Stéphane Ribas (dir.),
Stéphane Ubeda, Patrick Guillaud

Logiciels & objets libres

Animer une communauté autour d'un projet ouvert

Framabook
Le pari du livre libre
Inria
INVENTEURS DU MONDE NUMÉRIQUE

Framasoft est un réseau d'éducation populaire, issu du monde éducatif, consacré principalement au logiciel libre. Il s'organise en trois axes sur un mode collaboratif : promotion, diffusion et développement de logiciels libres, enrichissement de la culture libre et offre de services libres en ligne.

Pour plus d'informations sur Framasoft, consultez
`http://www.framasoft.org`.

Se démarquant de l'édition classique, les Framabooks sont dits « livres libres » parce qu'ils sont placés sous une licence qui permet au lecteur de disposer des mêmes libertés qu'un utilisateur de logiciels libres. Les Framabooks s'inscrivent dans cette culture des biens communs qui favorise la création, le partage, la diffusion et l'appropriation collective de la connaissance.

Pour plus d'informations sur le projet Framabook, consultez
`http://framabook.org`.

Institut national de recherche dédié au numérique, sous double tutelle des ministères en charge de la recherche et de l'industrie, **Inria** a pour missions de produire une recherche d'excellence dans les champs informatiques et mathématiques des sciences du numérique, et de garantir l'impact, notamment économique et sociétal, de cette recherche. Inria couvre l'ensemble du spectre des recherches au coeur de ces domaines d'activités, et intervient sur les questions, en lien avec le numérique, posées par les autres sciences et par les acteurs socioéconomiques.

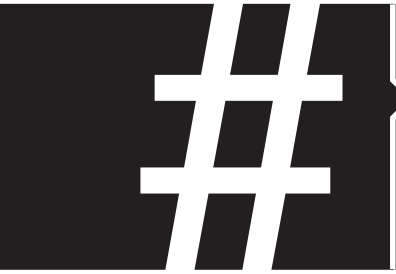
Inria emploie 2700 collaborateurs issus des meilleures universités mondiales, qui relèvent les défis des sciences informatiques et mathématiques. Son modèle ouvert et agile lui permet d'explorer des voies originales avec ses partenaires industriels et académiques.

ISBN : 979-10-92674-12-5
Prix : 12 euros
Dépôt légal : juin 2016

Icônes issues de Noun Project (Creative Commons Attribution) : Gerald Wildmoser, Gregor Cresnar, Bluetip Design, Scott Lewis, Gautam Krishnan, Rick Pollock, Fabrice Van Der Clerk, Pavel N., Aldric Rodríguez Iborra

Mise en page avec LaTeX

SOMMAIRE



À propos

The world is open by default ;
technology is just catching up.
James CORBETT (*open journalist*, manager de la communauté Corbett Report, #fOSSa2012)

À qui s'adresse ce guide ?

Vous souhaitez diffuser largement vos résultats, pérenniser une base de code, augmenter le nombre de développeurs contribuant au code, accroître le nombre de membres de la communauté, mais vous ne savez pas comment vous y prendre ? Vous commencez à manquer de temps et de ressources pour tout gérer ? Alors vous avez probablement besoin de renforcer vos compétences dans le domaine de la gestion de communauté et ce guide est fait pour vous ! Il est centré sur la création et l'animation d'une communauté autour du développement de logiciel libre et propose à ce titre une aide méthodologique tournée vers la gestion des communautés.

Ce guide a été initialement développé pour répondre aux attentes des chercheurs et ingénieurs de développement logiciel d'organismes publics

tels que Inria, le CNRS, l'INRA, etc. Cependant les réponses génériques qu'il apporte sur les méthodes de gestion de communauté sont applicables dans d'autres contextes.

Ce que n'aborde pas ce guide

Les questions liées aux modèles économiques du libre et qui ont déjà donné lieu à une publication par les auteurs de ce guide (Ribas, Guillaud, Ubéda, 2013) ne sont pas abordées ici.

De même, la description des licences *open source*, qui est l'objet d'une abondante littérature, n'est pas notre sujet.

Exemples par la pratique

À la fin de ce guide sont présentées deux études de cas réels. Les actions menées et leur impact y sont décrits afin d'illustrer la mise en œuvre des méthodes proposées. Nous avons aussi inséré deux interviews de *community managers* afin de vous faire partager leurs expériences en la matière et donner leurs points de vue sur le sujet.

Introduction

Community over code !
Ross GARDLER, Vice-Président de Apache Software Foundation, #fOSSa2009

Pour les chercheurs travaillant dans les sciences du numérique, le logiciel joue un double rôle en étant à la fois un objet d'étude (langages, compilateurs, systèmes d'exploitation, *frameworks*, etc.) mais également un moyen de contribuer à une production scientifique (pour renforcer la véracité, la validité et la robustesse des résultats scientifiques). Le monde de la recherche a une inclination naturelle à utiliser le logiciel libre pour le développement et la diffusion des connaissances et des technologies, l'accroissement de la visibilité des chercheurs et la constitution de masses critiques de développement. Mais pour en tirer profit, il est nécessaire d'organiser la communauté autour du code et de trouver les motivations qui réuniront les personnes autour de valeurs partagées.

Ce guide traite de la création et de l'animation d'une communauté autour du développement de logiciel libre. Le présent document n'a pas d'autre prétention que d'être une aide méthodologique abordant la question des communautés, en résumant les concepts, prodiguant des conseils et présentant un ensemble de bonnes pratiques. Il décrit des comportements et des processus permettant d'appréhender les concepts généraux

autour des communautés dans le domaine du développement de logiciel libre. Il regroupe des éléments bien connus mais les organise de manière à accompagner les équipes de recherche autour de cette question, et se veut avant tout pratique. À la fin de ce guide sont proposées des références permettant d'aller plus loin ainsi qu'une bibliographie.

Mais avant d'aller plus loin, il est important de rappeler quelques définitions autour des communautés (il est primordial de comprendre comment une communauté émerge et quelle est sa dynamique). Le prochain paragraphe propose donc une définition du terme *communauté*, puis nous abordons les grandes lignes de la méthode exposée dans ce livre.

Qu'est ce qu'une communauté ?

Une communauté est un groupe de personnes partageant des préoccupations, une passion ou un intérêt communs, qui s'organise et engage des actions concertées afin de résoudre un problème partagé.

On distingue plusieurs types de communautés : les communautés d'apprentissage, les communautés d'intérêt, les communautés de « passion », les communautés de pratique.

Une communauté d'apprentissage est un groupe de personnes qui partagent des valeurs communes et qui sont activement engagées dans un processus d'apprentissage mutuel (par exemple, le projet Dessine moi un robot [1], ou encore la communauté Pixees [2]). De telles communautés sont devenues un modèle initial pour les autres types de communautés. Les membres d'une telle structure ont un sens fort de loyauté envers le groupe qui alimente leur désir de continuer à travailler et à aider les autres, de produire un effet sur la communauté en étant non seulement réactif mais actifs. Une communauté d'apprentissage laisse une liberté suffisante pour que les membres puissent exprimer des opinions personnelles, demander de l'aide ou des informations spécifiques et commenter des événements.

Une communauté d'intérêt peut être vue comme un groupe de personnes partageant des thèmes qui ne nécessitent pas de structure formelle mais qui mènent des discussions tournées vers la collaboration et le partage des

1. `https://dm1r.inria.fr/`.
2. `https://pixees.fr/`.

connaissances (comme les ingénieurs systèmes et réseaux participant aux conférences JRES [1], les ingénieurs de recherche et développement logiciels participant à la liste de diffusion DEVLOG [2]). Elles peuvent consister en des groupes de personnes faiblement connectées sans grande implication en terme de production commune. Elles restent cependant bien au fait de leurs thèmes et posent des questions.

Une communauté de « passion » est basée sur un groupe de personnes dont les activités, la gouvernance et la structure sont les plus riches et les plus formelles. Les membres y jouent un rôle particulier (ex. conseil en sécurité de réseau), ils aident activement les autres membres à coller à leur rôle et visent à maîtriser la discipline. Les communautés de passion sont davantage tournées vers le partage de connaissances dans un domaine donné que sur la réalisation d'un projet particulier. Leurs membres peuvent intervenir dans des communautés de pratique en tant qu'experts ou consultants.

La structure d'une communauté de pratique est moins formelle et basée sur des spécialités communes (ex. les groupes d'utilisateurs de GNU/Linux [3]). Les membres ont un rôle ou une spécialité spécifique (ex. la sécurité, le développement J2EE) et se focalisent sur le développement de l'expertise et des compétences qu'ils ont dans ce rôle ou cette spécialité. Un facteur essentiel de motivation est l'apprentissage de la spécialité et la résolution de problèmes.

La communauté de pratique, concept hérité des sciences humaines et sociales et antérieur au développement de l'informatique, présente trois caractéristiques spécifiques :

1. elle permet de résoudre collectivement et de manière efficace un problème partagé et, durant ce processus
2. elle permet à une proportion importante de ses membres de suivre un processus d'apprentissage
3. les activités et interactions qui se déroulent en son sein ont pour effet de produire des connaissances nouvelles. Elles sont donc productrices d'innovation.

1. `https://www.jres.org`.
2. `http://devlog.cnrs.fr/`.
3. `http://www.linux.org/groups`.

La méthodologie présentée dans ce livre a été ébauchée à partir de ce modèle et appliquée à ce type de communauté, mais pas seulement…

Les membres de ces communautés ont des interactions quasi quotidiennes avec des personnes extérieures au projet – ces dernières forment parfois entre elles des réseaux d'entraides, produisent des communautés… communautés d'apprentissage ou d'intérêt. N'oublions pas ces communautés : sans ces « utilisateurs » nous ne pourrions pas disséminer nos résultats de recherche, ni nos productions de logiciels. Ces communautés sont vitales pour la pérennité des projets de développement logiciel. Des conseils afin d'attirer leur attention, les encourager à utiliser votre logiciel, à rejoindre votre communauté et à contribuer, sont exposés tout au long de ce guide.

La communauté des développeurs dans le monde du logiciel libre

La collaboration a toujours été très présente chez les développeurs, particulièrement depuis l'arrivée d'Internet. Les communautés de développeurs ont montré leur efficacité dans de multiples domaines, et cela est tout particulièrement vrai pour le développement de logiciels libres qui se prêtent, par construction, aux effets précédemment mentionnés : résolution de problèmes, montée en compétence des développeurs et production d'innovation.

Pourquoi créer une communauté ?

L'existence d'une communauté active autour d'un code, qu'il s'agisse de développeurs ou d'utilisateurs, permet de disposer de ressources pour le faire évoluer, pour améliorer sa qualité et garantir une bonne réactivité face aux évolutions, qu'elles soient scientifiques, techniques ou d'usages. Une autre caractéristique essentielle des communautés étant de produire des connaissances nouvelles et de stimuler l'innovation, le partage d'informations et de connaissances, ainsi que les interactions au sein du groupe permettent de faire émerger des idées, de lancer de nouvelles pratiques, de faire naître des usages en rupture. Plus la communauté s'étend

et se diversifie, plus elle devient un gage de qualité et de dynamisme. Une communauté est également un puissant vecteur de diffusion permettant à un logiciel d'acquérir visibilité et réputation, ce qui peut constituer un gros atout en cas de collaboration avec le monde économique.

Par où commencer ?

Le point de départ consiste à qualifier le besoin initial afin d'évaluer la pertinence de votre projet de communauté : s'agit-il d'un besoin susceptible d'être partagé par un nombre suffisant de personnes, qu'il s'agisse de développeurs et/ou d'utilisateurs ?

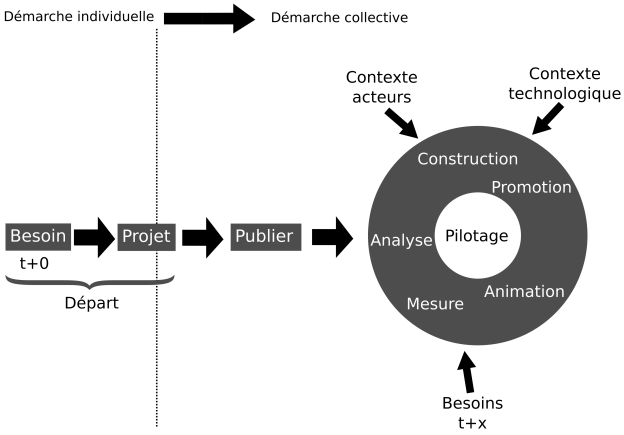
Si le besoin est identifié et semble suffisamment fédérateur, alors la première étape consiste à définir le projet puis à publier le code du logiciel (si ce n'est pas déjà fait) afin d'améliorer son développement, en faire la promotion afin d'amorcer la communauté, puis la faire croître et l'entretenir. Se lancer dans la création d'une communauté demande du temps et des ressources, aussi la décision de tenter cette aventure doit-elle être prise avec discernement et de manière non systématique car il faut s'assurer, avant de se lancer, que les contraintes engendrées n'excéderont pas les avantages espérés.

Une communauté naît rarement de façon spontanée et il faut être prêt à consacrer du temps pour la faire émerger, la faire croître puis maintenir la motivation de ses membres. La base de toute communauté est le partage, ce qui implique en contrepartie une certaine dilution du contrôle qu'il faut évaluer en fonction de ses objectifs (diffusion d'une méthode, standardisation, transfert technologique, promotion d'une école de pensée, etc.).

Il faut ensuite définir clairement ce que l'on souhaite partager et la forme que doit prendre ce partage. Cela permet de déterminer une architecture de code qui facilitera le type de partage envisagé (dans son entier, par module, ou autres) et par un choix pertinent de licence. On se focalise très souvent sur le choix de cette dernière, mais l'organisation du code et, plus tard, le mode de gouvernance autour de ce code sont tout aussi importants.

L'organisation et les outils d'animation constituent l'architecture de participation de la communauté. L'ensemble des valeurs et des objectifs partagés par cette communauté sont les ressorts qui l'animeront. Les valeurs

peuvent être scientifiques, techniques, économiques, éthiques ou sociétales. Dans la pratique on retrouve souvent une combinaison de ces valeurs qu'il faut hiérarchiser autour d'une valeur principale qui définira la communauté. Négliger cette étape de réflexion sur les objectifs et les valeurs est l'une des premières causes d'échec. En effet, la première chose que feront les participants potentiels à cette communauté sera d'évaluer l'adéquation des valeurs qu'elle affiche par rapport aux motivations qui les animent.



Les différentes étapes

Dynamique de la méthode

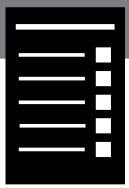
Ce guide propose un mode d'emploi de la gestion des communautés à base de bonnes pratiques. La méthode proposée, itérative, comprend les étapes suivantes : définition, analyse du contexte, structuration du projet, mise en œuvre de l'architecture de participation, publication du code, promotion du logiciel, animation de la communauté, suivi et évolution.

```
1  Program 'Construire une communauté.c'
2  /* LICENCE is CC-BY-SA */
3     #include "guide gestion de communauté.h"
```

```
4     #define rêve = undefined
5     #define projet défini = undefined
6     #define inventaire = undefined
7     #define objectifs valeurs = undefined
8     #define gouvernance = undefined
9     #define habitat numérique = undefined
10    #define répondre à un besoin partagé = true
11
12  void setup() /* Tâches à faire séquentiellement */
13  {
14          rêver(); /* optionnel */
15          définir projet();
16          faire inventaire();
17          définir objectifs et valeurs();
18          faire veille autour du projet();
19          définir habitat numérique();
20          mise en oeuvre habitat numérique();
21  }
22
23  void loop() /* Tâches pouvant être effectuées en
        parallèle */
24  {
25          coder();
26          faire promotion projet();
27          animer communauté();
28          augmenter nombre de membres();
29          faire veille autour du projet();
30          suivi et évolution du projet();
31  }
```

Le chapitre 1 concerne la première étape, la définition du projet. En effet, il est primordial de répondre aux questions suivantes : quel est le concept ? à quel(s) besoin(s) répond-il ? quels sont les objectifs ? quelles sont les ressources disponibles ? en somme, de quoi je pars et vers où je veux aller.

1 Définition du projet



Be open, be yourself, listen & engage
Tristan NITOT (Mozilla Europe Founder, #fOSSa2013)

1.1 Le besoin

Ce qui caractérise un projet de développement de logiciel libre par rapport aux autres projets de développement, c'est qu'il doit avant tout répondre à un besoin partagé par un groupe suffisamment large pour pouvoir créer un effet de levier en répartissant l'écriture du code entre de multiples développeurs. La première étape consiste à bien identifier précisément le besoin auquel on souhaite répondre pour aboutir à une définition claire du logiciel que l'on envisage de développer et/ou de promouvoir. Il est crucial de s'assurer que les points suivants sont respectés :

— le besoin identifié est partagé par un public suffisamment large ;
— un nombre suffisant de personnes sont susceptibles de contribuer à la satisfaction de ce besoin ;
— ces personnes trouveront un intérêt à participer, contribuer et utiliser le logiciel.

Il est impératif que ces trois points soient vérifiés, ce sont les facteurs clés de succès d'un projet de développement communautaire.

Une fois que le besoin est précisé, il faut définir la finalité du projet : s'agit-il d'un objet de recherche ? d'un démonstrateur d'une théorie scientifique ? d'un démonstrateur d'une technologie ? d'un intégrateur de plusieurs technologies ? d'une preuve de concept autour d'un service ? d'un producteur de données ? d'un système centralisé ou distribué ? d'un outil de communication ? Cette liste n'est pas exhaustive.

Il faut préciser ensuite la nature du logiciel : s'agit-il d'une bibliothèque ? d'un logiciel autonome ? d'un *framework* ? d'un environnement de développement ? d'un logiciel embarqué ? de l'implémentation ou de l'amélioration d'un algorithme ? d'une API ? d'un assemblage de composants ? d'un ensemble d'outils de gestion d'une plateforme logicielle ou d'une infrastructure ? d'un démonstrateur ? d'une extension ? du *fork* d'un autre logiciel ? d'un logiciel nouveau créé à partir de zéro ? d'un portage vers un autre système ou vers un autre langage ? d'une réécriture ?

Enfin, il faut définir à l'aide d'une poignée de mots clés le domaine correspondant au projet et son environnement, par exemple. Identifier aussi les thématiques (et au besoin les sous-thématiques) auxquelles correspond le logiciel envisagé : système, traitement d'image, protocole de communication, sécurité, sémantique, *big data*, etc.

Ces étapes permettent d'y voir plus clair et facilitent la réflexion de ceux avec qui vous envisagez de vous lancer dans l'aventure ou qui hésitent à vous rejoindre.

C'est alors le moment de faire l'inventaire des ressources disponibles.

1.2 L'inventaire

Cet inventaire débute par une revue de code qui consiste à examiner le code préexistant (que ce soit une ébauche ou un prototype déjà avancé) et les méthodes employées pour le produire et le maintenir.

1.2.1 Architecture du logiciel

Il convient de décrire l'architecture générale du logiciel : est-il monolithique ? modulaire ? un mélange des deux ? le code utilise-t-il des patterns connus ? quels sont les langages utilisés ? l'architecture est-elle documentée, sous quelle forme ? où le code est-il stocké ? en existe-t-il plusieurs versions ? nécessite-t-il des environnements techniques spécifiques ?

1.2.2 Aspects juridiques

Dans le même temps, une étude de nature juridique doit être menée sur le code en considérant les aspects liés à la propriété intellectuelle. Bien que les aspects techniques et juridiques puissent sembler indépendants, ils résultent tous deux d'une analyse du code. Il est nécessaire de vérifier les deux aspects que l'on vient d'évoquer, fichier par fichier. On doit le cas échéant compléter cela par la caractérisation des données entrantes et des données produites. On caractérisera les formats de ces jeux de données, leur provenance, les questions juridiques liées à leur utilisation, etc. Au passage, on vérifiera si des licences sont associées à ces données (type *Creative Commons* ou autres).

La dimension sociale doit aussi recevoir la même attention : à qui appartient le code ? qui sont les créateurs du code ? quels sont leurs statuts, leurs employeurs, leurs activités ? qui sont les autres contributeurs ? sont-ils encore actifs ? Etc.

1.2.3 Méthodes de production du logiciel

La revue technique doit également porter sur les méthodes qui ont servi à produire le logiciel, les bonnes pratiques qui ont été employées : utilisation d'une forge logicielle, d'un outil de gestion de versions, d'outils de compilation, d'outils de tests automatiques. Tout ce qui concerne la gestion d'un projet logiciel, jusqu'à l'existence d'une feuille de route technique pour ses évolutions futures, doit être inventorié.

1.2.4 Documentation

L'analyse de la documentation, aussi bien interne au code (entêtes, commentaires) qu'externe (guide du développeur, guide de l'utilisateur, guide d'installation, exemples commentés, etc.) doit être réalisée minutieusement et la qualité de ces documents doit être évaluée (sont-ils à jour ? etc.). Le couplage entre un code et sa documentation est un élément primordial lorsqu'on démarre une communauté.

1.2.5 L'équipe principale et leader

Il s'agit de cerner clairement l'équipe principale (la *core team*), le noyau dur autour duquel le code va vivre. Pour chaque membre, il faut se poser la question de son profil, de ses domaines de compétences, du temps qu'il pourra consacrer à cette aventure et surtout de sa motivation. On doit aussi répondre à la question de savoir qui assumera le leadership de cette équipe ; on parle bien du *leadership* du petit groupe chargé de faire naître et croître la communauté. Dans les étapes de démarrage cette fonction peut se confondre avec le *leadership* de la communauté tout entière mais elle sera remplacée à terme par une architecture de participation spécifique.

