

Guide Méthodologique « Projets Open Source »

Projet	ODETTE
Auteurs	AVRAM Gabriela, GAUTHERON Laurent, LEIDNER Stefan, RAMEL Sophie Centre de Recherche Public Henri Tudor 29, Avenue John F.Kennedy L-1855 Luxembourg-Kirchberg Tél.: +352 42 59 91 - 1 Fax: +352 42 59 91 – 777 http://www.tudor.lu/
Date de création	07/09/04
Version et date	« 1.0 », 07/07/2005

Table des matières

1. Avant Propos du Guide.....	2
2. Introduction.....	4
2.1 Cycle de vie.....	5
2.3 Bulle structurelle d'un projet Open Source.....	6
3. Contexte économique de l'Open Source.....	7
3.1 La culture informatique de l'entreprise.....	7
3.2 Le marché.....	7
3.3 Les sociétés de service en Logiciel Libre.....	8
4. Analyse d'opportunités / Faisabilité.....	11
4.1 L'initiateur du projet et son environnement.....	11
4.2 Le projet et son contexte.....	11
4.3 Identification des utilisateurs.....	11
4.4 Facteurs et indicateurs de succès d'un projet Open Source.....	13
4.5 Choix de la licence.....	14
5. Spécifications et Évolutions des besoins.....	17
5.1 Spécifications générales.....	17
5.2 Spécifications détaillées.....	17
5.3 Gestion de l'évolution des besoins, risques et impacts.....	18
5.4 Pondération des exigences.....	18
6. Création et animation d'une communauté.....	21
6.1 Préparation de l'infrastructure.....	21
6.2 Le cadre organisationnel.....	24
7. Promotion du projet.....	31
8. Outils de gestion et de développement du projet.....	33
8.1 Outils de communication: différents types d'outils.....	33
8.2 Systèmes de gestion des versions.....	33
8.3 Outils de compilation et de déploiement.....	34
8.4 Systèmes de tests unitaires.....	34
8.5 Environnements de développement intégrés.....	35
8.6 Sites de gestion de projets Open Source.....	35
9. Le Développement.....	38
9.1 Communication sur les développements.....	38
9.2 Règles de programmation / Conventions de code.....	38
9.3 Gestion des versions.....	40
10. Processus de réutilisation de composants Open Source.....	42
10.1 Typologie et caractérisation des composants.....	42
10.2 Vue globale du processus de Réutilisation de composants.....	45
10.3 Phase 1 : Pré-sélection de composants.....	45
10.4 Phase 2 : Sélection et Modélisation des composants.....	46
10.5 Phase 3 : Réutilisation de composants.....	54
10.6 Phase 4 : Retours des composants modifiés / retour d'expérience.....	56

11	Phase de tests et de packaging.....	58
11.1	Tests.....	58
11.2	Packaging et release.....	58
12	Activités de support.....	60
12.1	Pratiques de support des projets Open Source.....	60
12.2	Demandes d'évolutions / rapports de bugs.....	61
12.3	Documentation/Guide utilisateurs.....	62
13	Références.....	63
14	Annexes.....	65
14.1	Répartition des licences sur sourceforge.net au 25/11/2004.....	65
14.2	Les licences de logiciels libres compatibles avec la G.P.L.....	66
14.3	Les licences de logiciels libres non compatibles avec la GPL.....	68
14.4	Questionnaire aux SSL.....	73

1. AVANT PROPOS DU GUIDE

Les notions de Logiciel Libre, de Logiciel Open Source, d'intégration de plate forme professionnelle Open Source, de réutilisation de composants Open Source ... toutes ces notions que l'on retrouve régulièrement dans la presse spécialisée, dans des séminaires, dans le cadre d'orientations politiques et que l'on a à intégrer dans son jargon et dans son capital de connaissances, et ce quelque soit sa fonction et son niveau de décision dans l'entreprise, ne sont pas évidentes à appréhender et surtout à mettre en oeuvre. La densité d'information sur ces problématiques est tellement considérable qu'au final nous ne savons plus bien identifier les tenants et aboutissants de cette « révolution » dans la façon de voir l'organisation, le développement et la distribution de tels développements logiciels.

D'où provient l'Open Source ? Quel en est son essence, ses principes, ses règles, ses droits (types de licences) ? A quels modèles organisationnels fait-il face ? Quel est son impact sur l'économie et l'organisation du marché ? Quels outils permettent d'accompagner sa réalisation ? Autant de questions à se poser lorsque l'on décide d'adopter un processus de développement logiciel basé sur l'Open Source. Autant de questions pour lesquels le décideur, le chef ou le promoteur du projet, ainsi que les développeurs du projet, ne disposent que de très peu d'outils d'aide pour appréhender l'entièreté des problématiques et des impacts et risques liés ...

Le but de notre guide est donc de proposer un outil d'aide et de support à destination d'utilisateurs plus ou moins initiés ayant des niveaux différents de décision et d'implication au sein de leur entreprise et leur projet Open Source ou libre. Il a l'objectif ambitieux d'être une aide pour mieux appréhender les questions cruciales liées à l'Open Source : Comment gérer la mise en configuration et l'animation d'un tel projet ? A quels critères porter attention dans le choix des licences ? Comment gérer le développement et quelles grandes règles respecter ? Comment réutiliser des composants ? etc.

Ce guide repose avant tout sur notre retour d'expérience sur l'organisation, l'animation et le développement du projet Open Source AnaXagora, une plate forme e-Business qui réunit des outils intégrés de e-learning, gestion des compétences, BPM et Knowledge Management (Vous pouvez trouver plus de détails à l'adresse suivante : <http://www.anaxagora.tudor.lu/>).

Nous avons voulu structurer ce guide suivant deux vues principales :

- une vue « Processus » permettant d'aborder l'ensemble des problématiques liées aux différentes phases de la gestion du projet Open Source, de l'étude d'opportunité jusqu'à la phase de mise en service et de maintenance: cette vue est précisée dans la partie Cycle de vie page 5 ;
- une vue « Systémique » permettant d'aborder l'ensemble des problématiques liées à l'environnement du projet, des différents acteurs et de leurs interactions, qui est précisée dans la partie Bulle structurelle d'un projet Open Source page 6.

Nous espérons que vous prendrez plaisir à lire ce guide et surtout qu'il vous fournira une base d'informations suffisamment claires et structurées vous permettant de mieux appréhender les problématiques et décisions que vous aurez à prendre dans le cadre de vos propres projets.

L'équipe de projet ODETTE

2. INTRODUCTION

Même si Open Source ne veut pas forcément dire gratuit, une majorité de logiciels Open Source sont disponibles gratuitement ou moyennant un coût très réduit.

Les développements de logiciels Open Source sont réalisés de façon collaborative, en communauté. Ainsi ces logiciels évoluent grâce à des centaines de contributions individuelles, ce qui permet une évolution rapide et des corrections rapides des problèmes, généralement plus rapidement que pour des logiciels commerciaux. De plus, ces évolutions sont le choix des utilisateurs eux-mêmes qui ainsi ne sont pas tributaires de la stratégie d'une entreprise développant un logiciel propriétaire.

Par ailleurs, les codes sources sont revus par plusieurs programmeurs ce qui permet d'atteindre un bon niveau de sécurité. Tout programmeur peut les modifier pour les étendre, personnaliser ou intégrer à une autre application.

Des logiciels comme Linux, Apache, ... prouvent que ce type de développement aboutit à des produits d'excellente qualité.

Sans entrer dans les détails, voici un tableau reprenant les avantages et inconvénients qui sont généralement avancés à propos des produits Open Source :

POUR	CONTRE
<i>Qualité (qualité technique, révision des bogues)</i>	<i>Finition (interfaces moins finies)</i>
<i>Réactivité (mises à jour et corrections fréquentes)</i>	<i>Risque de divergence (projet qui se scinde en plusieurs projets)</i>
<i>Pérennité (garantie par une communauté de développeurs suffisamment grande et active)</i>	<i>Image de marque (l'Open Source garde encore une image de produits « pas sérieux », « de bidouilleurs »)</i>
<i>Coûts (coût d'acquisition souvent gratuit ou réduit)</i>	<i>Manque d'un interlocuteur unique (morcellement de l'offre entre différents interlocuteurs potentiels)</i>
<i>Liberté (Indépendance des choix stratégiques de sociétés commerciales)</i>	<i>Copyleft (non exclusivité de la création logicielle)</i>
<i>Concurrence (prévient l'apparition de monopoles basés sur la fermeture du code source)</i>	
<i>Réutilisabilité (échange et réutilisation de composants)</i>	

Évidemment, ces arguments sont tous discutables et ne sont montrés ici que pour citer les grandes lignes de discussions des « pro » et des « anti » Open Source.

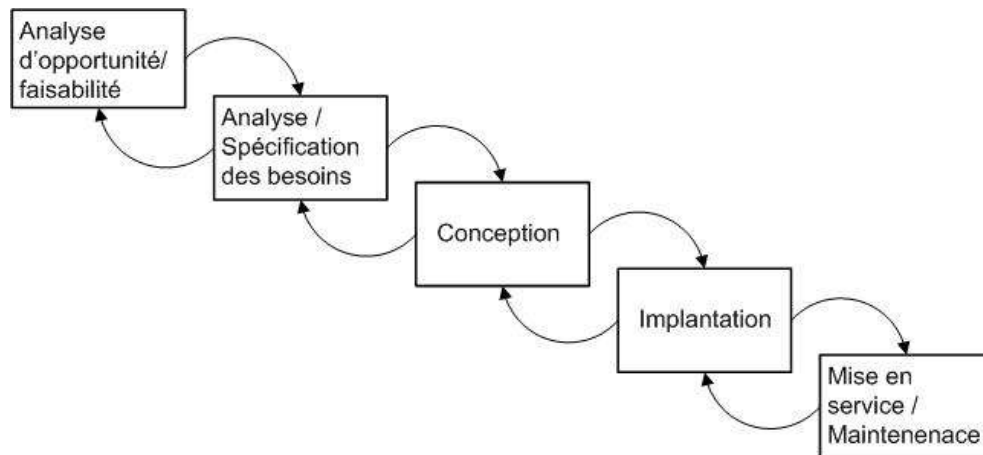
Historiquement, le logiciel libre a débuté avec Richard Stallman à la création de la Free Software Foundation en 1984 à l'origine du développement d'un système d'exploitation libre et complet nommé GNU (acronyme récursif pour « GNU's Not UNIX »). Le terme logiciel Open Source est utilisé par certaines personnes pour signifier plus ou moins la même chose que logiciel libre. En effet, le terme « free software » déplaît à certains, du fait que "free" signifie à la fois libre et gratuit. Or, un logiciel libre n'est pas toujours gratuit. Pour éviter le détournement du terme Open Source, une définition du concept a été réalisée dans le cadre de l'Open Source Initiative créée en 1998 par des membres actifs et reconnus de la communauté du logiciel libre (Todd Anderson, Chris Peterson, John maddog Hall, Larry Augustin, Sam Ockman et Eric Raymond).

2.1 Cycle de vie

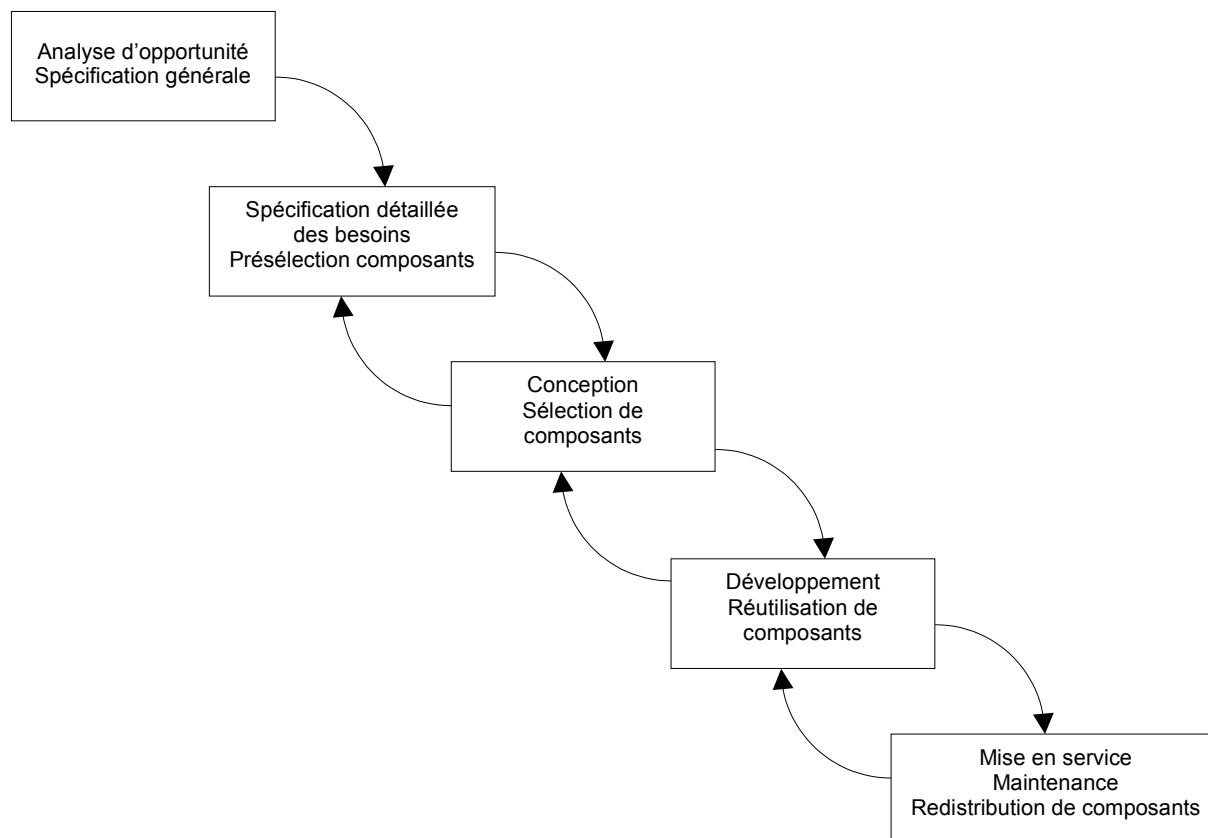
De manière à proposer un cycle de vie qui soit mieux adapté aux projets Open Source, nous avons choisi un cycle de vie simple (cycle de vie en cascade), que nous avons modifié pour y intégrer les éléments propres aux projets Open Source et à la réutilisation. Ce cycle de vie n'est qu'une proposition: nous aurions bien sûr pu utiliser un autre cycle de vie, mais avons choisi le cycle en cascade car c'est un des plus fréquemment utilisé.

Un autre cycle de vie aurait par exemple pu intégrer la notion de release fréquente, qui provient de l' « Extreme Programming » et qui est souvent vraie pour les projets open source.

2.1.1 Cycle de vie en cascade (Waterfall)



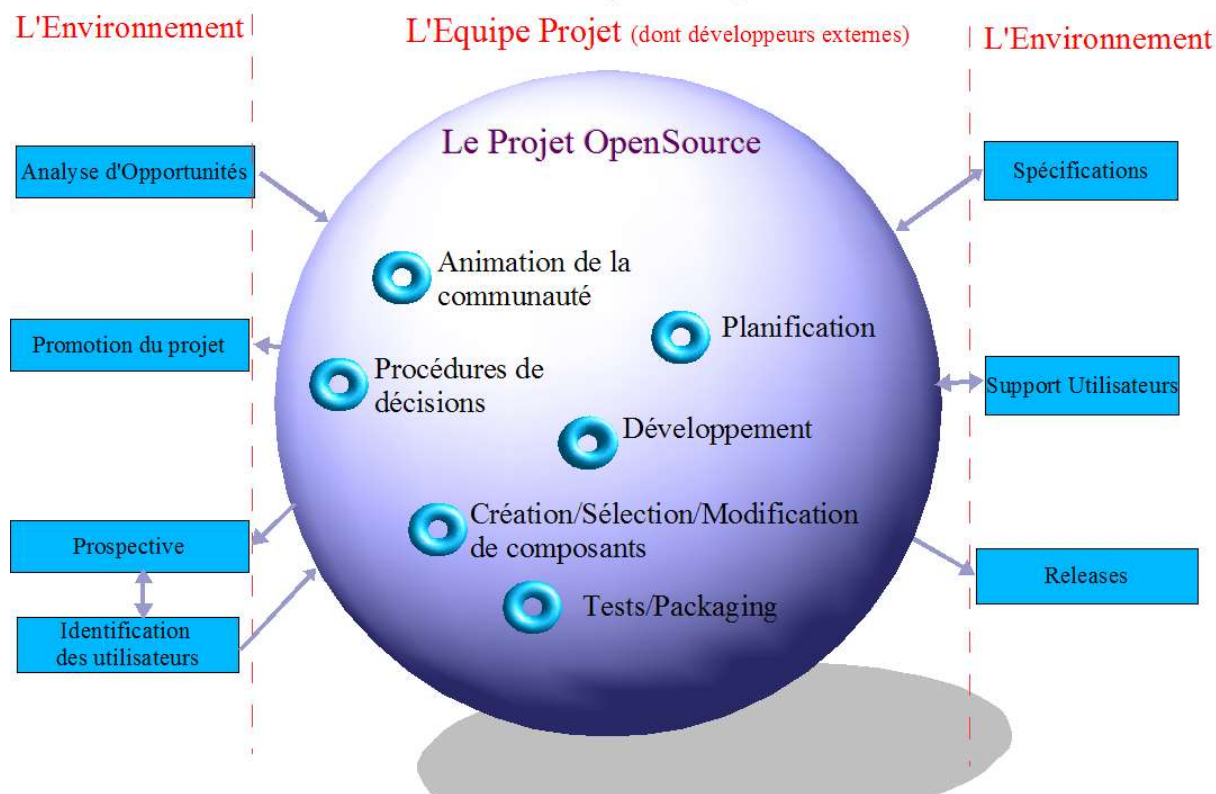
2.2.1 Proposition de cycle de vie en cascade adapté aux projets Open Source



2.3 Bulle structurelle d'un projet Open Source

Un certain nombre de points concernant la gestion et le déroulement des projets Open Source, et donc des parties de ce guide, ne figurent pas dans le cycle de vie du projet, qui est une vue trop chronologique. Nous avons donc réalisé un autre graphique, qui permet de montrer un découpage structurelle des différentes parties du guide méthodologique.

La « Bulle -Projet » OpenSource



3. CONTEXTE ÉCONOMIQUE DE L'OPEN SOURCE

Cette partie du guide méthodologique aborde certains aspects économiques de l'Open Source utiles à toute personne désirant créer un projet Open Source, avant de détailler le concept de Société de Services en Logiciel Libre (SSLL).

3.1 La culture informatique de l'entreprise

Une entreprise désirant se lancer dans un projet Open Source devra se poser quelques questions :

- les employés sont-ils familiarisés avec l'Open Source en générale?
- les employés maîtrisent-ils les techniques nécessaires au développement Open Source (CVS, ...)?
- est-on prêt à développer en collaboration avec des organisations externes?
- le développement de logiciel libre fait-il partie de la stratégie de l'entreprise?

3.2 Le marché

Il faudra mener une véritable veille concurrentielle tant du côté logiciels « propriétaires » :

- Que font les concurrents?
- Que font les logiciels du marché?
- Quels sont les fonctionnalités offertes?

que dans le domaine du logiciel Open Source:

- Existe t-il des logiciels Open Source dans le même domaine?
- Que font-ils?
- En quoi mon projet est-il différent? Où se situe le valeur ajoutée?

C'est étape devra être faite de façon approfondie et pourra être complétée par des tests et des évaluations des logiciels mais également de la réactivité des équipes projets ainsi qu'une vérification de la fréquentation et de la mise-à-jour des sites des projets.

Cette veille peut vous permettre d'affiner votre idée du projet avec ce qu'il doit comporter ou non, ainsi que de mieux définir les moyens de promotion de votre site internet.

Cette étude pourra aussi remettre en cause de l'opportunité de votre projet initial et pourquoi pas déboucher sur son abandon ou encore sur la participation à un projet Open Source existant. Il existe trop souvent des projets Open Source redondants.

3.3 Les sociétés de service en Logiciel Libre

3.3.1 Définition

Un certain nombre de SSII (Sociétés de Services et d'Ingénierie Informatique) ont adapté leurs services vers des solutions Open Source. D'autres sociétés sont quant à elles apparues en se positionnant directement dans une stratégie de distribution, d'intégration ou d'adaptation de solutions Open Source pour ses clients.

Afin de ne pas être confondues avec les activités des SSII classiques, activités connotant bien souvent des relations privilégiées de la société avec des éditeurs de logiciels propriétaires, ces sociétés ont pris la dénomination de sociétés de services spécialisées dans les logiciels libres (SSLL). Ce terme de SSLL a été initié par le PDG de la société Linagora, Alexandre Zapolsky et est depuis le terme couramment utilisé dans la presse spécialisée informatique et commence même à faire partie progressivement du jargon du marché. Il a l'avantage d'avoir une connotation orientée « services » mais dans un contexte « Open Source ».

3.3.2 Offres de service / Avantage concurrentiel / Modèle économique ?

Ces SSLL ont un modèle économique différent en ce sens qu'elles orientent leur système de production vers les communautés de développeurs et non pas vers les éditeurs de logiciels propriétaires. C'est également le cas des SSII qui travaillent occasionnellement sur des projets Open Source.

Malgré un business model différent, les SSLL offrent des services relativement similaires à ceux offerts par les SSII. Si on se réfère à la nomenclature de services offerts par les SSII et identifiée par la Syntec en France (Chambre Syndicale des SSII et des Éditeurs de Logiciels), trois catégories de prestations émergent :

- Les prestations intellectuelles recouvrant le conseil, l'audit, les études d'architecture de systèmes informatiques, l'intégration des systèmes, l'ingénierie, la maintenance, l'assistance technique et la formation.
- Les prestations machines recouvrant les réseaux, les services liés à valeur ajoutée et l'infogérance.
- Les prestations de conception de logiciels

Les services offerts par les SSLL sont très comparables et ce n'est pas ce qui les différencie fondamentalement des SSII classiques.

Par contre, les SSLL n'ont pas d'activité d'édition (de logiciels métiers par exemple) en ce sens que leur but n'est pas de faire leur marge sur la vente de licences.

Les concurrents de ces SSLL sont alors bien les SSII et non pas les SSLL elles-mêmes. En effet, les SSLL sont bien sûr concurrentes entre elles mais pas de la même manière que vis à vis des SSII. Même si une SSLL remporte un marché sur une SSLL concurrente, elle contribuera d'un certain côté à la promotion et à l'augmentation de la crédibilité de l'ensemble des acteurs de ce marché. On parle alors d'un schéma de « coopération » et non de compétition: le terme « coopération » est un mélange entre « coopération » et « compétition » qui illustre ce principe.

Les SSLL se différencient également par une stratégie d'extrême adaptabilité des services et prestations offertes, ceci grâce à la nature même du logiciel libre basée sur des composants modulaires, adaptables, interopérables et réutilisables permettant une offre diverse en adéquation avec les besoins spécifiques des clients avec des économies d'échelle au niveau du développement.

3.3.3 Modèle organisationnel

La SSLL se positionne entre la communauté et le marché. Le rôle de l'expert SSLL étant bien de rendre commercialisable et d'un certain côté d'augmenter la crédibilité ces logiciels sur le marché. La communauté n'a pas forcément l'envie ni les compétences nécessaires pour le faire. Bien que l'implication et la participation de la SSLL dans la communauté soit très variable, une relation gagnant-gagnant-win se construit par le fait que la communauté aura de toute façon toujours un retour bénéfique, une aura quant à son image de marque par rapport aux implantations du logiciel dans les entreprises. Un certain nombre de communautés proposant des plate formes de CMS comme Zope ou OpenCMS ont tiré ainsi une grande aura de leurs différentes intégrations chez des grands comptes et administrations : un phénomène de « lobbying » s'est finalement développé ...

Concernant la gestion des salariés de SSSL, le système est un peu différent du schéma classique en ce sens ou bien souvent cette gestion consiste en un arbitrage entre règles et libertés. Fixer des règles bien sûr concernant des dead-lines sur les résultats à produire tout en fournissant au salarié une certaine liberté d'action quant à ses horaires et la nature du travail qu'il effectue dans l'entreprise. Ainsi bon nombre de SSSL ont conscience que les salariées passent du temps à travailler sur des projets personnels pendant les heures de bureau mais elles savent très bien que ces activités permettent en quelque sorte de s'assurer que le salarié fait ainsi une sorte de veille technologique et est ainsi à l'écoute des toutes dernières évolutions technologiques. Selon la société française Alcôve, 10 à 20% du temps de la société sur des projets libres.

3.3.4 Synthèse des réponses à un questionnaire à destination des SSSL

Cette synthèse concerne les grandes idées tirées des réponses au questionnaire à destination exclusive des SSSL (voir Questionnaire aux SSSL en annexe). Vous trouverez les résultats complets du questionnaire en annexe. Pour cette raison, il convient d'interpréter ces résultats dans la seule vue des sociétés d'intégration en logiciel libre et ainsi éviter de tirer des conclusions hâtives d'un marché beaucoup plus complexe dans lequel les SSII se positionnent de plus en plus à l'orée de l'activité des SSSL.

Les Maîtres mots: qualité, personnalisation, souplesse et conviction

A La question « Pourquoi n'avoir choisi d'intégrer que des composants Open Source ? », la majorité des SSSL soulignent la capacité d'innovation pour les entreprises liées à l'introduction de solutions et composants Open Source ceci en terme

- de qualité/fiabilité/pérennité de ces produits logiciels (par rapport à certaines solutions propriétaires),
- de personnalisation de ces produits aux besoins des clients/entreprises
- de souplesse par rapport à la palette offerte quant au choix de composants existants
- d'indépendance financière et commerciale vis à vis d'éditeurs ou de fournisseurs

Mais une des raisons majeure et essentielle est l'aspect « Conviction », conviction et croyance de la SSSL en la culture et le phénomène Open Source et communautaire.

La contribution au monde Open Source ... une composante essentielle : le débogage

Concernant la contribution des SSSL aux projets et composants Open Source qu'elles intègrent, la majorité d'entre elles soulignent leur participation active à la phase de débogage de ces composants et ainsi à leur pérennité. Une partie des SSSL participe au développement de nouveaux composants dans le cadre de frameworks ou plate formes Open Source (tel que Zope) et les mettent à disposition de la communauté.

SSII et SSSL : deux mondes et modèles différents

La plupart des SSSL considèrent leur approche business radicalement différente du modèle des SSII dans le sens ou ces dernières s'appuient plus volontiers sur des solutions propriétaires et ne produisent des composants applicatifs que sur demande express du client.

Les réponses soulignent le degré d'expertise plus élevé offert par les SSSL dans le cadre de leurs prestations notamment en terme de fiabilité, personnalisation et adaptabilité et sécurité des solutions mises en place chez le client.

Un nouveau modèle économique

L'ensemble des SSSL consultées est unanime : l'Open Source infère un nouveau modèle économique. Un nouveau modèle économique qui remet notamment en cause le pouvoir et un certain monopole qu'on put détenir les éditeurs qui sont du même coup obligé de trouver et de proposer de nouveaux panels de solutions et surtout de nouvelles valeurs ajoutées aux clients (notamment en terme de personnalisation des solutions et de leur adéquation aux besoins des clients).

Être à l'écoute des clients, avoir un haut niveau d'expertise technique

Pour la majorité des SSSL consultées, les qualités essentielles d'un consultant en logiciel libre sont :

- en premier lieu, l'écoute du client ;
- une connaissance et expérience du contexte de l'Open Source;
- de solides connaissances en développement et en chefferie/direction de projet.

4. ANALYSE D'OPPORTUNITÉS / FAISABILITÉ

Cette étape primordiale va dépendre de plusieurs paramètres en fonction du type d'«initiateur» du projet et de la nature du projet lui-même. Les paramètres à prendre en compte seront différents s'il s'agit d'une personne individuelle ou s'il s'agit d'une entreprise qui lance le projet.

Les paramètres à prendre en compte dépendant du projet sont :

- les objectifs recherchés au travers de ce projet (besoin, pas de solution Open Source, ...)
- le contexte du projet.

4.1 *L'initiateur du projet et son environnement*

Si l'initiateur du projet Open Source est une entreprise, les critères à prendre en compte seront assez semblable à un projet classique :

- l'environnement de l'entreprise:
 - évolution des marchés
 - concurrence
 - législation (licence)
 - tendances au niveau du secteur d'activité, ...
- le métier de l'entreprise:
 - activité
 - services/produits
 - savoir-faire humain et technique, ...
- la culture de l'entreprise (histoires, valeurs, ...).

Pour une personne individuelle les éléments sont différents:

- un besoin personnel,
- affirmer sa notoriété / réputation, son expertise du domaine,
- conviction, idéologie,
- apprentissage,
- amusement, défi,
- ...

4.2 *Le projet et son contexte*

Il est préférable de bien identifier un certain nombre de point avant le lancement du projet comme:

- les enjeux stratégiques du projet,
- les conditions de déroulement (étapes, ...),
- la description de la solution attendue (périmètre fonctionnel, aspects techniques),
- l'analyse des expériences passées,
- les autres projets en cours dans différents domaines,
- les contraintes internes et externes relatives au projet,
- les risques.

Il peut s'agir de trois types de projets :

- Développement d'application Open Source,
- Intégration d'application Open Source (distribution, support, installation),
- Personnalisation d'application Open Source.

Il existe également un cadre particulier pour la distribution, d'intégration ou de personnalisation de solutions Open Source, nommé « Société de Service en Logiciel Libre ».

4.3 *Identification des utilisateurs*

De manière globale, nous pouvons différencier quatre types d'utilisateurs :

- Les utilisateurs cibles
- Les utilisateurs pilotes
- Les utilisateurs potentiels
- Les utilisateurs « Lambda »

4.3.1 Identifier le marché les utilisateurs cibles

Tout d'abord il s'agit d'identifier les marchés cibles, ainsi que le type et le niveau des innovations attendues par les parties prenantes du projet Open Source.

De cette étude, un certain nombre d'utilisateurs cibles va pouvoir être identifiés parmi lesquels certains seront amenés à être des utilisateurs dits « pilotes »

4.3.2 Identifier et gérer les utilisateurs pilotes

Idéalement les utilisateurs pilotes sont en avance sur les tendances. Comment définir la tendance ? Pour le savoir, la meilleure méthode est de parler avec les experts du terrain ayant une vision large des technologies émergentes et des applications de pointes dans le secteur étudié.

