Je tiens à remercier tout particulièrement :

La Team Samba et plus particulièrement Jelmer Vernooij.

Les développeurs du projet Debian.

Wido Depping, développeur de Luma.

Stig Venaas, développeur de Bind.

Les personnes qui m'ont aidé dans mes deux projets de logiciel libre, sambadoc et kservertools.

Richard Stallman pour ses conférences claires sur ce qu'est le logiciel libre

Les créateurs des logiciels libres Linux, KDE et Openoffice.org qui ont permis de réaliser ce mini mémoire.

Table des matières

1 Introduction....3
1.1 Le Logiciel Libre....3
1.2 Les hackers....3
1.3 La cathédrale et le Bazar....4
1.4 S'appuyer sur ce modèle....5
2 Opter pour ce type de développement....6
2.1 Objectifs et opportunités....6
2.2 Les défis à relever....7
3 L'étude du projet....9
3.1 L'élaboration des spécifications....9
3.2 La planification....10
3.3 Méthode et cycle de développement....12
3.4 Gestion de la propriété intellectuelle....14
3.5 La divulgation du projet....16
4 Design du logiciel....17
4.1 Coding Style....17
4.2 L'architecture du logiciel....17
4.3 Le Framework....19
5 Le Management des équipes....20
5.1 Le Chef de Projet....20
5.2 Créer une communauté....21
5.3 La politique organisationnelle....22
5.4 La gestion des équipes internes et externes....23
5.5 L'infrastructure de support....24
6 La gestion de la qualité....25
6.1 La qualité dans le logiciel libre....25
6.2 Faire correspondre cette approche avec les besoins de l'entreprise....26
6.3 Le système de suivi....28
6.4 L'importance des utilisateurs....29
6.5 La gestion des releases....31
7 Conclusion....32
Références :....33

1 Introduction

1.1 Le Logiciel Libre

Le mouvement du Logiciel Libre a été fédéré par le développeur Richard Stallman[1] alors développeur au laboratoire d'intelligence artificielle du MIT. Il a été créé en réponse à la propriétisation du code source des logiciels qui eut trois conséquences :

- Un chaos juridique autour d'UNIX sans précédent
- Cela bloquait la diffusion de la connaissance à laquelle les scientifiques de l'époque étaient habitués
- Cela empêchait de s'intéresser à un projet technologiquement intéressant

La Free Software Foundation définit les quatres libertés suivantes qu'un logiciel doit posséder pour être considéré comme libre :

- La liberté d'exécuter le programme, pour tous les usages.
- La liberté d'étudier le fonctionnement du programme, et de l'adapter à vos besoins.
- La liberté de redistribuer des copies.
- La liberté d'améliorer le programme et de publier vos améliorations, pour en faire profiter toute la communauté.

Nous pouvons de fait remarquer que les points 2 et 4 nécessitent la disponibilité du code source, ce que les milieux industriels du milieu informatique considèrent pourtant comme leur bien le plus précieux.

1.2 Les hackers

Dans son livre Il était une fois Linux, Linus Torvald commence son propos par les mots « Qu'il est loin le temps où les hommes étaient des hommes et écrivaient eux mêmes leur pilotes de périphériques ».

Les développeurs du logiciel libre sont traditionnellement issus de la culture UNIX où le code est considéré comme un art et la connaissance un bien universel. Ces programmeurs sont surnommés les hacker au sens noble du terme, avant qu'il ne soit détourné par les médias, désignant les

1 Richard Stallman : http://fr.wikipedia.org/wiki/Richard_Stallman

passionnés. En français, il pourrait être traduit par “bidouilleur”.

Les principes de l'éthique des hackers ont été formulés pour la première fois en 1984 par Steven Levy dans son livre Hackers : héros de la révolution informatique[2]. Il définit 6 points décrivant brièvement les hackers :

- L'accès aux ressources informatique devrait être gratuit et illimité.
- L'information doit être gratuite et doit circuler librement.
- Méfiance à l'égard de l'autorité, promotion de la décentralisation.
- Les hackers doivent être jugés sur leur aptitude à programmer et non pas sur leur diplôme, leur âge, leur origine ou leur grade.
- La programmation créé de l'art et de la beauté.

Les hackers se reconnaissent donc entre eux par leur nom et leur contribution avant tout. Ce qui importe c'est la reconnaissance par leurs pairs. Wietse Venema est un hacker très réputé pour le développement de Postfix, moins de gens savent qu'il travaille pour IBM.

Nous observons ainsi un choc des cultures impressionnant entre la rigeur, la normalisation des développements logiciels et l'obscurantisme du code source habituellement pratiqué et le chaos auto-régulé qui s'applique aux programmeurs de logiciels dont le code est ouvert.

1.3 La cathédrale et le Bazar

En 1997, le développeur Eric S. Raymond, également réputé pour ses nombreux essais sur le logiciel libre et son travail de relation publiques avec les entreprises (c'est notamment lui qui convainquit Netscape de libérer le code source de leur navigateur, ce qui donna Mozilla), écrivit une thèse intitulée La Cathédrale et le Bazar[3].

Il y oppose le développement le travail de développement et de conception habituel des logiciels, fermé isolé avec celui des logiciels libres, principalement Linux, dans lequel tout se fait de manière ouverte et déléguée à outrance. Il y montre et s'étonne de comment ce mode de développement apparament anarchique et non normalisé puisse produire des logiciels de très grande qualité, au moins en ce qui concerne le développement d'un système d'exploitation.

2 Source : http://www.volle.com/lectures/citations/hackersethic.htm
3 http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

1.4 S'appuyer sur ce modèle

L'introduction des points précédents était nécessaire pour présenter le contexte dans lequel le développement des logiciels libres s'effectue. Il me fallait montrer comment le manque de cartésianisme et l'importance des programmeur influençait le mouvement avant de présenter comment la conciliation des deux était envisageable.

Malgré toutes les oppositions présentées précédement, des entreprises comme Red Hat, Ximian ou Idealix se sont créées autour de cet écosystème, d'autres comme IBM et Novell ont embrassé le mouvement. La conduite d'un projet de logiciel libre pose de nouveaux défis et demande une nouvelle approche aux entreprises et aux chefs de projet. Nous allons voir comment concillier au mieux les exigences de suivi et de qualité des entreprises et les pratiques à observer dans le cadre de la gestion d'un projet de logiciel libre.

2 Opter pour ce type de développement

2.1 Objectifs et opportunités

Choisir de développer un logiciel sous la forme d'un logiciel libre représente un virage relativement important dans un certain nombre de structures n'ayant aucune expérience du développement de logiciels libres.

Dans un certain nombre de circonstances, la question du choix ne se pose pas, notamment lorsque le produit logiciel final se base sur un logiciel libre. Il est courant de voir des logiciels proposés comme des adaptations de logiciels libres existants. Ainsi, de nombreux routeurs ADSL (Asus, Linksys[4]) sont basées sur des versions modifiées de systèmes Linux, ou encore EDF a implémenté le support Terminal Server sous Samba.

Ainsi, les logiciels libres ont l'avantage de founir un développement plus flexible. En offrant la possibilité de personnaliser le logiciel ils permettent de fournir un produit logiciel complet et ce plus rapidement et moins cher que si l'ensemble des composants logiciels devaient être redéveloppés avec les avantages des fondations techniques du logiciel sur lequel on s'appuie. L'adaptation d'un logiciel permet ainsi de fournir une solution plus facilement conforme aux exigences de la maitrise d'ouvrage.

C'est ainsi que lorsque le gouvernement allement a souhaité créer son serveur groupware[5], le choix technologique a été de se baser sur KDE pour l'un des deux clients mail ainsi qu'un certain nombre de modules serveurs libres (Postfix, Openldap et Cyrus-imap) ainsi cela éviter de recréer ce qui l'avait déjà été.

De plus, cela a permis de mutualiser les coûts entre les différents groupes de développeurs responsables d'un sous ensemble du logiciel et les trois sociétés de services chargées de suivre le développement (chacune ayant sa spécialité).

Ensuite, dans un certain nombre de cas, le logiciel est moins rentable pour une entreprise que la fourniture des services associés. La société Jboss développe ainsi le serveur J2EE éponyme et fournit ainsi des services d'expertise autour de son produit.

4 Distributions des firmware Linux par Linksys : http://lkml.org/lkml/2003/6/7/164
5 Kroupware : http://kroupware.org/

Ensuite, le logiciel possède ici une caractéristique particulière qui le différencie des produits manufacturés. Son développement a un coût lambda, cependant son coût de reproduction avoisine le zéro. Logiciel libre n'équivaut donc pas à logiciel gratuit, cependant la réutilisation des composants permet de réduire très fortement les coûts de départ.

Enfin, les dont le code source est disponible possèdent un certain nombre d'avantages - plus ou moins justifiés – sur lesquels il est commode de s'appuyer. La communauté du logiciel libre attache une grande importance au respect des standards et par conséquent cela facilite l'intégration et l'interopérabilité des logiciels. De plus, la qualité du code source de ces logiciels est globalement reconnue (souvent au grand damn de la documentation).

