Serveur de logiciel libre
Rapport de projet 3A IT et Mastère ISIC

Cédric BLANCHER, Marc DAUPLAIT,
Cyril DEUBEL, Antonio FIOL, Yves FLAMANT

9 mars 2000

Ecole Nationale Supérieure des Télécommunications de Bretagne - Brest
Scolarité 1999-2000
Encadrants : Eric COUSIN, Nicolas JULLIEN, Gérald OUVRADOU

Remerciements

Nous tenons à remercier les personnes citées ci-dessous pour l'aide et les conseils qu'ils nous ont apportés au cours du projet.

- Encadrants : Eric COUSIN, Nicolas JULLIEN et Gérald OUVRADOU
- Correspondant LRI : Daniel LE GLEAU
- Correspondant DF : Michel BRIAND
- Consultants techniques : Olivier AUBERT, Aymeric POULAIN MAUBANT
- Consultant juridique : Yann DIETRICH

Nous tenons également à citer nos collègues de l'option CMSI pour leur participation au projet sur les aspects juridiques et marketing :

- Camille CACHEUX, Thu Dung LE BINH, Elena VEGA FERNANDEZ.

Résumé et mots-clés

Résumé

Le présent rapport présente le projet "Serveur de logiciels libres" dans son ensemble. Ce projet consiste à mettre en place un serveur de logiciels libres au sein de l'école et à capitaliser le savoir-faire qui en découle. Le résultat doit permettre d'aider au lancement d'un "club libre", pôle d'activités et de réflexion autour du logiciel libre.

Le document est centré sur deux objectifs :

- Présenter le projet et les résultats obtenus
- Rassembler le savoir-faire acquis au cours du projet pour qu'il soit capitalisable

Mots-clés

Serveur, logiciel libre, licence, droit d'auteur, GPL, recommandations, code source, installation, CVS, ACLEditor, OCDB, Wetu, Xtalk, Nest, PhpBookAdmin, Cartographie de cours.

Table des matières

1 Rapport

1.1 Préambule

Le projet "Serveur de logiciels libres" a été réalisé à la fois dans le contexte du colloque "Autour du Libre" (ENSTB Février 2000) et des projets d'élèves de 3ème année et mastères (ENSTB, options IT, CMSI, ISIC, cursus 1999/2000). Les acteurs sont donc les élèves ayant choisi ce projet mais aussi l'école et le GET qui ont participé activement à l'élaboration de documents et à la mise à disposition de moyens informatiques.

Le présent rapport tente de faire une synthèse de l'ensemble du projet. De ce fait, il reprend des passages ou fait référence à d'autres documents rédigés dans le cadre du projet. En particulier, le rapport des élèves de CMSI (voir section 2) fournit déjà beaucoup d'informations (démarche, licences, marketing) et reste indissociable du présent document. Aussi, il est parfois difficile de citer tel ou tel auteur à cause de l'aspect communautaire du travail. Pour y pallier, il convient de préciser que les acteurs du projet sont aussi des auteurs directs ou indirects de parties de ce document. Ces acteurs sont cités en première page dans les remerciements. On retrouve l'ensemble des documents et leurs auteurs dans l'espace de travail partagé associé :
`http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/481706`.

1.2 Les objectifs

1.2.1 Contexte

Chaque année voit la réalisation à l'école de nombreux travaux d'élèves, le plus souvent, sous la forme de logiciels. Certains d'entre eux, de grande qualité, peuvent avoir une portée suffisamment générale pour intéresser la communauté scientifique, voire d'autres milieux professionnels ou associatifs. Malheureusement, force est de constater que dans la grande majorité des cas ces travaux finissent tôt ou tard aux oubliettes. Certes, l'avènement du projet d'ingénieur (S4), il y a trois ans maintenant, a notablement amélioré cette situation sur le court et moyen terme en reliant ces projets à des préoccupations industrielles ou socio-économiques.

On peut toutefois penser que l'école, institution publique, et à ce titre subventionnée par la collectivité nationale, et autres, serait parfaitement dans son rôle en s'efforçant de valoriser des travaux réalisés dans un cadre pédagogique par leur mise à disposition de la collectivité sous la forme de logiciels libres (cette question mérite aussi d'être examinée pour les documents pédagogiques qui pourraient faire l'objet d'une licence "open content"). La mise à disposition des sources sur l'internet est aussi une opportunité pour que ces travaux s'intègrent dans des applications variées et engendrent ainsi une dynamique qui les fera vivre et s'enrichir en stimulant les échanges entre l'école, ses élèves, ses enseignants-chercheurs et son environnement national et international.

Il faut également noter qu'une activité de développement de logiciels existe sur le campus. Ces développements sont réalisés par des étudiants en dehors des travaux "scolaires", le plus souvent dans le cadre de clubs tels que le Resel, la BDTèque... Ces produits sont généralement réalisés dans l'esprit "logiciels libres" avec un niveau de qualité et d'exploitabilité suffisamment élevé pour prétendre à être valorisés sur l'internet au même titre que des projets du cursus scolaire.

1.2.2 Objectifs

Le but du projet "serveur libre " est de mettre à disposition les logiciels développés par les étudiants à l'ENST Bretagne. La façon la plus logique est de créer un serveur de logiciels libres au sein de l'école. Plus que la création même du serveur, la finalité du projet est de définir une démarche de mise en libre de logiciels existants à offrir à la communauté et de proposer une procédure standard pour systématiser la mise en libre de la production de nouveaux logiciels au sein de l'ENST Bretagne (une cinquantaine par an). Cette démarche pourra également créer une base de travail pour les universités et les écoles qui voudraient participer à l'effort du libre.

On conserve à l'esprit que l'idée première du serveur de logiciel libre est de valoriser la production intellectuelle de logiciels du campus de l'ENST Bretagne pour

- offrir à la communauté scientifique ou à d'autres institutions les logiciels développés dans un établissement public d'état ;
- promouvoir l'image de l'école et de ses départements d'enseignement-recherche au travers de la production de logiciels évolués et la participation à la démarche volontariste du libre.

1.3 Le logiciel libre

1.3.1 Définition

La définition qui a été adoptée au cours du projet et que l'on retrouve dans plusieurs documents est : " Par logiciel libre, on entend les logiciels disponibles sous forme de code source, librement redistribuables et modifiables, selon des termes proches des licences GPL, Berkeley ou artistique et plus généralement des recommandations du groupe "open source"
`http://opensource.org/` ".
Cette définition permet de comprendre ce qu'est réellement un logiciel libre. Libre ne veut pas dire gratuit. Il est important de bien intégrer cette notion avant de s'aventurer dans le monde du libre.

1.3.2 Les licences

Un des travaux a consisté en l'étude des licences "libres" existantes et d'en sélectionner une qui soit adaptée au contexte du serveur. Ce travail a été réalisé par les élèves de CMSI. L'analyse des licences GPL, LGPL, Mozilla, Artistic, BSD et SUN License est exposée dans leur rapport section 1.2 . Le choix a été porté sur GPL (GNU Public License) (voir section 2) qui a les spécificités suivantes :

- Les produits sous cette licence et les dérivés doivent rester "Libres".
- Les produits ne peuvent pas être inclus dans des logiciels propriétaires.
- Elle est la plus conseillées par la Free Software Foundation
  `http://www.gnu.org/`.

1.3.3 Droits d'auteur

Le fait qu'un logiciel soit libre n'exclut pas la reconnaissance des auteurs. En fait, licence et droits d'auteur sont des concepts différents qu'il convient

de distinguer. La licence expose la façon dont le logiciel peut être exploité et distribué (par exemple, un logiciel propriétaire peut être libre d'utilisation à usage privé et non redistribuable). Les droits d'auteur concernent les droits qu'ont les auteurs à définir la licence (par exemple, un auteur peut refuser toute diffusion du logiciel). La reconnaissance des auteurs est donc un point tout aussi fondamental que la définition de la licence. De ce fait, les auteurs de logiciels (une modification est un acte d'auteur) doivent être mentionnés dans les logiciels de manière à assurer cette reconnaissance. Dans le cadre de la mise en place du serveur, cet aspect revêt une importance particulière puisqu'il faut obtenir l'accord des auteurs sur la mise en libre pour que celle-ci soit possible.

