Logiciel libre et innovation technique

Bastien Guerry

avril 2001

> « Le cerveau ne fait pas apparaître miraculeusement la conscience, l'intelligence, le comportement orienté, dans un univers stupide et aveugle, et dans un organisme aussi stupide que cet univers. Il fait déborder l'organisation, instinctive et intelligente, déjà à l'oeuvre dans l'organisme, sur le monde extérieur. Il n'invente, en technique externe, que par la même invention organique par laquelle il s'est formé d'abord lui-même, et selon la même logique matricielle. »
>
> Raymond Ruyer [Ruy77, p. 151]

> « La raison est l'organe qui met l'accent sur la nouveauté. »
>
> Alfred North Whitehead [Whi69, p.115]

1 Introduction

Les textes abondent sur le thème des logiciels libres et du mouvement *open source* dans son ensemble. Les médias du monde informatique en font un sujet à la mode, les fervents défenseurs de ce mouvement en soulignent la portée idéologique, et ceux qui cherchent à gagner le marché en analysent minutieusement le modèle économique.

Du point de vue idéologique, le monde du logiciel libre est fortement marqué par certains mots d'ordre structurant la culture des *hackers*[1]: rendre l'ingénierie informatique transparente, libérer l'information, et éviter que les ressources techniques ne subissent la mainmise de certains monopoles (étatiques ou privés). Ce discours idéologique porte aussi bien sur la liberté revendiquée de pouvoir comprendre les procédés techniques que sur la libération des contenus véhiculés par les réseaux informatiques.

1. Le terme de hacker désigne ici tout passionné d'ingénierie informatique.

Du point de vue économique, le monde des logiciels libres défend schématiquement deux idées: d'une part celle d'une saine concurrence dans le domaine de la création de logiciel par l'ouverture des codes sources; d'autre part l'idée que l'accent doit être mis sur les services entourant les systèmes informatiques (services d'installation et d'assistance) plutôt que sur les revenus liés à la vente des logiciels. Les quelques réussites prometteuses de cette nouvelle approche économique de l'informatique[2] viennent solidement alimenter ce discours, montrant ses capacités d'anticipation.

Dans l'entrecroisement de ces deux rhétoriques, le mot qui revient le plus souvent est celui de « liberté »: qu'il s'agisse de libérer les codes sources ou de libérer le marché d'une concurrence malsaine, l'enjeu semble clairement contenu dans ce seul mot de « libre ». Mais au milieu de ces deux genres dominants de discours - idéologique et économique - existe pourtant une réalité qui mettrait plutôt l'accent sur les contraintes: la réalité technique.

Pour l'usager (particulier ou entreprise), le passage d'un système propriétaire à un système libre représente une contrainte. Pour le concepteur, le développement d'applications pour lesquelles quelques standards lui manquent représente une contrainte. Pour l'entrepreneur vivant de logiciel libre, le développement d'une rhétorique nouvelle, la mise en place de dispositifs de démonstration, la mise en confiance du client, tout cela représente de nouvelles contraintes de ventes auxquelles il devra s'adapter. Loin de nous l'idée que ces contraintes sont l'envers négatif des avantages indéniables du logiciel libre. Le gain de performance résultant du passage à un système libre, le plaisir d'y développer plus efficacement des applications libres, le sentiment d'élaborer un rapport plus rationnel à l'outil informatique par le biais d'un modèle économique nouveau: tout cela montre assez que les contraintes techniques participent de manière positive aux bénéfices que l'on peut tirer du logiciel libre. Mais d'une manière générale, pour aborder un système technique, il est utile de comprendre les différentes contraintes qu'il impose et par lesquelles il fonctionne: au concepteur désireux de participer à son élaboration, à l'entrepreneur souhaitant le promouvoir et le diffuser, à l'usager ayant choisi ce système pour parvenir à ses fins, etc.

Si nous insistons un instant sur ces contraintes, c'est seulement pour souligner l'existence des logiciels libres comme « objets techniques »[3] nouveaux, se développant et se diffusant selon des modalités particulières, et remettant en cause le rapport ancien que concepteurs et usagers établissaient avec les logiciels. Le mou-

2. La société VA-Linux a fait son entrée en bourse au Nasdaq le 9 décembre 1999: en une journée, une hausse record de 698%.

3. Concept que nous définirons par la suite.

vement du logiciel libre n'est pas seulement un retour idéologique aux conceptions de quelques précurseurs - même si des précurseurs tels que Richard Stallman ont impulsé le mouvement et le soutiennent toujours avec énergie; et il n'est pas non plus une excroissance supplémentaire du développement des logiciels, une branche juxtaposée aux précédentes, sur laquelle il suffirait aujourd'hui de s'asseoir pour faire de l'argent.

Son enracinement comme système de référence suppose et appelle une véritable *mutation technique*, tant du point de vue des concepteurs que de celui des utilisateurs, car l'innovation technique est autant dans les performances propres de chaque logiciel que dans une remise en cause des différentes manières de s'en servir. Et cette mutation n'est ni magique ni fortuite, mais réellement structurée.

Nous partons donc du constat suivant: un discours global sur le phénomène des logiciels libres comme phénomène technique est moins fréquent que les autres types de discours (économique, sociologique et idéologique). La profusion des débats idéologiques, économiques et sociologiques autour d'une innovation n'est évidemment pas un phénomène propre à celle que les logiciels libres incarnent. Dans toute phase de large diffusion d'une innovation, différents acteurs sociaux réfléchissent sur la manière de vivre cette innovation, sur son moteur idéologique, sur les conséquences qu'elle aura au quotidien, sur ses tenants et aboutissants économiques. Mais trop souvent, il est aussi de mise d'oublier le phénomène technique en tant que tel, de laisser son analyse aux seuls techniciens, ou de parler de « technique » de manière univoque pour désigner des phénomènes hétérogènes [4]. Pour ce qui concerne le logiciel libre, nous expliquons ce constat par quatre raisons valables et trois raisons douteuses. Voici les raisons valables:

- → la crainte de se prononcer dans un domaine exigeant quelques compétences techniques, et dans lequel une erreur de jugement sur une question pointue discrédite l'ensemble de l'approche.
- → la diversité du phénomène, qui le rend difficilement appréhendable dans une analyse globale.
- → les considérations techniques qui prolifèrent déjà à l'échelle du simple utilisateur. De l'ordinateur qui se documente lui-même à l'ensemble des ressources d'information, l'usager croule sous une telle masse d'informations qu'il est difficile de cerner l'ensemble du phénomène dans ses répercussions.
- → la priorité légitime des discours insistant sur ce qui sera le plus immédiatement visible du point de vue de l'utilisateur (priorité stratégique dans la

4. Il suffit de voir comment le terme de "réseau" suffit à donner un coloration technique à certains discours mélangeant des notions très disparates au sein de ce concept flou.

diffusion des logiciels libres)[5].

Voici maintenant quelques raisons qui nous semblent mal venues:

- → sous-estimation de la réelle innovation technique: les logiciels libres seraient des logiciels parmi les autres, avec deux fonctionnalités usuelles, nouvelles et accessoires: leur fréquente gratuité et la liberté d'accès à leur code source.
- → plongée de ce phénomène technique précis dans un tout-technique multiforme et indifférencié: le logiciel libre serait un « effet » dégagé par l'innovation informatique dans son ensemble, une variation épisodique dans la gamme des rapports entre concepteurs, logiciels et utilisateurs.
- → transformation de l'objet technique en boîte noire échappant à l'analyse: l'objet technique ne serait qu'un fait brut avec lequel les différents acteurs de l'innovation auraient à faire. Sans intériorité ni historicité, le logiciel libre surgirait droit des mains du concepteur pour se livrer à un processus linéraire de diffusion.

De toutes les fausses raisons invoquées ici, la première est évidemment la plus grave, mais les deux autres n'en sont pas moins menaçantes, car elles impliqueraient une mauvaise compréhension des enjeux de ce mouvement, enjeux que les discours idéologiques et économiques ne peuvent pas saisir. Au contraire, nous croyons que ce sont ces deux autres types de discours qui devraient permettre de valoriser l'innovation technique que ce mouvement porte en lui, tant cette innovation est prometteuse.

