Fiche 132 :

La dynamique *open source*

1. Définition

Les notions de logiciel libre (*free software*) et de logiciel à source ouverte (*open source*) se doivent d'être distinguées.

Un logiciel libre (FSF, 2002c) est un logiciel qui est fourni avec l'autorisation pour quiconque de l'utiliser, de le copier, et de le distribuer, soit sous une forme conforme à l'original, soit avec des modifications, ou encore gratuitement ou contre un certain montant. Ceci signifie en particulier que son code source[1] doit être disponible. Le terme logiciel *open source* est utilisé par certaines personnes pour signifier plus ou moins la même chose que logiciel libre. En effet, le terme *free software* déplaît à certains, du fait que *free* signifie à la fois libre et gratuit. Or, un logiciel libre n'est pas toujours gratuit.

Par ailleurs, *open source* est parfois utilisé dans une acceptation plus large, s'écartant de l'idée de logiciel libre. Pour éviter le détournement du terme *open source*, une définition du concept a été réalisée dans le cadre de l'*Open Source Initiative* par des membres actifs et reconnus de la communauté du logiciel libre. Pour être considéré comme *open source*, un logiciel doit répondre à neuf critères (Perens, 2002) :

1. Redistribution du programme libre et gratuite.
2. Livraison du code source avec le programme.
3. Distribution des travaux dérivés dans les mêmes termes que la licence du logiciel d'origine.
4. Préservation de l'intégrité du code source de l'auteur.
5. Absence de discrimination envers des personnes ou des groupes.
6. Absence de discrimination envers des domaines d'activité.
7. Pas besoin de se conformer à des termes de licences complémentaires.
8. Pas de licence spécifique à un produit.
9. Pas de licence imposant des restrictions sur d'autres logiciels.

Dans le cadre de ce document, nous nous intéresserons également à des licences qui ne respectent pas ces conditions. Ces licences présentent néanmoins de l'intérêt, en ce sens qu'elle permettent, à des degrés divers, le partage du code source et, donc, la co-innovation. La notion de co-innovation se retrouve d'ailleurs dans le mouvement *open innovation*, inspiré par l'*open source* dans le domaine logiciel (Horwitch, 2000).

[1] Au départ, un programmeur écrit un programme dans un langage de programmation particulier. Cette forme du programme s'appelle le programme source, ou plus généralement, code source. Le code source doit ensuite être traduit en langage machine, de façon à obtenir un programme exécutable par l'ordinateur.

2. Fondements

2.1. Historique

L'histoire de l'*open source* a commencé il y a une dizaine d'années avec Richard Stallman (RMS) et la création de la *Free Software Foundation*. Depuis 1984, la FSF œuvre au développement d'un système d'exploitation à la mode Unix, libre et appelé GNU (GNU's Not Unix).

Richard Stallman s'inscrivait dans une certaine tradition. En 1969, les laboratoires d'AT&T (Raymond, 1998c) avait en effet créé Unix (Uniplexed Information and Computer Service), un système d'exploitation multi-tâche et multi-utilisateur réputé pour sa stabilité. L'AT&T Unix a été par la suite distribué cher et sans support mais avec le code source ! En 1977, l'université de Berkeley (Raymond, 1998c) avait réalisé un noyau public type Unix baptisé BSD. Distribué sous licence BSD, moins restrictive que la GPL, le noyau BSD donna naissance à plusieurs variantes propriétaires (BSD/OS) et libres (NetBSD, FreeBSD et OpenBSD), toujours utilisées aujourd'hui.

La FSF a commencé par développer les outils nécessaires à la création du noyau du système d'exploitation (éditeur Emacs, compilateur gcc,..). En 1991, le système GNU était près, sauf le noyau du système d'exploitation. Créé par Linux Torvald en réaction au système d'exploitation Minix (Yamagata, 1997), destiné à l'enseignement, le noyau Linux a été progressivement inclu au système GNU. Cela donna GNU/Linux. La FSF continue à travailler sur son propre noyau, appelé Hurd (Stallman, 1997) et construit sur un micro-noyau Mach, afin de constituer GNU/Hurd. La FSF estime que 28 % environ du code source d'une distribution GNU/Linux provient du logiciel GNU, contre 3 % pour la source de Linux. D'autres projets (les 69 % restants) sont progressivement venus se greffer autour de ces différentes contributions, avec le succès que l'on connaît aujourd'hui.

Combinés à la qualité des développements libres et aux mouvements à l'encontre des éditeurs propriétaires monopolistiques, l'essor de l'Internet courant des années nonante et la facilité d'échange qui en a résulté ont grandement facilité le développement des logiciels libres. En 2002, plusieurs dizaine de milliers de projets *open source* étaient en cours (plus de 35.000 rien que sur l'hébergeur en ligne SourceForge.Net).

2.2. Tendances

Nesbitt (1999) souligne le fait que l'ouverture des logiciels s'inscrit dans une tendance historique à long terme. En 1970, l'informatique était dominée par les standards propriétaires. Le caractère propriétaire affectait tout, depuis le système d'exploitation lui-même jusqu'à la façon dont le codage du texte était réalisé. Les différentes alternatives étaient toutes plus ou moins incompatibles. La plupart d'entre elles (RSTS, Domain, Prime OS,...) ont aujourd'hui disparu. Cette disparition fut en grande partie provoquée par la sortie de UNIX en 1975 et par une propagation rapide de standards ouverts tels que ASCII pour le codage binaire du texte ou TCP/IP comme protocole de communication.

L'histoire se reproduirait en micro-informatique, très tôt dominée par les produits propriétaires comme ceux de Microsoft. L'ouverture défendue par les supporters de Linux et apparentés constituerait une sorte de bégaiement de l'histoire.

A côté de ces tendances historiques, constatons également que le mouvement *open source* s'inscrit dans une double tradition (Raymond, 1998b) :

- La tradition académique, qui privilégie la publication, mais après révision par des pairs.
- La tradition libertaire, où le partage est reconnu comme une valeur commune et où les aspects mercantiles sont généralement rejetés.

Ces valeurs de partage et le bénévolat des milliers de programmeurs participant à des projets *open source* ne peut par ailleurs subsister que dans une société d'abondance, où programmer peut être à la fois une source de revenus et de plaisir (Raymond, 1998b).

Le mouvement est aujourd'hui progressivement adopté par des sociétés commerciales dont la rentabilité est la préoccupation première.