L'image que l'entreprise véhicule quant à la qualité de ses réalisations techniques est d'une importance cruciale pour son acceptation et l'acceptation de ses logiciels. L'excellence du format de paquet et du système de gestion de mise à jour de la distribution Linux Debian GNU/Linux n'est pas étranger à son succès.

Lorsque la recette des logiciels est publiquement visible, la qualité du travail est immédiatement visible. Il ne faut pas perdre de vue l'importance de la reconnaissance par leurs pairs des développeurs de logiciels libres. On estime ainsi que les logiciels libres sont plus sécurisés et globalement moins buggés que leurs homologues propriétaire. Bien que ce soit une vérité plus ou moins établie selon les logiciels, certains logiciels sont particulèrement bien audités en particulier tout ce qui concerne la sécurité, les services réseau et les outils de programmation. Selon la nature du projet, cela peut donc être considérablement bénéfique pour le logiciel.

2.2 Les défis à relever

Critique

L'équipe en charge du projet doit être ouverte à la critique venant de l'extérieure. Trop souvent seule la critique interne aux membres du projet est évaluée. Bien que dans cetains cas celui puisse conduire à une réécriture partielle du code et donc peut être un allongement des délais, la capacité à intégrer les critiques est un élément clé de la réussite d'un projet de logiciel libre.

Coût

Le développement de logiciels libres sont soumis pour cela au mêmes impératifs que les logiciels propriétaires. De plus ils sont souvent assimilés à tort à des logiciels gratuits. L'avantage ici consiste en la mutualisation des coûts du au partage de la charge de travail.

Cohérence

Les projets de logiciels sont ou devraient être méticulesement suivi afin d'obtenir un produit logiciel cohérent. Il s'agit ici de s'assurer que l'architecture et le code source du logiciel conserve une certaine homogénéité malgré le développement ouvert.

Pertinence

Tout développement logiciel se base sur un cahier des charges définissant les fonctionnalités et le comportement du logiciel. En tant que responsible du développement, le chef de projet doit maintenir le cap dans cet direction.

Relationnel

A partir du moment ou le développement se fait en collaboration avec la communauté de développeurs et d'utilisateurs, vous vous devez d'assurer une certaine qualité relationnelle, et avoir une structure permettant l'accueil et l'échange d'informations avec la communauté.

Qualité

Les défis liés à la qualité possèdent une forte corélation avec ceux liés à la cohérence. La difficulté réside dans l'application du processus qualité et de sa forme au sein du développement logiciel.

Embrasser le mouvement

Ce dernier point est sans doute le plus sensible. À l'époque du développement de la couche de gestion des volumes logiques du noyau linux 2.6, deux projets étaient en concurrence. Le premier LVM2 développé par la société Sistina et l'aute, EVMS d'IBM. EVMS a été rejeté, bien que légèrement meilleur que LVM2 à cause du manque de coopération avec les développeurs du noyau Linux et de son manque d'intégration. Depuis, EVMS est devenu un projet de niche peu répandu et LVM2 a rencontré un franc succès.

3 L'étude du projet

3.1 L'élaboration des spécifications

Les programmeurs n'aiment pas faire de spécifications. Ils aiment programmer. Ils considèrent cela comme une perte de temps. Pourtant, les spécifications permettent d'éviter les problèmes en les identifiant au plus tôt. Ils mettent en avant les besoins en termes d'accessibilité, les nécessité d'effectuer des accès en entrée/sortie. Ils doivent être précis. “Ceci doit permettre à l'utilisateur de réaliser son travail est inéfficace”. Par contre, “Un clic sur ce bouton doit ouvrir une boite de dialogue permettant à l'utilisateur de valider xyz” établit le minimum de ce qui doit être fait.

Ainsi, tout projet de logiciel doit comporter avant la phase de réalisation une spécification. Le logiciel libre n'échape pas à la règle dans ce domaine. L'absence de celle-ci est cruciale pour la réalisation du planning. En effet comment se projeter dans le temps en l'absence d'une vision du travail à effectuer?

La spécification doit permettre de déterminer ce qui doit être fait et comment le réaliser[6]. Sa rédaction doit se faire de façon lisble, elle doit être accessible. Une spécification est caduque si elle est excessivement rebutante. Pour cela, ne pas hésiter à simplifier certaines parties. Une capture d'écran d'un prototypage graphique ou bien une référence à une ressource technique de type RFC est plus parlant qu'un long discours.

Etant donné que le projet s'ouvre vers la communauté du logiciel libre, le plus important est de réduire la barrière à l'entrée[7], aussi bien pour les développeurs que pour les documentalistes.

La spécification doit permettre d'éviter aux développeurs de gaspiller du temps en explications répétées.

Elle doit être de deux ordres :
- Fonctionnelle, que doit-on réaliser?
- Technique, à l'intention des développeurs, comment cela doit être implémenté?

L'élaboration des spécifications ne doit pas se faire à l'écart dans un processus décisionnel autoritaire. Les grands orientations des projets de

6 Spécifications liées au projet KDE : http://developer.kde.org/
7 Le cas du projet Apache : http://httpd.apache.org/dev/

logiciels libres sont définies en commités[8]. Les autres décisions qui ont un portée plus limitée se font souvent par approbation après discussion sur les listes de diffusion. Cela permet d'une part d'avoir une acceptation générale démocratique, et d'autre part de recueillir des idées nouvelles.

Cependant, cela impose au responsable de projet de trancher rapidement. Il ne faut pas que des discussions interminables mettent en péril l'avenir du projet. Dans l'écrit La cathédrale, le Bazar et le Conseil Municipal[9], le développeur Alan Cox explique comment cela a failli tuer le projet de portage de Linux sur processeur Intel 286.

Dans un certain nombre de circonstances, la thèse avancée est que le code EST la spécification. C'est un mauvais argument et généralement les logiciels conçus de cette manière souffrent de grossières limitations au niveau de leur architecture. Ils sont souvent de plus souvent abandonnés et dépassent rarement le stade du prototype. La spécification par programmation itérative est source de perte de temps et de mauvaise qualité.

Enfin, la spécification ne doit pas être excessivement rigide. Elle doit pouvoir évoluer. Si un contributeur ajoute une fonctionnalitée non prévue à l'origine mais qui s'intègre parfaitement, il serait ridicule de la refuser. De même il faut appréhender l'évolution du besoin. Cependant il faut clairement indiquer ce qui ne sera PAS fait. Le projet doit savoir tenir son orientation.

3.2 La planification

Les utilisateurs de Linux entendent parler depuis environ deux ans de la nouvelle version stable de la distribution Debian GNU/Linux. Les développeurs de cette distribution ont pris l'habitude de le plus utiliser que la version de développement sans ligne directrice et sans débugger le produit, aggravant l'obscolescence de la version stable. Celle-ci devait paraître fin 2003, on parle maintenant de juin 2005, c'est à dire “lorsque ce sera prêt”, et certains composants commençent même à être obsolètes avant sa sortie. Ce retard est devenu une source de moquerie récurrente depuis. Bon nombre d'entreprises ne peuvent pas se permettre de faire les mêmes erreurs.

Il est donc impératif de faire un planning. Les développeurs ne font pas de planning pourtant et les rares qui en font, le font uniquement car leur supérieur les en a contraint. Pour ne rien arranger, les développeurs font rarement de bons responsables de planning, on se heurete ici au

8 Réunion annuelle des développeurs de Samba : http://www.sambaxp.org/
9 http://www.linux-france.org/article/these/conseil-municipal/bazaar_fr.html

problème du principe de Peter[10]. Et pour ne rien arranger, les projets de logiciels ont la fâcheuse habitude de rarement se terminer dans les temps.

Les plannings souffrent de deux problèmes :
- Il est relativement difficile de faire un planning valable.
- Personne ne croit en la viabilité du planning.

Une contrainte supplémentaire dans la réalisation d'un planning pour un logiciel libre est qu'elle ne peut que difficilement tenir compte des retours qui seront émis sur le logiciel. La part de l'implication ou de la non implication externes de contributeurs peut sensiblement amener le planning à évoluer. Il va de soi qu'un développeur travaillant à une nouvelle fonctionnalité de complexité raisonnable ou une personne traduisant le logiciel dans une autre langue ne devrait pas avoir d'impact notoire dans les délais. Cependant, il faut dans un premier temps garder à l'esprit dans la constitution du planning que seule l'équipe interne en charge du projet participera et faire évoluer le planning au besoin.

La première consigne élémentaire pour réaliser son planning consiste à utiliser un tableur. Il ne doit pas imposer la lourdeur d'un outil outil dédié afin d'être accessible. Les logiciels comme MS Project et Taskjuggler tiennent trop compte des interdépendances entre les tâches évidentes en programmation. De plus, ils rendent difficile les changements d'affectation des développeurs sur une autre tâche. On peut prendre en exemple le cas d'un développeur qui ayant travaillé sur l'aspect système du programme, mettra du temps pour être opérationnel en programmation d'interfaces graphiques.