Prenons par exemple le projet Open Source OpenSST, ce projet a pour but de développer un protocole pour la sécurisation des transactions électroniques. Son organisation repose sur un partenariat entre les sociétés Aubay, Conostix et CRP Henri Tudor et ouvert à tous contributeurs externes intéressés. Il a donné lieu à la réalisation de deux prototypes : l'un développé en Java pour la sécurisation de transactions "on-line", l'un développé en "Perl" pour la sécurisation de transactions (type signature de documents) en mode "off-line".

Dans le cadre de ce projet, une certaine expertise du domaine des transactions électroniques et des pratiques inhérentes en terme de leur sécurisation dans les domaines bancaires et financiers principalement a dû être capitalisée avant d'envisager le départ concret du projet.

Il s'agit ensuite de commencer un travail de réseau pour identifier les utilisateurs à l'avant-garde du marché cible et des marchés qui lui sont liés ceci afin principalement de :

- collecter les informations qui permettront d'identifier des innovations particulièrement prometteuses et les idées conduisant au développement de produits innovant.
- donner une première forme à des suggestions de produit,
- évaluer le potentiel économique de ces concepts ainsi que leur adéquation aux intérêts du métier, de la branche ou du secteur d'activité donné.

Au niveau de la méthodologie d'animation, on peut organiser un groupe de travail avec plusieurs utilisateurs pilotes, une demi-douzaine de spécialistes du domaine avec des fonctions différentes (marketing, communication, technique, administratif) , et l'équipe-projet elle-même représentée par le correspondant projet. Ce genre de groupe de travail dure en général un ou deux jours. Pendant ce laps de temps, les participants travaillent d'abord en petits groupes puis tous ensemble pour donner forme finale aux concepts qui correspondent très précisément aux besoins de la communauté .

Il est à noter que pour que ce groupe de travail soit un succès, il est indispensable d'avoir identifié et impliqué les utilisateurs pilotes auparavant. Si ce n'est pas le cas, le groupe de travail risque de ne pas rassembler les utilisateurs pilotes et donc de ne pas aboutir (c'est le cas par exemple du groupe de travail mis en place au début du projet AnaXagora, et qui avait comme objectif d'identifier les besoins du marché pour ce projet).

Après ce groupe de travail, l'équipe projet peaufine encore un peu plus les concepts, s'assure qu'ils correspondent bien aux besoins des utilisateurs visés (par une validation) , et finalement présente ses recommandations à la structure décisionnelle du projet Open Source.

Cette animation peut-être faite idéalement en phase d'étude d'opportunité (cf section « Analyse d'opportunités ») du projet afin de tirer les principaux termes de références ... dont les exigences seront à affiner par la suite (cf section « Spécifications et évolutions des besoins »).

4.3.3 Identifier et gérer les utilisateurs potentiels

Lors de l'étude de la cible, un certain nombre d'utilisateurs a pu être repérés comme n'étant pas directement dans la cible du projet mais pour lesquels le projet pourrait avoir tout de même un impact pour leur travail, leur environnement.

Ce sont des utilisateurs qu'on désigne par « utilisateurs potentiels ».

Comment les intégrer au projet ? Essentiellement par une démarche prospective, c'est à dire par des activités de communication et de sensibilisation qui permettront à la fois :

- de recueillir des besoins et contraintes supplémentaires de ces utilisateurs,
- de permettre une synergie pouvant permettre l'initialisation d'une sorte de lobbying par le biais d'un phénomène que l'on pourrait appeler, comme désigné dans l'article publié dans la *Harvard Business Review* s'intitulant "L'innovation chez 3M" (Eric von HIPPEL, Stefan THOMKE et Mary SONNACK,) « La pyramide de l'expertise » et qui se fonde sur les réseaux pour identifier d'autres utilisateurs potentiels, cibles ou pilotes d'abord dans le marché cible puis dans d'autres domaines.

En quoi consiste la démarche prospective ?

Faire de la prospection auprès des utilisateurs potentiels identifiés, c'est principalement organiser un ensemble d'activités permettant de communiquer autour de ce projet par le biais par exemple :

- de présentations un-à-un des concepts, de la problématique, des enjeux avec l'utilisateur potentiel;
- de présentations de type conférence/séminaire voir groupe de travail professionnel;

Comme stipulé ci-dessus, les objectifs majeurs de la prospection sont d'une part de prendre en compte des contraintes supplémentaires au niveau du projet et de ses exigences et d'autre part de stimuler un phénomène de « réseau » par du « bouche à oreille », des publications ou autres moyens de diffusion.

Cette technique de prospection a été engagée dans le cadre du projet OpenSST. Un certain nombre d'activités a été mené :

- Des contacts téléphoniques ou mails vers des entreprises pouvant potentiellement être intéressées ou impactées par le domaine de la sécurisation des transactions électroniques.
- Une présentation concrète de la problématique de transaction et de sécurisation avec une présentation des concepts et standards existants (SSL, XML-SIG, ...) , leurs forces, limites et faiblesses, ceci afin de positionner concrètement l'apport du projet OpenSST dans cette jungle technologique de la sécurité. A noter qu'il est toujours essentiel d'avoir une argumentation solide à présenter pour justifier les avantages de l'Open Source. Par exemple, dans le cadre du projet OpenSST, voici les points que nous avons soulevés concernant les avantages du choix d'une solution de sécurité en Open Source :
 - Relecture multiple du code
 - Anticipation du débogage
 - Réactivité de l'Open Source
 - Réactions plus rapide de la correction d'un bug

4.3.4 Les utilisateurs « Lambda »

Ces utilisateurs « Lambda » sont une catégorie bien spécifique à une communauté Open Source. Ces utilisateurs peuvent être :

- des utilisateurs « vagabonds » qui se sont intéressé à l'outil au hasard de leur navigation;
- des utilisateurs « ponctuels » qui ont téléchargé une release pour leur propre usage sans réel objectif de participer ou de s'immiscer dans le développement collaboratif;
- des utilisateurs « feed-back » qui en plus de télécharger la release vont donner un retour par le biais des forums/mailings lists de la communauté;
- des utilisateurs « engagés » intéressés par une participation au développement du produit Open Source.

Ces utilisateurs sont très importants car ils sont des vecteurs importants de la communication de l'existence du projet par un phénomène de diffusion « réseau ». Par contre, ce sont les plus difficiles à gérer car on les rencontre très rarement en un-à-un et en présentiel et il est donc d'autant plus important de soigner la réactivité par rapport à leurs questions, interrogations ou reports de bugs. Par exemple le projet peut vite se trouver diffuser une mauvaise image par la faute d'un utilisateur déçu.

4.4 Facteurs et indicateurs de succès d'un projet Open Source

Cette partie présente quelques facteurs reconnus comme importants pour le succès des projets Open Source, suivis de quelques indicateurs permettant de percevoir son niveau de réussite, ainsi que des risques récurrents pour les projets Open Source.

Facteurs :

- Leadership du projet
- Nombre de développeurs et sa croissance
- Nombre d'utilisateurs et sa croissance
- Activité / Avancement
- Ouverture du projet (à d'autres participants)
- Nombre de projets basés sur celui-ci
- Qualité de la documentation (site Web, manuels, changelogs,...),
- ...

Indicateurs :

- Nombre de téléchargements
- Fréquentation des forums de discussion
- Activité des mailing-lists / nombre d'inscrits
- Nombre de bugs rapportés / résolus
- Références d'entreprises utilisatrices
- taille de la communauté
- ...

Les principaux risques qu'un projet Open Source puisse rencontrer sont :

- le manque de participation externe,
- mésentente pouvant mener à l'éclatement de la communauté en plusieurs projets,
- mauvaise image de marque du projet,
- amateurisme dans la gestion du projet,
- mauvaise éthique des membres du projet

On peut citer comme projets Open Source phares :

- Linux (18 millions d'utilisateurs estimés en 2001 selon Linux Counter, et une très forte progression),
- Apache (près de 70% de part de marché en juin 2005 d'après Netcraft),
- Perl (plus d'un million d'utilisateurs selon « Perl Journal »),
- PHP (près de 7 millions de sites internet d'après Netcraft en out 2001),
- OpenOffice,
- Mozilla,
- MySQL,
- ...

4.5 Choix de la licence

Dans l'esprit du grand public, les expressions « domaine public », « logiciel libre », « Open Source » désignent la même chose. En réalité, il n'en est rien: Il existe un grand nombre de licences, très différentes les unes des autres.

L'expression « logiciel libre », traduction de « free software » est très utilisée. Elle sous-entend que le programme soumis à une telle licence puisse être utilisé, copié, étudié, modifié et redistribué. Par la suite, pour éviter que le code source ne soit récupéré et transformé en code propriétaire à des fins lucratives, Richard Stallman a défini une licence sous le nom de « GPL » (GNU General Public License). Le principe essentiel de la GPL est le concept de « copyleft ». Le code source doit être librement accessible, diffusée et modifiable. L'auteur conserve son droit d'auteur, et chaque développeur qui l'enrichit conserve le droit d'auteur de sa modification. Toute modification doit se faire sous licence GPL. Par exemple, même si vous n'utilisez qu'une partie du code GPL dans un développement, celui-ci devra être diffusé sous licence GPL.

L'expression « Open Source » désigne l'ensemble des licences conformes à l'OSD (Open Source Definition) défini par l'OSI (Open Source Initiative, <http://www.opensource.org>): ces licences sont Open Source dans la mesure où elles demandent également la distribution des sources et autorisent leurs modifications, mais elles ne contiennent généralement pas de notion de « copyleft » et peuvent contenir d'autres contraintes plus spécifiques, par exemple sur les modes de redistribution. Le logiciel libre, quant à lui, est soutenu par la FSF (« Free Software Foundation », fondation pour le logiciel libre en français, <http://www.fsf.org>)

4.5.1 Définitions

Tout d'abord voici quelques définitions des termes afin de mieux les différencier:

– logiciel libre

Le logiciel libre est un logiciel fourni avec l'autorisation pour quiconque de l'utiliser, de le copier, et de le distribuer, soit sous une forme conforme à l'original, soit avec des modifications, gratuitement ou non. Pour être libre, un logiciel doit respecter la liberté d'exécution, d'étude, de redistribution des copies, d'amélioration.

Cela signifie en particulier que son code source doit être disponible.

– Open Source

Un logiciel libre est toujours Open Source, mais pour être considéré comme Open Source par l'OSI, un logiciel doit répondre à ces 10 critères :

- redistribution du programme libre et gratuite,
- livraison du code source avec le programme,
- distribution des travaux dérivés dans les mêmes termes que la licence d'origine,
- préservation de l'intégrité du code source de l'auteur,
- absence de discrimination envers des personnes ou des groupes,
- absence de discrimination envers des domaines d'activité,
- pas besoin de se conformer à des termes de licences complémentaires,
- pas de licence spécifique à un produit,
- pas de licence imposant des restrictions sur d'autres logiciels,
- neutralité vis-à-vis de la technologie utilisée.

– logiciel propriétaires:

Ce sont des logiciels qui sont distribués en version « exécutable », non modifiables, en comparaison aux logiciels libres et Open Source sont fournis avec leur « code source » modifiable. Cette catégorie inclue également les freewares et les sharewares.

– freeware:

Logiciel que son auteur a choisi de rendre absolument gratuit, soit parce qu'il désire le tester, soit parce qu'il désire en faire profiter la communauté. Ces logiciels deviennent parfois payants dans une phase commerciale. Cela n'exige pas que le logiciel soit Open Source.

– shareware:

Il est fondé sur le concept du "libre-essai". Il présente pour l'utilisateur l'avantage de pouvoir essayer le produit sans avoir à l'acheter. Si le produit vous plaît, vous êtes invité à acheter sa licence d'utilisation. Dans le cas contraire, vous avez l'obligation légale et surtout morale d'ôter le logiciel de votre ordinateur. Certains sharewares ne proposent que des fonctionnalités limitées tant que l'utilisateur n'a pas acheté le logiciel.

– Droit d'auteur:

En France, « l'auteur d'une oeuvre de l'esprit jouit [...] d'un droit de propriété [...] », sans formalité sa vie durant et pour une durée correspondant à l'année civile du décès de l'auteur et des soixante-dix années qui suivent, au bénéfice de ses ayant-droits.

Ainsi, il est essentiel lors de toute utilisation d'un programme ou d'une partie d'un programme d'avoir le consentement de son auteur, au risque sinon d'être condamné à payer des dommages et intérêts pour contrefaçon (sauf dans le cas de programmes Open Source ou libres).

Lorsqu'une personne travaille au nom d'une entreprise, le droit d'auteur est alors au nom de l'entreprise.

– copyright:

Le terme « copyright » désigne la notion de droit d'auteur dans la loi américaine. Aux États-Unis, il est conseillé de déposer le copyright, surtout en cas de litige. Les oeuvres protégées peuvent afficher le symbole ©, suivi de l'année de publication et de l'auteur.

– copyleft:

Le copyleft (gauche d'auteur) permet de rendre une oeuvre (ou programme) libre et d'obliger que toutes versions modifiées ou étendues soient libres également. Cela s'étend notamment à la notion de dépendance: tout programme dépendant d'un programme sous une licence de type copyleft doit également être soumis à une licence équivalente.

– domaine public:

On entend par logiciel du domaine public un logiciel qui n'est pas soumis aux droits d'auteurs. Cela ne veut pas dire qu'il soit Open Source, car dans certains cas, l'exécutable est dans le domaine public alors que le code source n'est pas disponible. D'autre part, si le

source est dans le domaine public, il est possible que certaines copies modifiées ne soit pas libre (pas de copyleft).

4.5.2 Critères de choix de licences

Les licences les plus répandues sont la GNU General Public License (GPL), la GNU Lesser General Public License (LGPL), ainsi que la licence BSD.

Néanmoins, il existe de nombreuses autres licences, comme le témoigne le tableau récapitulant les licences utilisées par sourceforge.net par nombre de projets (voir annexe Répartition des licences sur sourceforge.net au 25/11/2004).

Cependant, la tendance actuelle est au regroupement de licences équivalentes ou proches. On conseille donc d'utiliser une des licences les plus connues et de ne pas définir sa propre licence sauf en cas de besoin spécifique.

La principale différence entre ces licences est le fait que le code puisse être transférable dans un programme propriétaire ou non, avec ou sans modifications.

Nom	GPL	Transférable sans modification	Transférable avec modification
Academic Free License	<i>incompatible</i>	<i>oui</i>	<i>oui</i>
Apache 2.0	<i>incompatible</i>	<i>oui</i>	<i>oui</i>
Apple Public Source License	<i>incompatible</i>	<i>oui</i>	<i>non</i>
License artistique	<i>incompatible</i>	<i>oui</i>	<i>avec restrictions</i>
License artistique modifiée	<i>compatible</i>	<i>oui</i>	<i>avec restrictions</i>
Common Public License	<i>incompatible</i>	<i>oui</i>	<i>oui</i>
Eclipse Public License	<i>incompatible</i>	<i>avec restrictions</i>	<i>avec restrictions</i>
IBM Public License	<i>incompatible</i>	<i>avec restrictions</i>	<i>avec restrictions</i>
Intel Open Source License	<i>compatible</i>	<i>oui</i>	<i>oui</i>
Python License v2	<i>compatible</i>	<i>oui</i>	<i>oui</i>
Sun SSISSL	<i>incompatible</i>	<i>avec restrictions</i>	<i>avec restrictions</i>

Il est préférable de choisir une licence assez répandue et reconnue (par exemple la GPL si vous souhaitez garder la notion de copyleft), ensuite le choix dépendra des éventuelles contraintes spécifiques de votre projet (copyleft, redistribution avec ou sans modifications, ...).

Pour plus d'information sur ces licences, voici la liste des sites web de certaines d'entre elles :

Nom	Adresse du site
<i>Apache Software License</i>	<i>www.apache.org</i>
<i>Artistic License</i>	<i>www.perl.com</i>
<i>BSD License</i>	<i>www.berkeley.edu</i>
<i>General Public License (GPL)</i>	<i>www.gnu.org/fsf</i>
<i>Library or "Lesser" Public License (LGPL)</i>	<i>www.gnu.org/fsf</i>
<i>MIT License</i>	<i>web.mit.edu</i>
<i>Mozilla Public License (MPL)</i>	<i>www.mozilla.org</i>
<i>Zope Public License (ZPL)</i>	<i>www.zope.org</i>

5. SPÉCIFICATIONS ET ÉVOLUTIONS DES BESOINS

La définition des spécifications doit permettre de s'approprier les besoins exprimés par les utilisateurs et, inversement, aux utilisateurs de s'assurer que leurs besoins ont bien été compris. Bien sûr, ces besoins doivent être conceptualisés en tenant compte des contraintes liées à l'implémentation et la mise en place de la future application.

5.1 Spécifications générales

Les spécifications générales de la future application (appelées bien souvent aussi « termes de référence ») doivent être définies afin de cadrer et de préparer le document de spécifications détaillées. Il permet alors de mieux mesurer l'avancement réel des spécifications.

Ces spécifications générales sont la plupart du temps de grandes lignes directrices identifiées lors de l'analyse d'opportunité et validées par la structure décisionnelle et stratégique du projet.

Par exemple, dans le cadre du projet Open Source « OpenSST », voici les exigences générales au niveau technique qui avaient été définies :

1. *OpenSST doit pouvoir fonctionner sous les protocoles suivants : HTTP, SMTP*
2. *OpenSST doit pouvoir permettre de crypter toutes formes de données passées dans la transaction (exécutable, ascii, formats propriétaires, etc.)*
3. *OpenSST doit permettre l'authentification des différents acteurs de la transaction*

5.2 Spécifications détaillées

Selon le cycle de développement choisi, des spécifications détaillées peuvent être définies.

D'une manière générale, il est conseillé de recourir systématiquement à des spécifications détaillées pour préciser tous les points flous présents dans les spécifications générales.

Idéalement, on peut considérer les spécifications comme suffisamment détaillées lorsqu'il est possible de rédiger des jeux de tests directement à partir de celles-ci, sans aucune ambiguïté.

Aucune méthodologie de spécifications (UML, MDA, Merise...) n'est véritablement imposée même si UML reste le langage le plus utilisé par la communauté Open Source, de même que pour les logiciels propriétaires.

Ces spécifications détaillées sont bien sûr recueillies auprès des différents utilisateurs identifiés (cf chapitre dédié) ainsi que par de la prospection (cf chapitre dédié).

Dans le cadre de OpenSST, la phase de prospection a permis de rencontrer un certain nombre d'acteurs clés du domaine des transactions (transactions bancaires, financières, industrielles) pour des domaines métiers différents (banques, industries, instituts financiers, corps de métiers, ...) etc. Chaque interview de prospection a permis de récolter des exigences détaillées sur base des contraintes organisationnelles ou technologiques du client. Voici un exemple de fiche rédigée dans le cadre de l'interview des besoins dans le domaine des transactions électroniques d'un cabinet d'avocat que nous appellerons « avocats.com » sur Luxembourg.

prospection :
FICHE DE SYNTHÈSE
 INTERVIEW : AVOCATS.COM

Description de la société

avocats.com est un cabinet d'avocats d'affaires établi au Luxembourg qui fournit des services juridiques à une clientèle nationale et internationale.

Le cabinet a développé une forte expertise en fusions-acquisitions, banques et assurances, opérations de financement, droit commercial, droit des sociétés, fiscalité, droit du travail, propriété intellectuelle et industrielle, droit aérien.

Applications E-Business et Besoins en sécurité transactionnelle

Dans le cadre des activités du cabinet, une problématique récurrente est de pouvoir partager et diffuser des contrats ou des mandats avec le client. Actuellement, cet échange se fait avec certains clients par messagerie électronique mais avec le désavantage de ne pas avoir de garanti quant à la signature des documents transmis (problème d'authentification, d'encryption des données et également de timestamping).

Le cabinet utilise également un autre logiciel « BBM Connect » permettant de faire du dépôt de marque on-line. Ce système utilise une solution propre pour garantir un canal de sécurité des transactions passées.

Le cabinet va également s'équiper d'une application business de gestion de cabinet d'avocats « AVONCA » permettant de gérer notamment des agendas partagés, la gestion de contacts, la gestion de la facturation, un module de comptabilité etc.

5.2.1 Apports possibles de OpenSST

- *L'opportunité détectée se situerait essentiellement au niveau de la signature électronique de ces contrats et mandats échangés par messagerie électronique avec un certain type de clientèle.*
- *D'autres opportunités ont également été détectées par Mr xxxx mais plus difficile à implémenter de part l'architecture. Ainsi Mr xxxx a paru intéressé de disposer de systèmes d'authentification fort sur des sites d'intermédiation tel que « escrow.com » (utilisé pour de la consignation de services avec des tiers pour la réservation de noms de domaine) utilisant déjà https mais dont la sécurité apportée est très transparente pour l'utilisateur. Il a tenu à souligner d'ailleurs cet aspect psychologique dans l'aspect sécurité des transactions électroniques : l'essentiel est selon lui de fournir une bonne information des destinataires d'un tel système.*

5.2.2 Synthèse des exigences et contraintes identifiées

1. *OpenSST doit pouvoir fonctionner sous les protocoles suivants : HTTP, SMTP*
2. *OpenSST doit permettre de crypter toutes formes de données passées dans la transaction (exécutable, ascii, formats propriétaires, etc.)*
3. *OpenSST doit permettre l'authentification des différents acteurs de la transaction*

5.3 Gestion de l'évolution des besoins, risques et impacts

Comment gérer l'évolution des besoins ?

Tout d'abord il faut bien catégoriser le besoin : attente ? besoin perçu ? besoins réels ?

Face à cette évolution des besoins, des contraintes (organisationnelles, technologiques, financières, etc) se posent. Afin de ne pas plonger dans l'immobilisme, il faut faire une évaluation des risques rigoureuse et réaliste pour mieux appréhender l'impact de ces changements.

On commence à aborder la gestion des risques dès la phase de lancement (cf étude d'opportunité) en analysant les principaux risques identifiés dans l'étude préalable.

Par la suite, une organisation spécifique dédiée à la gestion des risques peut être mise en place au sein de l'équipe du projet.

5.4 Pondération des exigences

5.4.1 Principe

La pondération des exigences doit permettre de classer les différentes exigences formulées dans le cahier des charges afin de faire ressortir les exigences qui sont fondamentales pour le projet et celles dont l'intérêt est moindre.

C'est sur cette base que seront évaluées et planifiées les fonctionnalités à implémenter dans le cadre d'une release, ainsi que les composants à réutiliser: cette étape est donc primordiale pour la spécification des besoins.

5.4.2 Choix d'une échelle de pondération

La pondération peut se faire selon plusieurs échelles. Nous en distinguons deux intéressantes dans le cadre de la pondération des exigences : une échelle linéaire et une échelle logarithmique.

L'échelle linéaire

Une échelle linéaire est une échelle de pondération définie entre deux bornes (de 1 à 10 par exemple), dont l'intervalle entre chaque pondération sur l'échelle est constant (1 unité par exemple). L'avantage de cette échelle est qu'elle correspond à une notation classique et est donc assez intuitive. Toutefois, cette échelle pose le problème de la quantification des exigences. En effet, il est parfois difficile de quantifier avec précision une exigence pour pouvoir la positionner par rapport à une autre. Bien souvent les pondérations utilisées sont des standards (0...1...5...7...9...10) et ne permettent pas de hiérarchiser les besoins de manière précise : quelle est l'importance d'une exigence pondérée 8/10 par rapport à une exigence pondérée 7.5/10 ?

L'échelle logarithmique

Une échelle logarithmique est une échelle où la pondération la plus forte est une puissance (au minimum supérieure à 3) de la pondération la plus faible.

Qualificatif	Pondération
Stratégique	5
Très Important	2
Important	1
Peu Important	0,5
Accessoire	0,25

- Exemple d'échelle Logarithmique -

L'avantage de cette échelle est de pouvoir quantifier une exigence à partir d'un critère qualitatif. En affectant à chaque poids (maximum 5 poids dans l'échelle) un qualificatif on peut plus facilement quantifier et faire quantifier une exigence. Du fait de la pondération logarithmique, il est plus facile de dégager les exigences fondamentales des exigences de pure forme.

L'inconvénient de cette échelle est qu'elle nécessite une explication des qualificatifs employés : par exemple une exigence « stratégique » est une exigence dont l'entreprise ne peut se passer pour fonctionner, un besoin « très important » est un besoin dont on peut se passer, mais dont la lacune handicaperait lourdement l'entreprise...

L'utilisation de cette échelle de pondération, si elle permet de hiérarchiser plus facilement les exigences, nécessite néanmoins l'initiation –à défaut de l'accompagnement– des groupes utilisateurs en charge de la pondération.

Extrait de pondération d'exigences dans le cadre du développement d'un outil BPM Open Source sur base de l'échelle logarithmique présentée ci-dessus

Modéliser suivant un formalisme standard (Pondération : IMPORTANT)

Modéliser graphiquement les BP selon un formalisme standard.

La modélisation graphique des BP devra être conforme à une représentation graphique standard (ex:Qualigram, UML,...).

Différents formalismes pourront être utilisés éventuellement.

Ce formalisme pourra représenter les différents objets nécessaires à la définition des BP.

Exporter dans un format standard (Pondération : STRATEGIQUE)

Modéliser les flux (Pondération : TRES IMPORTANT)

Les flux devront pouvoir être représentés dans un diagramme. Par flux, on entend les informations (documents, outils, objet d'une base de données) en entrée et en sortie des étapes ou activités d'une procédure.

Le formalisme permettra d'expliciter si une information est en entrée (input) d'une activité, en sortie (output) ou en entrée-sortie (in/output).

Modéliser l'enchaînement des étapes (Pondération : TRES IMPORTANT)

Un diagramme devra permettre de modéliser et visualiser graphiquement le séquençage des étapes d'une procédure et de définir les différents itinéraires possibles dans la procédure.

Ce diagramme comportera des chemins avec des conditions permettant de choisir un itinéraire suivant une condition, des chemins parallèles, des itérations et des synchronisations.

Définir des sous-processus (Pondération : IMPORTANT)

Décomposer un processus en sous-processus

Afin de simplifier la représentation graphique ou de pouvoir ré-utiliser des parties du modèle, un processus pourra être décomposé en sous-processus.

Cette décomposition pourra être sur plusieurs niveaux ; c'est-à-dire qu'un sous-processus pourra lui aussi être composé de sous-processus.

Valider le modèle (Pondération : TRES IMPORTANT)

5.4.3 Stratégie de Pondération

L'identification des exigences est définie idéalement sur la base d'entretiens ou de groupes de travail (GT), menés et animés par un membre de l'équipe du projet Open Source (le chef de projet ou un analyste de l'équipe projet) que nous désignerons par « correspondant utilisateur », et impliquant les utilisateurs cibles (cf « Organisation et Structures d'un projet Open Source »). La pondération des exigences quant à elle doit revenir à ces groupes utilisateurs, accompagnés par le correspondant utilisateurs.

L'échelle de pondération est choisie en accord entre le « correspondant utilisateur », les utilisateurs eux-mêmes et éventuellement le commanditaire ou la direction du projet si elle est clairement identifiable dans l'organisation des structures du projet Open Source.

Ensuite, lors de l'entretien avec chaque groupe utilisateur, le « correspondant utilisateur » explique l'intérêt de la pondération ainsi que l'échelle utilisée. Puis chaque exigence identifiée est passée en revue et pondérée collectivement par le groupe utilisateur.

Une fois que tous les groupes utilisateurs ont pondéré les exigences, une synthèse est faite et transmise à la structure décisionnelle du projet (direction, comité de pilotage ou autre) qui décide de la pondération finale en cas de pondérations divergentes sur l'une ou l'autre des exigences.

6 CRÉATION ET ANIMATION D'UNE COMMUNAUTÉ

Pour créer une communauté de développeurs, un certain nombre d'éléments doit être pris en considération:

- l'infrastructure du projet, en particulier le cadre de communication;
- le cadre organisationnel (la définition des règles de participation, des rôles, des processus);

La communauté doit également avoir un but bien défini; pour cela, il faut fournir une description claire du logiciel libre qui sera développé.

6.1 Préparation de l'infrastructure

6.1.1 Choisir la solution pour héberger le projet

Il existe plusieurs possibilités pour héberger le site internet collaboratif de votre projet:

- utiliser un site consacré à l'hébergement de projets Open Source: Asynchrony, BerliOS, freemasonry, Gforge 2, Savannah, SourceForge (le plus connu), Tigris, OpenChannel, etc. Pour une comparaison détaillée, consultez <http://www.ibiblio.org/fosphost/exhost.htm>;
- avoir son propre site (auquel il faut alors rajouter des outils collaboratifs, soit directement sur le site soit à part)
- les deux (en utilisant certains outils fournis par un site d'hébergement, et d'autres sur son propre site, par exemple si les outils mis à la disposition par le site d'hébergement ne sont pas suffisants)

Les avantages du placement du projet sur un site Open Source bien connu sont liés à la visibilité du projet et à l'emploi des outils déjà familiers aux développeurs: CVS, forums, listes de discussion. Le groupe cible est formé des développeurs avec plus ou moins d'expérience, intéressés de participer au développement ou simplement de tester le logiciel. Il est plus facile de faire un projet remarqué sur un site qui a déjà un public important intéressé par le développement du logiciel libre que de promouvoir un nouveau site dédié au projet. C'est pourquoi nous vous recommandons dans tous les cas d'au moins héberger votre projet sur le site SourceForge, <http://sourceforge.net>.

A noter que quand on parle ici de l'hébergement d'un projet, cela signifie l'hébergement des sources du projet ainsi que des outils collaboratifs pour la gestion du projet: de nombreux autres sites peuvent diffuser les versions exécutables du produit, mais le site qui héberge le projet et sa communauté est généralement unique ou bien divisé entre le site de l'hébergeur et un autre site.

6.1.2 Description du projet

Une description du projet doit être publiée sur la page d'accueil – le texte sera adapté au groupe cible (Note sur les groupes cibles: dans les projets Open Source la frontière entre les développeurs et les utilisateurs est beaucoup plus floue que dans le cas des logiciels propriétaires – chaque utilisateur intéressé pouvant se transformer à un moment donné en développeur). Cette description du projet devra contenir des informations très claires sur l'idée du projet, son but, les licences employées, la personne, le groupe ou l'entreprise qui a lancé le projet, les utilisateurs potentiels, les technologies et langages utilisés.

