Les communs du logiciel libre : La naissance du commun, le cadre institutionnel

Pierre-André Mangolte

Les communs du logiciel libre

Première partie :

La naissance du commun, le cadre institutionnel*

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

version 1 - décembre 2010

I. Introduction : Une définition du terme « commun »

Aujourd'hui, tout logiciel est soumis aux « droits de propriété intellectuelle » dès sa création; il s'agit en général du copyright (ou droit d'auteur). Cela signifie qu'il existe toujours un propriétaire individuel, être humain ou firme, qui détient des droits exclusifs sur ce programme (code-source ou copies binaires[1]), et que celui-ci peut en interdire ou en contrôler légalement l'usage. Lorsque les logiciels sont devenus des marchandises, au cours des années 1970-1980, et que la question de leur qualification juridique a été réellement posée pour la première fois, on a décidé que les modes de partage existant antérieurement devaient disparaître, et que l'objet technique logiciel serait approprié de manière individuelle et exclusive, systématiquement et obligatoirement. Mais cette utopie propriétaire s'est très vite révélée inadaptée et même totalement absurde, compte tenu des réalités de l'industrie informatique et de l'économie des logiciels. Il a fallu alors le remettre en cause, avec en particulier la ré-émergence, le développement, et l'institutionnalisation de formes de mise en commun et partage du code et des programmes, en complète opposition avec la solution retenue juridiquement. Nous pouvons donc parler ici des « communs » de l'économie des logiciels.

Plusieurs communs sont en effet identifiables actuellement dans cette économie. On en trouve au minimum trois : (1) Le commun (ou les communs) du logiciel libre (*free software*), auquel nous allons consacrer ce papier; car dans ce papier, nous ne traiterons que des communs du logiciel libre, un commun qui repose sur des licences spécifiques accordant à tous les utilisateurs les mêmes droits, et la plus grande liberté dans l'usage du code, plus généralement du logiciel. (2) Le commun des interfaces et des infrastructures ouvertes (*open*), créé dans le but de permettre l'inter-opérabilité des programmes entre eux, et plus généralement le bon fonctionnement des systèmes informatiques et des réseaux. On peut évoquer ici en particulier les « standards de l'internet » gérés par le W3C, ou les règles juridiques particulières, introduites par la suite, pour pallier ce problème. (3) Le commun des graticiels (*freewares*) enfin. On a là trois formes de mise

* ANR Propice – Janvier 2011 – WP 3 L 3.2

1 Pour une définition et quelques éléments techniques, voir l'Annexe A.

en commun et partage de ressources, trois formes analytiquement différentes. Elles diffèrent par leurs origines, la nature des ressources prises en compte, l'étendue des droits d'usage possibles, les règles juridiques qui les accompagnent (et les sécurisent), leur permanence ou non dans le temps, etc. Mais toutes sont une remise en cause pragmatique, imposée la plupart du temps par la nécessité et l'expérience, du principe individualiste propriétaire.

Mais avant d'entrer dans le vif du sujet, dans l'analyse d'une sorte d'expérience historique de formation de rapports « communistes » au cœur même du capitalisme, il est nécessaire de préciser un peu ce que nous désignons par le terme « commun », d'introduire donc une définition générale de ce terme. Nous entendons par *commun* un ensemble de ressources, plus ou moins pérennes, dont l'accès et l'usage sont partagés dans un groupe très étendu, voire l'humanité toute entière. Il s'agit donc d'une propriété commune en matière d'accès et d'usage qui existe *de facto*, sécurisée ou non par des dispositifs juridiques particuliers. Nos sociétés marchandes, capitalistes, sont construites en principe, selon du moins la croyance et les affirmations de la majorité des économistes et des juristes, sur la propriété privée, un principe de contrôle exclusif des ressources. Mais, dans ces sociétés, il y a aussi de nombreux ensembles de ressources qui sont, qu'on en ait pleinement conscience ou non, des *communs*; et ces communs sont d'ailleurs indispensables au bon fonctionnement d'un grand nombre d'activités privées, et au bon fonctionnement de l'économie marchande dans son ensemble. Ainsi les routes, les trottoirs des villes, les parcs et autres espaces ouverts au public, les bibliothèques publiques, les musées sont des formes de communs. On peut aussi évoquer la langue (la langue française par exemple), ou les idées et les informations, qui circulent librement, par impossibilité d'empêcher réellement leur diffusion et leur partage, mais aussi et surtout en vertu de règles constitutionnelles de nos sociétés démocratiques (la liberté de pensée et d'information par exemple)[2]. Le domaine public des œuvres littéraires est aussi une forme de commun. Il en est de même pour certains pools de brevets, quand leur seule finalité est de mettre en commun et de partager un certain nombre de techniques brevetées.

Pour préciser les choses, trois éléments interviennent toujours dans la définition d'un commun. Il faut (1) une ou des ressources, (2) une population, un groupe, une communauté, et (3) des règles et institutions qui permettent, organisent, et régulent l'utilisation de ces ressources par les membres de la communauté, une utilisation partagée, de manière collective ou individuelle, simultanée ou successive, une utilisation qui peut aller parfois – ce qui est le cas pour les logiciels libres – jusqu'à la mise sur pied d'une production en commun de ces ressources.

(1) Les ressources peuvent être des productions de dame Nature ou des productions

2 L'économie de l'information, contrairement à ce que certains économistes affirment, citant Kenneth Arrow (1962), est largement construite sur un principe de non propriété de l'information, une fois celle-ci dévoilée pour la première fois. C'est en particulier le cas aux Etats-Unis, une règle imposée par le premier amendement de la Constitution américaine (*free speech*), interprété traditionnellement par la Cour Suprême comme le droit pour chaque individu d'accèder à l'information. Cet amendement interdit la mise sous copyright de l'information, et même de nombreuses expressions (écrits, films, etc.) de celle-ci; et c'est sur ce principe, et à partir des communs informationnels qui en découlent, que le secteur de la presse et des médias a pris son essor et a prospéré aux Etats-Unis et ailleurs.

humaines. Leur nature peut être matérielle ou immatérielle, tangible ou intangible, et cette nature intervient alors dans la forme particulière prise par tel ou tel commun, mais ce n'est jamais cette nature qui détermine l'existence de celui-ci. Les communs ne sont donc pas simplement des *res communes* ou définis par l'impossibilité ou une difficulté extrême à établir ici une appropriation individuelle, une difficulté qui découlerait directement de certaines caractéristiques des ressources considérées[3]. C'est particulièrement évident pour les logiciels, puisqu'on trouve aussi bien des logiciels libres que des logiciels dits « propriétaires », et qui s'agit fondamentalement des mms objets techniques, remplissant les mêmes fonctions, et fonctionnant en général de la même façon.

(2) On peut compter dans les membres de la population, du groupe nécessaire au commun, aussi bien des êtres humains que des entreprises (personnes physiques et morales donc). Ici la chose la plus importante est sans doute la taille de ce groupe. Il y a des communs dont les membres peuvent être peu nombreux, et qui restent fermés aux autres (cas fréquent pour les pools de brevets); d'autres sont ouverts à tous et à vocation universelle. Les conditions d'entrée ou d'adhésion peuvent être donc fort diverses d'un commun à l'autre. Mais dans le cas qui nous intéresse, le commun des logiciels libres, la communauté est par principe l'ensemble de l'humanité, sans distinction particulière,et ceci même si la compétence de chacun en matière de programmation peut en pratique limiter l'usage éventuel que l'on fait de la ressource. Ce commun est à vocation universelle.

(3) Mais le plus important pour définir un commun est le troisième élément, sa base institutionnelle, constitutionnelle même, c'est-à-dire la ou les règles qui relient les membres de la communauté aux ressources. Ces règles font que les ressources n'appartiennent plus au domaine de jouissance exclusive de certains individus, mais sont à posées comme communes, ou partagées, en matière d'usage. Qu'il s'agisse de simples règles de fait, ou de règles formellement établies, codifiées éventuellement sous forme juridique, elles définissent l'accès de chaque membre du groupe aux ressources, et les utilisations possibles (ou proscrites) de celles-ci. Elles fixent donc le comportement de chacun vis-à-vis des ressources, et ont aussi comme conséquence de définir les relations que les membres de la population concernée peuvent et doivent entretenir entre eux dans le cadre du commun, quand ils utilisent les ressources, contribuant à transformer un groupe de fait en une communauté ayant conscience d'elle-même. Souvent, ces règles ont comme fonction d'assurer la pérennité de l'ensemble des ressources, et donc du commun. Elles peuvent aussi être définies pour servir de cadre à la production des ressources considérées. Elles peuvent limiter ou interdire une éventuelle appropriation (ou ré-appropriation) privée de celles-ci, ou autoriser cette appropriation, dans la mesure où cela ne remet pas en cause l'existence du commun. On peut parler ici de règles constitutionnelles; et dans le cas des logiciels libres, ces règles constitutionnelles

3 Contrairement à ce que postule l'analyse économique standard des droits de propriété intellectuelle, celle de l'Université de Chicago du moins. Sa caractéristique principale est en effet de poser comme seule limite au copyright et aux autres droits de la « propriété intellectuelle » le niveau des coûts de transaction, et plus particulièrement des coûts liés au maintien de l'exclusivité et au prélèvement des droits. Sur ce point voir Landes et Posner (1989 et 2002).

sont les différents systèmes de licence (GNU-GPL, BSD, etc).

Dans la suite de ce papier, nous allons retracer l'évolution qui a conduit à l'innovation institutionnelle que représente le logiciel libre (*free software*) et à la mise en place d'un pool de ressources logicielles librement accessibles et partagées, sur un mode d'ailleurs universel (points II et III). Nous analyserons ensuite plus en détail les règles constitutionnelles du commun, c'est-à-dire les différents types de licences; et les règles spécifiques protégant ces ressources d'une re-privatisation éventuelle, rendant ainsi le commun pérenne (points IV et V).

Dans une deuxième partie, qui n'est pas disponible à l'heure actuelle, n'étant pas rédigée (un futur « livrable »), il est prévu de traiter de la « vie du commun » et de son intégration dans l'industrie informatique.

II. Le logiciel, une valeur d'usage transformée en marchandise

Le logiciel est dépendant dès leur création d'un système de droits de propriété intellectuelle exclusifs, imposé de manière absolue et obligatoire, une situation qui le différencient des autres objets techniques. En règle générale en effet, les connaissances et les artefacts techniques évoluent tous dans des systèmes de règles et de droits qui la plupart du temps peuvent être considérés comme non exclusifs; le contrôle éventuel de la technique par un individu ou une firme n'étant que partiel et temporaire. C'est le régime du secret avant divulgation, et le domaine public après divulgation. Seules les innovations brevetées échappent à ce cadre. En pratique donc, en dehors des industries nouvelles, émergentes, reposant sur quelques innovations mises sous brevet, ce régime général intègre une forme de commun (le domaine public), où les connaissances et les techniques sont librement utilisables et réutilisables pour toute activité industrielle et commerciale, ce qui est d'ailleurs une condition de la mise en concurrence dans les activités économiques. Mais les innovations logicielles, le stock même des logiciels existants, relèvent, en dehors des logiciels libres, d'un régime différent, un régime d'utopie propriétaire où tous les produits sont attribués, avec des droits exclusifs, à des personnes, physiques ou morales, des êtres humains ou des entreprises, juridiquement considérées comme des « auteurs ».

La situation était pourtant complètement différente au début de l'industrie informatique; personne ne se posant alors réellement la question du statut juridique des programmes destinés aux ordinateurs, à une époque où ceux-ci étaient considérés avant tout comme des valeurs d'usage et non comme des valeurs d'échange. Jusqu'au milieu des années 1960 en effet, les logiciels ne sont jamais vendus directement. Les constructeurs d'ordinateurs, au premier rang IBM, fournissent « gratuitement » le code avec les matériels, loués ou vendus. Il s'agit principalement du système d'exploitation, indispensable pour toute utilisation de l'ordinateur, ou de logiciels libres permettant la traduction des langages de programmation en langage machine. De leur côté, les utilisateurs développent eux-mêmes les logiciels d'application dont ils ont besoin. Une autre possibilité est la production par une société de service, le logiciel étant développé en réponse à une demande

spécifique, celle du client, en interaction étroite avec celui-ci. Le logiciel n'est jamais réellement vendu à l'époque en tant que tel, mais fournis plutôt dans le cadre d'une autre relation économique : vente de matériel ou relation de service. C'est d'ailleurs la pratique chez IBM, dont la politique officielle est : « *Nous vendons des services et non des matériels* »; cette entreprise facturant des prestations globales en fonction du « *service rendu par l'ordinateur* », et non en fonction de ses propres coûts (Dréan, 1996).

Parallèlement à la vente de services, IBM encourage et organise même le partage du code. Dans les années 1950-1960, les pratiques de coopération en matière de développement logiciel entre utilisateurs des mêmes machines sont en effet courantes et répondent d'ailleurs à un véritable besoin. Pour les scientifiques et les chercheurs, c'est une sorte de norme, et pour l'ensemble des firmes, une habitude qui remonte à l'époque de la mécanographie[4]. IBM fournit ainsi un « *soutien logiciel* » aux utilisateurs de ses ordinateurs, c'est-à-dire des programmes et une formation à la programmation. Les ingénieurs et commerciaux d'IBM collectent alors tout naturellement les sous-programmes (en langage d'assemblage à l'époque) qu'ils développent chez les clients, et éventuellement ceux que ces clients ont développé eux-mêmes, pour les redistribuer ensuite. Les coûts de développement et les résultats sont ainsi partagés; les machines peuvent être utilisées de manière plus efficace, et la firme IBM comme les utilisateurs y trouvent leur compte.