1.4 La démarche

La démarche globale de mise en place du serveur de logiciel libre à l'école est donnée ci-dessous. Pour chaque point de la démarche on cite le ou les documents qui traitent le sujet en détail et qui représentent l'expérience acquise au cours du projet :

1. Sélectionner quelques logiciels issus de l'école et des élèves à mettre sur le serveur. Voir rapport CMSI section 2.1 .
2. Reprendre ces logiciels si nécessaire et effectuer les actions pour qu'ils soient libres et "exportables". Voir annexe 1 "Processus initial de mise en libre" et annexe 2 "Recommandations sur la forme des sources d'un logiciel libre".
3. Mettre en place le serveur et outils associés. Voir les recommandations liées au marketing dans le rapport CMSI section 3.2. Voir également plus loin la section 1.6 dédiée au serveur, et éventuellement l'annexe 3 "Processus liés au serveur" pour les aspects pratiques de gestion liés au serveur.
4. Rendre le serveur opérationnel et favoriser sa pérennité. Voir plus loin la section 1.7 dédiée à la vie future du serveur.

En parallèle de cette ligne de conduite, il convenait de traiter des aspects tout aussi importants :

- Choix de la licence. Voir plus haut, section 1.3.2.
- Etudier les droits des auteurs des logiciels et les modalités d'obtention de leur accord pour la mise en libre. Voir rapport CMSI section 2.2 .
- Etudier la manière de faire connaître le serveur et préparer le colloque. Voir rapport CMSI section 3 .

1.5 L'organisation

Cette section traite de l'organisation du projet et son déroulement.

1.5.1 Moyens humains

8 élèves ont été volontaires pour participer à ce projet :

- 3 élèves de l'option CMSI. Leur action s'est dirigée vers la sélection des logiciels, les aspects marketing et les aspects juridiques.
- 5 élèves de l'option IT / ISIC. Leur action s'est dirigée vers la sélection et la reprise des logiciels ainsi que la mise en place du serveur.

- Parmis ces 8 élèves, 3 se sont portés volontaires pour assurer des fonctions plus spécifiques : 1 reponsable animation de l'équipe, 1 responsable communication avec l'extérieur de l'équipe et 1 responsable documentation.

3 encadrants ont orienté les activités et participé :

- 2 encadrants du département informatique.
- 1 encadrant du département économie et sciences humaines.

5 correspondants et consultants ont apporté leur concours en fonction de leur domaine.

1.5.2 Moyens matériels

Un espace de travail collaboratif (BSCW) accessible via le réseau a été utilisé tout au long du projet. Les communications ont été effectuées le plus souvent par mail avec utilisation d'une liste de diffusion. Les activités sur ordinateur (rédactions, développement) se sont effectuées sur les machines en libre service de l'école et sur les PC personnels des élèves. La gestion des versions a été effectuée avec CVS.

1.5.3 Déroulement

D'une façon générale, des réunions "tous participants" entre-coupées de réunions "élèves" et ont permis de coordonner les activités. Entre les réunions, chacun travaillait de façon autonome et utilisait le courrier électronique pour communiquer ses résultats ou demander des avis à la communauté. BSCW a été beaucoup employé pour y déposer et partager les productions de chacun. On y retrouve d'ailleur tous les documents de définition de structure et de déroulement du projet
`http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/481706`.

Dans le groupe des élèves, la première tâche a été de déterminer les 3 responsables (animation, communication, documentation) puis de définir les critères de sélection des logiciels à mettre en libre. Ensuite, BSCW a été exploité pour la recherche de logiciels potentiels dans les "réservoirs de travaux d'élèves" des années précédentes. Certains du groupe connaissaient bien les logiciels qui ont été développés par des élèves dans le cadre d'activités personnelles et les ont proposés lorsqu'ils correspondaient aux critères. Un planning à grosses mailles a été établi afin que chacun ait bien conscience des objectifs calendaires.

Après avoir fait une première sélection des logiciels potentiels, les élèves IT/ISIC ont effectué une analyse plus approfondie permettant de dégager 7 logiciels à mettre sur le serveur, chacun prenant une part du travail en fonction de son potentiel et de sa charge. En parallèle, les élèves de CMSI ont travaillé sur le marketing, les aspects juridiques et préparé la première maquette du serveur. Tous les élèves ont préparé le colloque de manière à pouvoir présenter les logiciels, la maquette du serveur et les affiches du stand.

Après le colloque, les élèves CMSI ont terminé leurs travaux et présenté leur soutenance. Les élèves IT/ISIC se sont concentrés sur les travaux logiciels puis sur la mise en place du serveur avec l'aide du correspondant LRI.

1.6 Le serveur

Le serveur est le moyen matériel qui permet la publication des logiciels sur l'internet. Toutefois, sa fonction ne s'arrête pas là. Il est aussi la vitrine du "club libre" (voir plus bas section 1.7), l'espace de communication et de discussion privilégié sur le logiciel libre ainsi que le moyen de centraliser et gérer les sources des logiciels en cours de développement. Chacune de ces fonctions est exposée ci-dessous. Les aspects marketing sont développés dans le rapport CMSI section 3.2 .

1.6.1 Le serveur vitrine du "club libre"

Il s'agit des pages html présentant
- le site internet,
- les logiciels que l'on peut y télécharger,
- la démarche de l'école vis à vis du libre à travers le serveur,
- les documents de recommandations techniques et les documents juridiques

Ainsi, les informations liées au club sont accessibles facilement au plus grand nombre. On cherche à faire partager notre expérience et notre vision du libre. Des recommandations sur la présentation des pages html sont disponibles dans l'espace de travail partagé du projet,
`http://bscw.enst-bretagne.fr/bscw/bscw.cgi/d673062/`, dans un fichier baptisé "RecommandationServeurLibre.html".

1.6.2 Le serveur espace de communication

Il s'agit à la fois de recevoir des retours extérieurs sur les logiciels (utilisateurs ou développeurs) et de faire interagir toutes les personnes intéressées par le logiciel libre. Ceci est rendu possible par la mise en place d'une boite aux lettres spécifique au "club libre" et d'un forum de discussion. Ces outils nécessitent un minimum de gestion et d'animation pour atteindre un niveau de qualité satisfaisant (voir plus bas section 1.7).

1.6.3 Le serveur gestionnaire des sources

Il s'agit de centraliser les sources des logiciels mis en libre et d'en gérer les versions. Pour cela, un serveur CVS est installé sur la machine supportant le serveur libre et contient tous les logiciels publiés ou en passe de l'être. Pour l'instant, CVS n'est accessible qu'en interne. Son ouverture vers l'extérieur en lecture puis en écriture dépendra du dynamisme du "club libre" et des choix qu'il fera en terme de gestion des développements externes. Les téléchargeables sont préparés à partir d'une extraction de CVS dans un fichier d'archive compressé. CVS est également utilisé pour gérer les évolutions des pages html du serveur. Les processus de gestion des versions des logiciels et des pages html sont exposés dans l'annexe 3 "Processus liés au serveur". Là aussi, un minimum de gestion est nécessaire pour assurer la cohérence des évolutions.

1.7 Le club

Le "club libre" est une notion survenue au cours du projet qui a été rapidement identifiée comme la clef de la pérennité et du dynamisme du serveur

de logiciels libres. Il s'agit de fédérer les énergies volontaires d'élèves et de personnels de l'école pour créer une vie autour du serveur où chacun trouve son intérêt. Il semble que des volontés existent bien dans ces deux populations et le défi consiste à trouver le bon équilibre permettant à chacun de s'exprimer pour créer un dynamisme d'ensemble. On trouvera des réflexions sur ce club dans le rapport CMSI section 3.1.3 et dans l'espace de travail partagé du projet :
`http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/673605`.

La structure du "club libre" serait basée sur un "noyau" (ou "bureau") compact composé de membres représentatifs des populations impliquées (élève, enseignant, logistique informatique). Ce noyau permettrait de garantir une existence minimale et permanente du club. Il permettrait également à chaque population de s'exprimer sur sa vision du club pour créer une synergie.

Le reste de la structure du club pourrait être beaucoup plus souple et s'adapter en fonction des volontaires et de ce que chacun veut approter. Toutefois, il y a 4 fonctions fondamentales qu'il faudra assurer sans préjuger de la façon de les répartir entre les acteurs :

- Publication : Faire vivre les pages web.
- Communication : Animer et coordonner les communications, orienter les mails reçus vers les bons acteurs (forum, mails...).
- Administration système : Animer et coordonner les aspects opérationnels et sécurité du serveur et de CVS.
- Coordination des projets : Pour chaque logiciel proposé sur le serveur, animer et coordonner les développements. A priori, un élève acteur significatif sur le logiciel ou un encadrant (pour les projets école).

