

# Le chaudron magique (*The Magic Cauldron*)

---

Auteur : *Eric S. Raymond* <<http://www.ccil.org/esr/>> ([esr@thyrsus.com](mailto:esr@thyrsus.com))

Traducteurs : *Sébastien Blondeel* <<http://www.linux-france.org/these/article/>> & *Emmanuel Fleury* <<mailto:fleury@lsv.ens-cachan.fr>> & *Denis Vauldenaire* <<http://www.multimania.com/dvaldenaire/>>  
juin 1999 ; traduit en juillet

Cet article analyse le fonctionnement du substrat économique du phénomène du code source ouvert<sup>1</sup>. Nous explorons d'abord quelques mythes archaïques concernant le financement du développement d'un programme, et la structure du prix d'un logiciel. Nous présentons ensuite l'analyse — suivant la théorie des jeux — de la stabilité de la coopération dans le monde du source ouvert. Nous présentons neuf modèles valables de financement d'un projet à source ouvert ; deux à buts non lucratifs, six montrant une volonté de dégager des bénéfices. Nous continuons en développant une théorie qualitative dictant quand il est rationnel, économiquement parlant, d'être à source fermé. Nous examinons ensuite les mécanismes récents que le marché est en train d'inventer pour financer des développements à source ouvert rentables, en particulier la réinvention du système du mécénat, et du marché à la tâche. Nous concluons avec des ébauches de prédictions pour l'avenir.

## Table des matières

<b>1</b>	<b>Indiscernable de la magie</b>	<b>2</b>
<b>2</b>	<b>Au-delà des dons des <i>geeks</i></b>	<b>2</b>
<b>3</b>	<b>L'illusion des biens manufacturés</b>	<b>3</b>
<b>4</b>	<b>Le mythe « il faut que l'Information soit libre »</b>	<b>6</b>
<b>5</b>	<b>Les <i>communs</i> inversés</b>	<b>6</b>
<b>6</b>	<b>Les bonnes raisons pour travailler en sources fermés</b>	<b>8</b>
<b>7</b>	<b>Les modèles financés par la valeur d'utilisation</b>	<b>9</b>
7.1	Le cas <i>Apache</i> : partage des coûts . . . . .	9
7.2	Le cas <i>Cisco</i> : étaler les risques . . . . .	10
<b>8</b>	<b>En quoi le prix d'acquisition pose problème</b>	<b>10</b>
<b>9</b>	<b>Les modèles exploitant une valeur d'acquisition indirecte</b>	<b>12</b>
9.1	Vendre à perte pour se positionner sur un marché . . . . .	12
9.2	Gel des gadgets . . . . .	12
9.3	Donner la recette, ouvrir un restaurant . . . . .	13
9.4	Accessoires . . . . .	14
9.5	Libérer l'avenir, vendre le présent . . . . .	14

1. N.D.T. : *Open Source*.

9.6 Libérer le logiciel, vendre la marque . . . . .	14
9.7 Libérer le logiciel, vendre le contenu . . . . .	14
<b>10 Quand faut-il ouvrir, quand faut-il fermer?</b>	<b>15</b>
10.1 Quels sont les bénéfices? . . . . .	15
10.2 Comment interagissent-ils? . . . . .	16
10.3 <i>Doom</i> : une étude de cas . . . . .	17
10.4 Savoir quand lâcher emprise . . . . .	18
<b>11 L'écologie des marchés des logiciels à sources ouverts</b>	<b>19</b>
<b>12 Composer avec la réussite</b>	<b>19</b>
<b>13 Recherche et développement ouverts et réinvention du mécénat</b>	<b>21</b>
<b>14 Se rendre d'un point à un autre</b>	<b>22</b>
<b>15 Conclusion : vivre après la révolution</b>	<b>23</b>
<b>16 Bibliographie et remerciements</b>	<b>24</b>
<b>17 Annexe : Pourquoi fermer ses pilotes fait perdre de l'argent à un fabricant</b>	<b>25</b>
<b>18 Historique des versions</b>	<b>26</b>

## 1 Indiscernable de la magie

Dans le mythe gallois, la déesse Ceridwen possédait un grand chaudron qui produisait magiquement une riche nourriture — quand elle le lui ordonnait par un sort connu par elle seule. Dans les sciences modernes, Buckminster Fuller nous a donné le concept de « l'éphéméralisation », une technique qui devient d'autant plus efficace et moins chère que les ressources physiques investies dans les premiers efforts de développement sont remplacées par de plus en plus de contenu informatif. Arthur C. Clarke a relié ces deux phénomènes en observant que « toute technique suffisamment avancée est indiscernable de la magie ».

Pour beaucoup de personnes, les succès de la communauté du source ouvert ont l'apparence d'une improbable magie. Des logiciels de haute qualité se matérialisant à partir de « rien », c'est bien tant que ça dure, mais cela semble peut praticable dans le monde de la concurrence, où on ne dispose que de ressources limitées. Quel est le piège? Le chaudron de Ceridwen n'est-il qu'un tour de passe-passe? Dans la négative, comment « l'éphéméralisation » fonctionne-t-elle dans ce contexte — quelle est la formule magique de la déesse?

## 2 Au-delà des dons des *geeks*

L'expérience du source ouvert a sans doute ébranlé en profondeur les convictions des gens qui ont appris le génie logiciel hors cette culture. L'essai « La cathédrale et le bazar » 16 () décrivait les moyens par lesquels le développement logiciel coopératif et décentralisé invalidait, dans la pratique, la loi de Brooks, menant à des niveaux de fiabilité et de qualité sans précédent pour des projets individuels. L'essai « À la conquête

de la noosphère » 16 () examinait, lui, le contexte social dans lequel évolue le style de développement en « bazar », en arguant qu'on le comprenait mieux, non pas en termes d'économie d'échange, mais par ce que les anthropologues appellent « culture du don », où la lutte pour un meilleur statut s'effectue au travers de dons. Dans cet article, nous commençons par réduire en miettes les mythes les plus communs concernant l'économie de la production de logiciels, puis nous prolongeons l'analyse de 16 () et 16 () dans les modèles de l'économie, de la théorie des jeux et des affaires, en développant de nouveaux outils conceptuels, dont nous avons besoin pour comprendre la façon dont la culture du don des développeurs à source ouvert peut être rentable dans une économie d'échange.

Pour poursuivre cette voie sans être distrait, nous devons abandonner (ou du moins accepter d'ignorer temporairement) le niveau d'explication de la « culture du don ». 16 () a suggéré que cette culture naissait dans des situations où les biens vitaux étaient assez abondants pour rendre le principe de l'échange sans intérêt. Alors que cela apparaît comme une explication suffisante du point de vue *psychologique*, ces explications connaissent en revanche des lacunes pour comprendre le contexte *économique* dans lequel les développeurs à source ouvert opèrent dans la réalité. Pour beaucoup, le jeu de l'échange a perdu de son intérêt, mais pas sa force de contrainte. Leur comportement doit avoir un sens économique suffisant comme on l'entend dans une économie où les biens sont rares pour pouvoir se maintenir également dans la zone où le surplus des biens fait que l'on participe à la culture du don.

C'est pourquoi nous considérerons (à *partir* du point de vue d'une économie de biens rares) les modes de coopération et d'échange qui rendent le développement à source ouvert possible. Ce faisant, nous répondrons à la question pragmatique « comment gagner de l'argent avec tout cela? » en détail et avec des exemples. Mais d'abord, nous verrons que la plupart des tensions que recouvre cette question proviennent du fait que nous pensons sur le modèle économique de la production logicielle, ce qui est une erreur.

(Une dernière remarque avant l'exposé : la dissertation et le plaidoyer en faveur du développement à source ouvert ne doivent pas être compris comme une thèse selon laquelle le développement à source fermé est intrinsèquement incorrect, ni comme un exposé visant les droits de propriété industrielle dans le logiciel, ni comme un appel altruiste au « partage ». Bien que ces arguments sont toujours les préférés d'une minorité dans la communauté du développement à source ouvert, l'expérience, depuis 16 (), a mis en lumière leur caractère facultatif. La défense du logiciel libre doit se tenir à des raisonnements économiques et pratiques — une meilleure qualité, une plus haute fiabilité, des coûts moindres et un choix plus diversifié.)

### 3 L'illusion des biens manufacturés

Nous devons pour commencer admettre que les programmes d'ordinateurs, comme tous les outils ou les biens en termes de capitaux, ont deux valeurs économiques distinctes. La *valeur d'utilisation*, et le *prix d'acquisition*, ou *valeur de vente*.

La *valeur d'utilisation* d'un programme est sa valeur économique en tant qu'outil. Son *prix d'acquisition* — ou *de vente* est sa valeur sur le marché (en termes économiques, la première est la valeur en tant que bien final, la deuxième en tant que bien intermédiaire).

La plupart des gens, quand ils tentent de raisonner sur l'économie de la production des logiciels, ont tendance à se placer dans un contexte « d'usine », qui est fondé sur les prémisses suivantes.

1. Une grosse partie du temps des développeurs est payée par le prix de vente.
2. Le prix de vente d'un logiciel est proportionnel à son coût de développement (c'est-à-dire le prix des ressources nécessaires pour le dupliquer) et à sa valeur d'utilisation.

En d'autres mots, les gens ont une forte tendance à considérer le logiciel comme n'importe quel autre bien manufacturé. Mais ces deux considérations sont fausses, comme on peut le démontrer.

D'abord, le code écrit pour la vente n'est que la partie émergée de l'iceberg de la programmation. Dans l'ère qui a précédé les micro-ordinateurs, tout le monde savait que 90 % du code était écrit en interne par des banques et des compagnies d'assurances. Ce n'est probablement plus le cas — d'autres industries sont devenues bien plus dépendantes du logiciel depuis, et la part de l'industrie de la finance a sans doute baissé — mais nous verrons sous peu que les faits empiriques montrent que 95 % du code est encore produit en interne par ceux qui en ont besoin.

Ce code inclut la plus grosse partie du système d'information et de gestion, les adaptations financières et de base de données sur mesure dont toute entreprise, grande ou moyenne, a besoin. Cela inclut également du code très technique comme les pilotes de périphériques (presque personne ne gagne de l'argent en écrivant des pilotes de périphériques, un point sur lequel nous reviendrons). Cela inclut également tous les types de codes embarqués, pour des machines de plus en plus bardées de puces depuis les machines-outils, jusqu'au grille-pain, en passant par les voitures, les avions, et les fours à micro-ondes.

La plus grosse partie de ce code développé en interne est à ce point dépendant de l'environnement que sa réutilisation, voire sa copie, sont très difficiles (cela est vrai que « l'environnement » soit un ensemble de procédures dans un bureau ou le système d'injection du fioul dans une turbine). Cela étant, l'environnement évoluant, il y a beaucoup de demande et de travail pour que le logiciel suive.

C'est ce qu'on appelle la « maintenance », et tout ingénieur ou analyste vous dira que c'est ce qui constitue la plus grosse partie (plus de 75 %) de la paye des programmeurs. Nous sommes d'accord pour dire que la plupart des heures des programmeurs sont employées (et d'ailleurs, les programmeurs salariés sont payés pour cela) à écrire ou maintenir du code interne qui n'a aucune valeur en termes de vente ou d'achat — un fait que le lecteur pourra constater facilement en consultant la liste des offres d'emplois pour programmeurs dans les petites annonces.

Parcourir les offres d'emplois du journal local est une expérience enrichissante que je presse le lecteur d'effectuer. Examinez les offres contenues dans la section programmation, traitement des données, et ingénierie du logiciel, et trouvez celles qui concernent le développement d'un logiciel. Répartissez-les en catégories suivant que ce logiciel est destiné à la vente ou à l'utilisation en interne.

Il deviendra rapidement clair que, même en donnant la définition la plus inclusive possible de « vente », dix-neuf offres sur vingt au moins concernent des logiciels développés dans le seul but d'être utilisés en interne (c'est-à-dire, en tant que biens intermédiaires). Ceci est la raison pour laquelle nous pensons que seulement 5 % de l'industrie se consacre à la vente. Remarquez, cependant, que la suite de l'analyse n'est pas étroitement liée à cette proportion ; si elle valait 15 % ou même 20 %, les conséquences économiques seraient les mêmes.

(Quand je donne une conférence sur le sujet, je commence mon discours en posant deux questions : combien de personnes dans l'audience sont payées pour écrire du logiciel et pour combien d'entre ces dernières le salaire est lié à la vente de leur logiciel. Généralement, une forêt de mains se dresse à la première question, et très peu ou pas du tout pour la deuxième, ce qui provoque toujours une grande surprise dans l'assistance.)