On peut donner un exemple ici, le programme SHARE, mis sur pied lors du lancement de l'IBM 704 en 1954. Ce nom n'était pas une abréviation, mais affirmait simplement Son objectif explicite était de partager son objectif, partager (*to share*) les informations et les programmes. Il s'agissait aussi de favoriser la communication entre les utilisateurs et les ingénieurs d'IBM, qui préparaient déjà les futurs produits de la firme. les économies attendues étaient considérables. On estimait que chaque utilisateur pionnier de l'IBM 704 devrait dépenser l'équivalent de sa première année de location à mettre au point ses programmes de base, qui étaient rendus accessibles gratuitement aux autres utilisateurs, membres du programme SHARE. Au bout d'un an d'existence, SHARE comprenait 62 organisations membres, aux Etats Unis ou ailleurs dans le monde, pour un parc de 76 machines IBM 704, avec une bibliothèque de plus de 300 programmes disponibles (Campbell-Kelly, 2003, p. 35-36[5]). SHARE servit aussi d'interface entre les besoins des utilisateurs et les services de programmation du constructeur, en permettant de dégager progressivement une définition commune des outils logiciels nécessaires aux utilisateurs.

4 A l'époque de la mécanographie, ce sont les schémas de cablage entre machines (l'équivalent d'un logiciel) qui sont collectés par les commerciaux d'IBM, pour être publiés ensuite dans une revue, et servir de modèle et d'inspiration aux autres utilisateurs des machines.

5 Voir aussi l'article d'Atsushi Akera (2001).

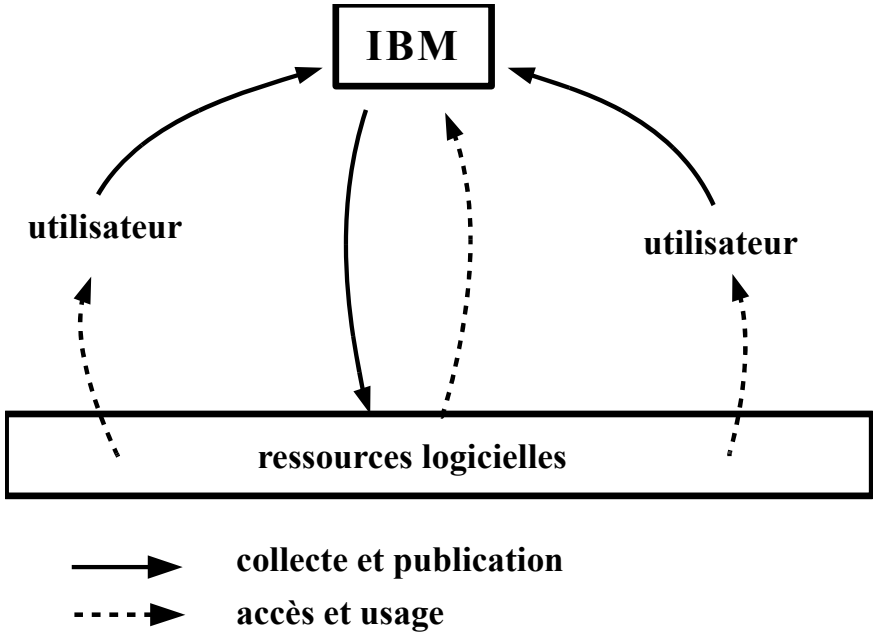


Schéma 1 : Share, un pool de ressources communes géré par une firme

A cette époque, personne ne se pose la question de la protection juridique des logiciels comme actifs de la firme, ni même celle de leur valeur marchande. Il n'y a ni vente directe, ni prix fixé, ni valeur d'échange attribuée au code, mais simplement un ensemble de règles et de formes organisationnelles présidant à une circulation plus ou moins étendue du code ou des programmes, une fois ceux-ci produits. Pour tous, le logiciel est une chose coûteuse et le but du partage est la diminution des coûts et la réduction des investissements nécessaires. Le logiciel est aussi une chose utile, très utile même, indispensable pour « donner une âme à la machine », selon l'expression de Dréan (1996), mais cette chose n'a encore aucune valeur d'échange, car il n'y a aucun commerce ici, aucun « marché ». Le logiciel circule alors comme circulent les « idées », les recettes, les méthodes de production, à travers les rapports que les différents programmeurs peuvent entretenir entre eux, dans les opérations de formation et de maintenance, à l'intérieur et entre les firmes, dans une relative indifférence des services juridiques ou marketing de celles-ci.

A partir du milieu des années 1960, les choses vont commencer à changer car les premières tentatives de commercialisation des logiciels vont apparaître. C'est le fait de certaines sociétés de service qui, ayant développé un programme particulier pour un client, pensent possible de vendre ce logiciel à d'autres. Un logiciel écrit pour traiter la paye d'une firme A est donc toiletté et ré-écrit pour une firme B, puis pour une firme C, à l'image d'un costume prêt à porter « de bonne qualité, mais toujours mal ajusté » (Campbell-Kelly, 2003). Le modèle économique de ces firmes reste cependant calqué sur celui d'IBM, construit sur la fourniture d'une solution sur mesure, avec un ré-emploi plus ou moins habile, plus ou moins important, des développements antérieurs, mais aussi un développement spécifique accompagné par des opérations de maintenance et de formation des

utilisateurs. Le logiciel n'est donc pas réellement vendu seul, indépendamment du reste, comme un produit autonome prêt à l'emploi. L'apparition de ce type de produits logiciels (ou progiciels) est en effet plus tardive, au cours des années 1970 essentiellement.

IBM représentait encore 70 % du chiffre d'affaires de l'industrie informatique américaine. Mais la vente des ordinateurs l'emporte désormais sur la location; et sur cette base de machines installées, un secteur autonome de fourniture de services est apparu, y compris pour les logiciels. La politique de facturation globale selon le service rendu (et vendu) est donc remise en cause. Pour les autorités antitrust, ce système est d'ailleurs susceptible de couvrir des pratiques de ventes liées. Il est donc suspect et IBM change alors en 1969 sa politique commerciale, en facturant désormais séparément ses différentes prestations (*unbundling*) : vente de matériels, ingénierie de système, formation des utilisateurs, maintenance, services de programmation, et... fourniture de *software*. Cette tarification dissociée est généralisée dès 1972, à l'exception de certaines prestations toujours incluses et « gratuites », comme le système d'exploitation compris dans le prix de l'ordinateur. Le code a dc maintenant officiellement un prix; il peut même être commercialisé en tant que tel.

La politique de l'*undbundling* donne alors au logiciel une valeur marchande qu'il n'avait pas auparavant. Elle donne aussi naissance à des revenus et des profits spécifiques. Et ce n'est pas le travail de programmation effectué pour le client, le travail sur le code qui acquière subitement de la valeur, car ce travail était déjà évalué, comptabilisé même comme coût. C'est le logiciel en tant que tel, comme produit indépendant de tout service associé, indépendant des machines, indépendant même de son propre développement et de tout travail sur le code. A l'époque cependant, la séparation reste embryonnaire, et même largement artificielle, dans la mesure où les développements logiciels sont encore la plupart du temps spécifiques à tel ou tel client et souvent accompagnés d'autres prestations. Les sociétés de service qui concurrencent IBM n'ont pas développé en effet un modèle économique réellement différent. Leurs programmes sont vendus comme les autres biens d'équipement, en quantité restreinte et accompagnés d'un fort soutien technique pendant et après la vente. La formation des utilisateurs, la fourniture d'une documentation et les opérations d'après-vente représentent encore la part la plus importante des prestations fournies.

La nouvelle approche pose cependant le logiciel comme un produit commercial comme un autre, qui peut être vendu indépendamment de toute autre prestation, ce qui va donner naissance un peu plus tard à un nouveau modèle économique, avec l'apparition et l'essor à la fin des années 1970 et au cours des années 1980 du marché des progiciels, des logiciels qui sont vendus « *comme des produits au sens le plus classique du mot, commercialisés prêts à l'emploi, utilisables indépendamment de leur auteur, définis pour un marché anonyme et non pour un utilisateur particulier* » (Dréan, 1996, p. 205). Ce type de logiciel est conçu pour une demande anonyme, bien définie, le choix des clients se faisant sur catalogue (ou sur étagère dans un magasin spécialisé). Le même progiciel est d'ailleurs diffusé en de nombreux exemplaires identiques. Le développement du

programme est alors séparé de sa diffusion, relevant de deux services différents dans la même firme, ou incombant même parfois à deux sociétés distinctes, les services d'accompagnement (formation, maintenance), étant devenus pratiquement inexistants.

Au cours des années 1970, la nouvelle politique d'IBM de vente des softwares est imitée par ses concurrents. La pression d'une concurrence par les coûts sur le matériel pousse d'ailleurs à l'abandon des modèles économiques où la vente du matériel subventionnait le développement logiciel. Le prix des programmes et progiciels s'envole alors, et les livraisons de logiciels gratuits deviennent de plus en plus rares (Dréan, 1996, p. 207). Le secteur des logiciels utilitaires généraux (comptabilité, paye, gestion des ressources humaines, etc.) est le premier à s'organiser ainsi, sans vente de matériel, sans fourniture d'autres services. Mais c'est avec l'arrivée de la micro-informatique, dans les années 1980, que le modèle économique du progiciel commercial va réellement prendre son essor; car la diffusion rapide des micro-ordinateurs crée le marché de masse nécessaire aux programmes développés comme progiciels[6]. Une multitude de petites entreprises, souvent formées au départ par une poignée de programmeurs, apparaissent alors, offrant chacune un logiciel particulier conçu pour une utilisation bien précise. Mais ce foisonnement quasi-artisanal va très vite laisser place à quelques grandes firmes à caractère industriel contrôlant des positions dominantes (comme Oracle ou Microsoft)[7].

Ces transformations posent à tous le problème de la caractérisation juridique des logiciels. Comme le code et les logiciels ont désormais une valeur marchande, et qu'ils génèrent des revenus et des profits, ils sont devenus des actifs de la firme. Ils doivent donc être protégés en tant que tel; ainsi pensent du moins la plupart des managers des entreprises. Vient alors la question pratique : trouver, parmi les dispositifs existants (*patents*, copyright, secret de fabrication, etc.), celui qui est le plus adapté au nouveau modèle économique des progiciels commerciaux. Pour les juristes bientôt mobilisés pour trancher le problème, il s'agit aussi de savoir de quel dispositif juridique dépendent les logiciels, et quelles modifications apporter éventuellement aux lois existantes, en particulier à la loi du copyright, pour adapter le droit à ces nouveaux objets que sont les programmes, plus généralement à tout ce qui peut être maintenant stocké dans les mémoires des ordinateurs.

Ce sont d'abord les *patents* qui sont utilisés en premier, l'USPTO acceptant assez vite (et de

6 Pour développer un logiciel, il faut pouvoir accéder à un ordinateur facilement et de manière permanente, ce qui était réservé jusqu'ici aux informaticiens travaillant dans une grande entreprise, un centre de recherche ou une université. Mais cet obstacle à la création de firmes indépendantes est levé avec l'arrivée des micros-ordinateurs, et de très petites firmes, des individus isolés même, peuvent désormais développer leurs propres programmes, ce qui explique la rapidité de l'évolution.

7 La logique générale de cette économie du progiciel est en effet monopoliste, avec « *la persistance de profitabilités très élevées et la domination quasi-absolue de Microsoft dans des segments comme les systèmes d'exploitation pour micro-ordinateurs (81 % avec MS-DOS), les tableurs (73 % avec Excel), ou à un degré moindre les traitements de texte (55 % avec Word),...* [Car] *le secteur du progiciel est en réalité formé d'un grand nombre de sous-secteurs, dont chacun est concentré.* [Même si] *dans chaque secteur, l'entreprise leader peut être de taille modeste* » (Dréan, 1996, 209-210). Le principe d'exclusivité du dpi ne fait alors qu'accompagner, renforcer et justifier cette structure monopoliste.

manière de plus en plus importante au cours du temps) d'accorder des brevets dans le domaine des *softwares*, en liaison ou non avec le matériel (*hardware*). Mais la validité de ces patents est déjà à l'époque largement contestée; les brevets étant traditionnellement conçus comme devant protéger des artefacts tangibles, et non des idées ou des actifs intangibles. Le logiciel, une forme d'écriture utilisant largement les algorithmes, les formules mathématiques et les symboles semble en effet plus proche des idées que de l'artefact tangible, et par là-même difficilement brevetable. Les *patents* accordés peuvent alors se révéler de faible valeur en cas de contestation juridique. D'un autre côté, le copyright est appliquable au code source, au programme écrit par le programmeur dans une forme compréhensible par les êtres humains, avec les commentaires et la documentation associée; et à partir de 1964, des programmes sont déposés sous cette forme à l'administration des copyrights. Mais cela protège assez mal le producteur du logiciel, puisque le code source est alors dévoilé, et que la loi du copyright n'accorde un droit exclusif que sur une « expression » particulière, excluant toute protection en matière « d'idées, de principes, de faits », etc[8]. La lecture du code source donne accès aux algorithmes et aux fonctionnalités, et permet la ré-écriture dans une « expression » différente, avec au final un programme équivalent.

La demande des éditeurs de progiciels est alors une inscription directe du logiciel dans la loi du copyright sans obligation de dévoilement, avec une extension de la protection à toutes les formes de « copies », une extension donc au code machine des fichiers binaires. Une commission, la commission CONTU (*National Commission on New Technological Uses of Copyrighted Works*), est constituée aux Etats-Unis pour trancher la question, et en 1978, cette commission préconise d'étendre aux logiciels, en tant qu'œuvres originales, la protection du copyright[9]. La seule opinion dissidente fut celle de John Hersey, président de l'association des écrivains américains et auteur lui-même, qui, distinguant le code source du code compilé (le code machine), proposa un amendement inverse à la loi de 1976 pour rendre explicite que le copyright portant sur le code source ne pouvait être étendu au code devenu un simple élément de la machine[10]. En 1980, le Congrès suivant les recommandations de la commission CONTU devait amender le *copyright Act* de 1976 pour étendre le principe d'exclusivité aux fichiers binaires.

---

8 Selon une jurisprudence de la fin du XIX^{ème} siècle (Baker v. Selden, 1879), un principe explicitement rappelé dans la loi du copyright de 1976. Cf. § 102 (b) « *In no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work* ».

