre, les développeurs (et les autres contributeurs) rencontre un problème technique qui pose un problème de choix pour l'orientation du travail de programmation, en cas d'opinions divergentes, le jeu reste ouvert, et l'exploration d'une solution dédaignée par une partie des développeurs est toujours possible pour les autres, en formant un f*ork*. Les deux voies seront alors explorées et, les résultats obtenus faisant le tri, les solutions non valables finiront par être rejetées. L'existence de cette liberté de former son propre embranchement permet ainsi d'éviter aux logiciels libres de se retrouver dans une impasse technologique, ce qui pourrait être le cas si une voie (et une seule) avait été explorée, alors même que d'autres étaient envisagées par une partie des utilisateurs-programmeurs, mais laissées de côté. Dans certains domaines, l'innovation est ainsi tout-à-la fois redondante et considérablement diversifiée, avec des composants logiciels nombreux et disponibles, alors même que leur utilisation effective est proche de zéro. Mais quand la situation change, quand l'environnement logiciel ou matériel évolue et pose un problème d'adaptation, le problème a souvent déjà été exploré et les éléments de solution existent. L'arrivée des netbooks n'a ainsi posé aucun problème au logiciel libre[22]; des solutions légères, adaptées à de petites configurations en terme de mémoire étaient en effet déjà présentes, toutes prêtes, dans le commun.

22 Ce qui ne fut pas le cas pour *Microsoft*, dont le système (Vista) trop lourd et trop gourmand de ressources demandait des machines plus rapides, et était donc inadapté. Microsoft a du alors, un peu en catastrophe, pour ne pas perdre ce nouveau marché où les ventes étaient en pleine expansion, ressortir XP dans une version allégée, un produit pourtant abandonné et déclaré officiellement obsolète.

IV . Economie des logiciels libres, production et partage

Dans les licences *open source* ou *free software*, il n'y a rien qui définisse la manière dont on doit produire le logiciel libre. Les licences n'imposent aucune forme sociale particulière de production. Elles donnent tous les droits d'usage à tous les utilisateurs et, avec la clause *copyleft*, créent un cadre juridique (et social) favorable au travail des programmeurs, à l'apparition d'innovations et à l'invention collective. Mais il n'y aucune forme prescrite pour le travail sur le code ou la façon d'organiser le développement d'un logiciel.

On peut donc produire un logiciel libre dans les formes les plus traditionnelles, comme d'autres produisent et vendent des pâtes, des automobiles ou des médicaments génériques; les conditions de distribution (fournir le code source avec tous les droits d'usage) ne s'y opposent pas, la clause *copyleft* non plus. Il n'y a pas non plus d'impossibilité économique à une telle situation, la meilleure preuve étant que de telles entreprises existent aujourd'hui[23]. Le commun a cependant donné naissance à des formes de production plus originales. Pour produire des logiciels de taille importante (tels Apache, Linux, Gnome, Mozilla-Firefox, etc.), on a vu émerger plus ou moins spontanément, sur le mode de la cristallisation et de la boule de neige, des formes nouvelles de production, non seulement par leur mode de financement (le don ou la dépense, et non l'avance d'un capital, dont on espère le retour), mais surtout par la forme prise par l'organisation du travail. Un très grand nombre de programmeurs dispersés sur le plan géographique entrent en relation par l'internet, en envoyant des contributions, en s'échangeant le code en développement, sans relations marchandes entre eux ou subordination à un quelconque pouvoir hiérarchique. Il faudra y revenir. Mais nous pouvons noter tout de suite que cette manière communautaire d'organiser le travail n'est pas la seule manière de produire des logiciels libres. Il y en a d'autres.

Formes de production et modes de financement

On peut en distinguer quatre formes de production, avec des possibilités de combinaison entre ces quatre situations, et des évolutions possibles d'une situation à l'autre pour le développement d'un logiciel particulier.

23 Un grand nombre de *Sociétés de Services en Ingénierie Informatique* (*SSII*) ont clairement opté en effet en faveur des logiciels et construit leur propre modèle économique avec du logiciel libre (la société *Bull* par exemple). Voir Coris, 2002. Nous y reviendrons dans notre point V.

situation	description	activité	mode de financement
(1) Le modèle de la « cave »	Un programmeur ou une toute petite équipe	non-marchande	Dépense (A - DVT)
(2) Le développement communautaire	Un réseau d'utilisateurs de code transformé en communauté	non-marchande	Dépense (A - DVT)
(3) La fondation, gestionnaire d'un projet ciblé	Une organisation du travail combinant salariés et contributions externes	non-marchande	Dépense (A - DVT)
(4) L'entreprise produisant et vendant du code, et des services associés	Une organisation traditionnelle avec salariés et sous-traitants	marchande	Investissement en capital (A – DVT - A')

Tableau 4 : Quatre formes de production en logiciel libre

(1) La production par un individu ou une petite équipe travaillant en relations étroites, ce qu'on peut appeler le « modèle de la cave », en référence à l'étude de Krishnamurty (2002) sur la base de données *Sourceforge,* constatant que la majorité des projets en logiciel libre, actifs et matures (avec existence donc d'une version opérationnelle téléchargeable et téléchargée), avaient été développés par une seule personne sans contributeurs extérieurs (plus de 50 % de l'ensemble des projets présents dans *SourceForge*). D'autres études ont conduit au même constat à partir de données différentes, comme Orbiten (2000)[24]. La production artisanale et solitaire – à l'opposé du modèle du développement communautaire – est donc bien représentée en logiciel libre. On a affaire généralement à une production engagée par un utilisateur pour ses propres besoins, ou par goût de l'innovation, ou par souci d'apprentissage personnel, un travail non commandé réalisé comme un logiciel privé. Ce logiciel, une fois terminé (suffisamment stable), est directement publié et mis dans le commun. Le travail de programmation ne demande d'ailleurs qu'un ordinateur personnel et quelques outils (éditeur de code, débogueur, compilateur, etc.) existant depuis les années 1990 en logiciels libres et disponibles sans frais; il en est de même des langages utilisés. Les dépenses monétaires liées à ce type de travail sont pratiquement nulles, et le seul « coût » est le travail fourni et le temps passé au projet, et évidemment toutes les difficultés que le programmeur va rencontrer, en fonction de sa propre compétence. Un très grand nombre de logiciels libres de petite taille (en termes de lignes de code[25]) sont produits ainsi, aujourd'hui comme hier[26].

24 Dans l'étude de Krishnamurty de 2002, 29 % des projets seulement impliquaient plus de 5 développeurs et concentraient la plus grande activité en termes de discussions. Le grand projet logiciel mobilisant un grand nombre de développeurs n'est donc pas la forme dominante de production en logiciel libre, même si c'est la forme la plus singulière. On verra d'ailleurs que, au sein même de ces projets, le modèle de la cave s'impose, module par module.

25 C'est une mesure conventionnelle de l'importance (et du travail) que représente différents logiciels ou projets de développement. Voir l'Annexe D (étude FLOSS de 2006).

26 Avec la modularité, et l'existence de composants et de blocs de code-source ré-utilisables, et les langages de haut

(2) Le développement communautaire, qu'on pourrait aussi appeler le modèle du « chaudron magique », en référence à l'article d'Eric Raymond (1999) sur Linux, Sendmail, et les projets *open source* impliquant un grand nombre de programmeurs travaillant en coopération en dehors de toute organisation formelle. A la base, il y a une masse de code importante et de nombreux problèmes techniques différents, ce qui représente une masse de travail que le modèle de la cave peut difficilement assumer. Un réseau ou une communauté se substitue alors au programmeur individuel. Les licences libres ont rendu possible cette forme de production. Chaque auteur renonce formellement à contrôler ce que deviendra sa contribution, et ce renoncement élimine à l'avance tout conflit sur les usages de « son » code (dont il peut garder formellement le copyright). Les relations qui se nouent sont alors foncièrement symétriques et égalitaires, car tous ont les mêmes droits sur le code source qu'ils transforment. Un grand nombre de développeurs indépendants peuvent donc travailler sur le même code-source, lequel évolue (et en principe s'améliore) en intégrant les contributions des uns et des autres. Ici, il n'y a pas d'échanges marchands et personne n'est payé à la fourniture de son code. La plupart des contributeurs réalisent d'ailleurs ce travail sur leur propre temps libre. Certains sont cependant payés aujourd'hui par des fondations ou des entreprises pour contribuer au projet communautaire; mais ce qui domine, aujourd'hui comme hier, est le don de code, et le don de son temps et de son temps de travail[27]. Le but pour chacun est d'ailleurs la valeur d'usage, le code lui-même et sa transformation. Les contributeurs sont aussi utilisateurs du logiciel et directement intéressés à voir celui-ci s'améliorer. On a donc bien une activité qui globalement comme individuellement relève en termes de financement de la dépense et non de la formule générale du capital[28].

(3) Les fondations, des organisations à but non lucratif, ont été créées pour réunir les fonds nécessaires à la mise sur pied d'un ou plusieurs projets de développement logiciel. Ces fonds servent à payer des programmeurs comme salariés pour qu'ils se consacrent de manière continue et à temps plein au développement d'un logiciel, en collectant des contributions externes comme

---

niveau actuel, certains programmes importants peuvent être pris en charge par des individus ou de toutes petites équipes travaillant dans leur coin. On a des exemples de cette possibilité dans l'Annexe B sur les moteurs de rendus html.

27 Une enquête ancienne, WIDI (Roblès et alii., 2001), montrait ainsi que les développeurs payés pour écrire ou développer du logiciel libre étaient largement minoritaires, 16 % de l'ensemble des contributeurs; 46 % d'entre eux par contre déclarait n'en tirer aucun gain monétaire ni direct, ni indirect. La plupart d'entre eux d'ailleurs n'y consacrant que moins de 5 h par semaine. L'étude FLOSS de 2006 (voir Annexe D) après examen du code et des crédits d'une base de données très large de logiciels libres donne comme proportions : Deux tiers écrit par des individus, 15 % venant des firmes et 20 % venant d'autres institutions (fondations, universités, etc.). Sur certains projets logiciels (Mozilla.org ou même Linux), la part des salariés payés pour produire du logiciel libre est sans doute aujourd'hui bien plus importante; voir : < http://www.zdnet.fr/actualites/open-source-75-des-contributeurs-de-linux-sont-remuneres-39712463.htm > et « Who wrote 2.6.20 ? » < http://lwn.net/Articles/222773/ >.

28 Le nombre d'heures de travail que peuvent représenter certains de ces projets communautaires est considérable. Des études ont calculé ce « coût » à partir du nombre de lignes de code, en utilisant un modèle de calcul des coûts de développement pour une firme produisant de manière traditionnelle du logiciel propriétaire. On obtient alors des sommes impressionnantes, mais évidemment, ces sommes n'ont jamais été réunies, ni réellement dépensées (voir l'Annexe D et Wheeler (2001), Gonzalez-Barahona et alii. (2002), Ghosh et alii. (2006)).

dans le mode communautaire. Le but de la fondation est aussi la gestion du projet lui-même, l'orientation éventuelle de son développement, et la prise en charge de l'organisation de la diffusion du programme. Le logiciel est alors le résultat d'une coopération associant un groupe de développeurs salariés, payés par la fondation, et d'un cercle beaucoup plus large de contributeurs externes signalant les défauts (bogues) et travaillant à leur correction. Ils sont là encore à la source de nombreuses innovations et donnent leur code et leur travail sans rémunération. En dehors de la *Free Software Foundation* qui produit du code libre depuis les années 1980, les fondations les plus connues sont les fondations *Apache*, dont la raison d'être première est le développement du serveur web du même nom, et *Mozilla.org* créée après la « libération du code » du navigateur de *Netscape* en 1998, mais aujourd'hui la plupart des projets de développement sont dotés de leur propre fondation, afin de pouvoir couvrir leurs besoins de manière autonome. Dans le modèle de la cave comme dans le modèle communautaire, la logique financière est celle de la dépense.

(4) Les entreprises produisant et vendant du logiciel libre et/ou des services associés ou contribuant à des développements *open source*. On trouve deux cas de figure ici. Premier cas, l'entreprise (ou toute autre organisation) fait des dons de code, ou apporte des fonds, ou paye des salariés pour qu'ils participent à un développement communautaire indépendant de la firme sur un pied d'égalité avec les autres contributeurs. L'autre cas est celui des entreprises qui produisent seules ou avec d'autres du logiciel libre, ou encadrent et orientent un développement communautaire lié à elles, et plus ou moins sous tutelle. Le logiciel est intégré à l'offre de la firme, et peut être vendu dans le cadre d'une relation de service. Il s'agit souvent d'un code produit sur commande et adapté à des besoins particuliers, ou lié à un matériel spécifique produit par l'entreprise. Le code source est bien distribué avec sa licence et les droits d'usage du logiciel libre, mais sa circulation est alors la plupart du temps limitée au cadre des relations commerciales que la firme noue avec ces clients. Ici la production du logiciel est intégrée à la formule générale du capital. Les décisions de poursuivre ou non le développement étant déterminées par les dirigeants de l'entreprise, en fonction de calculs de rentabilité, ou en relation avec des demandes précises passant par le marché et le service commercial. Alors que dans le premier cas, on a plutôt affaire à de la subvention ou à un financement croisé.

On voit que les trois premières situations diffèrent de la dernière. Elles diffèrent par le mode de financement et l'importance du non-marchand. C'est particulièrement net pour les deux premiers cas, deux situations où il n'y a - au niveau du développement du logiciel - ni salariat ni échange marchand. Il y a cependant des individus qui s'engagent dans ces développements et fournissent leur travail et leur code. Il y a aussi beaucoup d'échanges, de code ou d'autres choses (informations, etc.). C'est un des traits majeurs de l'économie des logiciels libres[29], elle est très

29 Précisons que par « économie », j'entends ce qu'entendait Adam Smith, c'est-à-dire la création, la circulation, et l'utilisation des « richesses », autrement dit des valeurs d'usage (« *nécessités et commodités de la vie* »), et non la gestion de ressources rares ou l'économie monétaire, ces deux approches étant par trop limitées pour notre objet et

largement située dans la sphère non-marchande. Il y a bien des activités marchandes qui utilisent les logiciels libres, et de plus en plus, mais ces activités sont séparées des développements logiciels alimentant le commun, et en quelque sorte construites à la périphérie de ces développements.

Une autre différence est le fait que la production est directement prise en charge par des utilisateurs, et des utilisateurs-programmeurs. Dans les industries produisant des biens manufacturés, comme l'industrie automobile par exemple, production et utilisation sont deux activités nettement séparées. C'est au niveau de la production que le produit est défini, c'est là qu'apparaissent les innovations, et les demandes des utilisateurs automobilistes n'interviennent que filtrées par les services marketing de l'entreprise. Dans l'auto-production logiciel comme dans le développement communautaire des logiciels libres, ce sont à l'inverse les utilisateurs qui prennent directement en charge la production. Ce sont eux qui sont à la source des innovations et assurent le développement et la maintenance, dans un aller-et-retour continuel entre l'usage et la transformation du code, ce qui donne d'ailleurs un mode de production parfaitement adapté à la nature du logiciel comme produit toujours imparfait, imposant une période prolongée de test et de correction des défauts (voir Annexe A, première partie). C'est dans le cadre des licences (et des règles du commun) que cette situation particulière est née, une situation qui explique les grands projets de développement communautaire.

Emergence des projets communautaires

On peut analyser les formes de production communautaire en logiciel libre comme des exemples particuliers de « *réseaux horizontaux d'innovation par et pour les utilisateurs* » (Von Hippel, 2002[30]), en désignant ainsi une situation où les utilisateurs d'une technique sont producteurs des innovations et à la source de la transformation de la technique, l'évolution de celle-ci étant directement pilotée par l'usage. Les utilisateurs d'un logiciel sont toujours intéressés à voir le logiciel qu'ils utilisent s'améliorer. Quand ils sont programmeurs et disposent du code avec le droit de le transformer (ce qui est le cas pour les logiciels libres), ils sont aussi naturellement intéressés à le faire, pour en obtenir une meilleure valeur d'usage. Ils produisent alors un nouveau code. Faire circuler ce coden'est guère problématique dans le cadre des licences ; et par ailleurs, cacher cette innovation ne rapporte rien à l'innovateur et la révéler ne l'appauvrit pas plus. La diffusion par contre peut permettre à d'autres d'innover à leur tour, avec au final un logiciel amélioré dont tous pourront bénéficier, y compris ceux qui ont révélé leurs innovations. On comprend alors pourquoi certains projets de développement en logiciel libre ont émergé aussi vite avec l'apparition d'une dynamique de coopération et d'invention collective.

sans intérêt heuristique, puisqu'un grand nombre d'activités de production et d'échange de logiciels s'effectuent en dehors de toute relation marchande, et qu'un très grand nombre de produits sont copiables à l'infini , ce qui rend assez vain le concept même de la « rareté ».