Ensuite, la théorie selon laquelle la valeur du logiciel est couplée à son développement ou à son coût de remplacement est encore plus facilement démolie si l'on observe le comportement réel des consommateurs. Il y a beaucoup de biens pour lesquels cela est vrai (tant qu'ils ne se déprécient pas) — la nourriture, les voitures, les machines-outils. Il y a même beaucoup de biens immatériels dont la valeur de vente est en rapport avec le coût de développement ou de remplacement — droits de reproduction de la musique, cartes géographiques, bases de données, pour ne citer que quelques exemples. De tels biens peuvent garder leur valeur de marché, et même l'accroître après que le vendeur originel ne soit plus pris en compte.

Au contraire, lorsqu'une entreprise qui vend du logiciel se retire du marché (ou lorsque le produit est presque abandonné), le prix maximum que les consommateurs accepteront de payer se rapproche très rapidement de zéro, indépendamment de son utilité théorique ou du coût de développement des équivalents fonctionnels (pour vérifier cette assertion, examinez les fins de séries chez n'importe quel vendeur de logiciels près de chez

vous).

Le comportement des revendeurs lorsque les vendeurs de logiciels cessent la production d'un logiciel est très instructif. Il montre qu'ils savent quelque chose que les fabricants, eux, ignorent. Voici ce qu'ils savent : le prix qu'un consommateur paiera pour un logiciel dépend effectivement *de la valeur de retour espérée en terme de service de la part du fabricant* (ce retour peut ici être compris au sens large des améliorations, des mises à jour, des projets s'élargissant, etc...)

En d'autres termes, le logiciel est largement une industrie de service sous hypnose, qui croit très fortement mais à tort qu'elle est une industrie de biens manufacturés.

Il est intéressant de se demander pourquoi nous sommes incités à raisonner ainsi. C'est tout simplement parce que la petite portion de l'industrie du logiciel qui est fabriqué pour être vendu est aussi la seule qui promeut ses produits. Ainsi, la plupart des produits dont on parle le plus sont des choses éphémères comme des jeux, dont on peut penser qu'on n'en attend pas de services (ce serait une exception plutôt que la règle) 16 ( ).

C'est également intéressant de remarquer que cette illusion des biens manufacturés encourage des structures de prix qui sont pathologiquement sans rapport avec les coûts de développement. Si (comme c'est généralement admis) plus de 75 % des coûts du cycle de vie typique d'un logiciel se trouvent dans la maintenance, les corrections d'erreurs, et les extensions, alors la politique qui consiste à fixer un prix élevé à l'achat et presque pas de budget pour l'assistance est destinée à produire des résultats qui desserviront tout le monde.

Les consommateurs y perdent, car bien que le logiciel soit une industrie de service, les motivations de ce modèle de l'usine empêchent le fabricant d'offrir un service *compétent*. Si le fabricant gagne de l'argent en vendant des octets, ses efforts viseront à produire et à sortir des octets. Le service d'assistance, qui ne fait pas de profit, deviendra un placard réservé aux moins compétents, et n'aura en fait de ressources que le strict nécessaire lui permettant de parer activement à la plus grosse partie des plaintes des clients.

L'autre facette de la médaille est que la plupart des fabricants qui raisonnent dans ce schéma de pensée échoueront à long terme. Produire des services d'assistance à perte, à partir d'un prix de vente fixé, ne peut fonctionner que dans le cas où le marché s'étend assez rapidement pour couvrir les coûts du support et ceux du cycle de vie des logiciels d'hier avec les revenus de demain. Une fois que le marché arrive à maturation et que les ventes ralentissent, la plupart des vendeurs n'ont d'autre choix que de réduire les coûts en cessant toute activité sur le produit.

Que ceci soit fait explicitement (en arrêtant le produit) ou implicitement (en rendant l'assistance difficile d'accès), cela a l'effet de conduire les clients chez le concurrent (parce que cela réduit à néant la valeur future que l'on peut attendre du produit, qui est conditionnée par ce service). À court terme, on peut s'en tirer en faisant passer des éditions avec des corrections d'erreurs pour un nouveau produit, avec un nouveau prix, mais le consommateur se fatigue rapidement. À long terme, la seule façon de s'en sortir est donc de n'avoir aucun concurrent — c'est-à-dire, d'avoir un véritable monopole sur son marché. À la fin, il ne peut en rester qu'un.

Et, bien sûr, nous avons régulièrement constaté que cette façon de ne plus assurer l'assistance pour ses produits a tué jusqu'à des concurrents qui tenaient une bonne seconde place dans une niche de marché (ceci doit sembler particulièrement clair à quiconque a observé l'historique des systèmes d'exploitation propriétaires des *PC* compatibles *IBM*, des traitements de texte, des programmes de comptabilité et n'importe quel logiciel financier en général). Les motivations perverses, induites par le modèle de l'usine, conduisent à un modèle où le gagnant gagne tout, et où même ses clients finissent par y perdre.

Alors, à défaut du modèle de l'usine, que prendre ? Pour pouvoir assumer les coûts du cycle de vie d'un logiciel de manière efficace (« efficace » étant à la fois pris dans ses sens informel et économique), nous devons trouver une structure de prix fondée sur des contrats de services, des inscriptions, et un échange *continu* de valeur entre l'acheteur et le vendeur. Dans les conditions de recherche de l'efficacité du libre marché, nous pouvons alors prévoir que c'est le type de structure de prix que la plupart des sociétés de

logiciels matures suivront, à terme.

Ces propos commencent à nous faire pressentir pourquoi le logiciel à source ouvert est un défi de plus en plus important, tant du point de vue technique que du point de vue économique, par rapport à l'ordre établi. L'effet de faire du logiciel « libre et gratuit », semble-t-il, est de nous forcer à nous situer dans un monde où le service serait tout puissant — et d'exposer la faiblesse du support en lequel nous avons toujours cru — le prix d'acquisition des bits des logiciels à sources fermés.

Le mot « libre » est également trompeur pour une autre raison. Baisser le prix d'un bien tend à augmenter, et non pas à faire baisser, l'investissement total dans l'infrastructure qui le soutient. Quand le prix des voitures baisse, les mécaniciens automobiles sont plus prisés — c'est pourquoi il est peu probable que les 5 % de programmeurs qui sont aujourd'hui financés par le prix d'acquisition ne souffrent dans un monde à sources ouverts. Ceux qui perdront au change ne seront pas les programmeurs, mais les investisseurs qui ont parié sur des stratégies à sources fermés là où elles ne sont pas appropriées.

## 4 Le mythe « il faut que l'Information soit libre »

Il existe un autre mythe, d'égale importance mais opposé à la modèle de l'usine, trompeur, qui lui aussi perturbe souvent la compréhension de l'économie du logiciel à sources ouverts. C'est le fait qu'« il faut que l'Information soit libre »<sup>1</sup>. En général, cela revient à dire que le coût marginal de reproduction de l'information, nul, implique que son coût de revient doit lui aussi être nul.

On balaiera d'un revers de la main la forme la plus générale de ce mythe en considérant la valeur des informations qui concèdent un droit à un bien disputé — une carte au trésor, par exemple, ou le numéro d'un compte bancaire en Suisse, ou une prétention à des services, comme le mot de passe d'un compte informatique. Même si on peut dupliquer à coût nul l'information concédant le droit au bien disputé, ce n'est pas le cas de ce bien lui-même. C'est pourquoi les renseignements dont il est question peuvent hériter du coût marginal, non nul, de l'objet.

On ne mentionne ce mythe que pour démontrer qu'il n'a aucun rapport avec les arguments participant de l'économie des sources ouverts ; comme on le verra plus tard, de tels arguments sont difficiles à contrer, même si on fait l'hypothèse que le logiciel *a* effectivement le structure de valeur (non nulle) d'un bien manufacturé. Il nous faut alors taquiner la question de savoir si le logiciel « devrait » être libre ou pas.

## 5 Les *communs* inversés

Ayant observé le modèle dominant d'un oeil sceptique, voyons s'il est possible d'en construire un autre — une explication têtue de ce qui fait que la coopération dans le cadre de projets à sources ouverts soit défendable économiquement.

Cette question mérite qu'on l'examine à différents niveaux. Il nous faut d'abord expliquer le comportement d'individus qui contribuent dans le cadre de projets à sources ouverts ; il nous faut aussi comprendre les forces économiques qui financent la coopération sur des projets à sources ouverts tels que *Linux* ou *Apache*.

Ici encore, il nous faudra d'abord démolir un modèle très répandu parmi les foules et qui entrave la compréhension. Sur chacune des tentatives d'explication de comportements coopératifs, plane l'ombre de la *Tragédie des communs*, de Garret Hardin<sup>2</sup>.

Dans ce texte célèbre, Hardin nous demande d'imaginer un pré, propriété collective d'un village de paysans, qui y font paître leur bétail. Mais les animaux dégradent les communs, en arrachant l'herbe et en laissant des

1. N.D.T. : phrase célèbre dans le jargon des informaticiens, "*Information wants to be free*".

2. N.D.T. : titre original: *Tragedy of the Commons*.

portions boueuses derrière eux, qui ne recouvreront que lentement leur couverture végétale. En l'absence de politique consentie (et appliquée, fût-ce par la force) pour allouer des droits et des limites à chacun, l'intérêt de chacune des parties est d'y mener un maximum de têtes le plus vite possible, pour en extraire toute la valeur avant que les communs ne soient réduits à une mer de boue.

La plupart des gens ont un modèle intuitif du comportement coopératif très semblable à cette fable. En réalité, ce n'est pas une bonne représentation des problèmes économiques des sources ouverts, qui relèvent plus du passager clandestin (sous-provisionnement) que de la congestion des biens publics (sur-utilisation). Néanmoins, c'est cette analogie qui a incité la plupart des objections spontanées qu'on m'a opposées.

La tragédie des communs ne prédit que trois résultats possibles. Le premier est une mer de boue. Le deuxième est qu'un acteur disposant d'un pouvoir coercitif ne fasse appliquer une politique d'allocation au nom du village (c'est la solution communiste). Le dernier est que les communs soient divisés et que les membres du village n'en clôturent des portions qu'ils pourront défendre et gérer correctement (c'est la solution des droits de propriété).

Quand, par réflexe, les gens appliquent ce modèle à la coopération dans le cadre de projets à sources ouverts, ils s'attendent à ce qu'elle soit instable et possède une demi-vie courte. Puisqu'il n'existe aucune manière évidente d'appliquer une politique d'allocation au temps des programmeurs sur l'*Internet*, ce modèle conduit tout droit à la prédiction que les communs vont être séparés, que diverses portions de logiciels seront rendues propriétaires, et que la quantité de travaux réinjectés dans le pot commun décroîtra rapidement.

En fait, les faits empiriques sont clairs, et c'est la tendance contraire qui a lieu. Le spectre et le modèle des développements à sources ouverts (mesurés, par exemple, en termes de soumissions sur *Metalab* ou en annonces sur *freshmeat.net* chaque jour) croît continûment. Il est clair qu'on assiste à un phénomène dont le modèle de la « Tragédie des communs » échoue à rendre compte.

Une partie de la réponse tient certainement au fait qu'utiliser du logiciel n'en décroît en rien la valeur. En fait, le fait qu'un logiciel à sources ouverts soit largement utilisé tend à en *augmenter* la valeur, puisque les utilisateurs contribuent par leurs propres corrections et ajouts (par des *patches* au code). Dans ces communs inversés, l'herbe repousse plus haut quand elle est broutée.

La réponse tient aussi dans le fait que la valeur supposée, sur le marché, de petits correctifs à une base commune de sources est difficile à capturer. Supposons que j'écrive un correctif pour un bogue irritant, et supposons que de nombreuses personnes comprennent que ce travail a une valeur économique; comment puis-je la collecter de tous ces gens? Les systèmes conventionnels de paiement ont des surcoûts suffisamment élevés pour rendre impraticables les micro-paiements qui seraient sinon appropriés.

On divaguera moins encore si on signale que cette valeur n'est pas seulement difficile à capturer, elle est même, dans le cas général, difficile à *déterminer*. Soit l'expérience de pensée suivante: supposons que l'*Internet* soit équipé du système de micro-paiements idéal en théorie — sûr, accessible à tous, sans surcoûts. Supposons maintenant que vous ayez écrit un *patch* intitulé « Divers correctifs pour le noyau *Linux* ». Comment un acheteur potentiel, n'ayant pas encore vu votre travail, peut savoir ce qu'il est raisonnable d'en proposer?