9 Dans cette commission, fait remarquable, il n'y avait aucun représentant de l'industrie informatique, aucun programmeur, aucune personne qui soit professionnellement, de près ou de loin, en rapport avec les softwares. A la même époque, la question était cependant débattue dans les revues et la presse; Stephen Breyer par exemple, dans son étude sur le copyright (1970), proposait de ne rien faire, amender la loi n'étant d'ailleurs pas nécessaire pour le développement de l'industrie.

10 Pour Hersey, le code source a deux caractéristiques : communiquer des instructions à la machine et communiquer avec des êtres humains (les programmeurs qui ont travaillé ou travailleront sur le programme); les commentaires sui parsèment les pages du code source ne sont en particulier destinés qu'aux êtres humains. Une fois compilé, comme fichier binaire, le programme ne peut servir qu'à communiquer avec la machine. Ce n'est plus alors un texte, ou une œuvre, selon la tradition du copyright, mais un simple élément de la machine, comparable à la pièce d'un moteur, à un vilebrequin (*a cam*) donnant des impulsions au reste de la machine.
Voir son analyse en ligne : < http://digital-law-online.info/CONTU/contu14.html > et Annexe B.

En pratique, ce changement de la loi ne faisait que valider, légitimer et renforcer des pratiques déjà bien établies. Les producteurs de progiciels avaient en effet mis au point des « stratégies propriétaires », selon l'expression consacrée, combinant le secret de fabrication, le droit du copyright, les *trademarks*, et des systèmes de licences. Le point clef était la seule vente de fichiers binaires, le code source devant toujours rester inaccessible et caché. Le logiciel était alors accompagné d'une licence restrictive en matière d'utilisation, interdisant toute décompilation ou rétro-ingénierie, limitant les installations possibles du programme (nombre de machines par exemple, etc.), et prohibant toute copie, toute modification, toute redistribution; la documentation accompagnant les binaires était mise sous copyright classique, et le personnel de la firme ayant développé le logiciel soumis à des clauses de non divulgation. Le partage et la circulation sans contrôle du code et des binaires deviennent alors des pratiques proscrites, stigmatisées sous le terme de « piraterie ». Le logiciel n'est plus qu'une propriété, qu'un actif commercial, un actif dont il faut tirer le maximum de revenus, le plus longtemps possible, en empêchant donc toute divulgation, et par là-même toute « copie », et toute activité de transformation, amélioration ou innovation à partir du code source.

III. Les années 1980 : enfermement propriétaire et naissance des logiciels libres

Dans l'industrie informatique américaine, il y a alors une sorte de rupture dans les normes de comportement régulant le partage et la circulation des logiciels, et particulièrement du code source. L'essor du marché des progiciels a changé la représentation qu'on se faisait de ce code. Pour les dirigeants et les services juridiques des entreprises, il y a là un actif, une forme de capital, une marchandise, et non plus un simple objet technique. Les programmeurs vont devoir intégrer cette nouvelle donnée, qu'ils le veuillent ou non. Le secret et la non communication du code source deviennent alors des normes absolues, pendant la phase de développement du logiciel, et même après quand le logiciel est utilisé par les clients. La seule forme réellement publiée est la version binaire, dont la copie et la décompilation éventuelle sont officiellement devenues illégales en 1980. Les politiques « propriétaires » de fermeture et de rétention du code deviennent une sorte de norme générale pour toute l'industrie informatique; et ceci, même dans des situations où les mecs des firmes ne sont aucunement menacés (bien au contraire) par le maintien des pratiques antérieures de partage du code[11].

11 On peut évoquer ici l'expérience vécue par Richard Stallman. Devant régler pour son laboratoire un problème de dysfonctionnement logiciel d'une imprimante, il se vit opposer la propriété intellectuelle du fournisseur, et refuser pour cette raison tout accès au code source. Il ne put donc résoudre le problème (première absurdité), ni améliorer le programme pour tous les autres utilisateurs (deuxième absurdité), pour un programme qui est livré avec une machine et ne peut guère être utilisé (et vendu) sans cette machine (troisième absurdité). Aujourd'hui, la règle générale pour les producteurs d'imprimante est de vendre les machines et les consommables (les encres et certains papiers), en fournissant gratuitement les pilotes (et leurs mises à jour logicielles); et souvent le pilote est maintenant disponible comme logiciel libre, pour permettre justement une transformation et amélioration éventuelle par les utilisateurs.

C'est en effet un renversement général des règles de comportement qui est alors prôné par les partisans des stratégies dites « propriétaires ». Car comme le note Richard Stallman, aux débuts de l'informatique la règle était plutôt le partage du logiciel, « *une notion aussi ancienne que les premiers ordinateurs, tout comme on partage des recettes depuis les débuts de la cuisine* » (Stallman, 1999, 61); et la pratique générale, pour tous les programmeurs, qu'ils soient chercheurs universitaires ou employés par des firmes de l'industrie, était de communiquer le code à la demande. « *Quand des ingénieurs souhaitaient utiliser l'un de nos programmes, nous leur en accordions volontiers l'autorisation. Et quand on voyait quelqu'un utiliser un programme intéressant mais inconnu, on pouvait toujours en obtenir le code source, afin de le lire, de le modifier, ou d'en vampiriser des parties dans le cadre d'un nouveau programme* » (op. cit., p. 62).

Mais la nouvelle représentation, contamine même les universités et les centres de recherche, des institutions où la norme est pourtant traditionnellement la publication et la mise dans le domaine public, la diffusion la plus large possible des résultats d'une recherche étant de l'avis général le moyen d'obtenir une progression rapide de la science et du savoir. Le renversement des représentations, des règles et des valeurs pose alors un problème aux chercheurs et aux programmeurs, et fait débat : S'agit-il d'une évolution somme toute inéluctable, l'économie du code devenant enfin « mature », car marchande ? Du simple exercice, normal et légitime, d'un « droit naturel » du propriétaire du code ? Le contrôle des usages et de ce que font les utilisateurs du code est-t-il réellement nécessaire, indispensable même, pour assurer le financement et donc la production de logiciels de bonne qualité, couvrant toutes les demandes sociales ? Ou s'agit-il simplement d'un « *système social, celui du logiciel propriétaire* », qui vous interdisant de partager et d'échanger le logiciel, « *est antisocial, immoral, et tout bonnement incorrect* » (Stallman, 1999, p. 62) ?

Le changement dans les normes et les règles de comportement entre en effet directement en conflit avec les formes de coopération nouées pour le développement de certains programmes. Le partage du code est maintenant proscrit; toute copie est interdite, définie comme de la "piraterie"; la diffusion étroitement encadrée et limitée; les pratiques d'entraide et de coopération entre programmeurs sont découragées et même rendues illégales. Les nouveaux ordinateurs apparus dans les années 1980, comme les Vax de Digital, ou ceux construits pour le processeur Motorola 68020, sont fournis comme avant avec leur propre système d'exploitation, mais accompagné par la nouvelle norme; car il faut maintenant signer des accords de non divulgation pour en obtenir des copies exécutables, et tout travail coopératif de transformation ou amélioration de ces programmes entre utilisateurs-programmeurs devient impossible. Les processus d'invention collective qui existaient çà et là dans l'industrie sont alors sapés à la base, et les communautés de programmeurs-utilisateurs constituées autour de certains développements logiciels menacées d'effondrement[12].

12 C'est ce qui devait arriver en particulier à la communauté des programmeurs constituée autour du système d'exploitation à temps partagé ITS (*Incompatible Timesharing System*), du laboratoire d'Intelligence Artificielle du

C'est le cas en particulier pour la communauté d'utilisateurs-programmeurs constituée à partir de 1974 autour du développement du système d'exploitation Unix, une communauté comprenant les Bell Labs, appartenant à AT&T et Western Electric, et un grand nombre d'universités et de centres de recherche, situés un peu partout aux Etats-Unis et dans le reste du monde. A l'origine de cette communauté, qui émerge plus ou moins spontanément, il y a l'impossibilité pour les Bell Labs d'envisager une exploitation commerciale du code, en vertu d'un *consent decree* de 1956 dans le cadre d'une procédure *antitrust* interdisant à la compagnie de téléphone toute activité commerciale en informatique[13]. L'exercice habituel des droits de *patents*, *copyright* ou secrets de fabrication est donc abandonné, et à l'inverse, la pratique pour tout ce qui relève de l'informatique est d'accorder automatiquement licence avec un *fee* peu élevé, et même sans aucun paiement, gratuitement, dans le cas des universités ou des centres de recherche. Cette situation particulière favorise la diffusion et la réutilisation du code d'Unix, et donne naissance à un réseau d'utilisateurs partageant leurs corrections de bogues et toutes les améliorations et développements qu'ils apportent au programme. Assez vite d'ailleurs, les Bells Labs, à l'origine du système, ne sont plus qu'un foyer de production et diffusion des innovations parmi d'autres, un foyer dont l'importance décline progressivement, l'Université de Berkeley s'imposant dès le milieu des années 1970 comme le foyer principal du développement d'Unix, avec les premières distributions de code BSD.

Mais au début des années 1980, la procédure antitrust est relancée sur d'autres bases avec comme perspectives la fin du monopole téléphonique, réalisé en 1984 avec l'apparition des Baby Bells, la séparation des Bell Labs du reste du groupe, et le droit pour ceux-ci de développer des activités commerciales en informatique, dans le domaine du *hardware* comme du *software*, ce qui sera décidé officiellement en juin 1984 par le *proposed decree* du juge Greene. On assiste alors à un changement complet de la politique de licence des Bell Labs, avec une augmentation importante des prix, y compris pour les universités, et une redéfinition restrictive des conditions d'utilisation du code[14]. L'un des effets de cette évolution est le transfert de fait à la communauté de la responsabilité du développement du système d'exploitation Unix, les programmeurs de Berkeley bénéficiant d'ailleurs pour ce faire de plusieurs financements de la DARPA accordés pour développer la portabilité du système et la gestion des réseaux (TCP/IP) nécessaire au développement de l'Arpanet[15]. Par ailleurs, les utilisateurs du code produit à l'université de Berkeley, un code accompagné d'une licence très permissive accordée par les Régents de

MIT, pour lequel Richard Stallman travaillait (Cf. Stallman, 1999; et Levy, 1985).

13 Pour l'histoire d'Unix, voir Salus (1994) et Behrendorf et alii. (1999).

14 En 1978, les droits (fee) étaient de 7000 $ pour une Université, plus pour une licence « commerciale ». Mais dix ans après (1988), le prix était bien plus élevé, environ 100 000 $; et en 1993, il fallait compter 200 000 $ pour obtenir le code source, ce qui éliminait les individus, les petites compagnies et une bonne partie des universités et centres de recherches. Cf. Salus (1994).

15 Un des problèmes préoccupant la DARPA était le vieillissement des matériels et leur hétérogénéité. Pour assurer un minimum de standardisation et de pérennité des logiciels, et permettre aussi la communication des machines entre elles, la DARPA mise alors sur le système d'exploitation Unix, car celui-ci avait déjà montré sa portabilité sur différentes machines; l'équipe de Berkeley fut plus spécifiquement retenue à cause des premières distributions BSD.

l'Université de Berkeley, doivent toujours obtenir une licence d'AT&T pour le cade provenant des Bell Labs. Devant l'augmentation du coût des licences, certains utilisateurs demandent une séparation entre le code produit à Berkeley et le code sous licence AT&T, ce qui conduit en juin 1989 à la sortie de la première distribution de code librement redistribuable de Berkeley, la Networking Release 1[16]. A cette époque, l'éclatement de la communauté Unix était déjà consommé. AT&T et les Bell Labs étaient engagé dans la mise au point de distributions commerciales stables, une démarche qui devait culminer dans la commercialisation d'une version supposée achevée, System V, et la formation de la société *Unix System Laboratoires*, qui devait commercialiser le System V, qui fut la base de la plupart des Unix propriétaires (AIX, HP-UX, IRIX, Solaris 2, etc.) des années 1980-1990. La séparation au niveau du code était donc inévitable. un travail de ré-écriture du code source provenant à l'origine des Bell Labs fut donc engagé par le réseau des utilisateurs, afin de « libérer » le code d'Unix, y compris les utilitaires et les bibliothèques, de la licence AT&T, ce qui devait donner finalement la *Networking Release 2* (juin 1991), et conduire à l'apparition de différents groupes : NetBSD, freeBSD, OpenBSD et BSDI prenant désormais en charge le destin du programme dans différentes architectures et selon différents mecs.

Le changement général des règles en matière de logiciel pose aussi problème à chaque programmeur individuel, en particulier au moment où il quitte l'université, et le monde de la recherche, pour rentrer dans l'industrie. Le choix est alors d'accepter les nouvelles règles ou de les refuser. Accepter donc, comme le dit Richard Stallman, de rejoindre le monde du logiciel propriétaire, en signant des accords de non divulgation, en promettant de ne pas aider les autres programmeurs, et contribuer ainsi à une évolution conduisant à un monde où toute communauté coopérative serait interdite, un monde où des murs de plus en plus hauts, ceux des différentes firmes, sépareraient les différents programmeurs (ou programmeurs-utilisateurs), les isolant les uns des autres (Stallman, 1999, p. 64). Le refus de cette perspective conduisit Stallman à démissionner du MIT pour démarrer un projet de création d'un ensemble de « logiciels ls » (*free softwares*) autour d'un système d'exploitation compatible Unix, lui-même « libre », le code source devant être disponible, et donc accompagné juridiquement de droits d'usage très étendus, des droits que les auteurs du code concéderaient à tous les utilisateurs. L'objectif est alors la maintien ou la renaissance des « communautés coopératives » formées pour le développement des logiciels. Ce projet fut bientôt désigné par le terme GNU, un acronyme récursif signifiant simplement "*GNU's Not Unix*" (GNU N'est pas Unix), pour bien marquer le refus éthique et politique de l'évolution en cours dans la communauté d'Unix, avec la fermeture d'une partie du code, la division de la communauté et l'apparition des différents « Unix propriétaires ». Pour entreprendre ce projet, il était cependant nécessaire d'être soi-même libre d'accorder juridiquement des droits aux autres. « *J'ai* [alors] *démissionné de mon poste du MIT,* [car] *le MIT aurait pu se déclarer*

16 Cette distribution comportait tout ce qui permettait la gestion des réseaux (TCP/IP) et la connexion à l'Arpanet, avec les utilitaires associés; l'essentiel de ce code ayant été produit en dehors des Bell Labs, à Berkeley ou dans d'autres universités.

propriétaire de mon travail, et lui imposer ses propres conditions de distribution, voire en faire un paquetage de logiciels propriétaires. Je n'avais pas l'intention d'abattre autant de travail et de le voir rendu inutilisable pour ce à quoi il était destiné : créer une nouvelle communauté qui partage le logiciel » (Stallman, op. cit., p. 66).