6.1.3 Choix des fonctionnalités du site web

Une autre décision est **le choix des fonctionnalités qui seront mises à disposition des développeurs et des utilisateurs du site:**

- **listes de discussion** (sur sourceforge, les projets ont souvent des listes séparées pour les développeurs et pour les utilisateurs);
- **forums publics;**
- **weblogs-** Un weblog (ou blog) est un site web proposant un journal en ligne tenu par une ou plusieurs personnes. Les weblogs se composent d'un carnet de bord recensant les hyperliens (vers des pages Web) que l'auteur a jugé intéressants, accompagnés de commentaires concernant ces pages. Le point commun de toutes les formes de weblogs est qu'on y retrouve à intervalles irréguliers les impressions de l'auteur du blog sur des sujets variés. Les éléments que l'on retrouve souvent sur un blog sont: un système d'archivage des articles selon des catégories prédéfinies ou selon la date de parution, un système de commentaires article par article, une liste de liens, tout particulièrement vers d'autres blogs. Les articles les plus récents sont généralement affichés en tête de page. Les articles des blogs peuvent être liés entre eux

via des « Trackback ». Une manière de créer une communauté autour de soi, un outil pour fédérer des membres d'une association, les weblogs :

- peuvent générer immédiatement des discussions, qui enrichissent les propos de son auteur ou l'éclairent sur ce qu'il dit;
- sont très ciblés, ils attirent donc une audience qualifiée pour le sujet dont parle(nt) l'auteur(s), comme un certain type de logiciel;
- les weblogs permettent de suivre les tendances en temps réel.

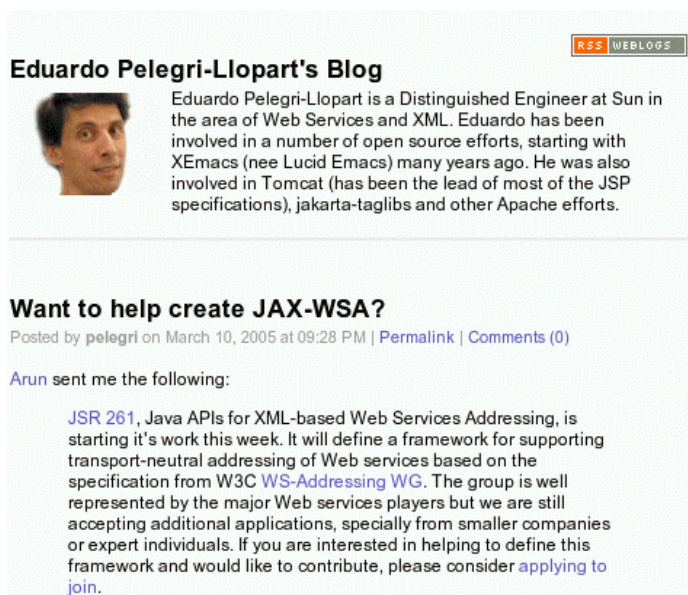


Figure 1 Eduardo Pelegri-Llopart's Blog

- **RSS** (Rich Site Summary ou Really Simple Syndication) - une famille de protocoles de syndication de contenu sur Internet, utilisant la technologie XML, utilisés principalement par les sites Web d'actualités et les weblogs. Il est à noter que plusieurs technologies RSS ont été développées indépendamment les unes des autres, ce qui fait qu'il existe plusieurs significations pour le même acronyme **RSS**. Sourceforge offre des RSS pour les projets hébergés sur son site: cela signifie que les visiteurs qui veulent être tenus au courant sur le progrès d'un ou de plusieurs projets peuvent s'abonner aux RSS, au lieu de revenir chaque fois sur les pages des projets pour vérifier s'il y a des nouveautés.
- **wiki**- Un wiki est la façon la plus simple de collaborer avec d'autres personnes. Les participants peuvent éditer des pages, les catégories et les principes d'organisation. Cet outil facilite le travail en collaboration sur des documents, la préparation des événements ou la coordination des projets. Voici la définition de Wikipedia (une encyclopédie basée elle-même sur un système de wiki): Un **wiki** est un site web dynamique dont tout visiteur peut modifier les pages à volonté. Il permet non seulement de communiquer et diffuser des informations rapidement (ce que faisait déjà Usenet), mais de structurer cette information pour permettre d'y naviguer commodément. Il réalise donc une synthèse des forums Usenet, des FAQ et du Web en une seule application intégrée (et hypertexte)
- **Serveur FTP** (File Transfert Protocol – protocole de transfert de fichiers) - On peut s'en servir pour envoyer et récupérer des fichiers utilisés dans le cadre du projet. Il faut créer une arborescence de répertoires spécifique aux besoins du projet sur le serveur ftp.
- **CVS Repository** - **CVS**, acronyme de **Concurrent Versions System** ou de **Concurrent Version(ing) System**, est un logiciel libre de gestion de versions. Un **logiciel de gestion de versions** permet de stocker dans un lieu donné des informations pour une ou plusieurs ressources informatiques permettant de récupérer toutes les versions intermédiaires des ressources, ainsi que les différences entre les versions. Ce genre de système est généralement utilisé pour stocker les différentes versions du code source des projets.
- **L'IRC**(Internet Relay Chat) et la **messagerie instantanée**(Instant Messaging)– peuvent être aussi utilisés pour assurer la communication instantanée. Plusieurs études ont montré que l'utilisation des moyens de communication synchrone (IRC, IM) pour résoudre des conflits et pour prendre des décisions importantes peut contribuer à la création d'une vraie communauté.

- Un **système de bookmarks collectifs** comme Del.icio.us: **Del.icio.us** est, à l'origine, un système de tagging social qui permet aux utilisateurs de sauvegarder et de classifier une collection personnelle de bookmarks. Tous les utilisateurs peuvent avoir accès aux bookmarks créés par une personne, et ils peuvent aussi utiliser les RSS pour s'abonner aux listes des bookmarks d'autres personnes. del.icio.us peut être utile aussi pour le travail en collaboration quand on a besoin de maintenir une liste de ressources commune pour le groupe. Pour faire ça, il faut premièrement créer un nouveau compte sur del.icio.us ([http:// del.icio.us](http://del.icio.us)), qui recevra un nom (peut-être le nom du projet même)- dans notre cas, c'était AnaXagora. Puis, on doit choisir d'un commun accord des catégories, pour que toute l'équipe accepte d'utiliser les mêmes catégories. Dans notre cas, on a crée deux catégories: EntreprisesLL (pour mémoriser les coordonnées des entreprises qui travaillent en Libre) et SitesOS (pour garder les URL d'autres sites dédiés au logiciel libre).

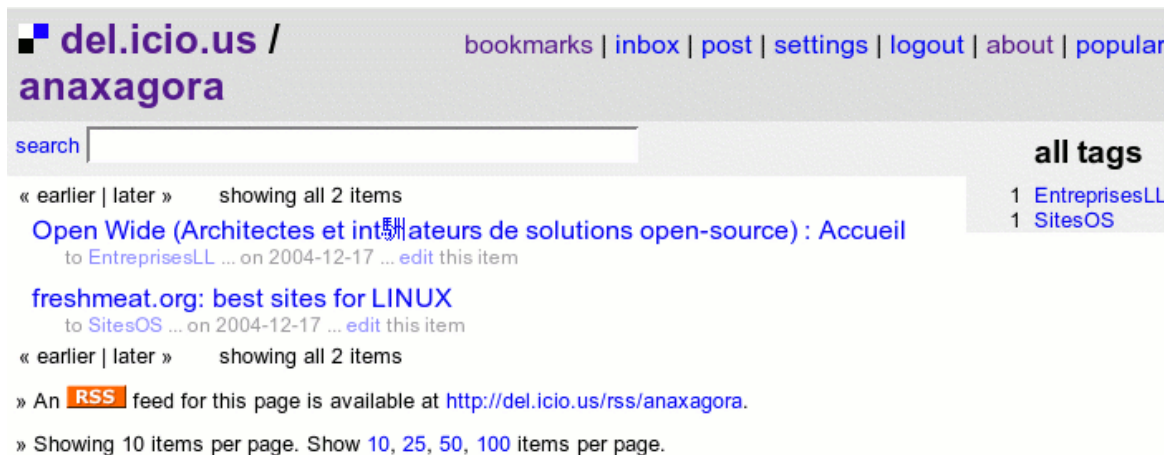


Figure 2 Page del.icio.us pour AnaXagora

Les adresses collectées de cette façon peuvent être affichées sur le site du projet en utilisant le RSS produit par del.icio.us pour chaque catégorie. Les membres du groupe peuvent s'abonner eux-mêmes aux RSS en utilisant un « news aggregator », pour être tenus au courant. Un service similaire est offert par Furl (<http://www.furl.net/index.jsp>).

Il n'est pas nécessaire d'utiliser la totalité de ces outils. Il suffit généralement de commencer avec le minimum indispensable, et de rajouter des nouveaux outils au fur et à mesure que des besoins spéciaux se manifestent.

A noter qu'il est fortement conseillé d'utiliser des outils Open Source pour réaliser ces fonctionnalités. Le fait d'utiliser des outils propriétaires dans le projet, même pour de la gestion, peut en effet être mal perçu par les participants. Les outils de développement seront présenter dans le chapitre 6.

6.1.4 Préparation du lancement du site du projet (sur sourceforge ou un autre)

Quelle que soit la solution choisie, on doit préparer le lancement du site en apportant une attention particulière aux éléments suivants:

- *la description du projet – voilà un exemple :*

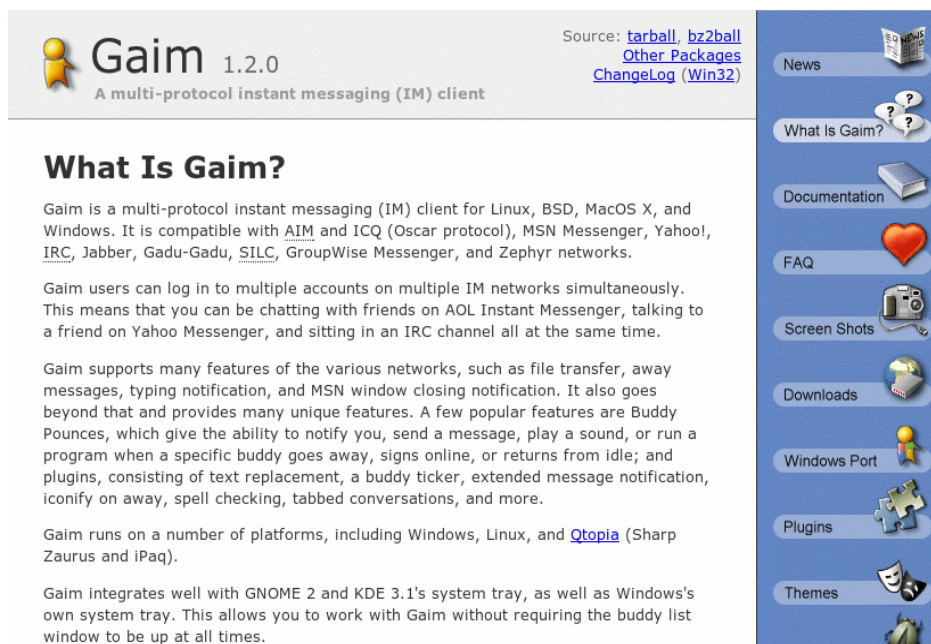


Figure 3 Page de description du projet Gaim

- *la présentation de l'équipe- des courtes notes, des photos;*

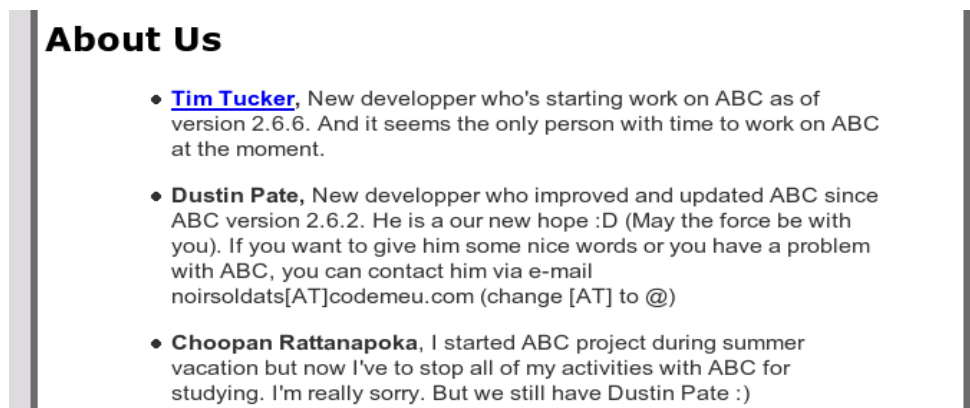


Figure 4 Participants au projet "pingpong-abc"

- *la présentation des règles de participation.*

6.2 Le cadre organisationnel

Par définition des projets Open Source, les règles d'organisation sont souvent minimalistes: seules les règles minimales permettant à la communauté de bien fonctionner sont définies. Il n'est donc pas rare qu'il n'y ait aucune hiérarchie parmi les développeurs d'un projet Open Source. Cependant, lorsqu'un projet concerne plus de personnes, la coordination devient plus compliquée à gérer, et une hiérarchie plus détaillée est souvent nécessaire. Les « gros » projets sont donc généralement plus structurés que les petits projets.

Contrairement à ce qui peut se rencontrer ailleurs, ce sont les développements du projet qui induisent généralement des changements organisationnels et pas le contraire: en effet, la communauté est conduite par les besoins en termes de développements, et c'est souvent seulement quand un blocage pour l'avancée ou le bon déroulement de ces développements se produit que les autres aspects sont considérés.

6.2.1 Les rôles du projet

La chose la plus importante dans ce contexte est de gérer les ressources existantes, dont les plus importantes sont probablement le temps et les compétences des participants.

Avant de lancer le projet, il faudra aussi définir **les différentes rôles** qu'on va attribuer aux participants. Cette définition doit être flexible - elle pourra évoluer avec le temps.

Les rôles attribués aux participants de projets Open Source peuvent être très divers. Il y a des superpositions entre les rôles : parfois les développeurs doivent s'occuper aussi de l'analyse, de l'architecture ou de la maintenance du logiciel. Les administrateurs du projet peuvent être en même temps des gestionnaires, des lobbyistes, des porte-paroles pour le projet. Voici une liste non exhaustive de quelques rôles qui couvrent la majorité des activités d'un projet Open Source:

Administrateur(s) du projet (propriétaire du code). *Les projets Open Source dépendent considérablement de ressources qui sont des ressources partagées. Les administrateurs doivent s'assurer que ces ressources sont disponibles, doivent se préoccuper des problèmes de sécurité, et doivent assurer les outils nécessaires pour le projet. Leur rôle est de faire fonctionner le projet. Ils doivent avoir la vision d'ensemble du projet et s'assurer que le projet ne va pas trop s'éloigner de ses objectifs. Ils sont aussi impliqués dans les activités de lobbying et leurs démarches pour obtenir les fonds nécessaires pour l'infrastructure du projet sont indispensables.*

Gestionnaires des modules- *ce sont les personnes en charge du développement de chaque module. Leur rôle est de coordonner l'activité et de s'occuper de résoudre les conflits possibles.*

Membres de l'équipe – *ce sont les développeurs-clés, qui prennent en charge la plupart du travail. Normalement, ce sont les initiateurs du projet. En tenant compte de la participation de développeurs occasionnels dans le projet (la durée de leur stage, leurs mérites, etc), on peut recruter certains d'entre eux dans l'équipe du projet.*

Développeurs occasionnels – *ce sont des personnes qui contribuent avec des patches ou développent des modules qui implémentent des nouvelles fonctionnalités de bon gré. Ils peuvent aussi être impliqués dans la traduction du logiciel dans d'autres langues, dans la rédaction de la documentation ou dans d'autres tâches connexes.*

Utilisateurs/testeurs – *la plupart des participants à un projet Open Source entrent dans cette catégorie. Ce sont des gens qui ont soit besoin des fonctionnalités implémentées par le logiciel et le téléchargent pour l'utiliser, soit veulent apprendre à écrire du code. Une infime partie d'entre eux va probablement se transformer en participants occasionnels, ne se limitant pas à signaler des erreurs, mais en suggérant aussi des patches (« bug fixers »).*

Il est à noter que la structure organisationnelle des projets Open Source est généralement assez peu hiérarchique, ce qui correspond mieux à la philosophie Open Source. Les rôles ne sont donc pas toujours clairement définis pour les projets, surtout si le projet est petit. Seul le rôle d'administrateur (aussi appelé chef de projet ou responsable du projet) existe dans tous les projets, même si pour certains petits projets ce rôle est implicite.

Exemple de rôles détaillés:

Pour le projet ArgoUML (<http://argouml.tigris.org/>), il existe différents types de rôles, en plus du découpage en sous systèmes. Les rôles possibles annoncés sur le site sont:

- *Developer role*
- *Editor of the User FAQ*
- *Mailing list moderator*
- *Back-up moderator of a mailing list*
- *Member administrator*
- *Editor of the Cookbook*
- *Release Responsible for all Releases*
- *Editor for the User Manual and Quick Guide*
- *Subsystem responsible*

6.2.2 La gestion du projet

Les gestionnaires des modules doivent avoir une représentation au moins approximative des ressources disponibles : les développeurs, leurs compétences individuelles et le temps qu'ils pourront passer à travailler sur le projet.

Pour l'inventaire des compétences existantes, sourceforge offre une matrice des compétences (skill inventory). Chaque personne a un profil des compétences sur ses connaissances en programmation. La majorité des utilisateurs choisit de restreindre l'accès à leur profil.

Skill Inventory

Skill	Level	Experience
Topic :: Software Development :: Object Oriented	Wizard	2 yr - 5 yr
Topic :: Software Development :: Object Oriented	Wizard	2 yr - 5 yr
Programming Language :: XSL (XSLT/XPath/XSL-FO)	Competent	6 Mo - 2 yr
Topic :: Formats and Protocols :: Data Formats :: HTML/XHTML	Wizard	5 yr - 10 yr
Topic :: Software Development :: Modeling	Competent	6 Mo - 2 yr
Programming Language :: ASP	Competent	2 yr - 5 yr
Database Environment :: Database API :: SQL-based	Competent	5 yr - 10 yr
Topic :: Text Editors :: Documentation	Wizard	> 10 years
Topic :: Text Editors :: Documentation	Wizard	> 10 years
Topic :: Security	Wizard	> 10 years
Topic :: System :: Networking	Wizard	> 10 years
Programming Language :: Java	Wizard	2 yr - 5 yr
Programming Language :: Perl	Competent	> 10 years

Figure 5 Exemple de "skill inventory" sur sourceforge.net

La majorité des projets Open Source a une base fluctuante de participants. Cela rend l'attribution des tâches aux participants encore plus difficile. La mise en place des matrices de compétences peut permettre aux participants de s'encadrer eux-mêmes dans des divers échelons de compétence, et aux autres participants de rajouter leurs avis à ce sujet. Pour le moment, sourceforge n'offre pas la possibilité de rajouter les opinions d'autres participants sur les compétences d'un certain participant.

L'estimation du budget de temps du projet se fait en fonction de la disponibilité déclarée approximativement par les développeurs pour travailler dans le cadre du projet. Pour une planification des développements, on compare le temps estimé nécessaire et les compétences avec l'existant, en tenant compte du fait que les développeurs indépendants vont choisir eux-mêmes les modules sur lesquels ils veulent intervenir. Cette planification ne peut être que floue, parce qu'elle implique des ressources qui ne sont que probables. S'il s'agit d'une entreprise qui travaille en libre, une planification un peu plus précise peut être élaborée, car elle peut s'assurer de la disponibilité d'au moins une partie des ressources.

On devra tenir compte en permanence des motivations spécifiques aux développeurs Open Source. Ces motivations peuvent être:

- Répondre à un besoin propre – ils envisagent une future utilisation du logiciel par eux-mêmes ou par leur entreprise ;
- Apprendre des nouvelles technologies, se perfectionner dans un langage de programmation, apprendre à travailler dans une équipe, etc ;
- Améliorer sa réputation, se faire connaître, la possibilité de rajouter la participation à des projets Open Source dans son CV;
- Participer au développement d'un produit qui sera mis gratuitement à la disposition de tout le monde;
- Gagner de l'argent en distribuant le logiciel et en offrant des services autour de celui-ci.

Même si elle est presque similaire à la motivation des développeurs de code propriétaire, la motivation des développeurs Open Source payés par une entreprise a quand même quelques particularités: ils ont le sentiment d'appartenance à la communauté Open Source, ils partagent la volonté de mettre le logiciel à la disposition de tout le monde. Parfois, cela se matérialise par une contribution individuelle volontaire au-delà des heures payées.

Pendant **le déroulement du projet**, il est nécessaire d'assurer la maintenance de l'infrastructure, la rédaction de la documentation, l'archivage des listes de discussions. De nouvelles fonctionnalités peuvent être rajoutées si on en perçoit la nécessité. Le cadre organisationnel du projet doit rester flexible et permettre des changements en cours de route pour les rôles, les règles de participation, etc.

6.2.3 L'animation du site et du projet

Techniques d'animation

Les techniques d'animation qui peuvent être utilisées sont très diverses. Elles proviennent en majorité du groupe de techniques employées normalement dans des communautés en ligne. Comme spécificité, on devra tenir compte en permanence du fait que les participants à un projet Open Source ont des motivations distinctes et un code éthique particulier.

Le rôle des animateurs

Les animateurs (qui peuvent être à la fois administrateurs du projet, gestionnaires des modules, ou d'autres membres de l'équipe) doivent:

- connaître les participants ;
- connaître l'état du projet, les problèmes ;
- intervenir pour faciliter la résolution des conflits ;
- répondre aux questions ou les renvoyer vers les personnes qui peuvent y répondre ;
- maintenir la communauté vivante, en employant les outils disponibles et en tenant compte des motivations spécifiques aux développeurs Open Source ;
- faciliter le travail en équipe ;
- entretenir la motivation pour continuer, en assurant une atmosphère de travail agréable, une transparence qui permet aux participants passifs ou arrivés plus tard la possibilité d'apprendre, et en exposant une gestion de projet garantissant la livraison d'un projet utile et de qualité ;
- promouvoir le projet sur l'Internet, dans des séminaires et conférences, par actions de communication, relations avec l'industrie, etc.

Comportement de l'animateur

- une vision claire sur l'architecture et les fonctionnalités du logiciel développé donne aux participants confiance dans la capacité des leaders ;
- l'exemple personnel des administrateurs et des gestionnaires des modules compte énormément; leurs énergie et ardeur peuvent beaucoup inspirer et mobiliser les autres participants ;
- une communication claire et transparente, régulièrement résumée et archivée donne le ton pour une bonne atmosphère de travail ;
- l'évaluation continue et la réflexion sur les progrès réalisés permet d'entretenir l'intérêt et la motivation des membres du projet.

Tâches devant être remplies par les animateurs

- Pour faciliter le fonctionnement et la communication dans le groupe, écrire des synthèses du travail devant être réalisé, et relancer si besoin le travail ;
- Il est essentiel que le groupe apprenne ensemble à améliorer son fonctionnement. Dans ce but, quand il s'agit d'une équipe distribuée, l'animateur peut essayer d'organiser de temps en temps des réunions via un outil de messagerie instantanée comme IRC, pour faire le bilan et partager des idées d'amélioration ;
- L'animateur doit veiller à ce que chacun des membres de l'équipe du projet y trouve satisfaction en fonction de ses besoins et intérêts ; Même s'il est bien sûr impossible de toujours contenter tout le monde, on doit toujours au moins essayer de résoudre une partie des problèmes.

Outillage de l'animation

La mise à disposition du code et de la documentation est un facteur d'incitation à la participation ou à l'utilisation du projet, mais l'ajout d'informations supplémentaires sur le développement du projet lui-même dans des blogs ou sur les forums permet de rendre le projet encore plus intéressant. Avoir la chance de percevoir des êtres humains au-delà du code mis à la disposition du public, pouvoir se documenter sur le progrès du projet et les principaux problèmes rencontrés apporte plus d'attractivité au projet. On doit trouver le moyen de montrer aux visiteurs du site du projet qu'ils sont les bienvenus, et que l'on tient compte de leurs opinions.

L'utilisation de blogs, la publication des RSS des blogs et du site du projet peuvent aider énormément les personnes intéressées par le projet à se tenir au courant. L'animateur peut activement chercher des blogs ou des sites similaires pour y publier des commentaires, ou peut commenter leurs articles sur son propre blog en utilisant le mécanisme de *trackback* pour amener des visiteurs sur le propre blog. Maintenir un weblog du projet n'est pas du tout facile, cela nécessite la volonté et le temps pour faire ce travail supplémentaire, ainsi que l'aisance de communiquer et de faire des synthèses dans la langue choisie.

L'utilisation des wikis donne une chance aux participants du projet de contribuer leurs propres idées sur les développements futurs. Même si cette façon de communiquer est très utile en principe, elle peut impliquer beaucoup de travail de la part des animateurs, qui peuvent se voir inondés par des propositions pour des nouvelles fonctionnalités plus ou moins utiles, ainsi que par des tentatives de changement du cours du projet et de sa vision d'ensemble. En fonction de la popularité du projet, le volume de travail pour discuter de ces propositions et pour maintenir une vision claire du projet peut être assez important.

On doit également organiser une veille régulière sur les mentions du projet sur l'Internet. Pour cela, on peut créer des alertes sur Google, Technorati et d'autres sites spécialisés, pour être alertés quand le nom du projet ou de ses développeurs est cité.

6.2.4 Communication sur le projet

Il faut gérer la communication avec chaque individu qui montre un intérêt pour le projet. Ils sont un capital social déjà existant qu'il faut garder. Des réponses promptes et aimables aux e-mails reçus, l'offre d'un abonnement à une newsletter, à une liste de discussions ou au RSS du projet peuvent faire revenir régulièrement les visiteurs occasionnels sur le site du projet.

Tout commentaire sur un blog ou forum doit recevoir une réponse le plus rapidement possible. Cela permet de rassurer les intervenants sur le fait qu'il s'agisse d'un projet « vivant ».

Avec les membres/développeurs

- *il est conseillé de toujours accepter les personnes voulant participer. Cela implique de passer du temps au début pour vérifier et tester leur travail, mais c'est un travail nécessaire pour pouvoir ensuite profiter des bénéfices de développeurs externes;*
- *discuter les choix techniques importants sur la mailing list avant de prendre des décisions (voir le chapitre La prise de décision page 28);*
- *respecter l'éthique Open Source, sinon le projet risque de ne pas attirer de développeurs ni d'utilisateurs;*
- *si possible communiquer via la mailing list ou le forum dédié aux développeurs, de manière à inciter des développeurs potentiels à participer;*

Avec les utilisateurs

- répondre aux questions ou demandes d'aides des utilisateurs, ou veiller à ce que quelqu'un y réponde. Si personne ne répond, l'utilisateur risque de croire que le projet n'est plus actif et décidera probablement de ne pas utiliser le produit. Dans le cas où on ne peut pas satisfaire une requête (de fonctionnalité ou de demande de correction de bug), il vaut mieux en informer l'utilisateur (en expliquant pourquoi ça n'est pas possible, par exemple).
- Il est déconseillé de faire de la publicité pour un produit ou service commercial dans un forum ou une mailing list d'un projet Open Source. Cela risque d'être mal vu car cela ne correspond pas à la « philosophie » Open Source en général.
 - *dans le cas d'un produit ou service externe au projet, c'est fortement déconseillé.*
 - *dans le cas d'un projet Open Source dont les membres du projet (ou la société à la base du projet s'il y en a une) fournissent des services sur le produit ou une documentation payante, c'est un peu mieux accepté. Cela doit néanmoins rester relativement « neutre » (pas trop orienté publicité), car le fait de vendre un produit ou service en profitant d'une mailing list Open Source va à l'encontre de l'éthique Open Source.*

6.2.5 La prise de décision

La prise de décision dans un projet Open Source peut être ou non clairement définie. Elle est souvent bien définie dans les gros projets, et plus floue pour les petits projets.

Les décisions discutées dans la communauté peuvent entre autre concerner:

- les grandes orientations du projet (respect de standard, cible privilégiée du produit, etc);
- les fonctionnalités futures que l'on va ou non implémenter;
- l'architecture du produit;
- les technologies utilisées.

De manière générale, on retrouve toujours certains principes:

- C'est le responsable / chef de projet qui prend les décisions finales, même si généralement il se base sur l'avis des autres développeurs;
- Les décisions sont généralement prises par consensus, c'est à dire que les décisions importantes sont discutées, par exemple sur une mailing list. Si aucun choix ne s'impose, les responsables de projets Open Source peuvent alors proposer un système de vote (souvent effectué simplement par réponse au mail) auquel toutes les personnes du projet peuvent participer. Il se peut que dans certains cas des décisions soient prises de manière autoritaire par un responsable, mais cela n'est généralement pas bien accepté par la communauté et peut être cause de "disputes" au sein de cette communauté pouvant même aller jusqu'à la division de la communauté en 2 projets distincts (cf chapitre intitulé 'Divisions de projets (« Forks ») et disputes').

Exemple: KDE – *The structure, entrance, production, motivation and control in an Open Source project - Dipl. Soz. Andreas Brand (<http://dot.kde.org/1065087742/>) :*

Decisions about the source code or about the future development of the project are made in consensus. Normally project members have a say in the subproject. But there is a tendency that the project coordinator and persons from the inner circle have a bigger influence than persons who only participate.

Les projets regroupant une grande communauté de développeurs définissent parfois une structure ou des règles plus précises permettant de faciliter la prise de décision.

Exemple: Open Office

Les décisions étaient initialement prises par vote sur les mailing lists des modules concernés, ou par décision du "leader" d'un module (<http://evote.cloh.org/hot/open-office.html>), mais la procédure de décision a par la suite été formalisée via un conseil de la communauté (<http://council.openoffice.org/>).

Ce conseil est composé de représentants de la communauté, qui sont élus par des membres de la communauté du projet, et qui doivent également en faire partie. Le conseil a pour but de régler les conflits, de préciser les objectifs du projet en cas de besoin, de prendre les décisions stratégiques nécessaires, etc. Une charte du conseil, qui a été acceptée par vote, décrit les rôles et responsabilités du conseil (<http://council.openoffice.org/CouncilProposal.html>).