1.2.6 Aspects communication

La dernière étape de l'inventaire des ressources concerne les moyens de communications, les partages déjà existants. Quel est le nom de votre logiciel ? est-il protégé, libre ? avez-vous un site Web ? les noms de domaines sont-ils réservés ? des listes de diffusions ? d'autres médias de communication (tel que les réseaux sociaux) ? existe-t-il des pratiques de dissémination ? le logiciel a-t-il un logo ? si oui, en avez-vous les droits ? avez-vous déjà des outils de collaboration ? maîtrisez-vous la communication grand public ? avez-vous réalisé et mis en ligne des vidéos ? présenté le projet et son code dans des événements ? déjà organisé des tutoriels ?

Toutes ces ressources doivent être inventoriées, regroupées, évaluées en termes de qualité et d'impact. Ce travail peut être long et complexe sur un code qui a plusieurs années d'existence ; mais réalisé sur un code jeune,

ce qui est souvent le cas, tout ceci peut être rapide et permettre de gagner beaucoup de temps par la suite.

Après avoir cerné le besoin auquel répond le logiciel initial et fait l'inventaire de nos ressources, il faut alors définir les objectifs de la communauté que l'on se propose de créer et qui définira la dynamique et les orientations du projet à partir de cet existant.

1.3 Les objectifs

La construction du projet est une tentative d'allier le rêve et la réalité : quelle serait la communauté idéale autour de ce logiciel ? Comment fonctionnerait cette communauté idéale ? Comment la réaliser avec les ressources dont on dispose ? Laissez libre cours à votre imagination dans un premier temps, mais le principe de réalité vous amènera probablement à revoir vos ambitions.

Souhaite-t-on structurer des développements autour d'une méthode ou d'une thématique scientifique large ? promouvoir une approche en rupture avec les approches classiques ? faciliter la diffusion et l'acceptation d'une approche ? illustrer en priorité les travaux d'un groupe ? rendre plus visible une communauté scientifique, technique ou économique existante ? promouvoir une école de pensée ? avoir un impact sur la société ? offrir une alternative à un marché fermé ? préparer un transfert, la création d'une entreprise ou d'une *startup* qui s'appuiera sur la communauté ? Les motivations peuvent être multiples et il est important d'identifier les objectifs que l'on souhaite atteindre et de les hiérarchiser s'il y en a plusieurs.

Il faut ensuite se poser la question de ce que l'on attend de la communauté : quels objectifs serviront au mieux sa finalité : partager des savoirs ? tisser des relations avec d'autres communautés ? faciliter le développement et la mise au point via des développeurs externes ? créer de la valeur socio-économique ? Rappelons que faire naître une communauté, l'entretenir et la faire croître supposent d'abandonner une partie du contrôle en échange de la participation de membres extérieurs à l'équipe principale initiale.

Ce qu'on peut espérer d'une communauté peut être classé en quatre grandes catégories :

1. extension de l'usage d'un logiciel (fonctionnalités et/ou usages) – renforcement de la diffusion / de la réputation / de la visibilité ;
2. amélioration de la qualité d'un code – amélioration / agrégation / pérennisation / mutualisation d'un effort ;
3. stabilisation d'une communauté scientifique et/ou technique, amorçage, maintien ou renforcement de collaborations ;
4. amorçage d'un modèle économique.

La définition des objectifs permet d'affiner la définition du ou des groupe(s) à qui cette communauté s'adressera (la communauté cible peut être légèrement différente de la communauté initiale), quelle masse critique d'utilisateurs/contributeurs elle doit et peut atteindre. S'il existe des communautés connexes sur lesquelles s'appuyer, il faut les identifier et collaborer avec elles pour ne pas refaire ce qui existe déjà. Cela pourra se faire par exemple en regardant comment la communauté scientifique est structurée autour des thèmes centraux, en identifiant des communautés de pratiques dans ces domaines, des consortiums industriels, etc.

1.4 Les valeurs

Enfin, ces objectifs doivent permettre de décider assez facilement quel sera le moteur du partage : les valeurs. Une communauté se base sur un partage organisé autour de l'intérêt (je contribue parce que j'ai un problème à résoudre et qu'il faut s'unir pour faire ensemble ce qu'on ne peut pas faire seul) et des valeurs (je contribue parce que je me retrouve dans les valeurs autour desquelles est bâtie la communauté). Croire que l'intérêt est primordial, que le système de valeur n'est qu'anecdotique, est une erreur. Une partie importante de l'activité au sein d'une communauté *open source* est basée sur le volontariat et chaque contributeur doit se reconnaître dans la communauté à laquelle il participe. Ces valeurs peuvent être basées sur la qualité scientifique, l'ouverture d'esprit, les problèmes de société, l'éthique, la création de valeur économique, la promotion d'un courant de pensée, etc. Le système de valeur a un impact sur de multiples aspects de la communauté : gouvernance, communication, outils de collaboration, etc.

1.5 Le partage

Arrivé à ce point il devient possible de décider, en fonction des objectifs et des valeurs, ce que l'on souhaite partager et comment. On l'a déjà énoncé, mais on ne le dira jamais assez : lancer une communauté nécessite d'abandonner une partie du contrôle sur l'objet produit en contrepartie du soutien de la communauté. Cependant ce principe ne s'applique pas de manière uniforme au projet, les codes sont rarement monolithiques et, en fonction de l'architecture du code, ce compromis entre partage et contrôle peut accepter une certaine hétérogénéité. Des modes de partage différents peuvent être appliqués aux différents composants du logiciel. Bien qu'il existe toutes sortes de façons de procéder, l'usage d'une licence libre permet d'ajuster le compromis entre partage et contrôle. Il existe aujourd'hui un grand choix de licences permettant de régler le mode de collaboration en fonction du compromis entre le contrôle que l'on veut conserver sur son logiciel et l'ampleur de sa diffusion.

1.6 Licences

Le choix d'une licence, ou d'une architecture de licences – un logiciel peut en combiner plusieurs – doit être précautionneusement déterminé en fonction des objectifs/finalités et des valeurs que l'on veut mettre en avant. Certains choix facilitent l'adhésion d'une large communauté, mais compliquent les possibilités de transferts vers l'industrie. Une bonne méthode pour choisir une licence consiste à retenir celle qui reflète le mieux les valeurs fondatrices de cette communauté et la finalité de son projet.

Lorsqu'un logiciel a déjà une licence au moment de structurer une communauté, il faut se poser la question de l'adéquation de cette licence au projet et, le cas échéant, ne pas hésiter à en changer. Un tel changement est rarement simple mais il est d'autant plus compliqué à réaliser que le temps passe et que les contributions s'accumulent. Du coup la meilleure chose à faire est de prendre le taureau par les cornes et de régler la question au plus tôt, quitte à réécrire certaines portions du code.

1.7 Méthode pour choisir une licence

1.7.1 Questions générales

Veillez à répondre aux questions suivantes :

1. quelle est la stratégie de votre employeur pour la propriété intellectuelle en général ?
2. quelle est la stratégie de vos partenaires pour la propriété intellectuelle en général ?
3. quelle licence est utilisée pour les composants et artefacts développés par l'équipe principale ?

1.7.2 Relation avec le monde extérieur ?

Identifiez les codes et bibliothèques externes à votre propre code projet :

1. quelle licence est utilisée pour les composants et artefacts externes au projet ?
2. quels sont les termes d'utilisation pour les composants et artefacts commerciaux (sous licences fermées) ?

1.7.3 Les autres ressources ?

N'oubliez pas de regarder les autres types de ressources tels que la documentation, les images, les vidéos, etc., et identifiez vos besoins en termes de partage :

— quel niveau de diffusion ?
— qui peut distribuer ces documents ?
— autorisez-vous la réutilisation et la modification ?

1.7.3.1 Le futur ?

Projetez-vous dans l'avenir selon ces préconisations :

1. identifiez les possibles directions, ou chemins que pourrait prendre votre projet sur le long terme. Pensez aux éventuelles restrictions dans le cas d'une future commercialisation de votre projet à travers des services, et autres modèles de capture de valeur. Pensez à de possibles extensions de votre projet sous formes diverses et variées (SAAS, vente d'objets, etc.) ;
2. quels seront les futurs développements du code (sur le long terme) ?
   — identifiez les directions possibles du code sur le long terme (e.g. *closed source/open source* ?),
   — identifiez les possibles agrégations de votre code avec d'autres projets.

1.7.3.2 Votre choix ?

Grâce à l'ensemble des informations réunies, vous devez à présent pouvoir identifier :

1. la (ou les) licence(s) d'entrée du logiciel ;
2. la (ou les) licence(s) de sortie des ressources associées au code.

N'hésitez pas à demander conseil et assistance auprès de personnes compétentes dans le domaine.

1.8 Propriété intellectuelle

Cependant, le choix de la licence (ou de l'architecture de licences) ne doit pas occulter un aspect encore plus important : la gestion de la propriété intellectuelle. On a déjà signalé que ce point doit être abordé préalablement au lancement de la communauté, mais il est également nécessaire de décider de la façon dont cette propriété intellectuelle va évoluer avec la communauté. Elle dépend de la finalité et du degré d'ouverture que l'on veut donner au code et, même dans le cas où l'équipe principale conserve la propriété et le contrôle du code, il est impératif de se poser la question

du statut du code produit par les contributeurs externes car cette question se posera un jour ou l'autre : à qui appartient le code produit par un stagiaire de passage, par un contributeur occasionnel ou par un contributeur régulier situé à l'autre bout du monde ? Ces éléments doivent alimenter la réflexion menée lors de la construction de l'architecture de participation (cf. chapitre 3).

La connaissance de la propriété initiale des différentes parties d'un logiciel influence le choix de la licence et de la gestion de contributions futures. Il en va de même pour les autres ressources du projet : possédez-vous le nom du projet ? est-il nécessaire de protéger le nom ? avez-vous déjà un nom de domaine en votre possession ? qui achète – et donc contrôle – le nom de domaine ? etc.

Vu le coût dérisoire d'un dépôt de nom de domaine il est vivement conseillé d'en réserver un le plus tôt possible. L'expérience montre qu'une négligence à ce propos peut coûter cher et devenir rapidement un casse-tête. Certains projets de développement de logiciel libre optent pour une protection de la « marque » (le nom du projet constitue bien souvent une marque) auprès d'organismes spécialisés dans la protection de la PI (en France il faut s'adresser à l'INPI, Institut National de la Propriété Industrielle). Déposer une marque ne signifie pas « protéger le code » qui lui correspond (le code est protégé en lui associant une licence et non en déposant une marque) mais permet un certain contrôle sur l'utilisation éventuelle de la marque par des tiers.

Quelle que soit la politique de gestion de la propriété intellectuelle envisagée, très ouverte ou très protectionniste, toutes ces réflexions doivent être menées au plus tôt.

1.9 Le nom du projet et la mission

À ce stade, vous avez défini le concept du projet, ses objectifs, les communautés qui sont ciblées, la valeur ajoutée apportée, etc. Il est temps de réfléchir à un bon nom de projet.

Si le nom de votre projet n'est pas très parlant, n'hésitez pas à en changer tant qu'il est encore temps. Vous pouvez aussi changer le nom plus tard, cela permettra, peut être, de donner une deuxième vie à votre

« projet-communauté » ou de communiquer sur le changement de nom. Cependant, changer de nom ultérieurement est à éviter dans la mesure du possible.

Vérifier que le nom du projet est disponible sur la toile (nom de domaine libre). Le nom du projet peut être un nom d'animal, un mot d'une langue étrangère, un nom qui a du sens (par son étymologie) ou qui porte des valeurs, ainsi :

— *Diaspora* (signifie « dispersion », vient du grec *sporo* qui veut dire « graine » ou de *speira* signifiant « semer ») ;
— *Redmine* (devinez !) ;
— *GIT* (personne illégitime pour répondre aux autres forges) ;
— *fOSSa* (nom d'un animal très agile vivant la nuit et dont l'histoire ressemble beaucoup à l'histoire des développeurs de logiciel libre, sigle signifiant aussi *Free Open Source Software for Academia*) ;
— *Ubuntu* (veut dire « générosité » dans plusieurs dialectes africains) ;
— *Poppy* (Poppy project) signifie coquelicot en français ;
— *Natron* (*an open-source – crossplatform nodal – compositing software*). Le natron est un minéral, symbole du Sodium (*Na*). Il s'agit du nom d'un effet très connu dans le domaine du *compositing* cinématographique ;
— pensez aussi à la manière de prononcer le nom du projet ;
— demandez à des proches ce qu'ils pensent du nom de projet.

À présent, et en vous appuyant sur les différentes informations recueillies précédemment, décrivez la mission (*mission statement*) ; voici deux exemples de textes :

1. *Projet AspireRFID*
   — The AspireRFID project aims at developing & promoting an free, open source, lightweight, standards-compliant, scalable, privacy-friendly, & integrated middleware along with several tools to ease the development, the deployment & the management of RFID-based applications & sensor-based applications. It implements several specifications from consortiums such as EPC Global, NFC Forum, JCP and OSGi Alliance.
2. *Poppy, the 100% open humanoid robot*

— Poppy is an open-source platform for the creation, use & sharing of interactive 3D printed robots. It gathers an interdisciplinary community of beginners and experts, scientists, educators, developers & artists, that all share a vision : robots are powerful tools to learn & be creative. The Poppy community develops robotic creations that are easy to build, customise, deploy, & share. We promote open-source by sharing hardware, software, & web tools.

1.10 Pour gagner du temps

L'ajustement entre le contrôle et la diffusion d'un code peut être perçu comme l'histoire du beurre et de l'argent du beurre : on ne peut pas avoir les deux, il faut choisir. L'idée qu'une licence magique ou que des agencements de licences compliqués permettraient de se soustraire à ce principe n'amène généralement que déceptions et pertes de temps.

Le projet de développement logiciel et la communauté ne sont pas des entités isolées dans leurs bulles ; ils sont immergés dans un environnement en mouvement avec lequel ils auront des interactions. Il faut donc connaître ce contexte et prendre acte des opportunités et des risques qu'il comporte. Le prochain chapitre traite du contexte lié au projet, l'analyse de son environnement : ce qui nous entoure.

2 Analyse du contexte

Heterogeneity and diversity are the fuels of evolution/innovation
Mark SOMERFIELD (Creative Innovation Founder, #fOSSa2013)

Une fois l'état des lieux du projet réalisé, c'est le moment de s'intéresser à l'environnement dans lequel il va s'intégrer et aux acteurs avec lesquels il va interagir : c'est la caractérisation de l'écosystème. Au cours de l'état des lieux, vous avez détecté s'il existait ou non un embryon de communauté pour votre logiciel, ne serait-ce que quelques personnes. Il s'agit de savoir où trouver des forces vives pour étendre ce noyau initial.

2.1 Les coopétiteurs

Vous devez identifier vos concurrents, les logiciels positionnés sur un segment identique ou proche. Vous devez caractériser ces projets. Dans quelle mesure sont-ils concurrents ? Quelles sont leurs similitudes et leurs différences ? Peut-on coopérer avec ces projets ? Il faut préciser en quoi votre programme se différencie : nature et étendue des fonctions, mode de

diffusion, modèle économique, performances, interface utilisateur, plateforme cible, etc.

Même si aucun concurrent direct au logiciel que vous proposez n'a été identifié, celui-ci s'insèrera dans un environnement, souvent en complément d'outils techniques ou méthodologiques : dressez-en aussi la liste. Existe-t-il des communautés autour du langage de programmation ? de la méthode de développement ? des méthodes scientifiques ? du site d'hébergement de votre code ?

Vous prendrez tout particulièrement en compte les éléments issus de l'analyse déjà réalisée de votre code (cf. chapitre 1), notamment en ce qui concerne les dépendances à d'autres environnements techniques, codes ou bibliothèques. Il est important de connaître la dynamique des communautés associées aux éléments externes dont votre code dépend. Cette analyse est susceptible de remettre en cause des choix techniques.

Ces projets et ces communautés déjà existantes sont qualifiés de *coopétiteurs* car, sauf cas particulier d'un concurrent direct – mais cela pose la question de la pertinence de l'existence de deux projets – on se trouve, suivant les activités, en situation de coopération ou de concurrence. Il faut comprendre les objectifs, le système de valeurs, la dynamique et les autres facteurs qui rapprochent les deux projets et ceux qui les différencient, pour comprendre dans lequel des deux types de relations s'inscrit une action envisagée. Il sera également nécessaire de comprendre les relations qui les lient et éventuellement l'existence de méta-structures les regroupant au sein d'ensembles plus larges (écosystèmes).

Identifiez aussi les domaines potentiels d'application de votre logiciel. Cela vous permettra de repérer des communautés actives dans d'autres domaines, qu'il vous faudra aussi analyser et qualifier. Les connexions possibles sont nombreuses : techniques, scientifiques, de valeurs, de pratiques, etc. Si une recherche exhaustive n'est pas impérative, il est indispensable d'identifier les communautés incontournables dans le domaine visé. Le besoin auquel répond le projet doit être le principal vecteur de recherche et peut amener à s'intéresser à des communautés qui ne sont que faiblement structurées ou dormantes.

Pour réduire le temps de recherche, on utilisera le couple (valeur, objectif) de notre projet (cf. chapitre 1) pour savoir où chercher

dans l'environnement (communautés scientifiques, communautés techniques, partenaires, etc.) et au niveau du code (chercher les communautés autour du langage de programmation, de l'environnement de développement, de logiciels similaires, de bibliothèques utilisées). Vous pouvez présenter à votre équipe la liste des outils de collaboration et de communication utilisés par ces communautés et discuter de leur adéquation à vos besoins.

Il peut être nécessaire de se rapprocher de certains acteurs des communautés avec lesquelles des collaborations sont envisagées et de les sonder pour voir dans quelle mesure et au prix de quels ajustements elles peuvent devenir des bassins de recrutement. Il faudra comparer le système de valeurs et les objectifs de ces groupes par rapport aux vôtres.

2.2 La décision

Cette première étape d'observation de l'écosystème, dont votre logiciel fait déjà partie ou dans lequel il s'insèrera, complète l'état des lieux que vous avez mené et doit permettre de prendre la décision principale : le *go / nogo*. En cas de *go*, il faut définir une stratégie par rapport à cet écosystème : doit-on chercher à créer une nouvelle communauté ? Si oui, comment s'appuie-t-on sur les communautés existantes pour recruter ? Si l'on veut s'insérer dans une communauté existante, comment peut-on exister en son sein ? Le choix entre ces deux options doit être mûrement réfléchi, car il implique des démarches différentes et aura des implications fortes sur l'ensemble du projet.

Renoncer à créer une communauté n'implique pas d'arrêter le développement logiciel ; il s'agit de bien cerner l'ambition qu'on donne à son projet par rapport au contexte et aux moyens qu'on est prêt à y consacrer. Si l'on renonce à dynamiser une communauté, on se centrera sur une promotion simple du code en l'orientant résolument comme un simple démonstrateur technologique et scientifique. La réflexion sur le *go/nogo* peut toujours reprendre quelques années plus tard en fonction de l'évolution du contexte et bien sûr de votre logiciel.

L'étape suivante de la prise de décision est de savoir si l'on rejoint une communauté existante ou si on en crée une *ex-nihilo*. Il faut pour cela répondre à deux questions :

1. le modèle de cette communauté est-il durablement compatible ou non avec les valeurs portées par le projet ? leurs outils sont-ils susceptibles de convenir ? la gouvernance de la communauté préexistante nous laissera-t-elle agir sur elle pour l'influencer ou l'orienter ou bien est-elle fermée ?
2. quelles actions sont envisageables avec cette communauté ? s'en servir pour faire de la promotion et développer la base d'utilisateurs ? est-il possible d'y recruter de nouveaux contributeurs ? se prête-t-elle à une intégration complète de votre projet ? ou bien à des collaborations avec votre future communauté ?

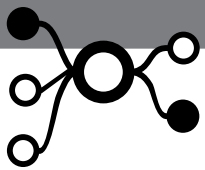
Rejoindre un projet existant ou tout du moins une communauté existante vous donnera moins de visibilité et limitera votre autonomie de décision et d'orientation. Cependant, cette option ne doit pas être écartée à la légère, notamment lorsque la disponibilité de l'équipe principale est insuffisante. Ceci peut être également une première étape avant une remise en cause du modèle et le lancement de votre propre communauté.

Si au terme de votre réflexion vous décidez de vous lancer dans l'aventure, l'analyse que vous avez menée doit se transformer en une veille concurrentielle qui devra être pérennisée de manière à repérer rapidement toute évolution ou repositionnement d'un projet concurrent, ou encore l'arrivée d'un nouvel entrant. Il est indispensable d'être capable à tout instant de situer son logiciel par rapport aux autres et de disposer d'arguments objectifs permettant d'expliquer et de justifier les spécificités de votre projet.

Il est recommandé de documenter le choix initial en l'expliquant et en l'argumentant. Cela évitera d'avoir à refaire l'histoire, en se reposant les mêmes questions six mois plus tard.

Une fois cette importante décision prise, vous avez une idée précise de la direction que vous allez prendre, vous savez avec qui vous allez mener le projet et vous connaissez votre environnement. Il est temps de vous concentrer sur votre projet. Le chapitre suivant aborde des points pratiques de sa mise en œuvre et de son organisation.