Le choix de développer un système d'exploitation compatible Unix était lui plus technique et stratégique; car un tel système serait immédiatement portable sur de nombreuses machines et accessible de surcroît à tous les utilisateurs d'Unix.

Le premier programme du projet GNU, écrit par Stallman, fut un compilateur traitant plusieurs langages pour plusieurs plate-formes, le compilateur GCC[17]; suivi dès 1985 par l'éditeur de code source Emacs, un outil de base pour tout programmeur[18]. Le nombre de participants au projet GNU augmentant, une association à but non lucratif, la *Free Software Foundation*, est fondée en 1985 afin de récolter des fonds pour le développement du projet GNU. Il s'agissait plus généralement de promouvoir la cause du logiciel libre et d'organiser les actions des différents participants au projet. La FSF récupère la distribution et la vente de logiciel libre sur bandes, la vente de manuels, et d'autres services associés; et les fonds recueillis payent le travail de différents programmeurs sur certains logiciels destinés au projet GNU, comme la bibliothèque de langage C, un ensemble de programmes que toute application fonctionnant sur un système d'exploitation Unix libre doit utiliser pour communiquer avec celui-ci, ou l'interpréteur de commandes BASH, développé par Brian Fox, un employé de la FSF. Le projet GNU hérite par ailleurs d'un ensemble de principes et outils (comme le langage C), caractéristiques des architectures de type Unix, et en particulier d'une conception modulaire de l'ensemble du système. Chaque logiciel est alors construit comme un ensemble de fichiers (module) indépendant, ayant chacun une fonction particulière; chaque module pouvant être écrit séparément, ce qui rend possible une construction progressive de l'ensemble.

L'histoire du projet GNU illustre parfaitement cette possibilité (technique) de débuter la construction du système par n'importe quel élément, de traiter les différents éléments sans ordre, en produisant par exemple en dernier ce qui semble être la base même du système, comme le noyau du système d'exploitation. « *Au commencement*, raconte Stallman (1998), *il s'agissait d'écrire n'importe quel composant, car tous les composants manquaient. L'ordre importait peu*

17 GCC (GNU Compiler Collection), dont l'écriture fut entreprise en 1985 par Richard Stallman et publié en 1987 pour la première fois, est une collection de logiciels capables de compiler divers langages de programmation (C, C++, Objective-C, Java, Ada, Fortran) pour un grand nombre de systèmes d'exploitation et microprocesseurs. Aujourd'hui, c'est le compilateur de base de Linux, BSD, Mac OSX, NeXTSTEP, BeOS/Haiku.

18 Ce logiciel est distribué à l'époque de deux façons, directement et gratuitement par l'intermédiaire d'un site ftp, et parallèlement vendu (bandes magnétiques) au profit de l'auteur. « *Etant sans emploi, je cherchais des moyens de gagner de l'argent grâce au logiciel libre. C'est pourquoi j'ai annoncé que j'enverrais une bande à quiconque en désirait une, en échange d'une contribution de 150 USD. De cette manière, je mettais en place une entreprise autour de la distribution du logiciel libre.*. » (Stallman, 1999, p. 68). Le même principe de double distribution (payante par envoi postal, avec éventuellement des services associés, etc., et comme *freeware* par simple téléchargement direct) fut repris par la suite pour les distributions BSD, et représente d'ailleurs toujours un mec possible du monde des logiciels libres.

alors. Plus tard, une liste de ce qui manquait fut établie, la task-list de GNU ; cette liste permettait de recruter des développeurs pour telle ou telle partie du projet ». Vers 1990, presque tout le système existait, composé entièrement de logiciels libres. Seul le cœur du système d'exploitation, le noyau, manquait encore. Mais la conception technique retenue par Stallman (Le projet Hurd basé sur un principe de micro-noyau) s'avérant difficile à réaliser, c'est finalement un noyau compact d'un style plus classique écrit par Linus Thorvald (Linux), qui est venu compléter et réaliser ce qui était le but même du projet GNU, constituer un système complet de logiciels libres.

Mais ce qui sépare les programmeurs travaillant sur le projet GNU ou sur d'autres projets logiciels libres, d'autres programmeurs travaillant à développer des logiciels propriétaires n'est ni les outils, ni les langages utilisés, ni même un comportement particulièrement innovateur, car souvent, il s'agit simplement de libérer un programme qui existe déjà, un programme dont les principes et l'architecture sont connus (Narduzzo et Rossi, 2003 ???), de le ré-écrire donc en mettant le code dans un autre système de règles. La différence se situe dans les licences accompagnant le code. Le projet GNU avait (et a) en effet deux objectifs : constituer d'un côté un stock de logiciels (système d'exploitation et logiciels d'accompagnement), que toute la communauté des programmeurs peut librement utiliser et ré-utiliser; et établir, d'un autre côté, les règles de partage et de mise en commun, définir donc les droits et obligations de chacun, définir les règles qui devraient donner à tous une grande liberté d'usage en matière de modification, transformation et redistribution des programmes, en précisant aussi les relations de la communauté à l'égard des tiers. Cet objectif va conduire à élaborer une définition du *free software* (logiciel libre), permettant de dire si un logiciel est réellement « libre » ou non, et aussi compatible ou non avec le projet GNU. Cela va donc conduire à la rédaction d'une licence, la licence GPL ou GNU-GPL (*GNU-General Public Licence*), qui accompagne tous les programmes développés dans le cadre du projet GNU[19]; ce qui sera principalement l'œuvre de Richard Stallman et d'Eben Moglen, professeur de droit et d'histoire à l'Université de Columbia, conseiller juridique de la *Free Software Foundation*.

IV. La base constitutionnelle du commun, les licences de logiciel libre

Le concept de logiciel libre (free software) – tout comme celui de logiciel *open source*, une expression apparue par la suite, et presque équivalente aujourd'hui -, est en rapport direct avec la licence accompagnant le code source, une licence où l'auteur, titulaire du copyright, accorde des droits d'usage particulièrement étendus à l'utilisateur. Celui-ci peut non seulement exécuter le logiciel, en l'utilisant pour ses fonctionnalités, mais il peut aussi le copier et le modifier comme il

19 Tous les logiciels développés pour le projet GNU sont en principe sous licence GNU-GPL (avec clause *copyleft*), à l'exception de certains programmes sous licence GNU-LGPL. Le système d'exploitation Linux, développé séparément, est lui aussi sous licence GPL. Mais d'autres logiciels libres existants ont été intégrés directement dans le « système GNU », comme Tex (formateur de texte) et le système de fenêtrage X avec leur propre licence. Le système GNU est donc différent du projet GNU. « *Il comprend des programmes qui ont développé par d'autres, dans le cadre d'autres projets, pour leur buts propres, mais qu'on peut réutiliser, car ce sont des logiciels libres* » (Stallman, 1999, p. 66).

l'entend, pour l'adapter à ses besoins, le transférer sur un autre système d'exploitation ou un autre type de machine, pour en corriger les défauts (bogues), ou encore pour l'utiliser comme matériau afin de construire un tout autre programme, en supprimant certaines parties du code, en réécrivant d'autres parties, en ajoutant des extensions, etc. En pratique, cela signifie qu'il faut non seulement pouvoir accéder au code source, mais pouvoir aussi le redistribuer à sa guise, modifié ou non. Sans ce droit, cette liberté, il n'y a en effet aucune possibilité de partage et de mise en commun, et aucune coopération possible entre programmeurs pour le développement d'un programme.

L'utilisateur peut donc toujours redistribuer le logiciel, modifié ou non, gratuitement ou non. Le logiciel est traité comme un simple objet technique, comme une ressource pour le travail de programmation. Il est traité du point de vue de l'utilisateur-programmeur, et non du point de vue d'un auteur d'origine, propriétaire ou ayant droit. Il est donc traité du point de vue d'une catégorie d'utilisateur qui, pour une raison ou pour une autre, ne peut se satisfaire de la simple exécution du programme tel qu'il est, avec ses qualités et ses défauts (ce qui peut suffire à des utilisateurs ordinaires), mais veut pouvoir le modifier et pouvoir produire un autre logiciel avec ce code; et qui de plus possède la compétence nécessaire pour le faire. La licence de logiciel libre est justement faite pour autoriser, favoriser même, une telle utilisation, où le logiciel est posé comme un paquet de code, comme un ensemble de pièces et d'éléments ré-utilisables, disponibles pour développer une version différente ou améliorée, ou même un autre logiciel (éventuellement concurrent du programme d'origine). On voit l'écart avec les licences commerciales habituelles, dont le principe est la protection du code et l'interdiction des copies, car toute copie est alors inévitablement conçue comme une perte de recettes; les droits de l'utilisateur ne peuvent alors qu'être limités à l'exécution du programme tel qu'il est, ce qui ne peut satisfaire que les utilisateurs ordinaires, et non les programmateurs.

C'est au milieu des années 1980 que le terme *free software* (ou logiciel libre) est introduit par Stallman et la *Free Software Foundation*, et c'est aussi à cette période que cette notion de logiciel libre est précisée et fixée, en opposition avec la notion de « logiciel propriétaire ». Depuis cette date, il existe une définition officielle du logiciel libre, fournie par la *Free Software Foundation*, une définition qui permet de dire si un logiciel est libre ou non, à partir de l'analyse des termes de la licence et des droits accordés par celle-ci à l'utilisateur. Cette définition est disponible sur le site web de la *Free Software Foundation*, un site volontairement éducatif, où les définitions, les distinctions introduites : logiciel privé, libre, copylefté, non copylefté, compatible ou non avec la GNU-GPL, propriétaire, semi-propriétaire, etc., sont en liaison directe avec l'objectif de création et maintien dans la durée d'un pool de ressources logicielles partagées dans une communauté, le projet GNU. Un logiciel est considéré comme libre si la licence accorde à l'utilisateur les quatre libertés suivantes :

(1) La liberté d'exécuter le programme, pour tous les usages et sans restrictions particulières.

(2) La liberté d'étudier le fonctionnement du programme et de l'adapter à ses propres

besoins, ce qui signifie pouvoir corriger des erreurs, supprimer certaines parties du code, ajouter des extensions, et fusionner son propre code (ou le code d'une tierce partie) dans les modules existants.

(3) La liberté de copier le logiciel et d'en redistribuer ses copies.

(4) La liberté d'améliorer ou transformer le programme et de publier les transformations, « *pour en faire profiter toute la communauté* ».

La redistribution du logiciel, modifié ou non (point 3 et 4), inclut le code source et les formes binaires, exécutables, du programme; une redistribution qui peut être faite gratuitement ou moyennant paiement, au choix de l'utilisateur. « *Vous êtes libre de redistribuer des copies, avec ou sans modification, gratuitement ou non, à tout le monde, partout* ». Car « l'expression *free software fait référence à la liberté et non au prix* » (FSF, 2009a). Le *software* est donc *free* au sens du « *free speech* » et non au sens de « *free beer* »; et il est bien précisé qu'une licence qui interdirait ou limiterait la vente ou revente des copies ne pourrait être considérée comme une licence de logiciel libre.

L'expression « *open source software* » (ou logiciel *open source*) est d'apparition plus récente, introduite par Eric Raymond, Bruce Perens, et quelques autres, en 1998 avec l'*Open Source Initiative,* dans le cadre d'une critique du discours de Richard Stallman et de la FSF, un discours jugé par eux trop éthique et idéologique. Comme le rappelle Bruce Perens (1999, p. 186), pour « *faire la campagne du logiciel libre auprès de ceux qui portent des cravates,* [...] *les hommes d'affaires, conservateurs, effrayés par le mouvement de liberté initié par Stallman* », un discours en termes plus pragmatiques (« qualité des logiciels », « efficacité du mode de développement », etc.) plutôt qu'en termes éthiques (« liberté », « entraide ») fut élaboré, et une stratégie de labellisation mise sur pied, afin d'imposer les critères du libre aux licences des entreprises séduites par l'expérience de développement de Linux. Le terme « *open source* » (le label) devait remplacer le terme *free*, jugé trop ambigu, puisqu'en anglais, *free* peut signifier aussi bien libre que gratuit[20].

Une définition de « *l'open source* » (*Open source definition*) fut rédigée par Bruce Perens, à partir du contrat social Debian, élaboré et approuvé un peu avant (1997) par les développeurs de ce projet de production d'une distribution de logiciels, tous ls, autour du système d'exploitation Linux. Cette définition sert alors de référence au processus d'attribution du label « *OSI-Approved Open Source* », et plus généralement à tous ceux qui utilisent le terme *open sourc software*. Elle se présente comme une liste de dix critères et non comme un ensemble de libertés (ou droits) comme dans la définition du *free software*. En pratique, les deux définitions donnent les mêmes résultats, car on retrouve fondamentalement les mêmes garanties, sur le mode : « *la licence ne doit pas restreindre...* ». Ainsi en juillet 2009, il y avait 66 licences labellisées sur le site de l'*Open Source*

20 Mais le terme *open* est tout aussi ambigu, ayant de plus déjà beaucoup servi dans l'industrie informatique, dans des contextes souvent proches mais différents des logiciels libres, comme dans toutes les initiatives visant à introduire le minimum de standardisation et de partage des interfaces (norme Posix par exemple) nécessaire au fonctionnement conjoint de systèmes par nature hétérogènes, car conçus séparément, et de plus sous dpis, ce qui pose le problème des droits d'usage et là aussi de la divulgation du code des interfaces.