3. Motivations

Eric Raymond (Raymond, 1999a) envisage différents modèles pour expliquer la motivation des *hackers*[2] :

- Le modèle de « l'artisanat », qui associe la motivation du *hacker* au plaisir de l'artisan (« On ne devient pas compositeur si on n'aime pas la musique ») ;
- Le modèle du « jeu des réputations », qui explique la motivation par la réputation acquise aux yeux des pairs.

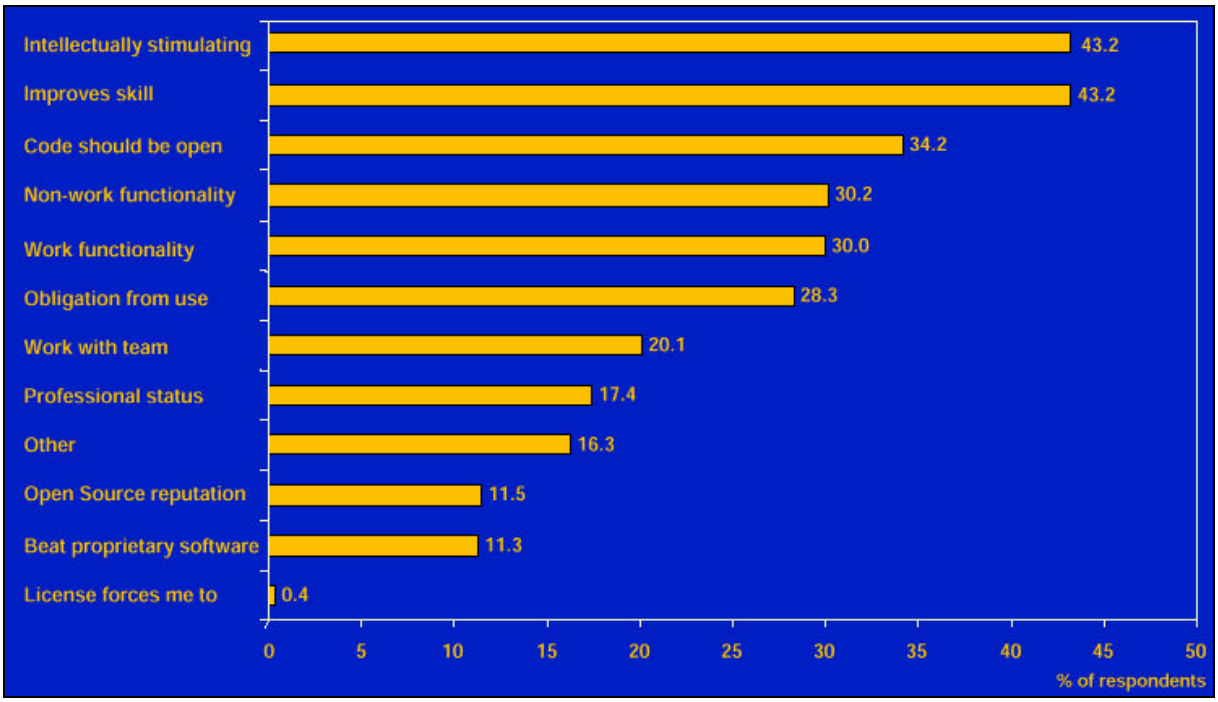
Ces deux modèles sont compatibles. Un travail bien fait est susceptible d'attirer l'attention de la communauté. L'attention peut conduire au prestige.

Le modèle du « jeu des réputations » s'inscrit dans le cadre d'une culture du don. Cette dernière est une réponse à une économie d'abondance, dans laquelle le statut vient du don plutôt que de la possession. Les échanges prennent la forme d'un commerce d'idées et de réputation.

La recherche de la réputation vise la satisfaction de l'ego, mais sans égocentrisme. L'humilité est une valeur importante, car les communautés *open source* fonctionnent suivant une méritocratie stricte. La dimension commerciale est absente, même si la réputation peut conduire à la création d'une société (exemple de Col Needham et de l'*Internet Movie Database* (Gosh, 1998b) ou l'obtention d'un emploi, comme Linus Torvald (Gosh, 1998a) chez Transmeta. La réputation conduit également à la constitution d'un réseau de confiance (Gosh, 1998a).

Boston Consulting Group (Lakhani, Wolf & Bates, 2002) a entrepris une étude systématique des motivations des programmeurs inscrits à SourceForge.Net, un *repository open source* hébergeant 33000 projets (janvier 2002).

[2] Dans la communauté du logiciel libre, le terme *hacker* désigne un développeur qui apporte des modifications ingénieuses à un programme (Smets et Facon, 1999).



Motivation des hackers, Boston Consulting Group, janvier 2002

L'étude révèle que les *hackers* recherchent d'abord la stimulation intellectuelle (créativité) et l'amélioration de leurs capacités. Par contre, peu contribuent pour vaincre les logiciels propriétaires. Les *hackers* peuvent être décomposés en quatre catégories :

- Les croyants, poussés par la conviction que le code devrait être ouvert ;
- Les professionnels, conduits par les besoins du métier et le statut professionnel ;
- Les hédonistes, conduits par des besoins privés et la recherche de stimulation intellectuelle ;
- Les experts, recherchant l'amélioration de leurs capacités.

L'étude révèle également une forte identification à la communauté.

4. Avantages et inconvénients

4.1. Avantages

- **Qualité** – La stabilité et les performances de logiciels *open source* tels que Linux ou FreeBSD ont souvent été saluées (Lang, 1997). Cette qualité est justifiée par la « loi de Linus » (Raymond, 1998a), qui peut s'énoncer par : « Etant donné suffisamment d'observateurs, tous les bogues sautent aux yeux ». Cette loi appelle néanmoins deux remarques :
  - La « loi de Linus » n'est d'application que lorsque la communauté de développeurs est suffisamment grande et réactive (Smets, 1998).
  - Les UNIX propriétaires (AIX, HP-UX, Irix, Solaris,...) présentent souvent des qualités techniques supérieures aux systèmes d'exploitation Microsoft Windows NT et dérivés (Kirch, 1999). Le caractère propriétaire d'un système d'exploitation n'est pas un gage fatal de non-qualité. L'observatoire de Netcraft (Netcraft, 2002) sur les sites les plus sollicités révèle par ailleurs des

performances honorables de la part des serveurs Microsoft Windows 2000 comparés à leurs homologues libres (Linux et FreeBSD).