Le chef de projet doit constituer quelque chose de simple pour chaque développeur de type :

Fonctionnalité	Tâche	Priorité	Estimation initiale	Estimation actuelle	Temps passé	Temps restant

Pour faire un bon planning, le responsable du planning doit décomposer le projet en tâches les plus simples possibles. Une tâche telle qu'un “Générateur de rapport” n'a pas de sens. Il faut décomposer les tâches en tâches les plus simples possibles, comme par exemple de façon grossière : “collecte des données”, “tri”, “affichage de graphiques” en approchant presque du nombre de points de fonctions calculé (méthode Cosmic FFP[11], norme ISO 19761).

10 Le principe de Peter : http://fr.wikipedia.org/wiki/Le_principe_de_Peter
11 http://www.lrgl.uqam.ca/cosmic-ffp/

La précision d'un tel planning a deux effets. En réalisant le planning avec les développeurs, après tout, eux seuls savent le temps dont ils vont avoir besoin pour effectuer un travail, vous obtenez un planning plus précis. Ensuite, vous proposez un travail intéressant dont les frontières sont clairement délimitées et ne nécessitant pas une trop grande implication à long terme pour un contributeur externe. C'est ainsi que le projet KDE marque les souhaits de fonctionnalité (wish list) sur son gestionnaire de bugs[12] de la marque JJ (Junior Job).
Le planning doit également tenir compte du débuggage et évoluer avec les résultats que produit le système de suivi des bugs. Le temps nécessaire au débuggage est par nature imprévisible et bon nombre de projets sont retardés à cause de ce problème. Il est impératif de débugger avant d'écrire du nouveau code, et donc de faire en permanence des tests en parrallèle au travail d'écriture. Les temps d'intégration doivent également être pris en compte. Ils sont souvent source de problèmes. Les développements d'applications graphiques peuvent avoir des problèmes de cohérence comme des raccourcis claviers incompatibles ou deux boîtes de dialogue différentes pour un même usage. Enfin, ces tâches doivent être attribuées et l'attribution des tâches doit être publique. Il faut explicitement dire “qui fait quoi?” et “Pour quel niveau avancement?” afin de faciliter la coopération inter-développeurs.

Comme nous l'avons dit précédemment les projets de logiciels ont l'habitude d'être livrés avec du retard et dans le cas contraire, ils sont buggés jusqu'à l'os. Il est important pour le chef de projet de savoir écarter des fonctionnalités non essentielles pour conserver le timming lorsque cela s'avère nécessaire. Dans le développement de logiciel libre, la difficulté étant que les programmeurs peuvent avoir tendance à apprécier de réaliser des fonctionnalités futiles.

Le projet Mozilla a passé un temps exagéré à coder un client IRC dont ils se seraient bien passés vu son faible succès et qui est en train de passer aux oubliettes dans le gestionnaire de code source depuis que le développement s'est recentré sur le coeur du logiciel (navigateur, composeur et logiciel de courrier). On peut estimer à environ deux ans de retard le temps qu'a coûté les erreurs de planning de Netscape 5.0 (dont Mozilla est issu) et des débuts de Mozilla.

3.3 Méthode et cycle de développement

Le développement des logiciels libres se base sur la proximité des agents au projet et sur la flexibilité. Il est ainsi difficile d'imposer la séquentialité et la lourdeur d'un cycle de vie en V. Beaucoup de projets utilisent plutôt

12 KDE Bugzilla : http://bugs.kde.org

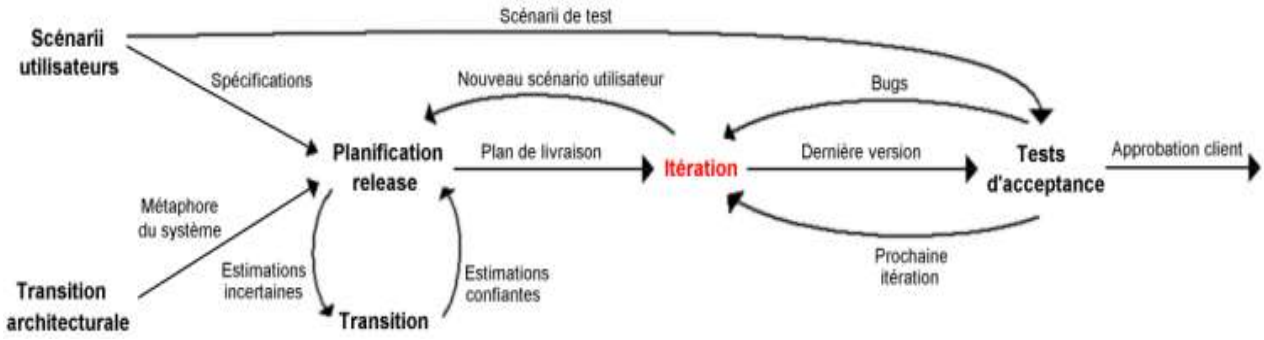
une méthode agile.

Ce type de méthode nécessitent une implication forte du client et de l'utilisateur afin d'augmenter la satisfaction client, ce qui correspond à ce qui est recherché comme nous le verrons plus tard. Ainsi, les méthodes eXtrem Programming[13] et DSDM (Dynamic Software Development Method) sont à favoriser.

Le logiciel libre se prête particulièrement bien aux quatres valeurs fondamentales de cette méthode :

- L'équipe : on met en avant les personnes aux processus et aux outils.
- La collaboration forte : interne (entre programmeurs) et avec le client.
- Le logiciel : on privilégie un code fonctionnel à une documentation abondante
- L'aptitude à réagir : Cela requièrt une grande flexibilité de la part des différents intervenants et dans la manière de gérer le projet. Le cahier des charges et le planning ne sont pas figés.

Néanmoins, elles nécessitent cependant d'être pleinement appliquées pour être utiles. Leur principal avantage est aussi leur principal défaut, elles sont insuffisament cartésiennes. Trop souvent, elles sont utilisées pour donner l'illusion de se baser sur une méthode de développement car les méthodes ne passionnent pas vraiment les développeurs.



Les méthodes de conception telles que UML et Merise sont également rarement utilisées. Elles sont davantage cantonnées au monde des basées de données où leur utilitée est unanime. L'archtecture du logiciel et la refactorisation du code y sont préférés. Il ne faut pas perdre de vue qu'en général, la conception repose souvent sur les développeurs qui préfèrent ainsi programmer que faire un travail d'analyse. De plus, la rigidité de ces méthodes rend plus difficile l'application des méthodes agiles.

13 Synthèse sur l'eXtrem Programming : http://www.idealx.org/doc/xp-synthese.fr.html

Pour contrebalancer la difficulté à faire utiliser une méthode les développeurs de logiciels libres ont néanmoins pour habitude d'appliquer des tests unitaires pour valider leur code. Ils ont l'avantage de très bien se prêter à une méthode de développement agile du fait de l'intimité entre le test le programmeur et le code. Ils offrent également la même flexibilité.

3.4 Gestion de la propriété intellectuelle

La question de la gestion de la propriété intellectuelle est une question que toute entreprise se pose. Beaucoup d'entre elles ont d'ailleurs un certain nombre d'a priori au sujet des licences de logiciel libre en particulier la GPL.

Les plus courants sont :

- « Les logiciels libres sont gratuits, il est nécessaire de donner ces logiciels. »

Les licences de logiciels libre stipulent clairement qu'il est possible de vendre le logiciel au prix qui vous semblera honnête. De plus, de nombreuses entreprises commercialisent ce genre de logiciels.

- « Même dans le cadre d'un usage interne à l'entreprise, il est nécessaire de redistribuer les sources d'un logiciel sous licence GPL. »

Un projet de développement de logiciel n'a pas forcément pour but d'être commercialisé, il peut consister à adapter un logiciel à des besoins spécifiques. La licence GPL implique une redistribution des sources uniquement lorsque le logiciel est distribué.

- «Distribuer un logiciel sous licence GPL revient a mettre le logiciel dans le domaine public. »

Le code de la propriété intellectuelle spécifie que vous tout oeuvre originale tombe automatiquement sous le coup du droit d'auteur et que vous êtes titulaires de ces droits tant que vous ne les cédez pas. Libre ensuite au détenteur des droits de définir comment son oeuvre sera diffusée. La GPL, bien qu'octroyant davantages d'autorisations à l'utilisateur, n'en reste pas moins le moyen par lequel l'auteur choisit de diffuser son oeuvre et il reste détenteur de ces droits. À noter que la licence ne concerne que le logiciel et non la marque, les marques étant couvertes par le droit des marques. La société RedHat distribue ainsi les sources des logiciels de sa distribution sous licence libre, mais retire tout autre signe de propriété intellectuelle.

La dissipation de certains a priori permet de soumettre le problème du choix de la licence. Le choix d'une licence personnalisée peut permettre d'avoir une licence en parfaite adéquation avec les nécessités de l'entreprise, cependant la multiplication des licences pose un certain nombre de problèmes :

- Craintes par les développeurs et les utilisateurs sur la sincérité d'une licence.
- Inutilité de réinventer la roue.
- Incompatibilités possibles avec les autres logiciels libres, en particulier la licence GPL qui est largement majoritaire, amoindrissant l'intérêt de la liberté du code.