Initiative, dont deux seulement étaient rejetées par la *Free Software Foundation* comme trop restrictives dans les droits accordés pour être pleinement considérées comme des licences de logiciels libres[21].

Pour permettre la formation du commun et assurer sa pérennité, les promoteurs du projet GNU inventèrent la clause dite « copyleft ». Attribuer des droits étendus de copie, de transformation et de redistribution du code (les quatre libertés d'un *free software*) aux utilisateurs, ou mettre le logiciel dans le domaine public, et promouvoir le partage du code, ne garantit pas que ce code va rester dans le commun. La transformation d'un logiciel du domaine public ou d'un logiciel libre en un logiciel propriétaire est en effet possible; car la loi du copyright n'impose aucune obligation de publication en matière de code source, et protège à l'inverse les copies binaires, que le code source soit publié ou non; et dans une stratégie propriétaire, il n'est et ne sera jamais publié. On peut donc facilement, tout-à-fait légalement, utiliser du code du domaine public pour réaliser un programme, qu'on distribue ensuite uniquement en versions binaires sous licence propriétaire, le code source (ancien et nouveau) restant caché. Pour éviter cette mésaventure aux logiciels libres, il fallait donc inclure dans les licences de *free sofware* une clause spécifique interdisant cette pratique, en imposant une redistribution du logiciel modifié (ou non) dans les mêmes conditions que la licence d'origine, autrement dit, la nouvelle licence devrait accorder aux utilisateurs les mêmes droits sur le code que la précédente. C'est le principe du *copyleft*, qui est d'ailleurs, parmi toutes les licences de logiciel libre (ou *open source*), une des caractéristiques de la licence GNU-GPL. Dans la philosophie du logiciel libre de Richard Stalman et de la *Free Software Foundation*, les libertés sont données aux utilisateurs (et à tous les utilisateurs), et non au développeur. Comme le logiciel est libre, il peut le modifier et en redistribuer une nouvelle version, mais, avec le *copyleft*, il doit transmettre les mêmes libertés (ou droits d'usage) aux autres, sans pouvoir ajouter ici de conditions restrictives, et sa propre licence doit comporter une clause *copyleft*. Avec cette clause, le développeur n'a pas plus de droits que les autres utilisateurs, ce qui fournit un cadre institutionnel égalitaire et pérenne, favorable à des développements logiciels prolongés. La privatisation d'un code partagé dans une communauté de programmeurs par introduction d'extensions, sans communication du nouveau code source, ou modifications des interfaces, comme c'était le cas à l'époque où cette clause apparaît pour le code produit par la communauté BSD, devient alors juridiquement impossible. GNU n'est pas Unix, et ne veut pas l'être. Le projet de développement du commun ne doit pas connaître le même destin que celui que connut la communauté Unix, l'éclatement avec apparition de versions différentes du même sys d'exploitation Unix, toutes propriétaires, avec pour les utrs de multiples problèmes de compatibilité.

En s'appuyant sur le principe propriétaire de la loi du copyright – le droit exclusif de l'auteur

21 Il s'agit en l'occurrence de la licence *NASA Open Source Agreement 1.3*, qui impose une « *création originale* » pour l'ajout de code source au code d'origine, interdisant donc l'utilisation d'un code source venu d'un autre logiciel, et de la *Reciprocal Public Licence 1.5*, qui limite le prix éventuel demandé pour la vente d'une copie et oblige à publier toute modification, même réalisée pour un usage purement privé, avec envoi d'une notification au développeur d'origine.

à contrôler les utilisations de son code – la clause *copyleft* en détourne l'exercice habituel. Elle crée une forme de domaine public sécurisé, intégrant un principe d'extension continue du commun, puisque tous les ajouts, toutes les modifications du code source sous *copyleft* entrent directement dans celui-ci, sans pouvoir en sortir. Il s'agit en effet, comme l'explique très bien Mélanie Clément-Fontaine (1999), de « *créer progressivement un fonds commun de logiciels libres dans lequel tout le monde pourrait puiser, auquel chacun pourrait ajouter, mais duquel personne ne pourrait retrancher* ». Ce dispositif juridique élémentaire sécurise à l'avance la production et la transformation des logiciels libres en installant dans la durée le principe de partage du code, le rejet par les auteurs-programmeurs de leur droit à l'exclusivité inscrit dans la loi du copyright. Il fournit alors le cadre institutionnel égalitaire et pérenne, favorable au développement prolongé des logiciels libres et à la mise sur pied de formes de production coopérative.

On peut noter qu'une licence de logiciel libre, qu'elle soit copyleftée ou non, donne à l'utilisateur le droit de modifier le code et le programme, mais n'impose aucune publication des modifications. La clause copyleft ne joue que si le logiciel est redistribué. Dans le cas d'une utilisation purement privée (logiciel privé), les modifications peuvent rester confidentielles ou « secrètes », un cas d'ailleurs extrêmement fréquent, celui, typiquement, du programmeur qui développe un nouveau logiciel ou écrit du code pour ses propres besoins, ou pour son entreprise ou une autre organisation, au cours d'opérations de maintenance par exemple. Une clause qui imposerait la publication ou la communication de toute modification serait même une atteinte aux droits de l'utilisateur-programmeur, limitant fortement en pratique le ré-usage du code source; et on ne pourrait plus parler alors d'un logiciel libre.

V. Limites et cartographie du commun

Presque trente ans aujourd'hui après la naissance historique des logiciels libres, on peut dresser une sorte de cartographie du commun actuel à partir des licences, à partir donc des règles portant sur l'usage du code. Notons tout d'abord que le nombre de licences possibles en matière de logiciel est *a priori* infini, ou du moins très grand. Tout propriétaire d'un copyright peut rédiger sa propre licence, avec ses propres clauses, en définissant à sa manière, comme il l'entend, les usages possibles du logiciel qu'il fournit ou distribue. Il n'existe pas en effet de statut ou de contrat type imposé à tous par les législations. Il n'y a que le principe - imposé à tous - selon lequel les logiciels (code source et binaires) relèvent du copyright (ou du droit d'auteur) et doivent donc être accompagnés d'une licence. Le nombre de licences propriétaires possibles est donc très grand; chaque firme ayant sa propre licence, voire des licences différentes, spécifiques à chaque logiciel, ou même plusieurs licences pour le même logiciel, suivant les situations, la segmentation des marchés et la discrimination des clients étant presque de règle dans les modèles économiques construits sur des stratégies propriétaires. Mais la variété potentielle est aussi une réalité du logiciel libre; la GNU-GPL n'est pas la seule licence de logiciel libre; ils en existent bien d'autres, et

tout programmeur peut d'ailleurs rédiger sa propre licence de logiciel libre avec ou sans clause copyleft.

Il y a deux distinctions fondamentales pour cartographier ce commun : (1) La distinction entre logiciel libre (*free software* ou *open source software*) et logiciel non libre (propriétaire, semi-propriétaire, etc.), qui établit la frontière entre le commun et le reste; (2) La distinction, interne au commun des logiciels libres, entre les logiciels qui sont copyleftés et les logiciels qui sont non-copyleftés. La clause *copyleft* distingue la licence GPL du projet GNU de la plupart des autres licences de logiciels libres. Elle rend en effet irrévocable la décision de partage du code source et maintient donc le développement éventuel du programme dans l'univers du commun[22]. Mais d'autres licences sont plus permissives et autorise une redistribution du code avec une licence qui restreint les droits des utilisateurs. Le nouveau programme peut alors passer dans la catégorie des logiciels libres non libres. C'est le cas par exemple pour les licences X11 (ou *MIT License*), les licences BSD (ou *BSD like*), pour la FreeBSD, pour les licences Apache, et pour beaucoup de licences émanant des universités américaines ou autres[23]. Il existe même une licence de ce type dans le projet GNU, la licence GNU-LGPL. A l'origine, quand cette licence est apparue en juin 1991, l'acronyme LGPL signifiait *Library General Public Licence*; le but étant alors pour la *Free Software Foundation* de faciliter la pénétration de certains logiciels libres dans l'univers des logiciels propriétaires, en permettant de lier un programme sous licence propriétaire, ou sous licence incompatible avec la clause *copyleft*, avec des bibliothèques développées comme logiciels libres[24]. Aujourd'hui, ce nom a changé, car certaines bibliothèques peuvent très bien être sous licence GPL; et pour la FSF, c'est de toute manière préférable.

Le schéma ci-dessous présente alors la cartographie du commun :

22 Il existe d'autres licences intégrant le principe du *copyleft*. En France, le CEA, le CNRS et l'INRIA ont ainsi développé leur propre licence de logiciel libre, la licence CeCILL, qui est présentée comme « *une licence libre de droit français compatible avec la licence GNU-GPL* », et reconnue comme telle par la FSF. Le « texte » de cette licence est cependant critiqué sur le site de la FSF, car « *il utilise des mots biaisés qui doivent être évités, comme « propriété intellectuelle » et « protection »* (…) *Cependant, cela ne pose pas de problème particulier pour les programmes utilisant la licence CeCILL* ».

23 Voir dans l'Annexe C le texte des licences MIT et BSD.

24 Les bibliothèques logicielles sont des collections de composants logiciels qui sont utilisés par d'autres programmes; ce qui évite d'avoir à réécrire à chaque fois un code que plusieurs applications peuvent utiliser. Sur le danger des bibliothèques propriétaires pour le projet de constitution le projet GNU, voir Stallman in Behlendorf et alii., 1999, p. 77. La *Free Software Foundation* a produit d'autres licences, comme la *GNU Free Documentation License*, conçue pour les documents accompagnant les logiciels, mais qui peut aussi être utilisée aussi pour d'autres catégories d'œuvres comme les manuels scolaires ou les dictionnaires, ou la *GNU Affero General Public License* (AGPL), une licence de logiciel libre avec *copyleft* pour des programmes utilisés par des réseaux.

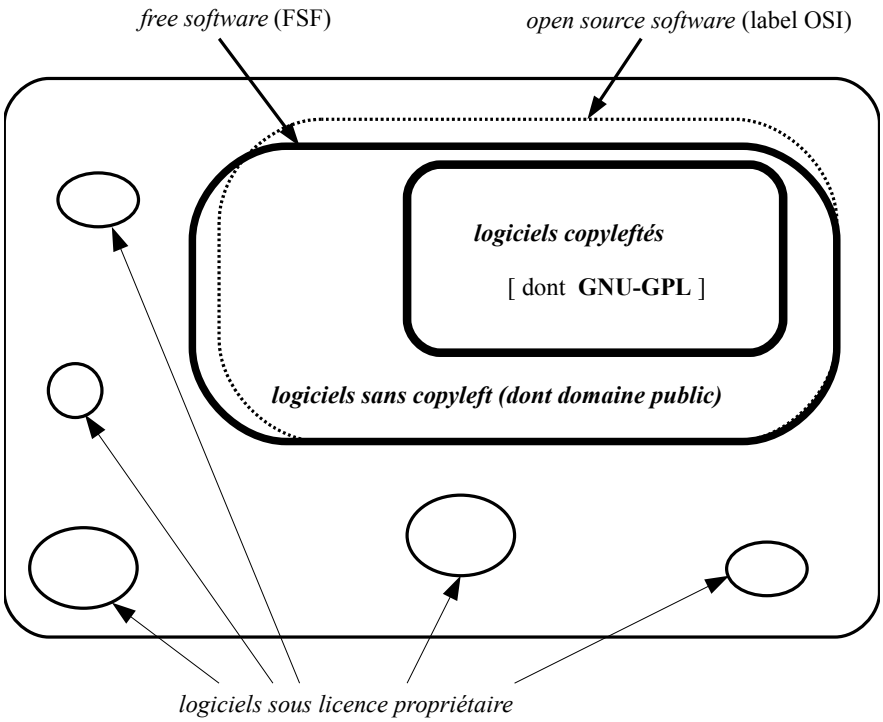


Schéma 2 : Le commun des logiciels libres copyleftés ou non

Il y a, comme nous l'avons déjà dit, des différences minimes entre le logiciel « libre » et le logiciel « *open source* », et la majorité des programmes relevant du commun appartiennent aux deux catégories. La clause *copyleft* représente par contre une distinction et un clivage important, en déterminant à l'avance la manière dont le code doit être redistribué, quand il y a redistribution évidemment, car pour une utn purement privée, cette distinction n'a pas d'importance. La clause inverse en effet le sens habituel des migrations des paquets de code d'un système de règles à l'autre. Ici, nous avons en gros trois systèmes de règles en présence : le système propriétaire, l'utn habituelle des « dpis » sur le mode de l'exclusivité, le système du logiciel libre sans *copyleft*, et le système du logiciel libre avec *copyleft*. Un paquet de code tiré de ce commun peut servir, si la licence n'est qu'une simple licence de logiciel libre non copyleftée, à construire un logiciel propriétaire, dont le code transformé échappe alors au commun. Il peut aussi servir à construire un logiciel libre avec clause *copyleft*. Mais du code copylefté ne peut être utilisé légitimement et légalement ni pour produire un logiciel propriétaire, ni même pour produire un simple logiciel libre (sans clause *copyleft*). La clause règle donc à l'avance le destin du code, au cours des opérations de transformations et développement des programmes et plus généralement des opérations de distribution et redistribution.

Pour préciser les choses au moyen d'exemples, prenons comme seules licences, une licence propriétaire, deux licences de logiciel libre sans copyleft, la licence BSD et la GNU-LGPL, et la

GNU-GPL, avec copyleft. On peut alors avoir les mouvements suivants :

BSD ➡ licence propriétaire,

BSD ➡ BSD et LGPL ➡ LGPL,

BSD ➡ LGPL,

BSD ➡ GPL et LGPL ➡ GPL,

GPL ➡ GPL;

et aussi :

licence propriétaire ➡ BSD (ou LGPL, ou GPL), par dévoilement du code et changement de licence, c'est ce qu'on appelle une « libération du code »[25],

mais on ne peut avoir :

GPL ➡ BSD,

GPL ➡ LGPL,

et encore moins GPL ➡ licence propriétaire.