- **Réactivité** – Les défenseurs du logiciel libre mettent en avant la réactivité des communautés de développeurs (Smets, 1998) et citent en exemple les cas de logiciels comme Apache ou Linux, dont les mises à jour apparaissent très vite. Cette affirmation appelle néanmoins deux remarques :
  - Dans le cas de Linux, si la réactivité est souvent excellente, elle n'est toutefois pas sensiblement supérieure à celle des développeurs de chez Microsoft. Par contre, l'ouverture du code empêche de laisser des bogues non corrigés pendant de longues périodes (Le Quérouzec, 2001).
  - La fréquence des mises à jour peut être pénalisante pour l'entreprise qui, pour des raisons de coûts, souhaite réduire la maintenance de son parc informatique au maximum.

- **Pérennité** – Continuer la maintenance d'un logiciel est un choix stratégique pour l'entreprise, pas forcément en phase avec les intérêts des consommateurs. C'est ainsi que le développement de produits tels que MS DOS, DR-DOS ou CP/M a été stoppé, au dépit des utilisateurs fidèles. L'existence d'une communauté de développeurs suffisamment grande et active garantit la pérennité des logiciels libres. C'est ainsi que des versions libres d'anciens OS comme MS-DOS ou CP/M ont été reprogrammées par des utilisateurs fidèles [Lang].

- **Coûts** – Une distribution Linux présente un coup d'achat très nettement inférieur à son équivalent propriétaire (Kirch, 1999). C'est une évidence. Le débat porte plutôt sur le TCO (*Total Cost of Ownership*) (Lang, 1997). Le débat est fortement marqué par la technique et sort du cadre de ce document.

- **Liberté** – Le consommateur n'est pas dépendant des choix stratégiques ou des limitations des sociétés informatiques commerciales (Browne, 1998). Chaque communauté de développeurs est libre de décider ce qu'il lui convient de faire, en toute indépendance.
  Le système d'exploitation IBM OS/2 a connu un rejet partiel en interne par peur du cannibalisme (Browne, 1998). Il n'a pas été porté par l'entreprise et a été balayé par les produits Microsoft. Dans un développement libre, les choix techniques prennent le pas sur les considérations commerciales. Le modèle décentralisé, qui privilégie la délégation, réduit également la portée de l'effet de « goulot d'étranglement » (Browne, 1998).

- **Concurrence** – Le logiciel libre prévient l'apparition de monopoles basés sur la fermeture du code source et des standards de communication. Il stimule non seulement la concurrence entre les entreprises, mais aussi la concurrence entre les états ! Il représente une opportunité de développement pour de nombreuses PME européennes, à l'instar de TCX ou Roxen en Suède.

4.2. Inconvénients

- **Protection de la propriété intellectuelle** – Deux philosophies s'affrontent sur ce point :

- L'industrie du logiciel en fait son principal grief à l'encontre du mouvement *open source*. Microsoft souligne la nécessité de protéger la propriété intellectuelle pour soutenir l'innovation ainsi que le caractère « viral » de la GPL (Microsoft, 2002a). Sun pointe du doigt les risques juridiques en matière de propriété intellectuelle (Gabriel, 2002).
- Les défenseurs de l'*open source* affirment que le brassage des idées (fertilisation croisée), bien plus que l'appropriation, favorise l'innovation. Ils sont également fortement opposés à l'idée d'instaurer des brevets logiciels [Free Patents, 2002 ; EuroLinux Alliance, 2002).

- **Finition** – Si les qualités techniques des logiciels libres ont été fréquemment saluées, il n'en est pas de même de leur interface. Celle-ci pêche souvent d'un manque de finition propre aux logiciels commerciaux (Yamagata, 1997). Ceci peut s'expliquer par les préoccupations des programmeurs, qui sont essentiellement techniques.

- **Risque de divergence** – Les orientations techniques sont laissées au choix de la communauté. En cas de différence de point de vue, des divergences peuvent apparaître (Raymond, 1998b). Souvent, la concurrence joue son rôle et seule une branche subsiste. Mais il n'en est pas toujours ainsi. Par exemple, le projet BSD s'est progressivement scindé en trois projets distincts : FreeBSD (la plus répandue et la plus conviviale, qui privilégie les performances), OpenBSD (qui privilégie la sécurité) et la branche initiale, NetBSD (qui privilégie l'adaptation aux matériels).
  Les scissions sont à priori plutôt rares. Primo, certains projets à succès deviennent des « tueurs de concurrence » (Raymond, 1998b). Se mesurer à la base déjà établie est une tâche trop dure qui rebute les plus motivés. Secundo, des propositions concurrentes peuvent évoluer en parallèle mais elles finissent généralement par s'éliminer et se fertiliser mutuellement. Les différentes versions de BSD sont plus une exception qu'une règle.

- **Image de marque** – Linux garde une image de produit « pas sérieux », maintenu par des « bitouilleurs[3] » pendant leurs loisirs. Comme auparavant avec IBM, celui qui aujourd'hui choisit Linux plutôt que Windows prend des risques par rapport à sa hiérarchie. Les sociétés commerciales telles que Red Hat ou Caldera y remédient en construisant la crédibilité commerciale des produits issus des développements libres.

- **Support** – L'accès au code source d'un logiciel n'est pas tout. Encore faut-il la documentation, l'aide et le support pour pouvoir l'exploiter. Le support technique pour les logiciels libres est un support vivant, basé sur les expériences pratiques des utilisateurs-développeurs. Néanmoins, ce support est morcelé, éparpillé à travers une constellation de sites Internet et de *newsgroups* (Browne, 1998). Outre la difficulté de la recherche, diverses barrières existent :

  - La manière de poser des questions est importante (Raymond, 2001). Un rituel doit être respecté et des précautions prises, sous peine de se faire « incendier ».
  - La langue véhiculaire est bien souvent l'anglais, malgré de réels efforts de traduction.

[3] Le terme « bitouilleur » est parfois utilisé comme traduction de *hacker*.

L'entreprise soucieuse de la qualité du support peut donc difficilement bénéficier d'un interlocuteur unique pour résoudre ses problèmes, excepté lorsqu'une entreprise commerciale assure la distribution du produit. Des projets comme *Linux Documentation Project* visent à combler ce manque de centralisation (Linux Documentation Project, 2002).

5. Communautés et propriété intellectuelle

Le principe de la communauté a été appliqué de diverses manières aux différentes étapes de la vie économique du logiciel. C'est ainsi que l'on trouve des communautés de développement, mais aussi des communautés de conception et des communautés de support.