A noter que le projet possède aussi un Comité de Pilotage d'Ingénierie (Engineering Steering Committee, <http://council.openoffice.org/esc/index.html>) pour les décisions d'ordre plus techniques.

6.2.6 Divisions de projets (« Forks ») et disputes

Il arrive que des participants à un projet Open Source ne soient pas d'accord sur certains points (décisions techniques, priorités et orientations des développements, changements organisationnels ou de licence, etc). Cela conduit généralement à des « disputes » sur les mailing lists des projets. Dans certains cas, si cette « dispute » n'est pas résolue, cela peut mener à une division du projet: certains membres de l'équipe décident de continuer le projet « ailleurs », sous un autre nom et avec d'autres responsables.

Ce type de division est fréquent dans le monde Open Source, et est intéressant car cela n'est pas possible avec les projets classiques, seules les licences Open Source le permettant.

A noter qu'il existe des divisions ne résultant pas de disputes mais de constatations (que le but du projet n'est pas le même, que les orientations sont différentes, ou encore simplement qu'il est plus simple d'un point de vue organisationnel de séparer les projets).

Même si on considère généralement ces divisions comme contre-productives, l'article « Flame wars, forks and freedom » de Vinayak Hegde (http://www.osnews.com/story.php?news_id=9501) argumente que ces disputes et divisions peuvent être bénéfiques à l'innovation en général, car elles permettent de créer des solutions alternatives, ou bien de remettre en cause des solutions techniques.

Quelques exemples de divisions célèbres:

- *emacs et xemacs;*
- *x.org et Xfree;*
- *JBoss: création de la société « Core Developers Network » et lancement du projet Apache Geronimo (cf http://www.infoworld.com/article/03/08/11/HNjbossapache_1.html).*

6.2.7 Devenir membre/développeur d'un projet Open Source

Les démarches pour devenir membre d'un projet Open Source varient généralement en fonction de la taille d'un projet, de son mode d'organisation, ou encore du besoin qu'ont les administrateurs du projet en « main d'oeuvre externe ».

- Pour les petits projets, il suffit parfois de communiquer son intention de participer à l'équipe ou au chef de projet, qui vous rajoutera alors à la liste des membres (exemple: chiba, <http://chiba.sourceforge.net/>)
- Dans d'autres cas, il faut d'abord montrer sa bonne volonté, généralement en répondant à quelques requêtes d'utilisateurs ou en corrigeant quelques bugs, avant d'être admis comme membre.

Exemple: JBoss (<http://www.jboss.org>), extrait du site (<http://www.jboss.org/community/index>)
If you think the JBoss Community is for you, then the best way to get started is to join our development lists and get involved with the patches and todos. [...]
The project schedule lists all tasks that have to be complete. If you locate a task that is not assigned or is incomplete, feel free to chime in. Send a note to the project lead or the [TODO forum](#) with a reference to the task, stating that you would like to own it. If there is an area where you want to make a contribution, but there are no relevant tasks announced, please post a message on the [TODO forum](#).
Repeat this operation a few times, and with a bit of luck and persistence you could catch a committers' attention and be rewarded with ReadWrite access to our tree.

Dans certains cas, les droits en écriture ne sont pas donnés tout de suite (ils peuvent parfois n'être donnés qu'après avoir résolu des problèmes et écrit des patches).

Exemple: Open Office (<http://www.openoffice.org>), extrait du site (<http://www.openoffice.org/contributing/programming.html>):

How to submit code to OpenOffice.org
We ask that all code submitted to OpenOffice.org be submitted via [IssueZilla](#). In your submission please list "Issue Type" as PATCH. Your code will be sent to the committer for the appropriate project.

Exemple: Apache (<http://www.apache.org/foundation/how-it-works.html#meritocracy>)

When the group felt that the person had "earned" the merit to be part of the development community, they granted direct access to the code repository, thus increasing the group and increasing the ability of the group to develop the program, and to maintain and develop it more effectively.

- Enfin, il existe des projets Open Source pour lesquels les membres externes ne sont pas les bienvenus, et ce pour différentes raisons généralement liées au temps nécessaire à la gestion et à la coordination de ces membres.

7 PROMOTION DU PROJET

Les gestionnaires d'un projet Open Source sont toujours intéressés d'exposer leur projet à leur public cible et aux autres développeurs, soit pour obtenir des retours sur les fonctionnalités du logiciel, soit pour attirer des testeurs ou même des développeurs. Nous suggérons ici quelques méthodes pour augmenter la visibilité d'un projet Open Source sur l'Internet.

1. Enregistrer le site auprès des principaux moteurs de recherche et dans les annuaires dédiés aux projets de ce type. Pour l'enregistrement chez Google, utilisez <http://www.google.com/intl/fr/addurl.html>.
2. Écrire un blog du projet et l'enregistrer chez les principaux moteurs de recherche spécialisés en blogs :
 - Technorati – <http://www.technorati.com>
 - Blogarama - <http://www.blogarama.com/index.php?show=add>
 - Blogwise - <http://www.blogwise.com/submit.php>
 - Blogsearchengine - http://www.blogsearchengine.com/add_link.html
 - Blogdex - <http://blogdex.net/add.asp>
 - Daypop- <http://www.daypop.com/info/submit.htm>

Il faut écrire fréquemment dans le blog du projet; lire des blogs sur des sujets proches et commenter leurs articles; reprendre des sujets débattus sur d'autres blogs dans le blog du projet et utiliser le mécanisme de « trackback » pour apporter du trafic sur le blog.

1. Échanger des bannières sur des sites qui offrent ce service gratuitement:
 - http://www.linuxwaves.com/Linux_Events/BE/
 - <http://www.pythonandzope.com/BannerInfo>
 - http://www.linuxexposed.com/internal.php?op=modload&name=Banner_Exchange&file=index

On peut trouver un **logiciel spécialisé dans l'échange des bannières**:

<http://cms.interaktonline.com/products/Banners/>

Voici la bannière produite pour AnaXagora:



5. Échanger des liens avec d'autres sites du même domaine (éducation, business, recherche) ou avec d'autres sites des projets Open Source (http://www.express-marketing.com/why_use_links.html)
6. Inclure l'adresse du site dans la signature des membres de l'équipe du projet, pour attirer l'attention sur le site.
L'URL doit être accompagnée d'une description du projet en quelques mots.
Voici la signature utilisée pour AnaXagora:

*Member of the AnaXagora Team (<http://www.anaxagora.tudor.lu>)
Open source change management platform integrating e-learning with knowledge and competence management*
7. Des cibles possibles pour présenter le projet pourront être les clients potentiels, les entreprises qui pourront être intéressées par le logiciel que l'on développe – ils pourront suggérer des améliorations ou des nouveaux besoins. Attention à ne pas encombrer ces entreprises avec du « spam », car cela peut avoir un effet contraire à celui espéré. De tels messages doivent être courts et explicites, en indiquant le but du message, les manières dont l'entreprise pourra profiter de la relation, et une offre d'explications plus détaillées si désirée. Pour identifier de telles entreprises, on peut par exemple organiser une veille à laquelle peuvent participer tous les membres de l'équipe.
8. Publier des courtes notes sur le projet dans des sites dédiés au logiciel libre et aussi sur des sites reliés à la thématique du logiciel développé. Voici quelques exemples:

- Journal du Net - <http://www.journaldunet.com/>
- Open the Source – <http://www.openthesource.fr/blog/blog.php>

9. Une technique est d'intervenir sur un forum ou un blog dédié aux développeurs en logiciel libre, en laissant l'adresse du projet et en posant par exemple une question non directement liée au projet. Un tel procédé peut apporter du trafic, mais aussi des ennuis: si le message est perçu comme une publicité, la réaction de la communauté peut compromettre définitivement le projet. Voici un exemple:

Ex: <http://ask.slashdot.org/article.pl?sid=01/02/27/0558250&tid=96&tid=4>
on Slashdot

**Posted by [Cliff](#) on Tue Feb 27, '01 09:58 AM
from the [increasing-site-visibility](#) dept.**

[TimRiker](#) asks: "I run the Open Source project [BZFlag](#) and would like to increase awareness of its existence. I see ads for Open Source projects on Slashdot, SourceForge, Freshmeat, etc. Is there an OpenSource banner exchange around that will get my ad shown on other sites in exchange for displaying their ads on my site? I'd prefer not to show adds on my site for any non-free software." This is an interesting thought and it would go about "getting the word out" in an efficient manner. I haven't heard of such a service, but I think it might be a good idea. What about you?

Seems to me that you've gotten decent exposure by asking a question tangential to your project and getting that link up there in the story for thousands of /.ers to click. You've got an uber-ad!

Dans le domaine du logiciel libre, la sélection des outils Open Source se fait surtout en mode « pull » (le client cherche et trouve lui même l'outil qu'il veut utiliser), la publicité n'est donc pas forcément une bonne pratique, même s'il faut quand même assurer une bonne visibilité au projet de développement.

Dans le cas des projet en Logiciel Libre, les règles de promotion sont plus cadrées, plus strictes que pour les projets commerciaux. L'éthique Open Source doit être respectée. Par exemple, on ne va jamais vanter son produit, mais plutôt présenter des références pour que les gens puissent se convaincre eux mêmes des qualités du produit.

10. Rajouter un utilitaire permettant d'enregistrer le nombre de visiteurs sur le site du projet. En utilisant un outil Open Source spécialisé, on peut aller plus loin et profiter de statistiques complètes sur les visites du site.

Exemples:

- AWStats (<http://awstats.sourceforge.net/>)
- phpMyVisites (<http://www.phpmyvisites.net>)

11. Fournir un produit: il est toujours plus facile d'attirer des utilisateurs sur le site si une première version du produit est disponible, qu'elle soit stable ou non. Il ne faut donc pas hésiter à faire des releases préliminaires du produit, en précisant bien sa stabilité (par exemple par un numéro de version approprié, comme « v0.14 », v0.8-alpha » ou « v1.02-rc2 »).

12. Faire communiquer suffisamment d'utilisateurs via le site web (à travers les outils de communication prévus). Plus il y a d'utilisateurs qui participeront au site, plus le projet sera attractif car considéré comme « actif » par les nouveaux arrivants. Lorsqu'un projet démarre, il est souvent difficile d'avoir les premiers utilisateurs du site; une solution pour accélérer les choses peut être de veiller à ce que les développeurs communiquent via la site (forum, mailing lists, blogs).

8 OUTILS DE GESTION ET DE DÉVELOPPEMENT DU PROJET

Cette partie présente les principaux types d'outils utilisés pour des projets Open Source, qu'ils concernent la gestion du projet ou le développement.

8.1 Outils de communication: différents types d'outils

8.1.1 Mailing list

Une « mailing list » est une liste de diffusion sur laquelle tous les utilisateurs inscrits peuvent envoyer et répondre à des mails. Une fois inscrit, les mails envoyés à la mailing list seront automatiquement reçus dans votre boîte mail.

C'est souvent l'outil de communication principal des projets Open Source: souvent plusieurs mailing lists sont définies, avec des buts différents:

- une mailing list pour les questions des utilisateurs, pour les retours d'expérience, etc
- une mailing list réservée aux discussions des développeurs ou des personnes qui connaissent le fonctionnement interne du produit
- selon la taille du projet, il peut y avoir des mailings lists sur des modules spécifiques du projet, pour réduire le nombre d'inscrit sur chaque liste et donc le nombre de mails reçus.

Le nombre de mailing lists dépend donc de la taille du projet: plus un projet est important, plus les mailing lists auront de trafic; dans ce cas, il est donc intéressant de les diviser pour que chacune des mailing lists reçoive un trafic plus modéré.

Les mailing lists sont souvent accompagnées d'un outil d'archives des mail sur le site du projet, ce qui permet aux utilisateurs (inscrits ou non) de rechercher parmi les anciens messages, un peu à la manière d'un forum.

Outils principaux: mailman (<http://www.list.org>), ezmlm (<http://www.ezmlm.org/>), Majordomo (<http://www.greatcircle.com/majordomo/>), beaucoup de systèmes de sites ou d'outils de travail collaboratifs (outils de groupware) proposent leur propre outil de gestion des mailing lists.

8.1.2 Forum

Un forum est également un outil de communication, au même titre qu'une mailing list, mais il fonctionne d'une manière différente: les messages ne sont pas envoyés dans la boîte mail des utilisateurs, mais restent accessibles via la page web du forum (normalement navigable par ordre chronologique et/ou par sujet).

Cela offre l'avantage que les boîtes mails ne sont pas encombrées par des mails qui ne sont pas toujours intéressants: les utilisateurs ne vont consulter le forum qu'en cas de besoin. Par contre, c'est également un outil moins pratique pour communiquer, puisqu'il faut aller voir le site explicitement pour être informé. On le réserve donc en général aux questions des utilisateurs occasionnels, et on préfère souvent utiliser une mailing list pour la communication entre les membres du projet ou avec des utilisateurs plus impliqués.

De même que pour les mailing lists, certains projets plus importants peuvent définir plusieurs forums avec des thèmes différents, un public cible différents, ou simplement un découpage en modules du projet. C'est le cas par exemple du projet JBoss (<http://www.jboss.org>), qui possède plus de 60 forums répartis en 10 catégories.

8.1.3 Suivi des bugs, évolutions et requêtes

Ce type d'outil permet de réaliser un suivi des bugs ou des demandes d'évolutions. Il est donc très pratique et conseillé dans le cas de projets de développement.

On y retrouve en général un titre, une description, une date à laquelle la demande a été réalisée, une priorité, une assignation dans l'équipe, un auteur.

Souvent l'item est représenté par une page sur laquelle il est possible d'écrire des commentaires ou des réponses, de manière à pouvoir discuter de l'item avec les membres de l'équipe ou les utilisateurs.

Outils principaux: bugzilla (<http://www.bugzilla.org>), « Tracker system » de sourceforge.net, Mantis (<http://www.mantisbt.org/>)

8.2 Systèmes de gestion des versions

Il existe de nombreux systèmes de gestion des versions, dont certains commerciaux. Les systèmes Open Source sont néanmoins préférés pour gérer les sources des projets Open Source ou libres.

8.2.1 CVS (Concurrent Versions System, <https://www.cvshome.org/>)

C'est l'outil le plus répandu pour la gestion des versions. Il est utilisé dans beaucoup de projets, et également par d'autres systèmes de gestion des versions.

Il ne possède pas d'interface graphique propre, mais il existe des interfaces graphiques Open Source pour CVS développées par d'autres projets (par exemple CvsGui, <http://www.wincvs.org/>). Il existe également des plug-ins dans de nombreux outils de développement, qui permettent de l'intégrer directement dans des environnements de développement.

8.2.2 Subversion (<http://subversion.tigris.org>)

C'est le projet présenté comme le successeur de CVS: l'équipe de développement est en effet composée en partie d'anciens développeurs de CVS. Subversion a pour but de répondre aux limites de CVS.

C'est un projet récent, donc qui n'est pas encore beaucoup utilisé, mais qui va probablement commencer à l'être suite à la sortie de la release 1.0 en 2004.

Autres outils: RCS (<http://www.gnu.org/software/rcs/rcs.html>), utilisé comme base de la plupart des autres outils de gestion des versions.

8.3 Outils de compilation et de déploiement

Étant donné que les développeurs de projets Open Source travaillent souvent sur des environnements entièrement différents, il est très utile d'utiliser un outil de compilation/déploiement multi-plateforme et commun à tous les développeurs.

8.3.1 Make (<http://www.gnu.org/software/make/make.html>)

Make est un système de compilation fonctionnant sous les systèmes Unix et Linux pour les programmes en langage C.

C'est un outil traditionnellement très utilisé dans les projets de développement unix/linux, mais assez ancien, et plus souvent utilisé pour le langage C: il est maintenant souvent remplacé par un outil multi-plateforme comme « ant ».

8.3.2 Ant (<http://ant.apache.org>)

« ant », du projet Apache, est un outil de compilation et de déploiement multi-plateforme, basé sur Java. C'est un outil qui se répand de plus en plus dans les projets Open Source, et notamment dans tous les projets multi-plateformes et Java. Il est donc conseillé de l'utiliser si votre projet est multi-plateforme.

Il fonctionne par un fichier de configuration au format XML, « build.xml », qui contient les actions à réaliser pour différentes « tâches » pouvant être exécutées.

Les actions peuvent être soit des actions standards fournies par l'outil, soit des actions fournies par des extensions de l'outil. Parmi les actions standards, on retrouve:

- des commandes du système de fichier: renommage, déplacement de fichier, changement des droits, copie de fichiers, création de répertoires, etc.
- des commandes pour la compilation et l'exécution dans différents langages de programmation
- des actions réalisées par des outils supplémentaires comme les outils de compression zip/gzip/tar, CVS, ssh, etc

On peut définir autant de tâches que l'on veut, et les réutiliser dans d'autres tâches. Les tâches que l'on retrouve fréquemment dans les projets Open Source sont:

- les tâches de compilation des sources
- les tâches de packaging du code compilé
- les tâches d'exécutions
- éventuellement des tâches de tests
- des tâches de déploiement

8.4 Systèmes de tests unitaires

Les tests unitaires étant très importants pour les projets Open Source, certains projets demandent d'utiliser un outil pour les réaliser, de manière à les automatiser.

Ces outils permettent de définir les exécutions à réaliser, et de comparer les valeurs attendues avec les valeurs retournées par l'exécution: on appelle ce genre de comparaisons des assertions.

Les tests unitaires sont la méthode de tests proposée dans la méthodologie de développement d'« eXtreme Programming » (XP).

Famille d'outils dite « xUnit »

Ce sont des outils de tests unitaires assez simples et très répandus, qui existent pour la plupart des langages de programmation (et dont le nom commence par une ou plusieurs lettres identifiant le langage de programmation). Ils permettent d'effectuer des tests par comparaison des résultats obtenus avec les résultats attendus. Une fois le jeu de test défini, les tests peuvent être exécutés de manière automatique.

L'idée initiale des tests unitaires provient de Kent Beck ([1]) qui proposait un tel framework pour le langage SmallTalk.

JUnit (<http://www.junit.org>) est un tel outil de tests unitaires pour les programmes Java, qui est beaucoup utilisé dans les projets Open Source Java. Il existe également de nombreux outils ou frameworks qui se basent sur JUnit pour fonctionner (exemple: Cactus d'Apache, <http://jakarta.apache.org/cactus/index.html>)

D'autres outils de cette famille sont CppUnit (<http://cppunit.sourceforge.net>) pour le C++, PHPUnit (<http://phpunit.sourceforge.net/>) pour les programmes PHP.

8.5 Environnements de développement intégrés

Les environnements de développement intégrés (en anglais IDE, « Integrated Development Environment ») sont des outils aidant à la programmation, généralement via des éditeurs adaptés à des langages de programmation, des remplacements automatiques, des facilités d'exécution et de tests des programmes, etc. Une IDE peut aller d'un simple éditeur amélioré jusqu'à une application complète de développement comprenant tous les outils nécessaires au développement d'une application.

Il existe de nombreux IDE Open Source, soit généraux, soit spécialisés pour un langage de programmation. Cependant, chaque développeur est libre d'utiliser l'outil qu'il souhaite car cela n'influence normalement pas le programme produit: il est donc conseillé de ne pas forcer l'utilisation d'un IDE plutôt qu'un autre, et de ne pas non plus stocker de fichiers liés à un IDE particulier dans le système de gestion des versions.

Les développeurs peuvent également utiliser des IDE propriétaires, mais ce n'est généralement pas conseillé car certains IDE propriétaires proposent par défaut d'utiliser des bibliothèques internes (donc non Open Source) pour faciliter l'intégration dans l'IDE.

Exemples Open Source:

- *eclipse* (<http://www.eclipse.org/>) est un IDE programmé en Java (donc indépendant de la plateforme) mais supportant divers langages de programmation
- *netbeans* (<http://www.netbeans.org/>) est un IDE programmé en Java pour le langage Java et soutenu par Sun
- *emacs* (<http://www.gnu.org/software/emacs/emacs.html>) est très répandu, surtout sur les systèmes Unix et Linux, mais plutôt difficile à utiliser. Il existe plusieurs versions distinctes, notamment *Xemacs* (<http://www.xemacs.org/>)

8.6 Sites de gestion de projets Open Source

Certains sites rassemblent des produits Open Source tout en offrant des outils permettant de gérer les projets Open Source associés. Ces sites ont plusieurs buts:

- héberger les sites des projets Open Source
- permettre la gestion des projets Open Source grâce à des outils de communication et de travail collaboratif mis à la disposition des projets
- offrir un catalogue de produits Open Source aux visiteurs

Ils sont généralement gratuits, mais peuvent posséder différentes contraintes limitant leur utilisation à certains projets. Ils sont au cœur de la communauté Open Source, puisque ce sont eux qui la matérialisent via des outils de communication. Le plus connu est le site sourceforge.net, mais il existe également d'autres sites, concurrents ou plus spécialisés.

8.6.1 Sourceforge.net

Sourceforge.net est le site d'hébergement gratuit de projets Open Source le plus connu et le plus utilisé dans le monde. En mars 2003 il hébergeait environ 98 000 projets et comptait 1 million d'utilisateurs inscrits. Il y a peu de contraintes pour pouvoir y faire héberger son projet: le projet doit être Open Source (diffusion par une licence Open Source approuvée par l'« Open Source Initiative », <http://www.opensource.org/licenses/>, ou par une autre licence Open Source répondant aux mêmes spécifications), il doit être un projet de développement logiciel ou de documentation, ou être associé à une communauté Open Source. Beaucoup de projets Open Source utilisent ce site, qui offre de nombreux avantages.

Sourceforge propose à chaque projet Open Source des outils pour gérer le projet: des forums et mailing lists, un outil de gestion de bugs, de patchs, un outil de demande d'évolution, une gestion de la documentation, un système de gestion des tâches, un outil de gestion des sources (CVS), la possibilité d'associer et d'héberger un site web propre au projet, un outil de releases et de téléchargement des releases, etc ¹

En plus de cela, sourceforge.net offre une grande visibilité du projet: c'est en effet le site principal de projets Open Source avec un nombre très élevé de projets et d'utilisateurs; utiliser ce site représente donc une réelle publicité pour le projet, à la fois pour les développeurs qui peuvent potentiellement participer au projet, et pour les utilisateurs cherchant des outils Open Source. Il est donc à la fois pratique et avantageux d'utiliser ce site pour un projet de développement Open Source.

Outils principaux

Chaque projet doit décider de quels outils proposés seront utilisés parmi tous les outils proposés aux projets hébergés. Il est conseillé de ne pas forcément utiliser « un peu » tous les outils proposés, mais plutôt de ne sélectionner que ceux qui sont vraiment utiles.

Description du projet

C'est la première partie affichée sur la page du projet. Elle contient différentes informations qu'il est important de bien remplir car c'est la page d'entrée du projet pour les recherches des utilisateurs.

- *Description*: quelques lignes qui résument le but du projet
- *Development Status*: indicateur de l'état d'avancement du projet.

Status	Description
1 - Planning	pas encore de livrable intéressant
2 – Pre-Alpha	début du développement
3 - Alpha	première phase de développement: livrables accessibles mais instables.
4 - Beta	phase plus avancée de développement: livrables accessibles mais potentiellement instables.
5 – Production/Stable	existence d'une release stable
6 - Mature	projet possédant une release stable depuis longtemps
7 - Inactive	projet plus maintenu (sur le site)

- *Environment, Operating System, Programming Language*: informations sur l'environnement de fonctionnement du logiciel
- *License*: licence Open Source du projet
- *Natural Language*: langue de communication du projet
- *Topic*: catégorisation par thème des projets

Mailing list

Sourceforge permet de créer plusieurs mailing lists, basées sur l'outil « mailman », avec des noms différents. L'inscription peut se faire en ligne (indépendamment de l'inscription au site sourceforge.net). Sourceforge permet aussi de naviguer et de faire des recherches dans les archives de ces mailing lists.

Dans la plupart des projets actifs du site, on retrouve au moins une mailing list « utilisateurs » et une mailing list « développeurs ».

Forums

Un outil de forum est également mis à la disposition des projets: Il peut être utile pour les discussions du projet, mais attention à ce qu'il ne fasse pas double emploi avec les mailing lists.

Tracker

Sourceforge.net met à la disposition des projets plusieurs systèmes permettant de gérer les bugs, les demandes d'évolutions, les patchs, etc. Il est conseillé de n'activer que le sous ensemble de ces outils que nous comptez réellement utiliser.

CVS

Le système de gestion des versions fourni par sourceforge est CVS. Des tests sont en cours pour permettre l'utilisation de l'outil Subversion, mais aucune date n'est encore annoncée.

1 La liste complète des services offerts est disponible dans la documentation du site sourceforge.net, http://sourceforge.net/docman/display_doc.php?docid=753&group_id=1

8.6.2 Autres sites de projets Open Source

Sourceforge.net n'est pas le seul site d'hébergement de projets Open Source: il existe d'autres sites, qui sont généralement moins connus, mais qui peuvent parfois être spécifique à une communauté particulière (par exemple dans un pays ou pour une langue particulière), ou à un thème de projets. Souvent, ces sites ressemblent à sourceforge de part leurs fonctionnalités, et parfois même de part leur aspect et leur utilisation, car certains de ces sites utilisent le même outil que sourceforge pour fonctionner.

Quelques exemples de sites:

- *Tigris.org (<http://www.tigris.org/>) est un site regroupant des projets de développement logiciel collaboratif. Il contient un nombre limité de projets, mais qui répondent chacun à un besoin particulier pour le développement logiciel. Il contient notamment les projets ArgoUML (Outil UML) et Subversion (système de gestion des versions)*
- *Savannah (<http://savannah.gnu.org/>) est un site qui reprend le principe de sourceforge, mais qui est destiné aux logiciels GNU. Un autre site lié à celui ci est <http://savannah.nongnu.org> pour les projets « non GNU » mais liés*
- *Apache (<http://www.apache.org/>) est une fondation qui propose de nombreux projets Open Source dans différents domaines (serveur web, outils de développement Java, XML, web services, etc). L'organisation est différente de celles d'autres sites comme sourceforge.net dans la mesure où seuls les projets faisant partie de la fondation apparaissent sur le site.*
- *BerliOS (<http://developer.berlios.de/>): un site allemand dans la même idée que sourceforge.net*

9 LE DÉVELOPPEMENT

Une des spécificités les plus importantes des projets Open Source et que les développements sont ouverts à tous: ils sont donc réalisés par plusieurs personnes, qui travailleront soit sur des parties différentes soit sur des mêmes parties du code à des moments différents.

Il est toujours plus compliqué de développer un produit de manière collaborative que tout seul: il faut pouvoir coordonner son travail avec celui d'autres développeurs, et respecter certaines règles sans lesquelles le code source des projets Open Source deviendraient vite chaotiques et illisibles. Cette partie contient donc des conseils pour réaliser des développements d'une manière collaborative pour un projet Open Source.

9.1 Communication sur les développements

Il est conseillé de discuter avec les autres développeurs et le responsable (généralement via une mailing list ou un forum de développements) avant d'entamer des développements, pour différentes raisons:

- *Si les développements sont réalisés directement dans le projet, la personne responsable doit donner son accord pour un développement avant qu'il ne commence.*
- *Cela peut permettre d'obtenir des idées des autres développeurs sur les moyens de réaliser ce développement ou les fonctionnalités à développer*
- *Cela évite que plusieurs personnes travaillent sur le même développement en même temps sans le savoir*

Le fait de ne pas communiquer ce que l'on a l'intention de réaliser peut avoir comme conséquences:

- *Que le travail soit rejeté par la personne responsable du module (si la fonctionnalité n'est pas désirée, ou si l'implémentation ne répond pas à certains critères)*
- *Que d'autres développeurs proposent des manières alternatives de réaliser la fonctionnalité*
- *Des conflits au niveau du système de gestion des sources si une autre personne a modifié les mêmes fichiers en même temps*

En conclusion, lorsqu'on décide de développer une fonctionnalité sans en avertir les membres du projet, il y a de fortes chances que le développement ne serve à rien ou soit à refaire.

9.2 Règles de programmation / Conventions de code

De manière à faciliter la relecture de code écrit par différents développeurs, il est souvent demandé de respecter des règles et conventions de programmations.

Dans la plupart des projets, ces règles sont documentées et vérifiées par le ou les responsables sur le code provenant d'autres développeurs du projet. Il existe également des projets utilisant un outil permettant de vérifier que ces règles sont bien respectées.

PMD (<http://pmd.sourceforge.net/>) est un tel outil pour les programmes Java, assez répandu. En plus d'être utilisable pour son propre projet avec ses propres règles, il fonctionne également chaque jour avec des règles prédéfinies sur un ensemble de projets du site sourceforge.net, et affiche les résultats compilés sur une page web (<http://pmd.sourceforge.net/scoreboard.html>).

Exemple de règles: règles de programmation pour le projet ArgoUML :
<http://argouml.tigris.org/documentation/default.html/cookbook/ch09.html> .

Ces règles peuvent couvrir différents aspects de la programmation, dont typiquement:

9.2.1 Les noms des identifiants du langage

Les noms des identifiants du langage (noms des attributs, constantes, variables, classes, méthodes ou fonctions, packages, fichiers, etc) peuvent être soumis à respecter des conventions particulières.

C'est très souvent la convention dite « CamelCase » (<http://en.wikipedia.org/wiki/CamelCase>) (ou parfois appelée « StudlyCase ») qui est utilisée: cette convention consiste à coller plusieurs mots sans mettre d'espaces entre eux mais en écrivant en majuscule la première lettre de chaque mot, de manière à les séparer visuellement. On distingue le « UpperCamelCase » qui impose que la première lettre soit en majuscule du « lowerCamelCase » qui impose le contraire.

Exemple: « ThisIsMyClass » (UpperCamelCase), « thisIsMyVariable » (lowerCamelCase).

9.2.2 La forme

Cela peut être les retours à la ligne ou non avant les accolades, les sauts de lignes, les espaces, tabulations et indentations, l'alignement des point-virgules, etc.

Certaines conventions précisent par exemple que pour indenter le code, le programmeur ne doit pas utiliser de tabulations mais des espaces (généralement 4 espaces = 1 tabulation, à configurer dans son éditeur), de manière à éviter que les tabulations n'apparaissent différemment selon les postes ou systèmes d'exploitation.