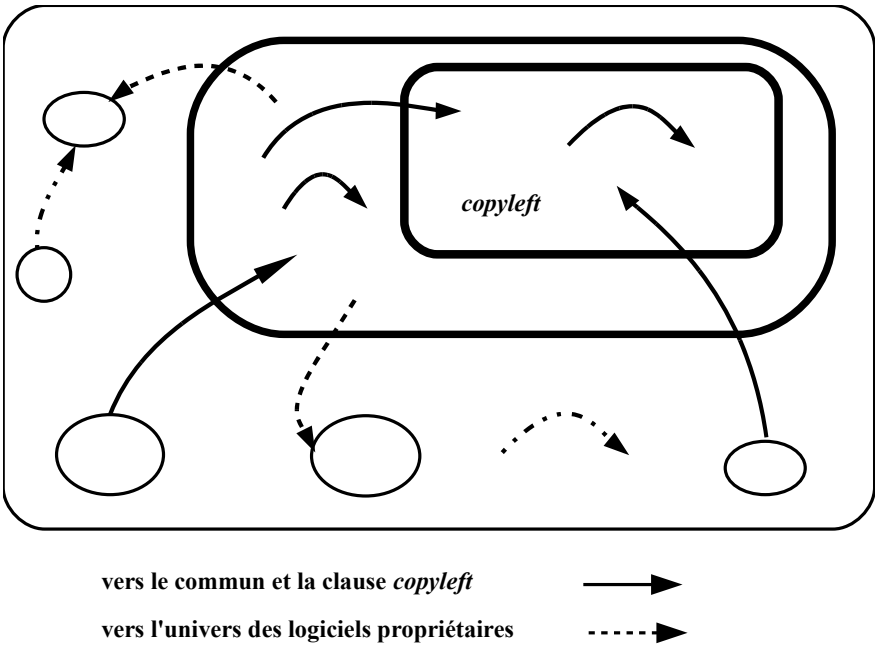


Schéma 3 : Le sens des migrations du code, d'un système de licence à l'autre

Les migrations d'un système de règles à l'autre sont donc ordonnées, avec deux grandes directions possibles (voir schéma) : (1) celle conduisant à la privatisation, qui ne peut puiser ses ressources que dans une partie du commun; (2) et le mouvement inverse qui aboutit au commun sécurisé par le copyleft.

La clause copyleft est d'ailleurs devenue une institution fondamentale du commun des

25 Comme le code de Navigator, le navigateur de Netscape en 1998, ce qui devait conduire à la formation de Mozilla.org (Firefox, Thunderbird, etc.), où celui de StarOffice, qui mis sous LGPL donnera naissance à OpenOffice.org en 2000, celui de Blender, un logiciel d'animation et rendu 3D en 2002, etc.

logiciels libres. Elle en assure la pérennité et le renforcement progressif, en orientant et contraignant à l'avance les mouvements de migrations, les changements possibles du système des règles juridiques accompagnant le code. Cette clause a inversé la représentation traditionnelle, en partie fausse d'ailleurs, du « domaine public », conçu comme une simple communauté négative, un espace de « non propriété », selon l'expression de Landes et Posner (2002), un domaine ou l'on peut donc puiser librement et s'approprier en toute exclusivité sans scrupule. Avec cette clause, le commun devient une communauté positive, excluant toute stratégie propriétaire qui réintroduirait le principe d'exclusivité, et rendant obligatoire le partage, ce qui est parfaitement adapté à la mise sur pied de projets de développement logiciel coopératif .

Développer un logiciel en utilisant du code source provenant de différents logiciels libres pose en effet deux problèmes de coordination, un problème technique et un problème juridique. Au niveau de la programmation et du développement lui-même, il faut rendre compatibles et intégrer dans un ensemble cohérent les différents paquets de code source, etc. Sur le plan juridique, rien n'interdit ce type de travail, inscrit dans les droits de l'utilisateur du logiciel libre, et pour toute utilisation privée, il n'y a même ici en principe aucun obstacle de nature juridique. Mais en cas de redistribution du nouveau code, y compris pour le partage dans une communauté de programmeurs travaillant à un projet commun, il faut aussi que les licences d'origine soient compatibles entre elles, et permettent de définir un ensemble de règles juridiques communes applicables à la combinaison (en pratique, soit une nouvelle licence, soit une ou plusieurs des licences d'origine). C'est bien un problème de coordination juridique, totalement différent du problème précédent. Les clauses spécifiques d'une licence peuvent ainsi faire obstacle à une combinaison de différents code source[26]. Pour sensibiliser les programmeurs à ce problème et aider au développement des logiciels libres, la *Free Software Foundation* distingue alors dans les différentes licences de logiciels libres, celles qui sont compatibles avec la GNU-GPL, et celles qui sont incompatibles. Quand les licences sont juridiquement compatibles, les deux codes peuvent être combinés avec du code sous licence GNU-GPL et l'ensemble redistribué sous licence GNU-GPL ou avec une licence analogue conservant le principe du *copyleft*. On peut par exemple combiner du code appartenant au *public domain*, du code sous licence *ISC* (OpenBSD) et du code sous licence GNU-GPL en mettant le tout sous la même licence GNU-GPL, car tous ces ensembles de règles sont compatibles avec la GNU-GPL. Par contre, on ne peut pas ajouter à cette combinaison du code sous licence *Mozilla Public License 1.1*, pour cause d'incompatibilité juridique, même si l'ensemble peut parfaitement bien fonctionner sur le plan technique. Sur le site de la FSF, on trouve ainsi une liste (non exhaustive) de licences; sur les 110 licences examinées, 82 sont des licences de logiciels libres,

26 Ainsi une licence qui précise que tous les litiges éventuels doivent être tranchés aux Etats-Unis dans l'Etat de Virginie est incompatible avec la licence GNU-GPL, qui exclut toute domiciliation judiciaire particulière (cas de certaines versions de la licence Python), etc. La combinaison de code source dans le cadre d'une même version de la licence GNU-GPL est possible. Il est de même pour le code source sous LGPL, compatible avec la GPL dans le cadre d'une même version. Mais entre les différentes versions (3 et 2 par exemple) existent certains problèmes d'incompatibilité.

43 licences sont compatibles avec la GNU-GPL et 39 incompatibles.

On peut terminer cette cartographie du commun par une mesure statistique de l'importance relative des différentes licences de logiciels libres, une importance mesurée en terme de popularité et adoption par les programmeurs, qui au moment de distribuer leur code, et/ou de lancer un projet de développement logiciel avec appel à la « communauté », doivent choisir ou rédiger eux-mêmes leur licence. La chose la plus frappante est la place centrale occupée par la GNU-GPL de la FSF dans l'ensemble des licences du commun des logiciels libres. L'examen des projets logiciels répertoriés sur le site *Sourceforge* est révélateur. Ce site répertorie un très grand nombre de projets logiciels *free* ou *open source* (plus de 160 000 au moment de notre décompte réalisé en juillet 2009). Il maintient donc en ligne la description du projet, un lien avec l'équipe de développeurs (adresses mèls, etc.), et un accès direct au code source en cours de développement, avec parfois la possibilité de télécharger directement, quand le code produit est suffisamment stable, des versions binaires déjà compilées pour les principaux systèmes d'exploitation. En règle générale, la ou les licences sont précisées; avec la plupart du temps une seule licence par projet, mais certains logiciels combinent parfois plusieurs licences (jusqu'à cinq). On a donc relevé ces différentes licences pour établir deux tableaux (voir l'annexe D), l'un calculé sur l'ensemble des projets présents dans Sourceforge, quelque soit leur vie effective et leur popularité, l'autre sur les « 300 logiciels les plus populaires », autrement dit les plus téléchargés (code source ou binaires).

Dans le décompte initial, il y a 68 items, des licences différentes donc, plus les catégories « *Public domain* », « *Autres OSI-Approved Open Source* », et « *Autres* » (la licence n'étant pas précisée). Mais un certain nombre de licences sont manifestement très peu utilisées. Ainsi les 15 licences les plus utilisées (sur 68) représentent 95 % du nombre total des logiciels. Dans ces 15, celle qui pèse le moins est la MPL 1.0, une des premières versions de la licence MPL du projet Mozilla.org, avec 865 références. A l'opposé, la GNU-GPL, dans toutes ses versions, est en quelque sorte plebiscitée, avec 62 % des projets logiciels; avec sa variante *lesser* au *copyleft* allégé, c'est 72 % des projets qui sont rattachés juridiquement aux règles définies par la *Free Software Foundation*, une simple fondation privée. Le deuxième tableau sur les 300 logiciels « les plus populaires », confirme le premier, avec 74 % pour les licences GNU (63 % pour la GNU-GPL).

Juridiquement, le monde de l'*open source* est donc remarquablement unifié, avec en gros trois situations (et choix) possibles pour le code source d'un logiciel libre : la situation du domaine public où le logiciel « appartient directement au public » et où aucun copyright n'existe, puis la licence style BSD, dénuée de tout *copyleft*, et enfin les licences de la FSF, avec leur principe du *copyleft*, et ici la GNU-GPL domine très largement. Cette unification est remarquable, car elle existe au niveau mondial, et s'est formée en quelque sorte spontanément, sans l'appui d'aucun gouvernement, à l'encontre même du discours dominant des économistes et des hommes politiques sur les bienfaits ou le caractère favorable pour l'innovation de la « propriété intellectuelle », c'est-à-dire du droit exclusif des producteurs (et des firmes) au contrôle de toutes

les utilisations de leurs œuvres (code source et binaires). Au moment où dans des négociations internationales autour de l'Organisation Mondiale de la Propriété Intellectuelle ou de l'OMC (accords ADPIC), on peinait à essayer d'unifier les principes, les règles et les législations, les développeurs de l'*open source*, en adoptant massivement la même licence et les mêmes principes, se sont dotés d'une sorte de statut juridique mondial du logiciel libre.

On peut se demander pourquoi et comment un tel résultat ? Quelles sont les causes de cette unification ? Et pourquoi cette cristallisation autour de la licence GNU-GPL, et non autour d'une autre licence (*BSD* par exemple[27]), ou plusieurs licences différentes ? Et pourquoi les tendances centrifuges qui opèrent en permanence, avec régulièrement l'apparition de nouvelles licences, ne l'emportent-elles pas ?

Une explication des explications possibles du succès de la GNU-GPL est sans doute son lien avec le projet GNU de la *Free Software Foundation* de construction d'un système complet (de plus en plus étoffé) de logiciels libres autour d'un (ou plusieurs) système d'exploitation Unix, tous ces logiciels étant sous licence GNU-GPL. Le statut de cette licence serait alors largement le résultat du travail accompli. Une autre explication est le caractère « général » et volontairement universel de cette *General Public License*, et du discours « philosophique » qui l'accompagne. Le principe du *copyleft* suscite en lui-même une forte adhésion. Il assure que toutes les modifications apportées au code seront, sauf usage purement privé, redonnées à la communauté, avec pour tous les mêmes droits d'usage que sur le code d'origine. Il met donc tous les utilisateurs sur le même plan en leur donnant les mêmes droits, ce qui a permis de mettre sur pied, sous cette licence GPL, des projets de développement de très grande ampleur, mobilisant de très nombreux développeurs répartis un peu partout dans le monde, comme Linux[28]. Et cette licence est générale dans sa rédaction, d'un point de vue purement juridique. Il n'y a pas de clause renvoyant à une législation particulière, ni de « clause publicitaire » (comme dans certaines versions de la licence BSD), et ainsi de suite. Bien que conçue dans le contexte juridique du copyright américain, elle peut être adoptée telle quelle par les programmeurs de tous les pays[29].

La clause *copyleft* est de plus contaminante, puisque tout nouveau logiciel utilisant du code source sous GNU-GPL doit être mis sous licence GNU-GPL (ou sous une licence équivalente

27 Dans les communautés du logiciel libre, certains refusent la clause *copyleft,* qui leur semble philosophiquement trop contraignante, d'autres pour des raisons plus pragmatiques et stratégiques. L'alternative est alors en général le choix d'une licence dite BSD, inspirée de la vieille licence de l'Université de Berkeley mais dans une version nouvelle, sans l'inconvénient de la clause publicitaire.

28 Le code du système d'exploitation Linux au début de son développement (octobre 1991) était sous une licence libre qui interdisait strictement toute distribution commerciale; Linus Torvalds voulant protester alors contre le fait qu'à l'époque il n'existait aucun logiciel bon marché destiné aux étudiants. Au printemps 1992, le code fut mis sous GPL au printemps 1992, la GPL garantissant que quiconque travaille sur le code dans l'avenir en ferait profiter la communauté; et Linus Torvalds devait déclarer plus tard : « *Protéger Linux par la GPL est la meilleure décision que j'aie jamais prise* » (Yamagata, 1997).

29 Depuis l'affaire Educaffix contre CNRS, Université Joseph Fourrier et autres (TGI Paris 28 mars 2007), les tribunaux français reconnaissent la validité de la GNU-GPL (y compris la clause *copyleft*); une décision importante, car il était à craindre que la France, l'un des pays les plus protecteurs en matière de droit d'auteur, considère la licence libre comme nulle en droit français. Cette jurisprudence a été confirmée depuis dans l'affaire AFPA contre la société EDU 4 (Cour d'appel de Paris - 16 septembre 2009). [source : Staub & Associés].

accordant les mêmes droits), ce qui décourage de choisir une autre licence, et renforce la tendance à l'unification. Les règles de compatibilité entre licences poussent dans la même direction. Le jeu des rendements croissants d'adoption contribue donc à faire de cette licence la référence en matière juridique des communautés *open source*, et donc une forme de standard, qui donne au code libre une unité en matière juridique, facilitant sa circulation et son ré-emploi, sans avoir alors à étudier dans le détail les clauses de différentes licences, à tenir compte des frontières, etc.