5.1. Communautés de développement

La protection de la propriété intellectuelle peut s'exprimer de manières différentes suivant le modèle économique mis en œuvre. Trois tendances peuvent être distinguées : l'*open source* radical, l'*open source* modéré et le *shared source*.

Open source radical

La forme la plus radicale d'*open source* est défendue par la *Free Software Foundation* de Richard Stallman (RMS). RMS est le fondateur du projet GNU (GNU's Not UNIX) et créateur de la licence GPL (Licence Publique Générale). La GPL protège des logiciels libres célèbres tels que Linux ou GNOME. La GPL est une licence *copyleft*, c'est-à-dire gauche d'auteur. Le principe du *copyleft* consiste à utiliser le *copyright* pour garantir qu'aucune personne morale ou physique ne pourra s'approprier un code source écrit sous GPL. Un code GPL appartient en fait à son développeur, qui peut décider à tout moment de changer la licence sous laquelle seront diffusées les versions ultérieures. Néanmoins, un code GPL est définitivement mis à disposition de la communauté qui lui a donné naissance et, par extension, au monde entier.

La licence GPL est utilisée par des sociétés commerciales telles que TCX (serveur de base de données mySQL) ou TrollTech (bibliothèque graphique multi-plateforme Qt).

Près des deux tiers (65,05 %) des 19.115 projets répertoriés[4] au 20-03-2002 par FreshMeat.Net sont développés sous licence GPL. Les autres licences occupent des parts de marché très faibles (moins de 5 %, excepté pour la LGPL et la BSD).

Open source modéré

D'autres sociétés ont cherché à adoucir le caractère héréditaire de la licence GPL, de façon à préserver la propriété intellectuelle. La licence BSD, antérieure à la GPL, permettait déjà l'appropriation du code source et des modifications apportées.

Beaucoup ont choisi de créer leur propre licence, parfois incompatible avec la GPL, mais sont restés fidèles à l'esprit du logiciel libre. On peut citer comme grands noms Netscape ou IBM.

[4] Détail intéressant, chaque projet répertorié sur FreshMeat se voit associer trois valeurs : la cote (sur 10), la popularité (%) et le vitalité (%). Des classements sont réalisés sur base de ces trois critères (FreshMeat, 2002).

Netscape a choisi en 1998 de rendre public le code de son navigateur. Une licence particulière a été créée : la licence libre NPL (Netscape Public Licence), basée sur la MPL (Mozilla[5] Public Licence) et incompatible avec la GPL. La motivation était de garantir l'ouverture du code et son retour à la communauté tout en maintenant la possibilité pour l'éditeur de pouvoir l'incorporer dans ses développements propriétaires.

IBM a créé une division spéciale, baptisée Alphaworks. Cette dernière, basée dans la Sillicon Valley, a pour mission d'identifier des projets porteurs au sein d'IBM et ailleurs dans le monde, et de les valoriser dans le cadre de développements libres. Ces développements portent principalement sur des logiciels d'infrastructures tels qu'un parseur XML ou un compilateur Java (Jikes™) et ne menacent donc pas l'activité marchande d'IBM. Ces logiciels sont publiés sous la licence libre IBM Public Licence.

Sun Microsystems a pour sa part opté pour une licence communautaire, baptisée SCSL, pour Sun's Community Source Licence. Incompatible avec la GPL et considérée comme non-libre par la FSF, la SCSL réduit le caractère *open source* à une communauté constituée de partenaires académiques et industriels. Elle autorise en outre l'appropriation des modifications, à l'exception des corrections de bogues. Appliquée à Java en décembre 1998, la SCSL a par la suite été étendue à d'autres produits Sun, dont les processeurs SPARC.

L'exemple d'Aladdin Enterprises[6] mérite également d'être cité. La disponibilité du code source de son logiciel GhostScript est en effet retardée (Ghostscript, 2002). C'est ainsi que coexistent AFPL Ghostscript sous licence non libre AFPL (Aladdin Free Public Licence) et GNU Ghostscript sous licence libre GPL, qui sort généralement peu après la version suivante d'AFPL Ghostscript. On peut parler dans ce cas-ci de licence chrono-dégradable (Smets et Faucon, 1999).

Shared source

Le *shared source* représente la réponse de Microsoft à la pression exercée par le mouvement *open source*. Microsoft critique la licence GPL, dont il dénonce le caractère « viral ». Le *shared source* est censé garantir l'innovation et le respect de la propriété intellectuelle, base du développement à long terme de l'industrie des produits numériques (Mundie, 2002). Le *shared source* prend la forme de plusieurs contrats de licence. Il en ressort que le *shared source* est surtout réservé à quelques institutions universitaires et aux grands partenaires commerciaux. Seules quelques universités ont le droit de modifier le code source. Les autres en disposent à des fins de consultation uniquement.

5.2. Communautés de conception

La communauté du logiciel libre connaît deux tendances quant au développement de nouveaux logiciels. Dans le premier cas, un concepteur isolé présente à la communauté un projet déjà implémenté et cherche à intéresser des contributeurs

[5] Mozilla est le nom adopté par la communauté de développeurs libres participant au développement de ce navigateur.

[6] En septembre 2000, la propriété de Ghostscript a été transférée à Artofcode (Aladdin, 2002). Ghostgum assure le développement de l'interface GSView (Ghostgum, 2002) et Artifex la distribution de Ghostscript (Artifex, 2002).

potentiels. Dans le second cas, un appel à contribution est lancé via une RFC (*Request For Comment*). Le travail initial de réflexion provient du contributeur isolé, mais une coopération (co-conception) peut s'engager par la suite.

Remarquons néanmoins que la communauté du logiciel libre privilégie plutôt une forme d' « implémentation itérative » plutôt que la réflexion. Les projets concurrents s'éliminent progressivement par fertilisation croisée et darwinisme technique. Pour Linux, jusqu'à 12 propositions concurrentes se sont déjà affrontées (Lang, 1997). Ce mode de fonctionnement pénalise les délais mais garantit, en principe, que seule la meilleure solution sera appliquée.

L'industrie du logiciel propriétaire applique aussi le principe des communautés de conception.