Nous voulons ici montrer que les logiciels libres créent une différence fertile au sein de l'innovation déjà foisonnante de l'informatique, différence spécifique dont l'analyse constituera le coeur de notre document. Ainsi, nous espérons nous pencher sur les logiciels libres comme phénomène technique, sans réduire le « technique » à la seule mécanique de fonctionnement d'un outil particulier, ni le diluer dans le social en en faisant simplement un nouveau type de comportement face à l'informatique.

2 Définitions

2.1 Les logiciels libres

D'une manière générale, un logiciel est un ensemble d'automatismes programmés pour traiter des données. Le logiciel est lui même un ensemble de données se

5. A noter néanmoins que cette priorité donnée à ce qui touche immédiatement l'utilisateur sera parfois source de confusion: on aura par exemple vite fait d'assimiler logiciel libre et logiciel gratuit.

présentant sous deux formes: le « binaire » ou « exécutable » représente le logiciel tel que le système d'exploitation pourra l'interpréter; le « code source » représente le logiciel tel qu'un être humain pourra l'interpréter, pourvu qu'il connaisse le langage de programmation dans lequel le logiciel a été écrit.

Un logiciel est dit « libre » quand il est protégé par licence autorisant l'accès, la modification et la libre redistribution du code source du logiciel, dans des conditions déterminées. La licence la plus employée est la General Public Licence (GPL), mais des variantes existent: la manière d'informer l'auteur d'un programme des modifications qu'on lui apporte, la liberté ou non d'insérer des morceaux de code propriétaire, le degré de « contagion » de la licence[6], autant de paramètres rendant nécessaires de nouvelles licences. Mais l'accès au code source et la possibilité d'adapter un logiciel à ses besoins propres sont les points les plus importants, toujours respectés quelle que soit la licence qui protège le logiciel libre.

La difficulté de l'analyse des logiciels comme « objets » techniques tient premièrement à leur caractère immatériel. Pour tous les outils matériels dont nous nous servons chaque jour, la simple vue suffit généralement à nous suggérer l'idée de leur utilité. Même pour les machines plus complexes, leur seul présence physique nous permet de les interroger, de fouiller leur mécanismes, et ce sans nécessairement les faire marcher: cette exploration nous indiquera souvent le *comment* et le *pourquoi*. Pour les logiciels, nous devons au contraire - à moins d'être programmeur et de disposer du code source - attendre de les faire fonctionner pour connaître leur utilité. Tous les processus techniques nous sont cachés pour assurer une meilleure ergonomie, un espace de travail dégagé de ce qui n'est pas censé intéresser l'utilisateur final. Donc nous ne savons pas très bien ce qui est vraiment « technique » dans le logiciel, à tel point que les plus ésotériques trouveront que tout y est magique et que les plus cartésiens verront de la technique partout (la technique du pointage à la souris, la technique de la justification des paragraphes, la technique d'ouverture d'une boîte de dialogue, etc.).

Il est tentant de réserver le terme de « technique » à tout ce que le simple utilisateur ne voit pas, tous les éléments qui, en coulisse, permettent le bon déroulement des tâches qu'il aura ordonnées. Bref: tout ce qui est lié de près ou de loin à la programmation et à la gestion interne du système d'exploitation. Mais si « l'outil n'est réellement que dans le geste qui le rend efficace » [LG65, tome II, p.35], alors nous devrons supposer que l'utilisateur intervient activement dans la définition des logiciels comme objets techniques. Nous verrons que cette activité supposée de l'utilisateur est une réalité constitutive des logiciels libres.

6. La GPL est connue pour son caractère très contagieux, puisqu'elle spécifie que chaque version modifiée d'un logiciel sous GPL devra elle-même être sous GPL.

Deuxièmement, quand un usager se sert d'un micro-ordinateur, il n'est jamais en face d'autre chose que de logiciels. Toute communication établie avec le matériel se fait par l'intermédiaire des logiciels, qu'il s'agisse du système d'exploitation ou d'une routine accessoire. Mais si tout est logiciel, il est évident que le degré de technicité est très variable: du noyau d'un système d'exploitation à une suite d'outils de bureautique le pas est déjà énorme, et de cette suite bureautique à des fonctionnalités ponctuelles et légères du système, le pas est encore grand. Notons déjà de plus que le système d'exploitation a un statut particulier, car il conditionne largement la possibilité d'installer tel ou tel logiciel sur un ordinateur. Donc non seulement la technicité des différents logiciels est de degré variable, mais cette technicité n'a de sens qu'au sein d'un ensemble plus large de logiciels, et la hiérarchie qui ordonne ces ensembles n'est pas aussi homogène qu'elle peut le paraître à l'utilisateur final.

Avant de continuer l'analyse des logiciels libres comme objets techniques, nous devons donc tenir compte de ces difficultés, et comprendre en un même concept ce que nous entendons sous le mot de « technique ».

2.2 Objets techniques

Même immatériel, un logiciel n'en est pas moins concret. Il n'est pas un ensemble abstrait de procédures surgissant de nulle part et fonctionnant partout. Il établit un mode de communication déterminé entre l'homme et la machine, proposant un nombre déterminé de possibilités à l'usager, exploitant un nombre déterminé de fonctionnalités du système (logiciel et matériel compris) sur lequel il est installé. Ce que nous nommons ici le caractère concret du logiciel n'est rien d'autre que cet aspect déterminé du rapport qu'il permet d'établir entre l'homme et la machine.

Nous appelerons donc « objet technique » tout instrument qui sert de support concret à des comportements possibles en vue de fins déterminées. Le marteau objective l'action d'enfoncer un clou à coups répétés. Dès qu'il est détourné de cet usage, il n'est plus un objet technique, mais un objet tout court. La voiture n'est objet technique que lorsqu'elle objective notre volonté de déplacement. Tout ce qui entre dans l'usage que nous faisons de la voiture comme outil de déplacement appartient donc de plein droit à sa définition comme objet technique, des performances mécaniques du moteur au confort de l'habitacle (intervenant dans les problèmes de sécurité).

Suivant la même logique, un logiciel se définit comme le support réel et concret d'une communication établie entre l'homme et la machine[7]; des bibliothèques de

7. Il y a bien évidemment des logiciels qui fonctionnent en permanence sans s'occuper de ce que

fonctions que sollicite une application à l'ergonomie de son interface graphique, de son code source aux habitudes que les utilisateurs prendront en le fréquentant, tout cela appartient à la définition concrète du logiciel. Considéré du seul point de vue de l'utilisateur, il est à peine un objet technique, car ce point de vue fait abstraction de l'essentiel des parties opérantes du logiciel, oubliant tous les processus invoqués aux niveaux bas de la machine. Néanmoins, comme un logiciel ne se plie pas à n'importe quel usage, même l'utilisateur lambda est contraint de comprendre comment il fonctionne, d'obéir à certaines techniques d'utilisation, au moins pour les quelques tâches auxquelles il le destine. Considéré du seul point de vue de son développement, le logiciel est hautement technique; mais ce point de vue n'est pas à même de prévoir entièrement quels seront les usages privilégiés par l'utilisateur, il fait en partie abstraction du « geste » accompagnant l'outil (même si le développeur ou le groupe de développeur, en tant que premier utilisateur, fait toujours une réflexion sur l'ergonomie du logiciel).

Cette définition montre assez qu'à isoler un logiciel de ses environnements de développement, de fonctionnement et d'usage, on risque de laisser l'objet technique dans l'abstraction. Ceci est d'autant plus vrai que les phénomènes de *path-dependance* [voir *2.3.3*] sont directement lisibles au coeur du logiciel. L'idée selon laquelle le contenu même de l'innovation est sensible au chemin parcouru pour la développer et la diffuser est très clairement illustrée par un logiciel, dont une comparaison des versions successives donne des indices patents de ce chemin parcouru. Nous verrons que dans le cas du logiciel libre, l'historicité du logiciel prend encore une autre dimension, puisque le logiciel libre rend transparent aux yeux de chaque utilisateur les différentes modifications que celui-ci a connu jusqu'à sa version actuelle.