9.2.3 Le contenu, la forme et la fréquence des commentaires

Il peut être imposé d'avoir au moins un commentaire par opération ou méthode, par exemple. Ou encore d'y faire figurer le nom de l'auteur d'une classe ainsi que sa date de création dans l'en-tête de cette classe.

La forme des commentaires peut être imposée pour les langages de programmation dans lesquelles ils peuvent être écrits selon plusieurs syntaxes.

Exemple: en Java et en PHP, on peut utiliser « `//...` » pour un commentaire sur une ligne et « `/* ... */` » pour les commentaires sur plusieurs lignes.

Des méta-données peuvent être demandées dans les commentaires, selon une syntaxe particulière si ces méta-données sont destinées à être traitées automatiquement. Elles peuvent contenir des informations générales comme la date ou l'auteur, ou des informations supplémentaires sur le code, comme le rôle des paramètres d'une méthode ou des informations sur le déploiement d'une classe.

En Java, cette pratique est de plus en plus répandue:

- les commentaires sous forme « javadoc » (<http://java.sun.com/j2se/javadoc/>) sont très fréquemment demandés: il s'agit d'informations de documentation mises dans des « tags » particuliers dans les commentaires (préfixés par '@'), et qui sont ensuite « compilées » dans une documentation HTML pour l'ensemble des classes du programme. C'est le mode de documentation le plus utilisé en Java.
- Certains outils, dont le plus connu est xdoclet (<http://xdoclet.sourceforge.net/xdoclet/index.html>), permettent de générer du code à partir de ces attributs ou méta-données, avec un syntaxe proche de javadoc. Ce code peut consister en d'autres classes, ou des fichiers concernant le déploiement de l'application. Xdoclet en particulier permet de générer du code pour beaucoup d'outils/frameworks, comme par exemple pour générer des EJB, des fichiers de configuration pour JBoss, etc.

9.2.4 D'autres règles concernant l'architecture ou la qualité du code

Des éléments liés à l'architecture du projet ou à la qualité du code peuvent être demandés, comme par exemple le découpage du code, le fait de plutôt écrire de nombreuses méthodes très simples ou de préférer des méthodes plus longues, etc. Ces règles sortent du contexte des conventions de code proprement dites; De plus ces règles dépendent énormément du projet, il n'existe pas vraiment de règles générales.

Il existe des outils permettant non pas de vérifier la syntaxe employée dans le code, mais vraiment de mesurer la qualité du code grâce à des métriques particulières (voir le chapitre Phase 2 : Sélection et Modélisation des composants page 46).

9.2.5 Réutilisation des conventions de code

Écrire un document complet de conventions de code peut être long, et il en est de même pour le lire. Donc, étant donné qu'un principe cher aux développeurs Open Source est la réutilisation, il est conseillé d'utiliser des conventions existantes et connues plutôt que de définir ses propres conventions, quitte à rajouter quelques adaptations en cas de besoin. Cela évitera aux développeurs d'avoir à relire un document important de conventions pour chaque nouveau projet si les conventions utilisées sont les mêmes.

En Java, on retrouve très souvent les conventions proposées par Sun (<http://java.sun.com/docs/codeconv/html/CodeConventions.doc8.html>).

D'autre part, un certain nombre de principes concernant la programmation sont sous entendus et communs à la plupart des projets Open Source. Ces principes sont des bonnes pratiques générales, mais leur respect est d'autant plus important si le code est Open Source: de manière générale, il est demandé de programmer d'une manière indépendante d'une machine ou d'un serveur (de ne pas mettre de chemins « en dur » dans le code, par exemple), de mettre suffisamment de commentaires pour que le code soit compréhensible par quelqu'un d'autre, d'utiliser des noms de variables clairs, etc. Étant donné que chacun peut voir le code produit dans un projet Open Source, et que ce code sera probablement maintenu ou modifié par des personnes autres que l'auteur, ces règles sont normalement plus contraignantes que pour les projets de développements classiques.

9.3 Gestion des versions

Le code des projets Open Source est presque toujours géré dans un système de gestion des versions comme CVS. Il existe certaines règles de bases à respecter lors de l'utilisation d'un tel système:

- Du fait que la plupart des systèmes de gestion des versions, notamment CVS, ne possède qu'une gestion simple des utilisateurs, chaque développeur du projet peut modifier le travail des autres sans que cela ne nécessite de validation supplémentaire. Il est donc très important de se mettre d'accord avec le reste de l'équipe sur ce qu'on compte développer avant de le réaliser, ou en dernier délai avant de le valider dans le système de gestion des versions.
- Ces systèmes permettent généralement d'entrer un commentaire accompagnant une modification lorsqu'elle est entrée dans le système: il est largement conseillé de bien utiliser ce commentaire pour décrire les modifications effectuées, de manière à ce que les autres développeurs puissent comprendre ces modifications.
- Habituellement, il est demandé que les modifications validées dans le système soient déjà fonctionnelles, de manière à ce que les personnes mettant à jour leur projet puissent obtenir un code qui puisse être directement compilé et fonctionner, même s'il n'est pas officiellement "stable". Il est souvent demandé d'effectuer des tests unitaires sur ces modifications avant de les valider (voir partie Phase de tests et de packaging page 58), mais même si ça n'est pas explicite, le code validé dans le système de gestion des versions doit être suffisamment stable pour ne pas bloquer d'autres développeurs dans leurs travaux.
- La plupart des systèmes de gestion des versions ne bloquent pas les fichiers sur lesquels des développeurs sont en train de travailler. Il est dans ce cas indispensable de communiquer ce que vous voulez faire avant de commencer, de manière à ce qu'un autre développeur ne modifie pas les mêmes fichiers en même temps.

Ces systèmes de gestion des versions peuvent être utilisés soit uniquement pour stocker le code source, soit de manière plus globale pour stocker toute une application, c'est à dire y compris les bibliothèques nécessaires et la documentation.

Cette dernière option a l'avantage de permettre l'utilisation des mêmes bibliothèques chez tous les développeurs, ce qui peut éviter de nombreux problèmes de compatibilités. Ceci est particulièrement vrai pour les langages de programmation indépendants de la plate-forme, comme Java, car dans ce cas le seul facteur externe à contrôler par les développeurs est la version de l'environnement du langage de programmation, puisque toutes les autres bibliothèques peuvent être fournies via le système de gestion des versions et que l'exécution du programme ne dépend pas d'autres facteurs.

De plus, ce type d'utilisation d'un système de gestion des versions permet de gérer les versions des bibliothèques utilisées, ce qui est très important pour les projets Open Source car ils utilisent généralement beaucoup de bibliothèques différentes, et qui, étant également Open Sources, sont souvent soumises à des évolutions régulières.

Cela a par contre comme conséquences que les données sur CVS sont plus grandes.

Exemple de stockage dans un système de gestion des versions de toute une structure pour un projet Java:

- un répertoire « src » contenant le code
- un répertoire de compilation (appelé souvent « build », « exec », « classes », etc), vide (Attention à ne pas stocker les fichiers de ce répertoire dans le système de gestion des versions, car ces fichiers sont destinés à être remplacés à chaque compilation)
- un ou plusieurs répertoires contenant les bibliothèques (« lib » par exemple)
- un fichier de configuration de la compilation, par exemple un fichier « build.xml » pour le système de compilation « ant », ou bien un fichier « makefile » pour « make ». Il est également très

important de partager ce fichier et de veiller à ce que tous les développeurs l'utilisent, de manière à ce que la compilation soit toujours effectuée avec les mêmes options

- *un répertoire contenant la documentation du projet et/ou une copie du site web*
- *tous les autres fichiers qu'il est nécessaire de partager pour le projet (fichiers readme.txt, fichiers de licences, etc)*
- *on ne retrouve généralement pas de fichiers qui sont propres à l'environnement de travail d'une personne, comme les fichiers souvent générés par les IDE pour stocker leur configuration. Attention à ne pas les mettre par accident dans le système de gestion des versions, notamment lors de la création du projet.*

10 PROCESSUS DE RÉUTILISATION DE COMPOSANTS OPEN SOURCE

L'objectif majeur de la réutilisation logicielle de composants consiste à réduire les coûts et les délais de conception, d'implémentation et de maintenance des logiciels. De part leur nature, les logiciels Open Source se prêtent bien à la réutilisation, puisque leur code est d'une part disponible et d'autre part modifiable.

La notion de composants et de réutilisation de composants dans le domaine de l'ingénierie logicielle est transversale aux différentes étapes du cycle de vie du processus de développement à savoir Expression des besoins, Analyse, Conception, Implantation, Tests.

Ces composants, suivant l'étape du cycle de vie en question, vont être de différentes natures :

- Entre les phases d'expression des besoins et d'analyse, les composants sont des Patrons d'analyse, des Modèles de domaine (Business Model Domain), des composants métiers conceptuels;
- Entre les phases d'Analyse (MDA/PIM) et de Conception (Design – MDA/PSM), les composants sont des Patrons d'architectures, des Patrons de conceptions, des Frameworks (Struts par exemple);
- Entre les phases de Conception et d'implantation (code), les composants sont des composants métiers Logiciels, des APIs, des Patrons d'implantation.

Dans les phases de conception et d'avant conception, le composant va prendre la forme d'un Patron ou « Pattern ».

En phase d'implémentation, le composant est bien souvent un composant métier, une bibliothèque de classes ou un « Pattern Technologique » (exemple : Design Patterns J2EE).

Cette problématique se rapproche de la notion de « Pattern Driven Architecture » introduite dans la démarche MDA (Model Driven Architecture) de l'OMG.

10.1 Typologie et caractérisation des composants

10.1.1 Différents types de composants

Les projets Open Source peuvent intégrer différents types de composants Open Source. Ces composants peuvent être séparés en 3 types:

- des « programmes » réalisant déjà une partie importante des fonctionnalités désirées, que l'on va modifier pour rajouter les fonctionnalités manquantes.
- des bibliothèques, réalisant des fonctionnalités souvent plus limitées, que l'on va utiliser directement:
 - bibliothèques pour manipuler un certain type de format de fichier, du XML, faire des requêtes Xpath, etc
 - bibliothèques pour écrire des logs (log4j)
 - ...
- des outils de type serveur ou langages permettant à l'application de fonctionner:
 - système d'exploitation (Linux, ...)
 - serveur web (apache, ...)
 - serveur de servlets / serveur applicatif (tomcat, JBoss, ...)
 - langage / environnement de programmation (PHP, Java, ...)
 - ...

A noter que les outils du dernier type ne doivent pas forcément être Open Source pour pouvoir être utilisés dans des projets Open Source, car ils ne sont pas forcément intégrés dans l'application produite: cela dépend des termes exacts de la licence.

Ils peuvent également souvent être intégrés dans l'application produite avec une licence particulière non Open Source (c'est par exemple le cas de Java).

Cela peut également être le cas des bibliothèques, en fonction de la licence cible du projet, et des licences des bibliothèques (voir la partie Choix de la licence page 14).

10.1.2 Caractérisation des composants

On voit donc que les composants réutilisables sont d'une très large variété et il est nécessaire, afin de les sélectionner et de les réutiliser au mieux, de les caractériser de façon claire.

Des critères de caractérisation possible au niveau des composants pourraient être les suivants :

- **couverture du composant** : générale (horizontale), métier (verticale), entreprise
- **portée** : étapes d'ingénierie,
- **nature de la solution** : logicielle ou conceptuelle,
- **technique de réutilisation** : spécialisation, composition, instanciation, duplication...
- **ouverture** : boîte noire, blanche ou en verre,
- **granularité** : taille du composant en nombre de classes, de modules,...

Il existe différents types de composants suivant l'étape du cycle de vie impactée :

composants-patterns

Christopher Alexander, précurseur du concept de patrons et auteur du livre « A Pattern Language » (A Pattern Language" by Christopher Alexander, Sara Ishikawa, Murray Silverstein, with Max Jacobson, Ingrid Fiksdahl-King and Shlomo Angel. Published by Oxford University Press, 1977) définit le patron de la façon suivante :

« un patron décrit à la fois un problème qui se produit très fréquemment dans votre environnement et l'architecture de la solution à ce problème de telle façon que vous puissiez utiliser cette solution des milliers de fois sans jamais l'adapter deux fois de la même manière»

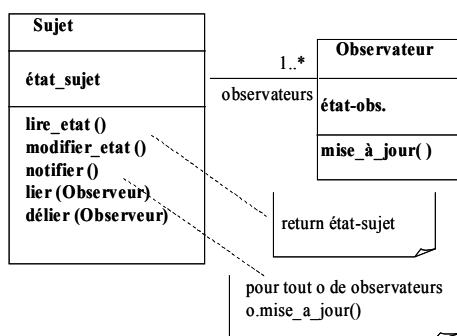
En reprenant nos critères voici comme on pourrait caractériser un composant-patron :

	Patron
Couverture	
Portée	
Nature de la solution	Conceptuelle
Technique de réutilisation	Imitation puis Intégration
Ouverture	Boîte blanche
Granularité	Faible

Prenons l'exemple du fameux Patron « Observateur » [Gamma 95] dont voici un bref rappel :

Contexte d'application : définir une ou plusieurs dépendances entre un sujet et ses observateurs de sorte que si le sujet change d'état, tous ses observateurs en soit informés et mis à jour.

Solution :



Voici une description possible de ce pattern avec notre caractérisation :

	Patron de conception
Couverture	Horizontale (générale)
Portée	Conception détaillée
Nature de la solution	Conceptuelle

	Patron de conception
Technique de réutilisation	<i>Imitation puis intégration</i>
Ouverture	<i>Boîte blanche</i>
Granularité	<i>Faible (2 à 4 classes)</i>

Composants-APIs

Dans de nombreuses réutilisations consistent en la réutilisation de bibliothèques logicielles ou APIs.

	API – Classe de Bibliothèque
Couverture	<i>Horizontale</i>
Portée	<i>Implantation</i>
Nature de la solution	<i>Logicielle</i>
Technique de réutilisation	<i>Spécialisation et Instanciation</i>
Ouverture	<i>Boîte en verre</i>
Granularité	<i>Très Faible</i>

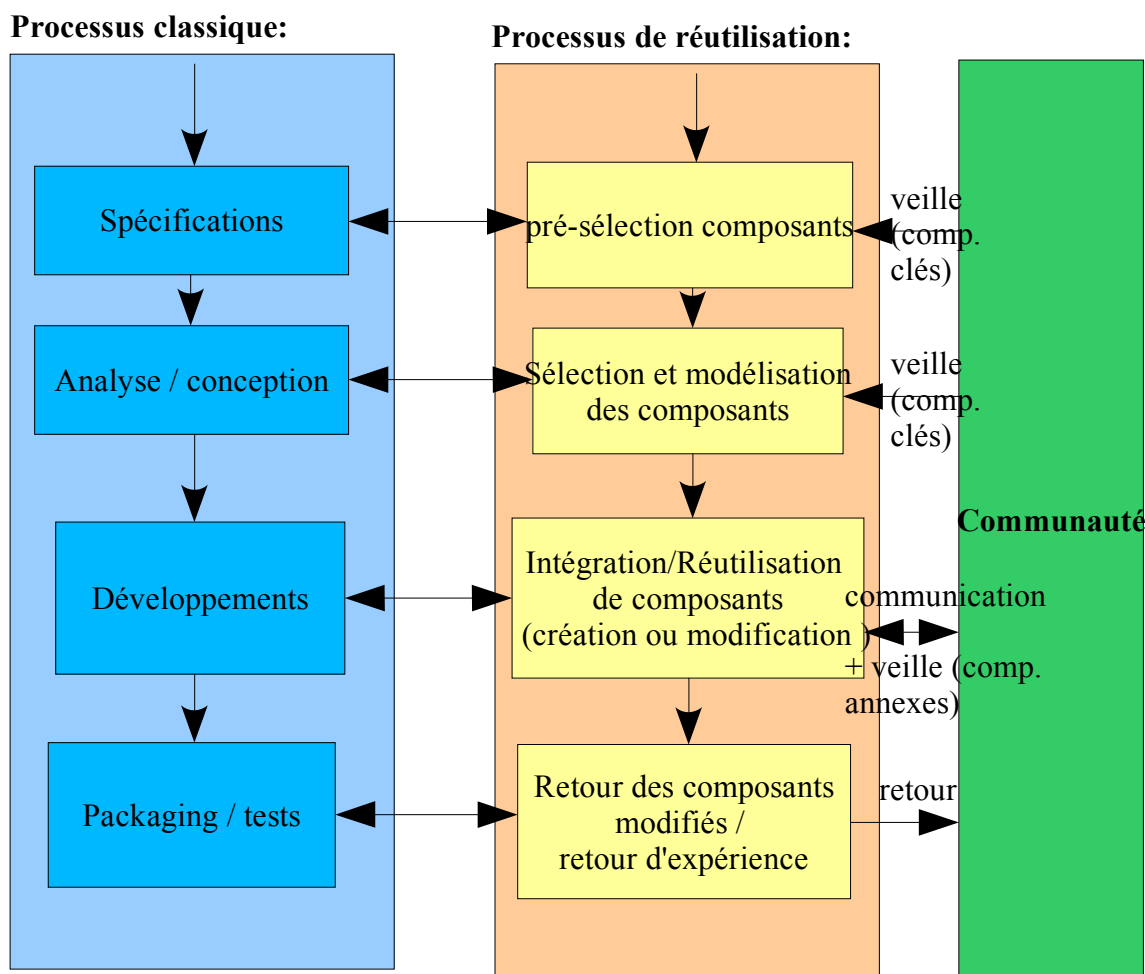
Composants-Composants Métiers

Un composant métier est une entité encapsulée et autonome de déploiement s'adressant à un domaine d'application bien spécifique ce qui le différencie des bibliothèques APIs.

	Composant Métier
Couverture	<i>Verticale</i>
Portée	<i>Analyse</i>
Nature de la solution	<i>Conceptuelle</i>
Technique de réutilisation	<i>Spécialisation et instanciation</i>
Ouverture	<i>Boîte blanche</i>
Granularité	<i>variable</i>

10.2 Vue globale du processus de Réutilisation de composants

Ce schéma met en correspondance le processus classique de développement logiciel avec un « processus de réutilisation » qui s'effectue en parallèle.



L'étape de spécification du projet s'accompagne généralement d'une première veille sur les composants (souvent de plus haut niveau) pouvant servir au projet: l'existence ou non de certains composants peut en effet influencer les objectifs définis lors des spécifications (voir partie Spécifications et Évolutions des besoins page 17).

Lors de l'analyse et de la conception, les composants importants du projet doivent être définitivement sélectionnés: leur utilisation influence en effet l'architecture globale du projet, ainsi que les technologies employées.

C'est lors de la phase de développement que l'on intègre réellement les composants. Il est également conseillé de communiquer avec la communauté lors de cette phase pour préparer la phase de retour du composant.

La dernière phase, durant laquelle le projet est packagé et testé, est le moment idéal pour effectuer un retour à la communauté sur le composant utilisé.

10.3 Phase 1 : Pré-sélection de composants

La phase de pré-sélection de composants est la phase en amont du processus de réutilisation et elle démarre bien souvent en parallèle de la définition des spécifications de l'application à développer. Cette phase peut permettre :de faire une veille sur les standards existants, les outils réutilisables existants et les meilleures pratiques dans le domaine et ainsi:

- de permettre une sorte de rétro-ingénierie fonctionnelle du composant permettant de mieux appréhender les besoins du domaine de son application et ainsi d'être plus exhaustif, complet et moins ambiguë au niveau de la description des exigences fonctionnelles de son application. Cette technique peut permettre, en utilisant le composant comme une maquette, d'accompagner

la phase d'élicitation des besoins avec les utilisateurs et de valider leur pertinence et leur faisabilité;

- de mieux appréhender les exigences non-fonctionnelles (en terme par exemple de protocoles ou formats d'échange, d'interopérabilité etc).

Par exemple, dans le cadre du développement d'un outil de BPM (Business Process Modeling), nous avons été amenés à étudier l'ensemble des composants Open Source existants dans ce domaine. Nous avons pu identifier plusieurs types de composants : composants permettant la modélisation de processus en vue d'une exécution par un moteur de workflow, composants permettant la modélisation de processus en vue d'une diffusion-publication sur un portail. Nous avons ainsi surtout pu compléter, par les fonctionnalités offertes par l'outil les principales exigences fonctionnelles du domaine et aussi appréhender les standards existants dans ce domaine (BPMN, SPEM, BPEL, BPML, ...).

10.4 Phase 2 : Sélection et Modélisation des composants

10.4.1 Sélection de composants

Les Principes à respecter dans la sélection de ses composants

L'ingénieur d'application doit ne sélectionner et ne réutiliser que des composants qui vérifie la célèbre règle suivante : Reliable, Reusable, Replaceable, Resilient), c'est à dire que le composant doit être fiable, réutilisable, remplaçable, élastique(souple, adaptable, modifiable).

Critères de qualité et de complexité d'un composant existant

Afin de préjuger de la maintenabilité et du degré de réutilisabilité et de complexité du composant « candidat » à la sélection, il est utile d'obtenir des indicateurs du code source de ce composant via un certain nombre de métriques

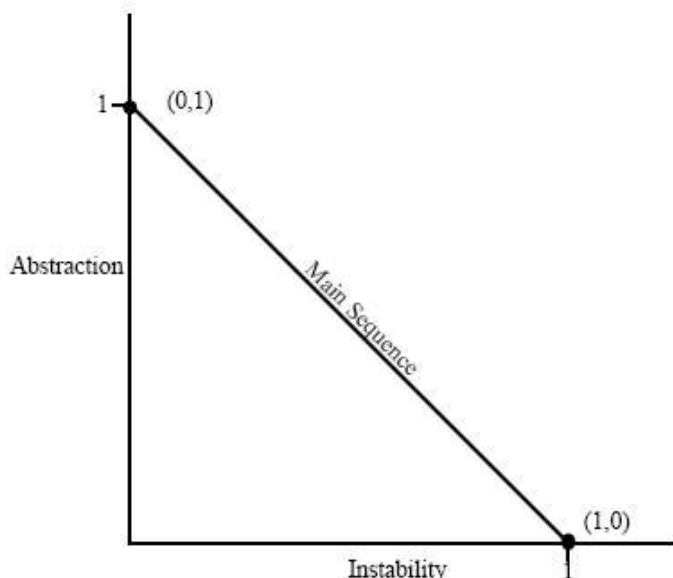
Dans son célèbre article « OO Design Quality Metrics » (1994), Robert Martin définit un certain nombre de métriques et d'indicateurs de qualité d'un design OO (orienté objet) parmi lesquels :

- *Indicateur CC* : Le nombre de classes concrètes d'un package (toutes les classes non abstraites et sans les interfaces);
- *Indicateur CA* : Le nombre de classes abstraites et d'interfaces d'un package;
- *Indicateur Ca* : Le couplage afférent, nombre de classes en dehors du package dépendantes de classes (/interfaces) définies dans ce package;
- *Indicateur Ce* : Le couplage sortant (efferent), nombre de classes du package dépendantes de classes (/interfaces) définies dans un package extérieur;
- *Indicateur A* : Le degré d'abstraction du package (nombre flottant variant de 0 à 1);
- *Indicateur I* : Le degré d'instabilité du package (nombre flottant variant de 0 à 1);
- *Indicateur D* : La distance à la séquence principale
- *Indicateur C* : L'existence d'une dépendance cyclique au niveau du package.

D'après Robert Martin, les packages stables (I=0) devraient être constitués de classes abstraites (en Java, par extension d'interfaces), tandis que les packages instables (I=1) devraient contenir des classes concrètes.

La ligne « idéale » ou « main sequence », d'après Robert Martin, permet de situer les packages relativement à un segment idéal et, par ce biais, de considérer si le degré d'abstraction d'un package (indicateur A) est cohérent avec son degré de stabilité (I) :

- (I=0,A=1);
- (I=1,A=0).



Via l'indicateur D (« distance from main sequence »), Robert Martin propose un moyen de focaliser l'attention sur les packages dont la distance (D) est loin de 0 (donc proche de 1). A partir d'un chiffre de 0.2 ou 0.3, il faut commencer à réexaminer de près un package.

Ces indicateurs peuvent être calculés automatiquement à l'aide d'outils spécifiques parmi lesquels « JDepend », outil développé par Jim Clarke.

Afin d'évaluer la complexité d'un composant, un certain nombre d'autres métriques peuvent être utilisées :

- Le nombre de paramètres;
- Le niveau de profondeur des imbrications d'instructions;
- La profondeur des héritages;
- Le niveau de documentation des classes et des méthodes;
- ... ;

Pour chacun de ces critères des « bornes de normalité » peuvent-être définies.

Pour évaluer la complexité d'un code, il est possible d'utiliser des logiciels dédiés. Un des logiciels les plus usités (IQ2) permet d'évaluer le degré de complexité de son code par le biais de la méthode Mc Cabe, méthode mondialement reconnue. Toutefois cette méthode, axée sur la notion de graphe, ne semble pas assez précise et suffisante notamment dans le cadre d'un développement avec les technologies objets.

Nous nous sommes donc tourné vers l'évaluation d'un framework appelé D-Side Dashboard, en cours de développement par les FUNDP de Namur (<http://d-side.cetic.be>).

Il permet d'évaluer la complexité d'un code suivant les métriques de base suivantes :

- Le nombre de fichiers;
- La complexité cyclomatique (Mac Cabe);
- Le fan-out externe (référence à des classes externes);
- Le fan-out interne (référence à des inner classes);
- Le nombre d'attributs en public;
- Le nombre de méthodes dans la classe;
- Le nombre de lignes de commentaires;
- Le nombre d'attributs en static ;
- Le nombre d'attributs;
- Le nombre de classes;
- Le nombre de lignes de code.

Des bornes inférieures et supérieures de « normalité » ont été fixées pour chaque critère et l'outil fournit un rapport final permettant d'analyser et de localiser les points de complexité de son code.

Ainsi, nous avons pu expérimenter l'outil D-Side Dashboard dans le cadre de l'évaluation du produit utilisé comme base pour la création du module AnaXagora BPM : APES. Les résultats fournis par D-Side Dashboard sont présentés dans les photos d'écrans suivantes.

- All
- [-] APES2
- [-] APES2.apes
- [-] APES2.apes.adapters
 - [-] APES2.apes.model
- [-] APES2.apes.model.spem
- [-] APES2.apes.model.spem.basic
- [-] APES2.apes.model.spem.core
- [-] APES2.apes.model.spem.extension
- [-] APES2.apes.model.spem.frontend
- [-] APES2.apes.model.spem.modelmanagement
- [-] APES2.apes.model.spem.process
- [-] APES2.apes.model.spem.process.components
- [-] APES2.apes.model.spem.process.structure
- [-] APES2.apes.model.spem.statemachine
- [-] APES2.org
 - [-] APES2.org.ipsquad
 - [-] APES2.org.ipsquad.apes
 - [-] APES2.org.ipsquad.apes.adapters
 - [-] APES2.org.ipsquad.apes.model
 - [-] APES2.org.ipsquad.apes.model.extension
 - [-] APES2.org.ipsquad.apes.model.frontend
 - [-] APES2.org.ipsquad.apes.model.frontend.event
 - [-] APES2.org.ipsquad.apes.model.spem
 - [-] APES2.org.ipsquad.apes.model.spem.basic
 - [-] APES2.org.ipsquad.apes.model.spem.core
 - [-] APES2.org.ipsquad.apes.model.spem.modelmanagement
 - [-] APES2.org.ipsquad.apes.model.spem.process
 - [-] APES2.org.ipsquad.apes.model.spem.process.components
 - [-] APES2.org.ipsquad.apes.model.spem.process.structure
 - [-] APES2.org.ipsquad.apes.model.spem.statemachine
 - [-] APES2.org.ipsquad.apes.processing
 - [-] APES2.org.ipsquad.apes.ui
 - [-] APES2.org.ipsquad.apes.ui.actions
 - [-] APES2.org.ipsquad.apes.ui.tools
 - [-] APES2.org.ipsquad.utils

Change tree :

ByPackage ▾

Ok

Name	Type	Language	Metrics (see legend)
[-] adapters	JavaPackage	Java	jNbOfFiles: 5 jCommentsDensity: 0.17 jHotMethodsDensity: 0.03 jNbOfLinesOfComments: 338 jNbOfClasses: 8 jNbOfLinesOfCode: 2007
[-] model	JavaPackage	Java	jNbOfFiles: 0 jCommentsDensity: N/A jHotMethodsDensity: N/A jNbOfLinesOfComments: 0 jNbOfClasses: 0 jNbOfLinesOfCode: 0
[-] TestContext	JavaClass	Java	jNbOfAttributes: 0 jNbOfMethodsInClass: 4 jSimpleExternalFanOut: 1 jSimpleInternalFanOut: 0
[-] FakeAction	JavaClass	Java	jNbOfAttributes: 0 jNbOfMethodsInClass: 1 jSimpleExternalFanOut: 2 jSimpleInternalFanOut: 0
[-] testGetInstance	JavaMethod	Java	jCyclomaticComplexity: 1
[-] setUp	JavaMethod	Java	jCyclomaticComplexity: 1
[-] testGetSetTopLevelFrame	JavaMethod	Java	jCyclomaticComplexity: 1
[-] testRegisterGetAction	JavaMethod	Java	jCyclomaticComplexity: 1
[-] actionPerformed	JavaMethod	Java	jCyclomaticComplexity: 1

Object details

Please select an object in the list above.