InSoft (Nambisam, 2000), société américaine de développement d'outils multimédias, a entrepris d'ouvrir un forum de discussion avant même la commercialisation de son produit, puis de permettre le téléchargement de versions de test. Un responsable a été nommé pour faire transiter les idées des clients vers le *team* interne et, inversement, communiquer sa réponse aux clients. La transparence a été développée et les contributions ont été encouragées.

En 1997, Netscape adoptait la méthodologie de développement flexible et choisissait avec succès pour le développement de son logiciel Navigator 3.0 de sortir un grand nombre de versions bêta intermédiaires de façon à intégrer au mieux le *feedback* des utilisateurs (Iansiti et MacCormack, 1997). D'autres méthodologies de développement informatique, regroupées sous l'appellation *Agile Software Development*[7], mettent aussi en avant le *feedback* du client (Van Damme, 2002).

5.3. Communautés de support

Dans le monde du logiciel libre, le support technique est assuré par la communauté de développeurs-utilisateurs elle-même. Ce support s'exprime à travers les *newsgroups* ou l'écriture de HOWTO thématiques. L'assistance est fortement fragmentée mais est enrichie par une expérience vivante. Sans pour autant remettre en cause les principes de base, la tendance actuelle est pourtant de séparer l'implémentation de la documentation, et de centraliser. C'est l'objet du Linux Documentation Project (Linux Documentation Project, 2002).

Le principe de l'automatisation du support technique à l'aide de bases de données de solutions (*self-support*) se généralise (Kay, 1999). Parallèlement à cette tendance, le développement des communautés de support continue et est repris par des sociétés commerciales telles que Borland et Microsoft.

Borland a principalement développé les *newsgroups* thématiques (Borland, 2002). Les utilisateurs de produits Borland y échange problèmes et solutions. Des forums existent, mais semblent moins utilisés.

Microsoft a davantage structuré son approche (Microsoft, 2002b ; MVP, 2002). En moins d'une décennie, Microsoft a créé près de 250 *newsgroups* thématiques. Les contributions sont évaluées par les pairs ainsi que par Microsoft. Les meilleurs

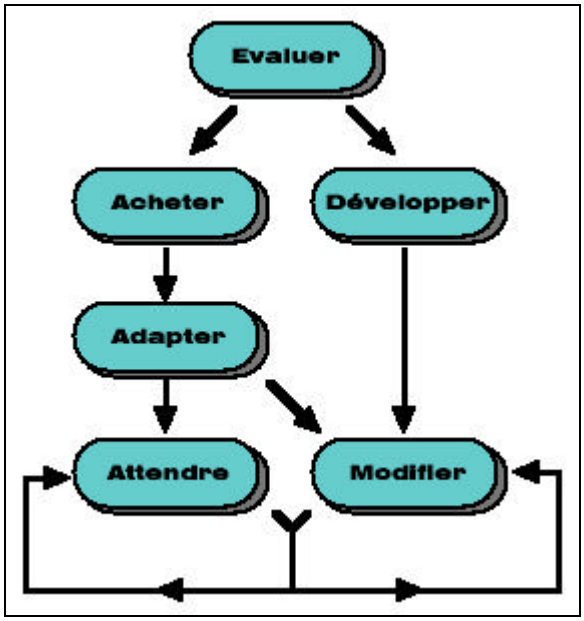
[7] Il s'agit notamment de l'*Extreme Programming* (XP), de la *Dynamic System Development Method* (DSDM), de l'*Iterative Application Development* (IAD) et du *Rational Unified Process* (RUP).

contributeurs sont nommés MVP (*Most Valuable Professionnal*) et valorisés pour la qualité de l'expertise technique mise au service de la communauté. Le programme MVP utilise la valeur de la réputation. Mais l'accent est mis sur la satisfaction du besoin d'émergence plutôt que sur l'humilité cultivée dans la communauté du logiciel libre (Raymond, 1998b).

6. Modèle économique[8]

6.1. Modèle théorique

Smets-Solanes (1998) présente un cycle de développement fonction du modèle de décision schématisé ci-dessous :



Les coûts informatiques totaux répondent au modèle économique général suivant (Smets, 1998) :

$$
\begin{aligned}
CT &= C_A + C_B \\
&= E + D + n \,.\, A \\
&\quad + \\
&\quad p \,.\, n \,.\, \inf [\, C_{B1}, C_{B2}) \\
&= E + D + n \,.\, A \\
&\quad + \\
&\quad p \,.\, n \,.\, \inf [\, r \,.\, B + a, (S + M) / (N \,.\, n) + m \,.\, B \,]
\end{aligned}
$$

C_A - Coûts d'acquisition du logiciel

E Coûts d'évaluation des solutions.
D Coûts d'adaptation du logiciel.
N Nombre de postes.
A Coût de la licence.

C_B - Coût des bogues

[8] Une première validation de ce modèle se trouve dans la fiche 220-2.

p	Probabilité de survenance d'un bogue.
C_{B1} – Coût d'attente d'une mise à jour	
r	Réactivité du fournisseur ou de la communauté.
B	Coût annuel par poste des bogues
a	Coût de mise à jour des licences.
C_{B2} – Coût de modification	
S	Coût d'acquisition des sources.
M	Coût de modification du programme.
N	Nombre de sociétés qui corrigent.
m	Fraction d'année

Ce modèle devient :

- Dans le cas de logiciels propriétaires
  CT = E + D + n . A + p . n . inf [r . B + a, (S + M) / n + m . B]
- Dans le cas de l'utilisation d'un logiciel libre
  CT = E + D + p . n . inf [r . B, M / (N . n) + m . B]
- Dans le cas de développement et utilisation d'un logiciel libre
  CT = E + D + p . n . inf [M / n + m . B]

Il en résulte que (Smets & Faucon, 1999) :

- Développer un logiciel libre est rentable lorsque l'offre commerciale est insuffisante, c'est-à-dire caractérisée par un coût élevé des licences et une réactivité faible de l'éditeur.
- Pour les projets stratégiques, le logiciel libre est un excellent choix, car il garantit l'indépendance.
- La qualité des logiciels libres s'adapte aux utilisateurs les plus exigeants, car ces derniers peuvent corriger les dysfonctionnements dès leur survenance.