30 Voir l'Annexe E de ce livrable, pour une présentation plus détaillée de cette notion.

Ces réseaux d'utilisateurs-innovateurs sont des situations de fait, dont l'unité peut sembler fortuite, reposant essentiellement sur l'unité de la chose dont il est fait usage, l'existence d'une structure de communication entre utilisateurs, et l'existence d'une population d'utilisateurs capables de produire des innovations. Mais cette situation de fait peut se transformer, comme le remarque Von Hippel, en « communauté », ce qui est presque systématiquement le cas pour les développements en logiciel libre[31]. Jusqu'ici, nous avons utilisé le terme « communauté » sans précaution particulière, nous devons maintenant en préciser le sens. La communauté d'utilisateurs diffère du réseau d'utilisateurs par le fait que les utilisateurs ont conscience d'appartenir à un groupe qui existe en tant que tel avec ses intérêts propres. Il y a donc naissance d'un sentiment d'appartenance et d'identification à un projet, aux règles du projet, au groupe lui-même. On peut d'ailleurs distinguer dans les communautés du logiciel libre, les communautés en quelque sorte élémentaires, nées d'une même situation d'utilisation d'un même programme, ou d'une même participation à un développement logiciel (qui conduit les utilisateurs de ce logiciel à le défendre, y compris éventuellement contre un autre projet concurrent en logiciel libre), et la communauté plus large des partisans du logiciel libre attachée à défendre et renforcer le commun et ses règles, dans un contexte juridique et économique dominé par le système des droits exclusifs. Ce point différencie sans doute les communautés du libre d'autres communautés d'utilisateurs dans d'autres domaines.

La formation de réseaux et communautés d'utilisateurs n'est pas une exclusivité du logiciel libre, et la transformation de ces réseaux en une force agissante, productrice de ressources, non plus. Cette transformation des utilisateurs en producteurs existe aussi dans l'univers MSWindows (et l'univers de la firme *Apple*). Les utilisateurs du système d'exploitation de Microsoft peuvent en effet trouver en ligne des tutoriaux, de l'aide, des forums, des logiciels, et bien d'autres services, toute cette production et diffusion n'étant pas le fait de *Microsoft*, mais le fait d'autres utilisateurs producteurs de solutions aux problèmes que rencontrent tous les utilisateurs des logiciels de Microsoft. La différence ici entre l'univers des logiciels propriétaires et le monde des logiciels libres est le fait que les licences de logiciel libre et le commun rendent possibles et sécurisent un travail commun sur le code, ce qui n'est pas possible quand le code est caché et les droits réservés. Ce sont les normes de partage et de libre circulation du code qui permettent, quand la situation le demande, c'est-à-dire essentiellement quand le défi technique et l'importance du travail à accomplir impliquent la coopération d'un grand nombre d'individus, l'émergence d'une communauté d'utilisateurs producteurs du logiciel. Ces développements communautaires ont alors

31 « *…user innovation networks mays, but need not, incorporate the qualities of user « communities » for participants, where these are defined as « …networks of interpersonal ties that provide sociability, support, information, a sens of belonging, ans social identity. » (Wellman, 2002).* (…) *In contrast, those who provide or download, for example, novel user-developed statistical techniques to or from a web page such as the Boston Archive may simply view and use that user netwark as a simple tool rather than a community* » (Von Hippel, 2002, 24).

comme caractéristique de commencer à un certain moment du temps mais de n'avoir aucune fin programmée, de ne connaître par ailleurs ni délais, ni rythme bien défini, et d'évoluer plutôt en fonction de l'importance des usages, de la popularité du programme, et de l'importance des contributions issues de la communauté des utilisateurs.

Le développement initial du système d'exploitation Linux est un bon exemple d'une telle émergence. En 1991, un étudiant finlandais en informatique, Linus Torvalds, décide (un peu par jeu et pour apprendre) d'écrire un noyau de système d'exploitation aux normes POSIX utilisant le système de fichiers de Minix, le système d'exploitation écrit par Andrew Tannenbaum pour les besoins de l'enseignement et de la démonstration. Le 5 octobre 1991, Linus Torvalds annonce cette ébauche en la postant sur le forum des utilisateurs de Minix en lançant un appel aux contributeurs. Le code est accompagné d'une licence permissive rédigée par lui (imposant en particulier la gratuité). Le projet démarre alors petitement. En 1992, il se dote d'un nom (Linux), d'une mascotte (le pingouin) et d'une nouvelle licence, la GNU-GPL[32]. A cette époque, il y avait une forte demande pour un système à base Unix pour les micro-ordinateurs, comme alternative à MSWindows. Il y avait aussi une communauté d'utilisateurs de Minix frustrée de l'absence de développement de ce système. Ces deux éléments expliquent l'augmentation du nombre de contributeurs (158 auteurs recensés en 1994), par un phénomène de cristallisation autour du projet puis de boule de neige (618 auteurs recensés en 1997, 2263 en 2002 (Gosh et David, 2003)) à mesure que le logiciel s'améliore et voit sa popularité augmenter. En effet, dans ce mode de développement, plus il y a d'utilisateurs, plus il y aura de contributeurs effectifs, et plus le logiciel s'améliorera, ce qui renforce son utilisation.

Structure et organisation du projet communautaire

A l'origine d'un développement communautaire, on trouve toujours trois éléments : (1) un problème logiciel à résoudre suffisamment important pour donner lieu à la contribution d'un certain nombre de programmeurs, (2) un fondateur (une personne, une équipe, une fondation, voire une entreprise) pour prendre en charge la maintenance du projet, (3) un paquet de code, instable en général et plein de défauts, lequel représente une première matérialisation du futur programme. Ce code, d'autres utilisateurs peuvent le tester, le transformer, l'améliorer. Le projet est donc lancé par un appel à la communauté virtuelle des programmeurs, une communauté dont le développement de l'internet a reculé les limites au monde entier. Cette communauté peut alors se mobiliser, au coup par coup, ou de manière plus systématique, en fonction des problèmes et de l'intérêt que suscite le logiciel. L'activité s'organise alors sans organisateur apparent. Il n'y a en principe ni design prédéfini, ni délais, et chacun semble faire ce qu'il veut, quand il veut. Et pourtant, tout un ensemble de programmeurs coopèrent pour produire du code, de la

32 « *Protéger Linux par la GPL est la meilleure décision que j'aie jamais prise* » (Linus Torvalds in Yamagata, 1997).

documentation, des supports techniques, et le résultat est bien un produit commun, un logiciel utilisable. Il faut donc s'interroger sur l'organisation du travail dans de tels projets. Comment des utilisateurs-programmeurs dispersés géographiquement, travaillant chacun dans leur coin, peuvent-ils réussir à se coordonner de manière somme toute efficace ?

Les projets de développement communautaire sont assez loin de l'image véhiculée par le texte d'Eric Raymond de 1998 (« la Cathédrale et le Bazar »), où le travail sur le code est le fait d'une communauté de pairs en interaction, se contrôlant mutuellement dans une sorte de transparence totale. Les projets qui se sont effectivement développés (comme Linux), sont plutôt caractérisés par l'émergence d'une structure et d'une administration pratique du développement, ce qui veut dire apparition d'une division du travail horizontale et verticale, avec répartition des rôles et mise en place de procédures de coordination. Les différents contributeurs ne sont alors ni strictement égaux, ni interchangeables. Certains ont de fait un rôle plus important que d'autres et influent par leurs opinions et leurs décisions sur le devenir du projet, même s'ils restent dépourvus de tout moyen pratique de contraindre les autres à accepter leurs décisions. Leur pouvoir est de plus largement limité par les contraintes techniques et leur autorité repose essentiellement sur la connaissance du code et l'expérience en matière de programmation, car la plupart des divergences et décisions portent sur des questions techniques, et comme le dit Linus Thorvald : « *Ce qui est agréable avec la technique, c'est que les problèmes techniques trouvent toujours une solution, alors que des questions non techniques n'ont parfois aucune réponse* » (in Yamagata, 1997). Ils sont de plus toujours, en cas de divergences graves, sous la menace du *fork*.

La coordination des activités est réalisée par plusieurs dispositifs : l'usage systématique de la modularité, une administration du projet exercée par un (ou des) mainteneur(s), et un grand nombre d'outils utilisant l'internet (CVS[33], forums, mails, système de suivi du traitement des bogues, etc.), qui permettent l'organisation routinière du travail d'ensemble.

La modularité n'est pas l'apanage des logiciels libres; elle existe dans le monde des logiciels propriétaires. Elle est cependant presque toujours présente dans les développements du logiciel libre, et par certains côtés, c'est une condition nécessaire pour qu'un développement s'appuyant sur une communauté de programmeurs existe[34]. Avec la modularité dans le domaine des logiciels, on contourne la « loi de Brooks » (1996), du nom de celui qui, le premier, a constaté qu'ajouter des

33 CVS (*Current Version System*) est un système de gestion de versions concurrentes qui aide les sources à converger vers la même destination. Il reçoit et stocke les différentes contributions, le code accompagné par la documentation, les informations sur l'auteur, etc. Son répertoire permet de retrouver les différentes versions et leur historique, et rend le code disponible pour toute la « communauté ». C'est un système centralisé (avec un serveur central). Il existe aussi des systèmes de gestion de versions décentralisés, comme Git créé par Linus Torvalds, Bazaar (de *Canonical*), etc.

34 Pour mettre réellement sur pied un développement *open source* à partir du code-source « libéré » de Netscape Navigator (1998), il a fallu découper le code original en modules, afin de créer des conditions techniques permettant à des développeurs externes de travailler sur le code; la gestion de l'ensemble étant assuré par un *staff*, composé d'ingénieurs payés par différentes entreprises (Netscape, Redhat, Oeone, Sun, IBM, etc.) (Voir Reis et alii., 2002).

programmeurs à un projet qui prend du retard fait chuter brutalement la productivité de l'ensemble de l'équipe[35]. Brooks expliquait cet apparent paradoxe par l'augmentation des problèmes de communication et coordination entre programmeurs, dans une activité où une action individuelle (écriture d'une ligne de code) pouvait interférer avec toutes les autres écritures présentes dans le code. L'approche modulaire du logiciel consiste alors à découper le code-source en différents modules (de taille inférieure en terme de lignes de code), lesquels sont reliés par des interfaces. Les modules sont alors traités comme des boîtes noires et la seule information révélée à tous est l'interface (Narduzzo et Rossi, 2003). Chaque module peut alors être développé séparément sans qu'il soit nécessaire pour ses développeurs de se coordonner avec ceux qui interviennent sur des modules différents. Il suffit de respecter rigoureusement les interfaces. La modularité rend alors possible l'établissement d'une division horizontale du travail et une spécialisation poussée de chacun, car la cohérence d'ensemble est assurée par le respect des interfaces communes et l'architecture générale du système, plus l'utilisation d'éléments communs, comme les bibliothèques.

Un programme aussi important que Linux, 3 millions de lignes de code et 2300 développeurs identifiés en 1993, repose sur une telle architecture modulaire, le nombre de modules augmentant d'ailleurs d'une version de Linux à l'autre (Gosh et David, 2003). Il n'y a ici que quelques composants intégrés larges, comme le module de gestion de la mémoire (mm) ou le noyau lui-même (*kernel module*), les autres sont de taille bien inférieure. Les modules sont développés alors par des équipes réduites (souvent un programmeur isolé), et très nombreux sont les auteurs qui ne travaillent que sur quelques modules; plus de 70 % des auteurs de Linux n'ont travaillé que sur un seul module (Gosh et David, 2003); et un programmeur seul peut facilement accéder à la compréhension d'un module tout entier. Contrairement alors à ce que véhicule l'image du « bazar », ce n'est pas l'intensité des communications dans une communauté indistincte, mais bien plutôt la suppression de tout besoin de communication qui fait la force des développements *open source*. C'est aussi ce qui rend possible le travail géographiquement distribué, et la mobilisation d'un très grand nombre de programmeurs en même temps, sans poser de problème particulier de coordination.

Le deuxième trait caractéristique des projets *open source* est l'existence d'une division verticale du travail, avec une stratification marquée de la communauté des contributeurs. L'intensité des contributions est en effet très variable et les rôles différenciés. L'étude de Mockus, Fielding et Herbsleb (2000) sur Apache est caractéristique : 15 % des programmeurs impliqués dans le projet avaient écrit 80 à 85 % du code. Et si plus de 3000 personnes avaient bien travaillé sur le logiciel, c'est en fait une toute petite équipe qui en avait assuré presque tout le

35 Brooks (1996) constate que le travail d'un programmeur pendant un an donne un résultat bien supérieur au travail d'une équipe de douze programmeurs pendant un mois. Et pourtant, « 1 homme x 12 mois = 12 hommes x 1 mois » ! Mais l'équipe est ici moins efficace que l'individu !

développement, les autres ne contribuant que de manière secondaire, essentiellement à la correction des bogues, cette activité étant par contre très largement distribuée. Il en est de même pour bien d'autres projets *open source*.

Le schéma général est alors celui d'un noyau de programmeurs relativement stable, qui contrôle le développement du code au niveau des modules, et intègre des contributions ponctuelles issues d'un groupe plus large de développeurs, entouré à son tour par ceux qui signalent ou fixent les défauts (bogues), avec enfin la communauté beaucoup plus large des utilisateurs ordinaires des versions (stabilisées) du logiciel[36] (Le même individu peut évidemment appartenir à plusieurs de ces catégories). Dans les projets de grande taille, la structure ci-dessus conduit à instaurer progressivement des formes organisationnelles et des outils spécifiques qui séparent clairement les rôles[37]. La transformation du logiciel repose toujours sur un principe de contributions libres, mais l'intégration de ces contributions à l'ensemble du code n'est pas automatique. Il y a toujours un filtrage assuré par les mainteneurs responsables de tel ou tel module. Ainsi, dans le projet Mozilla (Reis et alii., 2002), l'objectif est d'avoir à un moment donné une seule version du code, et un filtrage hautement formalisé assure cet objectif avec des outils spécifiques, comme Bugzilla par exemple[38]. Ce logiciel enregistre les demandes de modification appelées *bug* (correction d'un défaut, amélioration ou ajout d'une nouvelle fonctionnalité) et leur traitement en temps réel. C'est donc un ensemble de fichiers, accompagnés d'un statut (non-confirmé, nouveau, attribué, résolu, ré-ouvert, vérifié, clos), ce qui permet à chacun de savoir où en est le traitement du *bug*, et en fait un outil de communication et de coordination idéal pour les différents développeurs travaillant sur le projet. Une fois enregistré, le *bug* est l'objet d'une première revue (ligne à ligne) et éventuellement d'une super-revue pour les introductions les plus problématiques, effectuée dans le cas de Mozilla par un ingénieur plus expérimenté et plus familier du code-source.

La même structure a progressivement émergé au cours du développement de Linux (Kuwabara, 2000; Moon et Sproull, 2000), et Linux apparaît même comme hautement hiérarchisé, puisqu'en principe toute intégration de nouveaux *patchs* doit être soumise à Linus Torvalds qui accepte, demande des modifications, ou rejette la contribution. Mais au fil du temps, une liste de

36 Quand le programme est utilisé par un très large public et comporte une interface graphique, comme par exemple la suite bureautique OpenOffice.org (aujourd'hui LibreOffice ou Apache OpenOffice), à cette communauté des développeurs-programmeurs, il faut ajouter d'autres types de développeurs (dont certains d'ailleurs ne verront jamais une seule ligne de code), comme des dessinateurs, des traducteurs, des producteurs d'aide, etc. Autour d'OpenOffice.org, se sont en effet constituées des communautés linguistiques importantes dont certaines très actives, comme la communauté francophone. Voir aussi la note 10.

37 L'ensemble du projet Apache est géré centralement par une équipe de mainteneurs qui décide de manière collégiale (en votant si nécessaire) de toute inclusion ou changement dans le code. Cette équipe, dont la taille a augmenté au fil du temps, est composée de volontaires cooptés, chacun étant responsable d'un ensemble de modules.