- All
- APES2
- APES2.apes
- APES2.apes.adapters
 - APES2.apes.model
- APES2.apes.model.spem
- APES2.apes.model.spem.basic
- APES2.apes.model.spem.core
- APES2.apes.model.spem.extension
- APES2.apes.model.spem.frontend
- APES2.apes.model.spem.modelmanagement
- APES2.apes.model.spem.process
- APES2.apes.model.spem.process.components
- APES2.apes.model.spem.process.structure
- APES2.apes.model.spem.statemachine
 - APES2.org
 - APES2.org.ipsquad
- APES2.org.ipsquad.apes
- APES2.org.ipsquad.apes.adapters
 - APES2.org.ipsquad.apes.model
- APES2.org.ipsquad.apes.model.extension
- APES2.org.ipsquad.apes.model.frontend
- APES2.org.ipsquad.apes.model.frontend.event
- APES2.org.ipsquad.apes.model.spem
- APES2.org.ipsquad.apes.model.spem.basic
- APES2.org.ipsquad.apes.model.spem.core
- APES2.org.ipsquad.apes.model.spem.modelmanagement
- APES2.org.ipsquad.apes.model.spem.process
- APES2.org.ipsquad.apes.model.spem.process.components
- APES2.org.ipsquad.apes.model.spem.process.structure
- APES2.org.ipsquad.apes.model.spem.statemachine
- APES2.org.ipsquad.apes.processing
- APES2.org.ipsquad.apes.ui
- APES2.org.ipsquad.apes.ui.actions
- APES2.org.ipsquad.apes.ui.tools
- APES2.org.ipsquad.utils
- APES2.utils

Change tree :

ByPackage

Name	Type	Language	Metrics (see legend)	
<input type="checkbox"/> SpemVisitorStub	JavaClass	Java	jNbOfAttributes jNbOfMethodsInClass jSimpleExternalFanOut jSimpleInternalFanOut	1 23 24 19
<input type="checkbox"/> visitPackage	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitGuidanceKind	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitGuidance	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitFlowDiagram	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitExternalDescription	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitContextDiagram	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitProcessComponent	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitProcess	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitComponentInterface	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitApesProcess	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitActivity	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitActivityDiagram	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> isValid	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitStateMachine	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitWorkDefinition	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitResponsabilityDiagram	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitRole	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitProvidedInterface	JavaMethod	Java	jCyclomaticComplexity	1

Name	Type	Language	Metrics (see legend)	
<input type="checkbox"/> ValidateVisitor	JavaClass	Java	jNbOfAttributes jNbOfMethodsInClass jSimpleExternalFanOut jSimpleInternalFanOut	2 27 ‡ 24 19
<input type="checkbox"/> routingEnd	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> routingBegin	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> getHasErrors	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> checkRequiredProduct	JavaMethod	Java	jCyclomaticComplexity	8 ‡
<input type="checkbox"/> checkProvidedProduct	JavaMethod	Java	jCyclomaticComplexity	6
<input type="checkbox"/> visitProvidedInterface	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitRole	JavaMethod	Java	jCyclomaticComplexity	3
<input type="checkbox"/> visitResponsabilityDiagram	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitProcessComponent	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitProcess	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitProduct	JavaMethod	Java	jCyclomaticComplexity	18 ‡
<input type="checkbox"/> visitProcessPerformer	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitStateMachine	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitSpemDiagram	JavaMethod	Java	jCyclomaticComplexity	2
<input type="checkbox"/> visitRequiredInterface	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitApesProcess	JavaMethod	Java	jCyclomaticComplexity	1
<input type="checkbox"/> visitComponentInterface	JavaMethod	Java	jCyclomaticComplexity	1

Grille de Sélection de composants

La sélection d'un composant Open Source dans un but de réutilisation comporte des différences par rapport à une sélection classique de logiciels. Le composant sélectionné devra en effet être intégré au reste de notre application, il doit donc être facilement intégrable. D'autre part, la qualité du code joue un plus grand rôle, ainsi que les contraintes sur son environnement de fonctionnement, qui sont souvent des contraintes plus fortes que pour la sélection de logiciels.

Voici une liste des principales différences avec une sélection classique de logiciel:

- Environnement d'exécution, langage de programmation, etc: ces contraintes sont souvent plus fortes que pour la sélection de logiciels, car elles doivent absolument être respectées pour que le composant puisse être intégré avec le reste du projet.
- Dépendances importantes: la liste des dépendances (bibliothèques ou autre) du composant peut permettre de déceler des incompatibilités avec d'autres bibliothèques utilisées, ou bien des problèmes de licences pour la redistribution
- Présence d'une communauté de développeurs: la présence d'une communauté peut permettre d'aider l'intégration du composant, d'orienter les futurs développements du composant, etc
- Stabilité de la version: le fait qu'un composant n'existe pas encore en version stable peut être moins important dans le cas d'un composant Open Source, car au besoin l'équipe du projet peut s'occuper elle même d'effectuer des tests et corrections, ou même une personne ou société externe au projet. C'est donc un critère moins bloquant que dans le cas classique, que l'on peut donc prendre en compte pour comparer des logiciels.
- Documentation pour développeurs: pour réutiliser un composant, il peut être très utile d'accéder à une documentation orientée développeurs. Ce genre de documentation n'existe en général pas pour les logiciels non Open Sources.
- Qualité et documentation de l'architecture: de la même manière que pour le point précédent, la qualité et la documentation de l'architecture jouent un rôle très important. En particulier, si on ne veut réutiliser qu'une partie du composant, l'architecture du composant doit être modulaire
- Qualité du code: le travail d'intégration du composant est rendu plus facile si le code du composant est de bonne qualité, claire, documenté, etc. Ce sont des critères qui en général influencent moins la sélection de logiciels qui ne sont pas destinés à être modifiés.
- Intégrabilité, respect des standards: d'autres critères de qualité peuvent être pris en compte pour la réutilisation de composants, et notamment si le composant respecte des standards, ce qui le rendra plus facilement intégrable.

Pour réaliser la comparaison, on peut utiliser différents outils, comme par exemple le logiciel Opal qui permet de fixer des critères, de noter en fonction de ces critères et de générer un rapport permettant d'effectuer le choix définitif. Un autre moyen est d'utiliser un logiciel de tableur comme Open Office Calc (ou Microsoft Excel en logiciel propriétaire).

Voilà un exemple de fiche réalisée avec Open Office Calc (remplie avec des données d'exemples – les lignes suivies de « ... » sont à répéter/affiner en fonction du test):

Outils:	E-GroupWare	Coefficients (/100)	
	<i>réponses textuelles</i>	<i>notes /10</i>	
Site web	http://egroupware.org		
testé? (sinon: revue)	Oui		
Date	18/10/04		
Personne(s)	MLW		
Critères fonctionnels			
Fonctionnalité X	OK	8	4
Sous-fonctionnalité Y	ok mais bug lors de tests	9	8
...			
Interface graphique			

présence d'une UI pour ... ?	X	5	7
langue(s) de l'UI	bcp, y compris anglais, français et allemand	10	18
appréciation qualité de l'UI	ok, UI très bien finie	7	7

Critères non fonctionnels

Environnement (système, langages, etc + numéro de version)

Environnement (système)	indépendant du système	10	10
Environnement (langage de programmation)	PHP v1.4	8	5
Environnement (autre: BD, etc)	My SQL vx		

.. dépendances importantes (bibliothèques particulières, etc)

...

Stabilité, communauté

licence	GPL v2		
Numéro de version	1.0.00.005		
Date de la version	09/2004		
catégorie de version	stable	8	10
Encore maintenu?	X		
Communauté active? «Activity Percentile» sourceforge (si présent)	OK, site sourceforge 100.00%	10	1

Organisation de la communauté développeurs indépendants proposant du «sponsoring» + des services (formations...)

Autres infos / communauté « Project of the month » sourceforge en mai 2004

Stabilité testée pas testé

Documentation

documentation pour installation?	X	7	5
documentation pour utilisateurs?	X. Pas toutes les fonctionnalités	5	5
documentation pour développeurs?	X	8	

Architecture

Architecture décrite (site web ou doc) ?	non?	0	7
Architecture modulaire?	ok, outils de groupware plus ou moins indépendants	8	8

Qualité du code

Techniques spéciales/Framework de coding ?	Templates + Standards de codage PHP documenté: formattage, header standard, commentaires, position des accolades, etc	6	5
Le code respecte t-il de cette technique?	à vérifier		
modularité du code: OO?	Oui		
découpage en fichiers? ...	Oui		
appréciation de la qualité globale du code	ok		
qualité uniforme sur tout l'outil ?	ok		

Intégrabilité/Réutilisation

interface standard X	respect de X mais pas à 100%
Compatibilité avec standard X-vy.	import seulement

...

Conclusion

points forts	très belle UI, beaucoup de fonctionnalités, outil très connu
points faibles	

Avis personnel	bon outil. Presque trop complet/complexé pour nos besoins ?
----------------	---

Résultat e- Somme des groupware coefficients (=100):
 : 7.52 100

La case de somme des coefficients doit toujours être à 100: elle peut être calculée avec une fonction de somme, selon l'outil.

Dans Open Office Calc ou Microsoft Excel: =SUM(E3:E60)

La case de résultat peut être calculée avec des fonctions de sommes et de multiplications de colonnes, selon les outils.

Dans Open Office Calc ou Microsoft Excel: =SUM(SUMPRODUCT(C3:C60;E3:E60))/100

10.4.2 Modélisation de composants

Pour garantir une meilleur intégration des composants Open Source utilisés dans le projet, il est prudent d'inclure la modélisation de ces composants lors de la modélisation de l'application. Cela peut être fait grâce à l'utilisation de différents langages de descriptions de composants, synthétisés dans le document « Synthèse sur les langages de description de composants » par Damien Nicolas.

Parmi les langages de description de composants (CDL), les diagrammes de composants d'UML 2.0 peuvent être utilisés de manière à intégrer cette modélisation avec le reste de la modélisation de l'application, si elle est faite en UML. Ce langage est moins formel que beaucoup de CDL, et permet une modélisation plus visuelle des composants, et, s'il n'est pas suffisant, la modélisation peut être enrichie par des concepts propres rassemblés dans des « profiles UML ».

Outils: outil de modélisation pour le CDL choisi.

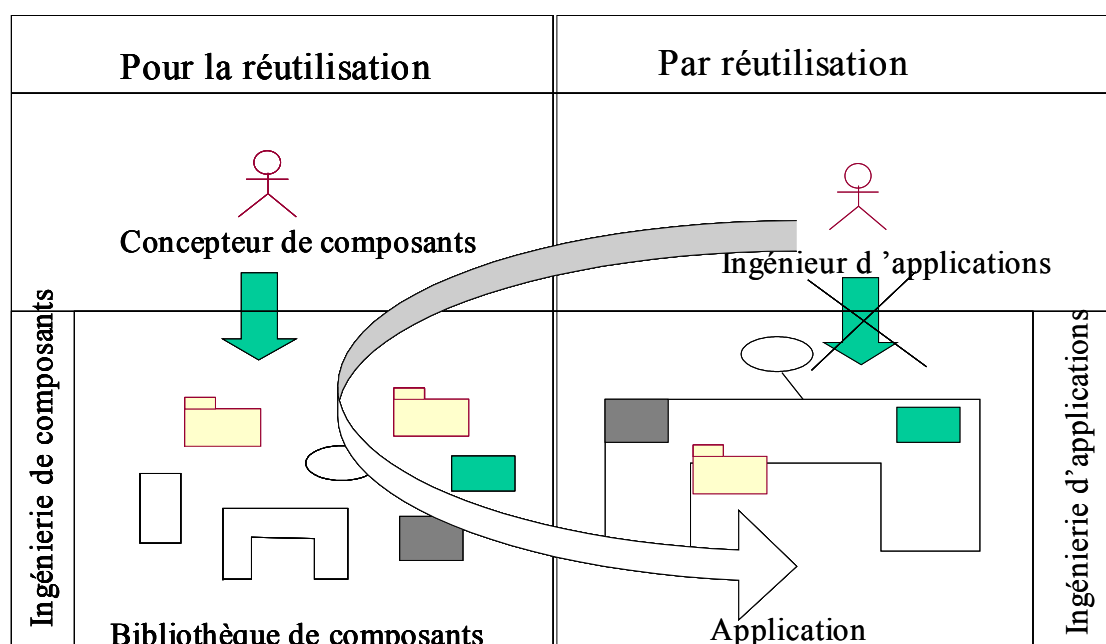
10.5 Phase 3 : Réutilisation de composants

Les composants peuvent être créés suivant plusieurs contextes différents.

- Les composants peuvent être issus d'applications existantes ou de parties d'applications existantes et identifiés par un processus de ré-ingénierie
- Ils peuvent avoir été créés spécifiquement soit dans le cadre du cycle de vie de développement d'une application soit de manière isolée comme des composants isolés à intégrer et à assembler de manière à créer des applications répondants à de nouvelles exigences.

Généralement on différencie les deux processus de développement :

- le processus de développement pour la réutilisation : Le concepteur d'application va concevoir des composants qui vont venir alimenter une bibliothèque suivant une catégorisation et taxinomie bien précise;
- Le Processus de développement par réutilisation : L'ingénieur d'application va réaliser son application sur base d'une sélection de composants existants dans la bibliothèque de composants.



On peut noter que cette différenciation est moins évidente pour les composants Open Sources que pour les composants propriétaires: la bibliothèque de composants est alors souvent remplacée par l'ensemble des composants disponibles sur Internet; dans ce contexte, tous les produits Open Source peuvent être considérés comme des composants « pour la réutilisation », qu'ils aient été réellement écrits pour la réutilisation ou bien par réutilisation mais dans un autre but. L'ensemble des critères ci dessous sont donc importants pour les composants Open Sources.

10.5.1 Processus de développement « Pour la réutilisation »

Les principes à respecter pour la création de composants

De grandes règles générales peuvent être émises pour l'écriture de composants réutilisables :

- Généraliser autant que possible : en langage objet, il s'agit d'utiliser l'héritage et le mécanisme de polymorphisme, ce qui permettra de reporter le choix de certaines des caractéristiques du composant au moment de son intégration et permettra d'obtenir un composant général paramétrable suivant le contexte;
- Créer une bonne structure : faire en sorte que le composant puisse être réutilisé le plus façon « boîte noire » possible en restreignant l'utilisation des variables globales, en créant des méthodes cohérentes et à granularité fini (« méthodes petites »);

- Créer modulaire : permettre la modification ultérieure de modules sans troubler le reste de l'application en prenant soin une bonne encapsulation de données, une bonne abstraction des structures de données ainsi qu'une interface claire;
- Gérer l'autonomie des composants : encapsuler au plus les sous-programmes externes au composant et bien documenter les dépendances externes d'un composant;
- Gérer la robustesse : prendre en charge correctement les exceptions en réservant des traitements par défaut aux cas d'erreurs;
- Éviter la complexité : cette partie est abordée dans une section suivante.

10.5.2 Processus de développement « Par réutilisation »

Les Différents types de réutilisation de composants

L'ingénieur peut réutiliser le composant qu'il a sélectionné dans le cadre du développement de son application suivant quatre modes bien distincts :

- le mode « Boîte noire » : il s'agit de réutiliser le composant sans connaître son implémentation, son code source.
- Le mode « Boîte blanche » : il s'agit de réutiliser le composant en l'adaptant à son nouveau contexte par le biais d'une modification de son code source.
- Le mode « Boîte grise » : il s'agit de réutiliser le composant en l'adaptant à son propre contexte par le biais d'une paramétrisation
- Le mode « Boîte en verre » : il s'agit de réutiliser un composant en consultant son code source mais sans le modifier. Ce mode est relativement intéressant puisqu'il permet de mieux appréhender dans quelle mesure le composant répond à la règle des cinq « R » explicitées ci-dessus.

Ainsi, on peut intégrer un composant Open Source à une application de différentes manières: si l'on essaye de minimiser les dépendances entre les composants, on parle d'intégration « faiblement couplée », par opposition à une intégration « fortement couplée ».

Pour une intégration faiblement couplée, le meilleur moyen est d'utiliser uniquement la description de l'interface du composants, sans consulter son contenu ou la manière dont le composant fonctionne (mode « boîte noire »). De cette manière, les applications utilisant cette interface n'auront pas besoin d'être mises à jour à chaque fois que les composants évolueront.

Au contraire, si le lien entre l'application et le composant intégré est « fortement couplé », il est plus difficile de faire évoluer le composant, car cela peut influencer le code de l'application. D'autre part, on peut vouloir remplacer le composant par un autre (si un composant plus intéressant est développé, si on découvre un problème non facilement résoluble dans le composant, etc), ce qui sera plus difficile si l'application et le composant sont fortement couplés.

Selon le type de langages, il existe des possibilités pour séparer la description « interne » du code d'une application de la description « externe », publiée, qui est destinée à être utilisée. Cela peut être fait de différentes manières:

- en **Java**, simplement par l'utilisation des modificateurs « public », « private » ou « protected » qui permettent de définir si les méthodes ou les champs peuvent être utilisés dans d'autres classes ou packages. Cette solution fonctionne mais est souvent insuffisante de part le faible nombre d'options offertes par ces trois modificateurs.
- les **web services** permettent de faire naturellement cette distinction, puisque les « services » publiés sont les interfaces utilisables directement, et le code de l'application n'est pas accessible directement. En plus de permettre l'indépendance du langage utilisé pour développer le composant, les web services sont donc un bon choix pour permettre une intégration peu couplée de composants.
- pour certains composants, il peut exister un **standard** respecté par le composant qui définit son interface. C'est le cas par exemple des parsers XML de type « DOM » (Document Object Model). Si un standard existe, il suffit d'utiliser les interfaces « standards » plutôt que les interfaces internes du composants pour garantir l'indépendance. Ce standard peut être:
 - une interface de programmation (API) accessible depuis le langage cible
 - un format d'échange, souvent basé sur XML
- indépendamment du langage de programmation, la distinction est souvent faite par la **documentation**, simplement en déclarant un ensemble d'interfaces comme « publiques » et

d'autres comme « internes » à l'application. De cette manière les développeurs s'engagent à modifier le moins possible les interfaces « publique »: il est donc dans ce cas très important de n'utiliser que ces interfaces dans l'application.

Quelles que soient les technologies utilisées dans le composant et dans l'application, il existe donc des moyens permettant de limiter leur couplage pour leur garantir une plus grande indépendance.

Outils:

- documentation, « javadoc » pour les composants Java
- description d'un standard
- fichier WSDL pour la description des services web
- fichier XMLSchema, DTD, ou un autre langage de schéma pour XML pour les descriptions de formats basés sur XML

10.5.3 Les Différentes techniques de réutilisation de composants

- *Imitation puis Intégration*

Cette technique est la plus parlante lorsque le composant réutilisé est un « patron » de conception. L'imitation consiste en la duplication et en l'adaptation (renommer, redéfinir, ajouter, supprimer des propriétés) du patron comme solution modèle. Il s'agit ensuite d'intégrer les imitations c'est à dire d'unir le rôle des classes obtenus (remplacer deux classes par une seule regroupant les propriétés des classes initiales par exemple). Le système est donc vu comme l'intégration d'imitations de patrons ou de façon plus large de composants.

- *Spécialisation et Instanciation*
- *Duplication.*

Modification des composants

Dans certains cas, il est nécessaire de modifier les composants que l'on utilise. Cela peut être dû:

- à un bug dans le composant
- à une fonctionnalité nécessaire à notre projet mais non implémentée dans le composant
- au fait que l'on veuille se servir du composant comme base pour implémenter des nouvelles fonctionnalités non prévues par ce composant.

Dans ce dernier cas, on « détourne » en quelque sorte le composant de son but initial, de manière à pouvoir l'utiliser pour répondre à des besoins différents. Ce dernier cas nécessite généralement des modifications importantes du composant. Par contre, dans les 2 premiers cas, on peut en général **redistribuer les modifications** (voir partie suivante) apportées aux composants à leurs développeurs: en effet, si les modifications ne sont pas spécifiques à l'application développée, elles pourront être intégrées dans les versions suivantes du composants.

Selon le type d'organisation du projet de développement du composant, il peut parfois être utile de modifier le composant de manière synchrone avec cette équipe (voir partie suivante). Dans ce cas, il faut modifier le composant selon les règles organisationnelles et techniques de ce projet.

Si ce n'est pas le cas, où si le composant n'est pas maintenu par une équipe de développement Open Source, le développement peut être réalisé selon les règles de votre projet.

Outils:

- outils de développements utilisés dans le projet, ou dans le projet de développement du composant

10.6 Phase 4 : Retours des composants modifiés / retour d'expérience

Les principes de l'Open Source reposent sur la collaboration de chacun pour faire avancer un projet. Lorsque l'on réutilise un composant Open Source, il est donc important de veiller à apporter quelque chose « en échange » de l'utilisation du composant.

Le fait de redistribuer les modifications peut également avoir une influence positive sur l'intégration du composant: si les modifications réalisées sont intégrées dans les versions suivantes du composants, il sera alors plus facile d'intégrer le composant avec l'application, et notamment de le mettre à jour, car ces modifications n'auront pas à être ré-appliquées si elles sont déjà incluses dans le composant.

Les modifications apportées aux composants doivent donc être dès que possible redistribuées aux développeurs, voire même idéalement développées en collaboration avec eux selon le mode d'organisation du projet gérant le composant.

En effet, la communication avec l'équipe de développement du composant au moment de la modification de celui-ci est une relation « gagnant-gagnant »:

- il peut être utile d'exposer son idée de l'utilisation ou de la modification du composant à ses développeurs pour bénéficier de leur avis, être informé d'éventuels développements similaires en cours ou prévus, bénéficier de leur aide sur la manière de modifier le composant, etc
- il est également utile aux développeurs du composant de savoir comment le composant sera modifié, car s'ils peuvent donner des conseils sur la manière dont le composant sera modifié, ils pourront plus facilement intégrer le résultat dans leur projet

Un retour sur un composant modifié peut être:

1. *Dans le meilleur des cas le composant modifié s'il a été modifié d'une manière suffisamment générique ou utile au projet de développement du composant*
2. *Sinon, un retour d'expérience sur l'utilisation du logiciel, des rapports de bugs, est déjà très utile. Le simple fait de communiquer avec l'équipe du projet de développement du composant pour les informer de l'utilisation que l'on en fait peut souvent leur être utile.*

10.6.1 Retour d'un composant modifié

Le meilleur moyen d'être certain que des modifications apportées à un composant seront prises en compte est de communiquer dès le début avec l'équipe du projet de développement du composant. De cette manière on peut déterminer si la modification à apporter les intéresse, et éventuellement l'adapter à leurs besoins.

Selon l'organisation de l'équipe de développement du composant, un accès direct au système de gestion des sources de cette équipe peut être proposé: dans ce cas, les développements peuvent directement être réalisés sur ce système, et de cette manière, si l'équipe de développement est d'accord et en est informée, les développements seront directement intégrés au projet composant. Dans le cas où un tel accès direct n'est pas mis à disposition, il faut généralement envoyer les modifications par mail aux personnes concernées, qui vont alors les évaluer avant de les intégrer au projet du composant.

Outils:

- *CVS, Subversion, ou un autre système de gestion des versions*
- *Outils de communication: mails, mailing list, forums, ...*

10.6.2 Retour d'expérience

La plupart des projets Open Source proposent des outils de suivi des bugs et des demandes d'amélioration, ou au moins une mailing list. Le retour que l'on peut faire via ces outils sur le composant utilisé peut être très utile à l'équipe de développement du composant, pour leur permettre par exemple de régler des problèmes que l'on aurait rencontrés, ou d'orienter leurs futures modifications.

Le simple fait de mentionner que l'on utilise l'outil peut également être intéressant pour le projet, car le succès des projets Open Source est souvent mesuré par le nombre d'utilisateurs du composant.

Outils:

- *Outils de communication: mails, mailing list, forums, ...*
- *Système de suivi des bugs ou demandes d'améliorations du projet (par exemple bugzilla).*

11 PHASE DE TESTS ET DE PACKAGING

Cette partie décrit les étapes précédant la diffusion d'une release du produit Open Source, c'est à dire premièrement les tests (qui peuvent aussi être réalisés tout le long des développements) et ensuite le packaging, c'est à dire le fait de préparer effectivement le programme à être diffusé.

11.1 Tests

11.1.1 Tests unitaires

Étant donné le mode d'organisation des projets Open Source, il est généralement demandé que des tests unitaires soient effectués par les développeurs pour chacune des parties de codes développées: les tests unitaires ont l'avantage de n'impliquer qu'un morceau de code bien défini, et ils sont donc particulièrement adaptés aux projets Open Source regroupant généralement des développements de différentes personnes. C'est donc la responsabilité de chaque développeur de vérifier que ce qu'il a développé ou modifié fonctionne, en exécutant ces tests unitaires.

Selon le projet, l'exécution de ces tests peut être soit laissée au choix du développeur, soit spécifiée par un outil et/ou des règles strictes. Il peut par exemple y avoir un outil défini et des jeux de tests partagés, qui devront être lancés systématiquement après une modification du code.

Généralement, les responsables du projets relancent également une série complète de tests avant d'effectuer une release, mais il est normalement attendu que le code fourni par un développeur fonctionne avant la préparation de release.

Tests automatiques

Les tests unitaires sont souvent automatiques: ils sont programmés, peuvent être exécutés automatiquement et renvoient un résultat immédiat qui spécifie si le test a réussi ou a échoué. Ceci offre plusieurs avantages:

- ils sont rapides à appliquer: on peut appliquer beaucoup de tests à la fois
- l'interprétation du résultat est immédiate et non ambiguë
- ils peuvent être réutilisés par la suite pour vérifier la non régression

Ils sont cependant plus longs à développer.

11.1.2 Tests d'intégration et systèmes

Ce type de test est proche des tests unitaires dans la mesure où ils sont effectués par les développeurs. Selon les cas ils peuvent aussi être automatisés ou non.

11.1.3 Tests fonctionnels et d'acceptation

Ce sont les tests réalisés par les utilisateurs, pour vérifier que l'application leur convient et fonctionne comme ils veulent.

Ces tests sont généralement peu ou pas formalisés pour les projets Open Source car souvent l'application n'est pas destinée à des utilisateurs bien définis. Ils sont néanmoins généralement quand même réalisés, puisqu'une des forces des développements Open Source est la quantité d'utilisateurs impliqués dans le projet, qui peuvent tester le produit et faire part de leurs remarques à tout moment.

11.2 Packaging et release

Étant donné que les projets Open Source ont tendance à évoluer plus rapidement que les projets commerciaux, et également du fait que les projets Open Source peuvent être utilisés avant d'avoir une release « stable », la gestion des releases et des numéros de versions est d'autant plus importante.

11.2.1 Fréquence des releases

Généralement, le système de gestion des versions du projet est accessible en lecture pour tout le monde, donc le projet n'a pas besoin de fournir des fichiers à télécharger à côté. Cependant, il est plus facile de fournir des releases plutôt que de demander à des utilisateurs pas toujours expérimentés de récupérer le code sur le système de gestion des versions et de le compiler eux même: il est donc conseillé de créer des releases régulièrement.

Il est également intéressant de faire des releases avant que le produit ne soit considéré comme stable, car cela permet d'attirer des utilisateurs (ce qui peut être intéressant si on veut bénéficier de retours d'expériences) et aussi d'éventuels développeurs externes au projet; La fréquence des

releases doit donc être plus élevée que pour les projets classiques, et il ne faut pas hésiter à publier des versions instables, dans la mesure où l'état du produit est précisé clairement.

11.2.2 Numéro de versions

Le numéro de version de la release indique l'état d'avancement de cette release:

- *un numéro suivi de « alpha » ou « beta » indique que la release n'est pas stable mais que c'est une version préliminaire de l'outil. Généralement « alpha » signifie que le produit n'est pas dans une phase très avancée, et « beta » un peu plus. On peut trouver un numéro de version suivi de « alpha » ou « beta » suivi d'un chiffre n: cela signifie que c'est la nième version alpha/beta de la release (exemple: Mozilla 1.8 alpha 6)*
- *un numéro de version suivi de « RC » (généralement RC1 ou RC2) signifie que c'est une « Release Candidate », c'est à dire une version qui est probablement la dernière version à sortir avant la release stable, et qui ne sera plus modifiée à moins que des erreurs ne soient rencontrées. On retrouve souvent des projets qui font systématiquement 2 releases candidates avant chaque release stable.*
- *les numéros de versions entiers concernent généralement des releases stables*
- *pour les autres numéros de versions, comme 0.9.5, 1.4.5 ou 2.6, cela dépend du projet. Les numéros inférieurs à 1 correspondent très généralement à des versions beta ou alpha (exemple: Chiba 0.9.9) et ceux supérieurs à 1 à des versions stables (exemple: Xerces 2.6.2), mais cela n'est pas toujours le cas.*

12 ACTIVITÉS DE SUPPORT

Le support apporté pour les projets Open Source est un des critères important de différenciation par rapport aux autres types de projets. Les différences majeures avec le support de produits non Open Source sont:

- que le support fourni par le projet même (c'est à dire les participants directs au projet) est dépendant de la communauté Open Source
- que n'importe quelle société peut proposer un support à un produit Open Source.

Le support des produits Open Source est donc souvent plus diversifié, mais également plus complet. A noter que le support payant de produits Open Source par une société est un des principaux moyens de financement direct résultant d'un projet Open Source.

Cette partie du document se concentrera plus sur le support fourni par les membres d'un projet Open Source, et moins par des développeurs ou sociétés externes au projet.

12.1 Pratiques de support des projets Open Source

En plus des efforts de coordination à fournir pour faire fonctionner un projet Open Source, un support minimum est généralement attendu de la part des membres, même s'il n'est pas obligatoire (les développeurs ne sont pas tenus à fournir un support pour les produits Open Source, les licences Open Source précisant généralement que le produit est fourni « tel quel »).

Ce support est souvent réalisé simplement via des réponses à des questions posées sur une mailing list ou un forum du projet. Même si ce support est plus souvent fourni par les développeurs du projet, une particularité des projets Open Source est que n'importe quel utilisateur connaissant la réponse à une question est invité à y répondre. La majorité des projets Open Source fournit donc ce support gratuitement, mais il peut également y avoir des projets Open Source qui fournissent un support payant, plus souvent dans le cas de projets financés par une entreprise.

12.1.1 Supports gratuits

Dans les projets Open Source, le forum ou la mailing list ne se limite pas à un support donné par les membres du projet: il arrive fréquemment que des personnes externes au projet décident de répondre à des questions si elles connaissent la réponse. Ceci est à encourager dans la mesure où le forum ou la mailing list n'est alors plus un simple moyen de transfert pour le support technique, mais devient alors vraiment un outil d'échange d'informations entre tous les développeurs et utilisateurs; Dans ce cas, on attend que chaque personne connaissant la réponse à une question posée y réponde, le support devenant alors un support collectif.

Cette technique « donnant-donnant » fonctionne généralement bien, surtout pour les projets impliquant beaucoup de développeurs. Sur les projets où les externes sont surtout des utilisateurs non développeurs, il arrive que seule l'équipe interne au projet réponde aux questions des utilisateurs. Cette situation n'est pas à encourager dans la mesure où il est toujours utile d'avoir plus de personnes répondant au support technique, et aussi car cela implique plus les utilisateurs. La qualité du support dépend également du temps disponible des membres du projet pour l'effectuer, surtout si les autres utilisateurs ne participent pas: certains membres de projets n'ont pas suffisamment de temps pour répondre à toutes les questions, mais cela est assez mal vu pour les projets Open Source car cela sous entend que les membres du projet ne sont pas intéressés par une participation externe au projet.