6.2. Choix pratiques

Si le modèle de (Smets, 1998) est intéressant de par sa portée générale, il n'est pas assez nuancé. (Raymond, 1999a) distingue pour sa part deux modèles de financement :

- Le financement par la valeur d'utilisation, avec comme stratégies :
  - *Le partage des coûts* – Le serveur web Apache a été développé par un groupe de webmestres reliés par Internet.
  - *L'étalement des risques* – Le système de files d'impression à l'intention du réseau d'entreprise de Cisco a été rendu public.
- Le financement par la valeur d'acquisition indirecte, avec comme stratégies :
  - *La vente à perte pour se positionner sur le marché* – Par exemple, Netscape a choisi de rendre publiques les sources de son navigateur (projet Mozilla) afin de relancer l'innovation et ainsi de tenir tête à Microsoft.
  - *Le gel des gadgets* – Certains fabricants de matériel rendent publics le code source des logiciels associés (comme les drivers), ces derniers n'étant pas une source de bénéfices. Ainsi, Apple Computer a décidé mi-mars 1999 d'ouvrir la source de Darwin, le cœur de leur système d'exploitation *MacOSX*

- *Donner la recette, ouvrir un restaurant* – Red Hat et les autres sociétés distribuant Linux ne vendent pas le logiciel. En réalité, leur valeur ajoutée provient de l'assemblage, des 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.
- *Les accessoires* – Dans ce modèle, on vend des accessoires pour du logiciel à la source ouverte. Cela couvre toute la gamme, des objets comme les tasses et les *t-shirts* aux documentations éditées et produites par des professionnels. O'Reilly édite par exemple des livres sur les technologies libres.
- *Libérer l'avenir, vendre le présent* – Le logiciel est publié sous forme de binaires et la source sous une licence fermée, avec une date limite à laquelle les restrictions prendront fin On trouve dans cette catégorie les licences chrono-dégradables (Smets & Faucon, 1999). Aladin Enterprises suit ce modèle avec son logiciel *Ghostscript*.
- *Libérer le logiciel, vendre la marque* - Sun Microsystems procède de la sorte avec Java et Jini, dans le cadre de la *Sun's Community Source Licence*.
- *Libérer le logiciel, vendre le contenu* – Un site d'information vend de l'information, pas son infrastructure technique. Cette dernière peut avantageusement être libérée. AOL devrait ainsi libérer la source de son logiciel client.

Plutôt que de défendre l'*open source* généralisé, la question stratégique est plutôt de savoir quand il faut libérer le code source (Raymond, 1999a). Doivent être mis en balance :

- Les gains dus aux licences ;
- Les gains dus à la réactivité de la communauté.

Le jeu Doom d'ID Software (exemple développé infra) est un bel exemple d'ouverture de code réussie (Raymond, 1999a).

Par ailleurs, le modèle de développement *open source* est mieux adapté pour les infrastructures logicielles (serveur web, serveur de base de données,...), pour lesquelles la qualité visuelle est secondaire. Il convient moins bien pour les *middlewares* et surtout les applications même si, dans ce domaine, les choses évoluent (Star Office, KOffice, Gimp, Blender,...) (Raymond, 1999a).

6.3. Exemples

Société et produit(s)	**Commentaires**
IBM : - XML Parser ; - Jikes - ...	IBM a créé une division particulière, appelée Alphaworks (Alphaworks, 2002), située dans la Sillicon Valley. Elle surveille les technologies développées au sein d'IBM et ailleurs. Les technologies les plus prometteuses sont identifiées et mises à disposition de la communauté. Les éléments *open source* (généralement des logiciels d'infrastructure) sont améliorés par la communauté puis intégrés dans les produits commerciaux d'IBM, comme

	IBM WebSphere.
Sun Microsystems : ❑ Java ; ❑ Jini ; ❑ …	Soucieux de préserver la propriété intellectuelle, Sun Microsystems a ouvert le code source de certains produits (y compris en micro-électronique). L'ouverture n'est étendue qu'à une communauté restreinte de clients qui se sont acquittés de leur licence (Sun, 2002).
C2Net : Stronghold	C2Net commercialise la version la plus répandue des serveurs web sur base du logiciel *open source* Apache. C2Net met surtout en avant ses propres contributions en matière de sécurité et de cryptographie. C2Net a été rachetée par Red Hat.
ID Software : ❑ Doom ; ❑ Quake ; ❑ …	En 1993, *ID Software* a commercialisé son jeu *Doom*. Premier jeu 3D avec une animation temps réel, *Doom* a connu un record de ventes. En 1997, alors que les ventes déclinaient sous la pression de la concurrence, *ID Software* a libéré les sources. Dès lors, *ID Sofware* a fait du bénéfice sur les marchés annexes comme les recueils de scénarios (Raymond, 1999a).

7. Gestion des projets

Eric Raymond (1998a) oppose deux styles de gestion de projet :

- Le « style cathédrale », plus classique et fortement centralisé, suivi par la majeure partie de l'industrie du logiciel, mais aussi par la *Free Software Foundation*.
- Le « style bazar », plus récent et fortement décentralisé, suivi par Linus Torvald (Yamagata, 1997), le créateur de Linux, et une part croissante du monde *open source*.

Eric Raymond, auteur et coordonnateur du projet *fetchmail* (Raymond, 1998a), a analysé la gestion d'un projet réel de style cathédrale. Il a mis en évidence 19 points importants :

1. Tout bon logiciel commence par gratter un développeur là où ça le démange.
2. Les bons programmeurs savent quoi écrire. Les grands programmeurs savent quoi réécrire (et réutiliser).
3. Prévoyez de jeter un projet, car de toute façon, vous le ferez.
4. Si vous avez « la bonne attitude », les problèmes intéressants viendront à vous.
5. Quand un programme ne vous intéresse plus, votre dernier devoir à son égard est de le confier à un successeur compétent.
6. Traitez vos utilisateurs en tant que co-développeurs est le chemin le moins semé d'embûches vers une amélioration rapide du code et un déboguage efficace.
7. Distribuez tôt. Mettez à jour souvent. Et soyez à l'écoute de vos clients.
8. Etant donné un ensemble de bêta-testeurs et de co-développeurs suffisamment grand, chaque problème sera rapidement isolé, et sa solution semblera évidente à quelqu'un (« Loi de Linus »).
9. Il vaut mieux avoir des structures de données intelligentes et un code stupide que le contraire.
10. Si vous traitez vos bêta-testeurs comme ce que vous avez de plus cher au monde, ils réagiront en devenant effectivement ce que vous avez de plus cher au monde.