38 A l'origine Bugzilla était utilisé en interne par *Netscape* pour suivre le développement de Netscape Navigator. Depuis 1998, il est développé par la fondation Mozilla et est utilisé par de nombreuses autres organisations pour suivre le développement des logiciels, sur l'internet ou dans des réseaux privés. Les plus connues sont la Mozilla Foundation, Facebook, la NASA, YAHOO, GNOME, KDE, Red Hat, Novell et Mandriva.

mainteneurs responsables de tel ou tel module est apparue; et ce sont eux qui assurent le filtrage et définissent les nouveaux *patchs*. Certains sont même devenus complètement autonomes dans leur domaine, même si Linus Torvalds s'occupe toujours du développement du noyau central du système (le module *kernel*). Il annonce aussi toujours formellement les réalisations stables[39]; mais par ailleurs, plusieurs versions plus ou moins stables ou instables peuvent être développées en parallèle.

On le voit, la licence retenue fixe un certain nombre de règles, mais elle ne fixe ni les formes organisationnelles, ni les routines nécessaires au développement. La modularité permet un travail dispersé (et solitaire), mais l'intégration des différentes contributions reste toujours un processus délicat. Pour assurer la qualité du code, un filtrage est toujours nécessaire. Le projet *open source* repose donc fondamentalement sur un système de murs filtrants : les contributions à l'entrée sont triées et examinées avant d'être inscrites dans le code du logiciel, alors que la sortie est la plus ouverte possible, n'importe qui pouvant venir télécharger librement le code des différentes versions en développement. Le développement communautaire est une sorte de processus évolutionniste organisé, combinant accumulation et sélection. Les changements dans le code-source se font par une succession d'ajouts et de suppressions de lignes de code; et celui-ci change alors de manière incrémentale en fonction des problèmes rencontrés par les utilisateurs; il n'est donc pas nécessaire d'établir à l'avance un plan de travail général. Les pratiques de filtrage et l'intervention des mainteneurs suffisent à conserver et améliorer la qualité du code, et la modularité assure la cohérence d'ensemble.

Un monde anarchique, sans anomie

Une des spécificités du commun des logiciels libres, parmi tous les communs existants, est d'avoir donné naissance à des formes de production prises en charge par ses utilisateurs. Les ressources logicielles partagées génèrent d'autres ressources partagées, et le commun évolue en s'adaptant aux demandes de ses utilisateurs et aux transformations des environnements informatiques. Une économie du logiciel libre complexe et diversifiée est ainsi apparue depuis les années 1980, une économie qui a la particularité d'être et de rester pour l'essentiel en marge de la sphère marchande, les droits de copie et de libre circulation faisant, à cause de la forme numérique des fichiers, de la gratuité une obligation quasi-générale. Le financement de cette économie est alors assuré par des dons (code ou dépenses monétaires) et un ensemble de contribution volontaires non rétribuées; et les entreprises intéressées au développement de certains logiciels libres intègrent elles-mêmes ce schéma de la dépense à fonds perdus, où le seul retour espéré est

39 Voir la liste des mainteneurs sur < http://lxr.linux.no/#linux+v3.6.6/MAINTAINERS >. C'est une sorte de « *dictateur bienveillant* » (dixit Wikipedia), une autorité en termes de choix techniques et organisationnels. Les différentes versions du noyau publiées par lui s'appellent « mainline » ou « vanilla ». Ce sont ces noyaux vanilla qui sont intégrés par les distributeurs, avec l'addition de quelques patchs de sécurité, de corrections de bogue ou d'optimisation, produits par d'autres; et il existe alors de nombreuses versions concurrentes...

une certaine valeur d'usage.

Type de ressource		exemples
Don direct d'un logiciel	en nature	« logiciel privé » publié comme logiciel libre
Don de code	en nature	libération d'un code-source anciennement sous licence propriétaire, qui devient le point de départ d'un nouveau développement en logiciel libre
Ré-utilisation du code appartenant déjà au commun	en nature	de droit dans toute licence d logiciel libre (le code étant disponible dans différentes forges)
Don d'heures de travail consacrées au développement	en nature	travail des mainteneurs et contributeurs externes dans les projets communautaires
Salaires versés par des entreprises ou des fondations à des employés développant un logiciel libre	en nature et monétaire	mainteneurs et contributeurs salariés dans certains projets *open source*
Apports monétaires de particuliers ou d'entreprises	monétaire	collectés en particulier par les fondations

Tableau 5 : Les ressources de l'économie des logiciels libres

Une autre spécificité de ce commun est l'absence de toute instance ou organisation centrale ayant comme fonction la direction, la gestion ou la régulation du commun. Ce commun représente une forme d'anarchie au sens premier du terme, c'est-à-dire un monde sans hiérarchie instituée, sans commandement central, sans roi, sans chef, sans autorité unique. Il existe bien ça et là des organisations, des associations, comme la *Free Software Foundation, l'Open Source Initiative*, la *Fondation Apache*, etc., qui par leurs actions influent sur l'existence et l'évolution du commun. Elles déploient en effet une activité spécifique de définition et de défense des règles d'ouverture et de partage du code (dans des formulations qui peuvent d'ailleurs différer l'une de l'autre)[40]. Elles contribuent à orienter les efforts de la communauté dans telle ou telle direction. Mais elles sont plurielles, et souvent en rivalité les unes avec les autres. Aucune d'entre elles ne représente l'autorité unique, et aucune ne peut y prétendre, n'ayant aucun moyen d'imposer cette autorité. C'est plutôt la communauté et plus exactement les membres de la communauté qui, seuls ou en coalition, gèrent, régulent et font évoluer le commun, en adoptant tel ou tel ensemble de règles et de comportements, en se mobilisant ou non sur tel ou tel projet, sur telle ou telle tâche, sur telle ou telle cause, etc.

Il en est de même au niveau micro, au niveau des différents projets de développement logiciel. Les formes d'organisation et d'administration sont ici extrêmement variées. Il y a – cas très

40 La Free Software Fondation par exemple fait évoluer ses licences pour tenir compte des transformations dans l'utilisation des logiciels, et contrer donc certaines pratiques qui à ses yeux menace le logiciel libre. C'est le sens des changements introduits dans la GNU-GPL entre la version 2 et la version 3.

fréquent – le projet mené et géré par un fondateur (ou une toute petite équipe), qui produit une grande partie du code, au moins au départ, et oriente ensuite le développement du logiciel comme il l'entend, en intégrant ou non des contributions externes. C'est le modèle du dictateur supposé « bienveillant »; mais il y a aussi le modèle « méritocratique » et le modèle « démocratique » qui servent de référence à la plupart des grands projets communautaires.

Pour définir le modèle méritocratique, on peut prendre l'exemple de la *Fondation Apache* (c'est « d'*Apache way* »[41]). Le projet de développement du serveur web a démarré historiquement sans fondateur; c'est un petit groupe qui était à l'initiative du projet, une petite équipe de développeurs entourée de contributeurs externes. Quand le projet s'est structuré, pour assurer le renouvellement des développeurs d'origine, le principe retenu fut le « *gouvernement par le mérite* », un mérite apprécié en fonction de la participation au développement en termes de lignes de code. Le principe de sélection des individus est alors le même que celui qu'on applique au code : une grande ouverture à l'entrée suivi d'un filtrage, dans des formes extrêmement codifiées. Un *Board of Directors* (9 personnes élues) est chargé de gérer les affaires communes de l'*Apache Software Foundation*, essentiellement les questions juridiques ou les rapports avec l'industrie. Les projets logiciels abrités par la fondation sont gouvernés par des *Project Management Committee* (*PMC*), établis par le *Board* et éventuellement supprimés à tout moment par lui. Dans ces *PMC*, l'*Apache way* distingue et hiérarchise les rôles, en attribuant à chaque rôle des droits différents sur le développement. Il y a le simple utilisateur, le développeur (ou contributeur) externe, puis le *committer*. Ce *committer* est un développeur qui a le droit d'écrire dans le code (*code repository*). Il a signé un agrément de licence, dispose d'une adresse mèl *apache.org*, mais ne peut prendre que des décisions à court terme. Au dessus du committer, il y a le membre du *Project Management Committee*. C'est un développeur élu (à cause de son mérite) qui a libre accès à l'écriture et au dépôt, plus un droit de vote pour toutes les décisions concernant la communauté, et aussi le droit de proposer un utilisateur actif comme *committer*, etc. Tous les mainteneurs et tous ceux qui ont un droit de décision sont donc choisis (élus) pour leur « mérite », c'est-à-dire leur capacité à coder et à contribuer au développement du programme[42].

Un autre modèle est celui de l'association Debian, qui peut être définie comme communautaire et démocratique. Cette association gère la mise sur pied de distributions de logiciels libres autour des noyaux Linux ou FreeBSD[43], un projet totalement non commercial, qui comporte aujourd'hui plus de 29000 paquets logiciels, mobilise plus de 1000 développeurs, tous

41 Voir sur le site de la fondation Apache : < http://apache.org/foundation/how-it-works.html#meritocracy >.

42 Pour une discussion critique de la « méritocratie » comme mythe, voir : « Le logiciel libre et le mythe de la méritocratie » <http://www.framablog.org/index.php/post/2009/12/28/meritocratie-logiciel-libre-open-source >.

43 Debian est une des distributions la plus fiable et la plus renommée de l'open source. Elle est caractérisée en particulier par son gestionnaire de paquets (fichiers .deb), qui facilite les mises à jour et garantit un système homogène. C'est une méta-distribution, utilisée comme base pour la constitution d'autres distributions telles Knoppix, et Ubuntu (de la société *Canonical*).

bénévoles, et n'a de ce fait que des besoins assez réduits de financement, que des dons en nature (ordinateurs) et monétaires suffisent à combler. A la base de Debian, il y a un « *contrat social* » qui définit les « *principes du logiciel libre selon Debian* »[44] et une constitution qui règle les problèmes de fonctionnement, les méthodes de prises de décision, et le rôle des différents acteurs. Le projet est dirigé par un « chef de projet Debian » élu (et éventuellement réélu) chaque année par les membres de Debian, flanqué d'un comité technique et d'un secrétaire[45]. Les pouvoirs du chef de projet sont limités et toutes les décisions d'une certaine importance donnent lieu à un vote de la communauté. Un autre poste important est le *Release manager*, aidé par des assistants, qui a pour fonction de définir (avec la communauté des développeurs) les objectifs de la prochaine version, d'en superviser le processus et d'en fixer les dates de sorties.

Les réseaux d'utilisateurs qui se sont cristallisés sur certains projets de développement logiciel ont tous développés des formes de centralisation, une centralisation plus ou moins nécessaire pour assurer l'unité du projet et du produit, et la qualité du code (ou de la distribution envisagée). La même centralisation existe d'emblée dans le modèle du fondateur ou du « dictateur bienveillant ». On est loin cependant du modèle militaire traditionnel de l'entreprise capitaliste, avec un roi et des sujets, des dirigeants et des subordonnés, et une chaîne de commandement parfaitement opérationnelle. Ici, il n'y a pas de relation salariale et de chaîne de commandement. Les fondateurs, les dictateurs bienveillants, les chefs de projet, les membres des comités de management et les mainteneurs n'ont guère de moyens de contraindre réellement les comportements des autres développeurs. Les termes « fondateur », « chef », « dictateur », etc. ne doivent donc pas faire illusion. Aucun fondateur n'est propriétaire ici de ce qu'il a fondé, et aucun chef ne peut donner réellement des ordres, il peut tout au plus prendre une décision, quand il faut prendre une décision, c'est-à-dire fondamentalement pour assurer une certaine coordination. De toute manière, les enjeux réels sont le code et le pouvoir sur le code en développement, avec le contrôle du filtrage ou de l'intégration. Les conflits éventuels sont la plupart du temps d'ordre technique et portent sur le choix ou non de l'intégration de telle ou telle contribution, et éventuellement l'orientation du développement logiciel dans telle ou telle direction. Comme les différents contributeurs sont en principe volontaires et bénévoles, ils peuvent toujours déserter le projet ou aller voir ailleurs un projet parallèle et concurrent; ils peuvent aussi publier leurs propres modifications (sous forme de patches) ou lancer leur propre embranchement (*fork*). Le problème, pour les fondateurs, chefs de projet et mainteneurs, est plutôt de prouver en permanence qu'ils sont effectivement capables de remplir leur rôle de filtrage et d'amélioration du code, ou de réalisation des objectifs du développement, en montrant en quelque sorte qu'ils sont toujours « méritants ».

44 Cf. < http://www.debian.org/social_contract#guidelines >

45 La méthode de vote est la méthode Schulze, un vote avec classement des candidats (méthode Condorcet).

On a donc bien affaire a une sorte d'anarchie, de démocratie absolue, mais cette anarchie n'est pas dénuée de règles, de lois et de normes. Ce n'est pas un monde anomique, au sens d'Emile Durkheim[46]. Il y a ici des règles, des lois, et des normes de comportement, ce sont celles du commun, qui sont fixées ou découlent des différentes licences; et celles-ci juridiquement s'appuient formellement sur la loi du copyright et le droit des contrats, et en cas de besoin et de conflit sur ces règles, les tribunaux sauront trancher et les garantir[47]. Le titulaire du copyright a annulé ses droits exclusifs, mais cette annulation ne fait pas rentrer le logiciel dans un univers dénué de règles, et la mise en commun n'annule pas les autres règles (et lois) de la société marchande[48]. Il n'y a donc pas d'anomie ici.

V. Les entreprises et le commun, le commun et l'industrie

Les règles inscrites dans les licences des logiciels libres ne refusent ni la société marchande, ni le capitalisme. Elles interdisent simplement l'appropriation exclusive du code-source des logiciels libres et le contrôle exclusif du développement des logiciels. Les activités commerciales ne sont aucunement interdites, et les entreprises, comme tout autre utilisateur, peuvent utiliser le code comme elles l'entendent, en respectant évidemment les clauses de la licence. L'économie du commun est cependant très largement non-marchande, ce qui est particulièrement vrai pour les formes de production communautaires. Les modèles économiques commerciaux des entreprises sont alors configurés de manière à articuler la formule générale du capital (et la recherche de la rentabilité), et la dépense ou le financement à fonds perdus d'un certain nombre de projets – souvent communautaires – de développement logiciel, en rapport plus ou moins étroit avec le modèle économique des entreprises concernées. Ce ne sont d'ailleurs pas les règles des licences, ni le choix d'une mise en commun de la ressource (comme nous l'avons vu dans notre point I), qui fait du logiciel libre (et du développement logiciel) une activité en générale « non-marchande », financée par des dons, des subventions, des dépenses à fonds perdus. C'est la nature des ressources logicielles elles-même, avec un coût de copie et de duplication pratiquement nul, qui conduit à cette situation.

On peut remarquer de plus que si les licences abolissent de fait le droit de copyright accordé à l'auteur du code, et parfois les droits exclusifs qui pourraient découler de l'existence d'un brevet

46 Durkheim a introduit le terme en 1893 dans *De la division du travail social*, et l'a repris ensuite dans son ouvrage sur le suicide, avec la catégorie du suicide anomique. Il désigne par anomie (du grec a-nomos) un état de désintégration des règles sociales existantes ou une absence totale de règles, de lois, de normes, et donc la décomposition de l'ordre social.

47 Les procès, assez rares, ont d'ailleurs confirmé (en Allemagne, en France, aux Etats-Unis), la valeur juridique de la licence GNU-GPL. Le dernier en date en France, engagé contre l'opérateur *Free* pour violation de la licence GNU-GPL, s'est terminé par un accord, *Free* acceptant de publier comme la licence lui impose les sources de sa freebox qui intègre un noyau Linux.

48 La *Free Software Fondation* précise d'ailleurs qu'en cas de manque de la licence GNU-GPL (ou toute autre licence GNU), la situation particulière n'ayant pas été prévue, c'est la législation du pays d'accueil en matière de droit d'auteur qui s'applique.

logiciel (ce qui est le cas pour certaines licences comportant une clause de fin de brevet), elles ne le font que parce que l'auteur a explicitement renoncé à l'exercice de cette « propriété intellectuelle », un renoncement qui n'affecte pas ses autres droits, y compris les autres « droits de propriété intellectuelle » éventuellement accrochés au même code, comme par exemple un nom transformé en marque (*trademark*)[49]. Cette abolition du copyright et de l'exclusivité n'est d'ailleurs aucunement une abolition de toute forme de propriété. Le logiciel comme bien, une chose physique et tangible, est simplement ramené aux règles de la propriété ordinaire sur les biens, ces règles qui organisent et encadrent dans la société marchande la possession et la transmission des choses physiques et tangibles[50]. Le logiciel devient comme les scies, les marteaux, la farine, ou certaines machines qui ne connaissent ni le copyright, ni les brevets, et sont pourtant pour les juristes (et les autres) des objets de propriété. Les producteurs de farine par exemple ne peuvent commercialiser leur farine, même sous leur propre marque, à travers un système de licences restrictives, avec une licence pour les boulangers interdisant de faire des gâteaux, une licence pour les pâtissiers interdisant de faire du pain, une licence pour les usages domestiques, une pour le Carnaval, etc. Ils sont cependant propriétaires de leur farine, avec tous les droits d'*usus*, *abusus*, *fructus*, qui en découle. Ils peuvent donc la vendre et leur droit de propriété est intégralement transféré lors de la vente.