2.3 La « lisibilité » du logiciel libre

Même si les logiciels ne se livrent pas facilement à l'analyse du théoricien de la technique, il nous semble cependant que leur étude nous permet de retrouver quelques unes des thèses les plus récentes et les plus intéressantes en la matière. Une épistémologie de la technique a donc beaucoup à gagner en se penchant sur le monde des logiciels, puisque les thèses les plus diverses y trouvent une illustration très claire, en plus d'un terrain pratique de validation. Et le monde des logiciels libres renforce encore cette idée, non seulement relativement à la plus grande lisibilité technique des logiciels libres, mais encore par rapport à l'innovation même

fait l'utilisateur, sans qu'une interface de communication décide de ses opérations. Néanmoins ces logiciels, même invisibles pour l'utilisateur final, n'ont de nécessité qu'en ce qu'ils s'intègrent à un ensemble plus grand, ensemble qui va finalement mettre un être humain en face de choix à effectuer.

dont ces logiciels sont porteurs. Les logiciels libres ne sont pas seulement de merveilleuses archives pour qui veut comprendre l'enjeu technique des logiciels en général, mais de l'innovation technique à laquelle ils sont liés peut se dégager une compréhension plus sûre du phénomène technique dans son ensemble.

Quelques exemples vont éclairer ce point de vue.

2.3.1 La « socialité interne » de l'objet technique

Nous empruntons l'expression de « socialité interne » à Louis Quéré, critiquant les approches séparant technique et usage dans l'analyse d'un outil [Qué89]. Considéré comme simple « boîte noire », la structure même d'une invention technique serait indépendante de son processus de diffusion, généralisation de l'usage qui ne serait qu'un événement extérieur à l'objet technique. En allant avec Louis Quéré contre ce point de vue, nous voyons en quoi les logiciels libres illustrent parfaitement cette thèse. Car leur mode de développement implique directement les utilisateurs, et ce sous de multiples formes.

Les utilisateurs experts peuvent s'intégrer au développement d'un projet s'ils sont capables de prouver l'utilité de leur collaboration; d'autres peuvent apporter des corrections significatives sous forme de *patch*; ceux qui fréquentent régulièrement un logiciel et en maîtrisent bien le fonctionnement peuvent participer à la rédaction de manuels d'utilisation; d'autres encore peuvent traduire ces différentes documentations dans les langues qu'ils dominent; et enfin, l'utilisateur lambda peut toujours signaler des bogues, et accélérer ainsi de manière significative le développement de versions plus performantes. Il faut savoir que le travail de déboguage représente une part immense et fastidieuse de l'élaboration d'un logiciel: la distribution de cette tâche à l'échelle du réseau des réseaux, la mise en parallèle des efforts conjugués est l'un des moteurs du développement des logiciels libres. Cette collaboration des concepteurs et des utilisateurs permet non seulement d'accélérer le travail, mais fait aussi en sorte que le logiciel soit la réponse la plus adéquate à certains besoins, ceux-ci se déterminant de plus en plus précisément au fur et à mesure de la collaboration[8]. La *socialité interne* est donc tout à faite patente lorsqu'il s'agit de logiciels libres.

Notons au passage qu'à cette socialité interne des logiciels libres fait écho une socialité externe forte, la communauté des acteurs intervenant dans le développement de ces logiciels formant pour l'instant un tout à peu près cohérent (du moins dans les idées générales qu'ils défendent, et dans la manière de concevoir le rapport

8. De ce point de vue, le témoignage de Eric S Raymond concernant le développement du projet fetchmail est exemplaire.

au logiciel). Cette résonnance entre socialité interne et socialité externe est un fait structurant très fort, un atout pour l'enracinement définitif de ce nouveau mode de développement.

2.3.2 Le rendement croissant d'adoption

Cette expression de Brian Arthur désigne le fait, pour une invention technique, d'être de plus en plus efficace au fur et à mesure de son adoption. Contre l'idée que la technique choisie l'est nécessairement en fonction de son efficacité optimale, il s'agit ici d'insister sur le fait que, meilleure ou non, une technique devient plus efficace au fur et à mesure qu'elle s'impose comme norme [9]. Brian Arthur distingue cinq facteurs pour le calcul du rendement croissant d'adoption, facteur que nous ne ferons ici que mentionner:

1. L'apprentissage par l'usage: *learning by using.*
2. Les économies externes de réseau: plus les utilisateurs forment un réseau, plus il est intéressant pour l'utilisateur de faire partie de ce réseau.
3. L'apprentissage par la pratique productive: *learning by doing*. Dès qu'il y a production de masse, il y a en même temps invention de nouveaux procédés de production.
4. Rendement croissant de l'information: la circulation de l'information favorise une extension de la diffusion, et cette diffusion élargie rend l'efficacité de l'invention toujours plus évaluable.
5. Complémentarités techniques: plus une innovation se diffuse, plus elle forme un système cohérent et devient une norme pour les procédés qui s'y rapportent.

Dans le cas des logiciels libres, l'analyse dans les termes de ces cinq facteurs est essentielle:

1. Plus un logiciel a d'utilisateurs plus il gagne en efficacité, car il répond de mieux en mieux à l'ensemble des besoins exprimés.
2. Prenons l'exemple des formats d'enregistrement pour un traitement de texte: si un logiciel est utilisé par une large communauté, il va alors s'imposer peu à peu, chacun trouvant intérêt à pouvoir lire et écrire sous ce format.

9. La domination de Microsoft sur le marché des systèmes d'exploitation pour micro-ordinateurs illustre parfaitement l'idée qu'une technique arriérée puisse devenir norme d'utilisation: et de fait, une entreprise choisira de préférence une solution informatique à laquelle les utilisateurs sont préparés, car elle lui semblera moins coûteuse en terme de formation, quand bien même ce calcul serait faux.

3. Plus le développement de logiciels libres se généralise, plus il s'organise: aujourd'hui, de nombreux outils permettent la coordination des différents acteurs d'un projet [10].
4. Si la diffusion du système d'exploitation GNU [11]/Linux a connu une telle progression, c'est justement grâce à la rapidité d'information (grâce au réseau Internet) et à la possibilité d'évaluer toujours plus précisément son efficacité.
5. La question des complémentarités techniques ne peut être abordée d'un bloc: car d'un côté la diffusion des logiciels libres est accélérée par leur attention scrupuleuse à l'ouverture des standards et à la compatibilité des formats [12], mais d'un autre côté elle est ralentie par l'impossibilité d'exploiter (à un instant donné) toutes les spécifications du matériel présent sur le marché. Notons néanmoins que les constructeurs sont de plus en plus nombreux à vouloir que leur matériel fonctionne pleinement sous des systèmes libres, et que la diversité des machines utilisées par des développeurs de logiciels libres permet de vite corriger ce manque-à-gagner dans la vitesse de diffusion.

2.3.3 Le phénomène de *path-dependance*

Comme nous l'avons déjà suggéré, le logiciel libre est l'outil idéal pour mettre en valeur le phénomène de *path-dependance* dans la diffusion d'une innovation technique. L'ensemble du mouvement des logiciels libres repose sur des bases historiques plus anciennes que celles des logiciels propriétaires (et que ce qu'on imagine parfois un peu vite quand on parle seulement de *Freeware*). Donc le phénomène de *path-dependance* se mesure non seulement à l'échelle locale d'un logiciel particulier lorsque l'on compare les différents changements qu'il a subi au cours de son histoire, mais aussi à l'échelle globale du développement des Unix, systèmes d'exploitation qui ont permis le développement planétaire des logiciels libres [13]. Trente ans d'histoire du développement d'un système autrefois réservé aux experts et aux outils informatiques de pointe, connaissant aujourd'hui un engouement certain de la part du grand public: voilà un trésor d'informations pour celui qui cherche à montrer l'importance du phénomène de *path-dependance* dans le développement d'une innovation technique.

10. [cf. *infra* 2.3]

11. GNU's Not Unix: cf. *infra* 3.1.3

12. Il est aujourd'hui plus facile de lire un ancien document au format .doc avec un logiciel libre qu'avec le logiciel propriétaire qui a construit et diffusé ce format.