(à suivre...)

Deuxième partie prévue : La vie du commun

Je pense traiter de la gestion des licences et des migrations, plus généralement de l'organisation et la gestion du commun; de la production des logiciels libres : les grands projets et leurs formes d'organisation spécifiques; de l'importance du commun pour l'innovation logicielle : ré-emploi du code, trajectoires, embranchements (*forks*), projets concurrents, etc.; et tenter un bilan sur la diffusion et l'intégration du logiciel libre dans l'ensemble de l'économie informatique.

Références :

Abate, Janet, 1999, *Inventing the internet*, The MIT Press, Cambridge.

Akera, Atsushi, 2001, « Volontarism and the fruits of collaboration : The IBM' user group, Share », *Technology and Culture*, 42(9), october, pp. 710-736.

Arrow, Kenneth, 1962, « Economic Welfare and the Allocation of Resources for Innovation », in Nelson, editor, *The Rate and Direction of Inventive Activity*. Universities-National Bureau, p. 609-626.

Belhendorf B., S. Bradner, J. Hamerly, K. McKusick, T. O'Reilly, T. Paquin, B. Perens, E. Raymond, R. Stallman, M. Tiermann, L. Torvalds, P. Vixie, L. Wall, B. Young, 1999, *Tribune libre, Ténors de l'informatique libre*, Editions O'Reilly, Paris.

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.

Campbell-Kelly Martin, 2003, *Histoire de l'industrie du logiciel, des réservations aériennes à Sonic le hérisson*, MIT, trad. fr. Vuibert.

Clément-Fontaine Mélanie, 1999, *Une étude juridique de la Licence Publique Générale GNU*, mémoire de DEA en Droit, < http://www.crao.net/gpl/ >

Dréan Gérard, 1996, *L'industrie informatique. Structure, économie, perspectives*, Masson, Paris.

Foray Dominique, Zimmermann Jean-Benoît, 2001, « L'économie du logiciel libre. Organisation coopérative et incitation à l'innovation », *Revue Economique*, 52, numéro hors série, p.77-93.

Hersey, John, 1978, « Computers and copyright (ch 3), Dissent of commissioner Hersey », *Final Report of the National Commission on New Technology Uses of Copyrighted Works (CONTU)*, < http://digital-law-online.info/CONTU/contu14.html >

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.

Levy, Steven, 1985, *Hackers: Heroes of the Computer Revolution*, O'Reilly Media, réédition 2010.

Merges, Robert P., 2000, "One Hundred Years of Solicitude: Intellectual Property Law 1900-2000", *California Law Review*, vol. 88, pp. 2187-2240.

Moglen Eben, 2001, "L'anarchisme triomphant : Le logiciel libre et la mort du copyright", < http://emoglen.law.columbia.edu/publications/anarchism-fr.html >

Perens, Bruce, 1998, *La définition de l'open source*, in *Tribune libre*, 1999, p. 183-203.

Salus Peter H., 1994, *A Quarter Century of UNIX*, Addison-Wesley Publishing Company, Inc.

Stallman, Richard M., 1999, « Le système d'exploitation du projet GNU et le mouvement du logiciel libre », in *Behlendorf et alii*., 1999, p. 61-82.

Yamagata, Hiroo, 1997, « Le pragmatiste du logiciel libre : entretien avec Linus Torvalds »,
< http://www.linux-france.org/article/these/interview/torvalds/pragmatist-fr.html >

Sites internet :

< http:www.gnu.org/philisophy/philosophy.fr.html >
< http:emoglen.law.columbia.edu >
< http://www.gnu.org/licenses/license-list.html >
< http://www.fsf.org > (sote de la *Free Software Foundation*)
< http://opensource.org > (site de l'*Open Source Initiative*)

Annexes

A - Note technique et définitions

B - Commission CONTU – Opinion dissidente de John Hersey

C - Licences de logiciels libres (tirées du système d'exploitation de Mac OS X)

D - Un décompte statistique sur Sourceforge

E - Décompte des licences de Mac OSX

Annexe A : Note technique et définitions

Les logiciels (ou programmes) sont des ensembles d'instructions adressées à une machine informatique (ordinateur), qui se présentent toujours au départ sous forme de texte, autrement dit, un ensemble de signes ou symboles disposés linéairement. En règle générale, le logiciel est d'abord écrit au moyen d'un langage de programmation, plus ou moins proche du langage naturel (français, anglais, etc.), et donc directement compréhensible par l'être humain (programmeur[1]). C'est le « code source », un code indépendant de la machine à laquelle est destiné le programme. Ce code source doit ensuite être traduit en « langage machine », c'est-à-dire dans le langage natif du processeur. Pour réaliser cette transformation, sans laquelle le logiciel ne serait qu'un texte inerte, et ne servirait pas à grand chose, il y a deux possibilités :

(1) La compilation, au moyen d'un logiciel spécifique appelé compilateur, qui transforme le code source en produisant un fichier binaire (code machine), lequel peut être directement lu et exécuté par l'ordinateur à la différence du code source lui-même. Une application écrite en langage C, C++ ou un autre langage, peut ainsi être transformée par un compilateur - GCC par exemple - en différentes versions exécutables destinées à des systèmes d'exploitation, des environnements graphiques (GUI), des processeurs différents : une version pour Windows XP par exemple, une (ou deux) pour Mac OSX en différenciant (ou non) les processeurs PowerPC ou Intel, plusieurs pour Linux suivant le type d'environnement graphique, le système de bureau, etc.

(2) Une autre possibilité est l'interprétation, qui existe pour certains langages. Un logiciel spécifique décode (ou interprète) à la volée les instructions du code source en les transforment au fur et à mesure en instructions spécifiques pour exécution immédiate. C'est ce que font par exemple tous les navigateurs web (Firefox, Internet Explorer, Safari, Opéra, etc.); ils téléchargent en effet une simple feuille de code source écrite en langage html, et quelques fichiers binaires (images, sons, etc.), et interprètent le tout immédiatement - en pratique plus ou moins vite et de manière éventuellement différente suivant le navigateur considéré.

Un logiciel compilé est plus rapidement exécuté par la machine en règle générale qu'un logiciel interprété. Certains programmes, dont les systèmes d'exploitation, jeux, etc., ne reposent ainsi que sur des langages permettant la compilation, car même si cette opération peut parfois être fort longue, les gains de temps lors de l'exécution participent de la qualité du programme. Avec l'apparition du langage Pascal et de compilateurs commerciaux rapides comme Turbo Pascal à partir du milieu des années 1980, les langages interprétés connurent un fort déclin. Mais cette situation a changé aujourd'hui et ceux-ci sont de plus en plus utilisés, comme Perl, Python, Ruby, etc. (il s'agit en fait souvent de langages semi-interprétés[2]) . L'augmentation de la puissance des machines assure en effet aux programmes interprétés une rapidité comparable à celle des programmes compilés de la décennie précédente. De plus, l'automatisation rapide de certaines tâches complexes, avec des besoins de transformation presque continuelle, favorise cet essor. Il est en effet bien plus facile de faire évoluer un programme interprété, l'opération intermédiaire de compilation étant supprimée.

Dans les années 1940-1950, la programmation se faisait d'ailleurs directement en langage machine, sous forme de groupes de nombres binaires (*bits*), puis en utilisant - ce qui est encore le cas dans certaines situations - un code d'assemblage, un langage de très bas niveau où chaque

1 « *C'est un moyen de communiquer avec la machine et aussi un moyen de communiquer avec les autres programmeurs* » (Moglen, 2001). Le code source est d'ailleurs parsemés de commentaires rédigés en langage naturel, qui lors de la compilation, sont laissés de côté et n'influencent en aucune manière la production du fichier binaire nécessaire à la machine. Car ces commentaires sont uniquement destinés aux êtres humains, au programmeur lui-même qui peut se rappeler ainsi ce qu'il a fait et ce qu'il voulait obtenir, ou aux autres programmeurs qui pourraient avoir à modifier ce programme éventuellement aujourd'hui ou demain.

2 Certains sont en effet semi-compilés et/ou semi-interprétés, le programme une fois lancé produisant de lui-même la première fois une version partiellement compilée, laquelle est conservée en mémoire et utilisée par la suite par la machine. En cas de modification du code source, ce processus de compilation initiale est relancé. C'est le cas pour Python et Java, deux langages interprétés, très utilisés de nos jours.

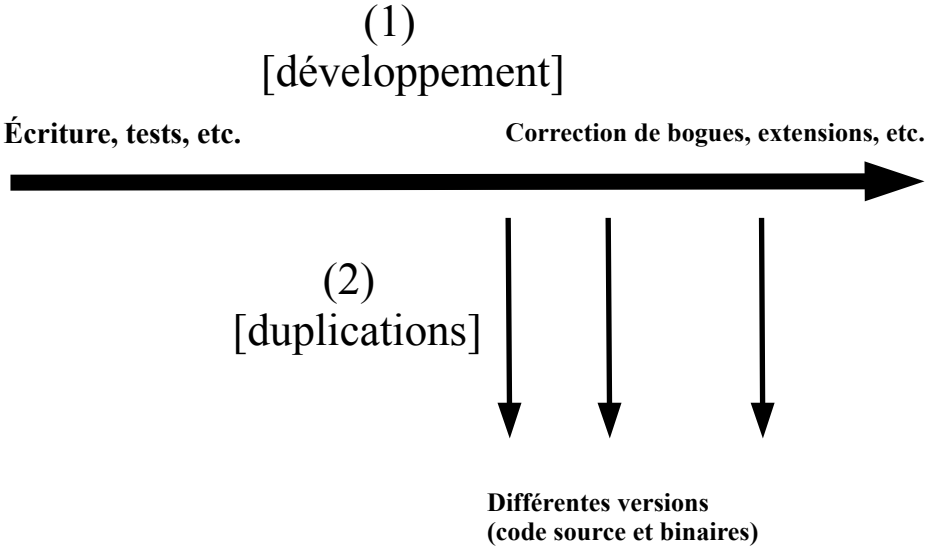
instruction élémentaire est notée par un symbole mnémotechnique, plus facile à retenir par les êtres humains que les nombres binaires. L'invention de langages plus élaborés (Fortran (1957), Pascal, C, etc.), plus proches du langage naturel et du langage mathématique, et parallèlement de compilateurs capables de le traduire et d'optimiser le code ainsi produit, va cependant changer complètement cette situation, représentant d'ailleurs une des innovations majeures de l'industrie de l'informatique et des logiciels.

Economiquement, on peut (et on doit) distinguer deux processus de production :

(1) Le développement du programme, c'est-à-dire toutes les activités d'écriture, ré-écriture, compilation, tests, etc., qui contribuent à la production et à la transformation progressive d'un objet unique, le logiciel, représenté ici essentiellement par son code source. Ce processus peut être fort long et s'étale nécessairement dans le temps, pour de multiples raisons techniques.

(2) Les opérations de copies (ou duplication) de ce programme, dans l'état où il se trouve à un moment donné du temps, une copie donc du code source, ou une copie d'une version directement exécutable du programme (fichiers binaires) après compilation du code source; ces copies n'affectant pas le programme initial.

La duplication (production 2) est généralement peu coûteuse et limitée au coût du support (bande magnétique, disquettes, CD, etc.) et au temps d'enregistrement, auquel il faut ajouter cependant la reproduction de la documentation. Ce coût est devenu négligeable avec l'apparition des téléchargements utilisant l'internet, et de plus directement supporté par celui qui réalise la copie; la documentation associée étant souvent elle-même sous forme électronique. Le développement (production 1) est à l'inverse un processus qui mobilise toujours un ou plusieurs programmeurs et représente souvent un grand nombre d'heures de travail en programmation (écriture, tests, etc.). Ce développement peut même, dans les projets de taille un peu importante, mobiliser un très grand nombre de programmeurs et s'étaler sur plusieurs années. Il est donc toujours coûteux.



Annexe B : Commission CONTU :

Opinion dissidente de John Hersey : The Act of 1976 should be amended to make it explicit that copyright protection does not extend to a computer program in the form in which it is capable of being used to control computer operations (chapitre 1 du rapport final).
< http://digital-law-online.info/CONTU/contu2.html >

Extraits de son argumentation :
(chapitre 3 du rapport final)

This dissent from the Commission report on computer programs takes the view that copyright is an inappropriate, as well as unnecessary, way of protecting the usable forms of computer programs. Its main argument, briefly summarized, follows.

In the early stages of its development, the basic ideas and methods to be contained in a computer program are set down in written forms, and these will presumably be copyrightable with no change in the 1976 Act. But the program itself, in its mature and usable form, is a machine-control element, a mechanical device, which on constitutional grounds and for reasons of social policy ought not be copyrighted.

The view here is that the investment of creative effort in the devising of computer programs does warrant certain modes of protection for the resulting devices, but that these modes already exist or are about to be brought into being under other laws besides copyright; that the need for copyright protection of the machine phase of computer programs, quite apart from whether it is fitting, has not been demonstrated to this Commission; and that the social and economic effects of permitting copyright to stand alongside these other forms of protection would be, on balance, negative.

The heart of the argument lies in what flows from the distinction, raised above, between the written and mechanical forms of computer programs: admitting these devices to copyright would mark the first time copyright had ever covered a means of communication, not with the human mind and senses, but with machines.

Are Mature Programs "Writings"?

Programs are profoundly different from the various forms of "works of authorship" secured under the Constitution by copyright. Works of authorship have always been intended to be circulated to human beings and to be used by them-to be read, heard, or seen, for either pleasurable or practical ends. Computer programs, in their mature phase, are addressed to machines.

All computer programs go through various stages of development. In the stages of the planning and preparation of software, its creators set down their ideas in written forms, which quite obviously do communicate to human beings and may be protected by copyright with no change in the present law.

But the program itself, in its mature and usable form, is a machine-control element, a mechanical device, having no purpose beyond being engaged in a computer to perform mechanical work.