Le logiciel libre est traité de la même façon, comme une chose dont le possesseur a la pleine propriété, avec un transfert intégral de cette propriété dans toute opération de distribution ou redistribution du code, qu'il y ait ici ou non paiement. Il n'y a donc pas de problème majeur pour articuler juridiquement le commun des logiciels libres et les activités marchandes, pas plus qu'il n'y a de problème majeur pour la plupart des entreprises à articuler la formule générale du capital pour l'ensemble de leur activité, et une logique de dépense sans recherche de rentabilité pour certaines activités dont les résultats peuvent être ré-utilisés par elles mais aussi par d'autres, qu'ils aient ou non contribué à ces activités. Les entreprises utilisatrices du commun ne sont d'ailleurs pas dans une situation très différente des autres utilisateurs. Certaines sont dans la situation des

49 Le droit des marques est tout entier conçu pour l'environnement marchand et ne peut guère bloquer un développement logiciel non marchand. Ainsi, « OpenOffice.org » était une marque appartenant à *Sun*, qui détenait aussi les droits de copyright sur le code-source de la suite bureautique. Quand *Oracle* a racheté *Sun*, elle a récupéré toute la « propriété intellectuelle » de l'entreprise, et donc cette marque et le copyright sur le code d'OpenOffice. Mais comme ce code était distribué sous licence GNU-LGPL, Oracle n'a pu empêcher la formation d'un *fork*. Quand la fondation *Open Document* a demandé à *Oracle* le transfert de la marque, la firme a refusé. Et il a fallu changer le nom du logiciel et inventer une nouvelle marque « LibreOffice », ce qui est bien peu de chose pour un projet de développement logiciel de cette envergure (et ceci d'autant plus qu'il n'existe pas de marché ici) ! Pour se protéger des *trolls* et autres spéculateurs, les noms des projets de développement sont souvent déposés comme marques aujourd'hui, même quand le projet n'a aucune activité commerciale. Ainsi Linux® est aussi une marque.

50 La situation est semblable en effet à celle des livres d'un auteur dont l'œuvre « tombe » dans le domaine public. Les règles de la propriété ordinaire qui s'appliquaient déjà aux livres comme objets tangibles s'appliquent toujours ici, avec simplement la disparition de l'interférence antérieure que pouvait représenter les droits dits de « propriété intellectuelle ». Il y a cependant passage du monopole au régime concurrentiel, avec comme base juridique les règles de la propriété privée ordinaire.

utilisateurs ordinaires, d'autres dans celle des utilisateurs-programmeurs.

(1) L'usage ordinaire

De nombreuses entreprises ou organisations ont besoin de logiciels pour assurer le fonctionnement routinier de leurs activités, sans être engagées la plupart du temps dans le développement logiciel ou avoir besoin d'intégrer d'une manière ou d'une autre ces logiciels à leur offre. Pour des tâches de bureautique par exemple, elles peuvent utiliser LibreOffice (OpenOffice.org hier) plutôt que MSOffice, pour la navigation web Firefox plutôt qu'Internet Explorer, et il en est de même pour leurs besoins en messagerie, en gestion de base de données, ou pour le système d'exploitation à installer sur leurs postes de travail. En choisissant des logiciels libres, elles économisent le prix des licences, le paiement d'une rente à Microsoft ou à un autre détenteur de droits, ce qui est loin d'ailleurs d'être négligeable; et c'est souvent la première raison de leur choix. La Gendarmerie Nationale (française) a en quelques années migré 15000 postes bureautiques de MSWindows et MSOffice vers Linux, OpenOffice, Firefox et Thunderbird (90000 sont prévus pour 2015); et l'économie réalisée est, selon le lieutenant-colonel Guimard de 50 millions d'euros de 2004 à 2009. Il fallait en effet acheter avant de 12000 à 15000 licences par an, un chiffre tombé à 27 [51].

D'autres entreprises vont utiliser certains logiciels développés dans l'économie du libre pour construire leur propre offre commerciale. Ce qui les intéresse alors est la valeur commerciale que ces logiciels peuvent acquérir une fois intégrés dans une offre plus large. Elles vendent des logiciels libres tirés du commun et obtenu « gratuitement », ce qui n'est évidemment possible que dans certaines conditions. On peut citer ici les distributions commerciales GNU/Linux, avec la construction de différentes collections de logiciels libres autour d'un noyau Linux axquelles sont ajoutées un certain nombre de services (installation, formation, maintenance, etc.). De telles offres, comme la pratique l'a prouvée, intéressent surtout les entreprises, grandes ou petites, et souvent ces offres commerciales sont flanquées d'un mode de distribution en gratuiciel réservé aux utilisateurs domestiques, sous forme communautaire, avec ici un retour d'expériences et l'apport de contributeurs externes (tests et correction de bogues, etc.) que les relations purement commerciales n'assurent pas. Ainsi *Red Hat* et le projet Fedora, *Suse* et OpenSuse, etc. D'autres firmes ont simplement renouvelé leur offre vieillissante, en abandonnant les Unix propriétaires pour Linux et tous les logiciels libres qui l'accompagne. Elles obtiennent ainsi une garantie de continuité dans le développement, ainsi que de nombreuses innovations et mises à jour. C'est ce qu'a fait *IBM* et bien d'autres. Des alliances de fait sont alors nouées entre ces entreprises et les développements communautaires. Les firmes financent ces développements ou y engagent une partie de leurs salariés programmeurs. En matière d'innovation ou de production de ressources,

51 La Gendarmerie Nationale est aussi productrice de logiciels libres pour ces propres besoins, mais aussi en partenariat avec la Caisse d'Assurance Maladie, le ministère du Budget et le ministère de la Défense (plugins de sécurité pour Thunderbird par exemple).

ces pratiques, où des firmes concurrentes financent des projets communs et coopèrent ensemble, n'ont d'ailleurs rien d'exceptionnel. En termes de management, on parle simplement « *d'une mutualisation des ressources* », et certains logiciels libres sont devenus de fait des ressources communes pour toute une partie de l'industrie.

(2) La ré-utilisation et l'innovation

D'autres entreprises sont, comme les utilisateurs-programmeurs, intéressées par la grande liberté dont dispose l'utilisateur en matière de ré-utilisation et de transformation du code du logiciel libre, avec en plus les facilités pratiques offertes par un commun qui a été entièrement construit pour faciliter le travail de développement logiciel. Au niveau le plus élémentaire, on trouve le code embarqué, dédié, intégré dans une machine ou un dispositif matériel particulier (ascenseur, machine à laver, téléphone portable, routeur, camescope, etc.[52]). C'est un domaine où le système d'exploitation Linux est aujourd'hui fortement présent, choisi non seulement pour ses qualités propres et l'existence de versions du noyau Linux adaptées à des capacités de stockage de mémoire extrêmement réduites, mais aussi pour son environnement de développement logiciel, avec le compilateur GCC, les bibliothèques..., et la connaissance que tous les professionnels de l'informatique ont de ces outils.

La même recherche de ressources utilisables pour l'innovation avec rejet d'une éventuelle solution propriétaire, explique l'importance aujourd'hui de Linux et des logiciels libres dans l'ingénierie des systèmes (en dehors évidemment des qualités propres des différents logiciels en présence). On peut donner deux exemples ici, qui sont aussi deux modèles économiques différents. Le premier exemple est le système d'exploitation OS X (alias Darwin), dont l'origine est NeXTEP, le système d'exploitation orienté objet de la société *NeXT* fondée par Steve Jobs après son départ d'*Apple* en 1985 (une société rachetée par *Apple* en 1996). Sa base est un système FreeBSD avec un micro-noyau Mach développé à l'Université Carnegie-Mellon[53]. OS X intègre de surcroît un grand nombre de logiciels libres, le tout combiné avec une interface graphique propriétaire Aqua et son moteur Quartz. Le commun avec toutes ses ressources a permis alors à *Apple* de doter ses machines d'un très bon système d'exploitation, sans obstacles techniques ou juridiques insurmontables, la cohabitation de logiciels libres et propriétaires étant parfaitement possible dans le même produit commercial[54], et sans problème non plus pour son propre modèle économique (une offre conjointe de design, matériel et de logiciels). L'emprunt n'est d'ailleurs pas à sens

52 Voir en Annexe F l'exemple d'un camescope Sony et le patchwork des droits de propriété intellectuelle entassés sur cette petite machine, qui intègre un noyau Linux et d'autres logiciels libres.

53 Mach est l'une des premières implémentations réussie de la théorie des micro-noyaux. Développé à l'origine à l'Université Carnegie-Mellon, il a été repris par la suite par l'*Open Software Foundation* (OSF), une organisation fondée par tout un ensemble d'entreprises en 1988 en vue de créer un standard ouvert pour le système d'exploitation Unix. Depuis un *fork* a donné naissance à GNU Mach, destiné au projet Hurd de la *Free Software Foundation*.

54 La version libre du système s'appelle Darwin, utilisable en ligne de commande, sans Quartz et l'interface graphique Aqua. Il existe aussi une version libre de NeXTStep, GNUStep dérivé d'OpenStep, le projet commun de *NeXT* et de *Sun* (1994).

unique, car une partie du travail de développement du système et de ses logiciels est rétrocédé au commun (voir l'encart sur les moteurs de rendus html et WebKit, Annexe B)

Le deuxième exemple que nous pouvons donner est celui des supercalculateurs, des machines dédiées aux calculs à très grande vitesse, utilisés pour la prévision météo, l'étude des climats, les simulations aéronautiques, les calculs de résistance de matériaux, la simulation des explosions nucléaires, etc. Aujourd'hui, la majorité de ces machines utilisent Linux (92,4 % des 500 supercalculateurs les plus puissants en juin 2012[55]) et d'autres logiciels libres. Ce choix s'explique aisément. Ainsi, quand le *CEA* a lancé en 2004 un appel d'offre pour doter sa *Division des Applications Militaires* d'une puissance de calcul suffisante pour simuler les explosions nucléaires, une des conditions de l'appel était l'utilisation de logiciels libres pour toute la partie logicielle de la machine. C'est *Bull*, une SSII qui avait fait du logiciel libre l'axe de son développement et qui maîtrisait une partie de la technologie nécessaire, avec ses serveurs Novascale, qui emporte alors le marché. Tera 10, le supercalculateur, est décrit comme occupant 800 m² au sol sur plusieurs niveaux[56]. Il utilise 4352 processeurs intel disposés en grappes, avec sur chaque processeur un noyau Linux. Chacune de ces grappes est gérée par un serveur Novascale (au total 602 serveurs), utilisant le système de fichiers distribués Lustre[57]. Tera 10 mobilise pour son fonctionnement presque un millier d'ingénieurs, dont certains ont d'ailleurs comme tâche le développement logiciel et la recherche d'une amélioration permanente de la puissance de calcul. Comme l'expliquait au journal *Le Monde* en 2004, le directeur de la R&D de *Bull* « *Il est indispensable de pouvoir « limer » au plus juste la partie logicielle. Cette optimisation est possible grâce à l'accès au code source des programmes libres tels que Linux et ne l'aurait pas été avec Windows* »[58]. De plus, le développement logiciel d'une telle machine est commencé par la SSII (*Bull*) qui fournit l'ensemble, mais doit nécessairement continuer chez l'utilisateur (la *Division des Applications Militaires* du *CEA*) et être pris en charge par lui. C'est une situation fréquente dans l'industrie informatique, ce qui rend les principes du logiciel libre quasiment incontournables, tout d'abord, parce que le code est alors disponible, mais aussi, parce que la relation nouée par la licence entre les parties contractantes ne peut conduire à une situation d'enfermement.

55 Cf. < http://linuxfr.org/news/sortie-du-top-500-de-juin-2012 >.

56 En 2006, lors de son inauguration, Tera 10 était le supercalculateur le plus puissant d'Europe (le 6e au niveau mondial). Sa puissance de calcul est d'environ 60 teraflops (soit 60×10^{12} flops); le « flops » étant une unité conventionnelle de mesure du nombre d'opérations de calcul en virgule flottante par seconde. Depuis 2006, le nombre des machines a fortement augmenté dans le monde (et en Franc), avec une croissance considérable des puissances de calcul. Ainsi en 2010, Bull a livré au CEA une nouvelle machine Tera 100, dont la puissance théorique est de 1,25 pétaflops (10^{15} flops), soit 20 fois celle de Tera 10 (17ème rang mondial en juin 2012, et 2ème en France, derrière le TGCC).

57 Lustre (contraction de Linux et *cluster*) est un système de fichiers distribués utilisé pour de très grandes grappes de serveurs (*cluster*) comportant plusieurs centaines de nœuds, avec une capacité de stockage de plusieurs petabytes. Utilisé dans les supercalculateurs, ce logiciel est sous licence GNU-GPL, avec comme développeurs la société *Cluster File Systems*, à l'origine, puis *Sun* avec d'autres entreprises (dont *Bull* et le *CEA*). Suite au rachat par *Oracle* de *Sun*, le développement, toujours assuré par des entreprises, est devenu « communautaire ».

58 *Le Monde* du 12 décembre 2004 : « *Le CEA fait appel à Bull pour fabriquer son nouveau supercalculateur* ».

(3) L'usage stratégique des logiciels libres et du commun

Nous avons affirmé précédemment que le logiciel envisagé du point de vue de l'usage, était un produit complexe. Comme code-source et binaires, il connaît deux grands types d'utilisations, pour l'exécution de ses fonctionnalités ou l'innovation, avec deux catégories d'utilisateurs. C'est bien ce que nous avons retrouver ici, et à côté de l'économie non-marchande et communautaire du libre, et plus ou moins en symbiose avec elle, une économie purement capitaliste de production de logiciels libres s'est constituée sur ses deux types d'utilisations. Mais ce simple constat est insuffisant pour rendre compte de l'articulation de cette économie capitaliste avec le monde des logiciels libres. Le logiciel libre est en effet un produit encore plus complexe que le logiciel en tant que tel (libre ou non). Il fournit une autre valeur d'usage, car le code libre est accompagné de règles juridiques spécifiques. Les quatre libertés du logiciel libre (plus éventuellement la clause *copyleft*) représentent en effet une sorte de « valeur d'usage », une valeur sociale en quelque sorte, qui peut être recherchée en tant que telle, aussi bien par les utilisateurs ordinaires ou les utilisateurs-programmeurs, que par les entreprises ou les organisations, c'est-à-dire – pour parler plus rigoureusement – par les dirigeants de ces entreprises ou organisations.

Ici, ce n'est pas la possibilité d'exécuter le programme ou la possibilité de s'en servir pour innover et créer un autre programme qui est recherchée, mais les règles elles-mêmes avec les comportements qu'elles induisent, qu'elles encadrent, qu'elles proscrivent. On peut même utiliser le commun pour ses règles, quasiment indépendamment des ressources elles-mêmes. Le commun a en effet transformé et transforme toujours l'économie des logiciels, et influence même les activités où les logiciels sont présents. Nous parlerons alors d'un usage stratégique en distinguant celui-ci de l'usage ordinaire, final, d'un programme, et de l'usage pour l'innovation que l'on peut faire du code. Dans l'usage stratégique du commun et des logiciels libres, ce que l'on cherche, c'est l'abolition de fait du « droit de propriété intellectuelle » et de tout ce qu'il porte en lui comme possibilités de verrouillage technico-juridique, de création de monopole, de situation d'enfermement et de mises à rançon éventuelle[59].