Nous nous trouvons en présence de l'image à travers une maison aux miroirs de foire du « problème de calcul » de F.A. Hayek — il faudrait un sur-être, à la fois capable d'évaluer la valeur fonctionnelle des correctifs et habilité à positionner les prix en fonction, pour lubrifier le commerce.

Malheureusement, il y a rupture de stock sur les sur-être, aussi l'auteur du correctif, T. Codeur-Lambda a deux possibilités: garder le correctif par devers lui, ou en faire gracieusement bénéficier les autres. La première possibilité ne lui apporte rien. Le second choix ne lui apportera peut-être rien, mais il peut aussi encourager d'autres à donner des correctifs qui résoudront certains des problèmes de T. Codeur-Lambda à l'avenir. Le second choix, apparemment altruiste, est en fait optimalement égoïste au sens de la théorie des jeux.

En analysant ce type de coopération, il est important de remarquer que même en présence d'un problème de passager clandestin (il se peut que le travail se fasse mal en l'absence d'argent ou de compensations équiva-

lentes), il ne grandit pas avec le nombre d'utilisateurs finals. La complexité et les surcoûts en communication d'un projet à sources ouverts ne dépend pratiquement que du nombre de développeurs impliqués ; disposer d'un plus grand nombre d'utilisateurs finals, qui n'iront jamais inspecter le code source, ne coûte rien, dans la pratique. Cela peut augmenter le nombre de questions idiotes sur les listes de diffusion du projet, mais on peut faciliter anticiper cela en maintenant une foire aux questions (FAQ) et en ignorant avec insouciance ceux qui posent des questions sans l'avoir consultée (et en fait, ces deux pratiques sont typiques).

Les véritables problèmes de passagers clandestins dans le logiciel à sources ouverts dépendent plus des coûts de friction lors de la soumission de correctifs que de quoi que ce soit d'autre. Un contributeur potentiel peu averti du jeu culturel des réputations (voir 16 ()) peut, en l'absence de compensation financière, penser « Cela n'est pas la peine de soumettre ce correctif car il me faudra nettoyer le *patch*, écrire une entrée dans le fichier *ChangeLog*, et signer les papiers légaux de la FSF... ». C'est pour cette raison que le nombre de contributeurs (et, au second ordre, la réussite) des projets est fortement liée et inversement proportionnelle au nombre de sas que chaque projet impose à chaque utilisateur de traverser avant de l'autoriser à contribuer. De tels coûts de frictions peuvent être politiques ou mécaniques. Corrélés, ils peuvent expliquer pourquoi la culture *Linux*, amorphe et lâche, a attiré à elle une énergie coopérative de plusieurs ordres de grandeur supérieure aux efforts plus sévèrement organisés et plus centralisés des projets *\*BSD*, et pourquoi la montée de *Linux* a fait de l'ombre à la *Free Software Foundation*.

Tout cela est bel et bon. Mais c'est une explication *a posteriori* de ce que T. Codeur-Lambda doit faire de son correctif une fois qu'il en dispose. Il faut aussi donner une explication économique au fait que T. C-L a eu les moyens pour écrire ce correctif, plutôt que de devoir travailler sur un programme à sources fermés qui aurait pu lui octroyer un revenu sous la forme de prix d'acquisition. Quels modèles économiques créent des niches dans lesquelles le développement à sources ouverts peut prospérer ?

## 6 Les bonnes raisons pour travailler en sources fermés

Avant de dresser l'herbier des modèles économiques fonctionnant avec des logiciels à sources ouverts, il nous faut d'abord estimer le profit qu'on peut tirer de l'exclusivité. Que protègent exactement ceux qui ferment leur code source ?

Supposons que vous embauchiez quelqu'un pour écrire un paquetage de comptabilité spécialisé pour votre société. Ce problème ne sera en rien mieux résolu si les sources sont fermés au lieu d'être ouverts ; la seule raison rationnelles qui peut vous faire préférer des sources fermés est l'intention de revendre ce paquetage à d'autres, ou interdire à vos concurrents de l'utiliser.

La réponse évidente est que ce faisant, vous protégez le prix d'acquisition, mais pour 95 % des logiciels écrits pour un usage interne cela ne tient pas. Quels autres gains peut-on retirer du fait de fermer ses sources ?

Le deuxième cas exposé ici (protéger un avantage compétitif) mérite qu'on l'examine de plus près. Supposez que vous ouvriez les sources de ce paquetage de comptabilité. Il devient populaire et profite des améliorations de la communauté. Désormais, votre concurrent lui aussi commence à l'utiliser. Il en retire donc le bénéfice sans avoir participé au coût de développement, et vous reprend des parts de marché. Cet argument s'oppose-t-il à l'ouverture des sources ?

Peut-être — et peut-être pas. La véritable question est de savoir si vous gagnez plus à étendre la base de développement que vous ne perdez suite au regain de compétitivité du passager clandestin. Nombreux sont ceux qui raisonnent incorrectement ici (a) en ignorant l'avantage fonctionnel apporté par le renforcement de l'équipe de développement. (b) en ne considérant pas que les coûts de développement comme irrécupérables, car par hypothèse, il vous a fallu les acquitter de toutes manières, aussi les faire rentrer dans le bilan du coût de l'ouverture des sources (si telle est votre décision) est une erreur.

On trouve d'autres raisons de fermer les sources, complètement irrationnelles celles-là. Vous pouvez, par exemple, travailler sous l'idée reçue incorrecte que fermer les sources rendra votre affaire plus sécurisée vis-

à-vis des intrus et autres indécats. Si vraiment c'est le cas, je vous recommande une thérapie immédiate avec un cryptanalyste. Les véritables professionnels de la paranoïa ne font pas confiance à des programmes aux sources fermés, car ils ont appris la leçon à leurs dépens. La sécurité est une question de fiabilité ; on ne peut faire confiance qu'à des algorithmes et des implémentations qui ont été consciencieusement revus par les pairs des programmeurs et des concepteurs.

## 7 Les modèles financés par la valeur d'utilisation

Il est capital de remarquer que dans la distinction entre valeur d'utilisation et prix d'acquisition, seul le *prix d'acquisition* est menacé par l'ouverture des sources ; la valeur d'utilisation demeure identique.

Si c'est la valeur d'utilisation qui pilote réellement le développement de logiciels, plus que le prix d'acquisition, et si (thèse défendue dans l'article 16 ()) le développement à sources ouverts est effectivement plus efficace et direct que le développement à sources fermés, alors il faut nous attendre à découvrir des circonstances pour lesquelles seule la valeur d'utilisation attendue finance de manière durable le développement à sources ouverts.

Et de fait, il n'est pas difficile d'identifier au moins deux modèles importants où seule la valeur d'utilisation finance les salaires de développeurs à plein temps sur des projets à sources ouverts.

### 7.1 Le cas *Apache* : partage des coûts

Supposons que vous travailliez pour une société qui a le besoin critique d'un serveur *web* de haute fiabilité et capable de traiter des volumes conséquents. Vous souhaitez peut-être l'employer à des fins de commerce électronique, vous êtes peut-être une entreprise de communication vendant des espaces publicitaires, ou encore un site portail. Il faut que votre serveur fonctionne 24 heures sur 24, 7 jours sur 7, il faut qu'il soit rapide, il faut qu'il soit personnalisable.

Comment allez-vous procéder pour arriver à vos fins ? Vous pouvez choisir l'une des trois stratégies suivantes :

- **Acheter un serveur *web* propriétaire.** Dans ce cas, vous faites le pari que le cahier des charges du vendeur correspondra au vôtre et que ce dernier aura la compétence technique de l'implémenter proprement. Même en ces deux assertions, il est probable que la personnalisation du produit montre ses limites ; vous ne pourrez le modifier qu'à travers les points d'accès que le vendeur vous aura fournis. Ce choix du serveur *web* propriétaire n'est pas celui pour lequel optent la plupart des décideurs.
- **Développer votre propre serveur.** Ce n'est pas là une option à écarter *a priori* ; les serveurs *web* ne sont pas très compliqués, beaucoup moins que les arpenteurs, et un serveur spécialisé peut très bien être incomplet et humble. En suivant cette voie, vous obtiendrez exactement les fonctionnalités et la personnalisation souhaitées, mais il vous faudra payer ces avantages en temps de développement. Votre société aura peut-être des problèmes le jour où vous la quitterez, pour la retraite ou pour un autre emploi.
- **Rejoindre le groupe *Apache*.** Le serveur *Apache* a été mis au point par un groupe de webmasters reliés par l'*Internet*, qui ont compris qu'il était plus sensé d'unir leurs efforts et d'améliorer un code commun que de démarrer un grand nombre d'efforts de développement parallèles. De cette manière, ils ont pu réunir à la fois les avantages du développement d'un serveur personnel et le puissant effet de débogage de la revue par un grand nombre de pairs en parallèle.

Les avantages du choix d'*Apache* sont certains. Dans quelle mesure, nous pouvons en juger en lisant tous les mois le sondage de *Netcraft*, qui a montré depuis sa mise en place une progression inéluctable du le serveur *Apache* au détriment de tous les serveurs propriétaires. En juin 1999, *Apache* et ses dérivés dirigent 61%

*des serveurs* <<http://www.netcraft.com/survey/>> *web* de par le monde — sans propriétaire légal, sans promotion, et sans société de services offrant des contrats de maintenance derrière.

On peut généraliser l'histoire d'*Apache* à un modèle où les utilisateurs de logiciels trouvent leur avantage dans le financement d'un développement à sources ouvertes car cette approche leur fournit un produit meilleur que ce qu'ils auraient pu sinon obtenir, à un prix plus faible.

## 7.2 Le cas *Cisco* : étaler les risques

Voici quelques années, deux programmeurs de la société *Cisco* (le fabricant d'équipements de réseau) ont eu pour mission d'écrire un système de files d'impression à l'intention du réseau d'entreprise de *Cisco*. C'était un véritable défi. Non content de proposer à tout utilisateur **A** la possibilité d'imprimer sur une imprimante **B** quelconque (qu'elle soit dans la pièce adjacente ou à quelques milliers de kilomètres), le système devait s'assurer que dans le cas d'une panne de papier ou d'encre la requête soit redirigée vers une autre imprimante, proche de l'imprimante cible initiale. Il fallait également que ce système rapporte de tels problèmes à administrateur des imprimantes.

Le duo a proposé des modifications astucieuses au logiciel standard de files d'impression sous *Unix*, ainsi que quelques scripts d'interfaçage, qui s'acquittaient de la tâche. Ils ont ensuite compris qu'eux, et *Cisco*, avaient un petit ennui.

En effet, aucun d'entre eux n'allait rester employé chez *Cisco* éternellement. À terme, ces deux programmeurs seraient partis, et le logiciel, perdant ses mainteneurs, aurait commencé à pourrir (c'est-à-dire à désynchroniser sans cesse davantage des conditions réelles d'exploitation). Aucun développeur n'aime voir son oeuvre sombrer dans un tel oubli, et notre intrépide duo a eu le sentiment que la société *Cisco* avait financé une solution en faisant l'hypothèse déraisonnable que cette dernière leur survivrait au sein de la société.

C'est pourquoi ils sont allés trouver leur chef et l'ont convaincu d'autoriser la publication du logiciel de files d'impression sous une licence à sources ouvertes. Ils ont fondé leur argumentation sur le fait que la société *Cisco* n'y perdrait rien en termes de prix d'acquisition, et avait donc tout à y gagner sur les autres tableaux. En encourageant la croissance d'une communauté d'utilisateurs et de co-développeurs issus de nombreuses sociétés, *Cisco* pouvait se prémunir efficacement contre la perte des développeurs originaux du logiciel.

On peut généraliser l'histoire de *Cisco* à un modèle dans lequel les sources ouvertes n'ont pas tant pour but de réduire les coûts que d'étaler les risques. Toutes les parties trouvent que l'ouverture des sources, et la présence d'une communauté qui collabore, financée par de multiples flux indépendants, fournit une assurance économiquement appréciable — suffisamment pour lui dégager des crédits.

## 8 En quoi le prix d'acquisition pose problème

L'ouverture des sources complique la tâche de capturer un prix d'acquisition direct sur le logiciel. Ce n'est pas une difficulté technique : le code source n'est ni plus ni moins copiable que les binaires, et l'application des lois de propriété intellectuelle visant à faire respecter les copyrights et les licences afin de permettre la collecte des prix d'acquisition n'a pas de raison d'être plus difficile dans le cas des produits à sources ouvertes que dans le cas des produits à sources fermés.