3 Structuration du projet



Today's society has a different topology – a peer-to-peer mesh
Simon PHIPPS (#fOSSa2011)

À ce stade, votre équipe principale doit être alignée sur les valeurs et les objectifs définis. Mais il s'agit du dernier moment où des dissensions pourront se manifester sans remise en cause douloureuse, aussi faut-il veiller à ce que la vision du projet soit partagée. Il s'agit d'une condition indispensable pour que l'équipe produise un effet d'entraînement sur votre future communauté.

3.1 Architecture de participation

N'oublions pas que l'objectif est de dynamiser le noyau dur et de développer votre communauté en y agrégeant toujours plus de membres. Pour cela, il faut définir clairement son organisation en répondant à une série de questions :

— quel sera le rôle de l'équipe principale par rapport aux autres membres de la communauté ? Cette équipe adoptera-t-elle un mode « dynamique » ou « stable » ?

— quel équilibre des pouvoirs entre l'équipe et les contributeurs externes ? Comment se prendront les décisions au sein de la communauté ?
— quelles méthodes de développement seront utilisées ? Quelles règles seront employées pour la production du code et de la documentation ?
— comment seront gérés les utilisateurs non contributeurs ? Quels seront leurs droits et leurs obligations ?
— comment sera animée la communauté ?

Tout ceci constitue ce qu'on appelle l'architecture de participation, dont l'ossature est la gouvernance de la communauté. Gardons en mémoire que pour obtenir une communauté réellement engagée, il faut troquer une partie du contrôle de son projet contre de la participation active et engagée. Il faut donc un certain degré d'ouverture dans l'organisation et dans les processus. Mais une communauté *open source* n'implique pas une absence de règles et, suivant les projets, le contrôle de l'équipe principale peut être plus ou moins fort. Entre le bazar et la cathédrale, le champ des possibles est grand et l'essentiel est d'avoir une architecture de participation qui reflète les valeurs qui ont été définies au préalable.

Pour autant deux notions resteront incontournables : la transparence et l'agilité. Les participants à la communauté n'adhéreront aux valeurs et à l'organisation que si elles ont été clairement explicitées. Enfin, il faut une architecture souple car il est difficile de prévoir avec certitude vers où une communauté peut emmener votre projet et dans le cas de communauté à l'architecture de participation très ouverte, de savoir si celui-ci restera encore votre projet. La perte (ou au moins une forte diminution) de contrôle de votre projet ne doit pas être une crainte pour vous mais une évolution possible assumée.

Passée la phase de démarrage, l'architecture de participation doit permettre de cristalliser un consensus autour des valeurs du projet au fur et à mesure de l'arrivée ou du départ de participants à la communauté. L'obtention de ce consensus ou, ce qui revient au même, le partage d'une vision commune sur le projet est un moment tout à fait important du lancement d'un projet de communauté *open source* ; le maintien de ce consensus sur la durée est le véritable défi.

Il conviendra de mettre en valeur les différents documents fondateurs de la communauté : la licence (ou les licences), le contrat de contribution (CLA ou *Contributor License Agreement*), les règles de décision, les possibilités offertes de participation, la définition des rôles, les valeurs portées par le logiciel et le modèle économique – car même s'il ne s'exprime pas sous la forme de chiffre d'affaires, il y a forcément un modèle de coût associé au projet. Il faudra veiller à rester simple, agile mais non dénué de rigueur : une alchimie complexe.

Lorsque la croissance du projet mène à la création d'un consortium ou d'une association, il est possible de décider une cession totale des droits des contributeurs sur les codes produits, on parle alors de *Copyright Assignment*. Dans ce cas, la communauté ou la structure légale qui l'anime et qui porte le projet devient alors l'unique propriétaire des codes et donc l'unique décideur concernant les droits patrimoniaux, c'est-à-dire ceux liés aux aspects économiques et légaux (par ex. un changement de licence devient alors extrêmement simple). Dans le cas d'une cession partielle des droits, pour lesquels l'auteur qui contribue garde encore une partie de ses prérogatives, on parle en général d'un *Contributor License Agreement*. Ces documents qui seront signés par les contributeurs doivent être doublés d'un document annexe signé par l'employeur du contributeur lorsque ses contributions sont réalisées dans le cadre d'une activité salariale.

3.2 Les valeurs

Il est recommandé de publier les valeurs portées par le logiciel. Ces documents seront également mis à profit pour encadrer et éduquer les membres de la communauté.

N'oublions pas que l'objet de cette communauté est de produire et de faire vivre un logiciel et que les règles de production de code sont un ensemble essentiel pour l'adhésion des contributeurs. Il s'agit de s'inspirer davantage des méthodes agiles que des cycles en V pour permettre différents niveaux d'implication. Les règles de développement (*coding rules*) gagneront à être aussi précises que possible, depuis la production du code ou des documents jusqu'à la revue des différents éléments. Si de telles règles sont déjà en place au sein de votre équipe principale, assurez-vous

qu'elles sont bien compatibles avec votre architecture de participation et ajustez-les au besoin.

3.3 Comment choisir une méthode de gestion de projet ?

Il n'existe pas de bonne ou de mauvaise méthode de gestion de projet mais des méthodes plus ou moins adaptées aux circonstances. Lorsqu'un projet n'est pas bien défini, qu'il est incertain, qu'il est complexe, qu'il concerne un objet aux interactions complexes, alors il est probable que le recours aux méthodes agiles est une bonne idée. Si par contre il peut être décomposé en phases et en tâches distinctes et prévisibles, qu'il met en œuvre du matériel, qu'il est tributaire d'aléas externes, alors les méthodes traditionnelles planifiées sont probablement plus adaptées. Une bonne approche consiste à considérer le projet dans son ensemble et à en identifier les parties à traiter de façon agile et celles à traiter de manière traditionnelle.

Pour permettre à la machine de bien fonctionner, il faudra trouver le bon *timing* pour transformer votre équipe principale en véritable bureau exécutif (*board*). On commence par affecter aux membres de l'équipe motivés les rôles définis dans l'architecture de participation. Attention, certains membres de l'équipe principale pourront ne pas désirer un rôle autre que développeur et il faut évidemment en tenir compte. Il peut être efficace pour apporter du sang neuf d'intégrer au *board* une personne extérieure à l'équipe pour questionner vos certitudes. Une ouverture est nécessaire car il n'est pas toujours facile de conserver une position lucide lorsque l'on est totalement impliqué dans un projet.

3.4 Habitat numérique

L'habitat numérique (*digital habitat*) désigne l'ensemble des outils techniques permettant de présenter le logiciel et de gérer la communauté. L'habitat numérique est en quelque sorte le « lieu de vie du projet ». L'habitat numérique est constitué d'une partie « dure », matériel, outils, serveurs, listes de diffusions, blogs, forum, etc. mais également de règles de vie, d'un code de bonne conduite. Enfin, comme une maison classique, un

habitat numérique n'est pas isolé du monde : il possède des fenêtres sur l'extérieur, des moyens d'accès, etc. Cette analogie facilite la réflexion.

L'habitat numérique prend en charge les trois dimensions suivantes :

3.4.1 La gestion du code

Ce point est central pour attirer des développeurs ; la forge logicielle utilisée (ici le mot forge est employé de façon générique pour désigner l'ensemble des outils permettant la production et la maintenance du code) doit permettre la mise en œuvre des *coding rules*. Attention, des outils trop complexes, même s'ils sont puissants, présentent une barrière à l'entrée qui risque de dissuader les contributeurs et de limiter le nombre de participants potentiels. Par contre, utiliser des outils trop simplistes ne permettant pas une gestion efficace de la production de code risque, *a contrario*, de faire fuir les développeurs de haut niveau. Le choix de la forge doit donc être déterminé avec soin en prenant en compte le profil de contributeurs que l'on vise.

3.4.2 L'animation

Les outils de gestion de code peuvent proposer des fonctions d'animation de la communauté. Si ce n'est pas le cas ou si ce n'est pas suffisant, le volet collaboration de votre projet devra être complété. Ces outils d'animation doivent essentiellement faciliter la participation, la collaboration, la diffusion d'information et le suivi (*monitoring*) de la croissance du projet (nombre de visiteurs, nombre de membres, etc.)

3.4.3 La vitrine

Enfin, pour recruter les utilisateurs et les développeurs nécessaires pour atteindre vos objectifs, il faut faire connaître votre projet. Pour cela, utilisez des outils de promotion et faites en sorte que l'utilisation de l'habitat numérique soit simple et directe. Un point clé est de garantir l'accessibilité du code qui doit être téléchargeable très facilement (affichage clair sur la page d'accueil) et directement (nombre minimum de clics, etc.) et proposer de la documentation, des exemples et des tests. En plus de cet accès

direct aux contenus techniques, des outils doivent permettre d'informer la communauté – et le reste du monde – sur l'activité de ce projet de manière synthétique (nouvelles versions, extensions fonctionnelles, présentation des nouveaux membres, utilisateurs et / ou contributeurs, nouveaux partenariats, etc.) Il faut veiller à ce que ces outils permettent une gestion simple et efficace de la promotion de votre logiciel. Le téléchargement du code, notamment, doit être totalement libre, sans être assujetti à une prise de contact, à une autorisation, ni même au remplissage d'un formulaire.

3.5 De l'analyse à la veille

L'analyse que vous avez menée doit devenir maintenant une veille permanente pour maintenir à jour les informations sur l'écosystème, parce que certains groupes peuvent s'y révéler particulièrement dynamiques, dans le temps. Il est donc utile d'identifier les personnes clés de ces communautés, sans se limiter aux acteurs ayant une position officielle dans l'organisation. Par exemple le fait d'identifier un développeur très actif qui n'est pas référencé comme membre de l'équipe principale est important. Il est nécessaire également de comprendre le mode de gouvernance de ces communautés (formelle, informelle, etc.), les outils communautaires sur lesquels les interactions s'appuient (extranet, intranet, wiki, listes de diffusions, réseaux sociaux, forges, etc.). Le fait de visiter régulièrement ces différents sites et outils permet de mesurer la dynamique de ces groupes.

3.6 Les actions de base de la veille

Pour une veille efficace, on peut :

— maintenir une liste de projets, les contacts des personnes clés, le profil de la communauté et vérifier régulièrement l'avancement de ces projets, même superficiellement ;
— obtenir des informations par des développeurs issus de la communauté de logiciels complémentaires ou des bibliothèques employées ;
— identifier les lieux de communication, de partage et d'échange de votre domaine ;

— utiliser des outils de veille automatisables.

Au-delà du Web, il convient de compléter la veille Internet en regardant d'autres médias, notamment la presse écrite, les articles de recherche, la participation des communautés à des salons, des ateliers scientifiques, la mise en place d'opérations de promotion propre, l'organisation d'actions dédiées aux développeurs et / ou d'opérations de formation, etc. Vous collecterez ainsi des informations qui pourront être utilisées lors de la phase de promotion de votre logiciel sous la forme de vecteurs de communication et de lieux ou évènements permettant les partages et les échanges.

3.7 Ne pas hésiter à demander conseil !

N'hésitez pas à échanger, à demander conseil auprès de personnes impliquées dans d'autres projets de logiciel libre et dans l'ingénierie du développement logiciel afin de définir au mieux votre infrastructure de collaboration.

À ce stade, vous avez défini votre habitat numérique : murs, fenêtre, porte, règles de vie. À présent, il ne reste plus qu'à le construire (ou, dans le cas où il existait déjà, de le rénover) ; dans le prochain chapitre, nous aborderons la mise en œuvre de votre architecture de collaboration et la publication de votre code.

4 Habitat numérique et publication du code

Training tomorrow's engineers by contributing to today's free software
Albert COHEN (Inria Senior Research Scientist, #fOSSa2011)

La phase précédente a permis une définition précise de l'architecture de participation, des documents fondamentaux, des fonctionnalités et des outils techniques qui donneront vie au projet.

Suivant l'historique de votre projet, certains outils techniques peuvent déjà être en place, ne couvrant le plus souvent que partiellement les besoins fonctionnels exprimés.

Il convient donc maintenant de construire (ou étendre, rénover suivant l'état d'avancement) votre habitat numérique (votre lieu de collaboration entre utilisateurs et développeurs). Cette construction technique doit trouver le bon compromis entre fonctionnalités et simplicité. Il est important d'adapter les choix techniques définitifs (éventuellement en remettant en cause des outils préexistants) à vos capacités d'administration ; d'un autre côté, tout changement technique ultérieur perturbe la communauté et il convient d'anticiper au mieux les besoins futurs dans le choix des outils déployés.

Une architecture technique doit être mise en place pour chacune des trois dimensions de l'architecture de participation : la *gestion de code*, la *vitrine* et l'*animation*. Il faut éviter le réflexe de s'intéresser prioritairement à la *gestion de code* de votre habitat, tendance naturelle chez les développeurs. Il s'agit de commencer prioritairement par la *vitrine* qui sera votre vecteur de communication principal, suivi assez rapidement par le déploiement d'un minimum de fonctionnalités de l'*animation*.

Fort heureusement des outils, qu'ils soient complètement intégrés ou composés à la demande, existent pour faire vivre votre projet :

— ceux-ci doivent être testés,
— renseignez-vous auprès des autres communautés qui les utilisent et,
— comme ils seront très certainement *open source*, assurez-vous de leur stabilité et du soutien d'une communauté.

Les outils de collaboration, de participation et d'animation devront permettre d'avancer vers les objectifs et de mettre en lumière les valeurs telles qu'elles ont été définies précédemment (cf. chapitre 1) et dont on a tiré les éléments qui ont permis le choix de l'architecture de participation.

Lors de l'analyse du contexte (cf. chapitre 2) vous avez identifié – pour votre domaine – des vecteurs de communication habituellement utilisés, des lieux de partage et d'échange où vous pouvez toucher les contributeurs et utilisateurs. Posez-vous la question de savoir s'il vous faut créer/installer des outils spécifiques à votre projet. Il est toujours très tentant de mettre en place son propre forum, sa propre forge logicielle, de créer sa manifestation, son événement, mais le coût humain et/ou financier doit pousser à y réfléchir à deux fois.

4.1 Architecture de participation pour débutants

Si vous débutez, nous vous conseillons de mettre en place en premier lieu un habitat numérique simple composé de 3 lieux qui permettent de :

— présenter l'équipe, la communauté, et publier des informations diverses sur votre projet (ex. portail web : DokuWiki, Wordpress, Drupal, Elgg, etc.),

— de collaborer sur le code à plusieurs facilement (ex. un gestionnaire de version du code (SVN, CVS, GIT), un *issue tracker* (BugZilla, Track, etc),
— s'exprimer chacun de manière simple (ex. un canal IRC (Jabber, Freenode, etc.), une liste de diffusion (sympa, mailman…) ou un forum (phpbb, discourse, etc.).

Point important : il est indispensable de prendre les dispositions nécessaires pour garantir la tenue à jour de l'habitat numérique, que ce soit dans son contenu ou dans ses outils (définir un calendrier ainsi qu'un responsable).

Une perte de temps classique est d'attendre la mise en place de l'ensemble de l'architecture technique pour commencer à annoncer la création (ou le changement de dimension) de votre projet. Il est préférable d'annoncer sur le site initial que tout n'est pas encore en place, en présentant l'ensemble du projet de façon claire et précise. Ceci permet d'avancer la campagne de communication en diffusant l'URL du projet au plus tôt dans les médias adaptés aux cibles visées.

Très rapidement, il faudra pouvoir disposer d'un minimum d'outils de communication pour permettre aux visiteurs de ne pas être passifs face à votre site, mais de pouvoir laisser un commentaire, un souhait, s'inscrire sur des listes de diffusion, poser une question… et bien sûr recevoir une réponse.

4.2 La vitrine

La vitrine est un portail web permettant aux visiteurs de parcourir votre projet dans toutes ses dimensions depuis un point d'accès centralisé. Il existe une très large littérature pour proposer une présentation aussi attractive que possible. Les dimensions spécifiques au monde du logiciel libre qu'il faut particulièrement prendre en compte sont la transparence et l'ouverture : les valeurs que vous avez choisies doivent clairement transparaître dans la présentation de votre vitrine, l'accès aux documents et à l'information doit être simple et direct. On doit immédiatement comprendre que l'on est sur le site d'un projet de logiciel *open source*, voir l'état du développement actuel, comprendre comment la participation est

possible, accéder rapidement aux téléchargements (le code, une démo, un exécutable, les guides, etc.).

4.3 Message clair sur la première page du site vitrine

En arrivant sur la page d'accueil du portail, on doit voir immédiatement qu'il s'agit d'un site dédié à un logiciel *open source*, cerner en quelques secondes la fonction principale de ce logiciel, ce qui fait son originalité, sous quelle licence il est édité, quel est son statut courant, et repérer en un coup d'œil les liens vers le téléchargement et vers la page d'aide.

Ensuite, il faut appliquer les recettes qui permettent d'avoir un bon site web, une belle vitrine. L'une des particularités des sites de logiciels est la présence de visiteurs de types différents qu'il faut traiter par une navigation à facettes : l'utilisateur potentiel qui a besoin d'un outil ne cherche pas les mêmes informations que le développeur qui pourrait éventuellement contribuer. Il faut penser au primo visiteur sans oublier les habitués, contributeurs ou pas, qui visitent régulièrement le site. L'impression que tout visiteur du site doit ressentir est que la communauté est très active et que le projet est en mouvement. Il faut donner envie aux membres de revenir régulièrement découvrir des nouveautés.

L'aspect graphique doit refléter les valeurs du projet. Le recours à un CMS (Content Management System – système de gestion de contenu) permet une réalisation simple et efficace du portail web d'un projet. Il est possible d'acheter pour un prix modique des modèles de mise en page (*templates*) d'aspect professionnel. Mais on peut aussi demander à des étudiants de faire un design spécifique. Dans tous les cas il ne faut pas oublier de mentionner les noms des auteurs originaux du design utilisé sur le site. À ce propos, le fait de favoriser les licences qui permettent les modifications lors du choix des images et/ou du design permet ensuite de se simplifier la vie.

À l'affichage de la page d'accueil, on considère traditionnellement que la vitrine a 45 secondes pour convaincre. Le visiteur doit immédiatement pouvoir identifier le besoin adressé par le projet et trouver le lien de téléchargement des versions exécutables. Il doit rapidement pouvoir trouver

comment contacter la communauté. La lisibilité est capitale et on doit obtenir les informations essentielles au premier coup d'œil, sans surcharger le visiteur d'informations. Les éléments qui doivent impérativement être mis en avant sont :

— le nom du logiciel ;
— l'objectif de ce projet, en quoi il diffère des autres. Dites-le !
— l'état courant du développement (version bêta ? version stable ?) ;
— des liens directs pour télécharger les versions exécutables, le code (versions stables, versions en développement, anciennes versions, par plate-formes) ;
— des liens vers plus d'informations, pour suivre le projet, pour y contribuer ;
— des liens vers la description des principaux membres et comment les contacter.

Il est également conseillé d'insérer quelques liens pointant depuis la page d'accueil vers des pages internes :

— les actualités du projet : cette rubrique doit absolument être mise à jour très souvent pour montrer que le projet est bien vivant. L'affichage d'un fil de *news* peut être un moyen de le faire sans contrainte forte ;
— les valeurs du projet : les développeurs internes et externes participeront plus facilement s'ils reconnaissent leurs propres valeurs dans celles affichées (motivation intrinsèque) ;
— le planning (feuille de route) : même si les délais ne sont pas toujours tenus, ou si le planning est parfois mouvant, il faut afficher un planning simple et le mettre à jour très régulièrement.

Il est très important de proposer un habitat numérique facilitant la participation :

1. les utilisateurs peuvent-ils remonter des bugs ?
2. les utilisateurs peuvent-ils proposer de nouvelles fonctionnalités ?
3. les utilisateurs peuvent-ils facilement poser des questions ?
4. les développeurs externes ont-ils la possibilité de proposer leur code à l'équipe principale ?

Le contenu du site doit permettre aux visiteurs et contributeurs de comprendre votre projet, d'y participer et de le suivre.

N'hésitez pas à vous inspirer de sites web existants que vous considérez (ou que la communauté considère) comme « bien faits ».

4.4 Exemple de listes de diffusions

- nomprojet−contact@... : à utiliser quand...
- nomprojet−user@... : à utiliser quand...
- nomprojet−dev@... : à utiliser quand...
- nomprojet−discussions@... : à utiliser quand...
- nomprojet−webmaster@... : à utiliser quand...
- nomprojet−strategy@... : à utiliser quand...
- nomprojet−news@... : à utiliser quand...
- nomprojet−updates@... : à utiliser quand...

4.5 Exemple de catégories pour un forum

- *User Help* : This topic is dedicated to any user who wants to ask a question about −your project name− USAGE.
- *Tutorials & Videos* : This category is dedicated to Videos & Tutorials about the usage of −your project name−.
- *Community Initiatives & Events* : You want to post a news about −your project name− ? Post an event ? Propose an initiative ? Leave us a post in this category !
- *Developers Help* : This topic is dedicated to any user who want to ask a question about CODE DEV. around −your project name− (Plug ins, compilation of the −your project name− platform, Python scripting, etc.).
- *Report a bug* : This topic is dedicated to any user who want to post any bug encountered in −your project name−.
- *Features wishlist* : This topic is dedicated to any user who want to post ANY NEW FEATURE about −your project name−.
- *Tips & tricks* : Use this category to give advice and tips using −your project name− and discuss techniques and insights.