13. L'histoire précise est bien évidemment beaucoup plus complexe, mais l'enjeu général est bien celui-ci.

2.3.4 Les interactions entre choix stratégiques et choix tactiques

Nous prêtons à cette distinction le sens que Michel de Certeau lui donne [dC80]: la stratégie suppose la relative maîtrise d'une situation et la détermination conséquente de la manière dont on veut la faire évoluer. La tactique suppose au contraire l'objectif sans maîtrise préalable de la situation, de sorte qu'il s'agira de faire « avec les moyens du bord ». L'analyse des logiciels libres montre clairement les interactions possibles entre choix stratégiques et choix tactiques.

* Microsoft et/ou Unix

La stratégie de conquête du marché de l'informatique par la société Microsoft passe par la création du système MS-DOS, système implémentant les fonctionnalités minimales des systèmes existants déjà pour des machines lourdes. Du point de vue de la domination du marché cette stratégie a, comme on sait, porté ses fruits. Mais les choix tactiques qui ont présidé à la création de MS-DOS (faire avec les moyens proposés par les micro-ordinateurs bon marchés) ont déterminé un profil de développement du système MS-Windows qui le rend aujourd'hui inutilement complexe. Une simplicité mal définie du système MS-DOS a conduit à une complexité mal gérée du système MS-Windows, complexité voilée de façon hypocrite en termes de « nouvelles fonctionnalités » au fur et à mesure des différentes versions [14].

A l'opposé de cette stratégie commerciale, le développement du système Unix [15] (puis de l'ensemble des logiciels du GNU) a dès le début été orienté par des choix qui révèlent aujourd'hui toute leur pertinence:

- → un système orienté réseau et fonctionnant par couches autonomes: ce système permet une plus grande modularité et laisse le choix ouvert à une grande variété d'utilisations.
- → la portabilité assurée pour différents types d'ordinateurs: le système Unix a même été développé pour répondre à ce besoin précis d'un système permettant de faire communiquer les machines entre elles. On imagine mal aujourd'hui l'« autisme [16] » des machines de l'époque, livrées chacune avec un système différent et un langage propre.

14. Cf. à ce sujet l'article de Roberto Di Cosmo, *Piège dans le cyberespace* [Cos99].

15. Le premier Unix , l'Unix VAX n'est pas un logiciel libre, cette dénommination n'existant pas elle-même. Néanmoins, l'ensemble du projet Unix s'est orienté dans la pratique vers ce choix d'un partage du code source, avant même de donner naissance à la Berkeley Software Distribution (BSD), système libre.

16. L'expression est de Laurent Moineau et Aris Papathéodorou [eAP00].

→ un système écrit en langage C, langage le plus universel pour l'époque et qui continu aujourd'hui de prouver son intérêt [17].

A l'époque de la création du système Unix (projet initié en 1971), ces choix n'ont pas encore toute leur valeur stratégique du point de vue de la diffusion grand public, et s'inscrivent plus dans des tactiques locales visant à faire coopérer les systèmes informatiques. Mais l'accroissement considérable des capacités techniques des micro-ordinateurs au cours des dix dernières années (en termes de rapidité des opérations et de stockage de l'information) est venu donner une valeur stratégique à ces décisions tactiques, en permettant d'installer des systèmes très performants sur des machines ordinaires. La qualité des choix qui ont présidé à la création des Unix et au développement des projets du GNU se ressent donc aujourd'hui, et la pérennité de logiciels tels que GCC [18] en est le témoignage le plus sensible.

* Hurd et/ou Linux:

Autre exemple de confrontation entre choix stratégiques et choix tactiques: la question du développement d'un noyau pour l'ensemble des logiciels du GNU. En tant que clef de voûte du système et gage de cohérence pour l'ensemble du projet GNU, le développement du noyau (projet d'abord nommé « Alix », puis « Hurd ») était primordial, et les membres du projet GNU y travaillaient d'arrache-pied. Mais c'est finalement Linus Torvalds qui propose le premier un noyau pour le système, noyau qu'il appella « Linux » et qu'il développa en collaboration avec des utilisateurs situés aux quatre coins du monde par l'intermédiaire du réseau Internet.

Parmi les facteurs qui expliquent ce « retard » du projet Hurd par rapport à Linux, on peut souligner les suivants:

→ Les développeurs du Hurd sont des passionnés qui veulent exploiter au maximum le potentiel des machines pour lesquelles ils adaptent les logiciels du GNU. Ils tournent donc plus spontanément leurs efforts vers les performances locales que vers la portabilité globale du système. En outre, il était encore impossible de prévoir que l'architecture Intel i386 gagnerait une telle notoriété, ce qui explique un peu mieux le fait que le développement du Hurd n'ait pas pris tout de suite la direction d'une large portabilité sur ce type d'architecture.

17. La charge du développement du premier Unix était donnée à deux hommes par AT&T, Ken Thomson et Dennis Ritchie, celui-ci étant l'inventeur du C.

18. Le compilateur développé par Richard Stallman et l'équipe du projet GNU, pièce maîtresse du système GNU/Linux.

→ Linus Torvalds fait le choix de développer un noyau monolithique: il ferme momentanément les yeux sur ce qui est à l'époque à la pointe de la recherche (les micro-noyaux) pour assurer un fonctionnement optimal sur le plus grand nombre possible de machines.

→ Linus Torvalds invente une stratégie de développement très efficace: sachant stimuler et coordonner les efforts de développeurs éparpillés sur la planète, sachant reconnaître les bonnes contributions des mauvaises, il alimente son projet de corrections très rapides et lui assure une cohérence impeccable.

On voit bien que ce ne sont pas tant les hautes compétences techniques de Linus Torvalds qui lui permettent de mener à bien son projet que la pertinence de ses choix stratégiques, aussi bien en matière d'ingénierie que de gestion des « ressources humaines ». Ce sont ces choix stratégiques qui vont permettre aux logiciels libres de gagner toute la notoriété qu'ils ont aujourd'hui, offrant au grand public l'accès à tous les logiciels libres déjà développés et soutenus par le projet GNU, pour donner ensuite accès à tout logiciel libre quel qu'il soit.

3 Nouveaux modèles proposés par les logiciels libres

A travers le succès fulgurant de Linux, c'est tout un modèle de développement du logiciel qui fait ses preuves, réelle innovation technique dont les logiciels libres sont à la fois le fruit et le symbole.

3.1 Eléments d'histoire

Il y a quelques étapes historiques sans lesquelles on ne peut comprendre la portée générale du logiciel libre dans son ensemble [19].

3.1.1 La naissance d'Unix

En 1971, la AT&T donne à Ken thomson et Dennis Ritchie la responsabilité du projet Unix. L'objectif est de parer à l'hétérogénéité des solutions informatiques proposées par les fabriquants d'ordinateurs. Il faut se rappeler qu'à l'époque les machines sont surtout des super-calculateurs: leur utilisation nécessite une formation précise, chaque ordinateur est livré avec un langage et des logiciels qui lui sont propres. Même les techniciens passionnés d'informatique (les *hackers*)

19. Dans cette partie, nous nous appuyons largement sur l'article très complet de Laurent Moineau et Aris Papathéodorou [eAP00]. S'y reporter pour plus de détails.

n'aident pas à surmonter cet éclatement du monde informatique, car en construisant leurs propres systèmes informatiques, ils cherchent d'abord à les adapter à leurs propres besoins. A partir de 1974, le système Unix s'impose largement en raison de sa portabilité sur différentes architectures matérielles. Pour la première fois se constitue un ensemble homogène de logiciels, homogénéité qui va permettre d'accélérer la collaboration entre développeurs. D'autre part, le système Unix est intrinsèquement orienté vers la collaboration en réseau puisqu'il en implémente d'origine les principaux protocoles de communication, fait qui influera grandement sur son mode ultérieur de développement et de diffusion au sein de la communauté des informaticiens. Dernier aspect qui marque l'avancée considérable du système Unix: il est multiutilisateurs et multitâches. Ces besoins techniques correspondent à l'époque à des tâches très précises, mais ils anticipent sur les besoins actuels d'un système informatique ordinaire.

3.1.2 Le laboratoire d'intelligence collective de Berkeley

L'université de Berkeley adopte le système Unix en 1974, et travaillera à l'élaboration d'une version améliorée du système Unix[20], version distribuée sous le nom de *Berkeley Software Distribution* (BSD).