L'enfermement technico-juridique du droit du copyright n'a pas simplement comme conséquence la fermeture du code, le blocage du développement logiciel coopératif et de l'innovation, toutes choses qui mobilisent les utilisateurs-programmeurs (et aujourd'hui certaines

59 Dans une économie dominée par des droits exclusifs sur la technique, ces situations sont en effet non seulement possibles, mais banales. Donnons un seul exemple : En 1985, la compagnie de téléphone *Bell Atlantic* passait commande, par appel d'offres, d'un système de commutation numérique, et le fournisseur le moins cher et le mieux disant, *AT&T*, était alors retenu. Mais le système fourni par *AT&T* était propriétaire et à code fermé, et pour toute extension et toute modification, *Bell Atlantic* ne pouvait désormais que faire appel à *AT&T*, à l'exclusion de tout autre fournisseur. Quand la *Bell Atlantic* voulut ultérieurement faire évoluer son offre et introduire les numéros verts, il n'y avait plus d'appel d'offres possible. *Bell Atlantic* s'est retrouvée dans la situation d'une « vache à lait » pour *AT&T*, ce qui donna lieu à une action judiciaire au titre de l'antitrust pour « abus de position dominante » (Schapiro et Varian, 1999). Ajoutons qu'aujourd'hui les choses ont bien évolué, le concept du logiciel libre étant passé par là, et l'expérience aidant, il n'y a sans doute plus guère d'entreprises qui se laisseraient prendre ainsi.

entreprises), mais aussi l'élimination progressive de toute concurrence commerciale directe, la fermeture des marchés, et l'institutionnalisation de prélèvements sur la valeur marchande créée par les autres entreprises. Cet aspect du problème est généralement ignoré des communautés du libre, car leur économie est fondamentalement à l'écart des relations marchandes, et par là même n'est guère concernée par la domination commerciale ou la monopolisation de tel ou tel marché. Cette question à l'inverse va mobiliser au premier chef les entreprises bloquées dans leurs possibilités de valorisation de leurs capitaux par les droits exclusifs des autres, et l'usage qu'ils en font.

Ici certaines entreprises – celles qui sont les plus menacées ou souffrent le plus de ces blocages – vont alors se mobiliser pour changer la situation en adoptant le principe du partage, en produisant des logiciels libres, en soutenant certaines fondations et certains développements communautaires par des apports de fonds ou les contributions de leurs programmeurs salariés. De manière souvent pragmatique, elles cassent ainsi les situations de monopole qui les menacent, ce qui ne les empêchent pas par ailleurs de déployer souvent dans un autre contexte et un autre domaine des stratégies fondées sur l'exclusivité[60]. Cette utilisation du logiciel libre est d'ailleurs une stratégie bien plus efficace que l'alternative du recours à l'action antitrust. Car même si une condamnation en justice finit par être obtenue, après la longue marche des procédures et malgré les aléas judiciaires, la décision des juges ne peut produire le (ou les) logiciel(s) alternatif(s) permettant de rendre la dé-monopolisation effective. La production d'une base de logiciels libres réellement accessibles et utilisables élimine par contre le problème, comme l'expérience de la « bataille des navigateurs » l'a prouvé. On transforme alors profondément et durablement la situation.

Quand *Netscape* a été évincé du marché par *Microsoft*, Internet Explorer (lié à MSWindows) est devenu le seul navigateur disponible pour accéder au web, et partant le point de passage obligé pour tous ceux qui voulait développer des activités commerciales sur ce réseau, ce qui donnait à *Microsoft* un pouvoir particulièrement important dans ce domaine. On a vu alors que parallèlement aux procédures antitrust engagées contre *Microsoft*, ses rivaux dans l'industrie ont mis sur pied et financé la libération du code du navigateur de *Netscape* et le projet Mozilla.org.

Le même basculement en faveur de solution logicielle libre est devenu très fréquent aujourd'hui. La mise en commun (ou mutualisation) des ressources est non seulement une économie en matière de développement logiciel, mais surtout une garantie contre l'enfermement propriétaire et commercial, car elle donne quasi-automatiquement naissance à une situation plus

60 On observe alors des conduites et des discours contradictoires, qui frisent même la schizophrénie. Ainsi en 2005, IBM, le plus gros producteur mondial de patents, avec une moyenne d'environ 3200 patents déposés chaque année à l'USPTO (1300 de plus que toute autre compagnie), déclare ne jamais vouloir utiliser ceux-ci dans un procès intéressant le logiciel libre et offre de surcroît un usage *royalty-free* de 500 brevets logiciels importants aux développeurs *open source*. Et dans le communiqué où la raison explicite de l'annulation est la volonté de favoriser le développement et l'innovation logicielle, on ré-affirme l'importance des patents pour l'innovation !

concurrentielle. Ici le logiciel libre, avec ses licences universelles, rentre dans un mouvement plus général de recherche permanente par une partie de l'industrie d'une neutralisation des effets destructeurs pour l'ensemble des activités et des marchés de tous les verrouillages techniques renforcés par l'existence des droits de propriété intellectuelle (patents ou copyright). En effet, ces verrouillages sont créateurs de barrières à l'entrée, et interdisent même parfois toute possibilité d'activité économique, par paralysie mutuelle. C'est le cas dans le domaine des interfaces[61], ou en matière de format de fichier, ou pour les infrastructures communes (comme l'internet). Le partage et la mise en commun des informations techniques, avec annulation des droits de patents et de copyright, s'imposent alors, ce qui donne les « formats ouverts », ou les « standards de l'internet » du W3C[62], ou la revendication générale d'une informatique plus ouverte (« open » avec par exemple l'*Open Software Foundation*, puis l'*Open Group*[63]). Le rejet de toute forme d'exclusivité et de tout contrôle monopoliste du développement des logiciels est bien alors une option stratégique, et un choix pragmatique motivé par la nécessité d'un partage des ressources indispensables au déploiement de modèles économiques purement commerciaux, capitalistes et privés.

VI. Conclusion : le commun et le « modèle de la pâture »

L'expérience historique que représente les communs du logiciel libre est en parfaite contradiction avec le « modèle de la pâture » de Garrett Hardin (1968), un modèle où des ressources limitées restées en propriété commune sont victimes de surexploitation à cause de l'indifférence des utilisateurs à maintenir et reconstituer la ressource. La solution prônée est alors le découpage de cette propriété commune en propriétés exclusives (ou privées), ce qui aura pour conséquence de préserver la ressource et d'inciter à une exploitation rationnelle de celle-ci en permettant l'investissement (comme planter du trèfle par exemple), sans avoir à craindre l'arrivée de moutons divaguant. Ce schème idéologique en faveur d'une privatisation généralisée est ressassé depuis une trentaine d'années comme vérité d'évangile, et mobilisé par les économistes et juristes qui prônent « l'analyse économique du droit », et plus particulièrement du droit de la propriété intellectuelle, à partir de l'hypothèse selon laquelle « *la loi essaye de promouvoir*

61 Ce qui a conduit à des évolutions du droit en la matière, en Europe en particulier (Directive européenne du 14 mai 1991 et loi française « pour la confiance dans l'économie numérique » du 21 juin 2004),avec des obligations de révélation du code pour des raisons d'interopérabilité.

62 Les activités de l'internet repose sur des accords annulant de fait les patents des uns et des autres pour tout ce qui est lié au réseau et à son fonctionnement, une politique impulsée par le W3C (*the W3C patent policy*) qui a institué la pratique d'un usage libre, *royalty-free*, et sans nécessité de licence, de toutes les techniques sous brevet utilisées par l'internet pour tous les utilisateurs. Cf. « http://www.w3.org/2003/12/22-pp-faq ».

63 L'Open Software Foundation (OSF), fondée en 1988, avait pour but la création d'un standard ouvert pour une implémentation du système d'exploitation Unix, avec comme entreprises fondatrices : *Apollo Computer*, *Bull*, *Digital Equipment Corporation*, *Hewlett-Packard*, *IBM*, *Nixdorf Computer* et *Siemens*. *Philips* et *Hitachi* devaient rejoindre ce groupe ultérieurement. L'OSF a fusionné en février 1996 avec le consortium X/Open (*Bull*, *ICL*, *Siemens*, *Olivetti*, *Nixdorf*, *Philips*, *Nokia*, *AT&T Corporation*, *Digital*, *Unisys*, *Hewlett-Packard*, *IBM*, *NCR*, *Sun Microsystems*, *Prime Computer*, *Apollo Computer*, *Fujitsu*, *Hitachi*, *NEC*...), pour donner naissance à l'Open Group.

l'efficience économique » (Landes et Posner, 1987, p. 365)[64]. On trouve ici, pour donner quelques références, Landes et Posner (1987, 1989, 2002), Watt (2000), mais aussi Kitch (1977); tous ces auteurs justifiant l'existence et le renforcement des droits de propriété intellectuelle dans tous les domaines et sous les formes les plus variées, les *patents*, le *copyright*, les *trademarks*, mais aussi les secrets de fabrication, la protection des personnalités, des caractères, un ensemble de dispositifs juridiques posés comme des « *property rights* », lesquels sont définis par la faculté d'interdire et par l'exclusivité individuelle, comme « *le pouvoir légalement garanti d'exclure les autres de l'utilisation d'une ressource, sans qu'il soit besoin de contracter avec lui* » (Landes et Posner, 1987, p. 266), et non comme le droit de faire, ou d'exploiter, ou de jouir de la chose, comme droit d'usage pour un ou plusieurs individus, etc.

Leur analyse peut être résumée en trois points.

(1) Il y a tout d'abord un dogme à la base de leurs raisonnements qu'on peut appeler l'hypothèse marchande ou capitaliste. Selon eux, la production d'une invention ou d'une œuvre (comme de toute autre chose d'ailleurs) est assimilable à un investissement dont on espère retirer un profit. Les droits de propriété intellectuelle créent alors l'incitation nécessaire à cette production, et sont alors nécessairement exclusifs[65]. Comme le résument Besen et Raskind (1991) : « *L'objectif de la propriété intellectuelle est la création d'incitations qui maximisent la différence entre la valeur de la propriété intellectuelle créée et utilisée et le coût social de cette création, incluant le coût d'administration du système* » (p. 5). On trouve cette thèse dans des formulations différentes et plus ou moins absolues. La recherche d'un profit, d'un gain net, voire d'une « valeur pécuniaire » (Posner) est soit la seule incitation existante ou à prendre en compte dans l'analyse[66], soit l'incitation la plus importante (en concédant alors qu'il en existe bien d'autres), soit, plus subtilement, la seule incitation qui permette d'intensifier la production, d'augmenter donc la quantité et la qualité des œuvres littéraires, musicales, cinématographiques, artistiques, ou les activités de recherche et d'innovation...

(2) Il faut alors faire face au célèbre « cavalier seul » (le *free rider*), au resquilleur, au

64 Ce qui veut dire qu'on peut lui appliquer les modes d'analyse propres aux économistes, en l'occurrence ceux de la micro-économie néo-classique la plus standard. On peut remarquer que la « loi » dont il est question ici est la loi américaine. On chercherait en effet vainement dans cette littérature la moindre analyse comparée des différents principes et systèmes juridiques (copyright et droit d'auteur par exemple). Cf. Besen et Raskind (1991) pour un *survey* sur cette approche.

65 C'est laisser de côté, nier même, la grande variété, empiriquement constatable, des motivations créatrices, et le rôle important joué des motivations non monétaires, en rupture complète avec la littérature économique antérieure sur cette question. Sur le copyright voir Carey (1853), Plant (1934), et plus récemment Breyer (1970), lequel est d'ailleurs la cible du premier article sur le copyright de Landes et Posner (1989); Sur les patents voir Machlup (1958).

66 Dans leur article de 1989, « An economic analysis of copyright law », Landes et Posner pose d'emblée que pour qu'une nouvelle œuvre soit créée, le revenu espéré lié à la vente des copies doit dépasser le coût anticipé (coût de la création et coût de fabrication des copies, auquel il faut ajouter le coût de la répression des copieurs (coût de transaction)); et à partir de cette hypothèse, ils expliquent les principaux traits de la loi américaine du copyright : « protection de l'expression et non des idées », la doctrine du « *fair use* », etc.

pirate..., une menace permanente pour les échanges marchands portant sur ce « bien public » qu'est « l'information ». On parle aussi d'un « échec de marché » que seul l'établissement d'un système de droits exclusifs débouchant sur des échanges de droits permet de surmonter, ce qui est la seule manière d'atteindre l'efficience (en vertu du « théorème de Coase »); reste à évaluer les coûts de ce système. On peut aussi plus simplement mettre en avant l'écart entre le coût de la production initiale (élevé) et les coûts d'une copie (plus faibles); et cet écart est supposé miner toute incitation et ruiner toute création si l'activité des *free-riders* est tolérée[67] (en vertu de la première hypothèse).

(3) Autre dogme : sans un système de droits de propriété (privés, exclusifs, etc.), il n'y a pas d'activité économique efficace. C'est la reprise du « modèle de la pâture » ou des « *anti-commons* ». Les communs, ou le domaine public, avec leurs libertés d'usage, constituent un système inefficace en matière de gestion, conservation, et exploitation des ressources[68]. Ce point conduit directement à prôner l'établissement et le renforcement partout et dans tous les domaines d'un système généralisé de droits de propriété exclusifs (et la disparition progressive du domaine public). C'est « l'utopie propriétaire » défendue et mise en œuvre depuis les années 1980 par les partisans de cette « propriété intellectuelle » et toute une coalition d'intérêts, où l'on trouve quelques industries et entreprises productrices et propriétaires de titres, mais aussi les Offices de brevets, les cabinets d'avocats spécialisés, etc., un lobby très actif, ce qui explique le caractère très répandu de ces thèses.

Ce qui est intéressant dans l'expérience du logiciel libre est le démenti pratique de toutes ces apories. Les principes du logiciel libre et l'application qui en est faite par les utilisateurs de logiciels, y compris les entreprises, s'opposent en effet, point par point, à ces différentes thèses et au « modèle de la pâture ». Il en est de même pour beaucoup de pratiques et d'arrangements institutionnels dans l'économie des logiciels, et plus généralement dans l'économie des techniques,

67 Selon la célèbre formule « la piraterie tue le livre », voire le cinéma, voire le logiciel, etc. C'est évidemment une sorte de raisonnement à l'envers, qui confond les échanges de connaissances techniques, d'information, d'idées, et la création elle-même, et ce qui relève du commerce ou des échanges de titres. Avec des droits de propriété intellectuelle, un échange commercial préalable est bien une condition à la communication de l'information, mais cette communication et les productions qu'elle génère peuvent très bien exister (et existent) sans droits de propriété intellectuelle. On ignore alors tout ce qui relève de l'économie non-marchande, et tout ce que l'économie marchande (capitaliste) construit sur (ou avec) des informations appartenant au domaine public, c'est-à-dire les plus grands marchés d'information (comme la presse, la radio, la télévision, largement construite sur un principe de *free speech* (ou de libre parcours) et de non privatisation des idées et des informations).

68 C'est le raisonnement de Kitch (1977) pour qui le contrôle par le titulaire d'un patent de toutes les trajectoires de recherche (c'est-à-dire la propriété du *prospect*), permettrait d'éviter le gaspillage de ressources (*waste*) que représentent des recherches parallèles, dispersées et concurrentes. C'est aussi l'argument principal de Landes et Posner justifiant leur proposition d'un « *copyright* indéfiniment renouvelable » (2002), suite au vote du *Sonny Bono Act* par le Congrès américain. Pour contrer l'argument de leurs adversaires sur l'absence d'externalités négatives dans des « communaux inépuisables » (à la différence des pâtures), ils introduisent une distinction entre les « externalités technologiques » (brouter l'herbe) et les « *externalités pécuniaires* » (p. 13). Si l'icône nationale des américains, la petite souris Mickey, tombait dans le domaine public, l'icône serait victime de sur-utilisation, son image serait vite brouillée et le public vite fatigué, et sa valeur commerciale serait perdue (comme l'herbe du pré foulée par les moutons)...

quand ces arrangement et pratiques visent à neutraliser les effets désastreux pour l'activité économique et l'innovation des droits de propriété intellectuelle dans leur forme la plus exclusive (comme la *patent policy* du W3C par exemple, ou le renoncement ça et là à exercer les droits de brevet, ou les accords de mutualisation, ou certains pools de patents, etc.).

On peut remarquer que l'hypothèse fondamentale du calcul capitaliste du créateur d'une œuvre est dénuée de toute réalité, car au mieux applicable à un tout petit nombre de créateurs. C'est plutôt une transposition, qui se veut savante, car relevant de la science économique, plus exactement de la micro-économie, des calculs d'intérêts des investisseurs et propriétaires de titres soucieux de valoriser leurs actifs. Mais produire et « valoriser » un titre n'est pas produire une œuvre (quelque soit cette œuvre), et de plus à notre époque, les propriétaires de titres ne sont pas en règle générale des créateurs. C'est vrai pour les œuvres littéraires comme pour les inventions et les logiciels. Dans le domaine des logiciels libres, on voit des gens travailler gratuitement pour le commun, dans le seul but de produire ou d'améliorer un programme, ou pour participer à un développement logiciel, ou pour apprendre à programmer, ou par jeu, ou pour toute autre raison. C'est le cadre institutionnel des licences, la situation de production, et l'objet logiciel lui-même qui expliquent ce genre de comportements, lesquels n'ont d'ailleurs rien d'exceptionnel, et ne sont par ailleurs aucunement spécifiques du monde des logiciels.