12.1.2 Supports payants

Même si les utilisateurs s'attendent généralement à un support minimal gratuit fourni par les membres de projets Open Source, il se peut que la société gérant le projet ou qu'une autre société vende en plus du support ou de la documentation concernant ce projet.

Ceci ne pose généralement pas de problème dans le cas où ce support est effectué par une autre société ou par d'autres personnes. Néanmoins, quand les mêmes personnes participent au support payant et au support gratuit, cela peut amener des ambiguïtés sur leur volonté de répondre de manière complète aux questions posées par les utilisateurs. Ceci peut avoir pour conséquence de faire fuir les utilisateurs ainsi que les développeurs motivés pas les principes de l'Open Source. Il existe cependant de nombreux cas d'entreprises fournissant un support gratuit en plus du support payant et qui fonctionne très bien selon ce principe. Pour éviter les malentendus, il est simplement conseillé de bien définir la limite entre le support gratuit et le support payant, et de l'annoncer clairement (sur le site web, la mailing list ou le forum).

Exemple de limite claire entre support gratuit et support payant:

Message de Michael Lipp du 16/12/2003, forum du projet « WFMOpen » utilisant JBoss: This is definitely a JBoss issue. As my company provides support services for JBoss on a commercial basis, my possibilities to help you with this are somewhat limited -- else I would compromise our own business model.

Dans cet exemple, qui provient du forum d'un moteur de workflow nommé WFMOpen et utilisant le serveur applicatif JBoss, l'auteur précise que le support est fourni exclusivement pour WFMOpen, et pas pour les questions spécifiques à JBoss. De plus, aucune mention de la société et du support payant n'apparaît sur le site web du projet. La limite entre le support payant effectué par la société et le support gratuit sur le site web est donc claire, il n'y a pas d'ambiguïté.

Contre exemple: projet JBoss:

Le serveur applicatif JBoss est un projet Open Source géré par une entreprise commerciale (JBoss Inc.), qui vend du support, des formations, etc.

La limite entre le support gratuit et le support commercial de JBoss n'est pas toujours très claire; le forum du site jboss.org propose un support gratuit aux utilisateurs, mais il ne contient en réalité pas beaucoup de réponses aux questions, et souvent des réponses invitant l'utilisateur à consulter la documentation payante.

Discussion intéressante concernant la qualité et la fréquence des réponses apportées sur le forum de JBoss: <http://www.jboss.org/index.html?module=bb&op=viewtopic&t=55157>

De part ceci et d'autres « écarts » par rapport à l'éthique généralement attendue des projets Open Source (notamment des faux messages anonymes – pratique baptisée « astroturfing »- de la part d'employés de JBoss, <http://yro.slashdot.org/yro/04/05/18/2043206.shtml?tid=108&tid=126&tid=149&tid=156&tid=99>), JBoss a perdu en crédibilité dans la communauté Open Source. Des serveurs applicatifs Java concurrents sont d'ailleurs en cours de développement entre autre par des anciens développeurs de JBoss, notamment le projet Apache Geronimo (<http://incubator.apache.org/projects/geronimo/>).

12.2 Demandes d'évolutions / rapports de bugs

En plus du support direct aux utilisateurs via un forum ou une mailing list, il est fréquent que les projets Open Source proposent des systèmes permettant d'encoder soit des demandes d'évolutions, soit des rapports de bugs.

Il existe des outils Open Source implémentant ces fonctionnalités (voir la partie Outils de gestion et de développement du projet page 33).

12.2.1 Demandes d'évolutions

Les outils de demande d'évolutions permettent d'encoder les « vœux » d'utilisateurs ou de développeurs. Comme il s'agit de projets Open Source, ces demandes ne sont pas toujours réalisées, mais elles peuvent influencer les décisions des développeurs sur les futurs développements.

12.2.2 Rapports de bug

Ces outils permettent d'encoder des bugs rencontrés dans le système. La gestion des bugs est alors facilitée du fait que les bugs ont un statut (en cours, résolu, etc), que le système demande des informations précises à l'utilisateur (système, version qui pose problème, etc), etc.

12.2.3 Utilisation de ces outils

Ces outils de gestion des demandes d'évolutions et des bugs ne sont généralement pas spécifiques aux projets Open Source. Néanmoins, leur utilisation l'est: Dans les projets Open Source, ils sont ouverts à toute personne désirant proposer une évolution ou rapporter un bug (parfois il est demandé de s'enregistrer sur le site pour y accéder).

Le processus de décision associé dépend de l'organisation du projet: dans certains cas les utilisateurs inscrits peuvent voter pour signifier qu'ils aimeraient voir un bug résolu ou bénéficier une évolution. Certains projets s'engagent à respecter ces votes et à définir les priorités des développements en fonction des résultats de votes. D'autres projets utilisent simplement ces votes pour garder une idée des développements les plus attendus.

Exemple: KDE bug tracking system (<http://bugs.kde.org/>), conditions de participations, exemples de traitements des votes

For participating you need a personal account which will gain you the ability to post reports and comments as well as voting for specific reports and observe development [...]

- [*The most hated bugs*](#)
- [*The most severe bugs*](#)
- [*The most wanted features*](#)

12.3 Documentation/Guide utilisateurs

L'écriture de la documentation d'un programme n'est généralement pas une tâche appréciée par les développeurs. Étant donné que beaucoup de développeurs Open Source travaillent sur le projet par choix pendant leur temps libre, c'est une tâche souvent moins bien réalisée que sur les projets propriétaires (ou pas réalisée du tout).

Cependant, certains projets ont choisi de gérer la création de la documentation de la même manière que les développements Open Source, c'est à dire de manière libre et collaborative: la documentation est ouverte, chaque utilisateur peut y participer. Le projet doit pour cela créer sa documentation en ligne sur le site du projet, via un outil de CMS (Content Management System) ou de gestion collaborative de documents. Ce genre de système fonctionne bien si la nombre d'utilisateurs et de développeurs est suffisamment grand, mais il est inutile d'espérer que des utilisateurs écrivent la documentation pour vous dans le cas contraire.

L'article « Open-Source Documentation: in search of user-driven, just-in-time writing » détaille bien l'utilisation de techniques Open Source pour la création de documentation. Il explique également que le fait d'écrire la documentation de manière collaborative revient en fait à concentrer les connaissances des utilisateurs et développeurs dans une « source de documentation » pouvant être assimilée à un code source d'un programme à partir duquel on peut réaliser plusieurs documents.

Exemple de documentation collaborative: Documentation de l'outil de CMS/groupware « Tikiwiki », <http://doc.tikiwiki.org/>

13 RÉFÉRENCES

<http://www.globenet.org/horizon-local/perso/guideanim.html> - Animation de petits groupes de travail sur Internet

<http://www.f-d.org/animationgroupe.htm> - Animation du groupe – Marc Thiébaud

http://greg.abstrakt.ch/docs/OSP_framework.pdf - A Framework for Open Source Projects- Gregor Rothfuss- Master thesis

cvsbook.red-bean.com/OpenSourceDevWithCVS_2E.tar.gz- CVS book

www.fsf.org/copyleft/gpl.html - Free Software Foundation (1989/1991): GNU General Public License (Version 2).

www.fsf.org/philosophy/free-sw.html - Free Software Foundation (1996/2003): The Free Software Definition.

www.freebsd.org/doc/en_US.ISO8859-1/articles/committers-guide/rules.html- FreeBSD (1999/2003).

[The FreeBSD Committers' Big List of Rules.](#)

freshmeat.net/ - Freshmeat (2003): Website.

<http://www.ibiblio.org/fosphost/exhost.htm> – Comparison of OS projects hosting solutions

<http://www.openchannelsoftware.org/projects/> - OS projects hosting

<http://www.netobjectdays.org/pdf/03/papers/ws-ossie/508.pdf> - Interacting with the Open Source Community – Effective models for a Traditional IT-business -Dr. Wolfgang Thronicke and Frank Berger

http://www.dwheeler.com/oss_fs_why.html - Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!David A. Wheeler

<http://techrepublic.com.com/5100-22-5293051.html> – Breaking the rules with OS

<http://en.tldp.org/HOWTO/Software-Release-Practice-HOWTO/index.html> - Software-Release-Practice-HOWTO – Eric Raymond

http://www.loiclemeur.com/france/2004/06/les_blogs_et_le.html - Les blogs et leurs applications en entreprise et pour les médias

<http://fr.wikipedia.org/wiki/Blog> – Blog dans Wikipedia FR

<http://fr.wikipedia.org/wiki/Wiki> – Wiki dans Wikipedia FR

<http://www.adullact.org/>- ADULLACT - l'Association des Développeurs et des Utilisateurs de Logiciels Libres pour les Administrations et les Collectivités Territoriales

Guido Hertel Sven Niedner Stefanie Herrmann - Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel , University of Kiel, at <http://opensource.mit.edu/papers/preso-hertel.pdf>

Carrier, James (1991): Gifts, Commodities, and Social Relations: A Maussian View of Exchange. In: Sociological Forum 6 (1991), March, Nr. 1, S. 119–136
links.jstor.org/sici?sici=0884-8971... #

Dempsey, Bert J. / Weiss, Debra / Jones, Paul / Greenberg, Jane (2002): Who is an Open Source Developer? Profiling a community of Linux developers. In: Communications of the ACM 45 (2002), February, Nr. 2, S. 67–72 [doi.acm.org/10.1145/503124.503125](https://doi.org/10.1145/503124.503125) #

Mejias, Ulises Ali (2004): A del.icio.us study: Bookmark, Classify and Share, http://ideant.typepad.com/ideant/2004/12/a_delicious_stu.html#more

«Composants dans l'Ingénierie des SI» Franck Barbier, Corinne Cauvet, Mourad Oussalah, Dominique Rieu, Sondes Bennisri, Carine Souveyet

« OO Design Quality Metrics » (1994), Robert Martin

Berglund E. et Priestley M., Open-Source Documentation: in search of user-driven, just-in-time writing. SIGDOC 2001

Kent Beck, « Simple Smalltalk Testing: With Patterns »

(<http://www.xprogramming.com/testfram.htm>)

Eric von HIPPEL, Stefan THOMKE et Mary SONNACK : "L'innovation chez 3M" (Harvard Business Review)

14 ANNEXES**14.1 Répartition des licences sur sourceforge.net au 25/11/2004**

Licence	Nombre de projets
Academic Free License (AFL)	169
Apache License 2.0	238
Apache Software License	929
Apple Public Source License	51
Artistic License	1149
Attribution Assurance License	17
BSD License	4131
Common Public License	397
CUA Office Public License Version 1.0	2
Eclipse Public License	21
Eiffel Forum License	7
Eiffel Forum License V2.0	16
Entessa Public License	2
EU DataGrid Software License	1
Fair License	5
Framework Open License	1
GNU General Public License	39743
GNU Library or Lesser General Public License	6320
Historical Permission Notice and Disclaimer	10
IBM Public License	74
Intel Open Source License	22
Jabber Open Source License	40
Lucent Public License	1
MIT License	1027
MITRE Collaborative Virtual Workspace License (CVW)	2
Motosoto License	1
Mozilla Public License 1.0 (MPL)	225
Mozilla Public License 1.1 (MPL 1.1)	699
NASA Open Source Agreement	1
Nethack General Public License	26
Nokia Open Source License	8
OCLC Research Public License 2.0	1
Open Group Test Suite License	15
Open Software License	263
PHP License	106
Python License (CNRI Python License)	138
Python Software Foundation License	10
Qt Public License	214
RealNetworks Public Source License V1.0	1
Reciprocal Public License	16
Ricoh Source Code Public License	7
Sleepycat License	20
Sun Industry Standards Source License (SISSL)	47

Licence	Nombre de projets
Sun Public License	45
Sybase Open Watcom Public License	1
University of Illinois/NCSA Open Source License	37
Vovida Software License 1.0	6
W3C License	26
wxWindows Library Licence	48
X.Net License	6
zlib/libpng License	285
Zope Public License	31
Other/Proprietary License	988
Public Domain	1559

14.2 Les licences de logiciels libres compatibles avec la G.P.L.

Source: [www.gnu.org \(http://www.gnu.org/licenses/license-list.fr.html\)](http://www.gnu.org/licenses/license-list.fr.html), avec quelques adaptations, notamment le retrait des prises de positions

[La GNU General Public License, la License publique générale GNU](#), abrégée en GNU GPL.

Il s'agit d'une licence de logiciel libre et d'un «gauche d'auteur».

[La GNU Lesser General Public License](#), abrégée en GNU LGPL.

Il s'agit d'une licence de logiciel libre, mais pas aussi sévère en tant que gauche d'auteur, car elle permet l'intégration de modules non libres. Elle est compatible avec la GPL de GNU.

En passant de la version 2 à la version 2.1, le terme de GNU Library General Public License (License Publique Générale GNU pour les Bibliothèques), est devenu GNU Lesser General Public License (License publique moins générale GNU), ce qui reflète mieux son but réel. En effet, quoique son intitulé puisse faire croire, elle ne s'applique pas uniquement aux bibliothèques de programmes tandis que la [GPL de GNU est parfois plus appropriée pour ce genre de cas.](#)

La licence de Guile

Elle se compose de la GNU GPL, augmentée d'une assertion spéciale donnant la permission d'inclure des logiciels non libres. Il en résulte un gauche d'auteur souple, compatible avec la GPL de GNU. Elle est recommandée dans des cas spéciaux uniquement, à peu de choses près dans les mêmes circonstances que la [LGPL](#).

La licence sur les unités d'interprétation du compilateur GNU Ada

Celle-ci ressemble beaucoup à la licence de Guile.

[La licence de X11.](#)

Il s'agit d'une licence de logiciel libre simple et permissive, sans gauche d'auteur, compatible avec la GPL de GNU. Les anciennes versions de XFree86 utilisaient la même licence, et quelques variantes actuelles de XFree86 l'utilisent également. Les licences ultérieures de XFree86 sont distribuées sous la licence XFree86 1.1 (qui est incompatible avec la GPL). La licence est parfois appelée «licence du MIT» mais ce terme est trompeur : le MIT a publié ses logiciels sous diverses licences.

[La licence d'Expat.](#)

Une licence simple et permissive de logiciel libre sans gauche d'auteur, compatible avec la GPL de GNU. D'une manière ambiguë, on l'appelle parfois **la Licence du MIT**.

[Licence de copyright ML standard du New Jersey.](#)

Une licence simple et permissive de logiciel libre sans gauche d'auteur, compatible avec la GPL de GNU.

Le domaine public.

Le « domaine public » n'est pas une licence, mais cela signifie plutôt que le contenu en question est dépourvu de droit d'auteur et qu'aucune licence n'est exigée. Cependant, dans la pratique, si une oeuvre est dans le domaine public, c'est à peu près comme si elle était munie d'une licence de logiciel libre entièrement permissive, sans gauche d'auteur. La présence dans le domaine public est compatible avec la GPL de GNU.

[La licence générale Cryptix.](#)

Il s'agit d'une licence de logiciel libre simple et permissive, sans gauche d'auteur, compatible avec la GPL de GNU. Très semblable à la licence X11.

[La licence modifiée de BSD.](#)

Il s'agit de la licence BSD d'origine, mais privée de sa clause publicitaire. C'est une licence de logiciel libre simple et permissive, sans gauche d'auteur, compatible avec la GPL de GNU.

Si ce que vous souhaitez, c'est une licence permissive sans gauche d'auteur, alors la licence BSD modifiée constitue un bon choix. Cependant, recommander la «Licence BSD» est une opération risquée du fait de la confusion possible avec la version *d'origine*, de [la licence BSD](#). Pour éviter ce risque, suggérez plutôt l'emploi de la licence X11. La licence X11 et la licence BSD révisée sont très proches l'une de l'autre.

[La licence sur la ZLib.](#)

Licence de logiciel libre compatible avec la GPL.

La licence de iMatix Standard Function Library

Licence de logiciel libre compatible avec la GPL.

[La licence et la notice des logiciels du W3C.](#)

Il s'agit d'une licence de logiciel libre compatible GPL.

[La licence de Berkeley Database](#) (appelée aussi Licence du logiciel Sleepycat).

Il s'agit d'une licence de logiciel libre, compatible avec la GPL.

[La licence de OpenLDAP, version 2.7.](#)

Il s'agit d'une licence permissive, sans gauche d'auteur, compatible avec la GPL de GNU.

[La licence de Python, jusqu'à la version 1.6a2.](#)

Il s'agit d'une licence de logiciel libre compatible avec la GPL. Attention, certaines versions plus récentes de Python sont placées sous une licence différente ([voir plus bas](#)).

[La licence de Python 2.0.1, 2.1.1 et suivantes.](#)

Il s'agit d'une licence de logiciel compatible avec la GPL. Attention, certaines versions intermédiaires de Python (de 1.6b1, à 2.0 et 2.1) sont placées sous une licence différente ([voir plus bas](#)).

La licence de Perl.

Cette licence est une juxtaposition de la [Licence artistique](#) et de la [GPL de GNU](#), ce qui signifie que vous pouvez choisir entre les deux. Il s'agit bien d'une licence de logiciel libre, mais cela ne peut pas constituer un gauche d'auteur. Elle est bien sûr compatible avec la [GPL de GNU](#), puisque cette dernière est l'un des deux choix.

[La Licence artistique clarifiée.](#)

Il s'agit d'une licence de logiciel libre compatible avec la GPL. Elle propose l'ensemble des corrections nécessaires pour remédier à l'imprécision de la [Licence artistique d'origine](#).

[La Licence artistique 2.0](#)

Cette licence est une licence de logiciel libre compatible avec la GPL de GNU. À notre connaissance elle n'est pas encore utilisée, mais son usage fait l'objet de discussions pour Perl 6 comme élément d'une structure de licence duale.

Dans le cas où vous souhaiteriez publier un programme sous la [Licence artistique d'origine](#), envisagez plutôt de choisir cette version révisée.

[La Licence publique de Zope \(ZPL\), version 2.0.](#)

La ZPL est une licence de logiciel libre simple, permissive, sans gauche d'auteur. Elle est compatible avec la GPL.

[La Intel Open Source License \(publiée par l'OSI\)](#)

Il s'agit d'une licence de logiciel libre compatible avec la GPL de GNU.

La licence sur Javascript de Netscape

Il s'agit de la juxtaposition de la [Netscape Public License](#) et de la [GPL de GNU](#). De ce fait, c'est une licence de logiciel libre compatible avec la GPL de GNU, mais en tant que gauche d'auteur elle est un peu faible.

Cette licence est un bon choix si vous souhaitez écrire un paquetage à la fois compatible GPL et compatible MPL. Cependant vous obtiendrez le même résultat avec la LGPL ou bien la licence de Guile.

Une telle licence de juxtaposition peut être un bon choix si jusque là vous avez utilisé la MPL et que vous voulez évoluer vers une licence compatible GPL sans retirer pour autant les permissions que vous avez accordées pour les précédentes versions.

[La licence eCos, version 2.0](#)

La licence eCos version 2.0 est une licence de logiciel libre compatible avec la GPL. Elle reprend les termes de la GPL, avec une exception qui autorise les liens avec d'autres logiciels non placés sous GPL. Cette licence a les mêmes [points faibles](#) que la LGPL.

[La licence Eiffel Forum, version 2](#)

Il s'agit d'une licence de logiciel libre compatible avec la GPL de GNU. À noter : les [versions précédentes](#) de la licence Eiffel ne sont pas compatibles avec la GPL.

[La licence de Vim, version 6.1 et suivantes](#)

Il s'agit d'une licence de logiciel libre, partiellement gauche d'auteur mais pas vraiment. Elle est compatible avec la GPL, par une clause de conversion explicite.

14.3 Les licences de logiciels libres non compatibles avec la GPL

Source: [www.gnu.org \(http://www.gnu.org/licenses/license-list.fr.html\)](http://www.gnu.org/licenses/license-list.fr.html), avec quelques adaptations, notamment le retrait des prises de positions

Les licences suivantes constituent des licences de logiciels libres mais sont non compatibles avec la GPL de GNU :

La licence XFree86 1.1.

Il s'agit d'une licence de logiciel libre simple et permissive, sans gauche d'auteur, incompatible avec la GPL de GNU à cause des conditions qui s'appliquent à toute la documentation dans la distribution qui contient les remerciements.

Il y a actuellement plusieurs variantes de XFree86, et seules quelques unes utilisent cette licence. Certaines continuent à utiliser la licence X11.

[La Licence publique générale Affero](#)

La Licence publique générale Affero est une licence de logiciel libre, avec gauche d'auteur, incompatible avec la GPL de GNU. Elle est constituée de la GPL version 2 et d'un paragraphe supplémentaire ajouté par Affero avec l'accord de la FSF. Le nouveau paragraphe (2d), concerne la distribution des programmes applicatifs à travers les services web et les réseaux. Il rend la licence GPL Affero incompatible avec la GPL version 2, mais la rédaction de ce paragraphe rend possible la compatibilité ascendante de la future GPL version 3 avec la GPL Affero.

[La Licence publique Arphic.](#)

C'est une gauche d'auteur de logiciel libre incompatible avec la GPL. Elle est d'un usage habituel pour les polices de caractères, et dans ce cadre précis l'incompatibilité GPL ne pose pas de problème.

[La Licence BSD d'origine.](#)

(Note : sur la page référencée ci-dessus, la licence en question figure au paragraphe «UCB/LBL».) C'est une licence simple et permissive de logiciel libre, mais ce n'est pas une gauche d'auteur et elle comporte une clause publicitaire, qui peut provoquer des problèmes pratiques, et en particulier l'incompatibilité avec la GPL de GNU.

Si ce que vous souhaitez est une licence simple et permissive sans gauche d'auteur, il est conseillé d'adopter la [licence BSD modifiée](#) ou la licence X11. Cependant, il n'y a aucune raison de ne pas utiliser les programmes publiés sous la licence BSD originelle.

[La licence OpenSSL.](#)

La licence OpenSSL est la conjonction de deux licences, l'une d'elle est la licence de SSLeay. Vous devez suivre les deux. La combinaison résulte en une licence de logiciel libre avec gauche d'auteur qui est incompatible avec la GPL GNU. Elle contient également une clause publicitaire comme la licence BSD d'origine et la licence Apache.

[La licence «Academic Free License», version 1.1.](#)

La licence «Academic Free License», version 1.1, est une licence de logiciel libre sans gauche d'auteur incompatible avec la GPL de GNU pour diverses raisons. Elle est présentée comme une «mise à jour compatible» avec les «licences comme celle de BSD et celle du MIT», mais ce n'est pas le cas. La licence BSD révisée et la licence du MIT sont compatibles avec la GPL, mais pas l'AFL.

L'AFL est incompatible avec la GPL pour les deux raisons suivantes : la première, parce que les règles qu'elle énonce sur l'usage de la marque commerciale vont en fait encore plus loin que ce que réclame le droit des marques dans de nombreux pays, interdisant certaines choses qui seraient légales sans cela.

Une deuxième incompatibilité vient de la clause de «mutuelle terminaison de l'action en brevets» qu'elle contient. En dehors de la question difficile de savoir si c'est ou non une bonne idée, cette clause rend la licence incompatible avec la GPL.

[La licence «Open Software License», version 1.0.](#)

La licence «Open Software License», version 1.0, est une licence de logiciel libre. Son auteur affirme qu'il s'agit d'une gauche d'auteur, mais la FSF exprime des réserves sur la validité des garanties de gauche d'auteur indiquées. Pour diverses raisons, elle est incompatible avec la GPL.

[La licence d'Apache version 1.0.](#)

C'est une licence simple et permissive de logiciel libre sans gauche d'auteur présentant des [problèmes pratiques](#) semblables à ceux de la licence BSD, avec y compris l'incompatibilité avec la GPL de GNU.

[La Licence d'Apache version 1.1.](#)

Licence de logiciel libre sans gauche d'auteur, permissive, avec quelques aspects qui la rendent incompatible avec la GPL de GNU.

[La Licence d'Apache version 2.0](#)

C'est une licence de logiciel libre mais qui est incompatible avec la GPL. La Licence d'Apache est incompatible avec la GPL car elle a des conditions spécifiques qui ne sont pas dans la GPL : il y a des cas de résiliation de brevet que la GPL ne requière pas. (Nous ne pensons pas que ces cas de résiliation soient une mauvaise idée en soi, mais néanmoins, ils sont incompatibles avec la GPL GNU.)

[La Licence publique de Zope, version 1.](#)

C'est une licence libre avec gauche d'auteur simple, assez permissive, comportant des [problèmes pratiques](#) semblables à ceux de la licence BSD, qui conduisent en particulier à l'incompatibilité avec la [licence GPL de GNU](#).

La dernière version de Zope est publiée sous une licence compatible avec la GPL.

[La licence de xinetd](#)

Il s'agit d'une licence de logiciel libre avec gauche d'auteur incompatible avec la GPL. Elle est incompatible, parce que les conditions supplémentaires qu'elle pose sur la redistribution des versions modifiées entrent en contradiction avec les exigences correspondantes de la GPL.

[La licence de Python version 1.6b1 et suivantes, ainsi que 2.0 et 2.1.](#)

C'est une licence de logiciel libre incompatible avec la GPL de GNU. La première incompatibilité est le fait que la licence de Python est soumise aux lois de l'État de Virginie (USA). La GPL ne permet pas cela.

[L'ancienne licence d'OpenLDAP \(version 2.3\)](#)

Licence permissive de logiciel sans gauche d'auteur avec quelques exigences (paragraphe 4 et 5) qui la rendent incompatible avec la GPL de GNU. Notez bien que la version la plus récente de OpenLDAP est soumise à une [licence différente](#), compatible avec la GPL de GNU.

[La Licence publique IBM, version 1.0](#)

Bien que ce soit une licence qualifiant un logiciel libre, elle est incompatible avec la [GPL](#).

La Licence publique IBM est incompatible avec la GPL en raison de diverses exigences spécifiques qui ne se trouvent pas dans la GPL.

Elle exige notamment que certains droits soient accordés en-dehors de ce que la GPL prévoit. Ces droits constituent une incompatibilité avec la GPL.

[La Common Public License, version 1.0](#)

C'est une licence de logiciel libre mais elle n'est pas compatible avec la [GPL](#).

La Common Public License est incompatible avec la GPL parce qu'elle énonce diverses exigences spécifiques qui ne se trouvent pas dans la GPL.

Notamment, elle exige que certaines licences de brevet soient données, ce que la GPL n'exige pas.

[La licence de Phorum, version 1.2](#)

C'est une licence de logiciel libre mais elle est incompatible avec la [GPL](#). En particulier, les termes des paragraphes 3 et 4 sont incompatibles avec la GPL.

[La Licence publique du Projet LaTeX](#)

Cette licence ne contient pas tous les termes de la distribution de LaTeX. D'après ce qu'on peut y lire il s'agit bien d'une licence de logiciel libre, mais elle est incompatible avec la [GPL](#) du fait de nombreuses exigences qui ne figurent pas dans la GPL.

Cette licence comporte de nombreuses limitations sur la façon de publier les versions modifiées, notamment une clause qui la place à l'extrême limite de l'acceptable : le fait que tout fichier modifié doit être renommé.

Pour LaTeX cette exigence est acceptable parce que ce logiciel comporte une fonctionnalité permettant de faire correspondre des noms de fichiers entre eux, de spécifier par exemple : «utiliser le fichier bar en même temps que le fichier foo». À cause de cette fonctionnalité, l'exigence en question est simplement gênante; mais sans cette fonctionnalité la même clause constituerait un obstacle rédhibitoire qui mènerait à classer la licence comme non libre.

Dans la LPPL il est précisé que certains fichiers, dans certaines versions de LaTeX, peuvent opposer d'autres limitations qui les rendraient alors non libres. À cause de cela, il vous faudra sans aucun doute redoubler de vigilance si vous voulez produire une version de LaTeX qui soit un logiciel libre.

Note : les commentaires ci-dessus portent sur la version 1.2 de la LPPL datée du 3 septembre 1999.

[La Licence publique Mozilla \(MPL\).](#)

Il s'agit d'une licence de logiciel libre, pas très stricte en tant que gauche d'auteur; contrairement à la [licence X11](#) elle présente des restrictions complexes qui la rendent incompatibles avec la [GPL de GNU](#). En effet, on ne peut pas, légalement, lier un module couvert par la GPL et un module couvert par la MPL.

Cependant, la licence MPL 1.1 permet (section 13) à un programme ou à une portion de programme d'offrir le choix entre la MPL et une autre licence. Dans le cas d'une partie de programme qui offre la GPL de GNU comme choix possible, alors la licence de cette partie de programme est compatible avec la GPL.

[La licence Open Source Netizen \(NOSL\) version 1.0.](#)

C'est une licence de logiciel libre qui ressemble pour l'essentiel à la Mozilla Public License version 1.1. De même que la MPL, la NOSL présente des restrictions complexes qui la rendent incompatible avec la GPL. Ainsi, on n'a pas le droit de lier un module couvert par la GPL avec un autre couvert par la NOSL.

[Interbase Public License, Version 1.0.](#)

C'est une licence de logiciel libre qui ressemble pour l'essentiel à la Mozilla Public License version 1.1. De même que la MPL, l'IPL présente des restrictions complexes qui la rendent incompatible avec la GPL. Ainsi, on n'a pas le droit de lier un module couvert par la GPL avec un autre couvert par l'IPL.

[Sun Public License.](#)

Dans l'esprit de la Licence publique de Mozilla : une licence de logiciel libre incompatible avec la GPL de GNU. Attention à ne pas confondre avec la [Sun Community Source License](#), qui elle n'est pas une licence de logiciel libre.

[La licence Open Source de Nokia](#)

Dans l'esprit de la Licence publique de Mozilla : une licence de logiciel libre incompatible avec la GPL de GNU.

[La Licence publique de Netscape \(NPL, en anglais\)](#)

C'est une licence de logiciel libre, pas très stricte en tant que gauche d'auteur et incompatible avec la GPL de GNU. Elle est formée de Mozilla Public License et d'une clause permettant à la société Netscape d'utiliser le code que vous y avez ajouté *y compris dans leurs versions propriétaires du programme*. Bien entendu, vous n'avez pas le droit d'utiliser leur code en contrepartie.