Si la survie du club est assurée par le "noyau", sa réussite va certainement dépendre du dynamisme de chacune de ces fonctions. La question fondamentale est donc "Comment séduire les volontaires ?". Côté élève, nous pensons que ce qui séduira se résume en ces mots clés "liberté, création, ambiance conviviale, enrichissement personnel". Il faudra donc être attentif à la façon de présenter ces fonctions et l'intervention de la struture de l'école dans le club.

1.8 Les logiciels

Cette section présente sommairement chaque logiciel sélectionné pour être mis sur le serveur. On cherche surtout à mettre en avant les difficultés rencontrées sur la "mise en libre" et les enseignements retirés. Les informations complètes sur chaque logiciel et son état d'avancement sont accessibles sur CVS (point d'archivage de référence) et également dans l'espace de travail partagé du projet :
`http://bscw.enst-bretagne.fr/bscw/bscw.cgi/0/607776`.

1.8.1 ACLEditor

Interface graphique pour la gestion des ACL (Access Control List) sur des machines type UNIX. ACLEditor permet une gestion plus rationnelle des droits d'utilisateur d'un fichier ou répertoire et facilite leur visualisation.

La mise en libre de ce logiciel a été une expérience intéressante sur les aspects suivants :

- problèmes de portabilitée entre les systèmes ;

- a permis de mettre en avant le problème de la langue employée dans les commentaires ;
- mise en place d'une procédure d'installation.

1.8.2 OCDB

OCDB est un serveur CDDB pour Linux (CDDB est une base de donnée audio fournissant les informations relatives au contenu d'un CD, titre, nom de l'artiste, etc.). OCDB s'interface à une base de donnée standard et apporte de nombreuses améliorations par rapport au serveur CDDB classique (plus léger, plus polyvalent, plus flexible et beaucoup plus rapide).

Ce logiciel a été développé sous GPL. L'utilisation d'UnixODBC a cependant nécessité une étude plus poussée de ce module qui s'est avéré avoir été publié sous licenses GPL et LGPL. Ce produit est encore en cours de développement.

1.8.3 Wetu

Comme Xtalk, ce logiciel permet de localiser une personne sur un réseau, mais avec Wetu, cette personne peut être connectée sur un réseau distant. La consultation se fait à travers une interface WEB.

Ce logiciel a été développé sous GPL. Sa mise en libre n'a donc posé aucun problème. Je me suis attaché à corriger quelques bugs et fournir une documentation.

1.8.4 Xtalk

Logiciel qui permet la localisation des personnes connectées sur un réseau et l'établissement de communication avec celles-ci via talk. Il permet aussi d'obtention des informations sur ces utilisateurs, telles que le nom complet, le téléphone...

La mise en libre de ce logiciel a été une expérience intéressante sur les aspects suivants :

- mise en évidence du problème d'estimation du temps de travail ;
- amélioration de la portabilité en terme d'environnement réseau ;
- procédure d'installation complexe.

1.8.5 Nest

Logiciel destiné à étudier les phénomènes d'intelligence collective observée en particulier chez les insectes sociaux comme les abeilles. Il permet de simuler le comportement du groupe à partir de la description du comportement individuel. Les résultats sont évalués avec les algorithmes de Kohonen.

1.8.6 PhpBookAdmin

Ce logiciel écrit en Php3 permet la gestion d'une bibliothèque ou tout établissement permettant le prêt d'ouvrages. Il permet la gestion des membres de l'établissement, de ces ouvrages et des emprunts. Une seule interface Web permet la gestion complète du système grâce à une procédure d'authentification et une gestion de droits échelonnés.

La mise en libre de ce logiciel a été une expérience intéressante sur les aspects suivants :

- large public touché
- reprise d'un logiciel programmé par différentes personnes (langues, style de programmation) non mis en forme et non commenté.
- particularité de l'installation d'un logiciel formé de pages HTML, et utilisant des bases de données.

1.8.7 Cartographie de cours

Logiciel qui permet à l'apprenant de se définir un profil de formation au sein d'un corpus de cours. Ce corpus est visualisé sous la forme d'un graphe que l'apprenant peut explorer avec son navigateur web. Ce logiciel a été réalisé dans le cadre scolaire de projets et stages internes.

La mise en libre de ce logiciel a été une expérience intéressante sur les aspects suivants :

- Il exploitait 2 biliothèques externes propriétaires. La première avait été détectée lors de l'analyse mais la deuxième a été détectée tardivement, ajoutant un volume de travail non prévu à l'origine. Enseignement : Lors de l'analyse lister de façon rigoureuse toutes les bibliothèques exploitées et leurs licences.
- Il exploite 1 bibliothèque externe libre. Enseignement : Expérimentation de l'intégration d'une licence libre dans la licence GPL du produit global.
- Ajout de fonctions suplémentaires. Il manquait une fonction de base dans l'éditeur de graphes. Lors de l'ajout de cette fonction, il a fallu entrer complètement dans le code. J'ai pu voir alors qu'une partie du logiciel avait été mal structurée (fait en fin de stage du développeur) rendant difficiles les modifications. Enseignement : Le travail à réaliser a tendance à être sous-estimé au départ. Comme il est difficile de faire une analyse complète du code, il vaut mieux appliquer un coefficient sur la première estimation.

1.9 Conclusion

De notre point de vue, les objectifs du projet ont été atteints :

- Mise en place du serveur et de la structure minimale associée `http://libre.enst-bretagne.fr/serveur_libre`
- Définition de la démarche de mise en libre à l'ENST Bretagne à partir de l'expérimentation de mise en libre de logiciels (ce rapport + les documents auxquels il fait référence).

Concernant, les futurs travaux à réaliser sur les logiciels ou sur le serveur, consulter les fichiers "TODO" déposés dans les projets CVS associés au serveur.

Maintenant, comme il est précisé dans la section 1.7 décrivant la notion de "club libre", la pérennité du serveur et de l'activité qui l'entoure dépendent des énergies qui seront consacrées à la mise en place de ce club et à la promotion qui en sera faite auprès des élèves et du personnel de l'école pour leur donner envie d'être actifs dans ce domaine.

2 Sources d'information

- Qu'est-ce que le logiciel libre ? - 22 avril 1998 - AFUL - `http://www.aful.org/presentations/libre.html`
- Rapport de projet 3A CMSI "Serveur de logiciel libre" - Février 2000 - Camille CACHEUX, Thu Dung LE BINH, Elena VEGA FERNANDEZ
- GNU Public License - 3 janvier 2000 - Free Software Foundation - `http://www.gnu.org/copyleft/gpl.html`
- Categories of Free and Non-Free Software - 3 mars 2000 - Free Software Foundation - `http://www.gnu.org/philosophy/categories.html`
- GNU Coding Standards - 26 janvier 2000 - Richard Stallman - `http://www.gnu.org/prep/standards.html`
- Un exemple de site "serveur de logiciels libres" - Consulté le 7 mars 2000 - SourceForge - `http://sourceforge.net`

3 ANNEXES

- ANNEXE 1 - Démarche préalable au passage d'un logiciel à "l'état libre".
- ANNEXE 2 - Construire et conditionner un logiciel libre (recommandations).
- ANNEXE 3 - Procédures liées à l'utilisation et à la gestion du serveur.

4 ANNEXE 1 - Démarche préalable au passage d'un logiciel à "l'état libre"

Résumé

La présente annexe a pour objectif de guider une personne souhaitant reprendre un logiciel existant afin de le rendre libre. **Les conseils donnés ici peuvent également être mis à profit pour un nouveau développement.** Cette annexe apporte une aide relativement générale sur les aspects juridiques et techniques : licence, droits d'auteur, évaluation des travaux. Elle ne traite pas des aspects techniques détaillés liés aux sources, à la documentation ou à l'installation du logiciel (ces points sont traités dans l'annexe 2).

4.1 Introduction

Le passage d'un logiciel à l'état "libre" nécessite de se pencher sur l'intérêt et surtout sur la faisabilité de l'opération. Il est fortement recommandé de faire cette estimation avant de commencer les travaux pour éviter une grande frustration par la suite. Ceci est encore plus valable si les travaux doivent s'effectuer dans un temps limité. Les questions soulevées ici restent valables en préalable au lancement d'un développement qu'on envisage porter en libre.