11. Il est presque aussi important de savoir reconnaître les bonnes idées de vos utilisateurs que d'avoir de bonnes idées vous-même. C'est même préférable, parfois.
12. Bien souvent, les solutions les plus innovantes, les plus frappantes, apparaissent lorsque vous réalisez que votre approche du problème était mauvaise.
13. La perfection est atteinte, non quand il ne reste rien à ajouter, mais quand il ne reste rien à retirer.
14. Tout outil doit être utile par rapport aux utilisations qu'il a été prévu d'en faire. Mais on reconnaît un outil vraiment excellent au fait qu'il se prête à des usages totalement insoupçonnés.
15. Quand vous écrivez un logiciel jouant le rôle de passerelle quelconque, prenez soin de perturber le moins possible le flot de données et ne perdez jamais d'éléments d'information, à moins que la machine destinataire ne vous y oblige !
16. Quand votre langage est loin d'être « Turing équivalent », un peu de « sucre syntaxique » ne peut qu'aider.
17. Un système de sécurité n'est pas plus sûr que le secret (la clé) qui le garde. Gare aux pseudo secrets !
18. Pour résoudre un problème intéressant, commencez par trouver un problème qui vous intéresse.
19. Pour peu que le coordinateur du développement dispose d'un moyen de communication au moins aussi bon que l'Internet, et pour peu qu'il sache comment mener ses troupes sans coercition, il est inévitable qu'il y ait plus de choses dans plusieurs têtes que dans une seule.

Christopher Browne a analysé les avantages et inconvénients du modèle de développement décentralisé (Browne, 1998). Il souligne notamment que la FSF ne saurait plus contrôler tous ses projets en cours. Le modèle décentralisé, qui privilégie la délégation, réduit la portée de l'effet de « goulot d'étranglement ».

Le démarrage des projets peut se faire de deux façons différentes :

- Soit par soumission d'un projet via les RFC (*Request For Comments*) ;
- Soit par apport d'un code source sur lequel des pairs sont invités à contribuer.

Différents modes d'élection existent (Raymond, 1998a) :

- Le chef de projet est un dictateur bienveillant, en relation avec des responsables de sous-systèmes pour les décisions importantes (Exemple : Linux).
- Le chef de projet est élu par des co-développeurs réunis dans une commission de votants (Exemple : Apache).
- Le chef de projet est un dictateur bienveillant dont le titre est tournant (Exemple : Perl).

Ces deux derniers modes sont plus compliqués et instables.
Eric Raymond (1998b) s'est penché sur la propriété des projets *open source*. Se basant sur la théorie de la propriété de Locke, il identifie trois manières de devenir propriétaire :

- Créer le projet ;
- Se faire confier la charge par le propriétaire, en veillant à la légitimité ;
- Si le projet est en léthargie, annoncer son souhait de succession au propriétaire ou par *newsgroup* s'il n'y a pas de réponse.

Les scissions sont mal vues, de même que les *patchs* pirates. Dans une culture du don qui vise à maximiser la valeur de la réputation, tout ce qui entraîne la diminution de cette dernière est mal vu (Raymond, 1999a).

La résolution des conflits s'effectue suivant trois règles :

- L'autorité vient des responsabilités.
- Les seniors l'emportent.
- Lorsque les deux règles précédentes sont inefficaces, le chef de projet a le dernier mot.

Une étude du Boston Consulting Group confirme que le chef d'un projet *open source* doit être un pair (Lakhani, Wolf & Bates, 2002). Les contributeurs d'un projet *open source* attendent en effet d'abord du chef de projet :

1. Qu'il créé le code source initial ;
2. Qu'il contribue au code ;
3. Qu'il élabore un engagement / une vision ;
4. Qu'il initie le dialogue ;
5. Qu'il intègre les contributions.

Les fonctions classiques du chef de projet - c'est-à-dire la détermination et la délégation des tâches, le recrutement des contributeurs et la gestion du temps – sont considérées comme étant de moindre importance. L'absence d'échéance explique probablement ce constat.

Conclusion

L'utilisation des communautés de consommateurs, que ce soit pour la conception, le développement ou le support de produits informatiques, apparaît à la fois comme un danger et une opportunité pour les entreprises. Tout particulièrement, le co-développement de produits informatiques menace directement la propriété intellectuelle de l'entreprise qui s'engage dans cette voie, mais recèle un fort potentiel en matière d'innovation.

Il s'agit très clairement d'un choix stratégique. Des précautions sont à prendre par rapport au type de produits qui va être ouvert et à la façon de gérer le projet. Une analyse des réussites et des échecs dans le monde industriel devrait apporter un éclairage supplémentaire sur les orientations à adopter[9].

8. Sources

8.1. Livres

1. Dufour A. (1995), *Internet*, Que sais-je ?, PUF, Paris.
2. DiBona C., Ockman S. & Stone M. (ed.) [1999) ; *Tribune libre : les ténors de l'informatique libre*, Editions O'Reilly, Cambridge.
3. Smets-Solanes J.-P. & Faucon B. (1999) ; *Logiciels libres : liberté, égalité, business* ; Edispher, Paris.

[9] Voir à ce propos la fiche 220-2.

8.2. Articles papier

1. Delameilleure J. (2002), 'Ode à la créativité', *DataNews*, 8 février 2002, p3.
2. Lemmens J.-F. (2002), 'Logiciel open source : le service reste le maillon faible', *DataNews*, 8 février 2002, p10.
3. DeLong J. B. & Froomkin A. M. (2000), 'Beating Microsoft at its own game', *Harvard Business Review*, janvier-février 2000, P159-164.
4. Desmet P. (2000), 'Politiques de prix sur Internet', *Revue Française de Marketing*, n°177-178, p49-66.
5. Iansiti M. & MacCormack A. (1997), 'Developing products on Internet time', *Harvard Business Review*, septembre-octobre 1997, p108-117.
6. Le Quérouzec O. (2001), 'Round sécurité : le logiciel libre l'emporte aux points', *Informatiques Magazine*, 14 décembre 2001, p6-8.
7. Landry J. (2000), 'Profiting from open source', *Harvard Business Review*, septembre-octobre 2000, p22.
8. Van damme D. (2002), 'Les nouvelles méthodes de développement qui font fureur', *Datanews*, 10 mai 2002.