Notons ensuite que le logiciel libre et son économie ne craignent ni les « pirates », ni le « *free rider* ». La copie des logiciels n'affecte pas la production et le développement de ceux-ci, elle ne s'oppose pas non plus - quand évidemment la situation le permet - à la commercialisation du code source; elle exclut simplement le prélèvement d'une rente. Elle ne bloque ni la production, ni l'activité innovatrice, et ne détruit pas la ressource mise en commun. C'est même plutôt l'inverse, car la copie est une condition du développement des logiciels et du commun; en contradiction totale avec le « modèle de la pâture », plus nombreux sont les moutons broutant le pré, plus l'herbe qui repousse est grande et belle. C'est d'ailleurs la force du développement communautaire, avec cette forme de travail largement distribué passant par l'internet. Quand un développement a réussi à mobiliser une population suffisamment importante d'utilisateurs, êtres humains ou entreprises, les forces et les compétences nécessaires pour résoudre les problèmes du développement vont se présenter d'elles-mêmes.

La mise en commun, l'établissement d'une propriété commune en matière d'usage représente, comme l'expérience l'a montré et le montre encore, non une communauté négative, stérile et destructrice, mais à l'inverse une forme de communauté positive, source d'innovations et de production de nouvelles richesses. Le discours des « *anticommons* » est donc invalidé, et ceci non par une argumentation théorique, mais par l'action des premiers intéressés, les utilisateurs des logiciels. Il suffit de regarder ce qui s'est mis en place, et existe aujourd'hui, dans le monde économique réel. Ce qui est vrai pour le code-source est d'ailleurs vrai de la même manière pour

toutes les connaissances, informations et artefacts techniques, lequels sont à la fois les produits de l'activité inventive des êtres humains et des matériaux nécessaires à cette activité. Dans l'hypothèse où l'innovation est une chose positive, et où socialement, il est souhaitable d'encourager l'activité innovatrice et l'utilisation des innovations, le contrôle exclusif centralisé, avec création artificielle d'une rareté au moyen de droits de propriété intellectuelle, est une catastrophe qui ne peut que freiner et bloquer l'innovation et les activités économiques qui en dépendent. A l'inverse, comme le montrent les logiciels libres, la diffusion la plus large, la disponibilité, le libre accès sur un mode universel, sans exclusive, avec en plus une complète liberté d'usage et de ré-usage (y compris commercial), représentent manifestement un environnement plus favorable aux innovations et à un déploiement général des activités économiques.

–– o ––

Quelques références bibliographiques

Amor-Iglesias, Juan-José, Jesùs M. González-Barahona, Gregerio Robles-Martinez et Israel Herráiz-Tabernero, 2005, « Mesuring libre software using Debian 3.1 (Sarge) as a case study : preliminary results », june 2005, *Upgrade Magazine*.

Behlendorf, Brian, Scott Bradner, Jim Hamerly, Kirk McKusick, Tim O'Reilly, Tom Paquin, Bruce Perens, Eric Raymond, Richard Stallman, Michael Tiemann, Linus Torvalds, Paul Vixie, Larry Wall et Bob Young, 1999, *Tribune libre, Ténors de l'informatique libre*, Editions O'Reilly, Paris.

Besen, Stanley, Leo J. Raskind., 1991, « An Introduction to the Law and Economics of Intellectual Property », *Journal of Economic Perspectives*, 5, pp. 3-27.

Breyer, Stephen, 1970, "The uneasy case for copyright : a study of copyright in books, photocopies, and computer programs", *Harvard Law Review*, 84(2), december, pp. 281-351.

Brooks F. P., 1996, *Le mythe du mois-homme. Essais sur le génie logiciel*, International Thomson Publishing, Paris.

Browne C. B., 1998, « Linux et le développement décentralisé », trad. fr. de Sébastion Blondeel et Gaël Duval sur linux-france : < http://www.linux-france.org/article/these/lsf-fr/lsf-fr_monoblock.html >.

Carey, Henry Charles, 1853, *Letters on international copyright*, A. Hart, 2ème édition 1868, Philadelphia.

Coris, Marie, 2002, « Les sociétés de services en logiciels libres : l'émergence d'un système de production alternatif au sein de l'industrie du logiciel », *IFREDE-E3I, Document de travail, n° 2002-1*, Université Montesquieu, Bordeaux.

Durkheim, Emile, 1893, *De la division du travail social*, réédition PUF, 1978, Paris.

Franke, Nikolaus et Eric Von Hippel, 2003, « Satisfying heterogeneous user needs via innovation toolkits : thhttp://www.linux-france.org/article/these/e case of Apache security software », *Research Policy,* 32, pp. 1199-1215.

Gosh, Rishab Aiyer et Paul David, 2003, « The nature and composition of the Linux kernel developer community, a dynamic analysis », < http://dxm.org/papers/licks1/licksresults.pdf >.

Ghosh, Rishab Aiyer, éd., 2006, *Study on the Economic impact of open source software on innovation and the competitivness of the Information and Communication Technologies (ICT) sector in the EU*, UNU-MERIT, the Netherlands.

< http://ec.europa.eu/enterprise/sectors/ict/files/2006-11-20-flossimpact_en.pdf >.

Godfrey, Michael W. et Qiang Tu, 2000, « Evolution in Open Source Software: A Case Study », *Proceedings ICSM,* pp. 131-142.

González-Barahona, Jesús, M., Ortuno-Perez, Pedro de las Heras Quirós, José Centeno González, José Centeno González, Vicente Matellán Olivera, 2001, « Counting potatoes: the size of Debian 2.2 », décember 2001, *Upgrade Magazine*.

Hardin, Garrett, 1968, « The tragedy of commons », *Science*, 13 décembre.

Healy ,Kieran, Schussman Alan, 2003, « The ecology of open-source software development », University of Arizona, january, *Document de travail.* < kieranhealy.org/files/drafts/oss-activity.pdf >.

Horn, François, 2004, *L'économie des logiciels*, La Découverte, collection « Repères », Paris.

Kitch, Edmund W., 1977, « The nature and function of the patent system », *Journal of Law & Economics*, 20, pp. 265-290.

Koch, Stefan, Schneider Georg, 2000, « Results from software engineering research into open software development projects using public data », Vienne, *Document de travail.*

Koch, Stefan, 2005, *Free/Open source software development*, Idea Group Publishing.

Krishnamurthy, Sandeep, 2005, « Cave or community ? An empirical examination of 100 mature open source projects », *First Monday* (10).

Kuwabara, K., 2000, « Linux: a Bazaar at the edge of chaos », *First Monday*, 5(3).

Landes, William M. et Richard Posner, 1987, « Trademark law : An economic perspective », *Journal of Law and Economics*, Vol 30 (2), pp. 265-309.

Landes, William M. et Richard Posner, 1989, « An Economic Analysis of Copyright Law », *Journal of Legal Studies*, Vol. XVIII (2), pp. 325-363.

Landes, William M. et Richard Posner, 2002, « Indefinitely Renewable Copyright », University of Chicago Law & Economics, *Olin Working Paper*, No. 154, pp. 1-41.

Machlup, Fritz, 1958, *An economic review of the patent system*, Study of the Subcommittee on Patents, Trademarks, and Copyrights, n° 15, US Government Print Office, Washington.

Marx, Karl, 1872-1875, *Le Capital, critique de l'économie politique*, édition française, ré-impression Editions Sociales, 1950, Paris.

Mockus, Audris, Roy T. Fielding, et James D. Herbsleb, 2002, « Two case studies of open source software development: Apache and Mozilla », *ACM Transactions on Software Engineering and Methodology,* 11, 3, pp. 309-346. (pdf)

Moon, Jae Yun et Lee Sproull, 2000, « Linux kernel, distributed work », *First Monday*, 5 (11).

Narduzzo, Alessandro et Alessandro Rossi, 2003, « Modularity in Action : GNU/Linux and Free/Open Source Software Development Model Unleased », *Document de travail.*

Nuvolari, Alessandro, 2005, « Open source software development: some historical perspectives », *First Monday*, 10, (10), 3 October.

Orbiten, 2000, « The Orbiten free software survey », first edition, < http://orbiten.org/ofss/01.html >.

Plant, Arnold, 1934, "The economic aspects of copyright in books", *Economica*, 1(2), p. 167-195.

Raymond, Eric, 1998, « The Cathedral and the Bazaar », *First Monday*, 3 (3).

Raymond, Eric, 1999, « The Magic cauldron », juin, *Document de travail.*

Reis, Robottom Christian et Renata Pontin de Mattos Fortes, 2002, *An overview of the software engineering process and tools in the Mozilla project*, *Proceedings of the Open Source Software Development Workshop*, 8 février, pp. 155-175.

Robles, Gregorio, Hendrik Scheider, Ingo Tretkowski, Niels Weber, 2001, « Who is doing it? A research on libre software developers », Berlin, *Document de travail.*

Rosenberg, Nathan, 1982, *Inside the Black Box : Technology and Economics*, Cambridge University Press.

Sagot-Duvauroux, Jean-Louis, 2006, *De la gratuité*, Editions de l'Eclat, Paris.

Salus, Peter H., 2008, *The Daemon, the Gnu, and the Penguin*, Reed Media Sarvices.

Shapiro, Carl et Hal R. Varian, 1999, *Economie de l'information, guide stratégique de l'économie des réseaux*, De Boeck Universités, Bruxelles.

Von Hippel, Eric, 2002, « Open source projects as horizontal innovation networks – by and for users », *MIT Sloan School of Management Working Paper,* n° 4366-02 juin.

Watt, Richard, 2000, *Copyright and Economic Theory: Friends or Foes ?*, Edward Elgar, Cheltenham, UK.

Wheeler, David A., 2004 (révisé en 2011), « The Linux Kernel: It's Worth More! »,

< http://www.dwheeler.com/essays/linux-kernel-cost.html >.

Yamagata, Hiroo, 1997, *Le pragmatiste du logiciel libre : entretien avec Linus Torvalds*, trad. fr. Blondeel

< http://www.linux-france.org/article/these/interview/torvalds/pragmatist-fr.html >.

Annexes

A – Le site Framasoft et son annuaire de logiciels libres

B – Les moteurs de rendu (et le ré-emploi du code)

C – Les *forks* d'OpenOffice.org

D – Evaluations statistiques dans l'étude FLOSS de 2006

E – Von Hippel, une analyse des développements logiciels communautaires comme « réseaux d'utilisateurs-producteurs d'innovations »

F – Handycam, un patchwork de « droits de propriété intellectuelle »

Annexe A : Framasoft et son annuaire de logiciels libres

Framasoft est une association (loi de 1901) issue du monde éducatif, à partir d'un réseau de sites web ayant comme point commun le logiciel libre et sa philosophie. La particularité de l'association est de viser le grand public et les utilisateurs ordinaires des ordinateurs et des programmes, en encourageant ces utilisateurs finaux à migrer du système d'exploitation de Microsoft vers le système GNU-Linux, et en les aidant aussi à remplacer les logiciels propriétaires par des logiciels libres équivalents. Cette association est financée classiquement par des cotisations et des dons ; elle fonctionne avec des bénévoles et aujourd'hui deux emplois salariés permanents, le dernier ayant été créé l'été 2012. C'est aussi un local, 8 serveurs, 30 applications web déployées, et un trafic conséquent (un million de visites par mois sur l'ensemble du réseau).

Le site est apparu pour la première fois en novembre 2001, comme simple page perso, à l'initiative d'un professeur de mathématiques Alexis Kauffmann, suite à un projet interdisciplinaire avec une professeure de lettres Caroline Atabekian, les deux enseignant en Seine Saint-Denis. L'association ne fut créée qu'en 2004 et Framasoft devient alors un véritable projet collaboratif. Le service le plus connu de Framasoft est son annuaire des logiciels libres[1], qui a grandi progressivement et où l'on trouve plus de 1600 logiciels répartis dans une centaine de rubriques et sous-rubriques, et couvrant les domaines les plus variés. On peut trouver aussi sur ce site des tutoriaux (une centaine environ), un projet d'applications portables (Framakey), un blog, un espace de discussion (forum Framagora), et une équipe de traducteurs.

Composition de l'annuaire par système d'exploitation*			
	mai 2009	sept 2012	
Nombre total de logiciels	1451	1611	[%]
disponibles pour :			
MSWindows	1228	1345	83,5
Linux	1042	1185	73,5
Mac OS X	657	740	45,9
non disponibles :			
MSWindows		266	16,5
Linux		426	26,4
Mac OS X		871	54
*note : certains logiciels sont disponibles pour plusieurs systèmes			

Tous les logiciels de l'annuaire sont des logiciels libres, en très grande majorité sous licence GNU-GPL ou GNU-LGPL (plus de 80 % au total), et aussi, bien que cela ne soit pas une règle systématique, des gratuiciels, avec un accès à des versions binaires, installables directement sur tel ou tel système d'exploitation. Les décomptes réalisés par moi en mai 2009 et septembre 2012 montrent bien l'orientation prioritaire du site Framasoft vers les utilisateurs de MSWindows. Ainsi, 83,5 % des logiciels présents dans ce répertoire en 2012 peuvent être utilisés dans Windows, un % qui dépasse celui de Linux. On voit aussi la variété et l'importance de l'offre en matière de logiciels libres à laquelle les utilisateurs du système de Microsoft peuvent accéder. Notons même que certains de ces programmes n'ont été produits que pour les utilisateurs de Windows et n'existent que pour ce système.

1 < http://www.framasoft.net/rubrique2.html >

Annexe B : Les moteurs de rendus, un exemple de réutilisation logicielle et de création de variétés pour les utilisateurs finaux

(1) Moteurs de rendus et navigateurs		
Moteur de rendu	**Navigateurs (et développeurs)**	
Gecko (1998) licences : MPL ou GNU-GPL ou GNU-LGPL, développé par la *Fondation Mozilla*	**Mozilla Firefox** (tous systèmes) MPL ou GPL ou LGPL	*Fondation Mozilla*
	Camino (OS X) MPL ou GPL ou LGPL	*The Camino Project*, soutenu par la *Fondation Mozilla*
	K-Meleon (Windows) GPL	Christophe Thibault (2000) et d'autres...
	Epiphany (Linux Gnome) GPL	communauté Gnome (depuis 2009, Epiphany repose sur WebKitGTK+)
	Kazehakase (Linux) GNU-GPL	Hiroyuki Ikezoe et Takuro Ashie (2008)
KHTML (2000) (GNU-LGPL) un élément du projet KDE	**Konqueror** (Linux KDE, MSWindows, OS X) GNU-GPL	communauté KDE (peut depuis août 2010 utiliser WebKit en place de KHTML
WebKit (2002) (GNU-LGPL et BSD) développé par *Apple Inc* [à partir de KHTML et KJS du *projet KDE*]	**Safari** (OS X, Windows, iOS) lic. propriétaire	Apple
	OmniWeb (OS X) lic. propriétaire	*OmniGroup* (développé à l'origine pour NeXTStep, partagiciel, puis gratuiciel depuis 2009)
	Midori (Linux Xfce) LGPL	Christian Dywan (utilise WebKitGTK+)
	Epiphany (depuis sept. 2009) GPL	voir plus haut
	GoogleChrome (Windows, OS X, Linux, Android, iOS) lic. propriétaire	*Google* (apparu en 2008)
	Chromium (Linux FreeBSD, OpenBSD OS X, iOS, Android) BSD, MIT, LGPL, Ms-PL, MPL/GPL/LGPL	projet de navigateur libre sur lequel GoogleChrome est basé (code source à compiler)
	SRWare Iron (Windows) licences opensource	*SRWare* basé sur Chromium, supprime la collecte des données personnelles de Google Chrome
	Arora (Linux, UNIX, FreeBSD, Mac OS X, Windows) GPL	Benjamin C. Meyer utilise QtWebKit