The stages of development of a program usually are: a definition, in eye-legible form, of the program's task or function; a description; a listing of the program's steps and/or their expression in flow charts; the translation of these steps into a "source code," often written in a high-level programming language, such as FORTRAN or COBOL; the transformation of this source code within the computer, through intervention of a so-called compiler or assembler program, into an "object code." This last is most often physically embodied, in the present state of technology, in punched cards, magnetic disks, magnetic tape, or silicon chips-its mechanical phase.

Every program comes to fruition in its mechanical phase. Every program has but one purpose and use – one object: to control the electrical impulses of a computer in such a particular way as to carry out a prescribed task or operation. In its machine-control form it does not describe or give directions for mechanical work. When activated, it does the work.

An argument commonly made in support of the copyrightability of computer programs is that they are just like ordinary printed (and obviously copyrightable) lists of instructions for mechanical work. The computer

report calls programs forms of writing which "consist of sets of instructions." But this metaphor does not hold up beyond a certain point. Descriptions and printed instructions tell human beings how to use materials or machinery to produce desired results. In the case of computer programs, *the instructions themselves eventually become an essential part of the machinery that produces the results.* They may become (in chip or hardware form) a permanent part of the actual machinery; or they may become interchangeable parts, or tools, insertable into and removable from the machine. In whatever material form, the machine-control phase of the program, when activated, enters into the computer's mechanical process. This is a device capable of commanding a series of impulses which open and close the electronic gates of the computer in such order as to produce the desired result.

Printed instructions explain *how* to do something; programs are *able* to do it. The language used to describe and discuss computer programs commonly expresses this latter, active, functional capability, not the preparatory "writing" phases. For example, the Commission's report on new works uses the following verbs to characterize the doings of various programs in computers: *select, arrange, simulate, play, manipulate, extract, reproduce*, and so on. It is not said that the programs *describe* or *give instructions for* the functions of the computer. They *control* them. This is the mechanical fact.

Issue of Communication

The Commission report on computer programs suggests that musical recordings also do work, analogous to what we have been describing. "Both recorded music and computer programs are sets of information in a form which, when passed over a magnetized head, cause minute currents to flow in such a way that desired physical work is accomplished." But these are radically different orders of work, and the difference touches on the very essence of copyright.

We take it as a basic principle that copyright should subsist in any original work of authorship that is fixed in any way (including books, records, film, piano rolls, videotapes, etc.) which communicate the work's means of expression. But a program, once it enters a computer and is activated, does not communicate information of its own, intelligible to a human being. It utters work. Work is its only utterance and its only purpose. So far as the mode of expression of the original writing is concerned, the matter ends there; it has indeed become irrelevant even before that point. The mature program is purely and simply a mechanical substitute for human labor.

The functions of computer programs are fundamentally and absolutely different in nature from those of sound recordings, motion pictures, or videotapes. Recordings, films, and videotape produce for the human ear and/or eye the sounds and images that were fed into them and so are simply media for transmitting the means of expression of the writings of their authors. The direct product of a sound recording, when it is put in a record player, is the sound of music-the writing of the author in its audible form. Of film, it is a combination of picture and sound-the writing of the author in its visible and audible forms. Of videotape, the same. But the direct product of a computer program is a series of electronic impulses which operate a computer; the "writing" of the author is spent in the labor of the machine. The first three communicate with human beings. The computer program communicates, if at all, only with a machine.

And the nature of the machine that plays the second recording is fundamentally and absolutely different from that of the machine that uses software. The record player has as its sole purpose the performance of the writing of the author in its audible form. The computer may in some instances serve as a storage and transmission medium for writings (but different writings from those of the computer programmer – i.e., data bases) in their original and entire text, in which cases these writings may be adequately secured at both ends of the transaction by the present copyright law. But in the overwhelming majority of cases its purposes are precisely to use programs to transform, to manipulate, to select, to edit, to search and find, to compile, to control and operate computers and a vast array of other machines and systems, with a result that the preparatory writings of the computer programmer are nowhere to be found in recognizable form, because the program has been fabricated as a machine control element that does these sorts of work. It is obvious that the means of expression of the preparatory writing-that which copyright is supposed to protect-is not to be found in the computer program's mechanical phase.
(…)
< http://digital-law-online.info/CONTU/contu14.html#sec2.1.9. >

Quelques licences de logiciels libres trouvées dans le système d'exploitation Mac OS X (Tiger)

Source : Disque dur > Bibliothèque > Documentation > Aknowledgements.rtf.

- **Deux formulations très simples (licences permissives sans clause *copyleft*)**

Dave Coffin (dcraw)

```
Copyright © 1997-2004 by Dave Coffin, dcoffin a cybercom o net
This is a portable ANSI C program to convert raw image files from any digital
camera into PPM format.  TIFF and CIFF parsing are based upon public
specifications, but no such documentation is available for the raw sensor data,
so writing this program has been an immense effort.  This code is freely
licensed for all uses, commercial and otherwise.  Comments, questions, and
encouragement are welcome.
```

World Wide Web Consortium (tidylib)

```
Copyright © 1998-2003 World Wide Web Consortium (Massachusetts Institute of
Technology, European Research Consortium for Informatics and Mathematics, Keio
University).  All Rights Reserved.
This software and documentation is provided "as is," and the copyright holders
and contributing author(s) make no representations or warranties, express or
implied, including but not limited to, warranties of merchantability or fitness
for any particular purpose or that the use of the software or documentation will
not infringe any third party patents, copyrights, trademarks or other rights.
The copyright holders and contributing author(s) will not be held liable for any
direct, indirect, special or consequential damages arising out of any use of the
software or documentation, even if advised of the possibility of such damage.
Permission is hereby granted to use, copy, modify, and distribute this source
code, or portions hereof, documentation and executables, for any purpose,
without fee, subject to the following restrictions:
1. The origin of this source code must not be misrepresented.
2. Altered versions must be plainly marked as such and must not be
misrepresented as being the original source.
3. This Copyright notice may not be removed or altered from any source or
altered source distribution.
The copyright holders and contributing author(s) specifically permit, without
fee, and encourage the use of this source code as a component for supporting the
Hypertext Markup Language in commercial products. If you use this source code in
a product, acknowledgment is not required but would be appreciated.
```

- Licence BSD (Berkeley Software Distribution) avec clause publicitaire :

University of California, et. al. (BSD Operating System)

```
This product includes software developed by the University of California,
Berkeley, and its contributors.
Copyright © 1980-1997 The Regents of the University of California.
Copyright © 1994 Ugen J.S.Antsilevich.
Copyright © 1998 Apple Computer, Inc.
Copyright © 1994 The Australian National University.
Copyright © 1986 by Robert V. Baron.
Copyright © 1991, 1992, 1993, 1994, 1995, 1996 Keith Bostic.
Copyright © 1993 Daniel Boulet.
Copyright © 1992, 1993 John Brezak.
Copyright © 1996 Charles D. Cranor and Washington University.
Copyright © 1994-1996 Cronyx Engineering Ltd.
Copyright © 1988, 1989 by Adam de Boor.
Copyright © 1992,1993 Theo de Raadt <deraadt@fsa.ca>.
Copyright © 1989 Stephen Deering.
```

Systems Programming Group of the University of Utah Computer Science Department; the UCLA Ficus project; the Laboratory for Computation Vision and the Computer Science Department of the the University of British Columbia and the Computer Science Department (IV) of the University of Erlangen-Nuremberg, Germany; Scooter Morris at Genentech Inc.; Chuck Karish of Mindcraft, Inc.; Mike Karels at Berkeley Software Design, Inc.
The United States Government has rights in parts of this work pursuant to contract no. DE-AC03-76SF00098 between the United States Department of Energy and the University of California.
Some code is derived from software contributed to Berkeley by the Center for Software Science of the University of Utah Computer Science Department. CSS requests users of this software to return to css-dist@cs.utah.edu any improvements that they make and grant CSS redistribution rights.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the University of California, Berkeley and its contributors. 4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

> [Analyse : Cette licence couvre le système d'exploitation BSD. Le point 3 constitue la « clause publicitaire » si controversée, car susceptible de poser problème, comme on le voit ici, en cas de développement prolongé du logiciel. Ce document est intéressant par ailleurs, car il donne une idée de l'importance du travail coopératif engagé : un grand nombre de contributeurs, éparpillés un peu partout dans le monde (Etats-Unis, Australie, Canada, Irlande, Angleterre, Allemagne), et une durée de la coopération de 1980 (*The Regents of the University of California)* à 1998.]

- **Exemples de nouvelles licences BSD, sans clause publicitaire :**

Tito Ciuro (QuickLite)
Copyright © 2004, Tito Ciuro. All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
• Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
• Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
• Neither the name of Tito Ciuro nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE

DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Digital Equipment Corporation (bind, BSD kernel)

Portions Copyright © 1993 by Digital Equipment Corporation.
Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies, and that the name of Digital Equipment Corporation not be used in advertising or publicity pertaining to distribution of the document or software without specific, written prior permission.
THE SOFTWARE IS PROVIDED AS IS AND DIGITAL EQUIPMENT CORP. DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL DIGITAL EQUIPMENT CORPORATION BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

• Appel à la GNU GPL

Chris Allegretta, et al. (nano)

Copyright © 1999-2004 Chris Allegretta, et al.
The nano software is free software and is licensed under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. You may obtain a complete machine-readable copy of the source code for the nano software under the terms of GPL, without charge except for the cost of media, shipping, and handling, upon written request to Apple. The nano software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GPL for more details; a copy of the GPL is included with this product.

Free Software Foundation (various GNU projects, including autoconf, automake, bash, bc, bison, cvs, diffutils, emacs, gas, gcc, gdb, gm4, gnumake, gnutar, gperf, grep, groff, patch, screen, texi2html, texinfo)

Copyright © 1988-2003 Free Software Foundation, Inc. (FSF).
Parts of this product include certain software owned by the FSF and licensed by Apple. You may obtain a complete machine-readable copy of the source code for the FSF software under the terms of GNU General Public License (GPL), without charge except for the cost of media, shipping, and handling, upon written request to Apple. The FSF software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

[Pour une copie de la Licence GNU GPL, voir le site de la FSF < http://www.gnu.org/licenses/gpl.html >, et un extrait ci-dessous]

Extraits de la licence GNU GPL V2 (clause 1 et 2 (copyleft))

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an

appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program,and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

- **Choix entre la licence d'origine ou la GNU GPL :**

Brian Gladman (AES implementation, SHA2 Message Digest)

```
Copyright © 2003, Dr Brian Gladman, Worcester, UK.   All rights reserved.
LICENSE TERMS
The free distribution and use of this software in both source and binary form is
allowed (with or without changes) provided that:
1. distributions of this source code include the above copyright notice, this
list of conditions and the following disclaimer;
2. distributions in binary form include the above copyright notice, this list of
conditions and the following disclaimer in the documentation and/or other
associated materials;
3. the copyright holder's name is not used to endorse products built using this
software without specific written permission.
ALTERNATIVELY, provided that this notice is retained in full, this product may
be distributed under the terms of the GNU General Public License (GPL), in which
case the provisions of the GPL apply INSTEAD OF those given above.
DISCLAIMER
This software is provided 'as is' with no explicit or implied warranties in
respect of its properties, including, but not limited to, correctness and/or
fitness for purpose.
```

Annexe D : Décompte des licences dans Sourceforge

Tableau 1 : Ensemble des projets présents sur Sourceforge (Juillet 2009)

	Nombre	en %
GNU General Public License (GPL)	101425	62,44
GNU Library or Lesser General Public License (LGPL)	17440	10,74
Licences style BSD	11257	6,93
Licences Apache	6352	3,91
Licences style X11 (ou MIT license)	4263	2,62
Academic Free License (AFL)	2812	1,73
Licences Artistic	2030	1,25
Mozilla Public licence	2385	1,47
Common Public License 1.0	1394	0,86
Open Software License 3.0 (OSL3.0)	1261	0,78
Domaine Public	5896	3,63
Autres licences (moins de 1000 projets)	5913	3,64
TOTAL (autres (NP) exclus)	162428	100
Autres (licences non précisées)	1889	

Tableau 2 : Les 300 logiciels les plus téléchargés "*all time*"

	Nombre	en %
GNU General Public License (GPL)	203	63,24
GNU Library or Lesser General Public License (LGPL)	44	13,71
BSD License	12	3,74
Mozilla Public License	12	3,74
zlib/libpng license	11	3,43
Autres (< 10 références)	39	12,15
Total	321	100

[Ces tableaux ont été établis en juillet 2009]

Annexe E : Décompte des licences de Mac OS X (Tiger)

	Nombre	en %
GNU GPL	49	30,8
Licences style BSD	34	21,4
Autres licences libres	54	33,9
Droits réservés (non libres)	22	13,8
Total	159	100

(source : « Acknowledgements.rtf » - 15 décembre 2010)

Note :

Il y a 159 paragraphes dans la page « Acknowledgements ». Chaque paragraphe correspond à un ou plusieurs titulaires de copyright et une licence (qui parfois permet cependant de choisir entre différentes licences types; voir les exemples dans l'Annexe C). Certaines de ces licences portent sur un seul logiciel, d'autres sur plusieurs.

Ainsi la *Free Software Foundation* est créditée quatre fois (pour trois licences GNU GPL et une licence GNU LGPL), mais un des crédits porte « *de nombreux projets GNU* », dont une vingtaine, et non des moindres, sont cités : on y trouve le shell bash, l'interpréteur des lignes de commande du système, le compilateur GCC, etc.

La statistique ci-dessus reflète aussi l'origine historique du système exploitation, qui contient des parties directement issues de FreeBSD et NetBSD, et d'autres éléments produits à partir du système Unix développé à l'Université de Berkeley. Ces éléments, d'autres venus d'ailleurs (du projet GNU entre autres) et un micro-noyau Mach mis au point à l'Université Carnegie-Mellon furent incorporés ensuite dans NEXTSTEP (l'origine la plus directe de Mac OS X) un système d'exploitation développé par NEXT, une société fondée par Steve Jobs après son départ d'Apple en 1985, rachetée en 1997 par Apple, Steve Jobs ayant réintégré cette firme.