La difficulté est plutôt inhérente au contrat social qui sous-tend le développement à sources ouvertes. Pour trois raisons se renforçant mutuellement, la plupart des licences à sources ouvertes interdisent la plupart des restrictions sur l'utilisation, la redistribution, et la modification, qui pourraient faciliter la capture de revenus directement liés à la vente. Pour comprendre ces raisons, il nous faut examiner le contexte social dans lequel ces licences ont évolué ; la culture *hacker* <<http://www.fermigier.com/fermigier/hacker-howto-fr.html>> sur l'*Internet*.

En dépit des mythes qui circulent encore largement (en 1999), en dehors de celle-ci, sur la culture *hacker*, aucune de ces raisons n'a pour origine une hostilité au marché. Même si une minorité de *hackers* demeurent hostiles aux motivations lucratives, la communauté, en général, souhaite coopérer avec des distributeurs de *Linux* à but lucratif comme *Red Hat*, *SuSE*, *Caldera*, ce qui montre que la plupart des *hackers* sont heureux de travailler avec le monde des entreprises quand celui-ci sert leurs fins. Les véritables raisons pour lesquelles les *hackers* apprécient peu les licences permettant l'appropriation d'un flux de revenus direct sont plus subtiles et intéressantes.

L'une de ces raisons est en rapport avec la symétrie. Alors que la plupart des développeurs à sources ouverts ne sont pas intrinsèquement opposés à l'idée que d'autres profitent des cadeaux qu'eux font à la communauté, la plupart exigent également qu'aucune partie (à l'exception possible de celle qui est à l'origine d'une portion de code) ne se retrouve dans une position *privilégiée* pour tirer profit d'un projet. T. Codeur-Lambda accepte que *Toto & compagnie* puisse gagner de l'argent en vendant ses logiciels ou ses correctifs, si T. C-L peut lui aussi, potentiellement, en faire autant.

Une autre de ces raisons est en rapport avec les conséquences non désirées. Les *hackers* ont observé que les licences qui incluent des restrictions et des taxes pour l'utilisation « commerciale » ou la vente (la forme la plus courante de tenter de capturer une valeur commerciale directe, et ce n'est pas à première vue un désir déraisonnable) ont divers effets secondaires dissuasifs. L'un d'entre eux est de faire planer l'ombre des poursuites en justice sur les activités de redistribution dans des compilations sur *CD-ROM* peu onéreux, qu'on souhaiterait idéalement encourager. Plus généralement, les restrictions sur l'utilisation, la vente, la modification, la distribution (et d'autres complications des licences) entraînent un surcoût temporel et financier pour s'assurer qu'on respecte bien toutes les conditions et (alors que le nombre de paquetages manipulés augmente), une explosion combinatoire d'incertitudes et des risques juridiques réels. De tels résultats sont considérés dommageables, et la pression sociale est donc forte qui tend à promouvoir des licences simples et dénuées de telles restrictions.

La dernière raison, la plus critique aussi, est en rapport avec la préservation de la revue par les pairs et de la dynamique de la culture du don décrite dans 16 (). Les restrictions sur les licences mises au point pour protéger la propriété intellectuelle ou pour capturer des sources de revenus ont souvent l'effet de rendre impossible toute scission du projet (c'est par exemple le cas de la licence de la société *Sun*, prétendue « sources communautaires » (*Community Source*), mise au point pour les programmes *Jini* et *Java*). Même si on n'aime pas les scissions et si on n'y a recours qu'en dernière extrémité (pour des raisons exposées en détail dans 16 ()), ce dernier recours a une importance capitale dans les cas où le mainteneur est incompetent ou déficient (dans les cas par exemple où il fait le choix d'une licence plus fermée).

Les *hackers* de la communauté lèvent pour certains le pied sur l'argument de symétrie ; c'est pourquoi on tolère des licences comme la *NPL* de la société *Netscape*, qui donne des privilèges d'exploitation à ceux de qui le code est issu (spécifiquement, dans le cas de la *NPL*, le droit exclusif d'utiliser le code ouvert du projet *Mozilla* dans des produits dérivés, y compris à sources fermés). Moins nombreux sont ceux qui ferment les yeux sur la raison tendant à éviter les conséquences non désirées, et personne n'accepte l'impossibilité de toute scission (c'est pourquoi les schémas de « licences communautaires » de la société *Sun* pour *Jini* et *Java* ont été rejetées en masse par la communauté).

Ces raisons expliquent les clauses de la *définition de l'Open Source*, qui fut écrite pour exprimer le consensus de la communauté des *hackers* à propos des spécificités critiques des licences standard (la *GPL*, la licence *BSD*, la licence *MIT*, et la licence *Artistic*). Ces clauses ont l'effet (mais pas l'intention) de rendre extrêmement difficile la capture directe d'une valeur commerciale.

## 9 Les modèles exploitant une valeur d'acquisition indirecte

Néanmoins, il y a deux manières de gagner des marchés avec des services liés au logiciel, qui capturent ce qu'on pourrait appeler une valeur d'acquisition indirecte. On dénombre dans ce domaine cinq modèles connus et deux modèles pressentis (mais on pourra en développer d'autres à l'avenir).

### 9.1 Vendre à perte pour se positionner sur un marché

Dans ce modèle, on utilise des logiciels au source ouvert pour créer ou maintenir une position dans un marché pour un logiciel propriétaire qui engendre un flux financier direct. Dans la variante la plus fréquente, le logiciel client, au source ouvert, favorise la vente d'un logiciel serveur, ou des revenus pour les inscriptions et la publicité associés à un site portail.

La société *Netscape Communications, Inc.* avait opté pour une telle stratégie quand elle a, au début de l'année 1998, publié le source de son arpenteur *Mozilla* sous une licence libre. Le côté arpenteur de leurs activités correspondait à 13 % de leurs bénéfices et a commencé à céder du terrain quand la société *Microsoft* a mis sur le marché son arpenteur *Internet Explorer*. Une campagne promotionnelle agressive (et des pratiques de ventes par lots contestables qui leur vaudraient ensuite une poursuite en justice pour abus de position dominante) ont rapidement mis à mal la part de marché du navigateur de la société *Netscape*, la société *Microsoft* menaçant d'obtenir le monopole sur le marché des arpenteurs et d'utiliser le contrôle qu'ils auraient ainsi obtenu sur le langage *HTML* pour expulser *Netscape* du marché des serveurs.

On ouvrant le source de leur arpenteur, qui avait encore les faveurs de beaucoup, la société *Netscape* a tout à coup ôté à *Microsoft* la possibilité de contrôler entièrement le marché des arpenteurs. Ils s'attendaient à ce que la collaboration dans le cadre d'un projet de développement au source ouvert accélérerait la mise au point et le débogage de leur arpenteur, et espéraient que l'*IE* de *Microsoft* serait réduit à les suivre à la trace, définitivement empêché de redéfinir seul la norme *HTML*.

Cette stratégie a fonctionné. En novembre 1998, la société *Netscape* a commencé à reprendre des parts de marché à *IE*. Quand, début 1999, la société *AOL* a racheté *Netscape*, l'avantage sur la concurrence que leur conférait le projet *Mozilla* était si clair que l'un des premiers engagements publics d'*AOL* fut de continuer à encourager le projet *Mozilla*, bien qu'il en était encore au stade *alpha*.

### 9.2 Gel des gadgets

Ce modèle concerne les fabricants de matériel (dans ce contexte, on entendra par « matériel » toute l'électronique, des cartes *Ethernet* ou autres cartes périphériques aux systèmes informatiques complets). Les pressions du marché ont forcé les sociétés de matériel à écrire et à maintenir du logiciel (des pilotes de périphériques aux systèmes d'exploitation complets en passant par les outils de configuration), mais le logiciel n'est pas en soi une source de bénéfices. C'est un surcoût — et souvent il est non négligeable.

Dans cette situation, ouvrir le source ne fait pas un pli. Il n'y a aucune source de revenus à perdre, il n'y a donc aucun inconvénient. Le vendeur y gagne un nombre de développeurs significativement plus important, des réponses plus rapides et plus souples répondant aux besoins des consommateurs, et une meilleure fiabilité à travers la revue par les pairs. Il obtient gratuitement des ports vers de nouveaux environnements. Il y gagne probablement, également, des consommateurs plus fidèles, car les équipes techniques de ses clients consacrent plus de temps au code pour développer les personnalisations dont ils ont besoin.

Certaines objections que les fabricants formulent couramment concernent spécifiquement l'ouverture des sources des pilotes matériels. Plutôt que de les mélanger avec la discussion générale, j'ai écrit une 17 (annexe) traitant exclusivement de ce sujet.

L'effet « à long terme » du source ouvert est particulièrement fort dans le cadre du gel des gadgets. On produit

et fournit une assistance technique pour les produits matériels durant une période finie, après laquelle les consommateurs sont livrés à eux-mêmes. Mais s'ils ont accès au source du pilote et peuvent le corriger selon leurs besoins, il est plus probable qu'ils seront heureux de choisir la même société pour leurs achats futurs.

Un exemple très marquant du gel des gadgets fut la décision de la société *Apple Computer*, mi-mars 1999, d'ouvrir le source de *Darwin*, le coeur de leur système d'exploitation *MacOSX*.

### 9.3 Donner la recette, ouvrir un restaurant

Dans ce modèle, on ouvre le source d'un logiciel pour ouvrir un marché, non pas pour un logiciel fermé (comme dans le cas *Vendre à perte pour se positionner sur un marché*) mais pour des services.

(J'avais d'abord appelé cette section « Donner le rasoir, vendre des lames de rasoir », mais le couplage est moins fort que l'analogie du rasoir et des lames de rasoir le laisse penser.)

C'est que la société *Red Hat* et d'autres distributions de *Linux* font. En réalité, ils ne vendent pas le logiciel, les bits eux-mêmes, mais la valeur ajoutée par l'assemblage, les tests d'un système d'exploitation garanti (ne serait-ce qu'implicitement) comme étant de qualité commerciale et compatible avec d'autres systèmes d'exploitation portant la même marque. D'autres éléments de leur proposition sont une assistance technique gratuite à l'installation et la fourniture d'options pour des contrats d'assistance étendus.

L'effet de construction de marché du source ouvert peut être très puissant, surtout pour les sociétés qui débutent forcément en tant que prestataires de services. Un cas de figure récent et très instructif est celui de *Digital Creations*, société de conception de sites *web* créée en 1998, spécialisée dans les bases de données complexes et les sites de transactions. Leur outil principal, leurs bijoux de famille de propriété intellectuelle, est un logiciel de publication objet qui a connu différentes incarnations et noms mais qui porte désormais celui de *Zope*.

Quand les gens de *Digital Creations* ont recherché des capitaux risque, leur investisseur a soigneusement évalué leur niche de marché prévue, leurs employés, et leurs outils. Il leur a ensuite recommandé d'ouvrir le source de *Zope*.

D'après les standards traditionnels de l'industrie, cela semble un choix complètement fou. La sagesse conventionnelle des écoles de gestion dicte que le coeur de la propriété intellectuelle d'une société, comme *Zope*, représente ses bijoux de famille, et qu'il ne faut en aucun cas s'en séparer. Mais l'investisseur de capitaux à risque a eu deux bonnes intuitions. L'une est que le véritable avantage de *Zope* réside en les cerveaux et le talent de ceux qui l'ont développé. La deuxième est que *Zope* produira plus de revenus en tant que catalyseur d'un nouveau marché qu'en tant qu'outil secret.

Pour s'en convaincre, comparons deux scénarios. Dans le scénario traditionnel, *Zope* demeure l'arme secrète de la société *Digital Creations*. Supposons qu'elle est très efficace. Cela aura pour résultat que cette entreprise sera capable d'une meilleure qualité pour des délais de livraison courts — *mais personne ne le sait*. Il sera facile de satisfaire les consommateurs, mais bien plus difficile de se bâtir une clientèle.

L'investisseur, au contraire, a compris qu'ouvrir le source de *Zope* pourrait fournir à *Digital Creations* une publicité remarquable pour sa véritable qualité : ses employés. Il a prédit que les consommateurs évaluant *Zope* trouveraient plus efficace de louer les services d'experts en la matière que de développer une expertise maison pour *Zope*.

L'un des responsables de *Zope* a depuis publiquement confirmé que leur stratégie de code source ouvert avait « ouvert de nombreuses portes qui seraient restées closes sinon ». Les clients potentiels répondent vraiment à la logique de la situation — *Digital Creations*, comme prévu, prospère.