Pour simplifier l'approche du problème, il faut voir qu'il existe grossièrement trois licences et quatres catégories de besoins principaux qui conviennent à la plupart des logiciels :

- Les licences libres « gauche d'auteur » comme la licence GPL. Elles ont un caractère viral, c'est à dire qu'un code logiciel se basant sur un code couvert par une licence de ce type doit être libre. Le firewall Linux iptables est sous cette licence, et de nombreux fabricants de routeurs ADSL l'utilisent en rajoutant une surcouche de fait GPL à ce logiciel.
- Les licences libres autorisant à un autre programme même propriétaire de s'appuyer dessus (ex: la LGPL), le logiciel de base restant libre. La bibliothèque graphique GTK+ à la base de Gnome et de The Gimp est sous cette licence.
- Les licences permettant de faire un logiciel propriétaire comprenant du code sous une licence de ce type (exemple la licence BSD).
- Enfin, il reste possible de distribuer un logiciel sous une double licence. La suite bureautique OpenOffice.org notamment est distribuée sous licence LGPL et sous licence Sun. De même la société trolltech développe la bibliothèque graphique QT, et la distribue sous licence GPL ou sous licence Qt Public Licence, selon que vous souhaitiez faire un logiciel libre ou propriétaire. C'est sans doute le mode de diffusion le mieux adapté au business model de l'entreprise.

D'un point de vue juridique, la seule personne morale apte à faire valoir ses droits est le détenteur du droit d'auteur. Comment faire valoir ses droits face à des contrevenants lorsque ce droit peut être partagé? Bon nombre d'entreprises et d'institutions demandent à ce que le droit d'auteur leur soit transféré afin d'avoir un poids juridique, mais aussi une capacité financière à ester en justice supérieure. La Free Software Fondation agit de la sorte pour tous les logiciels officiels GNU et de même pour la société SUN sur le développement de OpenOffice.org. Ce

peut être une option envisageable selon la position du responsable de la maîtrise d'oeuvre, cependant, cette conception se heurte à un degré d'acceptabilité de la requête variant selon la crédibilité l'entreprise.

3.5 La divulgation du projet

La question du lancement d'un projet de Logiciel Libre est d'une importance capitale. Il est utopique de penser qu'il sera possible de créer une communauté active autour d'un projet à coup d'annonces sur des forums et de listes de diffusion ou bien de communiqués de presse.

Autant le débuggage, les tests et la programmation d'un logiciel de manière collaborative a montré ses preuves, autant il est improbable de construire un projet de ce type à partir de rien. Linus Torvalds a tout d'abord commencé seul dans son coin avant de proposer Linux au reste du monde. Et bien que la première version de Linux était tout juste capable de lancer un shell minimal, il présentait ce dont il était capable.

Afin d'intéresser les développeurs potentiels, il est indispensable d'avoir un morceau de code à montrer afin qu'ils puissent « jouer » avec. Aussi incomplet, inparfait et buggé soit-il, votre premier programme doit clairement établir que vos intentions est promesses crédible. Peu importe les erreurs que vous pourriez avoir faites, ce programme ne doit pas manquer d'être en mesure de prouver qu'il puisse évoluer en quelque chose d'intéressant dans un futur pas trop lointain.

Le fait de démarrer un projet de manière isolée n'implique cependant pas le secret absolu. Le projet doit être cependant suffisament mature pour être crédible, mais l'embryon de logiciel doit être techniquement apte à proposer une base de travail. Et ceci est difficilement réalisable sans une coordination suffisament cadrée.

4 Design du logiciel

4.1 Coding Style

L'écriture d'un logiciel par les programmeurs possède une analogie avec l'écriture manuscrite, chaque personne possède son style. Or, lorsque de nombreux contributeurs travaillent en parallèle, et qu'ils n'utilisent pas un style d'écriture uniforme, cela baisse leur productivité en les empêchant de se concentrer sur le code. Quoi qu'il en soit, les développeurs ne doivent jamais oublier que :

- Le code doit être facilement maintenable.
- Les développeurs lisent le code des autres.
- Les développeurs débuggent le code des autres.
- Les développeurs se basent sur le code des autres.

La lisibilité du code peut dépendre beaucoup du langage de programmation utilisé. Dans certains cas, des Coding Style sont devenus des références, notamment les styles Kernighan&Ritchie, GNU ou Linux[14] pour le C sous UNIX, et il pourrait être avantageux de les réemployer.

Les développeurs du projet Samba ont d'ailleurs tiré les leçons du laxisme à ce niveau et ont imposé depuis l'ouverture de la branche Samba 4 d'utiliser le coding style que celui employé par les développeyrs du noyau Linux. Dans tous les cas, le chef de projet doit établir un document normalisant les éléments suivants :

- Le nommage (la nomination?) des variables, classes, fonctions etc...
- Le niveau d'indentation
- La langue par défaut des messages et commentaires
- Les commentaires doivent expliquer pourquoi le code fait ces instructions, jamais comment il le fait.

4.2 L'architecture du logiciel

Il n'est pas nécessaire de proposer une architecture nouvelle. La raison pour laquelle beaucoup de logiciels libres rencontrent du succès est justement parce qu'ils s'appuient sur des concepts éprouvées. Linux a grandi en s'appuyant sur une base d'Unix et Samba comme un simple client pour une technologie réseau existante.

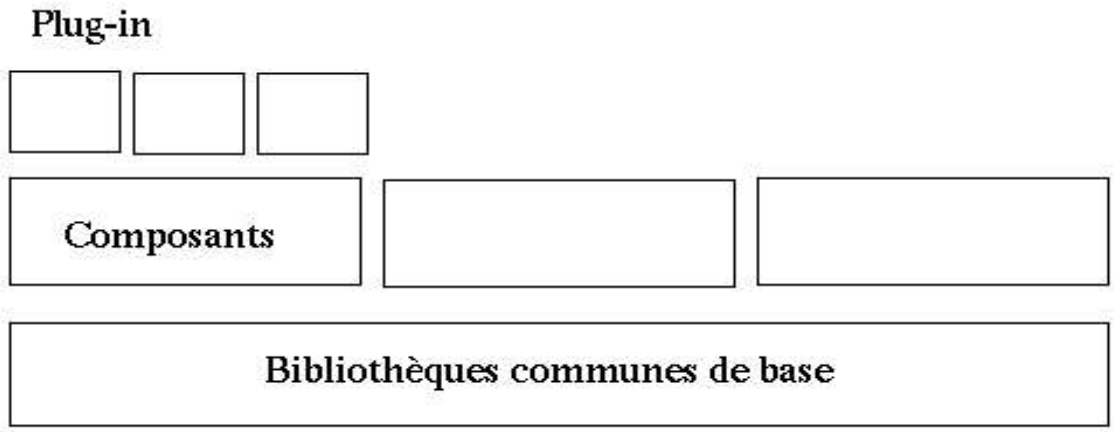
14 http://www.kroah.com/linux/talks/ols_2002_kernel_codingstyle_talk/html/

Une chose importante est le degrée de complexité. Il faut penser le logiciel de manière à ce qu'il puisse grandir. Ironiquement, une erreur de conception serait de faire esthétique et sophistiqué alors qu'il suffirait de faire simple est robuste. En faisant de la sorte, vous prenez le risque que votre projet croule sous son propre poids. C'est notamment ce qui est arrivé au système d'exploitation GNU Hurd qui a pris le parti de se construire sur un micro noyau et non comme un noyau monolithique comme habituellement sous Unix et qui est toujours en gestation depuis 15 ans. De plus, il est toujours plus difficile de persuader des contributeurs de travailler sur un projet dont la complexité est rebutante au premier abord et dont l'avenir est flou.

La conception de l'architecture des composants doit respecter la consigne du “KISS” en privilégiant les structures des données. C'est une expression utilisée par les développeurs de logiciels libres signifiant “Keep It Simple Stupid”. C'est à dire, simplifier l'architecture et la lecture du code de façon à ce que ce soit ridicule de la documenter. C'est bien entendu exagéré mais cela permet de bien saisir le concept.

Le problème qui se dégage des conceptions simples et qu'il faut prévoir de réécrire ou au minimum de refactoriser certaines parties du logiciel. C'est un aspect qui caractérise le logiciel libre, sa refactorisation. Samba 4 est d'ailleurs une réécriture quasi complète du logiciel. Dans cette optique, il peut être judicieux d'utiliser des méthodes de génération dynamique du code source.

Le choix de l'architecture du logiciel impact immédiatement sur la facilité de coordoner le projet. Il est donc important de modulariser le code. La modularisation permet aux programmeurs de ne pas se soucier des autres composants du logiciel. Seule l'interface de programmation (API) du module sur lequel ils s'appuient les concerne.



C'est l'approche qui est généralement adoptée. Du fait des méthodes de développement cela évite aux développeurs de perturber le travail des

autres. Ainsi il convient d'adopter une architecture basée sur les composants enfichables (plug-in).

De nombreux logiciels libres adoptent cette stratégie. La conception de base consiste plus ou moins en un moteur pour d'autres briques logicielles. Le client groupware du projet KDE consiste en un espace de travail sur lequel se greffent des composants Kpart. De la même manière, le serveur Web Apache sait, de base, à peine acheminer des requêtes http. Tout le travail se fait par des bibliothèques modulaires chargées dynamiquement.