Jabber Open Source License, Version 1.0

C'est une licence de logiciel libre incompatible avec la GPL. Elle donne le droit de redistribuer sous une certaine catégorie de licences, celles qui reprennent toutes les exigences de la licence Jabber. Comme la GPL ne fait pas partie de ladite catégorie, il est impossible de redistribuer sous GPL du code soumis à cette licence. D'où l'incompatibilité.

[Sun Industry Standards Source License 1.0](#)

Licence de logiciel libre, avec gauche d'auteur pas très sévère, incompatible avec la GPL de GNU mais pas tant à cause de l'esprit, surtout à cause de certains détails.

[La Licence publique 'Q' \(QPL\), version 1.0.](#)

Il s'agit d'une licence de logiciel libre qui n'est pas une gauche d'auteur, incompatible avec la GPL de GNU. Elle présente aussi de gros inconvénients pratiques, parce que les sources modifiés ne peuvent être redistribués que sous forme de correctifs.

Puisque la QPL est incompatible avec la GPL de GNU, il est tout à fait impossible de lier un programme sous QPL avec un autre sous GPL.

Cependant, si vous avez écrit un programme utilisant une bibliothèque couverte par la QPL (appelée FOO) et que vous voulez distribuer ce programme sous la GPL de GNU, là c'est possible sans problème.

La licence FreeType

La licence FreeType est une licence de logiciel libre sans gauche d'auteur incompatible avec la GPL pour des raisons techniques.

[La licence de PHP version 3.0.](#)

Cette licence est celle de la majeure partie du code de PHP4. C'est une licence de logiciel libre sans gauche d'auteur, incompatible avec la GPL de GNU.

La licence de Zend, version 2.0

Cette licence est utilisée pour une partie de PHP4. Il s'agit d'une licence de logiciel libre sans gauche d'auteur incompatible avec la GPL de GNU, qui présente quelques [problèmes pratiques](#) comparables à ceux de la licence BSD d'origine.

[La licence Vita Nuova Liberal Source](#)

C'est une licence de logiciel libre avec gauche d'auteur, incompatible avec la GPL GNU.

[La licence Plan 9 du 9 juin 2003](#)

C'est une licence de logiciel libre, incompatible avec la GPL GNU.

[La licence Apple Public Source \(APSL\), version 2.](#)

C'est une licence de logiciel libre, incompatible avec la GPL GNU.

14.4 Questionnaire aux SSLL

14.4.1 Liste des Questions posées

Q1 : Pourquoi n'avoir choisi que d'intégrer des composants Open source? Par choix? expérience? pour instaurer une alternative?

Q2 : Avez vous des composants phares? si oui lesquels? à quelle cible business sont-ils destinés ?

Q3 : Quelle est votre contribution aux projets Open Source que vous intégrez chez les clients (rôle moteur, debugage, developpement)?

Q4 : Dans le cadre de l'intégration de ces produits, à quelles étapes du cycle de vie du projet intervenez vous?

Q5 : Considérez vous vos missions identiques à celles d'intégrations plus classiques prestées par les SSII?

Q6 : Avez vous un plan de communication spécifique sur ce genre de composants ? quels genres d'atouts, d'images sont présentés aux clients ? quel est la réaction des clients?

Q7 : Comment gérez vous l'évolution des composants Open Source déjà installés chez les clients?

Q8 : Pensez vous que ces composants ou solutions induisent un nouveau modèle économique?

Q9 : Pensez vous que l'Open Source est porteur de nouveaux métiers dans l'informatique ? si oui lesquels ? et quelles sont leurs caractéristiques ?

Q10 : Pour vous quelles doivent être les qualités principales d'un bon consultant/expert en Open source?

TAIKA - INFORMATIQUE LIBRE ET OPEN SOURCE
17, rue Mélingue - 14000 CAEN
Tél : 02.31.85.92.70 - Fax : 02.31.85.53.71
contact@taika.fr - www.taika-informatique.com

Description:

L'agence TAIKA est une société informatique spécialisée dans l'intégration de solutions libres, la formation et le développement d'applications en "Open Source" sous Linux.

Q1

Nous sommes prestataire de services informatiques. A ce titre nous recherchons les meilleures solutions pour nos clients. Nous pensons que le libre est ce qu'il y a de mieux, aujourd'hui, en informatique et que les logiciels et formats libres sont l'avenir de l'informatique.

Q2

Nous avons TAIKA PGI Suite en développement depuis près d'un an et demi maintenant. Ce produit est très orienté client/serveur avec des interfaces XUL pour les clients et une base de données Mysql pour le serveur. XUL devient notre spécialité. si vous ne connaissez pas je vous conseil de parcourir le site <http://www.xulfr.org>.

Q3

Tout d'abord nous sommes tous membres actifs d'un Lug (Calvix pour moi). A partir du moment ou la société intègre un logiciel libre, nous cherchons à défendre au mieux les intérêts de ce logiciel. Par exemple, pour Mandrake, Nous sommes partenaires, membre du club, et incitons nos clients utilisant cette distrib à devenir membre du club également.

De plus :

- Les développements de TAIKA sont libres (TAIKA PGI suite et LINAO voir linao.taika.fr).
- Nous lancons ce mois-ci une offre ASP basée sur Egroupware et reversons 10% des abonnements à la communauté Egroupware

Q4

je ne comprend pas votre question.

Q5

Non

Q6

le point le plus important pour nos clients : avoir des formats de fichiers perennent et ouverts, maitriser parfaitement le fonctionnement de leur système d'information, avoir une informatique parfaitement adaptée et parfaitement adaptable à leurs besoins.

Q7

par contrat d'assistance ou contrat d'infogérance.

Q8

oui, vraiment différent de l'existant. Très orienté sur le service, la transparence et la compréhension des outils que nous distribuons.

Q9

Je ne sais pas si l'on peut vraiment parler de nouveaux métiers, mais plutôt d'une approche différente, plus compétente.

Q10

Etre à l'écoute de ses clients et parfaitement connaître le monde de l'entreprise et le monde du libre. Avoir de solides connaissances en développement, administration système, direction de projet.

ATREAL

113, Bd. de Pont-de-Vivaux, 13010 Marseille France
Email : contact@atreal.net -
Tél: +33 (0)4 91 29 42 81 - Fax: +33 (0)4 91 29 42 82

Description:

SSSL de référence du Sud de la France, spécialiste des technologies ZONE et PLONE pour le Développement d'applications, de sites et de services orientés Web. Accompagnement, maîtrise d'oeuvre et formation en migration de systèmes, de réseaux et d'applications.

Q1

Nous avons choisi de travailler sur la base de logiciels libres, ce qui est plus qu'Open Source. En effet, le modèle des logiciels libres permet à participer à un vivier d'entreprises qui co-développent un super-logiciel qu'elles ne pourraient pas réaliser autrement. Ce mode de développement permet d'augmenter très sensiblement la capacité d'innovation d'une PME en lui donnant la puissance d'un amalgame d'acteurs.

Q2

Nous proposons des solutions d'applications web sur mesure. Notre approche, par composants intégrables, nous permet de ne développer pour chaque client que le ou les modules qui lui sont nécessaires, le reste de la prestation technique étant l'assemblage et l'intégration de composants existants, tout en prenant en compte les spécificités du client tant dans ses habitudes de travail que sur le plan technique. Nous proposons par exemple une plateforme d'Espace Numérique de Travail pour l'éducation nationale, intégrant dans le même espace une gestion électronique de documents, des espaces de travail collaboratif pour les enseignants et les élèves, des outils de travail collaboratif, un logiciel de gestion des absences en temps réels directement dans les salles de classes, etc. Autre exemple : un bureau virtuel pour les PME permettant de connecter le système de fichiers local à l'extranet et la gestion documentaire. Les documents sauvegardés dans le lecteur F par exemple sont stockés de façon transparente dans notre serveur d'applications et bénéficient d'une prise en charge automatique permettant l'indexation plein texte du document, sa prise en charge en extranet (droits de publication/visualisation), pour les contenus multimédias la visualisation en streaming au sein du portail web, etc. Il s'agit ici de la fusion entre le dossier de travail local et l'extranet/intranet dans lequel les documents que l'on souhaite peuvent être rendu accessibles.

Q3

Un fondamental est le débogage. Il est fondamental que les SSSL participent activement à la pérennité des logiciels qu'elles utilisent. Lorsque nous rencontrons des bugs, nous entrons en contact avec les développeurs du code concerné et soit participons à la résolution du problème soit fournissons les informations les plus détaillées possibles afin d'aider à résoudre le problème. Nous développons aussi nos composants libres que nous mettons à disposition des autres, surtout basés sur le serveur d'applications Zope.

Q4

Cela dépend essentiellement du client et de ses besoins. Lorsque des

composants existants correspondent en partie aux besoins du client, nous apportons les améliorations nécessaires et les fournissons aux équipes de développeurs du composant. Lorsque le besoin est nouveau et que le client est prêt à en financer le développement, nous développons le composant et l'ajoutons à la bibliothèque de composants disponibles ; c'est ce qui permet d'accroître la compétitivité du groupe de développeurs.

Q5

Nos modes de fonctionnement sont foncièrement différents. Une SSII traditionnelle cherchera le composant le plus abouti pour l'intégrer à son travail, et ne produira un composant que sur demande expresse du client. Elle entrera très rarement dans une démarche de construction conjointe avec d'autres développeurs : les méthodes de travail sont très éloignées des méthodes traditionnelles.

Q6

La réaction des clients est totalement éhéroogène. Peu de clients souhaitent communiquer sur les composants libres, même lorsqu'ils y participent en toute connaissance de cause. A contrario, certains clients présentent leur projet libre comme réalisé par eux et le valorisent au maximum. Nous respectons le souhait de chaque client, l'important étant que l'ensemble du travail étende la base de composants libres.

Q7

Comme n'importe quelle SSII, nous assurons le support sur les logiciels que nous développons. Lorsque des évolutions hors débbugage sont nécessaires, nous proposons un devis, que le client peut accepter ou pas. Si le client le souhaite, d'autres SSSL peuvent prendre le relais sur le travail réalisé. J'ai créé à cet effet le réseau d'entreprises Libertis, qui garantie au client que chaque SSSL membre a le devoir de communiquer tous les éléments nécessaires à la poursuite du travail à un autre prestataire. La sensation de liberté est fondamentale, et il nous semble très important de la garantir.

Q8

Indéniablement. Nous introduisons la liberté du client, son indépendance vis-à-vis du prestataire. Le choix doit se faire sur les compétences, la qualité de services et le prix.

Q9

De nouveaux métiers, je ne sais pas. De nouvelles méthodes de travail et de communication, oui.

Q10

L'écoute du client, comme pour tout prestataire, la connaissance intime du fonctionnement des communautés de développeurs afin de pouvoir entrer en contact avec eux de manière constructive. Le mode de communication, en rapport avec les méthodes de travail, est très différent des SSII. Un développeur qui ne "pense pas" logiciel libre peut avoir du mal à s'y retrouver et ne pas être accepté. C'est pour cette raison que nous avons imposé, pour qu'une société devienne membre de Libertis, qu'elle ait à son actif au moins une contribution aux logiciels libres, ce qui assure qu'elle sait de quoi elle parle.

BASHPROFILE SARL

4b Rue Augusta
13010 - Marseille
Tél.: 04 91 78 69 22
Fax: 04 91 78 69 22

Description:

BASHPROFILE est une jeune Société de Services en Logiciels libres basée à Marseille. Propose au secteur des PME-PMI des solutions informatiques Systèmes Réseaux et Sécurité basées exclusivement sur des technologies opensource (Linux et Unix like). Spécialisés dans la mise en place de passerelles sécurisées . Accompagne les entreprises dans leurs désir de migration vers des technologies Open Source.

Q1

Nous avons choisi d'intégrer des solutions Open Source d'une part par expérience (ingénieur certifié), et pour d'autres raisons qui sont :

- une meilleure qualité des produits (logiciels) contrairement à d'autres systèmes propriétaires que je ne citerai pas ici, mais dont vous connaissez le nom.
- ensuite ce choix est aussi un choix basé sur l'honnêteté. En effet ou les marges et les tarifs proposés dans le cadre de solutions propriétaires sont prohibitives, pour des produits qui n'ont pas la qualité requise(stabilité,sécurité, réactivité en matière de développement).
- les aspects configuration et souplesse (grand choix de paquetages d'environnement) sont également un atout majeur car nous sommes pour la personnalisation des solutions que nous fournissons à nos clients. Il n'est pas nécessaire de fournir au client pour un prix prohibitif un environnement qu'il utilisera à 15% (ex:microsoft)

L'open Source permet cela.

- ensuite nous l'avons choisi également par conviction. En effet l'esprit communautaire est très important pour nous et nous participons activement à la promotion ,à la distribution de solutions open Source, à l'information également vous avez pu le voir sur notre site. Nous sommes en train de développer également des applications open Source dans le secteur des logiciels métiers (gestion...)
- de plus pourquoi faire payer à un client un produit que l'on peut trouver gratuitement. La valeur ajoutée est réalisée sur le service,donc sur la matière grise : Que peut-on faire de plus noble?

Q2-Q4

Nous intervenons surtout sur les phases de distribution et de promotion notamment pour des éditeurs comme Mandrake soft pour lequel nous sommes partenaires.Nous sommes également ouverts à tous les autres outils open Source, pas seulement Mandrake. Nous destinons nos offres aux pme pmi. Généralement ce sont des entreprises qui exigent un parc de qualité et qui n'ont pas forcément les moyens des grands comptes.Nous sommes une petite entreprise donc c'est une cible qui nous correspond bien.

Q3

Nous avons également un rôle de débogage Par l'intermédiaire de nos clients et par nos observations propres nous faisons remonter l'information aux éditeurs et partenaires avec lesquels nous travaillons, afin de permettre une meilleure intégration dans des parcs qui peuvent être hétérogènes et difficiles à travailler, et bien sûr de favoriser l'interopérabilité des systèmes présents dans ces parcs.

Q5

Nos missions je pense ne sont pas superposables à celles réalisées par les SSII.Celles-ci s'appuient généralement sur un modèle propriétaire, et la valeur ajoutée est réalisée ailleurs.De plus

nous accompagnons nos clients longtemps c'est notre objectif de les suivre. L'open Source met à notre disposition énormément d'outils et il n'est pas forcément nécessaire de faire des développements importants sur des applications. De grands projets existent déjà et sont ouverts, de simples modules additionnels sont juste à développer.

Q6

Sur le plan communication, les clients ne connaissent pas du tout l'open Source c'est une observation que je vous fais par expérience et nous avons d'abord un travail d'information à faire. Je pense que certaines SSLL ne font pas leur travail à ce niveau et essaient d'avoir une attitude "propriétaire" avec quelque chose qui ne l'est pas de plus elles se contentent souvent de faire simplement un site vitrine avec des tarifs et des produits sans autres informations. Ce n'est pas la voie que nous avons choisie. Je pense que vous avez vu notre portail.

Q7

En ce qui concerne la surveillance et l'évolution des solutions présentes chez nos clients la communauté des éditeurs met à disposition des correctifs rapides. La durée de vie d'une distribution est de 6 mois environ ce qui est très court. Les mises à jours disponibles sont nombreuses. Nous faisons également de la veille technologique afin de proposer à nos clients les solutions optimales dont ils ont besoin.

Q8

L'open Source est un nouveau modèle économique bien sûr. Les SSLL sont l'interface entre la communauté et le client final. Je pense que les SSLL doivent comprendre cela. Elles ont une grande responsabilité. Si l'open Source a des difficultés à s'implanter c'est avant tout la responsabilité des SSLL. Elles s'appuient souvent sur des méthodes propriétaires pour proposer des solutions qui ne le sont pas. La vision du client est ainsi brouillée et celui-ci ne voit pas la différence. Il y a des SSLL qui passent à l'open Source et qui diversifient leur activité; Elles s'en servent simplement pour un objectif économique et n'y connaissent pas les règles, la communauté, ce qui nuit au libre.

Q9

Le libre induit et va induire de nouveaux métiers. Beaucoup d'ingénieurs et de cursus ne répondent plus aux exigences du libre et sont trop généralistes. Les certifications apportent des réponses sur ces environnements.

Nous allons vers des formations orientées "produit" beaucoup plus performantes. C'est une nécessité. D'autre part il est important aujourd'hui d'étudier, de travailler ensemble et pas les uns contre les autres. Nos concurrents ne sont pas les autres SSLL, mais les systèmes propriétaires.

Il est important de déplacer la source de la valeur ajoutée d'une prestation ou d'une formation.

Nous ne vendons pas des "boîtes", mais des solutions.

En ce qui concerne les nouveaux métiers quelques pistes pour nous:

- intégrateur
- sécurité informatique
- Interopérabilité
- Veille technologique

Il convient également et cela commence à se voir de repenser le marketing autour du libre;

Q10

Enfin les principales qualités d'un bon consultant :

- Informer
- ne pas proposer un environnement, une solution, mais utiliser toutes les ressources du libre disponibles
- Très bonne connaissance et Maîtrise des outils open Source mais également propriétaires (Intégration)
- Bien identifier la problématique
- Personnalisation de la solution proposée
- Discours ouvert
- Respect des contraintes, des délais

Il y en a d'autres mais celle-ci sont les plus importantes.

PRO-FORMATIQUE

Pro-Formatique - 14 rue Anatole France - 92800 Puteaux
Tel : 01 41 02 92 80 - Fax : 01 47 74 63 75
sylvain.boily@pro-formatique.com - www.pro-formatique.com

Description:

Créée en 2004, Pro-Formatique a fait le choix de l'avenir en se spécialisant dans le domaine des logiciels libres. Sa parfaite maîtrise de l'univers des logiciels libres est un gage de sérieux et de qualité pour ses clients.

L'ambition de Pro-Formatique est de vous prouver que les logiciels libres s'intégreront parfaitement dans l'entreprise et que miser sur le logiciel libre est un pari d'avenir.

Q1

Nous avons choisi principalement par expérience et comme alternative. Par contre nous sommes plus accès Logiciel Libre.

Q2

Nos produits phares sont les serveurs de fichier, firewall, serveur mail, poste client sous linux. Ils sont destinés à toute type d'entreprise car très adaptable.

Q3

Tout ce qu'on sait faire.

Q4

N'importe quel moment, car nous proposons des audit jusqu'aux formations en passant par des simples interventions.

Q5

C'est plus ou moins semblable mis à part que nous utilisons du logiciel libre.

Q6

Les atouts sont les principaux du logiciel libre. La réaction des clients est souvent une non connaissance, mais très intéressé.

Q7

Selon la demande.

Q8

Oui.

Q9-Q10

Oui il est très porteur, autant en développement, qu'en prestation de service. Le principal atout étant la flexibilité du logiciel libre et leur puissance.

Il doit bien connaître l'environnement et être très autonome.

NetAktiv

223, rue de Charenton F-75012 PARIS
Tel : 01 40 02 92 22 - Fax : 01 40 02 01 02
info@netaktiv.com - www.netaktiv.com

Description:

NetAktiv déploie depuis 1994 des CMS basés systématiquement sur des éléments logiciels libres : sites Web, Intranet, Extranet, base de données collaborative, gestion documentaire ou comptable, syndication mutualisée...

Avec un CA de 400 keuros (45% de collectivités territoriales) NetAktiv allie depuis près de dix ans les exigences d'une structure industrielle et un engagement de Liberté Logicielle, ce qui en fait probablement la première SSL de France.

Q1

Les trois. Et par conviction qu'ils sont une alternative de meilleure qualité et perenne.

Q2

Les CMS Libres en direction des collectivités territoriales : Lutece, EdiKt, SPIP, Agora...

Q3

L'ajout de fonctionnalités et leur reversement aux communautés de développeurs en accord avec les licences.

Q4

Toutes

Q5

Non. Nous sommes signataire de la charte des SSL qui nous différencie des SSII
(cf. la charte des SSL en fin de cette annexe)

Q6

Liberté, indépendance, perennité, fiabilité, maturité des produits, choix politique.
Difficulté parfois de compréhension.

Q7

En contrat de TMA classique adapté au Libre.

Q8

Oui, assurément.

Q9-Q10

Curiosité, honneteté, respect de la charte de l'ass2l (voir l'annexe L'Association ASS2L.ORG page 87)

SYLIBRE

11 rue de Belchamps 57000 METZ FRANCE
Tél / fax : 03 87 64 49 32 (+33 387 644 932)
Tél : 03 87 57 85 60 (+33 387 578 560)
<http://www.sylibre.com>

Description:

Société de service spécialiste des logiciels libres. Audit, conseil, sécurité, conception de produits informatiques, formations.
Outils de communication : conception et hébergement de sites Internet, CD-ROM d'entreprises (annuaire, catalogue, présentations) capable de fonctionner avec Windows, Mac OS et Linux.
Réseaux informatiques (intranet), serveurs, déploiement de postes de travail.

Q1

Nous n'avons pas choisi de n'intégrer que des composants Open Source (ou mieux, des composants libres). Nous n'intégrons des composants Open Source que lorsqu'ils sont plus intéressants que leurs homologues propriétaires (et vice versa) ou lorsqu'ils sont la seule solution existante.

Un des intérêts d'utiliser les composants Libres est d'être indépendant financièrement et commercialement des éditeurs, tout en utilisant leurs produits et en participant quelquefois à la qualité de ces produits (rapports de bugs) ou à leur évolution (ajout de fonctionnalités).

Q2

Nos solutions phares sont des composants d'infrastructure (partage de fichiers, messagerie, accès internet). Notre domaine de prédilection est la sécurité (audit, solutions de sauvegarde, protection réseau, antivirus, antispam, contrôle d'accès à internet, chiffrage, formation des utilisateurs).

Notre valeur ajoutée ne réside pas seulement dans des produits "clés en main" ou des "composants phares" mais dans notre capacité à gérer des projets, à choisir les bonnes méthodologies et les bonnes technologies.

Nos domaines de prédilection sont le conseil, la formation, la sécurité informatique et le développement de systèmes en réseau.

Q3

Si nous développons une solution pour un client, nous gérons l'ensemble du projet, depuis la rédaction du cahier des charges jusqu'à la recette et la formation utilisateurs.

Q4

Nous intervenons aussi bien en amont, dans la définition du projet, que lors de phases spécifiques (développement, intégration dans un environnement informatique, mécanique etc.).

Toutes

Q5

Non car en complétant les solutions traditionnelles par des solutions Libres, nous proposons plus de solutions, certaines sont même plus fiables, plus sécurisées, plus adaptables aux besoins des clients.

Nous proposons les mêmes prestations mais avec un degré d'expertise plus élevé.

Q6

Nous communiquons sur la fiabilité et la sécurité de nos solutions et sur la qualité de nos services.

usqu'à présent, ils sont toujours très satisfaits. Lorsque nos interlocuteurs ont la fibre technicienne, ils sont surpris de découvrir la fiabilité et la flexibilité des solutions Libres.

Q7

Nous assurons les corrections et les mises à jour de sécurité suivant le niveau demandé et le type de contrat.

Les demandes de développement évolutives font l'objet de commandes spécifiques.

Nous proposons aussi des évolutions pour anticiper l'interopérabilité avec de nouveaux composants.

Q8

Le modèle économique change surtout pour les éditeurs qui se voient fortement concurrencés et doivent proposer de nouvelles offres. Pour les entreprises de service, c'est plutôt le panel de solutions qui s'élargit, la possibilité de personnaliser facilement ses solutions et de mieux en maîtriser les composants. Mais cela nécessite un plus haut niveau de qualification du personnel.

Q9

Il y a des besoins en veille technologique et en aide juridique. Nous recherchons aussi des organismes capables de nous aider à communiquer efficacement pour expliquer les avantages du Libre, surtout auprès des PME. Le Groupes d'Utilisateur de Logiciels Libres le font très bien auprès du grand public, il n'y a pas de structure qui fait ce travail localement auprès des entreprises, à part les Centres de Ressources Technologiques, mais les efforts sont très insuffisants.

Q10

De bonnes bases en informatique complétées par une bonne culture générale sur les solutions Libres.

Une bonne capacité à définir les besoins des clients, choisir et mixer les solutions Libres et propriétaires pour y répondre.

LINBOX FAS

152, rue de Grigy 57070 - Metz France
Tél.: +33 (0)3 87 50 87 90 Fax: +33 (0)3 87 75 19 26
<http://www.linbox.com>

Description:

Free et ALter soft est une SSSL pionnière. Depuis sa création en 1996, la raison d'être de F.A.S. réside dans sa capacité à intégrer à des systèmes informatiques hétérogènes, les solutions libres les plus standardisées comme les plus pointues. Les prestations proposées sont conseil, développement, portage (Irix, HP-UX, Solaris), intégration, formation, support, maintenance.

Q1

La question s'est posée en 1996, donc il y a assez longtemps. La réponse pour moi à l'origine était la suivante : la conception de produits de tout type intègre une partie logicielle de plus en plus importante. L'intégration logicielle impose l'utilisation de logiciels libres. Certes un OS offre des services, certes des liens de type com sont pratiques, mais à la fin des fins, il faut un contrôle total sur le logiciel, donc le code source. Pour cette raison, je suis convaincu que le logiciel libre s'imposera pour la conception des produits de la plupart des entreprises. L'aspect financier est important, mais pas primordial.

Q2

Nous sommes en train de travailler sur le passage d'un de nos logiciels en libre, sinon, nous utilisons des dizaines de logiciels libres, en particulier, tout ce qui touche au réseau et aux infrastructures informatiques. Et bien sûr en interne, toute la bureautique, les clients mails ...

Q3

Bogues reports, contribution de logiciels complets (siglab, linbox converter), mise à disposition d'archives de programmes libres pour différentes plateformes (solaris, irix, hpux), emploi de 2 développeurs debian, financement des flyers debian en France, ...

Q4

Tous, de la spécification jusqu'au passage en industriel.

Q5

Absolument, avec un gros plus : la maîtrise totale des logiciels que nous intégrons, ce qui nous permet de garantir la qualité pour nos clients.

Q6

Nous contribuons à l'éducation du marché de manière assez résiduelle, à vrai dire, le marché apprend plutôt pas mal. Les gens connaissent leur besoin, et les problèmes qu'ils rencontrent, si l'on répond à ce besoin, et si les esprits sont suffisamment ouverts, on a toutes les chances de réussir à rentrer.

Q7

Il y aurait beaucoup à dire. Faire une distribution linux n'est pas

difficile. Maintenir une distribution linux est beaucoup plus complexe. Après des années d'expérience, nous avons choisi Debian, pour la facilité de maintenance des composants libres.

Q8

Oui et non. Oui, car le rôle des éditeurs de logiciel et les financements associés change. Non, car sinon, c'est un modèle économique de SSII assez classique.

Q9

Pas plus qu'avant.

Q10

Il y a plusieurs profils : l'expert doit être un expert (donc avoir participé à des projets libres), le chef de projet doit avoir une bonne connaissance technique associé à une capacité à comprendre les problèmes des clients (donc une formation et une attitude plus généraliste).

14.4.2 L'Association ASS2L.ORG

extrait de <http://www.ass2l.org/rubrique2.html>

L'ASS2L est l'Association des Sociétés de Services en Logiciels Libres. L'ASS2L a pour objectif de représenter les sociétés de services en logiciels libres (SSLL ou SS2L) au niveau national et européen.

L'ASS2L défend les intérêts des SS2L, notamment auprès des pouvoirs publics et des syndicats professionnels pour que soient prises en compte les spécificités de leur profession et de leur modèle économique. L'ASS2L a ainsi pour vocation d'intégrer les groupes de travail mis en place par la Commission Européenne, d'être un interlocuteur identifié des parlementaires et des responsables politiques, sur le logiciel libre, et de siéger dans des organismes comme le Syntec ou le MEDEF.

L'ASS2L intervient notamment lorsque des processus législatifs européens ou nationaux sont susceptibles de menacer les intérêts des SS2L (directive sur les brevets logiciels, transposition de la directive EUCD, ...), ou lorsque des politiques d'aide à l'innovation ou à l'investissement sont discriminatoires pour les SS2L car ne prenant en compte leurs spécificités (cas des conditions nécessaires pour obtenir le label Jeune Entreprise Innovante et donc un statut fiscal privilégié). L'ASS2L assure également la promotion de la profession au travers de conférences, de séminaires et de débats. Elle organise de tels événements ou y participe pour donner aux SS2L une meilleure visibilité sur le marché. Elle intervient auprès des organisateurs de salons pour obtenir pour ses membres des tarifs avantageux.

In fine, l'ASS2L est une fédération professionnelle qui souhaite regrouper l'ensemble des initiatives de l'industrie du logiciel libre, représenter et défendre les intérêts des acteurs de cette industrie, et participer au développement d'un savoir-faire technologique indépendant, national et européen.

CHARTRE DE ASS2L.ORG

extrait de <http://www.ass2l.org/rubrique3.html>

Les SS2L respectent les valeurs qui sous tendent le mouvement du Logiciel Libre : mutualisation, interopérabilité, partage.

Les SS2L s'engagent en particulier à :

1. Proposer à leur clients, utiliser, développer ou intégrer exclusivement des solutions faisant appel aux logiciels libres, sauf s'il n'existe aucune alternative compétitive, et faire ainsi leur la devise : "du Libre partout où c'est possible, du propriétaire seulement si nécessaire".
2. Respecter les licences utilisées pour protéger les logiciels libres, en particulier reverser les développements à chaque fois que la licence l'impose, et chaque fois que c'est possible dès lors que le client final en est d'accord. Le lui proposer systématiquement.
3. Ne pas détenir de brevets sur des algorithmes ou sur des méthodes d'affaires ou d'éducation, et plus largement sur des méthodes intellectuelles. Ne pas chercher à en déposer à l'avenir.
4. Ne protéger ses créations qu'avec des licences reconnues par la FSF ou l'OSI.
5. Respecter et promouvoir les normes et les standards ouverts (*).
6. Informer systématiquement clients et partenaires de leurs droits en tant qu'utilisateur de logiciels libres, et de leurs obligations s'ils redistribuent des logiciels libres.
7. Respecter les usages et les modes de fonctionnement des communautés du logiciel libre

() On entend par standard ouvert tout protocole de communication, d'interconnexion ou d'échange et tout format de données interopérable et dont les spécifications techniques sont publiques et sans restriction d'accès ni de mise en oeuvre ». Loi 2004-575 du 21 juin 2004 pour la confiance dans l'économie numérique*