(2) Autres utilisations		
Moteur	**Logiciels (type et développeurs)**	
Gecko	**Thunderbird** (multiplate-forme) MPL ou GPL ou LGPL	logiciel de messagerie issu du projet Mozilla, *Mozilla Corporation*
	SeaMonkey (multiplate-forme) MPL ou GPL ou LGPL	suite internet (navigateur, client couuriel, agenda, tchat, éditeur web) issu du projet Mozilla, *Le Conseil SeaMonkey*
	Songbird (multiplate-forme) GPL	lecteur audio, gestion bibliothèque musicale, navigateur web *Pioneers of the Inevitable (POTI Inc)*
	KompoZer (multiplate-forme) MPL ou GPL ou LGPL	éditeur html (suite à NVU, lui même basé sur Mozilla Composer) Fabien Cazenave
	BlueGriffon (multiplate-forme) MPL ou GPL ou LGPL	éditeur html (suite de NVU) Daniel Glazman
WebKit	**Iweb** (OS X) lic. propriétaire	création de site web (de iLife) *Apple*
	Mail (OS X, iOS) lic. propriétaire	logiciel de messagerie de Mac OS X *Apple*
	Dashboard (OS X) lic. propriétaire	logiciel de widgets de Mac OS X *Apple*
	RapidWeaver (OS X) propriétaire et commercial	création de site web *RealMac Software*
	Sandvox (OS X) propriétaire et commercial	création de site web *Karelia Software*
	TextMate (OS X) propriétaire et commercial	éditeur de texte (code) Allan Odgaard
	Yelp (systèmes Unix) GPL	visionneur d'aide sous Gnome Gnome Foundation
	Liferea (Linux, BSD, Solaris) GPL	lecteur de flux RSS projet open source
	Steam (Windows, OS X, Linux, iOS, Android) lic. Propriétaire (freeware)	plate-forme de distribution de contenu en ligne (principalement des jeux vidéos) *Valve Corporation*

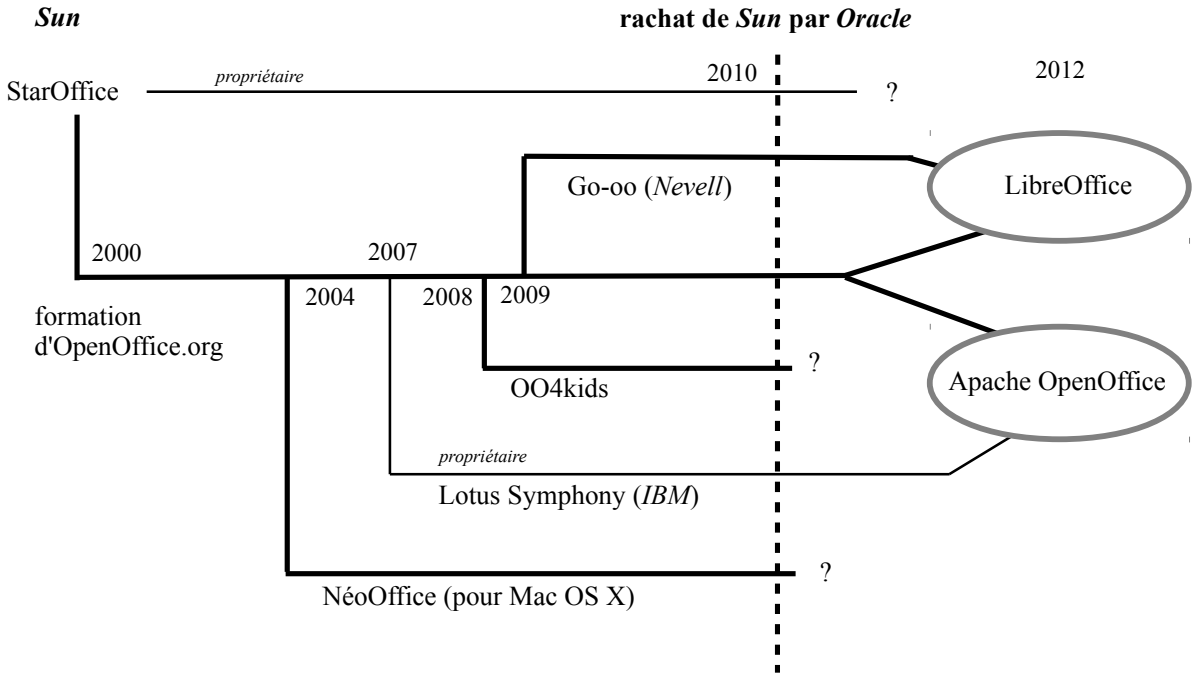
(3) Les moteurs de rendus propriétaires		
Moteur	**Navigateurs**	**Autres produits**
Presto** 2003 *Opera Software*	Opera (tous systèmes) Nintendo DS Browser (navigateur lié à la Nintendo DS) Opera Wii (navigateur lié à la Nintendo Wii)	Macromedia Dreamweaver (éditeur de sites web WYSIWYG) de Adobe Systems Adobe creative suite 2 (suite logicielle de graphisme professionel)
Trident** 1997 *Microsoft*	Internet Explorer (V4 etc.) AOL Explorer Yahoo Explorer Maxthon (voir plus haut)	Windows Explorer (explorateur de fichier) Windows Help (aide de Windows) Outlook (messagerie) Windows Media Player (lecteur multimédia)

Annexe C : Les *forks* d'OpenOffice.org

Un peu d'histoire. En 1999, *Sun* rachète une société allemande *Star Division* qui, au cours des années 1980, a développée une suite bureautique Star office. Pour *Sun*, c'est une décision stratégique. Avoir une suite bureautique sous Solaris, son propre système d'exploitation, est en effet indispensable ; cet achat permet de plus à l'entreprise d'économiser les licences payées à *Microsoft* pour MSWindows et MSOffice, ce qui rentabilise l'opération en quelques années.

Un an plus tard, *Sun* « libère » le code sous le nom d'OpenOffice.org, le plaçant sous une double licence GNU-GPL et SISSL. Un développement open source est mis sur pied; c'est OpenOffice.org. Les contributeurs doivent cependant accepter de transférer leur copyright à *Sun*, ce qui permet à cette société de maintenir, parallèlement au projet open source, un développement pour sa suite StarOffice propriétaire vendue aux entreprises. Certaines fonctionnalités introduites dans StarOffice ne sont d'ailleurs pas intégrées dans OpenOffice.

OpenOffice est la seule suite disponible pour Linux, et pour les autres systèmes Unix; Microsoft Office n'existant que pour Windows et Mac. Sa gratuité est de plus un atout indéniable pour les particuliers, les entreprises et les administrations. Des communautés importantes d'utilisateurs se constituent alors langue par langue, avec de nombreux contributeurs, des programmeurs certes, mais aussi des traducteurs, ou des producteurs d'aides et tutoriaux[2]. La communauté francophone est alors particulièrement active.



Les premiers *forks*

De 2000 à 2010, le développement d'OpenOffice reste sous le contrôle de *Sun*, et les communautés d'OpenOffice ne sont pas réellement indépendantes. Le développement du logiciel n'est d'ailleurs pas très rapide, ce qui suscite mécontentement et critiques. La suite bureautique acquière alors une réputation (bien justifiée) de lenteur au démarrage. Le poids en mémoire est jugé disproportionné, et l'interface jugée vieillissante. La gestion du projet est aussi mise en cause, en particulier le filtrage par *Sun* des contributions externes, et le frein à l'intégration de certaines

2 Notons qu'OpenOffice.org est un des plus gros projets open source, avec plus de 200 modules, 35000 fichiers, et plus de 5 millions de lignes de code.

contributions et innovations. Tout cela va donner naissance, avant même le rachat par Oracle, à plusieurs embranchements différents, aux causes multiples : création de variétés pour des publics différents, production d'une nouvelle suite propriétaire, ou contestation de la conduite du développement par *Sun*.

• NeoOffice pour Mac OS X : En 2004, le projet NeoOffice/J voit le jour. Il s'agit de produire à partir du code d'OpenOffice, en utilisant le langage java, une version pleinement intégrée à l'interface Aqua de Mac OS X. Un projet parallèle, NeoOffice/C, utilisant directement le langage cocoa de l'interface des macs est bientôt abandonné. NeoOffice/J se transforme alors en NeoOffice, disponible au téléchargement à partir de 2004 pour les processeurs Power PC (puis plus tard pour les processeurs Intel). NeoOffice se veut résolument un logiciel libre, étant diffusé sous la seule licence GNU-GPL avec clause copyleft. Ce programme, développé par Patrick Luby et Edward Peterlin, de *Planamesa Software*, existe toujours et son développement continue, avec corrections de certains bogues d'OpenOffice et introduction de fonctionnalités propres[3].

• Lotus Symphony (IBM) : En 2007, *IBM* crée un embranchement propriétaire, en combinant le code du cœur de la suite bureautique avec une nouvelle interface. C'est Lotus Symphony, disponible pour Linux, Mac OS X et MSWindows, un logiciel sous licence propriétaire, mais distribué sans frais. Lotus Symphony est intégrée à l'offre bureautique et logicielle d'*IBM* en direction des entreprises, sur Linux en particulier. L'objectif étant pour *IBM* bien évidemment de se passer de MSOffice (et MSWindows).

• Go-oo, un *fork* de Novell) : En 2008, exaspéré et même frustré par les difficultés à obtenir de *Sun* l'ajout au logiciel de modifications importantes et la non correction de défauts, inquiet aussi devant les difficultés financières de la firme et la réduction de son investissement dans OpenOffice[4], Novell lance son propre embranchement. C'est Go-oo, qui comprend de nombreuses nouvelles fonctions, intègrant en particulier tous les patchs d'amélioration disponibles jusqu'ici uniquement dans la distribution Suse. Le projet Go-oo, sous licence GNU-LGPLv.3, prône aussi un modèle plus ouvert que Sun. Il n'exige pas l'assignation du copyright, mais à l'inverse critique cette pratique : « *Nous pensons que cette assignation à une seule entité commerciale ouvre la porte à des abus envers les intérêts de la communauté des développeurs. Nous préférons une propriété collective (cf. Mozilla, GNOME, KDE, Linux) ou une fondation indépendante et méritocratique (sf. Eclipse, Apache) pour détenir les droits* »[5].

• OOo4kids et OOoLight : A l'initiative de la partie francophone du projet OpenOffice.org Education, en 2009, une version fortement allégée d'OpenOffice[6], avec une interface simplifiée (et rajeunie), est lancée en direction du monde éducatif. C'est OOo4kids, sous licence GNU-LGPL v.3. Le tableur est ainsi repensé pour être utilisable par des écoliers de 7 à 12 ans, et l'interface est réglable via les préférences du logiciel selon niveaux de difficulté. Des polices d'écritures cursives ont été ajoutées (sous licence OFL), etc. Un système d'annotation des présentation inventé (et repris par la suite dans Openoffice). Pour orienter le développement, les avis de professionnels de l'éducation sont régulièrement recueillis. A noter que ce projet a donné naissance plus récemment à une version de la suite destinée aux adultes, mais tout aussi légère, OOoLight (sortie en juin 2012).

Le rachat de *Sun* par *Oracle* et l'éclatement

En 2009, Oracle rachète Sun et hérite d'une certaine manière de son modèle économique dans le logiciel, et de tous les accords et liens tissés avec les communautés open source. Mais

3 La suite OpenOffice.org n'est au départ disponible pour Mac qu'avec X11. A partir de 2008 (Tiger) et l version 3 d'OpenOffice, il existe (en plus de NeoOffice) une version native sous Aqua, l'interface de Mac OS X.

4 Cf. les déclarations de Michael Meeks, en charge d'OpenOffice chez *Novell*, au quotidien allemand *Der Standard*. <http://derstandard.at/1216917892794/Sun-dropping-out-of-OpenOfficeorg-development-wouldnt-be-an-entirely-negative-thing >

5 Cf. <http://go-oo.org/> et <http://linuxfr.org/news/go-oo-une-alternative-à-openoffice>.

6 Le module Base (base de données) a été supprimé, ainsi que le support des langages Java et basic; un allègement d'environ 40 %.

Oracle, qui de nombreuses fois s'est montrée un ennemi acharné du logiciel libre, est strictement incapable de gérer cette situation. Les différents projets open source (OpenSolaris, MySQL, OpenOffice.org) sont donc condamnés à s'émanciper ou à disparaître.

• LibreOffice : En 2010, inquiets, des membres de la communauté OpenOffice créent donc la Ducument Foundation, une fondation allemande sans but lucratif, dont le but proclamé est de poursuivre le développement d'OpenOffice, sans dépendre d'Oracle. Comme le nom OpenOffice.org appartient maintenant à Oracle, la fondation demande à Oracle de lui céder cette marque, mais la firme refuse. La nouvelle suite s'appellera donc LibreOffice, avec comme licence la GNU-LGPLv.3. Le changement de nom est forcé par les circonstances, mais symbolique, marquant bien la fin d'une époque et la transition vers un développement réellement communautaire, piloté par une fondation indépendante, que les entreprises peuvent soutenir, mais qui dans son fonctionnement doit rester méritocratique[7]. Le projet Go-oo se rallie alors à LibreOffice et fusionne avec lui. La *Free Software Foundation* et l'*Open Source Initiative* apportent leur soutien, ainsi que plusieurs éditeurs de distributions GNU/Linux comme *Canonical* (avec Ubuntu), *Novell* (avec Suse), *RedHat* (avec RHEL, et Fedora), mais aussi *Google*, Gnome, NeoOffice, l'AFUL, l'APRIL, etc. L'engouement des partisans et des communautés du logiciel libre est réel. Une première version stable de LibreOffice, toutes langues, sort en janvier 2011 (version 3.3[8]), le développement et la chasse aux bogues s'organisent, avec une sortie régulière de versions améliorées. En octobre 2011, c'est l'annonce d'un projet pour porter LibreOffice sur Android et une première démonstration de LibreOffice Online, une version web de la suite. En Août 2012, une nouvelle version 3.6 est annoncée, en 104 langues, avec de nombreuses améliorations. Le projet revendique alors 500 développeurs ayant fourni de nouvelles fonctionnalités et des patchs depuis l'annonce du projet le 28 septembre 2010.

• Apache OpenOffice : Peu de temps après l'annonce de la création de The Document Foundation, en juin 2010, *Oracle* sous pression d'IBM, fait don à la Fondation Apache du code d'OpenOffice.org, ainsi que de la marque et des noms de domaine. OpenOffice.org entre alors dans « l'incubateur » d'Apache, une procédure d'accueil des dons de code et des nouveaux projets open source. Les contributeurs sont pris en charge par un mentor qui leur inculque les principes de la communauté Apache et des processus de développement collaboratif et de gouvernance méritocratique. Cela implique aussi un changement obligatoire de marque et un examen légal du code avec changement de licence. Le nouveau logiciel s'appellera donc Apache OpenOffice (et non plus OpenOffice.org) et la licence sera la licence Apache, une licence sans copyleft, à la différence de la GNU-LGPL, qui autorise l'intégration du code source dans des logiciels propriétaires[9]. Les

---

7 Cf. cette déclaration sur le site de LibreOffice : « *The Document Foundation* [*TDF*] *est une organisation ouverte, indépendante, autonome et méritocratique, qui s'appuie sur dix années de travaux par la communauté OpenOffice.org. TDF a été créée avec la conviction que la culture née d'une fondation indépendante fait ressortir le meilleur des contributeurs professionnels et bénévoles, et produira la meilleure suite bureautique libre. TDF est ouverte à toute personne qui est d'accord avec ses valeurs fondamentales et contribue à ses activités. TDF accueille chaleureusement la participation des entreprises, par exemple, par le parrainage de contributeurs individuels pour travailler d'égal à égal aux côtés d'autres contributeurs de la communauté* ».

8 La numérotation des versions reprend celle d'OpenOffice.org, la version 3.3 de LibreOffice étant dérivée de la dernière version du tronc commun (version 3.2). Elle n'est d'ailleurs guère différente de la version 3.3 d'OpenOffice.org, distribuée avec la marque d'*Oracle*.

9 Voir la réaction et l'analyse de la *Free Software Foundation* : « *All Apache projects are distributed under the terms of the Apache License. This is a non-copyleft free software license; anybody who receives the software can distribute it to others under nonfree terms. Such a licensing strategy represents a significant policy change* (..)*. Previously, the software was distributed under the terms of the GNU Lesser General Public License (LGPL). The LGPL is a weak copyleft license, so programs that merely link to the software can be released under nonfree terms, but the software covered by the LGPL must always be released, along with its source code, under the LGPL's terms. Free software developers are clearly comfortable with a partial copyleft when it's appropriate; in numerous surveys of free software projects, the LGPL is commonly listed as the second-most popular license (after the GNU General Public License), or else follows close behind. While we do recommend the Apache License in specific situations, we do not believe it is the best choice for software like OpenOffice.org. This situation calls for copyleft, because the gains free software stands to make from a non-copyleft license don't justify giving a handout to proprietary software*

dépendances externes du projet, sous licence GNU-GPL ou GNU-LGPL seront donc réécrites ou remplacées par des équivalents sous licence Apache. En juillet 2011, IBM annonce faire don à la fondation Apache du code de Lotus Symphony et lui confier son développement en s'engageant à soutenir t contribuer au développement d'Apache OpenOffice. Les deux codes devraient donc dans l'avenir fusionner. Une première version d'Apache OpenOffice, numérotée v. 3.4, est sortie en 15 langues sous licence Apache le 8 mai 2012.