L'estimation de l'intérêt et de la faisabilité s'effectue par l'étude des points suivants :

1. intérêt du logiciel pour la communauté ;
2. motivation du ou des auteurs du logiciel pour le passage en libre ;
3. propriété du logiciel et de ses différents composants (librairies, interfaces graphiques, ...) ;
4. volume des travaux techniques à réaliser.

Chacun de ces points est développé dans la suite.

4.2 Intérêt du logiciel pour la communauté

L'étude de cet aspect ne demande pas une grande dissertation car il se traite au cas par cas. Toutefois, il est essentiel de se poser cette question, ne serait-ce que parce que la mise en libre d'un logiciel peut nécessiter des travaux relativement importants. Il serait frustrant et contreproductif de faire de tels efforts et de constater qu'ils ne profitent à personne. Toutefois, l'utilité avérée d'un logiciel ne suffit pas à elle seule à motiver son utilisation, encore faut-il le faire connaître au sein des communautés qu'il peut concerner. Il faut donc traiter conjointement cet aspect communication. La lecture du rapport CMSI du projet "Serveur de logiciel Libre"[1] apporte des éléments sur ces points.

4.3 Motivation du ou des auteurs du logiciel pour le passage en libre

L'opération de mise en libre d'un logiciel dont on n'est pas personnellement l'auteur ne peut s'envisager, légalement et pratiquement, qu'avec le concours

[1] "Projet Serveur de logiciel libre : le marketing, le droit d'auteur et les licences", C. Cacheux, E. Vegafernandez, T. Dung Le Binh, rapport de projet option CMSI, février 2000, ENST Bretagne.

de son ou de ses créateurs. Cette question doit être élargie aux commanditaires dans le cas d'un logiciel réalisé dans le cadre d'un contrat commercial.

Les aspects juridiques liés à la propriété intellectuelle et à la protection de l'oeuvre doivent faire l'objet d'un examen attentif. Le rapport mentionné ci-dessus apporte là encore des éléments sur le sujet. L'approche qui a été établie au cours de notre étude est que **la décision de passage en libre ne peut se faire qu'à l'initiative du ou des auteurs du logiciel**. Il peut donc y avoir besoin de faire un travail de sensibilisation et d'explication déterminant auprès du ou des auteurs sur le concept même de logiciel libre (ce qui suppose qu'on en ait soi-même une compréhension suffisante).

Selon les cas, l'auteur pourra être volontaire pour participer activement à la mise en libre de son logiciel, ou souhaitera simplement déléguer ce travail. Dans le premier cas, il faudra l'accompagner dans cette tâche en lui expliquant la démarche et en lui fournissant toute information utile telle que celles contenues dans le ce rapport. Dans le second cas, il faudra s'assurer de la bonne volonté de celui-ci à collaborer à l'opération car, le plus souvent, on aura besoin de ses lumières. On peut aussi avoir besoin de son accord pour modifier certaines caractéristiques du logiciel qui s'avéreraient peu appropriées dans le cadre d'une mise en libre.

4.4 Statut juridique du logiciel et de ses différents composants

Ce point demande à être bien examiné car il peut être bloquant pour la suite. Il comporte deux volets :

1. les licences associées au logiciel et aux librairies utilisées par le logiciel ;
2. les droits des auteurs et contributeurs du logiciel.

4.4.1 Les licences

La licence GPL (General Public License) établie par la *Free Software Foundation* a été choisie comme standard pour les logiciels libres qui seront publiés sur le serveur de l'école. Il convient de la connaître pour pouvoir estimer les travaux qui devront éventuellement être effectués et la façon dont le logiciel sera exploité :
`http://www.gnu.org/copyleft/gpl.html`.

La mise en libre du logiciel nécessite de bien maîtriser les parties régies par une licence. Ce peut être notamment le cas de certaines librairies. S'il n'y a aucune librairie sous licence, ce volet est réglé et on applique la licence GPL à l'ensemble du logiciel. Sinon, il est nécessaire de les lister de façon exhaustive et d'étudier leur définition afin de déterminer les possibilités d'utilisation et de redistribution des parties concernées. L'analyse consiste alors à les confronter à la licence GPL pour voir si elles sont compatibles. Si oui, ce volet est réglé.

S'il y a des incompatibilités, il faudra être prudent de ne mettre sous licence GPL que les parties qui ont été développées "en interne" les autres gardant leur licence d'origine. Aussi, on veillera à ce que les licences des librairies importées soient respectées. Par exemple, si une licence précise que la librairie n'est pas redistribuable, celle-ci ne sera pas intégrée dans le logiciel. Dans ce cas, l'utilisateur qui souhaite télécharger le logiciel, devra se procurer lui-même les librairies

et c'est la procédure d'installation qui devra détecter et intégrer les librairies au logiciel. Autre exemple, si une librairie n'est redistribuable qu'en format binaire, on veillera à ne pas redistribuer les sources et on se limitera à redistribuer le binaire selon les termes de la licence. Le texte définissant cette licence devra obligatoirement être intégré au paquetage mis à disposition sur le serveur.

D'une façon générale, suivant ce que recommande la FSF, il est préférable que le serveur publie des logiciels entièrement libres et sous licence GPL. Mais ceci n'est pas totalement strict et dépend finalement de la politique que l'on veut suivre vis-à-vis du libre. Ainsi, la FSF prévoit une licence GPL "affaiblie" (LGPL) qui permet à un logiciel libre de constituer un composant que l'on peut utiliser à partir d'un logiciel propriétaire. Toutefois, lorsque l'on est confronté à la situation inverse (i.e. le logiciel qu'on veut mettre en libre utilise un composant propriétaire), il faut avant tout rechercher l'existence d'un composant équivalent sous licence GPL. Ce n'est qu'en dernier ressort qu'on envisagera de reconstruire celui-ci ; cela ne se justifiant que si la tâche est modeste ou si sa portée est suffisamment générale.

4.4.2 Les droits des auteurs et contributeurs du logiciel

Les acteurs ayant participé au développement du logiciel et les contributeurs (financement, encadrement, conseil...) sont en quelque sorte les "propriétaires" du produit. La mise en libre passe donc par l'accord de ces acteurs et contributeurs. Si, parmi eux, certains refusent la mise en libre, on devra y renoncer. Il faut donc qu'une demande d'accord préalable intervienne suffisamment tôt dans le processus de mise en libre.

Pour formaliser ces accords, l'équipe du projet Serveur Libre a travaillé à la rédaction de textes contractuels en collaboration avec l'expert juridique du GET. Il y a différents textes qui correspondent aux différents contributeurs et statuts du projet :

1. Elève ayant participé au développement dans le cadre d'un projet scolaire.
2. Elève ou extérieur ayant participé au développement en dehors du cadre de l'école.
3. Encadrant ou tout autre membre de l'école ayant participé ou contribué au développement dans le cadre d'un projet de l'école.

Les textes des items 1 et 2 sont disponibles sur simple demande auprès des encadrants du projet Serveur Libre. L'item 3 est en cours de rédaction. A terme, ils seront disponibles directement sur le serveur.

Lorsque des librairies externes sont utilisées, on veillera à ce que les auteurs de ces parties soient cités pour leurs travaux. Il n'est pas nécessaire de leur demander un accord mais simplement de rester en conformité avec la licence de leur produit.

4.5 Volume des travaux techniques à réaliser

Cet aspect demande aussi une analyse approfondie, en particulier si les travaux doivent être effectués sur une durée précise. Les travaux techniques liés à la mise en libre sont de différentes natures :

- test aussi complet que possible du logiciel pour en vérifier les différentes fonctionnalités et établir, le cas échéant, un *bug report* initial ;

- remplacement éventuel des composants dont la licence est incompatible avec la GPL ; ajout éventuel de fonctionnalités dont l'absence pénaliserait lourdement la mise en libre initiale ;
- reformatage du code selon préconisation GNU (cf. annexe 2) pour favoriser les évolutions ultérieures du logiciel ; réalisation de script d'installation et rédaction de la doc associée ;
- reprise éventuellement du code et des scripts d'installation pour que le logiciel soit portable sur d'autres plates-formes que celle pour laquelle il a été développé (machine et OS) ;
- rédaction de la documentation (utilisateur, exploitation, technique).

L'expérience nous a montré que la tendance est souvent de largement sous-évaluer l'ampleur de ces tâches. L'un des facteurs le plus critique est d'estimer le temps nécessaire pour se familiariser avec le code écrit par un tiers afin d'en acquérir la maîtrise. En conséquence, il ne faut pas hésiter à appliquer un coefficient multiplicateur aux estimations initiales. La lecture de l'annexe 2[2] du présent rapport aidera à mieux estimer ces volumes (en particulier sur le code, l'installation et la documentation).