— *Share your creations* : Use this category to share your creations using –your project name– for other community members to give advices. You can also share other resources like plugins, addons, softwares, images.
— *Present yourself* : Community is important for us ! If you are a new comer, don't hesitate to present yourself in this section. Post what you are doing, what are your interests, and any other funny (or not) information that you want to share with the other members.
— *Forum, Website & Wiki help* : You encounter a issue with the forum ? the wiki ? the website ? Leave us a post in this category !
— *Guidelines to use this forum* : If you are a new comer, we advise you to read the guidelines to use this forum and the code of conduct. . . .
— *Staff Discussions* : Private category for staff discussions. Topics are only visible to admins and moderators.

4.6 Exemple de contenu pour un site vitrine

Contenu de la première page :

1. nom du projet, avec la citation « open source » et / ou « open hardware », et son *mission statement* ;
2. la licence utilisée ;
3. l'état de développement du projet ;
4. comment télécharger ;
5. comment installer et utiliser ;
6. comment contribuer ;
7. comment interagir avec les membres de l'équipe et la communauté.

Section « À propos » :

1. vision et concept du projet ;
2. quelques mots sur l'équipe principale ;
3. quelques mots sur les grandes lignes du projet, faits marquant. . . ;
4. qui utilise votre application et des témoignages ;

5. autres faits, anecdotes sur le projet.

Section « Visiteurs pour la première fois » :

1. comment démarrer. Quelles sont les premières étapes pour utiliser le projet ;
2. pourquoi devenir un membre ;
3. quelles sont les premières étapes pour contribuer au projet ;
4. quelques conseils aux contributeurs débutants et visiteurs, mais aussi ce qu'il ne faut pas faire ;
5. quel est le code de conduite que vous espérez des contributeurs, des visiteurs et en quoi vous vous engagez à votre tour ;
6. quels sont les amis (communautés externes qui vous soutiennent, etc.) ;
7. comment accueillir un débutant, une nouvelle personne sur le forum, etc.

Section « Gouvernance » :

1. modèle de gouvernance du projet ;
2. comment sont prises les décisions ;
3. comment sont gérés les possibles conflits. Avez-vous un médiateur (le CM) ou un groupe de personnes ? ;
4. comment on devient un nouveau membre, un nouveau contributeur ;
5. description de l'équipe principale, quels sont les rôles et responsabilités ;
6. quels sont les groupes, sous-groupes, groupes de travail ;
7. quelle est la stratégie de propriété intellectuelle, quelle est la licence du code et celle des autres ressources liées au projet.

Section « Communication » :

1. quel est le courriel de contact pour joindre l'équipe principale ? ;
2. quel est le courriel de contact pour joindre le responsable d'un sous-domaine du projet ;

3. quel est le canal IRC pour discuter entre membres de l'équipe et la communauté ;
4. quelles sont les listes de diffusions et archives.

Section « Activités quotidiennes » :

1. que fait le secrétariat : le CM fait… ;
2. règles de sécurité si besoin (expérience de laboratoire, *open hardware*, fablab…) ;
3. réunions (avec dates, modèles de rapports des réunions, liens vers les réunions passées, contenu des comptes rendus, etc.) ;
4. comment contribuer au code, qui contribue actuellement, etc. ;
5. comment contribuer à d'autres activités ;
6. comment un visiteur peut proposer une initiative, un événement :
   — listes des initiatives et événements,
   — documentation expliquant comment proposer et organiser un événement relatif au projet.

Section *Goodies* :

1. proposer le téléchargement de fonds d'écran, de modèles de présentations, de logos, de bannières pour site web, etc. ;
2. proposer un bouton « acheter » afin de vendre des *goodies* à la communauté.

Adapter ces sections à votre projet !

Nous vous conseillons d'adapter ces sections en fonction de votre projet.

— Pour exemple, *certains sites n'utilisent que 4 sections* : Accueil, Téléchargements, Documentation, Communauté. L'onglet *Communauté* contient des pointeurs vers le forum, liste de diffusion, etc. Cela produit, sur le visiteur, l'effet d'un site web simple et clair.
— Quoi que vous mettiez en place, il est important de respecter – a minima – le contenu de la section « Contenu de la première page ».

4.7 Référencement de votre site vitrine

Créez votre site de manière à faciliter le référencement de votre projet par les moteurs de recherche. La toile regorge d'informations sur ce point. Attention cependant car les critères pris en compte pour une indexation efficace évoluent vite. Cela suppose de faire une veille régulière d'autant plus importante que les premiers à prendre en compte les nouvelles méthodes bénéficient facilement d'une bonne visibilité.

4.8 Quand publier son code pour le rendre accessible à tous ?

Le slogan *Release early, release often* est souvent cité comme l'un des facteurs clés de succès pour l'amorçage des projets *open source*. Pour autant, si un code est publié trop tôt on risque d'avoir du mal à attirer des contributeurs car les visiteurs risquent de ne pas comprendre quels sont les objectifs du projet, quelles sont les valeurs portées par le projet, quel est son modèle de gouvernance, etc. Si par contre on attend d'avoir un code bien écrit, à l'état de l'art, bref, parfait, alors il est probable qu'il ne sera jamais publié (la perfection est longue à atteindre, surtout vers la fin…). Et si c'est néanmoins le cas, les contributeurs potentiels seront moins tentés de rejoindre un projet où la liberté de l'écriture est moindre qu'au début.

Pour autant, la simple mise à disposition du code ne suffit pas à assurer sa diffusion. Certains développeurs restent sur le concept « Je code, je publie, j'existe », alors que sans promotion ils n'ont à peu près aucune chance de faire décoller une communauté. Le prochain chapitre aborde la communication autour du projet. Une fois que votre habitat numérique est opérationnel et que les codes sont publiés, la préoccupation principale est donc de promouvoir l'existence de votre logiciel afin d'amorcer sa diffusion.

5 Promotion du projet

Celui qui unit ses forces avec d'autres projets sera payé en retour
Vieux proverbe libriste

Lorsque votre habitat numérique est en place et que vous avez un embryon de communauté, c'est le moment de promouvoir votre projet, d'annoncer qu'il existe, etc. Cette promotion peut prendre de nombreuses formes : participation à des manifestations, publication d'articles et de news, recours aux réseaux sociaux, etc.

Lorsqu'on débute, il est toujours difficile de savoir à qui et où envoyer des informations. Une méthode simple et efficace consiste à commencer à envoyer des mails à son réseau de contacts proches en demandant à l'équipe principale d'en faire autant. N'hésitez pas non plus à suivre des blogueurs et communautés proches des centres d'intérêt de votre projet (en vous inscrivant sur leur forum, leur liste de diffusion, en leur posant des questions, en intervenant dans les posts, etc.). Ces contacts s'ajouteront à votre liste de contacts pour la promotion. Il est même souhaitable de constituer cette liste de blogueurs le plus tôt possible et de la tenir à jour en la partageant.

Le fait de diffuser un même message sur plusieurs canaux renforce la cohérence de la communication. Une méthode éprouvée consiste à préparer un texte qui fait l'unanimité et de l'envoyer de manière synchronisée à travers tous les canaux et sur tous les réseaux possibles. Cela fait gagner du temps à l'équipe principale et permet au projet de parler d'une seule voix.

La phase d'analyse du contexte (cf. chapitre 3) doit aussi faciliter le travail de promotion en s'appuyant sur des personnes et des communautés identifiées et que vous suivez. N'hésitez pas à les mobiliser pour diffuser régulièrement des informations sur son évolution.

5.1 Quand promouvoir son code ?

Choisir le bon moment pour promouvoir l'existence de son projet est crucial pour créer une communauté motivée. Cette question doit se poser pendant les toutes premières étapes du projet. On peut déterminer un planning en surveillant l'avancement de chacune des facettes du projet et lancer les actions de promotion lorsque tout l'habitat numérique (base de code, contenu du site vitrine, veille et contexte, etc.) est prêt à accueillir décemment le flux d'utilisateurs et de contributeurs potentiels.

Les actions de promotion de base consistent à contacter les organisateurs de conférences, de colloques, d'ateliers, de journées et autres événements que vous avez repérés afin de leur faire publier une brève, un article autour de votre projet et pourquoi pas vous faire intervenir ou organiser un *workshop* dédié si le projet est suffisamment avancé.

Une cible à ne pas négliger : les éditeurs de contenus potentiellement intéressés par des informations sur votre logiciel. Cela inclut les éditeurs de webzines, les blogueurs, les rédactions des journaux et magazines de la presse écrite. Les médias numériques et les blogueurs sont généralement plus faciles à contacter et ont souvent une plus grande diffusion que la presse écrite mais cela vaut le coup d'essayer de publier dans cette dernière car son prestige reste élevé.

Pour cela vous ferez un communiqué de presse que vous diffuserez à des journalistes et des rédacteurs en chef de journaux et magazines. Les services de communication comptent souvent un chargé de relations presse

qui peut vous aider à la rédaction de votre communiqué ou à son adaptation dans un style journalistique, et à son envoi à des personnes de leur réseau. Les magazines spécialisés dans la programmation et l'informatique sont souvent preneurs de brèves sur les nouveaux projets *open source* et peuvent être contactés en direct. Il faut toutefois choisir le moment d'une telle communication. S'il y a une actualité trop dense cela n'est pas le meilleur moment pour soumettre un article long, mais cela permet, à l'inverse, de surfer sur la vague en soumettant une brève dans un numéro très lu. Le fait de décaler sa communication pour profiter d'une meilleure fenêtre de tir peut changer les choses.

Il faut aussi faire relire le communiqué de presse par quelqu'un qui n'a pas été impliqué directement dans le projet. Même un journaliste spécialisé n'aura pas les connaissances des scientifiques ou des développeurs qui ont travaillé plusieurs mois voire des années sur le projet. Il a peu de temps pour préparer le travail d'explication ou de vulgarisation vers son lectorat, par conséquent, plus le communiqué est clair, simple et structuré, plus il facilitera le travail du journaliste.

Il faut aussi définir la cible médiatique : envoyer un communiqué de presse à un journaliste qui ne se sentira pas concerné ne servira à rien. Au mieux il ne le lira pas, au pire il sera agacé car submergé d'informations inadaptées pour lui (ce qui nous énerve tous bien souvent). Plutôt que d'arroser trop de journalistes il est préférable de bien les cibler – en fonction du domaine. Vous devez leur apporter un service, une information utile, pour être entendus.

Si tout cela est bien fait, malgré tout, il ne faut pas être trop présomptueux sur l'impact attendu. Le journaliste n'est pas seul décideur dans une rédaction et il doit gérer, lui aussi, ses priorités. Peut-être ne pourra-t-il pas relayer votre communiqué de presse tout de suite mais vous appellera-t-il à votre grande surprise six mois plus tard pour une interview ? Il aura peut-être gardé votre projet en tête et vu l'opportunité d'en parler à ce moment-là. Il ne faut pas négliger la vision court terme, moyen terme et long terme lors de la phase de promotion.

Si votre communiqué de presse est repris dans un article mais qu'il vous déçoit, par exemple par son imprécision, le journaliste a probablement préféré mettre l'accent sur des points qui ne vous paraissent pas majeurs et inversement. Dans ce cas prenez du recul, sauf en cas d'erreur majeure :

on parle de votre projet, de votre équipe, de votre événement et finalement c'est déjà un grand pas.

5.2 Vous avez dit Buzz ?

Les *community managers* adorent lancer des buzz en amorçant une communication virale positive autour de leur projet. Comment s'y prennent-ils ? La clé est d'attirer l'attention par un propos décalé, amusant ou énigmatique. Prenons l'exemple du concours *Boost your code*, un appel à projet de logiciel libre lancé par Inria en 2011 grâce auquel le lauréat se voit proposé un CDD d'un an pour développer son logiciel en immersion dans une équipe de recherche. Le problème consistait à attirer l'attention des étudiants qui reçoivent déjà des tonnes de courriels publicitaires chaque jour. Il a été décidé de diffuser une nouvelle à travers les différentes listes de diffusion et les réseaux de contacts identifiés en choisissant la date du premier avril pour créer un doute : « Quoi ! ? Inria propose un contrat d'un an pour développer son propre projet de logiciel libre ? Impossible, c'est un poisson ! »

L'effet souhaité est généralement obtenu et la nouvelle se propage sur les réseaux sociaux et les forums étudiants… trop cool ce concours, mais c'est pour de bon ou c'est un poisson ? …c'est un poisson d'avril ! ils le disent dans le mail, …mais non, c'est du sérieux, regardez sur leur site !

5.3 Utilisation des réseaux sociaux

Pour communiquer sur un projet de logiciel libre, les réseaux sociaux constituent aujourd'hui un moyen très efficace et quasi incontournable. Il faut les mobiliser pour diffuser des informations sur la vie du projet au jour le jour, annoncer les manifestations et autres évènements importants (avant, pendant et après), etc. Il ne faut pas hésiter à suivre un maximum de monde, blogueurs et communautés (en s'inscrivant sur leur forum, leur liste de diffusion, en les questionnant et en interagissant). Ces actions permettent d'étendre ses réseaux et de faciliter la promotion du projet et le recrutement de nouveaux membres.

Cette multiplication des canaux de diffusion permet d'atteindre des personnes et des communautés diverses et d'augmenter le nombre de visites sur votre site. Attention cependant de ne pas « sur-communiquer » car il n'y a rien de pire pour un projet que de se faire coller une étiquette de gesticulateur voire de spammeur.

Il est important de créer des comptes pour votre projet sur différents réseaux sociaux existants et visibles : Identi.ca, Twitter, Whatsapp, FramaSphere, ResearchGate, etc.

Vous devrez tout d'abord vous abonner à d'autres personnes, les suivre, puis petit à petit, à votre tour, vous publierez des contenus « intéressants » sur ces réseaux sociaux, vous obtiendrez alors sûrement de nouveaux membres et une certaine viralité (vos suiveurs feront suivre vos informations à leurs suiveurs). La construction d'une base de suiveurs peut prendre beaucoup du temps, armez-vous de patience.

5.4 Utilisation d'une chaîne vidéo

Si les moyens et le temps le permettent, vous pourriez créer une chaîne vidéo « officielle » (Sur Dailymotion, Youtube, etc.) et inciter les membres de votre communauté à publier (ou relayer) des vidéos sur leurs propres chaînes. Que la vidéo soit diffusée sur la chaîne officielle du projet ou sur leur chaîne personnelle, il est important :

— de ne faire que des vidéo très courtes (2 à 6 mn max, en les découpant en parties si nécessaire) ;
— de veiller à une bonne qualité de son ;
— de toujours mettre dans les commentaires un lien vers le code et un lien vers le portail du projet ;
— de toujours, à la fin de chaque vidéo, rappeler au visionneur (qui par définition l'a suivie jusqu'à la fin) de la *Liker* et/ou de s'abonner à la chaîne.

Vous pouvez aussi proposer aux membres de soumettre leurs vidéos et intégrer les meilleures sur la chaîne officielle après mise à la charte (montage, insertion du logo, etc.).

5.5 Devenez un bon rédacteur

Synthétisez votre message jusqu'à obtenir un texte simple et clair. Pensez toujours à faire des courriels très faciles à lire (correctement structurés : paragraphes courts séparés par une ligne vide, publier les URL directement dans le texte pour alléger, placer des illustrations pour rythmer le propos, éviter les passages comportant trop de style en gras, etc.). Vous pouvez créer un message orienté grand public (pensez vulgarisation) et un autre plus technique. Cependant, ne cumulez pas les messages antérieurs, rappelez simplement le contexte. N'alourdissez pas inutilement vos courriels avec des informations annexes : utilisez des liens pointant sur les échanges précédents ou sur des documents en ligne.

La règle fondamentale consiste à ne communiquer que pour annoncer quelque-chose de nouveau, de réel, d'utile. Ainsi, lorsque vous mettez à jour votre site (uniquement pour des mises à jour significatives), que vous publiez une nouvelle version, que vous avez corrigé un bug récalcitrant, intégré une nouvelle fonction souvent demandée ou que vous lancez une opération importante, n'hésitez pas à communiquer tous azimuts (vos contacts et ceux de l'équipe principale, sur les réseaux sociaux, sur les listes de diffusion, dans les forums en vue, etc.).

5.5.1 Élaborer une annonce

Il est conseillé de suivre ces étapes :

1. définir l'histoire du projet (qui, pourquoi, comment) ;
2. trouver une accroche : un besoin, une question, un chiffre-clé, qui interpelle et donne envie d'en savoir plus ;
3. définir le/les porte-parole(s) : qui va parler du projet (parfois des porte-paroles différents en fonction des cibles), et les disponibilités de ces porte-paroles ;
4. définir à quel moment on décide de présenter le projet : quand est-il terminé ? à l'occasion d'un événement, d'une actualité ? ;
5. définir la/les cible(s) ;
6. préparer plusieurs types de courriels en fonction des cibles ;
7. préparer plusieurs *pitch* (5 min, 10 et 20 minutes).

5.5.2 Comment faire un communiqué de presse

Vous pouvez aussi faire un communiqué de presse sur votre participation à un événement ou sur le lancement de votre projet :

1. définissez l'objectif du communiqué de presse (afin, entre autres, de définir la cible, le style d'écriture du communiqué de presse et le timing) ;
2. dans le communiqué de presse, vous devrez expliquer le concept du projet ou de l'événement, décrire en une phrase à quoi sert le projet, qui le porte, ce qu'il va apporter concrètement ;
3. précisez les dates, lieux et un lien vers un site web, un nom de contact ;
4. écrivez un court résumé de quelques lignes qui reprend le message essentiel du communiqué ;
5. insérez une citation (une à deux phrases) de quelqu'un qui a participé au projet, ou qui en bénéficie et qui explique à quoi cela lui sert. Cela rend le communiqué plus vivant.

Lorsque vous écrirez le texte principal, pensez aux conseils suivants :

1. généralement 1,5 page maximum, scindée en 2 à 4 paragraphes avec un intertitre (attention à la densité, il faut faciliter la lecture le plus possible) ;
2. quand on peut insérer une photo, une infographie (courte), une petite vidéo dans le communiqué, cela le rend encore plus attractif et plus concret (mais ce n'est pas indispensable) ;
3. dire pourquoi vous vous impliquez dans cet événement ou pourquoi vous créez un tel projet ;
4. déclarer quels sont vos objectifs ;
5. expliquer la nature de votre contribution ;
6. dire comment vous vous différenciez des autres sponsors ou partenaires qui vont certainement communiquer également de leur côté ;
7. donner la liste des partenaires (avec leur logo) si nécessaire ;
8. terminer avec une phrase courte qui résume le communiqué ou un contact, un site Internet, le lien vers une publication scientifique. Cela doit donner envie au lecteur d'aller plus loin.

5.6 Lancement d'une campagne de promotion

Quelques conseils :

1. utiliser la liste de contacts pour annoncer l'existence de votre projet ;
2. envoyer des courriels à votre réseau de contacts proches ;
3. demander aux membres de votre équipe principale d'en faire autant en se calant et en se répartissant les cibles pour que tout le monde ne communique pas la même chose en même temps ;
4. envoyer les messages aux personnes, blogueurs et communautés que vous avez identifiés lors de la phase de veille ;
5. utiliser les réseaux sociaux pour annoncer l'existence de votre projet ;
6. faire des annonces lors de conférences, participer à un stand, etc. ;
7. écrire et publier des articles (publications scientifiques ou magazines plus généralistes) ;
8. ajouter dans votre signature de courriel un lien vers votre projet. Demander aux membres de l'équipe de faire de même, si cela leur est possible.

5.7 Autres moyens de faire de la promotion

Un autre moyen de faire la promotion de votre projet consiste à passer par des objets publicitaires. Ainsi, si vous avez un petit budget de communication, vous pourrez envoyer des *stickers* aux membres les plus actifs, ou même des tee-shirts si vous êtes riches. Vous pouvez aussi créer un carte de visite du projet, élaborer des tracts (à distribuer lors de rencontres) et lorsque le sujet s'y prête, prévoyez des visuels : photos, vidéos, infographies, etc. Les messages en seront d'autant plus relayés.

Enfin, on n'y pense pas toujours mais il existe aussi des sites qui proposent de publier des *podcasts* qui traitent des sujets autour du logiciel libre. N'hésitez pas à leur envoyer un message audio d'une minute.

5.8 Premier, deuxième et troisième cercle ? Késako ?

Lorsqu'on fait la promotion de son projet, on commence très naturellement par prêcher la bonne parole autour de soi, à son réseau de proches, à ses amis, à ses collègues. Les spécialistes de la gestion de communautés considèrent cet entourage comme le « premier cercle » de votre projet, de votre communauté. Il s'agit d'un cercle important en termes d'attachement au projet et dont les membres comptent parmi vos plus grands fans.

On appelle « deuxième cercle » les réseaux liés au premier cercle (réseau de réseau). Plus éloignées, les personnes du deuxième cercle sont attirées vers votre logiciel grâce aux informations sur l'existence de votre projet, qu'ils ont obtenues par le premier cercle. Ces deux cercles sont très importants et doivent être choyés car il s'agit de l'entourage de ceux qui croient le plus en votre projet.

Au-delà de ces deux premiers cercles vous toucherez (indirectement ou pas) un troisième cercle constitué de personnes au profil « grand public ». Plus difficile et plus long à atteindre, il vous faudra des mois, voire des années, avant de les atteindre. Mais une fois cela fait, il est probable que votre projet sera considéré comme étant populaire. Il est important de se concentrer d'abord sur les deux premiers cercles car ce seront eux qui vous permettront d'atteindre naturellement le troisième cercle. Vous pourrez aussi compter sur ces deux cercles pour constituer votre communauté principale, le noyau dur qui permettra de pérenniser votre base de code et votre projet.