Il y a donc aujourd'hui deux projets de développement, parallèles et concurrents, issus du même code, celui d'OpenOffice.org. Les deux suites bureautiques sont tous deux des logiciels libres, et les deux projets sont gérés par deux fondations formellement indépendantes et méritocratiques, mais ces deux fondation bénéficient de soutiens biens différents, qu'ils s'agissent des communautés ou des entreprises[10]. Les licences choisies sont aussi différentes, ainsi que la philosophie des deux projets. *The Document Foundation* a choisi la licence GNU-LGPL v.3 pour tout ce qui est hérité du projet OpenOffice.org (le copyright sur le code appartenant aujourd'hui à *Oracle*). Les nouvelles contributions seront elles mises sous une double licence, GNU-LGPL v.3 et MPL v.2, deux licences comportant une clause *copyleft*. Les auteurs conservent leur copyright, contrairement à la règle antérieure, l'assignation du copyright à *Sun*, une règle abandonnée donc. On peut noter que la licence Apache v.2 offre la possibilité aux développeurs de LibreOffice de reprendre éventuellement le code d'Apache OpenOffice, alors que le contraire n'est pas possible pour les développeurs d'Apache OpenOffice...

developers ». < http://www.fsf.org/news/openoffice-apache-libreoffice >

10 D'un côté *Oracle* et *IBM*, de l'autre les communautés du libre, ainsi que les distributions GNU/Linux, commerciales ou non, et des sociétés comme *Novell* et *Google*.

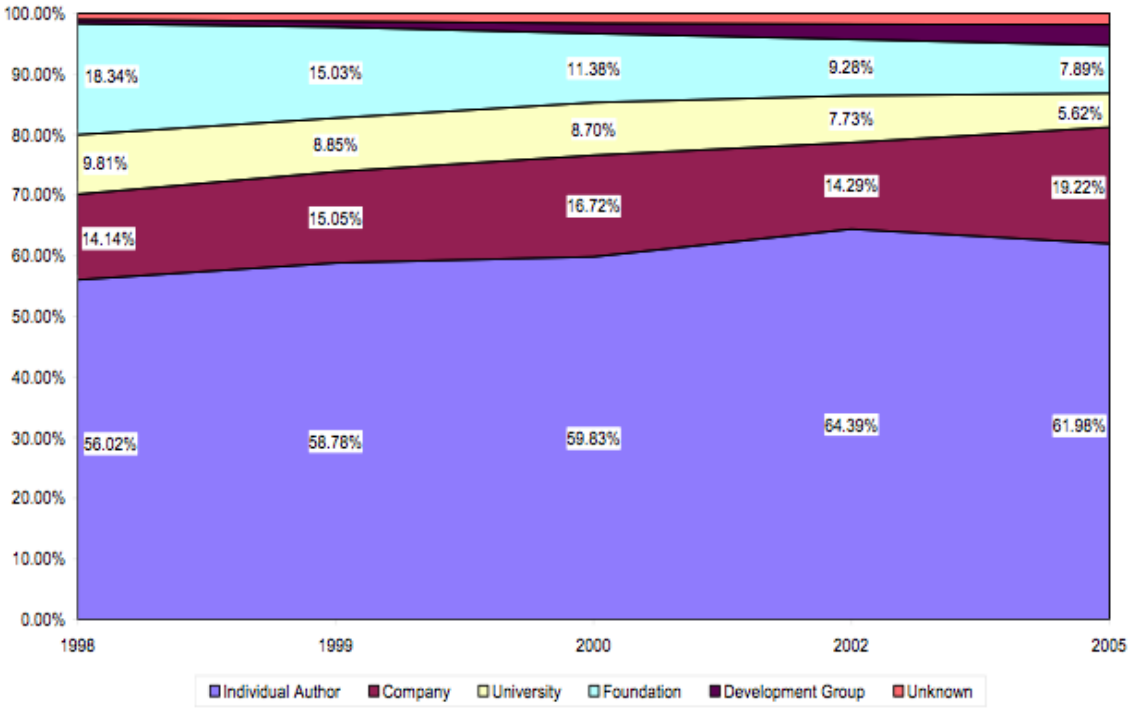
Annexe D : Estimations du coût de certains logiciels libres - étude FLOSS Impact publiée en 2006

Table 3: Production cost estimate for 5 largest FLOSS software products

Software package	Lines of code	Months	Person-months	Cost (million Euro)
openoffice.org	5181285	130	79237	482
kernel-source-2.6.8	4033843	160	145036	882
mozilla (firefox)	2437724	87	25339	154
gcc-3.4	2422056	113	54048	329
xfree86	2316842	90	27860	169

Copyright © 2006 MERIT.

Figure 28: Distribution of code output by individuals, firms, universities



Copyright © 2006 MERIT. Source: URJC. Shows code contribution share for the Debian collection.

Table 6: Estimated real effort and opportunity cost for the FLOSS codebase

Code base collection	Debian 3.1 FLOSS distribution (2005)
Source lines of code	221,351,503
Number of developers	17000 (estimate)
Estimated effort (*coding time only!*)	43 944 person years
Development cost estimate (*coding only!*)	Euro 2.67 billion
Development cost estimate, 5-year period	131 832 person years
Full-time employee equivalent (2006)	26 000

Copyright © 2006 MERIT.

Table 4: Contribution of FLOSS code by selected universities

Rank	Person-months	University / research institution
1	4955	Regents Of The University Of California
2	4774	Massachusetts Institute Of Technology
3	1687	Carnegie Mellon University
4	1340	University Of Chicago
5	1009	INRIA
6	982	University Of Amsterdam
7	870	Stichting Mathematisch Centrum Amsterdam
8	551	Ohio State University
9	518	University Of Utah
10	505	University Of Notre Dame

Copyright © 2006 MERIT. Shows estimated cumulative person-month contribution in Debian, by 2005

Table 5: Cost estimate for FLOSS code contributed by firms

Total contribution from firms			
Number of firms		986	
Source lines of code		31.2 million	
Estimated effort		16 444 person years	
Estimated cost		1.2 billion Euro	
Top contributors			
Rank	**Name**	**Person-months**	**Cost (mil euro)**
1	sun microsystems inc.	51372	312
2	ibm corp.	14865	90
3	red hat corp.	9748	59
4	silicon graphics corp.	7736	47
5	sap ag	7493	46
6	mysql ab	5747	35
7	netscape communications corp.	5249	32
8	ximian inc.	4985	30
9	realnetworks inc	4412	27
10	At&t	4286	26

Copyright © 2006 MERIT. Shows estimated cumulative substitution cost contribution in Debian, by 2006

Source : Ghosh, Rishab Aiyer, éd., 2006, *Study on the Economic impact of open source software on innovation and the competitivness of the Information and Communication Technologies (ICT) sector in the EU*, UNU-MERIT, the Netherlands.

Annexe E : Une analyse du développement logiciel communautaire, le réseau d'utilisateurs-producteurs d'innovations

L'économiste Eric Von Hippel (article de 2002 entre autres[11]) a livré une analyse intéressante des formes de production communautaire du logiciel libre en les interprétant comme des exemples particuliers de « *réseaux d'innovations horizontaux* » par des utilisateurs produisant eux-mêmes les innovations[12]. Von Hippel se réfère explicitement au développement du serveur web Apache et au système d'exploitation Linux.

Le logiciel Apache (Apache HTTP Server) est aujourd'hui le logiciel le plus utilisé par les hébergeurs et administrateurs de sites web (59,4 % de part de marché en 2012 selon Netcraft[13]). Il assure la gestion du stockage des sites et pages web, et l'interface entre le réseau et les différents utilisateurs (lecteurs comme producteurs de contenu), avec la nécessaire prise en charge de tous les problèmes de sécurité que cela suppose. L'origine du projet Apache est un serveur du *National Center for Supercomputing Applications*, dont le code avait été mis dans le domaine public. Un petit groupe d'utilisateurs (*web masters*) décidèrent d'en reprendre le développement, en en corrigeant les défauts et en ajoutant les nouvelles fonctionnalités qui leur semblaient nécessaires pour le bon fonctionnement de leurs propres serveurs. En 1991, ils diffusent une première version (Apache 1) du programme, dont le développement va continuer par intégration de contributions envoyées par d'autres administrateurs web qui l'utilisent. Un réseau se forme autour de la petite équipe qui assure le suivi du projet, composés de programmeurs tous intéressés à voir s'améliorer le produit qu'ils utilisent dans leur travail de *web masters*. Le programme n'a depuis cessé d'évoluer, produit de la même façon, en réponse aux problèmes rencontrés et aux demandes des utilisateurs. C'est un exemple parfait de ce que Von Hippel appelle un « *réseau horizontal d'innovation par et pour les utilisateurs* ».

Un tel réseau, nous dit Von Hippel, peut être entièrement indépendant du mode traditionnel de production des innovations séparé de l'usage, et ceci à trois conditions : (1) Certains innovateurs au moins sont incités à innover; (2) Certains utilisateurs innovateurs au moins sont incités à révéler leurs innovations; (3) La diffusion des innovations par les utilisateurs est peu coûteuse et peut concurrencer le mode traditionnel, commercial, de la production des innovations. Quand les points 1 et 2 sont seuls présents, le réseau des utilisateurs-innovateurs peut certes exister, mais la diffusion et l'adoption nécessitent une production et une distribution commerciale. Les études empiriques de telles situations et de tels réseaux montrent, ajoute Von Hippel, que les innovations sont souvent le fait d'un petit groupe d'utilisateurs fortement motivés à voir la technique en cause s'améliorer, car ils en bénéficient directement dans leur travail, leur activités de production, leur

11 Cf. « Horizontal innovation networks – by and for users », *MIT School of Management*, june 2002.

12 Il reprend ainsi le *learning by using* de Nathan Rosenberg, celui-ci ayant souligné l'importance dans de nombreuses industries des utilisateurs pour l'innovation, à l'inverse de l'analyse traditionnelle faisant de la production la seule source des innovations. Cf. Nathan Rosenberg, *Inside the Black Box: Technology and Economics*, 1982, ch 6, p. 140.

13 Voir : < http://news.netcraft.com/archives/category/web-server-survey/ >

formation, ou... leur pratique sportive[14]. On constate aussi que si les améliorations ont besoin pour être généralisées d'une diffusion marchande, celle-ci n'apparaît que plusieurs mois ou plusieurs années après l'expérimentation. L'innovation est donc supportée par le réseau sans qu'il y aît au départ et pendant longtemps un marché ou des des activités commerciales, la motivation principale étant la technique, son usage et son amélioration.

L'exemple d'Apache, cette auto-production collective tirée par l'usage, est assez facilement généralisable à l'ensemble des développements communautaires en logiciel libre. Reprenons les critères de Von Hippel : (1) L'incitation à innover existe bien chez les utilisateurs-programmeurs qui ont besoin de ce logiciel, et besoin que ce logiciel marche bien. (2) L'incitation à révéler l'innovation est un problème un peu plus compliqué, mais elle existe bien dans le cadre des licences dont les règles assurent que les révélations deviendront une sorte de propriété commune permettant à d'autres d'améliorer à leur tour le logiciel. Révéler l'innovation n'appauvrit pas l'utilisateur-programmeur, ne pas la révéler ne lui rapporte rien, et permettre l'évolution du programme et son amélioration progressive est bien plus rationnel, surtout quand la licence comporte une clause copyleft, qui garantit l'accès à toutes les améliorations ultérieures du logiciel. (3) La diffusion est de surcroît peu coûteuse, comme nous l'avons vu précédemment. Au niveau le plus élémentaire, celui du contributeur individuel, tout ce qu'il doit faire est nettoyer son code, ajouter la documentation, son copyright et la déclaration de licence, et envoyer le tout au mainteneur du module concerné. Le plus compliqué, le plus « coûteux » pour le programmeur, est le travail déjà accompli pour son propre usage. Au niveau global, les coûts privés sont finalement très faibles, et le bénéfice collectif très important, dans l'hypothèse où le développement donne effectivement naissance à un nouveau programme amélioré que tous peuvent utiliser.

14 Dans son texte de 2002, Von Hippel donne l'exemple des planches de surf, qui ont été mises au point et développées à l'origine par un petit groupe d'utilisateurs et sportifs passionnés partageant leurs innovations.

F – Handycam, un patchwork de droits de propriété intellectuelle

Extraits du mode d'emploi des caméscopes numériques *Handycam* de *Sony* (DCR-SR45/SR46/SR65/SR85), lesquels embarquent Linux, la C Library (GPL), Expat (MIT), Zlib (zlib), libjpeg, etc., mais aussi un portefeuille de brevets MPEG-2.

p. 97

À propos des marques de commerce

- « Handycam » et HANDYCAM sont des marques déposées de Sony Corporation.
- « Memory Stick », « », « Memory Stick Duo », « MEMORY STICK DUO », « Memory Stick PRO Duo », « MEMORY STICK PRO DUO », « Memory Stick PRO-HG Duo », « MEMORY STICK PRO-HG DUO », « Memory Stick Micro », « MagicGate », « MAGICGATE », « MagicGate Memory Stick » et « MagicGate Memory Stick Duo » sont des marques de commerce ou des marques déposées de Sony Corporation.
- « InfoLITHIUM » est une marque de commerce de Sony Corporation.
- Microsoft, Windows, Windows Media, Windows Vista et DirectX sont des marques de commerce ou des marques déposées de Microsoft Corporation aux États-Unis et dans d'autres pays et/ou régions.
- Macintosh et Mac OS sont des marques déposées de Apple Inc. aux États-Unis et dans d'autres pays et/ou régions.
- Intel, Intel Core et Pentium sont des marques de commerce ou des marques déposées de Intel Corporation ou de ses filiales aux États-Unis et dans d'autres pays et/ou régions.
- Adobe, le logo Adobe et Adobe Acrobat sont des marques de commerce ou des marques déposées de Adobe Systems Incorporated aux États-Unis et dans d'autres pays et/ou régions.

Tous les autres noms de produits cités dans le présent document peuvent être des marques de commerce ou des marques déposées de leurs sociétés respectives. En outre, les indications ™ et « ® » ne sont pas employées chaque fois qu'une marque est citée dans le présent Guide pratique.

Remarques concernant la licence

TOUTE UTILISATION DE CE PRODUIT POUR UNE UTILISATION AUTRE QUE PERSONNELLE, D'UNE MANIÈRE QUI SE CONFORME À LA NORME MPEG-2 VISANT L'ENCODAGE D'INFORMATION VIDÉO POUR LES MÉDIAS SOUS EMBALLAGE, EST EXPRESSÉMENT INTERDITE SANS UNE LICENCE CONFORME AUX BREVETS APPLICABLES DANS LE PORTEFEUILLE DE BREVETS MPEG-2, LADITE LICENCE ÉTANT DISPONIBLE AUPRÈS DE MPEG LA, L.L.C., 250 STEELE STREET, SUITE 300, DENVER, COLORADO 80206.

« C Library », « Expat », « zlib » et « libjpeg » sont des logiciels fournis avec votre caméscope. Ces logiciels sont fournis conformément aux licences d'utilisation des détenteurs des droits d'auteur. À la demande des détenteurs des droits d'auteur de ces applications logicielles, nous sommes dans l'obligation de vous communiquer les informations suivantes. Veuillez lire les sections suivantes.

Lisez le fichier « license1.pdf » dans le dossier « License » du CD-ROM. Vous y trouverez les licences (en anglais) des logiciels « C Library », « Expat », « zlib » et « libjpeg ».

À propos des logiciels sous licence GNU GPL/LGPL

Les logiciels soumis aux licences GNU General Public License (désignée ci-après par « GPL ») ou GNU Lesser General Public License (désignée ci-après par « LGPL ») sont inclus dans le caméscope.

Nous vous informons que vous avez le droit d'accéder au code source de ces programmes logiciels, de le modifier ou de le redistribuer aux conditions énoncées dans la licence GPL/LGPL fournie.

Vous trouvez le code source sur le Web. Consultez l'URL suivante pour le télécharger. Lors du téléchargement du code source, sélectionnez le modèle de caméscope DCR-DVD810.

http://www.sony.net/products/Linux/

Veuillez éviter de nous contacter concernant le contenu du code source.

Lisez le fichier « license2.pdf » dans le dossier « License » du CD-ROM. Vous y trouverez les licences (en anglais) des logiciels « GPL » et « LGPL ».

Pour visualiser le PDF, Adobe Reader est nécessaire. S'il n'est pas installé sur votre ordinateur, vous pouvez le télécharger depuis le site Web de Adobe Systems :

http://www.adobe.com/