[2] Projet serveur libre, annexe 2, "Construire et conditionner un logiciel libre, guide de recommandations", ENST Bretagne, mars 2000.

5 ANNEXE 2 Construire et conditionner un logiciel libre, guide de recommandations

Résumé

Ce document décrit les recommandations de base dans l'élaboration d'un logiciel pour pouvoir le mettre en libre, c'est-à-dire pour le rendre facilement utilisable et modifiable (qualités que devrait, du reste, offrir tout logiciel, qu'il soit libre ou propriétaire). L'ensemble des conseils fournis proviennent d'une adaptation à nos besoins des recommandations GNU (GNU Coding Standards, Richard Stallman) et de l'expérience acquise lors de notre projet. Il s'en suit que ce document s'adresse plus particulièrement à des logiciels écrits en langage C dans un environnement Unix/Linux. Toutefois, on y trouvera de nombreux conseils d'ordre général qui pourront être appliqués à de nombreux langages de programmation. Ces recommandations concernent aussi bien la façon de programmer que les documentations à fournir et le contenu du fichier contenant le logiciel libre. Ces indications visent à rendre plus facile la diffusion et la reprise des logiciels mis à disposition sur le serveur de logiciel libre de l'ENST Bretagne en en proposant une présentation standard.

5.1 Introduction

Ce document est le produit de la synthèse du travail de l'équipe d'élèves de l'option IT et du mastère ISIC qui a travaillé pour permettre la mise en libre de plusieurs logiciels. Pour ce travail, nous avons repris des logiciels faits par d'autres élèves de l'école, des logiciels qui sont déjà dans le domaine du libre et les recommandations pour les logiciels du projet GNU (GNU Coding Standards, Richard Stallman).

Nous avons rencontré des problèmes de différentes natures, et nous avons estimé nécessaire de creer une liste de recommandations pour ceux qui vont écrire prochainement des logiciels avec le but de les mettre en libre.

Une partie de ces recommandations est issue directement des recommandations GNU, mais d'autres sont beaucoup plus souples, et mieux adaptées au cadre du développement logiciel à l'ENST Bretagne. Et bien sûr, nous avons ajouté des recommandations à partir de notre expérience dans le projet "Serveur Libre".

Ce document doit être pris comme une série de conseils ou des règles à suivre de façon génerale, mais chaque logiciel est un cas d'espèce, avec ses spécificités, et doit être traité comme tel.

5.2 Statut de ce document

Ce document n'est qu'une petite partie de ce qui peut être écrit comme recommandations pour ceux qui vont écrire des logiciels pour la communauté. Les auteurs souhaitent encourager les personnes qui le lisent à ajouter, supprimer, ou modifier le contenu de ce document.

Il est donc librement copiable, diffusable et modifiable.

5.3 Comportement du programme

5.3.1 Robustesse

Évitez les limites arbitraires sur la longueur ou le nombre de toute structure de données, y compris les noms de fichier, de lignes, de fichiers, et de symboles, en assignant toutes les structures de données dynamiquement. Par exemple, dans la plupart des utilitaires d'Unix, les lignes trop longues sont tronquées silencieusement, ce qui peut parfois créer des problèmes.

Implémenter pour chaque fonction ou appel système **la gestion d'erreur**, sauf si vous êtes sûr de vouloir ignorer ces erreurs. Incluez le texte d'erreur système (du *perror* ou de l'équivalent) dans *chaque* message d'erreur en résultat d'un appel système ayant échoué, aussi bien que le nom du fichier et le nom de l'utilitaire. Un message tel que "ne peut pas ouvrir foo.c" ou encore "mode échoué" n'est pas suffisant.

Lorsque le contrôle d'erreur détecte des conditions "impossibles" le programme doit simplement s'arrêter. Il n'y a aucune raison d'imprimer un message. Ces contrôles indiquent l'existence d'anomalies (bogues). Celui qui veut réparer les anomalies devra lire le code source et exécuter un programme de déboguage. Expliquez le problème avec des commentaires dans le code source. Les données importantes seront dans des variables. Celles-ci sont faciles à examiner avec un debugger.

Si vous utilisez des fichiers temporaires, contrôlez la variable d'environnement TMPDIR; si cette variable est définie, utilisez le répertoire indiqué au lieu de "/tmp".

5.3.2 Options dans les lignes de commande

Veuillez ne pas faire dépendre le comportement d'un utilitaire du nom employé pour l'appeler. Il est utile parfois de faire un lien à un utilitaire avec un nom différent, et cela ne doit pas changer ce qu'il fait. Au lieu de cela, employez une option d'exécution ou un commutateur ou tous les deux pour choisir parmi les comportements alternatifs.

De même, ne faites pas dépendre le comportement du programme du type *d'output device* qu'il utilise. **L'indépendance vis-à-vis de l'entrée/sortie** est un principe important de la conception du système; ne la compromettez pas simplement pour éviter que quelqu'un tape une option de temps en temps (la variation de la syntaxe des messages d'erreur en fonction du terminal est par contre acceptable).

Si vous pensez qu'un *type de comportement* est plus adéquat quand la sortie est un terminal, alors qu'un autre est plus utile quand la sortie est un fichier ou un *pipe*, laissez le comportement par défaut lié au terminal, et définissez une option pour le comportement alternatif.

Pour définir un comportement alternatif d'un programme, il est intéressant de **passer des paramètres en ligne de commande**. Ces paramètres devront exister sous un format long et parlant pour rendre le programme plus convivial à utiliser (ex: '–verbose' plutôt que '-v'). Un format court alternatif pourra être défini pour les utilisateurs avertis. Pour permettre une rapide prise en main du logiciel, il est conseillé d'inclure à chaque fois un certain nombre d'options standards :

–version : Cette option doit amener le programme à imprimer des informations sur son nom, sa version, son origine et son statut juridique sur la sortie standard, et puis quitter directement. S'il y a d'autres options et arguments, ils doivent être ignorés, et le programme doit se terminer dès qu'il a affiché le message, sans exécuter sa fonction normale. La première ligne du programme doit être facile à analyser par le *parser*. Le numéro de version du programme doit apparaître après le dernier espace. En outre, il doit mentionner *le nom canonique* du programme, dans un format tel que : *"Gnu Emacs 19,30"*. Le nom du programme doit être **une chaîne de caractères constante** ; ne le calculez pas à partir d'argv[0]. L'idée est d'énoncer le nom standard ou canonique pour le programme, et non son nom de fichier. Il y a d'autres moyens pour découvrir le nom de fichier exact à partir de la variable PATH spécifiant les chemins de recherche des commandes. Si le programme ne constitue qu'une partie d'un plus grand module, mentionnez le nom du module entre parenthèses, comme ceci : *"emacsserver (GNU Emacs) 19,30"*. Ensuite doit suivre la déclaration que le programme est un logiciel libre, et que les utilisateurs sont libres de le copier et de le modifier sous certaines conditions. Si le programme est couvert par la license GNU GPL, dites le ici. Mentionnez également qu'aucune garantie n'est apportée. Il est d'usage de terminer ce texte par la liste des auteurs principaux du programme. Voici un exemple de la sortie qui suit ces règles :

> *GNU Emacs 19.34.5*
> *Copyright (C) 1996 Free Software Foundation, Inc.*
> *GNU Emacs comes with NO WARRANTY,*
> *to the extent permitted by law.*
> *You may redistribute copies of GNU Emacs*
> *under the terms of the GNU General Public License.*
> *For more information about these matters,*
> *see the files named COPYING.*

–help : Cette option doit envoyer un bref descriptif sur la sortie standard expliquant la façon d'appeler le programme, puis se terminer sans erreur. D'autres options et arguments doivent alors être ignorés et le programme ne doit pas exécuter sa fonction normale. Une adresse mail pourra être fournie pour effectuer les *'bug reports'*.

Dans un souci d'uniformisation, il est conseillé de se reporter au manuel "GNU Coding Standards" pour avoir une liste exhaustive des paramètres habituels pour nommer vos autres options.

5.3.3 Utilisation de la mémoire

Si un programme n'utilise que quelques Mo de mémoire, ne faites pas d'effort particulier pour réduire l'utilisation de la mémoire.