4.3 Le Framework

Lors de la conception du logiciel, l'architecte sera confronté au choix du cadre applicatif sur lequel le développement sera basé. Si le but du projet est justement l'écriture d'un nouveau framework, mieux vaut cibler un marché de niche car malheureusement ce type de projet a peu de chance de susciter l'engouement. Il est peu interressant de travailler sur quelque chose qui n'apporte rien de neuf. Et à ce propos, le but de l'utilisation même d'un framework est justement de ne pas avoir à effectuer ce travail.

En effet, les développeurs logiciels libres en ont déjà conçu une pléthore assez importante. Dans le domaine de la programmation J2EE notamment, la fondation Apache[15] a développé un certain nombre de framework qui font foi également pour les serveurs J2EE propriétaires.

Quoi qu'il en soit, le framework doit être libre ou au minimum réutilisable avec des technologies libres. Cela évite ce que la fondation du logiciel libre pour le cas des logiciels libres programmés en java, du “piège java[16]”. C'est à cette fin que les développeurs RedHat travaillent pour réaliser un compilateur en code natif java libre. Ainsi doivent être documenté les différents cadre d'application utilisés :

- Cadre d'infrastructure : Bibliothèque graphique, système de compilation, etc...
- Cadre d'intégration : Interfaces de communication : IPC, Webservices, format de fichier, etc...

15 Fondation Apache : http://www.apache.org/
16 Le piège java http://www.gnu.org/philosophy/java-trap.fr.html

5 Le Management des équipes

5.1 Le Chef de Projet

Dans tout projet informatique, le chef de projet supporte la responsabilité du projet. Il a la double casquette d'interlocuteur privilégié de la maîtrise d'oeuvre et de la maîtrise d'ouvrage.

Cependant, une caractéristique fondamentale qui diffère les chefs de projets de logiciels libres des autres chefs de projet, est qu'ils sont généralement des techniciens de qualité et également les architectes du projet. Cela est d'ailleurs plus visible dans les petits projets. La communauté du Logiciel Libre étant fortement liée à une approche technique du logiciel, le chef de projet doit être un interlocuteur sachant répondre à des demandes techniques très précises. Il doit prendre part à la programmation du logiciel.

De part sa position d'interlocuteur, le chef de projet doit être quelqu'un possédant une grande aptitude à écouter et à communiquer. Il doit être doué en relations humaines. Ceci a d'ailleurs toute son importance en cas de conflit. Bien que cette propriété n'est pas propre au logiciel libre, elle est ici aussi importante que sa capacité à concevoir.

Dans l'optique de créer une communauté active de développeurs et d'utilisateurs, le chef de projet doit séduire, intéresser les gens au projet et les encourager dans leurs démarches.

De bonnes connaissances techniques sont un prérequis, mais l'image que donne le chef de projet est très importante. Ce n'est pas pour rien si les gens perçoivent Linus Torvalds ou Miguel De Icaza (Gnome[17]) comme des gens sympathiques qu'ils sont prêts à aider. Cependant, le chef de projet ne doit pas pour autant faire preuve d'une certaine autorité communément acceptée. Eric S. Raymond, initiateur de l'Open Source Initiative[18] a ainsi donné l'expression de “dictateur bienveillant”.

Le chef de projet peut même être dans certains cas extremes, externe à l'entreprise. Mais généralement il apparait plus concevable d'employer cette personne afin qu'elle puisse travailler à temps plein et non sur son unique temps libre. Mattias Ettrich, le créateur de KDE est ainsi désormais le chef de projet pour la réalisation de QT auprès de Trolltech.

17 Site officiel de GNOME : http://www.gnome.org
18 L'Open Source Initiative : http://www.opensource.org/

Enfin, un bon chef de projet doit posséder une bonne aptitude à déléguer. La communauté des hackers attribue les responsabilités au mérite. Après avoir largement contribué à un module du code, un programmeur devient silencieusement le mainteneur et une sorte de référent pour les autres développeurs. Les choses doivent se mettre naturellement en place. Cela finit par créer une hiérarchie acceptée par tous, ce qui est primordial pour la cohésion du groupe de projet. L'idéal pour le chef de projet étant de faciliter ce genres d'approches afin d'améliorer le processus de création et l'investissement des différents contributeurs.

5.2 Créer une communauté

La création d'une communauté d'utilisateur n'est pas une chose aisée mais c'est essentiel pour la vie du logiciel. Nous avons évoqué précédement l'intérêt d'une cycle de développement légers, nécessitant les retours d'utilisateurs. Il est donc important d'avoir une base solide de personnes de profils différents utilisant le logiciel. Il y a toujours un certain nombre d'utilisateurs pour lesquels votre projet de logiciel libre est utile, il est important de pouvoir l'animer afin d'avoir son soutient et sa créativité.

Les communautés permettent de se décharger d'un certain nombre de tâches ingrates et coûteuses comme le support utilisateur, le débuggage, la rédaction de la documentation, l'internationnalisation et sont un excellent vecteur de communication.

La communauté permet de valoriser l'audace et les compétences de certains utilisateurs. Dans le meilleur des cas certains d'entres eux peuvent tout à fait devenir des utilis'acteurs. Les projets ayant les communautés les plus actives sont généralement celles pour lesquelles le logiciel répond d'une façon nouvelle à un besoin. Le projet Amarok a obtenu une quinzaine de développeurs actifs en moins d'un an. Il est important de donner à tous les contributeurs un but en correspondance avec les aptitudes de la personne. Il faut valoriser le cercle vertueux de l'utilisateur contributeur.

Dans certains cas, les contributions externes valorisent l'architecture en plug-ins que nous avons introduit, et l'entité responsable du projet a intérêt à soutenir ces initiatives. La fondation Mozilla a créé le site mozdev.org justement pour permettre la mise sous incubation de fonctionnalité ou encore pour permettre à certains utilisateurs d'avoir des fonctionnalités qui ne seront pas implémentées.

Une dernière façon de donner un coup de pouce à la communauté se formant autour du logiciel peut être de sponsoriser (ou un mécénat) un

certain nombre de contributeurs externes et d'évènements. Beaucoup de programmeurs de logiciels libres font ça sur leur temps libre et c'est parfois un facteur limitant dans l'avancement du logiciel. Ce mécénat peut être de façon interne ou externe. Ainsi, le créateur de Samba a été récemment employé par l'Open Source Development Lab afin de pouvoir travailler à plein temps sur son logiciel. Ou bien encore le programmeur de KDE David Faure est sponsorisé par Trolltech pour travailler sur l'environnement graphique KDE. Cela permet à Trolltech d'améliorer la qualité de KDE qui est la principale vitrine technologique des capacités de leur logiciel libre Qt, et par la même occasion ils disposent d'un programmeur expérimenté pouvant tester leur bibliothèque jusque dans ses derniers retranchements.

On peut d'ailleurs remarquer que la cultivation des esprits de communauté atteint son paroxysme dans le domaine des distributions GNU/Linux. Deux (Debian et Gentoo) des cinq plus importantes distributions Linux ont d'ailleurs pour raison sociale une association et une fondation. Ce sont de pures produit de la communauté. Et parmi les trois distributions restantes, deux sont intensivement développées sous la forme et avec l'aval d'une communauté (Mandrake avec le Club Mandrake et Cooker et Redhat avec le projet Fedora).

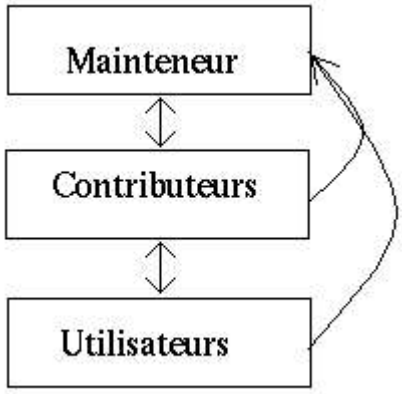
5.3 La politique organisationnelle

La manière dont l'organisation du projet sera faite dépend grandement de la taille et de la technicité du projet. Le développement des logiciels clés du système de base GNU/Linux a principalement été fait de manière relativement fermée. Bien que les orientations sur développement et la documentation du compilateur GCC soit publique, la difficulté d'évaluer la provenance d'un bug dans un compilateur plutot que dans le code source tend à limiter l'ouverture.

Cependant, bien peu de projets profitent d'une telle fermeture. Elle est même plutot généralement à déconseillée. Ce serait contradictoire avec l'esprit communautaire que le projet doit proposer d'autant que le logiciel libre est partiellement né pour permettre à quiquonque de travailler sur un projet attirant son intérêt personnel.

La notion même de hiérarchie est donc à reformuler. Il y a bien entendu la hiérarchie structurelle et naturelle découlant de l'entité institutrice du projet. Il faut toutefois de se garder de conserver une hiérarchie trop stricte à la fois au niveau du commité de pilotage du projet tout comme au niveau des programmeurs.