La faveur qu'obtient ce système dans la communauté des étudiants et des chercheurs de l'époque sera l'occasion de la définition consciente de ce que doit être le logiciel libre. Car les laboratoires d'AT&T font soudain un procès au Computer Systems Research Group de Berkeley pour avoir divulgué un « secret industriel » et passé outre la licence accordée. L'innovation gérée par l'entreprise AT&T vient s'opposer à l'innovation portée par la communauté travaillant au développement d'un système performant.

Le partage des sources d'un programme était ancré dans les habitudes de cette communauté très active de développeurs. En plus de donner un coup d'arrêt au projet BSD (qui ne continua de survivre que dans des distributions de plus en plus éclatées), ce procès souligna *a contrario* l'importance de pouvoir partager les sources, et ouvrit l'ère de l'informatique comme produit massif de consommation.

3.1.3 La Free Software Foundation (FSF)

En 1984, alors que les systèmes propriétaires gagnent de plus en plus le marché de l'informatique, Richard Stallman réagit en créant la Free Software Foundation (FSF), donnant ainsi un souffle neuf à la volonté de partager les sources d'un logiciel. Pour protéger les logiciels d'une éventuelle réappropriation commerciale

20. Notamment deux étudiants, Bill Joy et Chuck Halley, aidés de Ken Thomson, l'un des pères d'Unix.

qui interdirait l'accès et la modification du code source, il met au point le projet GNU [21] et la licence GPL. Les termes de la licence GPL assurent non seulement le libre accès et la libre modification du code source d'un logiciel, mais ils empêchent en outre toute réappropriation de ce code au sein d'un logiciel propriétaire, car tout programme intégrant des morceaux de code source sous GPL devra lui-même être sous GPL.

3.1.4 L'avènement de l'Internet et l'événement « Linux »

Enfin, dernière étape clef de la diffusion du logiciel libre et de son mode de développement, l'apparition au début des années quatre-vingt dix du noyau Linux, permettant de faire marcher les logiciels du projet GNU sur une machine ordinaire (notamment les processeurs bon marché d'Intel), et de mener à la sortie officielle du système d'exploitation GNU/Linux dès 1993. Cette étape ne fut possible qu'au regard du déploiement élargi de l'Internet, réseau lui-même issu de l'ARPAnet des années cinquante et de l'Usenet des années quatre-vingt. C'est en exploitant au maximum les possibilités de collaboration horizontale en matière de développement que Linus Torvalds a pu mener son projet à terme, projet qui rendra visible aux yeux du grand public l'existence et la performance des logiciels libres.

Aujourd'hui, le réseau Internet tout entier n'aurait jamais atteint un tel stade de développement mondial sans les logiciels libres. Le mode de développement des logiciels libres dépend autant de l'Internet que celui-ci dépend des logiciels libres. Et cette dépendance n'est pas seulement lisible dans la prolifération de logiciels clefs tels que les serveurs Apache (équipant aujourd'hui plus de la moitié des serveurs dédiés au web), mais aussi dans la diffusion à grande échelle de langage libres, de solutions de base de données libres, etc [22]. Or, pour qui veut profiter pleinement de ces solutions libres et des logiciels avec lesquels elles fonctionnent, le système GNU/Linux s'est avéré très performant.

Face aux logiciels, le comportement de l'utilisateur tend dès lors à changer. Il conçoit que ce n'est plus le choix de tel ou tel logiciel (propriétaire) qui va tracer les limites « raisonnables » de ses besoins, mais c'est la détermination exacte de ses besoins qui va orienter le choix des logiciels, dans un système de plus en plus modulable, de moins en moins monolithique. Et qui dit univers du logiciel de plus en plus modulable [23], dit en même temps nouvelle attention portée à la com-

21. GNU est un acronyme récursif, jeu de mot pour dire "GNU's Not Unix"

22. La triade *Apache-PHP-MySQL* étant l'une des plus prisées aujourd'hui.

23. Dans un article paru dans le numéro 1 de *Développeur référence*, Jean-Pierre Laisné ose le néologisme de « componet » pour désigner un univers informatique du tout-composant, du matériel au logiciel; il emploie la métaphore du Lego, illustrant ainsi les nouvelles exigences de compatibilité

munication des logiciels entre eux, au respect de standards ouverts, à la souplesse d'adaptation des langages de programmation. L'idée nouvelle d'un système libre où tout serait composant, où la transparence technique serait le gage d'une relation de confiance entre l'utilisateur et l'ensemble des producteurs de services informatiques, où la collaboration active des utilisateurs et des concepteurs ne serait plus dirigée par les seuls concepteurs.

La gratuité et la libre mise à disposition sur le web ont certainement contribué pour une large part à la diffusion très rapide des logiciels libres. Mais cette gratuité ne produirait aucun effet durable sans la haute qualité des performances offertes, ni cette qualité sans l'émergence d'un nouveau comportement de l'utilisateur, conscient de la souplesse des outils qu'il a en mains, conscient que tout système d'informatique a besoin de base solides pour mieux communiquer avec l'ensemble des autres systèmes. L'implémentation des protocoles les plus utilisés par l'Internet au coeur même du noyau Linux (notamment le protocole de communication TCP/IP) témoigne au sein de l'objet technique lui-même de cette double attention à la solidité et à l'ouverture, et il n'est donc pas étonnant que le système GNU/Linux soit aujourd'hui l'emblème privilégié de l'univers des logiciels libres.

3.2 « La cathédrale et le bazar »

3.2.1 Nouveaux cadres pratiques et nouveaux modèles théoriques

Cette expression, titre d'un texte de Eric S. Raymond [Ray99] dans lequel la communauté des *hackers* s'est immédiatement reconnue, met en opposition de façon imagée deux modèles de développement du logiciel[24].

D'un côté le modèle traditionnel de la « cathédrale »: l'écriture du logiciel y est une procédure fortement structurée avec des étapes bien séparées et bien hiérarchisées. De la détermination des besoins du produit (ou DSL: dossier de spécification du logiciel) aux essais *in situ*, le logiciel se construit d'une manière globalement linéaire.

De l'autre côté, le modèle de type « bazar »: même si un groupe possède la direction d'un projet, il est à l'écoute des propositions surgissant à l'horizontale (c'est à dire de l'ensemble potentiel des utilisateurs), et se sert même des utilisateurs volontaires en tant que co-développeurs. Ce mode de développement n'est plus pyramidal et linéaire, le logiciel ne se développe pas par des mécanismes disposés *en série*, mais par des voies travaillant *en parallèle*. Cette disposition en parallèle des efforts des co-développeurs permet d'être plus efficace dans l'implé-

et créativité potentielle.

24. Une version française de ce texte (traduction de Sébastien Blondeel) est disponible à l'adresse http://www.linux-france.org/article/these/cathedrale-bazar/cathedrale-bazar.html.

mentation des diverses fonctionnalités du logiciel, d'accélérer son débogage (phase qui est d'ordinaire grande consommatrice de temps), et de laisser libre l'expression de nouveaux besoins, la suggestion d'idées nouvelles... pourvu seulement que le noyau dur du projet sache distinguer les contributions utiles des agitations vaines.

Eric S. Raymond, *hacker* expérimenté et ouvert aux perspectives nouvelles, se dit lui-même avoir été surpris du succès que le mode de développement de Linux emporta. Alors même qu'il travaille déjà avec la FSF et collabore au projet GNU, la surprise n'est pas moindre lorsqu'il voit pour la première fois Linux « dans [son] radar ». C'est dire en quoi ce nouveau modèle de développement est novateur. Parmi les leçons que Eric S. Raymond tire de son expérience de développement coopératif (donnant naissance à *fetchmail*, logiciel de récupération de mails sur différents types de messageries), nous soulignerons celles qui illustrent au plus près l'innovation technique réellement portée par les logiciels libres.