Toutefois, pour des programmes comme *cat* ou *tail*, qui peuvent être utilisés avec des fichers de grande taille, il est important de ne pas fixer artificiellement de taille limite pour les fichiers traités. Par exemple, si un fichier traite les données ligne par ligne, il n'est pas utile de garder en mémoire plus d'une ligne.

5.4 Format des fichiers sources

La particularité des logiciels *Open Source* est que l'on donne l'autorisation de reprendre des parties de code et de modifier le programme. Or pour que cette particularité soit exploitable, il faut que les sources fournis avec le logiciel ne soient pas trop difficiles à relire. Se plonger dans les sources d'un logiciel inconnu est déjà un exercice laborieux, inutile de le compliquer avec des problèmes de mise en forme. Voici donc quelques conseils sur la façon de formater ces sources.

5.4.1 Commentaires des sources

Chaque programme doit commencer par un commentaire indiquant brièvement ce qu'il fait. Exemple : `'fmt - filter for simple filling of text'`.

En ce qui concerne la langue choisie pour les commentaires, voici les recommandations GNU :

> "Veuillez écrire les commentaires dans un programme de GNU en anglais, parce que l'anglais est un langage que la plupart des programmeurs dans tous les pays peuvent lire. Si vous n'écrivez pas bien l'anglais, écrivez les commentaires en anglais comme vous pouvez, et demander à d'autres de vous aider à les corriger. Si vous ne pouvez pas écrire des commentaires en anglais, veuillez trouver quelqu'un pour travailler avec vous et traduisez vos commentaires en anglais."

Il est bien évident que nos recommandations ne peuvent être aussi directives, étant donné que l'essentiel des travaux que nous allons intégrer au serveur seront effectuer dans le cadre de l'enseignement d'une école française. Par contre, écrire en anglais nous semble quand même une bonne chose, car cela peut faciliter, non seulement la reprise du logiciel par un plus grand nombre de personnes, mais aussi faciliter l'intégration dans un existant. Bon courage !

Veuillez mettre un commentaire avant le code de chaque fonction expliquant son action, l'action de ses arguments, et les différentes valeurs qu'ils peuvent prendre. Si un argument est utilisé de manière non standard, ou si certaines valeurs peuvent créer des erreures inattendues (ex : les chaînes de caractères contenant des interlignes), soyez sûr de bien le signaler. Expliquez également la signification de toute valeur de retour existante.

Veuillez mettre deux espaces à la fin d'une phrase dans vos commentaires, de sorte que les *commandes de phrase* d'Emacs fonctionnent correctement. Écrivez également des phrases complètes et mettez une majuscule au premier mot. Si un identificateur minuscule vient au début d'une phrase, ne le changez pas (vous parleriez alors d'une variable différente). Si vous n'aimez pas commencer une phrase avec une lettre minuscule, écrivez la phrase différemment (par exemple, " The identifier lower case is...").

Toute variable statique doit être accompagnée d'un commentaire, comme ceci :

```
/* Nonzero means truncate lines in the display;
zero means continue them. */
int truncate_lines;
```

De plus, si vous utilisez de nombreuses parenthèses et accolades, il est bon de mettre des commentaires sur la partie fermante. Ex:

```
if (test) {
.....
} // ends if (test)
```

5.4.2 Apparence du code source

5.4.2.1 recommandations générales Veuillez déclarer explicitement tous les arguments des fonctions. Ne les omettez pas simplement parce que ce sont de simples entiers.

Les déclarations de fonction externes et les fonctions dont l'implémentation n'apparaît que plus tard dans le code doivent être regroupées en un même endroit, de préférence en début de fichier (quelque part avant la première implémentation de fonction), ou dans un fichier *d'en-tête*. Ne mettez pas de déclaration de fonction ou de variable externes à l'intérieur des fonctions.

Trop souvent on utilise la même variable locale temporaire avec un nom peu explicite (ex : temp) tout au long des différentes fonctions. Cette variable a alors diverses utilités et change sans cesse de valeur. Il est préférable de déclarer à chaque fois une variable avec un non parlant. Ceci ne sert pas uniquement à rendre les programmes plus faciles à comprendre mais favorise aussi l'optimisation faite par les compilateurs.

Ne pas utiliser de variables ou paramètres locaux ayant le même nom que des variables globales. Pour déclarer de multiples variables d'un même type, il est plus propre de démarrer une nouvelle déclaration à chaque ligne.

péférez :

```
int foo, bar;
```

ou

```
int foo;
int bar;
```

à :

```
int foo,
    bar;
```

N'hésitez pas à utiliser des accolades et parenthèses. Ainsi à la place du code suivant :

```
if (foo)
  if (bar)
    win();
  else
    lose();
```

écrivez plutôt celui-ci :

```
if (foo)
  {
    if (bar)
      win();
    else
      lose();
  }
```

De plus, il est préférable de séparer les affectations de variables et les autres commandes. Un mauvais exemple est :

```
if ((foo = (char *) malloc (sizeof *foo)) == 0)
  fatal (“virtual memory exhausted”);
```

La version correcte serait plutôt :

```
foo = (char *) malloc (sizeof *foo);
if (foo == 0)
  fatal (“virtual memory exhausted”);
```

De nombreux exemples de ce type pourraient être montrés. Le but n'est pas ici d'en faire une liste exhaustive, mais plutôt de faire comprendre que pour permettre une bonne relecture de votre code, il faut séparer autant que possible les différentes fonctions. En effet, si le code est retouvé, deux lignes précédemment adjacentes ne le seront peut-être plus. De plus une décomposition étape par étape facilite l'implémentation. Ainsi, nous n'avons pas parlé des règles pour indenter (bien que chaque exemple l'illustre), mais cette mise en forme est indispensable.

5.4.2.2 nommage Le nom des variables et des fonctions des fichiers sources font partie intégrante des commentaires. Ne choisissez pas de nom trop court (ex: i,j,p,z...), mais recherchez plutôt des noms donnant des informations pertinentes quand à leur usage. Comme dit précédemment, il semble intéressant de les écrire en anglais.....

Ainsi les abréviations ne sont pas forcément les bienvenues. Bien entendu, si des concepts reviennent tout au long d'un programme, on explique au début les quelques abréviation utilisées (ex : db = database), mais l'excès arrive vite pour une personne reprenant votre code.

Si l'on suit les recommandations précédentes, on aura souvent affaire à des variables ayant des noms très longs. Deux formats sont alors principalement employés. Je vous laisse juger le format que préfère le projet GNU en reprenant son exemple : ignore_space_change_flag != iCantReadThis. Ceci est un détail, mais si cela peut aider certains a comprendre.....

Par contre, il est préférable de ne pas créer de nom de fichier supérieur à 14 caractères. Cela évite des problèmes sur les vieux systèmes de fichiers.

5.5 Mise en place d'une procédure d'installation

5.5.1 problèmes de portabilité

5.5.1.1 types de système Le mot portabilité fait référence aux différentes version d'UNIX existantes. Pour qu'un programme soit le plus utile possible, il faut le rendre le plus portable possible. Pour des systèmes qui ne sont pas *Unix-like*, tels que MS-DOS, Windows, Macintosh, VMS, ou MVS, la portabilité est souvent difficile à réaliser. Il peut donc, à part dans des cas particuliers, sembler plus rentable d'optimiser le logiciel dans sa version Unix/Linux que d'assurer une portabilité tout système.

Pour rendre la portabilité possible, il est intéressant d'utiliser les renseignements sur le système que peut fournir *Autoconf*. Cela permet de changer les différentes options de compilation. C'est pourquoi la plupart des logiciels disponibles doivent être recompilés avant utilisation.

5.5.1.2 types de CPU Ce problème peut vous sembler lointain, mais il faut savoir que certaines erreurs simples de programmation peuvent entraîner une non compatibilité. Ainsi, il est conseillé autant que possible d'utiliser les types de variables adéquats et ne pas utiliser des propriétés particulières de certains systèmes. Par exemple :

```
int c;
.....
while ((c= getchar())!=EOF )
  write (file_descriptor, &c,1);
```

Ce code, s'il est valabe pour tout processeur *little-endian* (PC essentiellement) , ne l'est pas pour pour les *big-endian*. Déclarez donc *c* en tant que *char*....

De même, evitez ceci :

```
int *p, i;
i= (int) p;
```

Ce petit bout de code peut en effet vous poser des problèmes car il joue sur la taille relatives des objets entiers et pointeurs (variable selon les processeurs).