5.9 Connaître les profils de vos utilisateurs

Que recherchent les utilisateurs ? les contributeurs potentiels à votre projet ? Quelles sont leurs motivations ? À quels besoins répond le logiciel auquel ils pourraient contribuer ?

Ces questions sont essentielles car un projet *open source* et sa communauté s'articulent autour de trois principes fondamentaux qui déterminent la probabilité de son succès. Il est nécessaire :

— qu'il réponde à un besoin largement partagé. Plus ce dernier est partagé plus il est facile de créer un réseau d'intérêt (de sensibilisation) puis une communauté de pratique (d'action) autour du projet ;
— qu'il soit publié pour élargir sa diffusion et augmenter les chances de bénéficier de contributions de développement externes (effet de levier) ;
— que chacun des membres de la communauté trouve un intérêt à participer, que ce soit en tant que développeur ou comme simple utilisateur (motivation).

Le dernier point soulève la question déterminante des motivations de chacun des membres de la communauté. Qu'est-ce qui pourrait inciter des personnes à rejoindre le projet, à télécharger le logiciel ou à participer au développement de son code ? Comment les contacter ? où les recruter ? comment les motiver ?

Une fois identifiés, ces éléments devront faire partie intégrante de la communication lors de chaque diffusion d'information : vous adapterez vos messages en fonction des personnes et communautés que vous visez mais aussi des profils types des utilisateurs et des contributeurs que vous avez identifiés.

5.10 Une vitrine doit toujours être bien propre et arrangée avec soin

Le site web du projet doit être tenu à jour, particulièrement les noms et adresses des contacts, les listes de diffusion, la feuille de route, les liens vers des sites web externes, de même que la documentation. Cette dernière sera à jour ou alors elle portera la mention obsolète. Cela évitera des déceptions chez les utilisateurs et rappellera aux contributeurs potentiels ou actuels qu'il faut y passer un peu de temps.

Le site web doit mettre en avant l'activité de la communauté en affichant régulièrement les nouvelles, en mettant en avant les utilisateurs et contributeurs actifs et/ou méritants. Lorsqu'un nouveau membre s'inscrit sur la liste de diffusion, sur le forum ou sur le site web c'est une bonne idée de le mentionner en première page, pour éventuellement le présenter et le remercier. Si un contributeur propose un code, remerciez-le aussi

pour son travail. Un bon *community manager* est quelqu'un qui a compris que le temps passé à remercier et à féliciter est du temps gagné.

5.11 Homme-sandwich ?

L'un des auteurs de ce guide a pour habitude d'imprimer des tracts (*flyers*) lorsque qu'il participe à des conférences. Plusieurs types de tracts sont distribués en fonction de la conférence, soit pour promouvoir certains projets *open source* de l'Inria, soit pour promouvoir sa présentation. Cela permet aussi à l'auteur de discuter avec des personnes avec qui il n'aurait jamais osé parler. Le tract est un moyen (physique) d'entrer en contact avec d'autres personnes. Vous tendez le tract gentiment, expliquez la nature du projet et engagez une discussion – en somme l'homme sandwich n'est pas mort...

5.12 Projet *Open source* ou produit *Open source* ?

Il est plus compréhensible pour le grand public et industriels de parler de « produit » plutôt que de « projet open source », c'est comme ça ! Mais en fait cela dépend beaucoup de son rôle : le terme « projet » aura du sens pour le contributeur alors que parler de « produit » paraîtra naturel pour le simple utilisateur.

5.13 Qui utilise votre logiciel ?

Il est très utile de garder une trace des utilisateurs : qui sont-ils ? Issus du grand-public ? Des professionnels ? Des organisations open source ? Des étudiants ? Des enseignants ? Depuis quelles zones géographiques se connectent-ils ? N'hésitez pas à afficher ces informations sur votre site vitrine. De belles références ou une identification aux utilisateurs préexistants affirmeront l'image de votre logiciel et inciteront les visiteurs à le télécharger.

Attention à la collecte de ces informations : les participants à des communautés ne supporteront pas des méthodes trop intrusives et il faut éviter de créer des freins à la participation. Une bonne façon de faire est de

permettre de rester anonyme pour une large partie des actions, comme le téléchargement du code, la lecture des forums, mais de demander une inscription (avec vérification de courriel) pour être informé sur les mises à jour, poser une question ou avoir accès à une forme de *hotline*. Lors de l'inscription, ne posez pas trop de questions, laissez-les venir. Apprenez plutôt à connaître les utilisateurs principaux via leurs messages sur les forums ou lors d'interactions directes avec le *board*. Certains projets proposent une inscription depuis leur site vitrine, d'autres proposent une inscription au lancement du logiciel sur votre machine. Vous pouvez aussi proposer aux membres de la communauté de remplir un sondage sur la base du volontariat pour mieux connaître leurs profils et donc être en mesure de mieux répondre à leurs attentes.

Par ailleurs n'hésitez pas à analyser les logs de connexion, ils contiennent beaucoup d'informations utiles (distribution géographique des visiteurs, récupération des noms de domaine à partir d'un reverse IP, etc.).

Lorsque vous apprenez que votre logiciel est utilisé par un utilisateur prestigieux (telle grande entreprise ou telle université réputée) n'hésitez pas à créer une section *success stories* sur votre site vitrine pour diffuser l'information. Il arrive bien souvent qu'un visiteur passe à l'acte (il télécharge votre outil) parce qu'une grande entreprise, une université ou une communauté logicielle réputée mentionne utiliser votre outil. De même, si on en parle dans la presse ou sur les sites web importants, créez une section coupures de presse. N'hésitez pas non plus à afficher les concours que vous avez gagnés grâce à votre projet.

La mise en place d'un outil d'analyse de site web est très utile pour obtenir des informations sur les profils des visiteurs : depuis quel pays se connectent-ils ? Quelles sont les pages qu'ils visitent le plus ? Quel est leur parcours de navigation sur votre site ? Combien de temps passent-ils sur chacune des pages ? Quelle est la fréquence de leurs visites ? Comment, et dans quel sens, évolue le nombre de visites ? etc.

À ce stade, vous avez défini et mis en place une architecture de participation, publié vos codes, et diffusé la bonne parole (l'existence de votre projet) dans l'ensemble de vos réseaux de contacts, à travers une campagne de publipostage (*mailing*), des messages publiés sur différents forums et réseaux sociaux, des articles publiés dans la presse et/ou dans des

revues scientifiques, etc. Parmi ceux qui auront lu votre message, certains vont être convaincus par le concept et les valeurs du projet. Ils visiteront votre site et téléchargeront éventuellement votre code. Certains iront plus loin, poseront des questions, proposeront des collaborations, des idées, des améliorations. Il faudra savoir comment les accueillir, comment répondre à leurs questions, comment les guider. Il faudra aussi leur proposer de travailler sur certains aspects du projet et les orienter vers des activités afin de donner du rythme à votre projet.

Même si vous avez le virus du développement il faudra vous résoudre à sortir la tête du code pour consacrer du temps à des tâches d'animation et d'encadrement. Cela peut paraître frustrant au départ mais vous découvrirez rapidement que le fait de passer du temps à superviser le développement fait par d'autres permettra à votre projet de passer à la vitesse supérieure en jouant sur l'effet de levier, ce qui pourra vous apporter de grandes satisfactions. Considérer ce type d'activité… de même que la communication… comme étant moins importantes que le temps passé à écrire ou récrire du code serait une erreur stratégique. Cela nous amène au chapitre suivant consacré à l'animation de la communauté.

6 Animer, entraîner

Good programmers make the others good programmers
Jim ZEMLIN (Linux Foundation Executive Director)

Lorsque la communauté est lancée et qu'une dynamique est amorcée, il faut tout faire pour la maintenir et la renforcer. L'animation d'une communauté est une tâche lourde qui nécessite une bonne organisation et qui suppose de respecter quelques principes. Le premier d'entre eux consiste à déléguer car, même si l'initiateur du projet y consacre tout son temps et son énergie, il ne pourra pas porter seul la communauté très longtemps. Cet effort doit être rapidement partagé et réparti. Les principes qui suivent montrent comment mettre en œuvre cette délégation aux autres membres de la communauté. L'objectif est de dépasser aussi rapidement que possible le périmètre de l'équipe principale ou du bureau exécutif s'il est déjà en place pour distribuer les tâches d'animation aussi largement que possible.

6.1 Les dix commandements du *community manager*

Les dix commandements de l'animation de communauté sont les suivants :

1. les gens d'abord !
2. raconter une histoire,
3. accueillir chaleureusement les nouveaux membres,
4. se rencontrer, sur le net mais aussi physiquement,
5. être transparent dans ses actions,
6. faire des retours positifs et réagir vite aux contributions,
7. toujours penser à attribuer les crédits (contributors.txt, *release notes*, etc.),
8. modérer les conflits, être un médiateur, agir en facilitateur, en catalyseur,
9. être le garant des règles de fonctionnement et des valeurs de la communauté,
10. ne pas hésiter à se remettre en question.

Toute action menée au sein de la communauté doit impérativement respecter ces dix commandements.

6.2 Rôle du *community manager*

Le rôle du *community manager* consiste à :

1. organiser les réunions et publier la restitution des discussions et des décisions,
2. aligner la vision entre les membres de la communauté,
3. vérifier que la feuille de route et la gestion de code fonctionnent, ne coincent pas,
4. agir en médiateur,

5. s'assurer, au besoin, que les projets et communautés connexes à l'écosystème mentionnent et utilisent correctement le nom du projet, le logo, etc.,
6. donner du rythme,
7. animer la liste de diffusion, le forum,
8. organiser (ou encourager) des initiatives permettant la promotion et le développement de la communauté et du code source,
9. motiver ses troupes,
10. donner de la reconnaissance aux membres qui contribuent,
11. veiller à ce que l'organisation du projet reste simple, avec peu de hiérarchie, peu de contrôle, beaucoup d'autonomie et de confiance.

Pour cela, cette personne doit avoir une attitude ouverte et accueillante, elle doit aussi avoir de solides connaissances en informatique, un grand sens de la diplomatie, de grandes qualités de rigueur et d'organisation.

6.3 La feuille de route

Une communauté doit se fixer une feuille de route car il est nécessaire que chacun comprenne où l'on va et comment on s'organise pour y aller. La feuille de route est un document de référence traitant à la fois des aspects techniques et organisationnels, elle doit être claire et fixer des objectifs réalisables compte tenu des moyens dont dispose le projet. Ces évolutions doivent se produire en toute transparence et respecter l'architecture de participation.

La feuille de route doit être ouverte à tous, elle doit susciter l'attention pour déclencher l'envie de participer. Pour cela elle propose des moyens d'agir aux membres les plus actifs.

Il est important de fournir un planning prévisionnel de diffusion des prochaines versions mineures et majeures. Il vaut mieux mentionner des dates cibles, quitte à émettre quelques réserves en indiquant par exemple qu'il s'agit d'un délai que l'on essaiera de tenir, plutôt que ne rien donner du tout. Le fait de présenter cela comme une sorte de défi motivera les utilisateurs et les rendra tolérants en cas de retard. Le fait de lancer des

appels à contributions très explicitement sur le site permet de convaincre les indécis et d'augmenter la participation au projet. Il faut bien sûr que l'ambition de la feuille de route soit en cohérence avec les ressources dont on dispose. Cela suppose que l'élaboration de la feuille de route soit faite par des développeurs suffisamment expérimentés pour évaluer son réalisme.

6.4 Donner du rythme

Un projet logiciel ne fonctionne correctement que s'il est constamment en mouvement. Il est donc souhaitable d'organiser régulièrement des événements comme des rencontres physiques ou en ligne, des journées, des assemblées générales ou encore des conférences téléphoniques avec les participants clés, etc. Il est également important de rendre compte en permanence de la vie quotidienne de la communauté : publier les comptes-rendus des réunions, proposer des webinaires, animer les forums, les alimenter à l'occasion avec des informations provenant de l'extérieur, etc. Certaines communautés enregistrent leurs réunions et les publient.

6.5 Comment gérer les forums et les listes de diffusion

La chose la plus importante est de tenir ces lieux rangés. Ensuite quelques règles simples permettent d'être plus efficace. Lors des questions récurrentes, même si la réponse est très simple, toujours renvoyer les utilisateurs chercher la réponse dans le forum avant de poser leur question. Il faut que cela devienne un réflexe. Ainsi aucun nouveau *thread* ne doit être ouvert avant de s'être assuré que le sujet n'a pas été déjà traité. Laisser faire cela est contre-productif car cela introduit de la confusion et fait baisser la valeur du forum. Idem pour les archives de listes de diffusion.

Il est important d'expliquer aux utilisateurs comment gérer leurs courriels par fils de discussion et ouvrir des nouveaux sujets plutôt que de poser de nouvelles questions dans un fil de discussion qui a été ouvert sur un sujet sans rapport. Sans quoi les lecteurs seront très vite perdus et la liste perdra de son utilité.

6.6 Penser à tous les publics

Ce rythme que l'on insuffle au projet ne doit oublier personne. Il faut bien penser à alterner des évènements techniques destinés aux contributeurs (*coding sprint*, *coding contest*, *barcamp*, etc.) et des évènements tournés vers les utilisateurs (formation, échange, club utilisateurs) ou vers le grand public (vulgarisation). Ces activités sont appréciées, elles renforcent la visibilité du projet, produisent du lien et renforcent les collaborations.

L'une des fonctions de la veille active déjà évoquée est de permettre de détecter si d'autres communautés ou projets existants pourraient être intéressés par l'utilisation de votre logiciel, par la ré-utilisation d'une partie de votre code, ou d'une coopération. Dans le cas où de telles communautés seraient identifiées, et notamment lorsque le budget du projet est restreint, le fait de contacter leurs leaders et d'explorer avec eux les possibilités de coopérations et de synergies est un moyen efficace d'élargir son réseau de contacts, d'augmenter la dynamique d'un projet, sa visibilité et le nombre de ses contributeurs.

6.7 Que dire aux contributeurs de code sur les bugs ?

Les rapports de bug sont des lettres d'amour qui décrivent avec tact et délicatesse les problèmes rencontrés. Les membres doivent être informés de cet état de fait dès leur arrivée. Il en est de même pour les outils d'intégration continue. Les rapports d'erreur émis par ces outils sont des moyens d'améliorer la communication entre un leader technique et un contributeur. Il ne faut pas se servir de ces outils pour juger le travail du développeur mais plutôt comme un moyen de discuter avec lui des méthodes d'ingénierie logicielle, de lui faire découvrir les bonnes pratiques de codage et de mise en production. Toute montée en compétence d'un contributeur se traduit par une montée en qualité du logiciel.

6.8 Savoir motiver ses troupes

Le principe qui met une personne en action est la motivation. Ce principe qui s'applique à toutes les activités humaines est donc aussi valable

pour les projets de développement de logiciel libre. Cela signifie qu'un contributeur à un projet, a fortiori si cette contribution est bénévole, ne s'engage dans des actions qui lui demandent des efforts que s'il est motivé, que s'il a une bonne raison pour le faire.

La motivation au sein d'un projet de logiciel libre est essentiellement de nature intrinsèque, c'est-à-dire interne à l'individu, comme la nécessité, la passion, le plaisir, le jeu, la reconnaissance, etc. L'autre forme de motivation est la motivation dite extrinsèque. Elle est le produit de facteurs extérieurs à l'individu, comme le salaire, la promesse d'une récompense, la menace, la soumission à l'autorité, le cadre législatif, etc. Il est facile de comprendre qu'étant basé sur le bénévolat, un projet de développement de logiciel libre repose essentiellement sur la motivation intrinsèque de ses contributeurs. L'une des raisons du succès du modèle de l'*open source* est que la motivation intrinsèque s'avère beaucoup plus efficiente que la motivation extrinsèque. Aussi a-t-on tout intérêt à exploiter au maximum ce facteur pour animer une communauté basée sur le bénévolat.

On comprend pourquoi le chapitre 1 insistait sur le fait que les valeurs jouent un rôle moteur pour les communautés : elles sont l'un des ressorts de la motivation intrinsèque. Concrètement, cela signifie que lorsque l'animateur d'une communauté cherche à gérer l'activité des contributeurs il faut qu'il commence par identifier leurs envies et leurs aspirations afin de leur distribuer des tâches qui déclencheront chez eux une motivation intrinsèque : il en fera des contributeurs productifs, fidèles, heureux, et le projet avancera beaucoup plus vite.

Malgré la nature essentiellement intrinsèque de la motivation chez les membres des communautés *open source*, la motivation extrinsèque peut très bien s'y rencontrer aussi. C'est le cas des développeurs rémunérés pour travailler sur le logiciel ou encore d'un acteur économique qui fait de votre logiciel un élément de sa stratégie. Lorsque vous devez sélectionner un nouveau collaborateur, assurez-vous que la motivation extrinsèque n'est pas la seule raison de sa présence dans le projet. Il est souvent préférable de sélectionner un collaborateur un peu moins pointu techniquement mais passionné plutôt que le contraire.

Évidemment, les deux sortes de motivations sont importantes dans une communauté, même si cette importance peut varier d'un projet à l'autre. Enfin, les motivations peuvent se combiner, la rémunération du travail

étant renforcée par la passion. De même il n'est pas rare de passer d'un type de motivation à l'autre : on voit souvent des développeurs continuer à contribuer à un code après la fin d'une mission rémunérée.

6.9 Maintenir son esprit ouvert

Il ne faut pas hésiter à consacrer des ressources à l'organisation de la connaissance dans la communauté en proposant des outils de partage comme les webinaires (séminaires en ligne), les ateliers, les cas d'utilisation, mais aussi :

— publier de la documentation sur l'utilisation du logiciel développé par la communauté,
— commenter le code dans la perspective d'un usage documentaire pour les développeurs qui interviendront ultérieurement sur le logiciel. En d'autres termes, faire en sorte que la lecture du code soit instructive,
— ne pas hésiter à franchir les barrières et à aller chercher les bonnes idées dans les autres communautés, comme cela se pratique beaucoup dans les communautés de jeux vidéo,
— adopter l'*open-attitude*, inviter les utilisateurs à faire des retours (« que pensez-vous de… ? »),
— solliciter des bêta-testeurs, fournir aux contributeurs des moyens de trouver facilement ce dont ils ont besoin : leur fournir des pages de liens bien documentées, leur pointer les pages où ils trouveront des réponses (le gestionnaire de bug, les forums, la foire aux questions, les téléchargements, etc.).

Vous pouvez aussi mettre en place des outils de veille automatique (Google Alerts, Yahoo Alerts, Framabee, inscription à des blogs et webzines, inscription à des flux RSS, etc.) afin de glaner des informations sur votre projet et des informations sur des domaines connexes.

6.10 Faciliter la participation

Faites en sorte que les processus de retour d'information (retour des utilisateurs sur le projet, le logiciel, un événement, etc) et de contribution

restent extrêmement simples. La liste suivante vous permettra de vérifier quelques points qu'il faut absolument rendre très simples d'accès :

— l'inscription,
— le rapport de bug,
— l'accès aux outils de développement et leur utilisation,
— la prise de contact avec vous,
— les propositions d'initiatives,
— les contributions au projet (quelle que soit la forme que ces contributions peuvent prendre),
— la réalisation ou la correction d'une traduction,
— la correction d'une erreur dans la documentation,
— la modification d'une entrée de FAQ,
— etc.

Enfin, le site web doit être intelligible pour des non-spécialistes. Pour cela il faut éviter de revenir sans cesse à la technique mais plutôt parler finalité, fonctionnalités, usages, bénéfices du projet.

D'autres facteurs permettent d'inciter les utilisateurs externes à contribuer activement : d'abord, leur accorder de la reconnaissance et partager la connaissance avec eux. Pour cela on peut afficher sur la page d'accueil les noms ou les pseudos des meilleurs correcteurs de bugs ou mentionner le nombre de posts à côté de la signature des auteurs sur le forum, etc. Dans la même veine, il arrive parfois que les membres d'une communauté proposent spontanément des fonds d'écrans et autres ressources graphiques à l'effigie du projet. En publiant ces artefacts sur votre site vitrine, d'autres personnes seront sûrement attirées à leur tour et participeront en soumettant des icônes, des fonds d'écran, etc. N'oubliez pas de les remercier et d'afficher sur votre site vitrine leurs travaux avec le nom des auteurs originaux (reconnaissance).

6.11 Liste des tâches faciles à faire

Réservez des tâches faciles pour les nouveaux arrivants. Publier cette liste sur le site permet d'attirer de nouveaux contributeurs et de les aider à avoir du succès lors de leurs premières contributions et donc de les inviter

à s'impliquer plus et à progresser pour qu'ils soient capables de traiter des problèmes de moins en moins triviaux.

Certains nouveaux arrivants proposeront plein de nouvelles idées. Ne les écartez pas, dirigez-les simplement vers la liste des tâches faciles puis, en fonction de leur implication et de leurs résultats, vous pourrez leur confier des tâches plus complexes, voire les amener à développer eux-mêmes les idées qu'ils proposaient.

L'équilibre entre la stabilisation de la communauté et la recherche de son extension n'est pas facile à mettre en œuvre. Cela passe par la promotion du projet qui, en augmentant sa visibilité permet de recruter de nouveaux membres mais en affichant une stabilité rassurante. Cela peut aussi passer par le lancement de collaborations avec d'autres organisations, mais à la condition de faire des choix pertinents.