Un autre exemple d'actualité est celui de la société *e-smith, inc.* Elle vend des contrats d'assistance technique pour un logiciel serveur clef en main pour l'*Internet* à source ouvert, un *Linux* personnalisé. L'une des responsables, décrivant la masse de téléchargements de leur logiciel sur leur site, déclare « La plupart des

sociétés parleraient de piratage de logiciel ; nous considérons que c'est de la libre mercatique. »

## 9.4 Accessoires

Dans ce modèle, on vend des accessoires pour du logiciel au source ouvert. Cela couvre toute la gamme, des objets comme les tasses et les *t-shirts* aux documentations éditées et produites par des professionnels.

*O'Reilly & Associates*, éditeurs de nombreux excellents ouvrages de référence sur le logiciel au source ouvert, fournissent un bon exemple de société d'accessoires. En réalité, la société *O'Reilly* emploie et assiste financièrement des *hackers* bien connus du monde du source ouvert (comme Larry Wall et Brian Behlendorf), afin de se construire une réputation dans le marché qu'elle s'est choisi.

## 9.5 Libérer l'avenir, vendre le présent

Dans ce modèle, on publie le logiciel sous forme de binaires et le source sous une licence fermée, avec une date limite à laquelle les restrictions prendront fin. Vous pouvez par exemple écrire une licence qui autorise la libre redistribution, interdit l'utilisation dans un but commercial sans acquitter d'obole, et garantir que le logiciel sera diffusé selon les termes de la *GPL* un an après sa sortie ou si le vendeur fait banqueroute.

Dans ce modèle, les consommateurs peuvent s'assurer que le produit est personnalisable selon leurs besoins, car ils disposent du source. Le produit est résistant à l'usure du temps — la licence garantit qu'une communauté travaillant selon un modèle de code source ouvert peut reprendre le produit si la société initiale passe la main.

Le prix et le volume de vente reposant sur ces attentes du consommateur, la société initiale peut fort bien disposer d'une source de revenus plus élevée en procédant ainsi qu'en le publiant uniquement sous une licence à source fermé. De plus, le code ancien passant sous *GPL*, il bénéficiera d'une sérieuse analyse par les pairs, de corrections de bogues, et d'ajouts de fonctionnalités mineures, qui ôteront à l'auteur originel du source une partie de la charge de la maintenance, qui représente 75 % du travail dans le cadre classique.

Ce modèle est suivi avec bonheur par la société *Aladdin Enterprises*, fabricants du célèbre programme *Ghostscript* (un interpréteur *PostScript* qui peut s'exprimer en le langage natif de nombreuses imprimantes).

Le principal inconvénient de ce modèle est que les restrictions et le caractère fermé des premières versions de la licence ont tendance à inhiber la revue par les pairs et la participation aux premiers stades du cycle du produit, là où précisément ils sont le plus requis.

## 9.6 Libérer le logiciel, vendre la marque

C'est un modèle économique encore à l'état de théorie. On y ouvre le source d'une gamme de logiciels, on met en place une batterie de tests ou un ensemble de critères de compatibilité, et on vend aux utilisateurs une marque certifiant que leur implémentation de ces techniques est compatible avec tous ceux qui font état de cette même marque.

(C'est la manière dont la société *Sun Microsystems* devrait gérer *Java* et *Jini*.)

## 9.7 Libérer le logiciel, vendre le contenu

Encore un modèle économique théorique. Imaginez un service d'inscription dans le style d'un centre d'informations. La valeur ne réside ni dans le logiciel client ni dans le serveur, mais dans la fourniture d'une information fiable et objective. Ainsi, on libère le logiciel et on vend des inscriptions au contenu. Au fur et à mesure que des *hackers* assurent le portage vers de nouvelles plates-formes et améliorent le logiciel de diverses manières, votre marché se développera en conséquence.

(C'est pourquoi la société *AOL* devrait libérer le source de son logiciel client.)

## 10 Quand faut-il ouvrir, quand faut-il fermer ?

Après avoir dressé un bilan des modèles économiques qui financent le développement des logiciels à sources ouverts, nous pouvons à présent nous poser la question plus générale de savoir quand le fait d'être à sources ouverts ou à sources fermés a une utilité économique. En premier lieu, nous devons découvrir les bénéfices que l'on peut retirer de chacune de ces stratégies.

### 10.1 Quels sont les bénéfices ?

L'approche sources fermés permet de faire du bénéfice grâce au secret de votre code ; d'un autre côté, cela vous prive de la possibilité de bénéficier d'un avis vraiment indépendant de vos pairs sur vos sources. L'approche sources ouverts permet d'avoir l'avis de vos pairs, mais ne vous permet pas de faire du bénéfice grâce au secret de votre code.

Le bénéfice que l'on obtient en gardant son source secret est facilement compréhensible ; traditionnellement, les modèles de l'industrie du logiciel sont construits autour. Encore récemment, les avantages que l'on pouvait retirer des critiques de ses pairs n'étaient pas bien compris. Le système d'exploitation *Linux* nous a pourtant enseigné une leçon que nous aurions dû apprendre depuis plusieurs années à partir de l'histoire des logiciels qui sous-tendent le coeur de l'*Internet* et les autres branches de l'ingénierie — et c'est que la critique de ses pairs est la seule méthode robuste pour obtenir une fiabilité et une qualité élevées.

Dans un marché concurrentiel, les clients qui recherchent une fiabilité et une qualité élevées choisiront de préférence les producteurs de logiciels qui ont choisi d'ouvrir leurs sources et qui ont trouvé comment se financer grâce aux services, à la valeur ajoutée, et aux marchés annexes associés à leur logiciel. Ce phénomène est à la base de l'étonnant succès de *Linux*, qui, parti de rien en 1996, est passé à 17 % du marché des serveurs à la fin de 1998 et semble bien parti pour dominer le marché d'ici deux ans (début 1999, *IDC* a annoncé que, d'après ses prévisions, *Linux* devrait progresser plus vite que l'ensemble de tous les autres systèmes d'exploitation jusqu'en 2003).

Un avantage tout aussi important des logiciels à sources ouverts est qu'ils peuvent être vus comme un moyen de diffuser des normes ouvertes et de construire des marchés autour de celles-ci. La spectaculaire progression de l'*Internet* est surtout due au fait que personne ne « possède » le protocole *TCP/IP* ; personne ne détient de droits sur les protocoles qui constituent la base de l'*Internet*..

Les effets réseau qui ont induit les succès de *TCP/IP* et de *Linux* apparaissent de façon évidente. D'une part, ils empêchent totalement l'apparition de problèmes de confiance mutuelle, et d'autre part ils offrent une certaine symétrie — les acteurs potentiels d'une infrastructure commune placeront d'autant plus naturellement leur confiance dans celle-ci s'ils peuvent voir quels en sont ses mécanismes internes, et ils préféreront une infrastructure dans laquelle tous les partis ont un rôle symétrique à une infrastructure où un seul parti possède le privilège d'en extraire du bénéfice ou d'exercer un contrôle.

Il n'est, cependant, pas vraiment nécessaire d'admettre l'existence des effets réseau pour comprendre que la symétrie est une chose importante pour les utilisateurs de logiciels. En effet, aucun de ces utilisateurs ne voudraient raisonnablement choisir de s'enfermer lui-même dans un monopole contrôlé par son fournisseur en devenant dépendant de sources fermés s'il existe une quelconque solution à sources ouverts de qualité raisonnable. Cet argument est d'autant plus fort que le logiciel considéré est crucial dans le travail de l'utilisateur — plus le logiciel est vital, moins le client peut tolérer de le voir contrôlé par un parti extérieur.

Pour finir, un des importants avantages pour le consommateur des logiciels à sources ouverts, relié à la confiance mutuelle, est l'assurance d'un suivi futur de ce logiciel, le rendant *résistant au temps*. Si les sources sont ouverts, le client a certains recours si le vendeur met la clef sous la porte. Cela peut-être particulièrement

important pour le modèle du « gel des gadgets », car le matériel informatique tend à avoir un court cycle de vie, mais l'effet est plus général et augmente ainsi la valeur des logiciels à sources ouverts.

## 10.2 Comment interagissent-ils ?

Lorsque le bénéfice que l'on fait sur le secret des sources est plus élevé que celui qu'on dégagerait sur des sources ouverts, il est logique, économiquement parlant, de fermer ses sources. Lorsque le bénéfice des logiciels à sources ouverts est plus élevé que celui basé sur le secret des sources, il est logique d'ouvrir ses sources.

En elle-même, cette observation est triviale. Elle devient plus complexe lorsqu'on remarque que le bénéfice d'un logiciel à sources ouverts est plus difficile à mesurer et à prédire que celui des logiciels à sources fermés — et que ce bénéfice est plus souvent outrageusement dévalué que surestimé. En effet, jusqu'à ce que le courant de pensée du monde des affaires revoie ses prémisses en suivant l'exemple de l'ouverture des sources de *Mozilla* au début de l'année 1998, les bénéfices des logiciels à sources ouverts étaient faussement mais très communément supposés comme nul.

Mais comment peut-on évaluer le bénéfice que l'on peut retirer des logiciels à sources ouverts ? C'est, en général, une question difficile, mais nous pouvons l'aborder comme n'importe quel autre problème prévisible. On peut partir des cas observés où l'approche sources ouverts a réussi ou échoué. Nous pouvons essayer de généraliser ces exemples en un modèle qui donne, au moins, une intuition qualitative quant aux contextes dans lesquels l'ouverture des sources procure un net gain à l'investisseur ou à l'entreprise visant à maximiser les retours sur capital. Nous pourrions alors revenir aux cas réels et essayer de raffiner un peu le modèle.

D'après l'analyse présentée dans 16 (), nous pouvons supposer que les logiciels à sources ouverts permettent de réaliser un bénéfice important lorsque (a) la fiabilité/stabilité/pérenité sont critiques, et (b) la qualité de la conception et de l'implémentation sont difficilement vérifiables par un autre moyen que l'analyse critique des pairs (le second critère se retrouve, dans la pratique, pour les logiciels complexes).

Le désir naturel d'un consommateur d'éviter d'être enfermé dans un monopole dirigé par son fournisseur augmentera d'autant plus son intérêt pour les logiciels à sources ouverts (et, donc, la compétitivité des fournisseurs qui ouvrent leurs sources) que le logiciel est important pour lui. Ainsi, un autre critère, (c) joue en faveur des logiciels à sources ouverts lorsque le logiciel est crucial pour l'activité du consommateur (comme, par exemple, dans beaucoup d'entreprises, le département comptable).

Comme pour l'étude de cas, nous avons précédemment observé que les logiciels à sources ouverts créaient un climat de confiance mutuelle et une symétrie parmi les acteurs, ce qui, à long terme, tend à attirer plus de consommateurs et à dépasser les stratégies à sources fermés ; et il est souvent meilleur d'avoir une petite part d'un marché qui croît rapidement qu'une grosse part d'un marché fermé qui stagne. Par conséquent, en ce qui concerne les stratégies de logiciels, les logiciels à sources ouverts qui jouent sur la diffusion à outrance ont plus de chance d'atteindre un bénéfice à long terme qu'une stratégie à sources fermés qui joue sur la location d'un droit de propriété intellectuelle.

En fait, la capacité des consommateurs potentiels à raisonner sur les conséquences futures des stratégies des vendeurs et leur réticence à accepter le monopole d'un fournisseur induisent une forte contrainte ; bien que celle-ci ne soit pas accablante, vous pouvez choisir soit une stratégie de diffusion sources ouverts, soit une stratégie de revenu immédiat source fermés — mais pas les deux (des analogies à ce principe sont visibles un peu partout, par exemple dans les marchés électroniques où les clients refusent souvent d'acheter tout à la même source). On peut voir les choses de façon moins négative ; là où les effets réseau (sous forme de rétro-actions positives) dominant, les logiciels à sources ouverts sont probablement le bon choix.

On peut ajouter à cela que les logiciels à sources ouverts semblent être doués pour générer un plus grand retour de bénéfices que les logiciels à sources fermés dans des logiciels qui (d) établissent ou mettent en oeuvre des opérations informatiques courantes et des infrastructures de communication.

Enfin, on peut remarquer que les fournisseurs d'un service unique, ou même de services très différents les

uns des autres, ont toutes les raisons de craindre que leurs concurrents copient leurs méthodes, au contraire des vendeurs de services, dont les algorithmes sont connus et compris de tous. Par conséquent, les logiciels à sources ouverts sont plus probablement à même de l'emporter lorsque (e) les algorithmes clés (ou leurs équivalents fonctionnels) font partie de la base de connaissance commune des informaticiens.