Il faut aplatir le niveau hiérarchique jusqu'à obtenir un modèle correspondant plus ou moins à la pyramide suivante. Éventuellement il peut être possible de déléguer une partie de la maintenance à des contributeurs réguliers. Dans tout les cas, le mieux est de laisser la hiérarchie se formuler par elle même sous une forme de pseudo méritocratie.



Le problème de modèle réside dans le trop grand flou qui entoure dans les différentes tâches de gestion du projet. Pour remédier à ce problème, beaucoup de projets ont intégré la notion de release manager. Le leader du projet Debian a notamment un impact assez faible sur les releases managers. Ils permettent de contrebalançer le pouvoir des développeurs et les libèrent de cette tâche. En général, ils ont également la charge de gérer la plannification. Ils constituent également en quelque sorte un représentant des intérêts des utilisateurs. Bien souvent le créateur du projet est tellement impliqué dans le projet qu'il n'a pas un regard objectif sur la meilleure façon de le conduire.

5.4 La gestion des équipes internes et externes

Lorsque un projet de logiciel est mené par une entité (entreprise ou administration) souhaitant à la fois faire collaborer l'équipe de développeur interne et des développeurs externes répartis géographiquement se pose la difficulté d'avoir à gérer ces deux équipes. La question se pose particulièrement dans le cas d'une libération des sources et c'était une des premières difficultés qu'à rencontré la fondation Mozilla lors de l'ouverture des sources du navigateur de Netscape.

Pour que l'entente entre les développeurs se passe bien, il faut traiter à égalité les développeurs externes et les développeurs internes. Il ne faut pas que les contributeurs se sentent exclus du processus décisionnel.

Les programmeurs internes peuvent même proposer une sorte de parainage des nouveaux développeurs en proposant leur connaissance du code source.

L'entité devrait également favoriser la création de commités. La fondation Apache et le serveur d'affichage graphique Xorg procèdent de la sorte afin de capter l'acceptation de la direction à suivre pour le projet. Et les membres quelle que soit leur appartenance possèdent la même force décisionnelle. Cette façon de procéder permet d'éviter ce qu'on appele un “fork”. Cela arrive lorsqu'un groupe de développeur suffisament important est en désacord avec la tête d'un projet.

Originellement, le serveur d'affichage graphique sous Linux, Xfree86 était dirigé de manière autocratique par un groupe de développeurs centraux, qui pourtant effectuaient une part assez faible du travail. Cela a conduit à un nouveau projet créé par la grande part de hackers mécontents. Bien que ce fork a été quasiment unanimement suivi, cela reste une perte de temps et une duplication partielle des efforts.

5.5 L'infrastructure de support

L'infrastructure de support est un élément central dans le développement des logiciels libres. Elle y est même particulièrement développée. Elle est présente en réponse aux carrences de communication que peuvent présenter la difficulté d'être géographiquement éloigné pour les contributeurs et permettent de recevoir les retours des utilisateur. C'est un élément indispensable pour fédérer une communauté.

Les listes de diffusion sont généralement préférées car elles ne sont pas dépendantes du fuseau horaire. Elles n'impose pas non plus l'interactivité des auditeurs et permettent de s'adresser à un public plus large. Il est également possible de segmenter les listes de diffusion selon le thème. Ainsi avec des logiciels libres comme Majordomo ou GNU Mailman il est possible d'avoir une liste pour le support utilisateur, une autre pour les développeurs ou encore une autre pour ce qui touche à l'accessibilité.

De la même façon, les Webblogs de développeurs ainsi que les discussions temps réél sur l'Internet via IRC sont également très utilisées. Les discussions temps réél permettent de mettre en relation plus facilement les développeurs avec la communauté. La discussion temps réél a un caractère moins formel que par messagerie électronique. Elles permettent aussi aux programmeurs d'échanger rapidement des idées.

Ensuite, l'infrastructure Web doit également venir faciliter l'accès et le partage de connaissances. En complément du site web, du support documentaire et du gestionnaire de bugs, il faut pouvoir visualiser le dépot. Les gestionnaires de code source CVS et Subversion peuvent tout deux être navigables avec un navigateur via les outils en CGI Webcvs et Websvn.

Un nouveau concept, le Wiki, a récemment débarqué sur Internet. Un Wiki est un site Web que chacun peut modifier à volonté. Un wiki est moteur pour créer des documents de manière collaborative. Wikipédia, l'encyclopédie libre, fonctionne d'ailleurs sur ce principe et possède désormais la plus grande base d'articles de toutes les encyclopédies électroniques existantes.

Les logiciels libres étant massivement liés à Internet, ils n'ont pas non plus échappé à l'engouement des wiki. La mise en place d'un Wiki permet de mettre en place rapidement une ressource documentaire en profitant des corrections et améliorations des contributeurs. De la même façon les Wiki sont très utiles pour recueillir et de partager des informations entre les développeurs sur l'avancement, les objectifs et les évolutions du logiciel.

6 La gestion de la qualité

6.1 La qualité dans le logiciel libre

Dans le domaine du logiciel libre, la programmation fait davantage foi pour la conception que les modèles conceptuels des systèmes informatiques tels que UML et Merise.

La communauté de développeurs de logiciels libres a ainsi provilégié une démarche d'implémentation itérative à la réflexion. Parfois, plusieurs développeurs proposent des solutions concurrentes à une même fonctionnalité. Habituellement, le mainteneur refuse les deux propositions lorsqu'elles sont de qualité égale. A terme, la meilleure solution se différencie, et de temps en temps, différentes solutions deviennent complémentaires. Les projets concurrents s'éliminent progressivement par fertilisation croisée et darwinisme technique.

Dans cette problématique, l'accent est largement mis en avant sur la qualité des structures de données. Un programme peut être mal codé au possible, tant que les structures de données sont propres. La démarche du développement des fonctionnalités du noyau Linux se fait de la manière suivante :

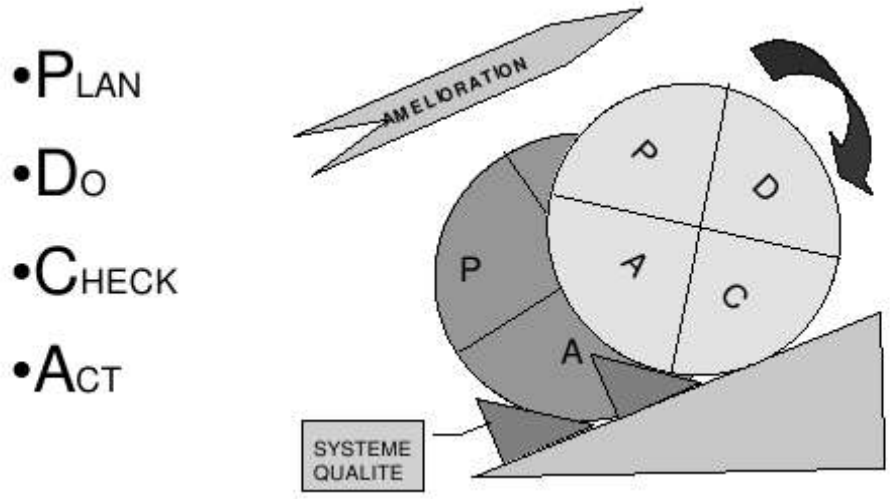
- Faire des interfaces propres
- Faire en sorte que ça marche
- Faire une implémentation propre
- Optimiser

Cette démarche est assez peu conformise (ce mot existe?), étant donné que tout individu et toute entreprise tend naturellement à vouloir arriver directement au point 3, voire 4 a pourtant montré que le résultat en valait la peine, au détriment de l'allongement des délais.

On peut cependant rapprocher cette approche de la qualité avec la méthode qualité PDCA (Plan, Do, Check, Act) autrement appelée Roue de Deming[19] du nom du statisticien Edwards Deming, comportant les quatre points suivants :

- Plan : ce que l'on va faire
- Do : production
- Check : mesure, vérification
- Act : décision améliorative ou corrective

19 http://www.actu-environnement.com/ae/dossiers/iso14000/iso_principe.php4



Une caractéristique prêtée au logiciel libre est son absence relative de bugs. L'étude du développement collaboratif a amené le hacker Eric S. Raymond dans son écrit "la cathédrale et le bazar"[20] a élaboré la "Loi de Linus" disant que lorsque de nombreux programmeurs scrutent le code, les bugs sautent aux yeux. Ce n'est malheureusement valable que lorsque le projet a atteint une masse critique de développeurs suffisament expérimentés.

6.2 Faire correspondre cette approche avec les besoins de l'entreprise

La production industrielle a produit un certain nombre de normes visant à garantir la qualité dans le processus de développement du logiciel. La norme SEMA[21] permet de mesurer la qualité d'une équipe de développeurs et la norme ISO9000[22] notamment vise à accroître, au travers d'une approche processus, la satisfaction client au travers de leurs exigences.

Le problème de la norme SEMA est sa complexité qui la rend applicable dans la douleur. De plus, la distributivité du processus de développement n'y arrange rien. Le développement de logiciels libres en entreprise a appliqué les méthodes classiquement utilisées dans le développement bénévole. Ces méthodes sont davantages basées sur le pragmatisme, qui peuvent sembler quelque peu rudimentaires, mais qui sont éprouvées et approuvées par les développeurs.