La transformation de la documentation existante sous forme d'ouvrages papier, ou de documents numériques structurés et éventuellement téléchargeables, peut aussi être un vecteur de développement de la communauté. Des associations spécialisées dans l'écriture de manuels sur l'utilisation, la programmation du logiciel (ex. FLOSS Manuals) proposent des services de rédaction.

6.12 Les différents types de membres et leurs motivations pour participer au projet

Le *community manager* :

— aboutissement du projet (résolution du problème initial par la communauté),
— cohérence de la communauté et motivation des membres (efficience),
— dynamisme du projet (rayonnement, notoriété, etc.),
— influence sur le projet (pouvoir le piloter comme il l'entend).

L'adopteur (dont l'adopteur précoce) :

— possibilité de personnaliser et d'étendre,
— disposer d'un modèle de programmation facile à utiliser,
— avoir accès à des API fiables, ouvertes et documentées,

— ne pas être gêné par des barrières à l'entrée (ergonomie, conformité aux standards).

L'utilisateur :

— qualité du logiciel (stabilité, ergonomie, performances),
— qualité de la documentation (complète, ouverte, à jour),
— facilité à trouver, installer et utiliser le logiciel,
— accessibilité du support (disponible, modulable, efficace).

Le développeur :

— réaliser le rêve de participer en tant qu'acteur à un projet cool,
— rencontrer des barrières à l'entrée faibles et progressives,
— contribuer à un projet dont il partage les objectifs et les valeurs,
— sentir que les choses avancent vite et dans la bonne direction,
— contribuer à un projet qui le laisse exprimer ses talents.

6.13 Recruter de nouveaux membres

Lorsque le projet prend de l'ampleur, les opérations quotidiennes (administration système, corrections de bugs, mise en ligne de documentation, traductions, réponses aux questions, etc.) prennent de plus en plus de temps et il arrive un moment où elles doivent être déléguées. Il faut alors pouvoir s'appuyer sur des membres actifs et motivés. Cela passe par l'accroissement de la communauté.

À cette fin toutes sortes d'initiatives sont mises en œuvre : organisation d'évènements, concours de programmation, « code camps », « *project summit* », mise en place de structures plus pérennes, comme des clubs d'utilisateurs, des collaborations avec des écoles et universités, etc. La création de rôle d'ambassadeur peut être aussi un bon moyen de représenter le projet dans les régions ou des domaines difficiles à toucher autrement. Vous pouvez aussi créer des rôles au sein de votre structure de gouvernance, pour inciter certains membres à prendre en charge l'un des rôles (et responsabilités) proposés, ils pourront ainsi prendre part à la construction du projet et influer sur son avenir. Il faudra alors accompagner les nouveaux contributeurs en les formant et en leur faisant suivre un parcours d'apprentissage.

Un principe efficace pour accroître la communauté de contributeurs consiste à inverser la logique d'intérêt habituellement adoptée « Qu'est ce que ce contributeur peut nous apporter ? » Et se poser la question « Qu'est ce que le projet peut apporter à ce futur collaborateur ? ».

6.13.1 Comment attirer de nouveaux membres ?

— poser la question « qu'est-ce qui pourrait inciter une personne extérieure à rejoindre la communauté ? » et y répondre !
— créer des rôles,
— réserver (et publier) des tâches simples pour que les nouveaux arrivants puissent commencer à apprendre tout en résolvant des problèmes réels mais simples,
— trouver votre communauté et structurer votre projet afin qu'il apporte une valeur ajoutée à vos contributeurs,
— consacrer une partie du site vitrine afin d'exposer les dernières contributions et citer les contributeurs méritants,
— aller à la pêche aux nouveaux membres : arpenter les forums, proposer aux « posteurs » qui paraissent intéressants de rejoindre le groupe,
— faire connaître votre projet par tous les moyens possibles,
— transmettre cette mission à tous les membres de la communauté.

Les universités et les écoles sont des lieux à privilégier pour recruter de nouveaux membres : les élèves ou étudiants sont susceptibles de contribuer à une partie de votre logiciel dans le cadre de travaux personnels ou collectifs et, pourquoi pas, de devenir des membres actifs de la communauté.

6.13.2 Relancer une communauté dormante ?

Votre projet peut être un amplificateur ou un réactivateur d'une communauté ou d'un projet qui existe déjà. Vous pouvez bénéficier de leur potentiel et le renforcer en initiant de nouvelles collaborations.

6.14 Le *community manager* d'un projet de logiciel *open source* : un chef de projet nouvelle génération

Comme le montre le guide, le *community manager* doit disposer d'un large éventail de compétences : gouvernance, gestion de la propriété intellectuelle, connaissance des modèles économiques, conduite de projets de développement logiciel, aptitudes au management… Il aime travailler avec les autres, les motiver, emporter leur adhésion, susciter leur enthousiasme pour donner du rythme au projet. Il est à l'aise avec l'organisation, l'animation, la communication, la médiation. Il aime coder.

Le *community manager* que nous décrivons ici est en réalité un chef de projet à large spectre qui ne doit pas être confondu avec le *community manager* employé par les grandes entreprises qui se consacrent essentiellement à gérer des groupes/pages Facebook pour développer une communauté centrée sur un produit ou une marque. Il s'agit dans notre cas d'animer et de développer une communauté de pratique, ce qui est plus large et probablement plus riche aussi.

6.15 L'agenda du *community manager*

Il est primordial de bloquer dans votre agenda un créneau d'au moins une heure par jour pour les tâches de *community management*. Cela vous laisse largement du temps pour coder le reste de la journée. D'autres personnes préfèrent se bloquer deux demi-journées dans la semaine, l'important étant de consacrer assez de temps afin de réaliser « au mieux » les tâches d'animation.

À présent, nous allons expliquer dans le prochain chapitre pourquoi il est important de remettre en cause le projet et son management, notamment lorsque la communauté s'agrandit de manière significative.

7 Suivi et évolution

Code is not so important when users are in the loop
Vieux proverbe libriste

Lorsqu'un projet décolle, que ce soit dans le monde des communautés de logiciels ou dans d'autres domaines, il est facile de se faire déborder et de ne plus avoir le temps de prendre assez de recul. Pour éviter cet écueil, il faut mettre en place un monitoring de son projet. Ne pas oublier que la communauté s'est fondée sur un problème partagé et une volonté de le résoudre collectivement.

7.1 Est-ce qu'on est toujours bien dans le modèle ? Est-ce que l'on en dérive ?

Le problème qu'on résout est-il bien toujours celui qu'on s'était fixé ? Si ce n'est pas le cas, l'ajustement doit être répercuté dans l'ensemble des processus. La motivation des membres est-elle toujours bonne ? Le moteur de cette motivation est-il toujours le même ? Toutes ces réflexions peuvent nécessiter soit une remise en cause plus ou moins importante,

soit le passage du projet à une autre échelle si les difficultés viennent de la taille de la communauté.

Un processus de contrôle doit être mis en place pour évaluer les progrès et mesurer les effets des actions menées. Il doit être simple à mettre en œuvre et suffisamment informatif pour permettre de tirer des conclusions. Ce monitoring doit avoir deux axes : le code et la communauté.

7.2 Monitoring du code

Concernant le code, il existe à nouveau deux familles de métriques : celles qui s'intéressent à la qualité du code et celles qui concernent l'activité autour du code. Ce qui est communément admis est la nécessité de surveiller l'évolution de l'architecture du code, son design général ; il est alors possible de voir apparaître des effets spaghettis : des branches, des dépendances, trop de modules, trop de niveaux d'imbrication, etc. En cas de dérive importante, il est temps de faire une opération de ré-architecture.

L'interface utilisateur est un élément également important et qu'il convient de tester régulièrement. Une analyse des messages de nouveaux utilisateurs sur les forums permet également de vérifier qu'il ne se complexifie pas trop. Le *redesign* de l'interface utilisateur en cas de dérive est quant à lui à envisager avec précaution : il faut limiter les perturbations qui pourraient entraîner des départs d'utilisateurs vers d'autres logiciels.

Enfin, le dernier axe sur la qualité du code doit s'occuper du code lui-même : dénombrer les *commits*, demander une revue du code (éventuellement partielle) par une personne extérieure à l'équipe principale, vérifier que les *Coding Rules* sont bien appliquées, regarder l'historique des campagnes de tests (couverture et fréquence), la fréquence de rapport de bug. Ne pas oublier de s'intéresser à la documentation (guide utilisateurs, guide du développeur, etc.).

7.3 Suivi de la communauté

Concernant la communauté, les mesures quantitatives sont plus faciles à réaliser : nombre de téléchargements, nombre d'utilisateurs enregistrés

ou identifiés, nombre de contributeurs, nombre de messages sur les forums. Il est intéressant, et souvent assez simple, d'automatiser la récolte de ces mesures et la construction de tableaux de bord (certains outils d'animation le font partiellement). Il faut également penser à qualifier autant que possible sa communauté : répartition géographique, catégories socio-professionnelles et sûrement aussi quelques paramètres spécifiques au domaine adressé par le logiciel.

En complément, il est intéressant de faire un inventaire des aspects de la communication : nombre d'évènements (organisés et/ou dans lesquels on est intervenu), revue de presse. Ce dernier axe fait partie du travail de veille que la méthode vous a poussé à mettre en place.

Concernant la communauté, il existe un aspect beaucoup plus subjectif qui n'est pas facile à appréhender : l'état d'esprit de la communauté. On trouve cela dans le contenu des échanges au sein de la communauté. Quelques exemples :

— les membres les plus actifs sont fortement engagés et fiables ;
— la communauté comporte un ou plusieurs leaders ;
— la participation des membres est récurrente et soutenue ;
— il règne un climat de confiance, de reconnaissance et d'écoute ;
— il y a un foisonnement d'actions et un rythme soutenu ;
— les relations et collaborations débordent largement le cadre du logiciel et de la communauté ;
— il existe un sentiment de neutralité, d'impartialité, au sein du projet (qui se ressent aussi depuis l'extérieur).

7.4 Gestion de crise

Il faut sensibiliser l'équipe principale au risque de conflits, de divergence dans la vision, les valeurs, les choix stratégiques et la feuille de route. Sans toutefois occulter les difficultés qui peuvent survenir au cours du projet, même quand on suit les bonnes pratiques. Des désaccords peuvent survenir dans la communauté. Les projets peuvent *forker* (souvent à cause de divergences sur les valeurs) avec une scission de la communauté, ou simplement mourir lorsque la crise fait se désintégrer la communauté.

7.5 Lorsque la communauté s'agrandit

À partir d'une petite centaine de membres, la gestion de la communauté et particulièrement son animation devient difficile à réaliser comme une activité en marge de son activité professionnelle. Tous ceux qui ont eu l'occasion d'animer une communauté en croissance forte savent qu'il arrive un moment où il n'est plus possible d'accompagner cette croissance sans un changement d'organisation.

Il est alors nécessaire de renforcer le principe de délégation au sein de son architecture de participation. Ce passage du « faire » au « faire-faire », ou de l'action au management, peut présenter des difficultés : impression de perte de contrôle, frustration de ne plus être au cœur de la production, de ne plus pratiquer son activité préférée (le développement !)…

Mais le jeu en vaut la chandelle car un responsable de projet qui a assimilé les bases du management et de la délégation, peut accéder à la vitesse supérieure. Cela ne l'empêche en rien de conserver une activité directement productive. Initialement, la délégation au sein d'une communauté est souvent basée sur le modèle du *leadership*. C'est naturel : au départ le noyau dur est généralement limité, il comprend le créateur du logiciel et une ou deux autres personnes. Il est tout à fait possible de rester sur le mode du leadership, mais il en existe d'autres, notamment ceux de la méritocratie et du consensus.

La méthode du *leadership* consiste à gérer le projet et la communauté de manière directive. Le leader exerce son autorité pour prendre les décisions, définir les objectifs, la feuille de route, etc. Ce mode de gouvernance présente une rapidité dans la prise de décision. Par contre, comme tout repose sur une personne ou un tout petit groupe il y a un risque d'aveuglement. Par ailleurs cela ne convient pas à tous les contributeurs : certains apprécient de travailler de façon très encadrée mais les caractères indépendants risquent de se sentir enfermés.

La méthode par méritocratie est, comme son nom l'indique, basée sur le mérite. Les personnes reconnues compétentes, fiables et engagées se voient affecter des rôles et des responsabilités plus importants. Ces personnes sont souvent élues par vote. La communauté fait alors confiance à ses représentants. Un inconvénient est le risque de trop hiérarchiser l'organisation.

La troisième méthode est basée sur le débat et la discussion. Les membres de la communauté échangent leurs points de vue sur un sujet et tentent de tomber d'accord avant de prendre une décision. L'avantage est de pouvoir explorer différentes pistes, avoir différents points de vue, mais surtout, une fois la discussion terminée et la décision prise, cette dernière sera plus facilement acceptée par le groupe. Le *leader* joue le rôle de médiateur et synthétise les échanges puis, une fois les débats terminés, il peut proposer de mettre la décision au vote. L'avantage de ce mode de fonctionnement est l'acceptation de la décision par le plus grand nombre. Si l'on parvient à la prendre, car le gros inconvénient de cette méthode de gouvernance, est d'être longue, usante et vite paraître stérile si les débats s'éternisent. Les décisions obtenues sont le plus petit dénominateur commun.

L'expérience montre que l'architecture de gouvernance qui produit la meilleure efficience est celle qui se base sur la confiance et qui évite les empilements hiérarchiques.

7.6 Prendre son autonomie

Enfin, il peut arriver un moment où l'activité est telle que la délégation n'est plus suffisante. Il existe un critère simple pour constater qu'on est arrivé à ce point de décision : le moment où il devient nécessaire de déléguer non plus les actions opérationnelles mais le management lui-même.

Il est alors possible de construire un modèle économique et d'appuyer la communauté sur une structure légale, comme une association ou une société anonyme.

Lorsqu'un projet qui se développe dans un environnement de recherche grossit, il arrive un moment où l'implication de l'équipe dans le développement d'un logiciel libre arrivé à maturité devient difficile à justifier. Le lien avec la recherche peut devenir ténu. Par ailleurs, si la finalité du projet n'est pas commerciale, des questions se posent : quand faut-il s'arrêter ? comment se désengager en douceur ? Ces points dépassent largement le cadre de ce guide centré sur l'animation d'une communauté.

Mais que votre projet soit adossé à un institut de recherche, une université, une association ou qu'il soit devenu une entreprise, la suite présente les points à surveiller.

7.6.1 Neutralité

Il est important de rester aussi neutre que possible quant à la composition du *board* pour que les contributeurs n'aient pas l'impression de travailler pour un groupe particulier ou pour une entreprise. C'est un point important qu'il ne faut pas négliger, au risque de voir la motivation des contributeurs (bénévoles) fondre comme la neige au soleil.

7.6.2 Stratégie

Passer sur un mode de management stratégique, c'est appuyer les décisions non plus sur des opportunités conjoncturelles, comme c'est souvent le cas au départ, mais plutôt en suivant les orientations définies initialement et en s'en tenant aux jalons prévus dans le cadre de la feuille de route du projet. Cela s'applique notamment au choix des collaborations et des partenariats. Cette façon de fonctionner suppose d'aligner ses décisions et ses actions sur la vision à long terme et d'apprendre à dire non même face à des propositions qui peuvent paraître alléchantes dans un futur immédiat.

7.6.3 Communication

Renforcer et professionnaliser la communication : il ne faut surtout pas négliger les contacts de presse, l'identité graphique, les campagnes de publicité, les interventions lors de colloques, ateliers ou conférences, qui deviennent des outils centraux pour le développement du projet.

On peut aussi imaginer la création de *goodies* à l'effigie du projet (T-shirt, tasses, peluches, etc.) : cela permet aux membres d'afficher visiblement, voire de revendiquer, leur appartenance au projet, de communiquer sur le logiciel et accessoirement d'alimenter la cagnotte du projet.

Enfin, vous pouvez créer ou faire créer des bannières, des fonds d'écrans et icônes à l'effigie du projet. Certains membres de la communauté les

utiliseront et publieront probablement à leur tour sur leur blogs. N'oubliez pas que les contributions peuvent revêtir toutes sortes de natures et qu'elles ne se cantonnent pas au développement logiciel : graphisme, ergonomie, traduction, etc.

Et pour affirmer la transparence du management de votre projet, vous pouvez aussi enregistrer certaines de vos réunions et les publier sur la toile.

7.6.4 Ressources financières

Lorsque la taille d'un projet augmente, ses besoins humains et logistiques s'accroissent proportionnellement. Il existe mille et une façons d'obtenir des ressources, que ce soit en trouvant des sponsors, en proposant des *goodies*, en vendant du service ou même en lançant des opérations de *crowdfunding* pour obtenir rapidement un coup de *boost* financier. Les aides apportées en nature (partage de stand, salons, prise en charge de déplacements, mise à disposition de développeurs, etc.) sont très intéressantes car elles permettent de se soustraire à toutes sortes de contraintes.

7.6.5 Standards

Plus un code est utilisé, plus il est important qu'il respecte les standards, notamment pour mettre le logiciel à la portée du plus grand nombre. De même une attention particulière doit être portée aux API qui facilitent l'insertion du projet dans l'écosystème du logiciel libre.

7.6.6 Utilisation du nom du projet par vos communautés amies

Lorsque la communauté s'agrandit, des projets connexes peuvent apparaître, basés sur vos résultats. Il est alors important que le *community manager* vérifie que des informations sur votre projet (liens croisés) apparaissent sur ces sites et lettres d'informations. Il faut s'assurer que les projets et communautés connexes à votre écosystème mentionnent et utilisent correctement le nom de votre projet, votre logo, etc. C'est une raison qui justifie l'importance déjà évoquée de déposer le nom de votre projet en tant que marque et de déposer aussi les principaux noms de domaines

(que ce soient des noms de domaine sur le web mais aussi sur les forges, réseaux sociaux, etc.). Ce contrôle sur la marque peut ensuite servir de monnaie d'échange. En effet, vous pouvez en retour proposer à vos partenaires d'utiliser votre réseau afin de publier des informations sur leurs propres activités.

7.6.7 Internationalisation

Lorsque le nombre de ses membres s'accroît significativement et que la visibilité du projet commence à franchir les frontières, on constate que la communauté devient progressivement multiculturelle. Il est alors temps de s'occuper activement de l'internationalisation du logiciel. Cela se fait en le dotant d'outils facilitant la traduction de l'interface graphique, des menus d'aide, de la documentation, etc.

Ces tâches seront grandement facilitées si l'architecture du logiciel a été prévue dès le départ pour cela. Il faut notamment s'assurer que les membres de la communauté d'utilisateurs peuvent facilement assurer la traduction de ces interfaces sans avoir de compétences particulières en informatique. Cela permet, en augmentant considérablement le vivier des traducteurs potentiels, de multiplier le nombre de versions linguistiques et d'augmenter significativement la qualité et la tenue à jour des traductions. Dans le même ordre d'idée il est souhaitable de disposer le plus tôt possible – si possible dès l'origine – d'une version en anglais qui sera accessible à un grand nombre de contributeurs potentiels.

Internationaliser un logiciel augmente fortement sa diffusion à travers le monde. N'hésitez pas à solliciter les contributeurs pour réaliser ces traductions. Lancez pour cela des appels et insistez sur la simplicité et l'ergonomie de l'activité de traduction. Ouvrez grande la porte à la communauté d'utilisateurs, invitez-les à contribuer, montrez-leur qu'il est facile de participer, que c'est à leur portée et que leur implication sera très appréciée. Pour cela remerciez-les ostensiblement dans les fils de news du projet, faites apparaître leur nom dans le générique ou dans la fenêtre « à propos ». Pensez aussi à traduire l'ensemble de l'habitat numérique du projet, forums, wiki, lettre d'information, etc. Si la communauté est de taille modeste et que la réalisation de multiples versions linguistiques de

tout l'habitat numérique paraît trop lourde, privilégier l'anglais est un bon calcul.

Une fois la masse critique atteinte, et à ce moment-là seulement, il devient pertinent de segmenter les forums par pays et par langue. Dans ce cas, les points importants (les « meilleures réponses » aux questions les plus fréquemment posées) pourront être traduits et reversés dans la version anglaise. Une solution intermédiaire consiste à proposer dans un premier temps une version en anglais et une autre dans la langue majoritaire de l'équipe principale, qui est généralement celle du pays dans lequel le projet a vu le jour.

7.6.8 Renforcement de la gouvernance

Lorsque la communauté s'accroît on ne peut plus se contenter d'une délégation informelle. Il faut formaliser les choses, attribuer des rôles aux membres (ex. modérateur de forum, webmestre du site vitrine, architecte des greffons), distribuer des rôles clairement définis dans l'architecture de participation. Il faut toujours accorder une contrepartie, ou une reconnaissance, face à l'engagement des membres et aux responsabilités qu'ils assument. Cela peut être un droit de vote pour les arbitrages techniques, la participation à des groupes de travail chargés de faire des choix (d'outils ou autre), des invitations à se déplacer sur des évènements, etc.

7.6.9 Mort d'une communauté

Le document aborde essentiellement la naissance et la vie d'un projet de développement logiciel libre. Mais les projets ont comme toute chose un cycle de vie et donc peuvent mourir. Souvent, la mort d'un projet est suivi d'une renaissance (à travers un *fork*, une reprise partielle du code, un changement de cible, un changement de gouvernance, le portage vers une nouvelle technologie, etc.) mais ceci est une autre histoire et sort du cadre de ce guide.