Les logiciels qui sous-tendent l'*Internet*, *Apache*, et l'implémentation de l'interface de programmation (*API ANSI Unix* de *Linux*) sont les plus importants exemples de ces cinq critères. Le chemin vers les logiciels à sources ouverts dans l'évolution de tels marchés est bien illustré par une re-convergence des protocoles réseau sur *TCP/IP* au milieu des années 1990, après cinquante ans de tentatives ratées pour construire un empire sur des protocoles fermés tels que *DECNET*, *XNS*, *IPX*, et compagnie.

D'un autre côté, les logiciels à sources ouverts semblent être inefficaces pour des entreprises qui ont pour unique produit une technique logicielle originale (remplissant exactement le critère (e)) et qui est (a) relativement tolérant aux fautes, qui peut (b) facilement être vérifié par d'autres moyens que l'analyse critique des pairs, qui n'est pas (c) crucial au bon fonctionnement de l'entreprise, et qui ne verra pas sa valeur substantiellement augmentée par (d) les effets réseau ou la diffusion à outrance.

Pour donner un exemple de ce cas extrême, au début de l'année 1999 une entreprise qui écrivait des logiciels pour optimiser la longueur des planches extraites de coupes des troncs pour des scieries m'a demandé « Devrions-nous passer en sources ouverts? ». Ma conclusion fut « Non ». Le seul critère qui était pleinement rempli était le (c) ; mais, à la rigueur, un opérateur expérimenté pouvait optimiser lui-même la découpe.

Il est important de remarquer que lorsqu'un produit particulier ou une technique spécifique résident sur ces considérations, les métriques peuvent évoluer à travers le temps, comme nous le verrons dans les cas que nous allons étudier plus tard.

En résumé, les points suivants poussent à ouvrir les sources de votre logiciel :

- (a) lorsque la fiabilité/stabilité/pérenité du logiciel sont critiques
- (b) lorsque la qualité de la conception et de l'implémentation du logiciel sont difficilement vérifiables par un autre moyen que l'analyse critique de ses pairs
- (c) lorsque le logiciel est indispensable à l'activité du client
- (d) lorsque le logiciel établit ou met en oeuvre une infrastructure banale, d'informatique ou de communications
- (e) lorsque les algorithmes clés du logiciel (ou leurs équivalents fonctionnels) font partie de la base de connaissances communes des informaticiens

### 10.3 *Doom* : une étude de cas

L'histoire du record des ventes d'*ID software*, *Doom*, illustre les façons dont la pression du marché et l'évolution d'un produit peut modifier de façon conséquente l'importance des bénéfices entre un logiciel à sources fermés et un logiciel à sources ouverts.

Lorsque la première version de *Doom* est sortie fin 1993, c'était le premier et unique jeu à un joueur avec une animation temps réel (l'antithèse du critère (e)). Ce n'était pas seulement l'impact visuel de la technique qui

étonnait, mais surtout le fait que pendant de nombreux mois, personne n'est arrivé à comprendre comment cela avait été réalisé sur les microprocesseurs sous-puissants de l'époque. Ce secret valait bien le prix du jeu. De plus, le bénéfice potentiel que l'on pourrait en retirer en passant en sources ouverts était faible. En tant que jeu à un joueur, le logiciel (a) pouvait se permettre d'avoir quelques bogues, (b) n'était pas extrêmement difficile à contrôler, (c) ne prenait pas une place cruciale dans l'activité professionnelle du client, (d) ne bénéficiait pas des effets réseau. Il était donc naturel pour *Doom* d'être en sources fermés.

Cependant, le monopole de *Doom* ne fut pas éternel. Des concurrents inventèrent des techniques d'animations similaires, et d'autres jeux semblables, comme *Duke Nukem*, sont apparus. Au moment où ces jeux commencèrent à grignoter des parts de marché à *Doom*, le bénéfice que l'on pouvait tirer du secret des algorithmes de représentation en vue subjective a disparu.

D'autre part, les efforts pour diffuser ce partage firent apparaître de nouveaux défis techniques — une meilleure fiabilité, plus de fonctionnalités de jeu, un plus grand nombre d'utilisateurs et de plates-formes. Avec l'avènement des jeux de « combat à mort » en multi-joueurs et des services de jeu de *Doom*, le marché commença à ressentir de manière substantielle les effets réseau. Tout cela demandait des heures de programmation qu'*ID* aurait préféré passer sur la version suivante.

Toutes ces tendances ont augmenté le bénéfice que l'on pouvait retirer d'une ouverture des sources. À un certain moment, la situation a rendu économiquement acceptable l'ouverture des sources de *Doom* par *ID* et le fait de faire du bénéfice sur les marchés annexes tels que les recueils de scénarios. Peu après, c'est effectivement ce qui s'est passé. La totalité des sources de *Doom* furent mis en libre accès fin 1997.

#### 10.4 Savoir quand lâcher emprise

*Doom* est un cas intéressant car il n'est ni un système d'exploitation, ni un logiciel de communication réseau ; il est donc très éloigné des exemples habituels des logiciels à sources ouverts. En effet, le cycle de vie de *Doom*, achevé par le passage en sources ouverts, peut illustrer de manière typique le cas des logiciels d'application dans l'écologie du code actuellement ; où les communications et les calculs distribués créent à la fois des problèmes de robustesse, de fiabilité, de changement d'échelle que seule la revue des pairs permet de résoudre, et franchissent fréquemment les frontières séparant les environnements techniques et les acteurs en compétition (avec tous les problèmes de confiance et de symétrie que cela implique).

*Doom* a évolué d'un jeu à un joueur à un jeu multi-joueurs. De plus en plus, les effets du réseau ont remplacé la puissance de calcul. Des tendances similaires sont visibles, même dans les marchés les plus importants, comme les *ERP*, qui ne se sont jamais autant occupé de réseau avec des fournisseurs et des clients — et, bien sûr, il y a toute la structure implicite du *Web*. On peut déduire de tout cela qu'à peu près partout le bénéfice que l'on peut retirer des logiciels à sources ouverts est progressivement en train de croître.

Si la tendance se maintient, le défi principal du siècle prochain en conception des logiciels et en gestion des ressources sera de savoir quand il faut ouvrir — quand faut-il garder ses lignes de code secrètes, et quand faut-il ouvrir ses sources afin de mieux profiter de l'analyse critique de ses pairs et d'augmenter ses bénéfices sur les services et les marchés annexes.

Il existe une très forte motivation économique à ne pas manquer le moment où le passage en sources ouverts est rentable. Au-delà de ce moment, il existe un sérieux risque si vous tardez trop — un concurrent peut vous couper l'herbe sous le pied et ouvrir les sources de son logiciel dans le même secteur d'activité que vous.

La raison pour laquelle il ne faut pas manquer ce coche est que les utilisateurs et les talents disponibles pour participer aux logiciels à sources ouverts dans une catégorie donnée sont limités, et le recrutement a tendance à perdurer. Si deux producteurs sont le premier et le second à entrer dans une compétition fondée sur un marché à sources ouverts dans des secteurs similaires, le premier va probablement attirer la plupart des utilisateurs et les plus motivés des co-développeurs ; le second devra se contenter des restes. Le recrutement tend à perdurer, car les utilisateurs ont assimilé le logiciel et les développeurs investi du temps

dans le programme lui-même.

## 11 L'écologie des marchés des logiciels à sources ouverts

La communauté du source ouvert s'est organisée d'une manière qui tend à amplifier les effets de productivité du source ouvert. Dans le monde *Linux*, en particulier, c'est un fait d'une grande signification économique qu'on dénombre de nombreux distributeurs de *Linux*, formant une entité bien distincte des développeurs.

Les développeurs écrivent du code et le rendent disponibles sur l'*Internet*. Chaque distributeur sélectionne un sous-ensemble du code disponible, l'intègre, l'empaquette et le revend sous sa griffe aux consommateurs. Les utilisateurs choisissent parmi plusieurs distributions, et ont toujours la possibilité de compléter une distribution en téléchargeant directement du code depuis les sites des développeurs.

Cette séparation claire a pour effet de créer un marché interne très fluide, encourageant les améliorations. Les développeurs sont en concurrence les uns avec les autres et se disputent l'attention des distributeurs comme des utilisateurs, jugeant la qualité de leurs logiciels. Les distributeurs se disputent l'argent des utilisateurs, jugeant la pertinence de leurs politiques de sélection, et sur la valeur ajoutée qu'ils apportent aux logiciels.

Un effet visible de la structure interne de ce marché est qu'aucun noeud du réseau n'est indispensable. Les développeurs peuvent cesser le travail ; même si personne ne reprend directement leur code, la concurrence en vue de l'attention des utilisateurs aura tendance à engendrer rapidement des solutions de remplacement remplissant la même fonction. Les distributeurs peuvent faire banqueroute sans que cela n'endommage ou ne compromette le pot commun de sources ouverts. Dans son ensemble, l'écologie répond plus rapidement aux exigences du marché, et elle est davantage capable de résister aux chocs et de se renouveler en permanence que tout fabricant jouissant du monopole sur un système d'exploitation à sources fermés.

Un autre effet important est de diminuer les surcoûts et d'accroître l'efficacité à travers la spécialisation. Les développeurs ne subissent pas les pressions qui compromettent couramment les projets à sources fermés conventionnels et les transforment en pièges sans fin — pas de listes de fonctionnalités inutiles et distrayant leur attention de l'essentiel exigées par le département de mercatique, pas d'ordres de la part des supérieurs d'utiliser des langages de programmation ou des environnements de programmation dépassés, pas besoin de réinventer la roue d'une manière encore nouvelle et incompatible au nom de la différenciation du produit ou de la protection de la propriété intellectuelle, et (avant tout) *pas de dates butoir*. Pas de course de vitesse pour sortir une version 1.0 avant qu'elle ne soit correcte — ce qui (comme DeMarco et Lister l'ont remarqué au cours de leur discussion traitant du style de gestion de projet à la « réveille-moi quand ce sera terminé » dans 16 ()) conduit généralement, non seulement à une meilleure qualité, mais en réalité à la meilleure vitesse de livraison d'un résultat vraiment opérationnel qu'on connaisse.

Les distributeurs, d'un autre côté, se spécialisent dans les choses que les distributeurs font le mieux. Libérés du besoin de financer des développements logiciels massifs en permanence pour rester en lice, ils peuvent se concentrer sur l'intégration du système, l'empaquetage, sur des garanties de qualité, et sur le service.

Les distributeurs et les développeurs restent honnêtes sous l'influence des commentaires incessants et de la surveillance de leurs utilisateurs, qui font partie intégrante de la méthode du source ouvert.

## 12 Composer avec la réussite

La tragédie des communs n'est peut-être pas applicable au développement à source ouvert tel qu'il se produit de nos jours, mais cela ne signifie pas qu'il n'y a aucune raison de s'interroger sur le fait que l'inertie acquise désormais par la communauté du code source ouvert suffira. Les acteurs clef quitteront-ils la voie de la coopération quand les enchères auront atteint une certaine somme ?

On peut poser cette question à plusieurs niveaux. Notre contre-argument de la « comédie des communs » repose sur l'argument qu'il est difficile de chiffrer la valeur de contributions individuelles au pot commun du source ouvert. Mais un tel argument perd beaucoup de sa force pour des sociétés (comme, disons, des distributeurs de *Linux*) qui disposent déjà d'un flux de revenus associé au code source ouvert. Leur contribution est déjà quotidiennement chiffrée. Leur coopération actuelle est-elle une situation stable?

L'examen de cette question nous mènera à certaines réflexions intéressantes sur l'économie des logiciels à sources ouverts dans le monde d'aujourd'hui — et sur ce que l'idée d'une industrie véritablement fondée sur le service implique pour l'industrie du logiciel à l'avenir.

Au niveau pratique, si on l'applique à la communauté du source ouvert telle qu'elle existe aujourd'hui, cette question prend en général l'une des deux formes suivantes. Un : *Linux* se fragmentera-t-il? Deux : à l'inverse, *Linux* se développera-t-il au point d'écraser tout le reste et jouer un rôle dominant, de quasi-monopole?

L'analogie historique à laquelle de nombreuses personnes font appel quand elles suggèrent que *Linux* se fragmentera est le comportement des fabricants d'*Unix* propriétaires dans les années 1980. Malgré d'interminables discussions sur l'ouverture des normes, malgré de nombreuses alliances et consortiums et accords, les *Unix* propriétaires se sont écroulés. Le désir des fabricants de différencier leurs produits en ajoutant et en modifiant des fonctionnalités du système a été plus fort que l'intérêt qu'ils voyaient à augmenter la taille totale du marché *Unix* en maintenant la compatibilité (et par conséquent, en abaissant à la fois les coûts d'entrée pour les développeurs indépendants de logiciels et le coût total d'acquisition de leurs clients).