Pour aborder les enjeux techniques de ces conseils, nous nous appuierons sur une analyse de Patrice Flichy. Dans l'*innovation technique* [Fli95], celui-ci parle de « cadre de référence » pour désigner l'ensemble socio-technique des éléments donnant sens à un outil. Hors de ce cadre de référence, ni le fonctionnement de l'outil ni son usage ne sont compréhensibles; pour comprendre l'utilité d'un vieil outil trouvé dans une brocante, il faudra - en technicien ou en historien - déterminer à quelles fins cet instrument est destiné pour comprendre les moyens qu'il met en oeuvre. L'auteur décompose ce cadre de référence en cadre de fonctionnement et cadre d'usage: le cadre de fonctionnement concerne le technicien et le cadre d'usage concerne l'usager. Il va de soi que ces deux cadres ne sont pas réellement séparés, et que le technicien intervient en se représentant un usager, tandis que l'usager intervient parfois en tant que technicien. Dans cette analyse du cadre de référence, l'auteur fait une analogie intéressante, disant que le cadre de fonctionnement est au signifiant ce que le cadre de référence est au signifié: la conséquence immédiate d'une telle analogie est de souligner l'arbitraire qui lie le cadre de fonctionnement du cadre d'usage, thèse que nous devrons nuancer lorsqu'il s'agit de logiciel libre.

Pour l'objet spécifique qui nous occupe, introduisons une troisième dimension à cette analyse: le cadre de développement. Car le cadre de fonctionnement d'un logiciel est double: du point de vue de l'usager, il s'agit des interfaces qui lui permettent de communiquer avec le programme et qui structureront le cadre d'usage; du point de vue du concepteur, il s'agit de ce qu'il y a encore derrière les interfaces et qui communique avec la machine. Ce qui va donner sens au cadre de fonctionnement d'un logiciel du point de vue de sa conception, appelons-le « cadre de développement ». On obtient donc ainsi un parcours qui semble linéaire, du cadre de développement au cadre d'usage, en passant par le cadre de fonctionnement. La

documentation d'un logiciel détaillera, selon son extension, ce cadre de fonctionnement du logiciel, tantôt en insistant sur certains points du cadre de développement (notamment pour expliquer le *pourquoi* et le *comment* de nouvelles fonctionalités), tantôt en insistant sur certains points du cadre d'usage (en donnant des exemples d'utilisations les plus courantes, de raccourcis-claviers les plus employés, etc.); il s'agira néanmoins chaque fois de montrer que les trois cadres forment un tout homogène.

Dans ces circonstances, est-il vrai que le rapport du cadre de fonctionnement au cadre d'usage est arbitraire ? Si l'on rapporte les parties les plus « basses » du fonctionnement d'un logiciel (celles qui sont au plus près de la gestion du matériel) à l'interface ultime de communication avec l'utilisateur, alors on peut certes parler d'un lien quelque peu arbitraire entre l'usage courant du programme et son fonctionnement. Non pas que ce lien ne soit pas strictement déterminé, mais les choix rencontrés dans l'écriture du programme peuvent être ouverts: le choix du langage dans lequel il est écrit, la manière dont il va gérer la mémoire qu'il occupe dans l'ordinateur, les bibliothèques graphiques qu'il va exploiter, etc.

Mais si, au lieu de prendre les parties les plus distantes du cadre de fonctionnement et du cadre d'usage, on en prend des parties plus rapprochées, alors le lien entre les deux cadres devient de moins en moins arbitraire; le choix du langage de programmation correspondra à des besoins spécifiques d'utilisation[25], la manière de gérer la mémoire dépendra de l'utilisation courante qu'on fera du logiciel, etc. On peut finalement porter plus loin l'analogie de Patrice Flichy en disant que, de même qu'un poème désamorce l'arbitraire du signe en donnant sens au rapport signifiant/signifié, un projet qui s'élabore élimine en progressant toute contingence dans le rapport entre les cadres de développement, de fonctionnement et d'usage. Et ceci est d'autant plus vrai pour les logiciels libres, puisque leur mode de développement brise l'apparente linéarité entre cadre de développement, cadre de fonctionnement et cadre d'usage, comme le montrent les conseils de Eric S. Raymond que nous pouvons maintenant aborder.

3.2.2 Cadre de fonctionnement

La collaboration étroite des équipes de développement et des utilisateurs modifie la façon de concevoir le fonctionnement du logiciel, met les problèmes liés à son fonctionnement au coeur des choix de développement.

→ « [Leçon] 1: Tout bon logiciel commence par gratter un développeur là où ça le démange »

25. Dans la pratique, ce choix dépend aussi des connaissances du (ou des) programmeur(s).

Le développeur étant lui-même utilisateur de logiciels, il peut repérer un besoin auquel le logiciel ne répond pas et imaginer déjà la manière de répondre à ce nouveau besoin, pourvu qu'il puisse connaître la manière dont le programme est écrit. Mais le logiciel a d'autant plus de chance d'être utilement écrit qu'il répondra à une « démangeaison » plus forte.

→ « [Leçon] 2: Les bons programmeurs savent quoi écrire. Les grands programmeurs savoir quoi réécrire (et réutiliser) »

Ce principe insiste sur l'idée de « paresse constructive »: il est inutile de refaire ce qui a déjà été fait, le mieux étant de défaire un peu pour refaire autrement. En interdisant l'accès au code source, les logiciels propriétaires condamnent des développeurs à refaire ce qui a déjà été fait ailleurs. En autorisant l'accès à leur code source, les logiciels libres évitent les gaspillages d'énergie, stimulent une saine émulation entre développeurs, empêchent les efforts de se déployer à l'encontre de l'innovation.

→ « [Leçon] 3: On ne comprend souvent vraiment bien un problème qu'après avoir implanté une première solution »[26]

Ce principe insiste sur ce qu'on peut gagner à proposer aux utilisateurs une version même prématurée d'un logiciel. D'une manière générale, le problème apparaîtra toujours plus clairement quand différents points de vue entreront en résonance pour s'exprimer sur sa solution en cours.

3.2.3 Cadre d'usage

→ « [Leçon] 7: Distribuez tôt. Mettez à jour souvent. Et soyez à l'écoute de vos clients. »

D'abord donné aux développeurs, ce conseil change aussi les rapports de l'utilisateur à ses logiciels dans la mesure où il devient responsable devant la détermination de ses propres besoins. Connaissant les fonctions implémentées par un logiciel dans sa version vX.X, ainsi que celles qui sont prévues pour la vX.X+1, il ne tiendra qu'à lui de télécharger la nouvelle version si elle correspond mieux à ses attentes. Et le dynamisme d'un projet souvent mis à jour lui assure la prise en

26. Nous donnons ici l'explication de la règle, dont l'énoncé exact est le suivant: « Prévoyez d'en jeter un, car de toutes manières, vous le ferez. » (Fred Brooks, *The mythical man-month,* chapitre 11)

compte rapide de ses attentes, qu'il les aient lui-même exprimées ou qu'elles aient été exprimées par quelqu'un ayant les mêmes.

→ « [Leçon] 14: Tout outil doit être utile par rapport aux utilisations qu'il a été prévu d'en faire. Mais on reconnaît un outil vraiment excellent au fait qu'il se prête à des usages totalement insoupçonnés. »

Un logiciel est rarement une suite linéaire d'opérations s'enchaînant après une injonction unique de l'utilisateur: c'est le plus souvent un outil ouvert et malléable, configurable selon nos besoins propres, programmable pour des tâches spécifiques. Dès lors, toutes ses potentialités ne sont pas nécessairement explorées par le noyau dur de ses concepteurs. Plus les utilisateurs sont considérés comme co-développeurs du logiciel, plus celui-ci développe la capacité à répondre de près ou de loin à un grand nombre d'attentes, voire à anticiper sur des besoins. Dire que le logiciel a une socialité interne, c'est donc aussi prendre en compte l'invention singulière à laquelle un utilisateur lambda pourra soumettre un logiciel.

→ « [Leçon] 15: Quand vous écrivez un logiciel jouant le rôle d'une passerelle quelconque, prenez soin de perturber le moins possible le flot de données - et ne perdez *jamais* d'éléments d'information, à moins que la machine destinataire vous y oblige! »

Eric S. Raymond explique qu'il a écrit son logiciel de retrait du courrier électronique en laissant son code compatible avec le format MIME en 8 bits, format standard pour l'échange de document par mail. Lorsque ses co-développeurs réclament que Fetchmail puisse utiliser ce format, il peut très facilement leur donner satisfaction en raison de la compatibilité de son code initial.