7.7 Exemple de méthode de suivi et indicateurs

Nous terminons ce chapitre avec un exemple de méthode de suivi et une liste d'indicateurs.

7.7.1 Méthode « classique »

1. identifier 3 objectifs ;
2. identifier 3 mesures pour chacun de ces objectifs ;
3. identifier les moyens afin de suivre ces indicateurs ;
4. mettre en œuvre le suivi ;
5. analyser les mesures prises ;
6. identifier les actions qui fonctionnent et celles qui ne fonctionnent pas ;
7. définir les actions correctives et / ou les améliorations possibles ;
8. re-boucler (re-exécuter, re-mesurer. . .).

7.7.2 Suivi du code

— mettre en place des campagnes automatisées de tests unitaires, de tests d'intégration et de tests de régression ;
— mesurer le nombre de *bugs* ouverts, fermés, le temps moyen pour résoudre un *bug*, le temps entre deux ouvertures de *bug*, le ratio entre les *bugs* corrigés et les *bugs* identifiés, etc. ;
— mesurer le nombre de demandes de modifications et d'extensions ;
— mesurer le nombre de téléchargements, le nombre de projets et d'organismes qui utilisent votre code ;
— mesurer aussi l'avancement des ressources associées à la base de code, telles que l'écriture de la documentation, etc.

7.7.3 Suivi des campagnes de promotion

— nombre de visiteurs nouveaux ;

— nombre de messages et de personnes qui parlent de votre projet sur les réseaux sociaux ;
— nombre de téléchargements supplémentaires ;
— nombre de messages supplémentaires sur la liste de diffusion ;
— nombre d'inscrits supplémentaires ;
— nombre d'articles/blog post qui mentionnent votre projet ;
— position et progression du référencement sur les moteurs de recherche. Est-il parmi les 4 premiers résultats ? ;
— etc.

7.7.4 Suivi des événements

Pour mesurer le « succès » de vos initiatives telles que la mise en place d'un atelier ou d'une conférence, vous pouvez *faire un sondage auprès des participants*, vous pouvez aussi utiliser les indicateurs suivants (source : Joe Maeda, *lawsofsimplicity.com*) :

1. quel est le retour sur l'image du projet ?
2. quel est le retour en termes d'attention ?
3. quel est le retour en termes d'émotion ?
4. quel est le retour en termes de réputation ?
5. quel est le retour en termes de « réseautage » ?
6. quel est le retour en termes de collaboration ?
7. quel est le retour en termes d'engagement ?

7.7.5 Suivi habitat numérique

Pour mesurer, de manière plus analytique, la dynamique de la communauté et de son habitat numérique, voici une liste d'indicateurs, quelque peu modifiée (source : Donna L. Hoffman and Marek Fodor, « Can You Measure the ROI of Your Social Media Marketing? », *MITSloan*, 2010) :

— **Outil Blogs / Site Vitrine**
  — nombre de visiteurs uniques ;
  — nombre de revisites ;
  — nombre de marque-pages ;
  — classement dans les moteurs ;
  — nombre de membres ;
  — nombre d'abonnés au fil RSS ;
  — nombre de commentaires ;
  — total des *user-generated content* (contenu généré par les visiteurs) ;
  — temps passé sur le site ;
  — nombre de réponses aux enquêtes, concours, etc. ;
  — taux de rebond ;
  — nombre de *backlinks* (liens croisés et articles repris sur d'autres sites) ;
  — nombre de partages d'images et de vidéos ;
  — nombre de citations ;
  — nombre de « J'aime » ;
  — nombre de commentaires ;
  — nombre d'abonnés ;
  — nombre d'*embeds* ;
  — nombre de liens entrants.
— **Commentaires sur le projet**
  — nombre de commentaires postés ;
  — aspect positif/négatif des évaluations ;
  — aspect positif/négatif des réponses ;
  — longueur des commentaires ;
  — pertinence des commentaires ;
  — utilité ressentie des commentaires par les autres utilisateurs ;
  — nombre d'ajouts ;
  — aspect positif/négatif des évaluations ;
  — aspect positif/négatif des réponses ;
  — nombre d'inclusions dans une liste.
— **Forums**
  — nombre de pages vues ;
  — nombre de visites ;

  - nombre d'utilisateurs répondant aux utilisateurs ;
  - aspect positif/négatif des posts ;
  - nombre de sujets ;
  - nombre de réponses ;
  - nombre d'inscriptions ;
  - nombre de *backlinks* ;
  - citations sur d'autres sites ;
  - tag sur des sites de partage de favoris ;
  - nombre de « J'aime ».
- **Co-création**
  - nombre de visites ;
  - nombre de créations
  - nombre de partage des créations.
- **Outils réseaux sociaux (Twitter et autres)**
  - nombre de tweets à propos du projet, de la marque, etc. ;
  - nombre de suiveurs ;
  - aspect positif/négatif des tweets ;
  - nombre de réponses ;
  - nombre de listes ;
  - nombre de *retweet* (RT) ;
  - nombre de « J'aime » ;
  - nombre de partages dans d'autres médias sociaux.

7.7.6 Suivi de la propriété intellectuelle

Voici une méthode permettant de suivre au mieux la propriété intellectuelle des ressources associées à votre projet :

1. faites une analyse de l'existant avec les membres de l'équipe (sans l'aide d'outils informatiques, ou presque)
2. identifiez et listez les composants internes au projet (développés par l'équipe) que vous utilisez :
   - quels sont les types de licences associées ?
   - est-ce que ces codes sources sont inclus sans modifications ?
   - est-ce que ces codes sources sont modifiés pour ensuite être inclus ou réinjectés ?

3. identifiez et listez les composants externes que vous utilisez :
   — quelle(s) est/sont la/les licence(s) associée(s) ?
   — est-ce un lien vers une bibliothèque (*library*) ?
   — est-ce que ce sont des binaires indépendants qui communiquent entre eux ?
   — est-ce que ces codes sources sont inclus sans modifications ?
   — est-ce que ces codes sources sont modifiés pour ensuite être inclus ou réinjectés ?
4. identifiez et listez les composants dont le code est fermé (commercial) que vous utilisez :
   — quels sont les termes de la « licence » ?
5. *formalisez les résultats dans un document*
6. utilisez des outils de contrôle de licence automatique tels que HP Fossology, Open Source Licence Checker (OSLC), des scripts fait maison, etc.
7. *confrontez les deux analyses* (les résultats produits par l'équipe et ceux produits par la machine), identifiez les potentiels/les incompatibilités ou conflits et proposez des actions correctives si besoin.

N'hésitez pas à demander conseil et à faire relire ces travaux par des personnes compétentes dans le domaine de la propriété intellectuelle.

7.7.7 Suivi de la méthode en elle-même

Vous ne devez pas oublier de suivre la progression de votre projet de création et d'animation de communauté :

— définir le projet : où en êtes-vous ?
— définir des objectifs et valeurs du projet : où en êtes-vous ?
— réaliser un inventaire : où en êtes-vous ?
— analyser son environnement : où en êtes-vous ?
— analyser vos besoins en termes d'outillage…
— etc.

8 Résumons

Nous voilà parvenus au bout de notre tour d'horizon de la gestion des communautés. Nous n'avons pas la prétention d'avoir apporté des réponses à tous les cas de figure. Notre ambition était simplement de parcourir l'ensemble des questions posées par la création et le management des communautés notamment dans le domaine des projets de développement de logiciels libres et *open source*. Le modèle est transposable à d'autres types de projets, par exemple dans le domaine de l'*open hardware*.

Il ne convient pas forcément de mettre en application toutes les préconisations, mais d'adapter la méthode à la taille de votre projet et à l'ambition que vous lui donnez. Pour les projets de taille modeste, la définition des objectifs, des valeurs et l'analyse du contexte peut se faire en quelques jours. Par contre, la construction de son habitat numérique sera sûrement plus consommatrice de temps qu'il n'y paraît.

Rappelons les trois facteurs clés de succès d'un projet :

— *Préexistence d'un besoin partagé* – cette condition est un préalable à toute communauté de développement ;
— *Ouverture du code dès le lancement du projet* – ouvrir son code c'est lâcher du contrôle pour maximiser la diffusion (donc l'impact) du projet ;

— *Une communauté motivée* – la motivation des membres de la communauté doit être l'objet de toutes vos attentions.

Vous commencez par faire l'inventaire des ressources dont vous disposez, de quoi je pars, et la définition les objectifs principaux du projet, vers où je vais. Vous analysez ensuite votre environnement, ce qui se passe autour de vous, et établissez un état de l'art des écosystèmes relatifs au domaine du projet, le contexte.

Vous pouvez alors choisir les moyens techniques qui seront mis en œuvre pour collaborer avec l'équipe principale et avec les contributeurs externes potentiels, définir l'architecture de participation et commencer à bâtir l'habitat numérique du projet. Vous avez plusieurs possibilités pour cela : rejoindre une communauté qui propose une architecture de participation suffisante en accord avec les besoins et les valeurs du projet ou bien mettre en place son propre habitat numérique composé d'un site vitrine, d'une forge et des outils de partage et de communication nécessaires.

Vous vous occuperez ensuite de la promotion de l'existence du projet, en utilisant différents supports de communication et en adaptant les messages en fonction du type de média et des publics que vous cherchez à toucher.

Il faudra ensuite accueillir, animer, entretenir et faire grandir le projet et sa communauté. Tout au long du cycle de vie du logiciel, il vous faudra être agile, *release early release often* et ne pas avoir peur d'avancer le plus vite possible ; la rapidité est un atout inégalable, *the early bird will make the deal* dit-on dans la Silicon Valley.

Ne perdez jamais de vue les valeurs du projet : croire que l'intérêt immédiat passe avant le système de valeurs est une erreur classique mais souvent fatale pour les projets.

Enfin et surtout, il ne faut pas se laisser décourager par les difficultés rencontrées : l'amorçage et le développement d'une communauté demandent du temps. Le système GNU/Linux et sa communauté ne se sont pas construits en un jour…

9 Pour aller plus loin

Eric S. RAYMOND, *The Cathedral and the Bazaar*, 1997. http://catb.org.

Etienne WENGER, *Communities of practice : learning, meaning, an identity*. Cambridge : Cambridge University Press, 1998.

Etienne WENGER, R. McDermott and W. M. Snyder, *Cultivating Communities of Practice*. Harvard : Harvard Business Review Press, 1998.

Stan GARFIELD, *Implementing a Successful KM Programme*, London, Ark Group, 2007.

Lydia PINTSCHER (dir.), *Libres conseils. Ce que nous aurions aimé savoir avant de commencer*, Lyon, Framasoft/Framabook, 2013, Framabook.org.

9.1 Aspects juridiques

François PELLEGRINI et Sébastien CANEVET, *Droit des logiciels – Logiciels privatifs et logiciels libres*, Paris, PUF, 2013. Voir sur www.droitdeslogiciels.info.

Olivia FLIPO et Benjamin JEAN, *Guide Open Source. Réflexions sur la construction et le pilotage d'un projet Open Source*, 2010. Site internet : guideopensource.info.

Benjamin JEAN, *Option Libre. Du bon usage des licences libres*, Lyon : Framasoft/Framabook, 2011.

RECUEIL DE FICHES sur les licences libres par l'Inria. Inria : Acteur du logiciel libre et fondateur de l'IRILL. www.inria.fr.

VIDEOS en français sur le sujet sur devlog.cnrs.fr.

Dan SHEARER (MariaDB), *Make a good contributor license agreement*, fOSSa 2012, video sur framatube.org.

Ross Gardler et Rowan Wilson, *Contributor Licence Agreements*, OSS-Watch.

Tony GUNTHARP, *What is a CLA and why do I care ?*, clahub.com.

9.2 Gouvernance

THE COMMUNITY ROUND TABLE, *The community maturity model*. Site internet : community-roundtable.com.

Heather J. MEEKER, *The Open Source Alternative : Understanding Risks and Leveraging Opportunities*, Hoboken, Wiley, 2008.

NOISEBRIDGE, *The Noisebridge Manual* sur le wiki de noisebridge.net.

STATUTS DE L'ASSOCIATION FRANÇAISE DES UTILISATEURS DE PHP, sur afup.org.

OW2 TECHNOLOGY COUNCIL, sur ow2.org.

FIWARE FOUNDATION, sur fiware.org.

9.3 Gouvernance orientées « développement de code »

Elizabeth K. JOSEPH, « The benefits of building an open infrastructure », sur opensource.com

« Debian System Administrators », sur debian.org

« Puppet Labs. Come join the infra team! », sur jenkins.io

« ReleaseEngineering/PuppetAgain », sur wiki.mozilla.org

9.4 Développements logiciels

Nancy WHITE, Etienne Wenger and John D.Smith, *Digital Habitats. Stewarding Technology for Communities*, Portland, CPsquare, 2009. Ressource sur fullcirc.com.

Open source software development methodology by OSS Watch, 2007-2011. Voir www.oss-watch.ac.uk.

Karl FOGEL, *Produire du logiciel libre*, Lyon, Framasoft/Framabook, 2012.

Elizabeth K. JOSEPH, « The benefits of building an open infrastructure ». opensource.com, 2015.

Debian System Administrators dsa.debian.org.

JENKINS, *Come join the infra team !* jenkins.io.

ReleaseEngineering/PuppetAgain wiki.mozilla.org.

How to contribute to the Hydrogen Open Source project Hydrogen.popez.org.

9.5 Modèles économiques

François ELIE, *Économie du logiciel libre*, Paris, Eyrolles, 2009.

Chris ANDERSON, *Free ! Comment marche l'économie du gratuit*, Paris, Flammarion, 2014.

Camille PALOQUE-BERGES et Christophe MASUTTI, *Histoires et cultures du Libre. Des logiciels partagés aux licences échangées*, Lyon, Framasoft/Framabook, 2013.

9.6 Référencements logiciels et mesures

Régis VANSNICK, *ROI et présence sur les médias sociaux, outils d'évaluation*. g1site.com. 2011.

Donna L. HOFFMAN and Marek FODOR, « Can you Measure the ROI of Your Social Media Marketing ». *MITSLOAN Management Review*, 52(1). 2010.

9.7 Habitats numériques – outils internes Inria

— Ferme Wordpress pour votre site vitrine
— Ferme de Wiki
— Forge (code repository, issue tracker. . .) et outils d'intégration continue
— IRC Jabber
— Mailing-List
— Etherpad alike
— Photothèque
— Dépôt de fichier Transfer
— Ligne Audio et Visio conference
— et bien d'autres choses. . .

N'hésitez pas à contacter vos équipes de moyens informatiques locaux et nationaux.

A Cas concret : la suite logicielle AspireRFID

1.1 Définition du projet

Le projet Aspire vise à développer une suite logicielle ouverte facilitant le développement d'applications basées sur les technologies RFIDs. Ce projet a émergé à la suite d'une rencontre entre des chercheurs Inria (David Symplot-Rill, Nathalie Mitton de l'équipe FUN, basés au centre de recherche Inria de Lille) et des chercheurs de l'université Joseph Fourier (Didier Donsez, basé à Grenoble) en 2007.

L'idée a été soumise au programme Européen FP7 (un programme de financement par la Commission Européenne de projets de recherche et de développement technologique) et, comme cela se pratique souvent dans ce type de montage, d'autres partenaires sont venus se greffer au noyau principal et former le consortium Aspire. Le projet a été financé pendant 3 ans, de 2008 à 2011.

1.2 Le besoin et cibles identifiées

Les besoins sont de :

1. produire une base de codes ouverts autour des technologies RFID permettant de favoriser l'adoption des technologies d'identification sans fil. Cela librement, sans licences commerciales. Les utilisateurs ciblés étaient les petites, moyennes et grandes entreprises et les sociétés de services (SSII) ;
2. créer et animer une communauté autour du projet afin de pérenniser la base de code sur le long terme ;
3. diminuer les coûts de production de logiciels autour de cette technologie.

1.3 L'inventaire

Existe-t-il déjà un code logiciel ? quelle est la licence associée ? quel est l'architecture du code existant ? quels sont les autres ressources disponibles ? quels sont les moyens de communication dont dispose le projet ? cet inventaire nous a permis d'ajuster nos ambitions à nos moyens.

1.3.1 Architecture du logiciel

Il n'y a pas à proprement parler encore d'architecture, chaque partenaire a dans ses cartons des briques logicielles qui ne demandent qu'à être exploitées et assemblées pour former la future suite logicielle AspireRFID. Plusieurs réunions techniques sont mises en place afin de définir l'architecture en fonction des apports de chacun des membres et identifier les codes manquants qui restent à développer. L'objectif est d'obtenir une bibliothèque facile à maintenir et permettant l'ajout de greffons facilitant l'ajout de nouvelles fonctionnalités produites par la communauté.

1.3.2 Aspects juridiques

La majorité des codes préexistants sont soit non publiés, soit ont une licence ouverte mais, dans ce cas, il faudra plus tard passer un peu de temps sur l'analyse de l'existant pour formaliser les choses. Nous détaillerons ce point plus bas, dans le paragraphe « la propriété intellectuelle ».

1.3.3 Méthodes de production du logiciel

Chaque équipe dispose de sa méthode pour produire et maintenir du code. Là aussi des réunions sont organisées afin de définir les bonnes pratiques de développement du code. Des règles et un habitat numérique sont mis en place. La méthode *release early, release often* fait l'unanimité au sein du projet.

1.3.4 Documentation

Il n'existe pas ou peu de documentation... un inventaire des manques et des améliorations souhaitées est produit. Un plan d'action est défini et mis en place. Une liste des bonnes pratiques est rédigée et publiée (ex. : commenter le code de manière à produire plus facilement une documentation).

1.3.5 *Core team* et leader

L'identification des membres et leurs intérêts est simplifié par le fait que nous sommes face à un projet européen. Nous disposons donc déjà d'une liste des membres et de leurs rôles respectifs.

1.3.6 Aspects communication

Il n'y a rien, tout est à faire (création d'un logo, rédaction d'une charte graphique, modèles pour les présentations institutionnelles, etc).

1.4 Définition des objectifs

Ils s'expriment comme suit :

— produire une base de code ouverte, développer un standard, une suite logicielle composée d'une bibliothèque *middleware open source* de bas niveau et des outils haut niveau permettant d'accélérer la construction d'applications (IDE) autour des technologies RFIDs ;

— promouvoir l'existence de la suite logiciel ASPIRE auprès des petites, moyennes et grandes entreprises européennes ;
— créer une communauté dynamique autour des technologies RFID en veillant à amorcer une base de code capable de continuer à évoluer après la fin du financement du projet européen ;
— enrichir la base de code avec de nouvelles fonctions pour le bénéfice du plus grand nombre ;
— diminuer les coûts de production de logiciels autour de cette technologie.

À partir de ces 5 objectifs nous avons écrit le *Mission statement* suivant : « La suite logicielle AspireRFID a pour but de développer un standard basé sur un *middleware open source* léger, conforme aux standards existants, évolutif en termes d'échelle, respectueux de la vie privée et intégré, pour faciliter le développement, le déploiement et le management d'applications basées sur des RFID, des senseurs, des capteurs. »

1.5 Définition des valeurs

Voici quelques extraits des conversations échangées lors des premières réunions : « il faut faciliter l'accès aux technologies RFID aux moyennes et petites entreprises », « il faut rendre les technologies RFID accessibles au plus grand nombre », « il faut décloisonner un marché monopolisé par de gros acteurs qui imposent leur point de vue commercial »…

1.6 Le partage

Le but du projet est de diffuser largement l'utilisation et l'amélioration du code. Il est donc nécessaire pour l'équipe que la licence associée à la suite logicielle et à ses ressources ne constitue pas un frein à son adoption par les industriels.

« Les entreprises doivent pouvoir intégrer cette suite dans leur système d'information sans nécessairement redistribuer l'ensemble du code » ; « il n'est pas question non plus qu'un éditeur de logiciel ou une grande entreprise s'empare de la base de code, la modifie sans redistribuer ces améliorations ».

Il faut donc que le code soit ouvert avec une licence – donc des règles de jeux – qui permette la diffusion, l'utilisation, et incite à la re-contribution. Le développement de la bibliothèque devra aussi se baser sur une architecture de code modulaire permettant également de faciliter l'ajout de modules par la communauté. Enfin, il faut que la communauté puisse directement accéder – en lecture et écriture – à la documentation du projet, ainsi cette dernière pourra être facilement améliorée par un grand nombre de personnes externes.

1.7 La propriété intellectuelle

Nous avons regardé les codes et ressources qui existent chez les partenaires du projet, fait une liste des licences entrantes, puis regardé les règles de propriété intellectuelle de chacun des partenaires (vision « institutionnelle ») ; nous avons analysé les comptabilités et incompatibilités des codes et briques logicielles.

L'analyse a été effectuée en deux étapes, les différents membres de l'équipe ont rempli une fiche de renseignements sur la liste des codes et de leurs licences associées, la liste des codes provenant de l'extérieur et de leurs licences associées. Nous avons aussi demandé la relation entre le code écrit par l'équipe et les librairies externes afin de mieux cerner d'éventuelles incompatibilités. Nous avons utilisé des outils automatisés d'analyse de code et de listage de l'ensemble des codes et de leurs licences associées (ces outils étaient OSLC et HP Fossology). Nous avons croisé les résultats obtenus par les formulaires complétés par les membres du projet avec les résultats produits par les analyseurs de code afin de relever d'éventuels manques ou incohérences.