5.5.1.3 Appels système Les appels systèmes différent d'un système à un autre. Par exemple, la variable de retour de *sprintf* diffère suivant les systèmes, *vfprintf* n'est pas toujours disponible, etc... Une liste plus exhaustive de ces problèmes est disponible toujours sur notre document favori "GNU Coding Standards", chapitre 5.7.

5.5.1.4 Internationalisation Pour que les interfaces des programmes puissent être traduites d'une langue à une autre (*i18n* = internationalisation), il est plus pratique d'utiliser *des fonctions d'appel de texte*. Ce texte sera ensuite affiché grâce à l'utilisation d'une bibliothèque dans la langue choisie par un paramètre de configuration. En C, la fonction *gettext* permet d'assurer cette internationalisation. Sinon, chacun peut au niveau de son logiciel faire appel à des variables d'environnement regroupées dans un fichier indépendant.

5.5.2 Makefile et procédure d'installation

Distribuer une version de votre logiciel est plus complexe que simplement rassembler vos fichier sources dans un fichier *tar* et le mettre sur un serveur FTP. Il faut que le logiciel puisse être configuré et installé sur différents systèmes.

5.5.2.1 Script de configuration Chaque logiciel doit être fourni avec un script ayant pour nom *configure*. Celui-ci décrit les types de machine et de système sur lesquels vous pouvez compiler le programme. Il mettra donc à jour les informations des *makefile* pour permettre la compilation du logiciel.

Il faut que le fichier *configure* soit édité par l'utilisateur avant de lancer la compilation. Il existe plusieurs manières de procéder.

1. Faire le lien entre un (ou plusieurs) fichier(s) de configuration avec le(s) fichier(s) de configuration du système choisi (ex : *config.h* avec *config.i386.h* ou *config.m68k.h*). Si vous procédez ainsi, il ne faut pas laissser dans votre *tarball* (au format *tar.gz* le plus souvent) de fichier *config.h*, de façon à ce

que l'utilisateur ne puisse pas compiler votre programme sans avoir exécuté *configure*. L'utilisateur sera donc obligé d'inspecter le(s) fichier(s) de configuration pour pouvoir compiler, ce qui peut être parfois un problème, mais qui en général aide beaucoup.

2. De même *configure* peut créer automatiquement un *Makefile* à partir d'un fichier *Makefile.in* qui contiendra les différents champs à éditer. Là encore, ne laissez pas traîner de *Makefile* pour que l'utilisateur soit obliger de lancer le script *configure.*
3. Une autre possibilité consiste à créer un fichier *config.h.in* pour que configure crée le fichier *config.h*.

Tous les fichiers générés automatiquement doivent avoir en entête un commentaire précisant la façon dont ils sont créés, pour éviter que les utilisateurs n'essayent de les changer à la main. Un fichier *config.status* pourra être écrit par le script *configure* afin de sauver les options avec lesquelles il a été executé la dernière fois, et permettre de regénérer le même *Makefile* (un *shell script* par exemple).

De nombreux points de détails pourraient ici être repris si l'on veut se conformer aux standards GNU. Le plus simple actuellement est quand même d'utiliser le programme *autoconf*, qui aborde plus ou moins automatiquement les problèmes de standardisation. Regardez la documentation de ce programme pour plus d'information.

5.5.2.2 Conventions pour les Makefiles

Là encore, une génération automatique est conseillée. Voici quand même quelques points importants :

1. Précisez le shell utilisé pour exécuter le script. Ex : SHELL = /bin/sh
2. Pour faciliter les modifications, comme toujours, il faut que ce fichier soit écrit de façon claire, et en particulier, il est souhaitable de ne pas faire apparaître plusieurs fois dans le makefile le même fichier. Il est préférable d'affecter des variables avec des listes de noms de fichiers, et d'utiliser ces variables.
3. Pour la gestion de dépendances, utilisez *makedepend* dans une cible spécifique.
4. Ajoutez au moins une cible *clean* pour éliminer les produits de la compilation (fichiers .o, etc). C'est aussi l'usage de prévoir des cibles *distclean* et *maintainerclean*, qui font des nettoyages plus profonds. La première doit laisser le paquetage dans l'état ou il était avant de le compiler. La deuxieme élimine aussi tous les fichiers qui peuvent être générés automatiquement (*Makefile*, *configure*) à partir d'autres fichiers. Ces fichiers ne sont pas éliminés par la cible *clean* parce qu'il y a beaucoup d'utilisateurs qui ne possèdent pas les outils pour les générer (notamment *autoconf*).

Pour la génération automatique, *automake* est l'outil conseillé par GNU. Il prend en entrée des fichiers *Makefile.am*, et genère des fichiers *Makefile*.

5.6 Documentation

5.6.1 Format du *tarball*

Le *tarball* est le fichier compressé regroupant l'ensemble des composants d'un logiciel. Il est en effet plus facile de diffuser un logiciel en un seul bloc (si la taille

le permet). Ce fichier contient un seul répertoire du nom du logiciel qui contient lui-même tous les sous répertoires. Voici quelques conseils visant à standardiser l'apparence d'un logiciel afin que les utilisateurs puissent s'habituer à utiliser des logiciels libres en cours de développement.

Pour créer le *tarball*, il faut premièremernt organiser un répertoire avec la structure décrite. Il est ensuite nécessaire de s'assurer qu'aucun fichier inutile est encore présent. Ceci est particulièrement vrai pour les fichier temporaires crées lors de l'édition, les fichiers de test, les binaires, etc... (une cible du *makefile* sera particulièrement bienvenue pour faire ce travail : "*distclean*"). De plus il est à noter que certains répertoires ne doivent pas être inclus (ex : CVS).

5.6.1.1 Structure Pour que la recherche d'information soit rapide, il est bon d'utiliser une structure de sous répertoire la plus standard possible. Voici la liste des répertoires les plus souvent utilisés :

src ce répertoire contient les fichiers sources du logiciel

doc rassemble la documentation

lib permet de stocker les bibliothèques spécifiques au logiciel

include contient les fichiers de déclarations (*.h) que peuvent utiliser les différents modules

bin rassemble les binaires une fois compilés.

5.6.1.2 Fichiers à inclure Voici une liste de fichier qu'il peut sembler bon d'inclure à la racine du *tarball* d'un logiciel libre. Le contenu de ces fichiers permettent aux utilisateurs ou à toutes personnes souhaitant reprendre le code du logiciel de s'informer.

README ce fichier doit contenir toutes les informations utiles à l'utilisateur qui prend contact avec le logiciel. Ainsi le nom de la license, la liste des principaux auteurs, les références au site de développement, et toute autre information qui pourrait aider un nouvel utilisateur.

COPYING ce fichier est une copie de la license qui régit l'usage de votre logiciel.

INSTALL instructions d'installation. Ce fichier peut être créé en supplément du README si ce dernier est trop long afin d'accéder rapidement aux informations essentielles.

AUTHORS il faut ici simplement donner le nom des auteurs et les parties respectives sur lesquelles ils ont travaillé (les adresses *E-mail* de chacun peuvent être les bienvenues....)

CONTENTS ce fichier à deux intérêts : il permet premièrement en cas de problème d'installation de vérifier si l'un des fichiers à été malencontreusement oublié dans le *tarball*. De plus, si une personne souhaite modifier le logiciel, ce fichier apporte un gain appréciable sur le temps de compréhension de la structure du logiciel. Il suffit souvent pour ce fichier de structurer de manière logique le résultat d'un *'ls'* et d'inclure le bref descriptif présent en entête de chaque fichier.

TODO ce fichier liste tous les changements qu'il est souhaitable d'apporter au logiciel. Bien entendu, si un fichier **BUGS** est déjà inclus, on ne recopiera

pas ici son contenu. Ce fichier montre plutôt quelles sont les possibilités d'extension pour l'application qui n'ont pas encore été mises en place.

BUGS Ce fichier contient une liste de tous les bugs trouvés jusqu'à présent dans une version donnée. Cela permet de montrer si un *bug report* est nécessaire lorsqu'on trouve un bug, et aussi de voir quel travail est à faire sur le logiciel ou ce qui a déjà été pris en charge par quelqu'un.

Les fichiers suivants sont des fichiers qu'il faut rajouter au *tarball* lorsque plusieurs versions du même logiciel ont été fournies. Ces fichiers doivent être remaniés, et l'entête remise à chaque changement de version. De plus, entre deux versions, il ne faut pas effacer les changements de la version précédente, mais simplement les laisser à la fin du fichier.