Cela a très peu de chances de se produire avec *Linux*, pour la simple raison que tous les distributeurs sont contraints d'opérer à partir d'une base commune de code source ouvert. Il n'est pas vraiment possible, pour aucun d'entre eux, de faire un écart, car les licences sous lesquelles le code de *Linux* est développé requiert effectivement de leur part qu'ils partagent le code avec tous. Dès qu'un des distributeurs développe une fonctionnalité, tous ses concurrents ont la possibilité de la cloner.

Puisque tout le monde comprend bien cela, personne ne pense à manoeuvrer d'une des manières qui ont conduit à la fragmentation des *Unix* propriétaires. Au lieu de cela, les distributeurs de *Linux* sont forcés de se mesurer les uns aux autres de manières qui *profitent* au consommateur et au marché en général. C'est-à-dire qu'ils peuvent se faire concurrence sur le service, l'assistance technique, et les paris de conception sur les interfaces qui faciliteront le plus l'installation et l'utilisation.

Le fait qu'ils utilisent une source commune interdit *a priori* toute monopolisation. Quand des gens de *Linux* s'inquiètent de cela, c'est souvent le nom « *Red Hat* » qui est cité, en tant que distributeur le plus important et à la réussite la plus éclatante (bénéficiant d'une part de marché estimée à 90 % aux États-Unis d'Amérique). Mais il est bien connu maintenant qu'en mai 1999, quelques jours après l'annonce de la version 6.0 de la distribution de *Red Hat*, attendue depuis longtemps — et avant même que les *CD-ROM* de *Red Hat* ne soient distribués — un éditeur de livres et plusieurs autres distributeurs de *CD-ROM* faisaient de la publicité pour des images *CD-ROM* de cette distribution, obtenues sur le site *FTP* public de *Red Hat*, qu'ils proposaient à un tarif inférieur à celui de la version officielle de *Red Hat*.

Les gens de *Red Hat* n'en ont pas pris ombrage, car les fondateurs de cette société comprennent très clairement qu'ils ne possèdent pas, et ne peuvent pas posséder, les bits de leur produit ; cela leur est interdit par les normes sociales de la communauté *Linux*. Pour faire suite au bon mot de John Gilmore selon lequel l'*Internet* interprète la censure comme un dommage et la contourne en modifiant ses routes, on a dit fort à propos que la communauté des *hackers* responsables de *Linux* interprète les tentatives de prise de contrôle comme un dommage et les contourne tout autant. Si la société *Red Hat* avait protesté contre la pré-publication de son dernier produit, cela aurait sérieusement compromis sa capacité à réunir de nouveau, sous son égide, une coopération de la part de la communauté des développeurs.

Ce qui compte peut-être plus encore à présent, c'est le fait que les licences de logiciels qui expriment ces normes communautaires sous forme légale liant les parties interdisent expressément à la société *Red Hat* de monopoliser les sources du code sur lequel est fondé leur produit. Tout ce qu'ils peuvent vendre, c'est une

relation reposant sur leur marque, sur leur service, sur leur assistance technique, à des clients qui choisissent librement de payer pour ces derniers. Ce n'est pas dans un tel contexte que la possibilité d'un monopole et d'un prédateur est le plus à craindre.

## 13 Recherche et développement ouverts et réinvention du mécénat

Il existe un autre aspect qui est modifié par l'infusion de l'argent du monde traditionnel dans le monde du source ouvert. Les vedettes de la communauté se rendent de plus en plus compte qu'elles peuvent gagner leur vie en faisant ce qu'elles aiment et veulent faire, au lieu de traiter le source ouvert comme un loisir qu'elles financent grâce à un autre emploi. Des sociétés comme *Red Hat*, *O'Reilly & Associates*, et *VA Linux Systems* sont en train de bâtir ce qui constitue des moyens de recherche semi-indépendants grâce à des politiques de constitution et de conservation de réservoirs stables de talents en matière de source ouvert.

Cela n'a de sens économiquement que si les frais de maintenance de tels laboratoires sont facilement compensés par les gains estimés suite à l'expansion plus rapide du marché de la société. La société *O'Reilly* peut se permettre de payer les principaux auteurs de *Perl* et d'*Apache* afin qu'ils continuent leur travail car elle s'attend à ce que ses efforts lui permettent de vendre un plus grand nombre de livres traitant de *Perl* et d'*Apache*. La société *VA Linux Systems* peut financer sa branche de laboratoires car l'amélioration de *Linux* augmente en flèche la valeur d'utilisation des stations de travail et des serveurs qu'elle vend. Et la société *Red Hat* finance des laboratoires de développement avancés *Red Hat Advanced Development Labs* dans le but d'accroître la valeur de son offre *Linux* et d'attirer plus de consommateurs.

Aux stratèges issus des secteurs traditionnels de l'industrie du logiciel, campant sur des positions estimant que les brevets, les secrets industriels et la propriété intellectuelle constituent les bijoux de famille d'une société, un tel comportement peut (en dépit de son effet positif sur la croissance du marché) sembler inexplicable. Pourquoi financer une recherche que chacun de vos concurrents est (par définition) libre de s'appropriier à coût nul ?

On dénombre deux raisons contrôlant ce choix. La première est que tant que ces sociétés tiennent le haut du pavé dans leurs niches de marché, elles peuvent s'attendre à capturer la part du lion correspondante des retours de la R&D ouverte. C'est loin d'être une idée neuve que d'utiliser de la R&D afin de se garantir des profits futurs ; ce qui est intéressant, c'est le calcul implicite que les gains futurs estimés sont suffisamment importants pour que ces sociétés puissent tolérer des passagers clandestins.

Même si cette analyse sur les gains attendus à l'avenir est nécessaire dans un monde de capitalistes purs et durs, les yeux fixés sur les bénéfices, ce n'est pas le mode d'explication le plus intéressant qui est avancé pour justifier l'embauche de ces vedettes, car les sociétés concernées proposent une raison plus floue. Elles vous répondront, si vous leur posez la question, qu'elles se contentent de faire ce qu'il faut faire au sens de la communauté d'où elles proviennent. Votre dévoué auteur connaît suffisamment bien les responsables des trois sociétés cités plus haut pour attester qu'on ne peut pas traiter ces protestations comme une manifestation de la langue de bois. En réalité, j'ai personnellement été engagé par *VA Linux Systems* fin 1998 dans le but explicite d'être disponible afin de les conseiller sur « ce qu'il faut faire », et je les ai trouvés très réceptifs et attentifs à mes conseils.

Un économiste a le droit de s'enquérir de l'avantage que je leur conférais, agissant ainsi. Si on accepte que traiter de faire « ce qu'il faut faire » a un sens réel, il nous faut ensuite nous intéresser à l'intérêt qu'a une société de faire « ce qu'il faut faire ». Et la réponse n'est pas non plus, en elle-même, surprenant ou difficile à vérifier si on pose les bonnes questions. Comme c'est le cas avec les comportements d'apparence altruiste dans les autres industries, ce que ces sociétés achètent, c'est la bonne volonté.

Travailler pour gagner de la bonne volonté, et la mettre en valeur comme un avantage présageant de futurs gains en parts de marché, n'est pas non plus une idée neuve. Ce qui est intéressant ici, c'est la haute prise en considération avec laquelle ces sociétés estiment la bonne volonté, si on se fie à leur comportement. Elles font

visiblement état de leur souhait d'engager des acteurs talentueux et chers pour des projets qui n'engendrent pas de revenus directs, même pendant les phases les plus avides de bénéfices de la vie d'une entreprise. Et, en tout cas jusqu'à présent, le marché a véritablement récompensé ce type de comportement.

Les responsables de ces sociétés sont eux-mêmes très clairs sur les raisons pour lesquelles ils apprécient particulièrement la bonne volonté. Ils reposent en grande partie sur des volontaires parmi leur clientèle, à la fois pour développer des produits et en tant qu'arme de mercatique informelle. Ils entretiennent une relation intime avec leur clientèle, relation souvent renforcée par des liens et amitiés personnels entre individus, au sein de la société comme à l'extérieur.

Leurs observations renforcent une leçon qu'on a apprise plus tôt et qu'on avait découverte en suivant un autre raisonnement. La relation que les sociétés *Red Hat / VA / O'Reilly* entretiennent avec leurs clients ou développeurs n'est pas une relation typique des sociétés fabriquant des produits. Elle porte plutôt à des extrémités intéressantes des idées caractéristiques des sociétés de services, reposant grandement sur des connaissances. Si on cherche en dehors de l'industrie des nouvelles techniques, on peut retrouver de tels comportements dans, par exemple, les cabinets d'avocats, les pratiques médicales, et les universités.

On peut observer, en fait, que les sociétés à sources ouverts engagent des *hackers* vedette pour des raisons très similaires à celles pour lesquelles les universités engagent ou invitent des célébrités du monde académique<sup>3</sup>. Dans les deux cas, on reconnaît, tant dans le mécanisme que dans les effets, la pratique du mécénat aristocratique qui a financé la plupart des beaux arts jusqu'à la fin de la révolution industrielle — et c'est un point commun dont nombreux des acteurs concernés sont pleinement conscients.

## 14 Se rendre d'un point à un autre

Les mécanismes de marché pour le financement (et tirer du profit de!) du développement à source ouvert évoluent encore très rapidement. Les modèles économiques qu'on a présentés dans ce papier précèdent probablement de nombreuses autres inventions. Les investisseurs réfléchissent encore aux conséquences de la réinvention de l'industrie du logiciel, qui met désormais clairement l'accent sur le service plutôt que sur la propriété intellectuelle fermée, comme c'était auparavant le cas, et cette tendance s'accroîtra encore ces prochains temps.

Cette révolution conceptuelle aura un coût en termes de manque à gagner pour ceux qui investissent dans ces 5 % de l'industrie du logiciel financés par la valeur de vente; historiquement, les sociétés de services sont moins lucratives que les entreprises fabricant des biens (même si tout médecin ou avocat pourrait vous dire que les retours vers les praticiens sont souvent plus élevés). Ce manque à gagner sera toutefois largement compensé par les bénéfices sur les coûts, car les consommateurs de logiciels font des économies colossales et gagnent énormément en efficacité en employant des produits à source ouvert (on peut ici tracer un parallèle avec les effets que le déplacement réseau de téléphonie vocale classique vers l'*Internet* produit un peu partout).

La promesse de ces économies et de ce gain en efficacité crée une possibilité de marché vers laquelle les entrepreneurs et les capitaux à risque se dirigent afin de l'exploiter. Alors que ce papier était encore à l'état de brouillon, la société de capital-risque la plus prestigieuse de la vallée du silicium (*Silicon Valley*) a pris une longueur d'avance en pariant sur la première société spécialisée en assistance technique pour *Linux*, 24 h / 24, 7 j / 7. Les experts prédisent que plusieurs autres sociétés en relation avec *Linux* et le source ouvert verront le jour d'ici la fin de cette année — et qu'elles auront une réussite éclatante.

On peut vivre un autre développement intéressant dans les débuts de tentatives systématiques de créer des marchés de tâches dans le développement à sources ouverts. *SourceXchange* et *CoSource* représentent des manières légèrement différentes de tenter d'appliquer un modèle d'enchères à l'envers au financement de développements à source ouvert.

3. N.D.T. : Aux États-Unis d'Amérique, les universités sont pour la plupart privées et en concurrence.

Les tendances générales sont claires. On a mentionné plus haut les prévisions de la société *IDC*, selon lesquelles *Linux* progressera plus vite que tous les autres systèmes d'exploitation réunis jusqu'en 2003. *Apache* en est à 61 % de parts de marché et continue de croître régulièrement. Les utilisations d'*Internet* explosent, et des sondages comme celui du compteur des systèmes d'exploitation sur l'*Internet* (*Internet Operating System Counter*) montrent que *Linux*, tout comme d'autres systèmes d'exploitation à sources ouverts, occupent déjà en masse les hôtes sur l'*Internet*, et gagnent sans cesse du terrain face aux systèmes fermés. Le besoin d'exploiter sans cesse plus les infrastructures à sources ouverts de l'*Internet* ne conditionne pas uniquement la conception des autres logiciels, mais également des pratiques commerciales et des politiques d'achat et d'utilisation de logiciels de toutes les sociétés du monde. Ces tendances semblent prendre de l'ampleur.

## 15 Conclusion : vivre après la révolution

Quelle sera l'allure du monde du logiciel quand la transition du source ouvert aura pris place ?

Pour étudier cette question, il sera utile de classer les différents types de logiciels selon le degré de complétude, selon la description du service qu'ils offrent selon des normes techniques ouvertes, et selon leur corrélation avec l'instrumentation en laquelle a évolué le service sous-jacent.