Ce conseil s'adresse donc immédiatement au programmeur mais il est symptomatique du nouveau cadre d'usage qui s'installe avec les logiciels libres. En s'adressant à un horizon très large d'utilisateurs, le logiciel libre est souvent confronté à des problème de standards et de formats. Pour répondre le plus adéquatement possible aux besoins exprimés *et à ceux qui sont susceptibles de s'exprimer par la suite*, l'utilisation des standards les plus ouverts et des formats les moins « propriétaires » s'impose naturellement au développeur. Les utilisateurs ont tout à y gagner, puisque cette volonté de respecter les standards les plus ouverts leur assure une pérennité totale de leurs documents, et ne les met jamais en situation d'otages face à un éditeur de logiciels propriétaires qui imposera ses propres standards.

3.2.4 Cadre de développement

→ « [Leçon] 6: Traiter vos utilisateurs en tant que co-développeurs est le chemin le moins semé d'embûches vers une amélioration rapide du code et un

déboguage efficace. »

Eric S. Raymond affronte ici le cliché du développeur solitaire, mettant tout son acharnement à venir seul à bout d'un projet qu'il a initié. Certes l'écriture du code dans le détail est effectivement une activité solitaire, mais celui qui dirige un projet a tout intérêt à considérer les portions de code qui lui sont proposées avec attention, de sorte qu'il pourra y repérer une nouvelle solution ou un nouveau problème, l'un et l'autre faisant de toutes façons avancer le projet. Cette promotion de l'utilisateur compétent et motivé vers le statut de « co-développeur » ne confisque aucun savoir-faire à l'auteur initial du projet, multiplie les suggestions d'utilisations nouvelles, et permet d'effectuer des procédures de vérification dès le stade embryonnaire du projet (sachant que la nécessité du débogage croît de manière exponentielle en fonction de la taille du projet). De plus, cette collaboration active d'utilisateurs considérés comme co-développeurs s'annonce décisive lorsqu'il s'agit de s'atteler aux tâches entourant le logiciel: rédaction et traduction de la documentation, maintenance de son site officiel, réponses aux questions couramment posées, etc.

Dans cette perspective, le bon programmeur n'est pas celui qui pourra enchaîner des lignes de code sans faillir, mais celui qui saura mettre en valeur tous ses collaborateurs, lesquels ne sont plus considérés comme extérieurs, mais définissent de l'intérieur l'identité multiforme du logiciel. D'où cette dernière leçon que nous citons:

→ « [Leçon] 11: Il est presque aussi important de savoir reconnaître les bonnes idées de vos utilisateurs que d'avoir de bonnes idées vous-même. C'est même préférable, parfois. »

L'auteur d'un projet s'est certes « gratté » là où ça le démangeait, mais il a aussi pu, parfois sans le soupçonner, toucher certains points sensibles de ses utilisateurs. Aussi doit-il écouter leur réaction comme il a écouté la sienne, et savoir reconnaître l'occasion d'un changement utile. Par exemple, sous la pression de ses utilisateurs, Eric S. Raymond choisit d'intègrer le multidrop[27] à Fetchmail, et il y trouve une occasion de déboguer plus rapidement la partie *single-drop* de son codage. Mais par ailleurs, il refuse de crypter les mots de passe contenus dans le fichier de configuration de fetchmail, car cette fonctionnalité n'aurait augmenté la sécurité que de manière illusoire, quand bien même cette illusion aurait réconforté certains de ses utilisateurs. Ainsi, en n'ayant plus seulement affaire à des opérations mais aussi à des volontés, le développeur prend une plus grande mesure de l'importance des choix qu'il opère, du point de vue du cadre de fonctionnement comme de celui du cadre d'usage.

27. Procédé permettant de retirer le courrier à une adresse unique et de le redistribuer ensuite dans différentes boîtes, comme cela a lieu dans la distribution du courrier dans les immeubles.

La socialité interne du logiciel libre n'est donc plus une abstraction venant du fait qu'un développeur se *représente* des usages seulement possibles, mais est la forme concrète de l'objet technique ainsi élaboré. Et si nous avons pris ces trois cadres dans un ordre qui brise la linéarité habituelle suivant laquelle nous les disposons, c'est justement pour montrer à quel point ils s'interpénètrent dans le développement et la diffusion du logiciel libre. Il ne s'agit pas seulement de rétroaction du cadre de fonctionnement sur le cadre de développement, puis du cadre d'usage sur le cadre de fonctionnement, car les trois cadre se développent en parallèle. Il peut très bien arriver qu'un besoin nouveau suscite l'intégration d'une nouvelle fonction dans un langage de programmation - nous pensons notamment à la manière dont s'est développé le langage Perl -, puis que l'existence de cette fonction se révèle très utile pour la réécriture d'anciens programmes, programmes qui voient alors se transformer leur cadre de fonctionnement.

Bref: l'ouverture des codes sources est l'occasion de véritables synergies de développement, synergies qui stimulent sainement et orientent utilement le désir d'innover.

3.3 Nouveaux outils de développement coopératif

Pour mener à bien un projet qui implique de nombreuses personnes et de nombreuses manières de contribuer, une gamme d'outils de plus en plus différenciés et spécialisés s'est créée. Citons trois des outils les plus représentatifs du genre:

3.3.1 *CVS*: Concurrent Version System

Le logiciel *CVS* permet de gérer le développement collectif d'un logiciel en archivant et contrôlant les différentes versions, qu'il stocke dans un espace centralisé (le *repository*). Afin d'économiser de la mémoire, le logiciel ne stocke que les différences entre les versions, ce qui permet en plus de donner une meilleure lisibilité à l'évolution du projet. Pour faciliter la collaboration, le système de contrôle compare les travaux isolés pour les fusionner et en marquer les différences, de sorte que plusieurs développeurs peuvent travailler à développer une même version sans que ce travail ne génère de conflit entre les versions.

Le système *CVS* est devenu une référence quasi incontournable en matière de gestion du codéveloppement. Il donne toute sa visibilité à ce que nous avons appelé avec Louis Quéré la « socialité interne » de l'objet technique, car la présence de l'utilisateur n'y est pas une simple représentation abstraite, mais un fait lisible dans l'évolution des versions du logiciel. Une fois que ce système de contrôle des versions s'ouvre à l'utilisateur quelconque (par l'intermédiaire de sites web ou de

logiciels avec interface graphique tels que *CVSup*), il montre bien comment celui-ci est toujours potentiellement au coeur d'un projet, car l'utilisateur peut à tout moment récupérer les sources d'une version antérieure du logiciel pour en développer une nouvelle s'éloignant quelque peu de la version « officielle ». Pour peu que sa version réponde à un plus grand nombre de besoins, elle tendra peut-être par la suite à devenir la nouvelle version « officielle ». Dans la pratique, ce genre d'événement est peu probable en raison de ce rapport étroit existant entre utilisateurs et concepteurs - étroitesse qui interdit les égarements et les mauvaises surprises - mais cette potentialité est emblématique de la dynamique de développement des logiciels libres, de leur consistance et résistance comme objets techniques.

3.3.2 Les sites Internet

Le rôle des sites Internet gravitant autour d'un logiciel libre sont multiples. Le plus généralement, ils servent:

- → à mettre à disposition de tous les sources du logiciel sous divers formats et pour différents systèmes d'exploitation;
- → à présenter l'historique des dernières versions;
- → à présenter les nouvelles fonctionnalités de la version la plus récente;
- → à faire le point sur ce qu'il reste à faire;
- → à proposer un emplacement centralisant le signalement des bogues et les nouvelles propositions.

Les sites www.mozilla.org et www.gnome.org, rendant compte de deux immenses projets, illustrent parfaitement cet esprit. Le site ne se contentera jamais d'être une vitrine promotionnelle, un lieu d'information, mais il aura aussi un rôle de coordination et de sollicitation des efforts de développement. De plus, les sites tendent à reproduire eux-même le principe de CVS, avec l'apparition de *WebCVS* et autres lieux de collaboration. Signalons enfin la coordination à grande échelle de nombreux projets libres dans des sites tels que *sourceforge.net*, lesquels deviennent peu à peu des références en matière de dynamisme de l'*open source*.