8.3. Articles électroniques

1. Lakhani K.R., Wolf B. & Bates J. (2002), 'Hacker survey', *Boston Consulting Group*, janvier 2002. Adresse : http://www.bcg.com/opensource (02-2002).
2. Browne C. B. (1998), 'Linux et le développement décentralisé', *Linux France*[1], 5 février 1998.
3. Free Software Foundation (2002c), 'Catégories de logiciels libres et non libres', *FSF*[1]
4. Free Software Foundation (2002a), 'GNU General Library Public Licence', *FSF*[1].
5. Free Software Foundation (2002b), 'GNU General Public Licence', *FSF*[1].
6. Gabriel R.P. & Joy W.N. (2002), 'Sun Community Source License Principles', *Sun SCSL*[1].
7. Gosh R.A. (1998a), 'Qu'est-ce qui motive les développeurs de logiciels libres ? (interview de Linus Torvald)', *Linux France*[1], mars 1998. Original : Firstmonday[1], vol. 3, n°3.
8. Gosh R.A. (1998b), Les marchés marmites : un modèle économique pour le commerce des biens et des services gratuits sur l'Internet, *Linux France*[1], mars 1998. Original : Firstmonday[1], vol. 3, n°3.
9. Horwitch M., Parikh M.A. & Ziv N. (2000), 'Open Innovation: Transferring Lessons from Software for Modern Value Creation', *Proceedings of the CISEP Workshop on Innovation and Diffusion in the Economy*, Lisbonne, Portugal, 24 janvier 2000.
10. Kay A. (1999), 'Reaching out for help', *CIO Magazine*, 1er octobre 1998. Adresse : http://www.cio.com (06-2002).
11. Kuan J. (2000), 'Open source software as consumer integration into production', *Free / Open Source Research Community (MIT)*[1], octobre 2000.
12. Kirch J. (1999), 'Comparaison Microsoft Windows NT serveur 4.0 – Unix', *Linux France*[1], 23 mars 1999.
13. Lang B. (1997), 'Ressources libres et indépendance technologique', INRIA, octobre 1997. Adresse : http://pauillac.inria.fr/~lang/ecrits/hanoi (06-2002).
14. Microsoft (2002a), 'Some questions every business should ask about the GNU General Public License (GPL)', *Microsoft*[1].
15. Mundie C. (2002), 'The commercial Software Model', *Microsoft*[1], New York University Stern School of Business, 3 mai 2002.

16. Nambisam S. (2000), 'Customer Networks, Entrepreneur Strategy, and Firm Growth : Insights From The Software Industry' ; *Rensselaer Polytechnic Institute*, avril 2001.
Adresse : http://www.babson.edu/entrep/fer/IV/IVD/html/iv-d.htm (04-2002)
17. Nesbitt R. (1999), 'Le dernier dinosaure et les mares au goudron du destin', *Linux France*[1], 10 janvier 1999.
18. Perens B. (2002), 'La définition de l'« *Open Source*', *Linux France*[1] ou *Opensource.org*[1].
19. Raymond E. (1998a), 'La cathédrale et le bazar' , *Linux France*[1], mars 1998. Original : FirstMonday[1], vol. 3, n°3.
20. Raymond E. (1998b), 'A la conquête de la noosphère', *Linux France*[1], octobre 1998.Original : FirstMonday[1], vol. 3, n°10.
21. Raymond E. (1998c), 'Une brève histoire des *hackers'*, *Linux France*[1], 1998.
22. Raymond E. (1999a), 'Le chaudron magique', *Linux France*[1], juin 1999.
23. Raymond E. (1999b), 'The case of the Quake cheats', *Tuxedo*[1], 27 décembre 1999.
24. Raymond E. (2001), 'How To Ask Questions The Smart Way', *Tuxedo*[1], 2001.
25. Smets-Solanes J.-P. (1998), 'L'économie du logiciel libre', *Smets.com*[1], 29 juin 1998.
26. Stallman Richard (1997), 'Linux et GNU', *Linux France*[1], 7 avril 1998.
27. Working Group On Libre Sofware (1999), 'Free Software / Open Source : Information Society Opportunities for Europe'.
Adresse : http://eu.conecta.it (02-2002).
28. Yamagata H. (1997) , 'Le pragmatisme du logiciel libre', *Linux France*[1], septembre 1997.

[1] Cfr. URLographie.

8.4. URLographie

1. Agile Manifesto : http://agilemanifesto.org (05-2002)
2. Alphaworks : http://www.alphaworks.ibm.com (01-2002)
3. Aladdin Enterprises : http://www.aladdin.com (03-2002)
4. Artifex Sotware : http://www.artifex.com (03-2002)
5. Artofcode : http://www.artofcode.com (03-2002)
6. The code project : http://www.codeproject.com (05-2002)
7. EuroLinux Alliance : http://www.eurolinux.org (03-2002)
8. Microsoft : http://www.microsoft.com (05-2002)
9. First Monday : http://www.firstmonday.dk (01-2002)
10. Free Patents : http://www.freepatents.org (03-2002)
11. Free Software Fondation – GNU : http://www.fsf.org (02-2002)
12. FreshMeat : http://freshmeat.net (03-2002)
13. Ghostgum Software Pty Ltd : http://www.ghostgum.com.au (03-2002)
14. Ghostscript Community Site : http://ghostscript.com (03-2002)
15. Information on Microsoft's "Shared Source" Licensing (Bernhard Rosenkraenzer) : http://www.shared-source.com (05-2002)
16. Ghostscript : http://www.cs.wisc.edu/~ghost/doc/AFPL (03-2002)
17. Linux Documentation Project : http://sunsite.unc.edu/mdw/linux.html (02-2002)
18. Linux France : http://www.linux-france.org (01-2002)

19.	Microsoft (MVP Program) :	Http://support.microsoft.com/support/mvp / (01-2002)
20.	Microsoft *shared source* :	http://www.microsoft.com/licensing/sharedsource/ (02-2002)
21.	Free / Open Source Research Community (MIT) :	http://opensource.mit.edu (05-2002)
22.	Most Valuable Professional (MVP) :	http://www.mvps.org (03-2002)
23.	Mozilla :	http://www.mozilla.org (03-2002)
24.	Observatoire Netscraft :	http://uptime.netcraft.com/up/today/requested.html (02-2002)
25.	Opensource.org :	http://www.opensource.org (02-2002)
26.	Smets.com :	http://www.smets.com (01-2002)
27.	SourceForge	http://sourceforge.net (03-2002)
28.	Sun SCSL (Sun's Community Source Licence) :	http://www.sun.com/981208/scsl/ (01-2002)
29.	Extreme Programming :	http://xprogramming.com (05-2002)
30.	Tuxedo :	http://tuxedo.org (04-2002)

9. Table des matières