Cet axe correspond assez bien à ce à quoi pensent ceux qui parlent d'« applications » (pas instrumentées du tout, normes techniques ouvertes faibles ou non existantes), d'« infrastructures » (services instrumentés, normes fortes), et de « logiciel médian » (*middleware*, instrumentés partiellement, normes techniques efficaces mais incomplètes). Les cas les plus flagrants, en 1999, seraient ceux d'un traitement de textes (application), une pile *TCP/IP* (infrastructure), et un moteur de bases de données (logiciel médian).

L'analyse des avantages effectuée plus haut suggère que l'infrastructure, les applications, et les logiciels médians évolueront de manières différentes et présenteront des équilibres mixtes différents entre les sources ouverts et fermés. On rappelle qu'il est également proposé que le fait que le source ouvert domine ou non dans un domaine spécifique de logiciels serait fonction de la présence ou non d'effets réseau substantiels dans ce domaine, des coûts des échecs, et dans quelle mesure le logiciel est un bien critique au fonctionnement de l'entreprise.

On peut se risquer à faire quelques prédictions si on applique ces heuristiques, non pas à des produits particuliers, mais à des pans entiers du marché du logiciel. Allons-y :

L'infrastructure (l'*Internet*, le *Web*, les systèmes d'exploitation, et les logiciels de communication au plus bas niveau, qui doivent faire le lien entre différentes parties séparées par des frontières) sera pratiquement entièrement constituée de logiciels à sources ouverts, maintenus de manière coopérative par des consortiums d'utilisateurs et par des sociétés de distribution et de services à buts lucratifs qui joueront le même rôle que *Red Hat* aujourd'hui.

Les applications, d'un autre côté, auront pour la plupart tendance à demeurer à sources fermés. Sous certaines circonstances, la valeur d'usage d'un algorithme ou d'une technique fermée sera suffisamment haute (et les coûts associés à sa non fiabilité seront suffisamment bas, et les risques associés au monopole d'un unique fournisseur suffisamment tolérables) pour que les consommateurs continuent à payer pour du logiciel à sources fermés. C'est ce qui va le plus probablement se produire dans les applications verticales indépendantes du marché, où les effets réseau sont faibles. C'est le cas de notre exemple de scierie, exposé plus haut ; parmi les secteurs en fort développement en 1999, les logiciels d'identification biométrique représentent un autre des candidats les plus probables.

Les logiciels médians (comme les bases de données, les outils de développement, ou les interfaces personnalisées de piles de protocoles d'applications) seront plus partagés. Les différentes catégories de logiciels médians passeront sous logiciels à sources ouverts ou demeureront sous logiciels à sources fermés selon le prix des échecs, des coûts plus élevés faisant pression pour une plus grande ouverture.

Pour compléter le tableau, il nous faut cependant remarquer que ni les « applications » ni les « logiciels médians » ne sont compartimentés en catégories vraiment stables. Plus haut, dans la section intitulée « Savoir à quel moment libérer », nous avons vu que les techniques mises en oeuvre pour chaque logiciel particulier semblent vivre un cycle naturel, où au début il est rationnel de fermer les sources, pour finalement trouver rationnel de les ouvrir. La même logique s'applique à des cas plus généraux.

Les applications ont tendance à devenir des logiciels médians au fur et à mesure que des techniques standardisées se développent et que des portions de services sont instrumentées (les bases de données, par exemple, sont devenues des logiciels médians après que le langage *SQL* ait découplé les interfaces des moteurs). Au fur et à mesure de l'instrumentation des logiciels médians, ces derniers auront à leur tour tendance à être qualifiés d'infrastructures à sources ouvertes — c'est une transition qu'on peut observer dès maintenant dans les systèmes d'exploitation.

Dans un avenir qui prend en compte la concurrence issue du monde des logiciels à sources ouvertes, on peut s'attendre à ce que la destinée finale de toute technique logicielle est de mourir ou d'être intégrée au sein de l'infrastructure ouverte. Même si cela représente une mauvaise nouvelle pour les entrepreneurs qui souhaiteraient collecter une rente à vie sur les logiciels fermés, cela suggère aussi que l'industrie du logiciel dans son ensemble *restera* le fait des entrepreneurs, que de nouvelles niches de haut niveau (applications) se mettront sans cesse en place et que les monopoles des *IP* fermées disposeront d'une espérance de vie limitée par le fait que les catégories de produits qu'ils embrassent seront progressivement qualifiées d'infrastructure.

Enfin, évidemment, cet équilibre sera formidable pour le consommateur de logiciels à l'origine de cette évolution. On aura de plus en plus de logiciels de haute qualité, disponibles en permanence, sur lesquels construire des logiciels plus fournis encore, au lieu de devoir les abandonner ou les enfermer dans le coffre-fort d'une société au fur et à mesure. Le chaudron magique de Ceridwen est, finalement, une métaphore trop faible — la nourriture doit être consommée ou pourrir, alors que les sources de logiciels peuvent potentiellement durer toujours. Le libre marché, dans son sens libéral le plus large, comprend *toutes* les activités non contraintes, qu'il s'agisse de commerce ou de cadeaux, et peut produire une richesse logicielle sans cesse renouvelée, à l'intention de tous.

## 16 Bibliographie et remerciements

[CatB] *La cathédrale et le bazar* <<http://www.linux-france.org/article/these/cathedrale-bazar/cathedrale-bazar.html>>

[HtN] *À la conquête de la noosphère* <<http://www.linux-france.org/article/these/noosphere/homesteading-fr.html>>

[DL] De Marco et Lister, *Peopleware: Productive Projects and Teams* (New York; Dorset House, 1987; ISBN 0-932633-43-9) (*Ressources humaines : projets et équipes productifs*).

[SH] Shawn Hargreaves a écrit une bonne analyse de l'applicabilité des méthodes à sources ouvertes aux jeux ; *Playing the Open Source Game* <<http://www.talula.demon.co.uk/games.html>> (*Jouer au jeu du source ouvert*).

Plusieurs discussions stimulantes avec David D. Friedman m'ont aidé à raffiner le modèle de la coopération à sources ouvertes en tant que « *communs inversés* ». Je suis également redevable à Marshall van Alstyne de m'avoir indiqué l'importance conceptuelle des informations portant sur des biens disputés. Ray Ontko, de l'*Indiana Group* (*groupe Indiana*), a critiqué mon travail de manière constructive. De nombreux membres des publics devant qui j'ai donné des conférences de juin 1998 à juin 1999 m'ont également assisté dans ma réflexion ; ils se reconnaîtront.

C'est une illustration supplémentaire du modèle des sources ouvertes que ce papier fut amélioré de manière substantielle par des courriers électroniques qui ont commenté cet article quelques jours après sa publication.

Lloyd Wood m'a fait remarqué l'importance du fait que le logiciel à sources ouverts est « résistant au temps » et Doug Dante m'a rappelé le modèle économique consistant à « Libérer l'avenir ». Une question posée par Adam Moorhouse m'a conduit à discuter des profits retirés de l'exclusivité. Lionel Oliviera Gresse m'a proposé un meilleur nom pour l'un des modèles économiques. Stephen Turnbull m'a fait rougir en signalant mon traitement superficiel des effets de passagers clandestins.

## 17 Annexe : Pourquoi fermer ses pilotes fait perdre de l'argent à un fabricant

Les fabricants de périphériques matériels (cartes *Ethernet*, contrôleurs de disques, cartes graphiques, etc.) ont, historiquement, rechigné à ouvrir leurs sources et diffuser leurs spécifications. La situation est en train de changer, grâce à des acteurs comme *Adaptec* et *Cyclades* qui commencent à donner systématiquement les spécifications et le code source des pilotes de leurs cartes. Néanmoins, la résistance demeure. Dans cette annexe, nous tenterons de dissiper plusieurs interprétations incorrectes de l'économie qui alimentent cette résistance.

Si vous vendez du matériel, vous pouvez craindre qu'ouvrir vos sources ne mette en lumière des détails importants sur le fonctionnement de celui-ci, détails que vos concurrents pourraient copier, obtenant ainsi un avantage déloyal sur vous. À l'époque où les cycles de produits prenaient de 3 à 5 ans, cet argument était recevable. De nos jours, le temps que les ingénieurs du concurrent devraient consacrer à copier et à comprendre votre pilote représente une portion substantielle du temps de cycle du produit, et ils ne consacreront *pas* ce temps à innover ou à se démarquer par leur propres produits. Le plagiat est un piège dans lequel vous *souhaitez* que vos concurrents tombent.)

Quel que soit votre choix, de tels détails ne restent plus longtemps dans l'ombre désormais. Les pilotes de périphériques ne ressemblent pas aux systèmes d'exploitation ou aux applications ; ils sont petits, faciles à désassembler, et faciles à cloner. C'est à la portée de programmeurs débutants, encore adolescents — et ces derniers ne s'en privent pas.

On dénombre littéralement des milliers de programmeurs pour *Linux* et *FreeBSD*, compétents et motivés par construire des pilotes pour une nouvelle carte. Pour de nombreux types de périphériques aux interfaces relativement simples et aux standards bien connus (comme des contrôleurs de disques et des cartes réseau), ces *hackers* impatientes peuvent souvent prototyper un pilote presque aussi vite que votre propre équipe, sans documentation et sans désassembler un pilote existant.

Même pour des pilotes plus malins, comme les cartes graphiques, vous ne pouvez pas grand-chose face à un programmeur astucieux armé d'un désassembleur. Les coûts sont faibles et les barrières juridiques bien poreuses ; *Linux* est un effort international et on trouvera toujours une juridiction où l'ingénierie à l'envers sera autorisée.

Si vous souhaitez des preuves concrètes de la véracité de toutes ces affirmations, jetez un coup d'oeil à la liste des périphériques reconnus par le noyau *Linux* ou dans les arborescences consacrées aux périphériques de sites comme *Metalab*, et observez la vitesse à laquelle de nouveaux pilotes viennent en renfort.

La morale ? Si vous gardez votre pilote secret, il séduira à court terme, mais c'est probablement un mauvais choix à long terme (et sans aucun doute si vous êtes en concurrence avec d'autres fabricants qui proposent déjà des pilotes au source ouvert). Mais s'il vous faut procéder ainsi, brûler le code dans la *ROM* de la carte, et publiez ensuite l'interface permettant d'y accéder. Soyez aussi ouvert que possible, afin de construire votre marché et de démontrer aux clients potentiels que vous êtes confiants dans votre capacité à tenir tête à vos concurrents en matière de performances de d'innovations là où cela compte.

Si vous restez fermé, vous réunirez le pire des deux mondes — vos secrets seront exposés, vous n'obtiendrez pas d'assistance gratuite au développement, et vous aurez gaspillé votre précieux temps à cloner vos concurrents.

Plus important, vous passez à côté de la possibilité d'encourager l'adoption rapide de votre matériel par un large public. Un marché étendu et avec de l'influence (ceux qui gèrent les serveurs qui font effectivement fonctionner la totalité de l'*Internet* et plus de 17 % des bases de données commerciales) décrira correctement votre société comme à côté de la plaque parce que vous n'avez pas compris tout cela. Et ils iront acheter leurs cartes chez quelqu'un qui aura compris.

## 18 Historique des versions

Le présent texte est l'adaptation en français de la \$Revision: 1.15 \$.

Les versions qui ne sont pas reprises dans la liste qui suit correspondent à des mises à jour mineurs du point de vue éditorial ou correction des fautes de frappe.

20 mai 1999, version 1.1 — brouillon.

18 juin 1999, version 1.2 — première version révisée privée.

24 juin 1999, version 1.5 — première version publique.

24 juin 1999, version 1.6 — mise à jour mineure ; précisions sur la définition de « *hacker* ».

24 juin 1999, version 1.7 — mise à jour mineure ; clarifié le critère (e).

24 juin 1999, version 1.9 — « résistant au temps », le modèle dit de « Libérer l'avenir », et une nouvelle section sur les profits qu'on peut retirer de l'exclusivité.

24 juin 1999, version 1.10 — trouvé un meilleur nom pour le modèle dit des « Lames de rasoir ».

25 juin 1999, version 1.13 — corrigé l'information parlant des 13 % des revenus de *Netscape* ; ajouté un meilleur traitement des effets de passagers clandestins, corrigé la liste des protocoles fermés.

25 juin 1999, version 1.14 — ajouté *e-smith, inc.*

9 juillet 1999, version 1.15 — nouvelle annexe sur les pilotes pour les matériels, et une meilleure explication des biens disputés que je dois à Rich Morin.