3.3.3 Les logiciels pour le signalement efficace des bogues

Nous avons déjà souligné qu'un des grands avantages tirés du développement coopératif était la localisation et signalisation extrêmement rapide des bogues. A l'échelle de projets modestes, ce signalement se fait de manière directe, et on peut se contenter de demander à l'utilisateur de vérifier qu'il ne va pas signaler un bogue déjà repéré. Mais à l'échelle de projets plus importants, la gestion des signalements

se fait au travers de logiciels. Par exemple, le logiciel *Bug-Buddy*, développé par l'équipe du projet *Gnome*, permet à l'utilisateur lambda de signaler utilement un bogue, car il détecte automatiquement les conditions générales dans lesquels le bogue est apparu (architecture du matériel, type de système d'exploitation et version du noyau utilisé, etc.) et demande à l'utilisateur d'en préciser les conditions exactes. Ensuite s'opérera un tri, selon le logiciel bogué, selon le type de bogue, selon sa gravité, etc.

La diversité des voies par lesquelles sera organisé le travail coopératif de développement montre bien le dynamisme de telles synergies, et la résolution rapide des nouveaux problèmes qu'elles rencontrent. Et l'on peut mesurer l'innovation technique portée par les logiciels libres au nombre de solutions qui sont aujourd'hui proposées pour permettre une coordination optimale des efforts de développement.

Comme tout nouvel objet technique, le logiciel libre apporte donc avec lui ses contraintes de déploiement et exploite de manière privilégiée un certain mode de diffusion. Les contraintes se lisent dans les efforts faits pour organiser le co-développement; la diffusion passe en grande partie par le réseau Internet, celui-ci jouant le rôle primordial de passerelle interactive.

4 Conclusion

Une fois admis que la définition de ce qu'il y a de technique dans un logiciel n'est réservée ni au technicien ni au sociologue, mais que le caractère concret de l'objet technique oblige à prendre en compte dans une même analyse son développement, sa diffusion, son fonctionnement et son utilisation, résumons en quatre points l'innovation portée par les logiciels libres en tant qu'« objets techniques »:

1. Ils sont les fruits d'un nouveau mode de développement, caractérisé en détail par Eric S. Raymond dans son exploration du modèle « bazar », qu'il oppose au modèle « cathédrale ». Protégeant le libre accès et la libre modification du code source (sous certaines conditions qui assurent la « contagion » de cette liberté), ce modèle peut profiter de nouvelles occasions de coopération, mais doit aussi faire face à de nouvelles contraintes d'organisation.
2. Les projets de logiciels libres progressent dans une interaction rapide entre le cadre de développement, le cadre de fonctionnement et le cadre d'utilisation, de sorte qu'ils répondent efficacement à des besoins très précis. Le fait que certains d'entre eux soient déjà devenus des références mondiales - notamment le serveur *Apache*, équipant aujourd'hui plus de la moitié des serveurs du réseau Internet - ne vient pas seulement de leur fréquente gratuité, mais de ce mode de développement lui-même.

3. En proposant un monde où nous ne payons plus le logiciel lui-même mais le service qui en optimise l'emploi, il permet que s'établisse une concurrence saine dans le monde de l'édition des logiciels, concurrence qui ne profiterait plus de la prise en otage des utilisateurs par certains standards de fait.
4. Le monde des logiciels libres est - idéologiquement autant que pragmatiquement - attaché à l'exploitation de standards ouverts, à la portabilité et à la lisibilité des différents formats de document, à la collaboration efficace des logiciels entre eux. Il assure donc une pérennité des données informatiques (pérennité qui est destinée à devenir un problème de plus en plus préoccupant pour les entreprises désireuses d'archiver leurs données) et anticipe sur une vision novatrice de l'informatique, vision dans laquelle les logiciels travaillent de plus en plus ensemble.
5. En faisant de tout utilisateur un développeur éventuel, les logiciels libres le responsabilisent devant son ordinateur, l'incitent à déterminer avec exactitude ses besoins réels. Cette détermination des besoins est d'autant plus aisée que les logiciels libres sont largement documentés, que les clefs sont données à l'utilisateur pour qu'il apprenne seul. De plus, les systèmes d'exploitation libres permettent de faire tourner de manière très économique, sans surcoût de mémoire, une installation destinée à des tâches très précises.

Tout au long de ce document, nous avons dû sacrifier certains sujets importants pour nous concentrer sur notre idée. Parmi ces sujets, soulignons-en très rapidement trois. D'une part, la question des **licences libres**[28]: la plus connue est la *General Public licence* (GPL), mais il existe de nombreuses déclinaisons différentes de cette idée d'une licence protégéant la liberté d'un logiciel. Ce type de licence s'est même vu adapté pour d'autres objets, tels que des oeuvres d'art ou même de simples documents. D'autre part, nous avons mis de côté la **gratuité** de la plupart des logiciels libres, et ce pour plusieurs raisons. D'abord, cette gratuité n'est pas systématique et n'entre pas dans la définition exacte du logiciel libre, même si, couplée à la mise à disposition sur le web, elle joue un rôle prépondérant dans la rapidité de sa diffusion, rapidité qui peut elle-même devenir une contrainte technique à prendre en compte. Ensuite (et surtout) la gratuité entraîne le plus souvent vers des débats idéologiques, débats qui structurent fortement la communauté du « libre », mais qui n'entraient pas directement dans le cadre de notre réflexion. Troisième débat absent de ce document, la question des **brevets logiciels**. Ces problèmes de législation déterminent largement le cadre de l'innovation dans le domaine du logiciel: certains usages abusifs des brevets empêchent

28. Pour un panorama réflexif de l'ensemble des problèmes liés à l'innovation, je renvoie à l'ouvrage d'Olivier Blondeau et Florent Latrive, *Libres enfants du savoir numérique [eFL00]*.

leur développement, en interdisant de plus une concurrence saine, orientée vers la performance.

Aujourd'hui, les logiciels libres remettent en cause le verrouillage technologique qui a failli s'installer avec la mainmise d'intérêts seulement économiques et financiers sur les enjeux de la diffusion vers le grand public de l'innovation informatique. Et leur force réside certainement dans le fait que leur modèle de développement empêche qu'un nouveau verrouillage, quel qu'il soit, puisse ralentir le dynamisme de ce secteur de l'innovation.

Nous espérons avoir montré que les logiciels libres étaient un terrain privilégié pour la lecture et la mise à l'épreuve de récentes thèses de sociologie et d'épistémologie de la technique. Il arrive bien sûr que nous tombions dans cette illusion que dénonçait déjà Henri Bergson, illusion rétrospective par laquelle nous réduisons le passé à ce que nous connaissons grossièrement du présent, prenant pour une nécessité réelle d'engendrement ce qui n'est que nécessité logique de reconstruction. Mais il arrive aussi que l'historien se serve de la richesse du présent pour mieux comprendre les processus fondamentaux qui ont présidé à l'avénement du passé. Ainsi, nous servant de la complexité socio-technique mise à jour par l'analyse du déploiement des logiciels libres, peut-être y trouverons-nous matière à mieux comprendre les phénomènes technologiques dans leur généralité.

Remerciements

Merci à Thierry Stoehr et Dams pour les corrections et conseils.

Références

[Cos99] Roberto Di Cosmo. *Piège dans le cyberespace*. Document web, http://alexandrie.bettybook.com/oeuvres/oeuvre045/resum.htm, 1999.

[dC80] Michel de Certeau. *L'invention au quotidien*. UGE-10/18, 1980.

[eAP00] Laurent Moineau et Aris Papathéodorou. Coopération et production immatérielle dans le logiciel libre. *Multitude*, Numéro 1, Février 2000.

[eFL00] Olivier Blondeau et Florent Latrive. *Libres enfants du savoir numérique*. Editions de l'Eclat, Paris, 2000.

[Fli95] Patrice Flichy. *L'innovation technique*. Editions de la découverte, Paris, 1995.

[LG65] André Leroi-Gourhan. *Le geste et la parole*. Albin Michel, Paris, 1965.

[Qué89] Louis Quéré. *Les boîtes noires de Bruno Latour ou le lien social dans la machine*. Number 36. Réseaux, Juin 1989.

[Ray99] Eric S. Raymond. *The cathedral & the bazaar*. O'Reilly, 1999.

[Ruy77] Raymond Ruyer. *La Gnose de Princeton*. Editions Fayard, Paris, 1977.

[Whi69] Alfred North Whitehead. *La fonction de la raison*. Editions Payot, 1969.

Table des matières