20 Cathedral & Bazaar: http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/
21 SEMA : http://www.sei.cmu.edu/sema/welcome.html
22 ISO9000 : http://www.iso.org/iso/fr/iso9000-14000/iso9000/selection_use/selection_use.html

Nous pouvons donc définir un nombre de huit points clés pour la qualité du code :

- Utilise-t-on un système de gestion de code source?
- Avons-nous une base de données des bugs?
- Corrigeons-nous les bugs avant d'écrire de nouvelles fonctionnalités?
- Des builds sont-ils faits quotidiennement?
- Le planning est-il à jour?
- Les spécifications sont-elles disponibles?
- Quelle part est donnée aux tests?
- Faisons-nous des tests d'usabilité de couloir?

Les deux premiers points garantissent un suivi du travail réalisé et seront développés dans la section suivante. La question de la correction des bugs est récurrente. La tentation est forte parfois pour les chefs de projet de faire respecter le planning à tout prix, particulièrement pour les projets ayant déjà du retard. Il faut garder à l'esprit que la satisfaction client ne sera que meilleure avec un compteur de bugs au plus bas, quitte à livrer le logiciel avec du retard ou avec des fonctionnalités manquantes, qui seront ajoutées dans les versions ultérieures. De plus, corriger les bugs longtemps après se révèle beaucoup plus coûteux, notamment si vous corrigez les bugs d'un autre programmeur. De même, les builds quotidiens permettent de s'assurer que l'ensemble du code compile. Enfin, avoir une version relativement bien débuggée permet d'avoir une version prête pour une release à tout instant.

La question de l'actualisation du planning et de la disponibilité des spécifications est du ressort du chef de projet. Le planning doit permettre de répondre aux questions “Où en est-on?” et “Que reste-t-il à faire?”. Il doit être clair et suffisament simplifié pour ne pas noyer son lecteur. Les détails ne concernant finalement que le chef de projet. L'écriture de la spécification n'est pas la tasse de thé des développeurs; cependant, tout bon responsable devrait interdire le passage à l'étape du code sans qu'une spécification ne soit écrite et disponible. La non application de ces deux points a tendance à amener les projets dans tous les sens et n'aboutissent souvent à rien ou amènent à des versions qui sortent “lorsque ce sera prêt”. C'est en partie ce problème qui a conduit le projet Mozilla[23] à ne livrer son navigateur -bien que finalement de qualité- qu'au bout de quatre longues années.

Les deux derniers points concernent les tests. Trop souvent, ce sont les programmeurs qui font eux mêmes les tests et vous rétribuez donc un développeur pour tester alors que vous pourriez l'employer à des tâches plus importantes. Nous verrons par la suite comment une bonne coopération avec les utilisateurs peuvent sauver la mise du chef de projet en assurant un bon suivi du (dys)fonctionnement du logiciel. Les tests de

23 Fondation Mozilla : http://www.mozilla.org

couloirs, quant à eux, doivent se faire parrallèlement au développement, ils ne doivent pas attendre une version d'essai. L'idéal est de prendre le premier utilisateur venu, de lui demander de travailler de façon inopinée sur l'interface graphique que les développeurs viennent de réaliser et d'observer les commentaires qu'il pourra apporter. En répétant cinq à six fois l'opération, l'essentiel des erreurs de conception des interfaces apparaissent. Moins l'utilisateur sera lié au projet, plus son avis sera objectif.

L'application de ces théories fondées sur le pragmatisme ne doivent cependant pas masquer les besoins internes à l'entreprise. En effet, celles-ci couvrent principalement le travail des développeurs. Le chef de projet et l'entreprise peuvent ainsi chapeauter la démarche qualité et ainsi les normes telle que la norme ISO9000 peut être favorable à celle-ci. En effet, elles permettent une approche qualitative globale, garantissant la traçabilité, impliquant l'ensemble des acteurs de l'entreprise.

6.3 Le système de suivi

Le système de suivi est l'outil de pilotage principal de la gestion du projet, il est notamment composé d'un système de gestion de configuration, mais ce n'est pas suffisant.

Celui ci comporte :
- Un système de gestion de version
- Il gère l'attribution des tâches
- Il permet de gérer la matrice de conformité
- Il assure le suivi des correctifs
- Enfin, il permet de gérer les bugs.

Le système de gestion de versions doit permettre un accès multi site et être accessible de l'extérieur. Bon nombre de projets de logiciels libre permettent un accès anonyme, parfois même via un navigateur, au dépôt. Ce n'est cependant pas un impératif, les créateurs du produit logiciel n'étant pas tenu de rendre les sources publiques. Cependant, il est tout de même de bon ton qu'il le soit. L'image du produit n'en est que meilleure, mais surtout cela permet un rapprochement avec la communauté de développeurs.

Il existe un certain nombre de systèmes de gestion de version libres suffisament fiables et complets pour qu'un projet puisse s'appuyer dessus sans sourciller. Cependant, il faut utiliser celui correspondant le mieux au mode de développement adopté. Dans le cadre d'une gestion de code source par fusion de branches, il est préférable d'utiliser un système de gestion de code source distribué (Exemple du noyau Linux : Bitkeeper et

maintenant Git), mais rare sont les développements logiciels avantagés par ce type de systèmes. L'outil de gestion de code source faisant référence est CVS. Bien que souffrant d'un certain nombre de limitations, il a l'avantage d'être universellement connu et utilisé.

De plus en plus de très gros projets se basent désormais sur une alternative conçue comme un remplaçant de CVS sans ses défauts. Des projets comme KDE (dépot de 15Go de sources) et Samba utilisent désormais Subversion comme gestionnaire de sources. Il permet notamment de gérer le rennomage de fichiers sans perte de l'historique ainsi que l'atomicité des commit. Il possède de surcroit une syntaxe très proche de CVS ce qui en fait le gestionnaire de source de choix pour tout projet de logiciel libre. Enfin il s'intègre parfaitement avec les principaux IDE (eclipse, Kdevelop, Anjuta).

A tout moment du développement, les développeurs doivent pouvoir consulter l'état du débuggage du logiciel, et éventuellement ajouter de nouveaux bugs et les assigner à un responsable. Ce support doit être public. La qualité par l'boscurantisme du nombre de bugs est très souvent décriée par la communauté de développeurs. Par exemple, Apple a annoncé il y a deux ans travailler sur un navigateur (Safari) basé sur le code source libre du moteur de KDE, khtml. Cependant la coopération avec Apple n'a jamais abouti, les développeurs de KDE n'ayant ni accès au gestionnaire de source ni à la base de données de bugs. Enfin la base de données des bugs doit comporter un système pour attacher des correctifs à un bugs afin qu'il puisse être revu et testé par d'autres personnes avant son intégration. En effet, plus un bugs est corrigé tard dans le processus de réalisation du logiciel plus les risques de casser une sous partie du logiciel sont importants.

L'ensemble du système de gestion de pilotage du logiciel doit permettre de maintenir un historique clair des modifications, afin de permettre le traçage des changements ainsi qu'une communication ouverte avec l'extérieur. Afin d'instaurer le dialogue, il doit être possible de déterminer qui travail sur quoi. Cela confère un rôle de référent au mainteneur d'une sous partie du logiciel, permettant de faciliter les relations publiques sur les questions techniques.

6.4 L'importance des utilisateurs

Dans le cycle de développement d'un logiciel propriétaire, les logiciels sont usuellement majoritairement débuggés par des testeurs dont c'est l'unique tâche. Le problème étant qu'avoir un nombre suffisament sérieux est varié de testeurs rend la chose coûteuse. Dans le pire des cas, ce sont les développeurs qui réalisent cette tâche, alors qu'ils pourraient faire un

travail plus intéressant et à la hauteur de leurs qualifications.
Nous avons vu précédement le phénomène social de la culture des hackers. Tenant compte du fait que les logiciels libres ont fortement une connotation UNIX, les utilisateurs de ces mêmes systèmes et logiciels ont une tendance similaire à avoir un profil de « bidouilleurs ».

Bon nombre de ces utilisateurs ont tendance à souhaiter des logiciels dont la version est proche de celle du développeur afin de posséder la dernière fonctionnalité réalisée. Un certain nombre d'entres eux d'ailleurs utilisent même parfois la version sortie du gestionnaire de code source. Il suffit pour s'en convaincre de voir les numéros de version des logiciels publiés sur Freshmeat[24], ou la part plus que majoritaire d'utilisateurs de version de développement de distributions Linux comme la Debian Unstable[25].

La contributions des utilisateurs nous permet donc de tirer trois avantages :

- Chute du coup de débuggage.
- Les développeurs reçoivent très rapidement des retours sur les fonctionnalités réalisées, la recette et la qualitification sont donc immédiatement quantifiables.
- Le nombre de bugs colle à la réalisation du produit.
- Les logiciels sont testés en conditions réelles.
- Les meilleurs testeurs apportent parfois des correctifs.

Ceci m'amène à citer la conclusion numéro 6 que Eric Raymond tire du développement de fetchmail dans la cathédrale et le bazar : « Traiter vos utilisateurs en tant que co-développeurs est le chemin le moins semé d'embûches vers une amélioration rapide du code et un débuggage efficace ». C'est également l'enseignement qu'en tire Linus Torvalds dans sa biographie. Avec un peu d'aide et de soutient, certains utilisateurs peuvent devenir de véritables co-développeurs.