ChangeLog Ce fichier contient tous les changements effectués sur tous les fichiers source entre deux versions. Ce fichier est important, car il permet de repérer plus facilement les changements ayant pu introduire à leur tour des bogues non présents dans les versions précédentes. Une bonne description de la façon dont ont peut écrire ce fichier est décrite dans le document "GNU Coding Standards", chapitre 6.5.

NEWS ce fichier contient tous les changements **visibles** entre deux versions pour un utilisateur. Il contient donc certaines informtions de *ChangeLog*, mais décrites plus brièvement.

5.7 Conclusion

Pour finir, un conseil vraiment général : essayez de vous mettre à la place d'un futur développeur qui veut reprendre votre logiciel ; pensez à ce qu'il va faire lorsqu'il téléchargera votre logiciel, et les difficultés qu'il va rencontrer lors de cette prise de contact, et plus encore lorsqu'il arbordera la reprise.

Ce serait un bon exercice qu'avant de finir votre logiciel, vous essayiez de reprendre un logiciel de quelqu'un d'autre pour voir comment ça se passe. Bonne chance !

6 ANNEXE 3 - Procédures[3] liées à l'utilisation et à la gestion du serveur de logiciel libre

6.1 Objet

La présente annexe a pour objectif de **guider un acteur souhaitant créer ou modifier un logiciel sur le serveur de logiciel libre** de l'ENST Bretagne. Elle apporte une aide sur les aspects techniques liés au serveur : gestion de version, fichiers téléchargeables et modification des pages html.

Elle ne traite pas des aspects liés aux sources, la documentation ou l'installation du logiciel, ni des aspects juridiques (ces points sont traités dans les annexes 1 et 2) .

6.2 Plan

La création ou la modification d'un logiciel libre sur le serveur consiste à exécuter 3 activités :

1. gestion de version ;
2. préparation du téléchargement ;
3. modification des pages html.

L'ordre dans lequel sont présentés ces activités correspond à peu près à le chronologie à suivre. Chacun de ces points est détaillé dans les chapitres ci-dessous.

6.3 Gestion de Version

6.3.1 Apport de CVS

Les logiciels libres du serveur sont destinés à évoluer grâce à la contribution de développeurs se trouvant en différents lieux et à différents moments. Il est donc important de gérer de façon rigoureuse les versions des sources.

Ceci s'effectue par l'intermédiaire de CVS *(Concurrent Versions System)* qui est un outil très bien adapté à ce type de problème. Il permet l'archivage/restitution de toutes les versions de chaque fichier source (ou documentaire ou script) ainsi que la gestion de la cohérence lorsque plusieurs développeurs travaillent en même temps sur un même logiciel. Il existe une documentation électronique interne à l'école pour prendre en main rapidement CVS
`http://perso-info.enst-bretagne.fr/ cvs/`.

Pour avoir des informations plus détaillées sur CVS et les produits associés, il existe plusieurs sites dont celui de *Cyclic*
`http://www.cyclic.com`.
On y trouve, en particulier, des clients CVS interactifs (fenêtres + boutons) pour toutes les plate-formes.

6.3.2 Utilisation de CVS du serveur de logiciels libres

Un serveur CVS est mis en place sur la machine dédiée au serveur de logiciel libre et spécifiquement pour ces logiciels. Il permet d'accéder aux fonctions et

[3] Informations à compléter dans cette annexe : localisation et structure des répertoires de téléchargement.

données de CVS à distance (le serveur est isolé des autres machines et ne partage pas un espace disque commun NFS comme les stations en libre service). Pour y accéder en lecture et écriture, il faut :

1. Demander à l'administrateur du serveur de créer un compte CVS personnel. Il transmettra le nom d'utilisateur et le mot de passe associé.
2. Définir sur le poste local la variable d'environnement
   `CVSROOT:pserver:<nom_utilisateur>@libre.enst-bretagne.fr:/CVS/LIBRE`
   (NB : en une seule ligne sans espace)
3. Se connecter / déconnecter à CVS par les commandes "`cvs login`" et "`cvs logout`". Le mot de passe est demandé au login.

Lorsqu'on est connecté, les commandes CVS sont opérationnelles comme si CVS était installé localement. Toute personne de l'école peut également accéder en lecture au CVS du serveur via un compte anonyme : Utilisateur "`anonymous`" mot de passe "`libre2000`". Pour changer un mot de passe utilisateur CVS, il faut passer par l'administrateur car il n'existe pas de commande associée.

CONSEIL 1 : Lorsqu'on utilise CVS, il est important de bien remplir les commentaires de changement de version dans les fenêtres éditeur qui s'affichent au moment des "cvs commit" de manière à conserver la trace des modifications et leurs raisons. Commentaires brefs et explicites.

CONSEIL 2 : CVS servira d'archiveur autant que de gestionnaire de version. Il faut donc y déposer tous les fichiers associés au logiciel qui seront nécessaires aux futurs repreneurs (scripts, docs...)

6.3.3 Utilisation de CVS par l'extérieur de l'école

Pour l'instant, CVS n'est accessible que depuis l'intérieur de l'école. Pour simplifier la gestion des comptes dans la phase actuelle de lancement du serveur, il n'y a qu'un *repository* contenant tous les projets (logiciels déposés sur le serveur). Ceci implique qu'un compte permet d'accéder à tous les projets de CVS sans distinction.

La structure d'accès est choisie en tenant compte du rapport simplicité / évolutivité. A priori, on peut imaginer trois phases d'exploitation de CVS avec le serveur libre :

1. (actuel) CVS est accessible en interne école seulement. Les paquetages téléchargeables *(tarball)* sont générés par le "club libre" à partir de CVS et placés dans le répertoire des téléchargeables du serveur.
2. (quand le club existera) CVS sera accessible de l'extérieur en lecture seule via le serveur par le compte anonyme. Ceci correspondra à la phase où des développeurs extérieurs participent à l'évolution des logiciels. Le retour des modifications externes s'effectue via le "club" qui enrichit CVS.
3. (plus loin ?) CVS sera accessible de l'extérieur en lecture et écriture. Ceci correspond à une phase où le club sera capable de gérer des développements collaboratifs avec l'extérieur (on n'y est pas).

La phase 3 ci-dessus nécessitera de modifier la structure de CVS pour que chaque projet ait un *repository* individuel afin de limiter les droits d'accès d'un utilisateur externe à un seul projet à la fois... Il faudra que le club soit déjà bien organisé pour bien gérer les multiples comptes CVS.

6.4 Préparation du téléchargement

Les logiciels sont téléchargeables via les pages html du serveur. Ils sont au format "archive compressée" *(.tar.gz)*. Pour cela, la procédure à suivre est la suivante :

1. Dans CVS, marquer les versions des fichiers sources qui vont composer le logiciel : commande *"cvs tag"* en utilisant un marqueur donnant le numéro de version du logiciel (ex : `version-1-2`).
2. Effectuer une extraction de CVS en utilisant le *tag*.
3. Réaliser le fichier d'archive compressé *(.tar.gz)* à partir de cette extraction.
4. Déposer l'archive compressée dans le répertoire de téléchargement du serveur.

6.5 Modification des pages html

Il reste à modifier l'information relative au nouveau logiciel ou à son changement de version dans les pages html du serveur. Les pages html sont gérées avec CVS sous le projet "`serveur_libre`". Pour modifier les pages et tracer les versions :

1. Générer une copie locale (machine utilisateur) du projet CVS gérant les pages html avec *"cvs checkout"*.
2. Modifier les pages concernées localement et vérifier la visualisation.
3. Mettre à jour la base CVS avec *"cvs update"* et *"cvs commit"*.
4. Sur le serveur, entrer en mode administrateur avec *"suprojet libre"* + mot de passe.
5. Aller dans la racine des pages du serveur de logiciel libre et faire une mise à jour des pages publiées avec *"cvs update"* (les pages publiées sont gérées comme une copie locale du projet CVS).

Ci-dessous une liste des points où l'on doit porter une modification (sous réserve de modifications dans la structure des pages du serveur) :

- Page de présentation des nouveautés.
- Page listant les logiciels disponibles sur le serveur (Index) avec les numéros de versions et des liens permettant le téléchargement. Vérifier également les informations présentées lors du téléchargement.
- Page de description des fonctionnalités du logiciel (accessible via la page précédente).
- Page des FAQ, voir si des questions se rapportent à la création ou la modification du logiciel et modifier les réponses en conséquence.
- Enfin, créer ou alimenter (donner l'information sur la modification) l'espace de discussion réservé au logiciel.