Soutenir ses utilisateurs, suppose de la part de l'équipe de développement, une certaine qualité d'écoute. Très souvent, les suggestions des utilisateurs, portent sur l'ergonomie de l'interface utilisateur, mais elles peuvent aussi guider les développeurs vers un framework en particulier. J'ai le souvenir des développeurs du lecteur audio Amarok[26] exprimant leur grattitude[27] à la présentation de la librairie d'effets graphique libvisual, permettant ainsi une réduction du temps de développement des fonctionnalités d'effets graphique ainsi qu'une amélioration de la qualité du code source.

24 Freshmeat : http://www.freshmeat.net
25 Debian Unstable : http://www.debian.org/releases/unstable/
26 Amarok : http://amarok.kde.org
27 Source : http://www.osnews.com/story.php?news_id=9105&page=2

Enfin, les retours de la part des utilisateurs sont souvent porteurs de demandent à propos des fonctionnalités. Par exemple, Linus Torvalds a ajouté la première version de la pagination disque sous Linux fin 1991 à la demande d'un allemand qui n'avait que 2Mo de mémoire vive, l'empêchant de compiler quoi que ce soit. Et pourtant, cette fonctionnalité est une fonctionnalité de base de tout système d'exploitation actuellement. Anticiper sur le besoin, est un facteur clé du succès d'un logiciel.

Cependant, dans bien des développements de logiciels, la liste des fonctionnalités est rigidifée très tôt. Il est nécessaire que l'équipe en charge du projet pemette une certaine marge de manoeuvre à ce niveau pour tenir compte des souhaits de fonctionnalités qui n'étaient pas prévus au départ mais qui s'avèreraient fortement bénéfiques au logiciel. Pour tenir les délais imposés malgré tout, il peut être nécessaire de retirer les fonctionnalités suscitant le moins d'enthousiasme.

6.5 La gestion des releases

Un élément essentiel du développement des logiciels libres est leur faculté à sortir rapidement de nouvelles versions. Bien que courament on y associe la caractéristiques de logiciels buggés, et admettant que la patience des utilisateurs est limitée, le phénomène inverse se produit.

Tout d'abord, le public habituel des logiciels libres qui sont des utilisateurs bricoleurs par nature sont avides des dernières fonctionnalités. La difficulté qui se créé ici est que la course à la fonctionnalité semble assez incompatible avec la stabilité recherchée avec une release, cependant, cela créé une sorte d'émultion qui tend à amoindrir ce soucis.

La règle est donc de distribuer tôt les nouveaux ajouts et de souvent mettre à jour le produit, en particulier pendant les périodes de développement les plus effrénées. Cela améliorera à la fois la qualité et permettra d'avoir des retours rapides sur l'efficience des nouvelles fonctionnalités. Dans les débuts d'Apache et du noyau Linux, il y avait d'ailleurs jusqu'à plusieurs mises à jour quotidiennes du logiciel, ce qui a donné le succès rencontré.

La décision de faire une nouvelle version majeure du logiciel est une question stressante et implique une grande quantité de travail. Une nouvelle version doit posséder à la fois suffisament de nouvelles fonctionnalités pour susciter l'intérêt mais être à la fois suffisament flexible pour évoluer.

Pour qu'une nouvelle version réponde à ses objectifs, elle doit répondre aux trois conditions suivantes :

- Contenir un nombre de correctifs ou de nouvelles fonctionnalité suffisant.
- Etre suffisament espacée de la version précédente afin d'avoir correctement évalué la recette.
- Offrir un nombre suffisant de fonctionnalités pour que l'utilisateur puisse effectuer son travail.

Ensuite, la création d'une nouvelle release doit se faire en admettant une date ciblée. Cela permet de mobiliser l'ensemble des personnes impliquées. Quelques temps avant la sortie d'une nouvelle version doit être établit ce que l'on nomme un gel de fonctionnalité (en anglais feature freeze). À ce moment, l'ajout de nouvelles fonctionnalité doit être interdit voire limité pour permettre de sortir un logiciel efficace et débuggé en inserant un cycle indentifiant le niveau de préparation sous la forme beta/alpha/release candidate.

- Un logiciel en version Alpha contient la plupart des fonctionnalités mais leur fonctionnement est soit partiel soit très instable.
- Dans la phase Beta, le logiciel est globalement utilisable mais encore très buggé. C'est en général peu avant que le gel des fonctionnalités a eu lieu. On effectue également ici le maximum de tests de non régression.
- Enfin en phase de release candidate, le logiciel entre dans sa dernière phase de test intensif qui devrait permettre de déceler les dysfonctionnements majeurs.

Enfin, lorsque le logiciel prêt et que le processus de release est terminé vient le moment de l'annonce du logiciel. Un message récapitulatif doit être effectué sur les listes de diffusion. Ensuite, les sites d'information collaboratives tels que linuxfr[28] sont à contacter. Et enfin, le site de référence pour la communication et la parution de logiciels libres, Freshmeat[29] permet d'exposer le logiciel à la plus large audience qui puisse être recherchée.

28 Linuxfr : http://www.linuxfr.org
29 Freshmeat : http://www.freshmeat.net

7 Conclusion

Cette étude a permis de clairement mettre en évidence l'importance des hommes dans la conduite du projet. Bien que présente dans tout projet, cette composante est ici davantage présente. Même salariés, les programmeurs sont connus avant tout sous leur propre nom selon leur contribution.

La communauté qui se forme autout d'une logiciel libre se place de fait sous la communauté du logiciel libre. Elle a tendance à mettre sur un pied d'égalité les utilisateurs et les personnes qui mettent en oeuvre le logiciel.

Devant le nombre et la variété de logiciels libres, il peut être délicat de mettre en place un nouveau projet qui aboutira. Cependant la nature de celui-ci peut exiger des ressources moindre par rapport à un logiciel propriétaire. De plus un certain nombre de portails mettent en place une infrastructure permettant à des projets libres de démarrer.

L'approche du développement d'une façon si peu rationnelle atteint aussi parfois ses limites. Les environnements particulièrement critiques où la sûreté fonctionnement est en jeu ne se satisfont pas d'une démarche reposant sur le pragmatisme. Et plier les développeurs de logiciels libre à des démarches d'analyse, de qualité et de gestion trop strictes n'aurait aucun avenir.

La difficulté principale reste néanmoins le financement d'un tel projet. Bien entendu, beaucoup de logiciels sont développés par des bénévoles sur leur temps libre. Mais une entreprise souhaitant se lancer dans un tel type de projet se heurterait à la difficulté de mettre en place un business model viable. Les services pour les entreprises autour d'un produit sont ainsi généralement la solution la plus rentable, ce qui limite le type de logiciel à conçevoir.

Références :

Livres :

Précis de conduite de projet informatique de Cyrille Chartier-Kastler.

Il était une fois Linux : L'Extraordinaire Histoire d'une révolution accidentelle, de Linus Torvald et David Diamond.

The Mythical Man-Month. Addison Wesley, 1975. Essai sur l'ingénierie logiciel par Frederic Brooks

Managing Open Source Projects : A Wiley Tech Brief de Jan Sandred

Ressources sur l'Internet :

Le “Howto” Gestion de Projet de Logiciel Libre :
http://mako.yukidoke.org/projects/howto/FreeSoftwareProjectManagement-HOWTO.html

Le projet Linux est-il un modèle possible d'entreprise innovante? :
http://www.libroscope.org/doc/linuxorga/

Netscape annonce la libération du code source de leur navigateur :
http://wp.netscape.com/newsref/pr/newsrelease558.html

L'expérience du projet Mozilla sur le changement de licence :
http://www.mozilla.org/MPL/relicensing-faq.html

La cathédrale et le bazar
http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

Dossier OpenSource de la SSII en logiciel libre Idealix
http://www.idealx.org/dossier/oss/OpenSource.fr.html

Comment poser des questions de manière intelligente ?
http://www.linux-france.org/article/these/smart-questions/smart-questions-fr.html

Qu'est ce qui motive les développeurs de logiciels libres?
http://www.linux-france.org/article/these/interview/torvalds/fr-lt_first_monday.199803.html

Mémoire en marketing sur le marché des systèmes d'exploitation :
http://www.linux-france.org/article/these/memoire-brisset/brisset-annexe.html

A la conquête de la noosphère :
http://www.linux-france.org/article/these/noosphere/homesteading-fr.html

Free Software Projet Management :
http://www.advogato.org/article/196.html

Managing Projet in the OpenSource Way :
http://www.welchco.com/02/14/01/60/00/10/3101.HTM

Le Kernel Coding Style par Gerg Kroah Hartmann
http://www.kroah.com/linux/talks/ols_2002_kernel_codingstyle_talk/html/

Organisation et structure sociale du projet KDE :
http://programming.newsforge.com/article.pl?sid=05/01/25/1859253

Étude de l'université de Maastrich sur le développement de logiciels libres :
http://floss.infonomics.nl/report/