Le projet partait de peu de choses, du code provenant principalement de l'Université Joseph Fourier de Grenoble, Inria et l'université d'Athènes. L'analyse fut au final plus rapide que prévu.

Il nous est apparu qu'une licence de type LGPL V2.0 serait une bonne réponse face au couple objectifs/valeurs pour être ouvert aux industries – risque de propriété intellectuelle réduit – et à la fois protéger le code d'un éventuel « détournement » par les grands acteurs du domaine.

Enfin nous avons choisi une licence Creative Common CC-BY-SA pour l'ensemble des images, vidéos, logos et documents produits par l'équipe. Cette licence donne des droits : la modification et l'amélioration de ces ressources, leur valorisation économique (ex. : produire et vendre un livre). Mais elle donne aussi des obligations : citer les auteurs originaux et redistribuer ces ressources sous les mêmes conditions de licence.

Nous n'avons pas acheté de nom de domaine car le projet est hébergé par le consortium OW2 qui attribue des noms de domaine sous la forme nomdeprojet.ow2.org.

La prochaine section présente l'analyse du contexte qui nous a amenés à prendre la décision de rejoindre une communauté déjà existante plutôt que de tout créer *ex nihilo*.

1.8 Analyse du contexte

Nous avons identifié les différents domaines d'applications, les noms des principaux acteurs, établi une liste des conférences ainsi qu'une liste de consortiums et organismes publics qui gravitent autour du sujet. Enfin, nous avons aussi identifié les projets similaires (ex. : projet *open source* Accada). Nous avons demandé à chacun des membres du consortium de fournir une liste de contacts dont la suite logiciel AspireRFID dépend, puis nous leur avons demandé de nous donner entre 2 et 4 contacts pour chacune des catégories suivantes :

— liste des communautés,
— liste des organisations,
— liste des codes et librairies,
— liste des sites web (blogs, webzines),
— liste des initiatives (nationales, internationales),
— liste des conférences.

Nous en avons fait de même avec les contacts potentiellement intéressés par l'application. Tout ceci a été documenté et partagé sur un wiki.

Lors de cette phase de veille, nous avons identifié certains projets et communautés *open source* pouvant être intéressés par la librairie ASPIRE. Notre attention a été particulièrement attirée par la communauté *open source* OW2.

Nous avons approché les leaders de la communauté OW2 et leur avons présenté le projet ASPIRE. Très vite nous avons pu constater une adéquation positive : d'un côté le projet ASPIRE visait à développer une plateforme logicielles *open source* tournée vers les RFID et, de l'autre, une nouvelle initiative de développement logiciel concernant les capteurs était soutenue par le consortium OW2. Le premier démarrait alors que le second cherchait de nouvelles contributions. Il s'agissait d'un accord parfait qui bénéficiait mutuellement aux deux composantes.

Nous sommes également tombés sur un autre point d'accord important. Nous avons demandé aux membres de l'équipe la liste de leurs besoins en termes de collaboration et de partage. Nous avons émis une liste de besoins et une liste d'outils (site web, gestionnaire de code, liste de diffusion, *bug tracker*, etc.) et avons constaté que nos besoins collaient parfaitement à l'architecture de participation proposée par OW2 (wiki, *webinar*, *mailing list*, forge, *bug tracker*, serveur de téléchargement, participation et organisation d'événements autour du *middleware*, etc.).

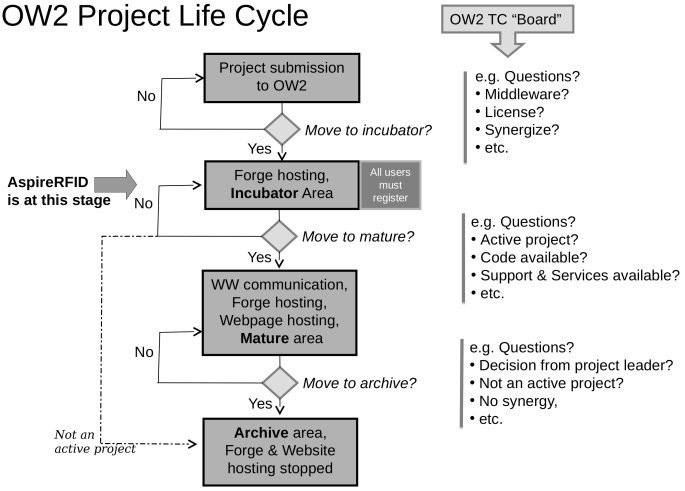
Lors de l'analyse du contexte, nous avons aussi identifié les synergies potentielles avec le consortium PICOM (Pôle des industries du commerce, Lille, France). Nous détaillerons ce point dans les sections « suivi de projet » et « résultats ».

Cette veille s'est achevée par une réunion de *brainstorming* afin de présenter l'analyse du contexte. Nous avons revu les objectifs, le concept, les besoins et avons discuté de la possibilité de rejoindre la communauté OW2 plutôt que de partir d'une feuille blanche. Au vu des objectifs de la communauté OW2, de ses valeurs, de sa taille (plus de 4 000 membres), des services de collaboration proposés, de son habitat numérique attractif, la décision a été prise de rejoindre OW2.

Cette analyse du contexte nous a donc permis de découvrir que le consortium OW2 pouvait nous aider à créer la communauté AspireRFID en nous faisant bénéficier de tous les outils dont avait besoin la communauté Aspire pour se lancer. Précisons que cette compatibilité s'étendait aux licences *open source*, les règles de propriété intellectuelle du consortium OW2 étant tout à fait en accord avec nos besoins.

1.9 Définition de l'habitat numérique et mise en place

Nous avons contacté les membres du comité technologique d'OW2 (comité regroupant essentiellement les chefs de projets de la communauté OW2 et qui ont - entre autres - la responsabilité d'accepter ou non d'accueillir de nouveaux projets au sein de leur habitat numérique). Après un vote positif, le projet Aspire a rejoint la communauté OW2 et a ainsi pu bénéficier de son infrastructure.



Exemple de document de travail tel que nous avions fait pour suivre le projet Aspire quant à son adhésion au consortium OW2

Les membres de la communauté Aspire ont décidé que le site vitrine du projet serait statique et sobre, basé sur une technologie de type Wiki. Cela s'est fait ainsi car les attentes des chercheurs ne sont pas les mêmes que celles du grand public, friand de sites web dynamiques. Nous ne détaillons pas cette phase plus en détail car elle est décrite dans le chapitre 4. Par ailleurs un exemple plus approfondi est décrit dans le cas concret numéro 2 (Appendice B).

1.10 Promotion

Grâce aux actions de promotion spécifiques telles que les publications dans des revues scientifiques et généralistes (que nous avions identifiées dans les premières étapes de la méthode), à travers la liste de diffusion de la communauté OW2, par la prise de contact avec des organisateurs de conférences traitant des technologies RFID, nous avons obtenu des résultats très rapidement : le projet a commencé à gagner en visibilité (augmentation significative du nombre de publications, nombre de blog post, nombre de téléchargements) et des moyennes et grandes entreprises ont commencé à nous contacter (ex. : prise de contacts avec PICOM, CNRFID, etc.).

1.11 Animation

Grâce au travail de promotion, nous avons pu organiser toutes sortes d'animations :

— des « code camps » dans plusieurs universités ;
— un concours de programmation autour de la technologie RFID (avec à la clé des kits Arduino et Tigitags, ce qui ne représente pas un gros investissement en termes de budget) ;
— une conférence (grâce à la communauté OW2) dont nous avons profité pour lancer un club d'utilisateurs et faire une assemblée générale, et à laquelle nous avons invité nos coopétiteurs.

Ces animations, qui ont touché beaucoup de monde, nous ont permis d'être rejoints par un nombre important de nouveaux membres. Nous avons envoyé à intervalles réguliers quelques lettres d'informations et répondu aux sollicitations qui nous arrivaient (corrections de bugs, améliorations, etc) via la liste de diffusion.

1.12 Suivi du projet

Le succès de la dissémination et de la promotion a renforcé les produits de l'exploitation, par exemple un benchmark RFID réalisé par le consor-

tium industriel PICOM. La plateforme logicielle AspireRFID fut comparée à des produits commerciaux et, en dépit du fait que la suite logicielle ne soit pas encore achevée et que les outils logiciels soient encore en cours d'élaboration, AspireRFID fut classée devant tous ses compétiteurs.

Dès lors, le consortium PICOM conseilla à ses membres d'utiliser la suite logicielle AspireRFID et, au-delà des qualités de cette dernière, PICOM nous a indiqué qu'ils avaient trouvés sur le site web du projet toutes les informations nécessaires à leur prise de décision : des informations sur l'organisation, sur le rôle de chacun, les objectifs et les valeurs du projet. La mention des valeurs véhiculées par le projet (permettre à des SSII d'accéder à moindre coût et sans barrière aux technologies RFIDs) a été très appréciée. L'utilisation d'une licence permettant à la fois de protéger la base de code libre tout en offrant la possibilité de rajouter des briques logicielles privées n'a fait que conforter leur choix car, selon leur analyse, cela limitait grandement le risque de conflits sur la propriété intellectuelle. On peut retenir de cela que la transparence et l'ouverture apportent des bénéfices. Publier ses valeurs sur le web est un bon moyen d'attirer le public.

Nous avons bien sûr mesuré l'audience de notre site web et le nombre de téléchargements, une carte du monde apparaît sur le site web du projet, nous avons aussi utilisé le service openhub.net pour le suivi de la communauté des développeurs.

1.13 Résultats

Le site vitrine a été parcouru par 20 000 visiteurs uniques entre 2010 et 2013, soit 6 visites uniques nouvelles par jour issues de plus de 70 pays. Ces visiteurs sont essentiellement des développeurs de logiciel et des ingénieurs de sociétés de services informatique.

500 000 lignes de codes ont été écrites dans le cadre du projet et nous avons augmenté le nombre de contributeurs de manière conséquente : de 4 développeurs en 2008 le projet en comptait 35 en 2013. Ces contributeurs sont essentiellement des chercheurs, des thésards et des ingénieurs de recherche.

La librairie est considérée comme stable depuis 2013. Il y a ce jour moins de *commits* mais cela n'empêche pas le nombre d'usages de gran-

dir : lorsque vous allez dans un centre commercial, dans un magasin de sport ou dans un supermarché, et que vous utilisez une douchette ou une caisse libre service, il y a de grandes chances pour que vous utilisiez la librairie Aspire.

Enfin, pendant que nous écrivons ces quelques lignes, plusieurs projets de financement autour du projet sont en cours d'élaboration. Il est important pour les groupes utilisateurs – majoritairement industriels – de la suite logicielle Aspire de continuer à développer la base de code et ses usages.

Cas concret : la plateforme robotique Poppy

Les auteurs de ce guide ont participé à la naissance et au suivi du projet Poppy qui constitue un cas illustratif de gestion de communauté depuis la définition du projet à la propriété intellectuelle en passant par l'habitat numérique, la promotion, l'animation, etc.

2.1 Définition du projet

Poppy humanoïd – coquelicot en français – est un petit robot haut de 84 cm et pesant 3,5 kg imprimable en 3D. Proposé en *open source* et *open hardware*, Poppy a été créé au centre de recherche Inria de Bordeaux par l'équipe de recherche Flowers que dirige Pierre-Yves Oudeyer. Le projet est né d'une thèse réalisée par Matthieu Lapeyre sur l'apprentissage de la marche humaine. Pour les besoins de sa recherche, Matthieu a dû concevoir un robot qui était un bon modèle de la géométrie humaine. Sans grande surprise, le robot qu'il dessina ressemblait à un jeune enfant. Les 25 articulations (on parle de degrés de liberté) et l'emplacement des articulations qui ajoutaient au réalisme, différenciaient Poppy des autres robots humanoïdes, comme Nao, etc. Lorsque Pierres-Yves a entrevu, au-delà de la thèse qu'il encadrait, le potentiel du projet, il a positionné Pierre Rouanet, un ingénieur de son équipe, virtuose du langage Python, pour ai-

der Matthieu. Pierre a finalement écrit l'essentiel du logiciel de gestion du robot. Cette collaboration marqua la naissance de la Poppy Team.

Les collègues qui empruntaient régulièrement la passerelle située face au labo s'intéressèrent à ce petit robot que l'on voyait naître à travers les baies vitrées. Poppy devint rapidement la mascotte du centre de recherche et l'équipe recueillit de multiples sollicitations de chercheurs souhaitant utiliser un Poppy pour faire leurs propres recherches.

Cela amena l'équipe à repositionner le projet dont l'objectif initial était de tester différentes morphologies mécaniques et différents algorithmes de contrôle du robot et de partager les résultats obtenus entre utilisateurs. Le repositionnement du projet amena l'équipe à produire et stabiliser une plate-forme robotique commune d'échange et d'expérimentation qui soit standardisée et ouverte. Il devint donc important de pérenniser la base de code et les éléments *hardware*, mais également de publier tous les codes du projet, qu'il s'agisse des logiciels ou des éléments 3D constituant le robot.

Fin 2013, l'ouverture du projet fut annoncée à l'occasion du séminaire LIFT organisé par la FING à Marseille. Le succès fut si large que la petite Poppy-team fut submergée de courriels provenant des quatre coins du monde et il fallut s'organiser différemment pour être capable de répondre à des contraintes qui pouvaient paraître contradictoires : diffuser largement l'utilisation de la plate-forme, accroître le nombre d'inscriptions, augmenter les usages du robot, le nombre de contributions, créer un écosystème autour de la plate-forme et ses concepts, absorber la demande en satisfaisant la curiosité des visiteurs, etc. Mais il fallait en même temps parvenir à libérer du temps aux membres de l'équipe pour qu'ils puissent travailler sans être sollicités de manière trop prégnante. La problématique ainsi posée, à savoir la recherche de la bonne réponse à ces différentes questions, depuis la définition d'une stratégie jusqu'à la mise en place d'actions pour promouvoir le projet dans la direction souhaitée en faisant émerger une communauté, fait précisément l'objet du présent guide.

2.2 La propriété intellectuelle

Concernant initialement un robot humanoïde, le projet a évolué vers une gamme constituée de trois modèles : le *Poppy humanoid* d'origine, *Poppy*

Torso, un buste de Poppy humanoïde, suffisant pour traiter la plupart des interactions homme-machine, et *Poppy Ergo Junior*, un bras robotique articulé. Le code informatique de ces trois robots est réuni dans une seule bibliothèque python commune baptisée Pypot. Le code a été publié sous une licence à copyleft fort GNU GPL V3 (la licence GNU GPL garantit que la base de code restera ouverte et impose aux contributeurs de publier les modifications qu'ils auront apportées au code). La documentation et les *designs hardware* des pièces mécaniques sont publiés sous licence Creative Common (CC-BY-SA). Cette licence impose de citer les auteurs mais permet une commercialisation, ce qui rend le projet attractif pour des acteurs industriels et commerciaux souhaitant développer une activité commerciale autour du projet.

Cependant, après différentes discussions et rencontres, nous ne sommes plus convaincus que la licence Creative Common soit la plus adaptée pour ce domaine. À l'heure où nous écrivons ces lignes, l'équipe travaille sur la possibilité de basculer la licence CC associée aux pièces mécaniques vers une licence plus spécifique au domaine *open hardware*.

Le plus important pour le choix d'une licence est qu'elle soit en adéquation avec les valeurs portées par les membres de l'équipe principale : souhaitent-il que le projet soit vraiment ouvert ? ou craignent-ils la perte de contrôle ? ont-ils la conviction que ces choix permettront de diffuser et exploiter largement ce qu'ils ont produit ? souhaitent-ils créer un standard dans leur domaine ou bien créer un écosystème ? les aspects technico-juridiques doivent être au service des valeurs et de la stratégie et non l'inverse.

Comme cela se pratique (souvent) dans les domaines *open source* et *open hardware* (ex. projet *open source* Red Hat, projet *open hardware* Arduino), il est habituel de protéger le nom du projet en déposant la marque et de réserver les noms de domaines correspondants. Poppy a donc été l'objet d'un dépôt de marque international. Le logo est apposé sur les pièces mécaniques du robot. Une personne qui souhaite produire et commercialiser les modèles de robots Poppy ne pourra pas utiliser le nom Poppy pas plus que faire figurer le logo Poppy sans un accord préalable et écrit (qu'il soit marchand ou non-marchand) avec Inria. Dans le cas d'un logiciel, ce dernier ne pourra être distribué sans le visuel protégé. De même, cette protection permet de créer des systèmes de partenariat / cer-

tification / labellisation, et ce de manière contrôlée. Le projet doit avoir atteint un certain niveau de réputation pour que la mise en place de ce type d'arrangement ait du sens. Cependant un déficit de protection peut rapidement mener à des situations délicates voire à de très mauvaises surprises si le projet décolle.

La stratégie décidée par la Poppy team a été de diffuser et de promouvoir l'exploitation du projet auprès d'un public aussi large que possible, d'attirer le plus d'attention possible sur le projet et de développer sa réputation pour produire le plus possible d'opportunités de collaborations marchandes et non-marchandes. Il est indispensable pour cela de s'assurer que les tiers qui collaborent au projet d'une manière ou d'une autre respectent un niveau de qualité minimum. Il faut par exemple absolument s'assurer que des entreprises qui produisent et commercialisent des robots vendus sous la marque Poppy proposent des produits et services présentant un niveau de qualité satisfaisant pour les clients. Le contraire aurait un impact catastrophique sur l'image et la réputation du projet.

L'un des intérêts de protéger la marque est d'être en mesure de labelliser les distributeurs et autres partenaires (formation, développement, etc.). Cela permet d'avoir une monnaie d'échange avec les tiers et donc d'exercer un certain contrôle sur ce qu'ils font et sur leur manière de travailler. Et l'on constate que les contraintes imposées pour la labellisation se vendent plutôt bien puisqu'elles permettent d'établir une situation gagnant-gagnant avec les entreprises distributrices : l'obligation de respecter un certain niveau de qualité dans leurs prestations sert également leurs intérêts puisqu'elles auront d'autant plus de commandes, et qu'elles seront d'autant plus préconisées par l'équipe que leur qualité sera meilleure que celle proposée par leurs concurrents. C'est pour cette raison que la labellisation en tant que *Poppy partner* est un statut qui fait briller les yeux et qui permet à l'équipe d'exercer un certain contrôle sur les canaux de distribution et de diffusion.

Cela permet de s'inscrire dans une dynamique vertueuse : plus l'image du projet Poppy et de la gamme de robots est positive, plus ces entreprises distributrices vendent, plus les produits et services vendus sont de qualité, plus cela renforce l'image du projet, etc. Cette attention portée à la qualité permet de limiter grandement la possibilité de litige. En effet, en cas d'incident, de dysfonctionnement, etc. sur un robot distribué sous le nom

Poppy par une filière non agréée, la responsabilité de l'équipe ne peut aucunement être engagée. Bien au contraire l'équipe est en mesure de se retourner contre le distributeur peu scrupuleux et de l'attaquer en justice pour contrefaçon.

Ce mécanisme de protection est assorti d'une clause de non exclusivité destinée à garder la possibilité de labelliser à tout moment de nouveaux distributeurs, voire devenir soi-même fabricant / distributeur de robots et de kits Poppy. En somme, la stratégie de propriété intellectuelle de Poppy n'empêche personne de produire son propre robot ; elle favorise au contraire sa dissémination. Par contre, elle impose certaines conditions aux entités qui s'engagent dans la commercialisation de robots sous la marque Poppy. Ce modèle nous a été inspiré par des conversations avec les fondateurs d'Arduino.

2.3 Habitat numérique

Après avoir identifié les besoins (ceux de l'équipe et ceux de la communauté ciblée) en matière de collaboration et de communication, nous avons sélectionné plusieurs outils susceptibles d'y répondre à nos besoins et compatibles avec les valeurs du projet.

Pour le site vitrine, l'outil CMS Wordpress a été retenu et nous avons acheté un patron (*template* graphique) de site web auprès d'un designer indépendant. Nous avons choisi Wordpress pour plusieurs raisons : sa facilité de mise en œuvre, le fait qu'il s'agit d'un standard, le choix et le faible coût des templates permettant d'adapter l'identité visuelle du site (les templates sont quasi gratuits, celui du site Poppy nous a coûté 58$), la facilité d'édition du site qui ne requiert quasiment pas de connaissances en informatique, le nombre important de modules (*plug-in*) disponibles permettant d'étendre très facilement les fonctionnalités du site, la taille de la communauté de développeurs et d'utilisateurs.

L'équipe a alors évalué les diverses possibilités d'hébergement du site vitrine (installation, administration). Les différentes offres d'hébergement disponibles, interne à Inria ou externes, ont été évaluées et finalement un hébergement interne a été retenu. La responsabilité de l'administration du

serveur et de l'installation de Wordpress a été déléguée au service informatique du centre de recherche de Bordeaux. L'administration et la mise à jour du site a été déléguée à deux personnes de l'équipe, Matthieu Lapeyre et Stéphane Ribas.

Nous avons regardé sur quelle plateforme web les développeurs en robotique se retrouvaient afin de mener leurs projets de développement collaboratifs : la majorité des publics visés, et notamment la plupart des contributeurs potentiels à la base de code Poppy, se retrouvent sur GitHub. Du coup, pour la collaboration entre développeurs, l'équipe a fait le choix de cette forge. L'ensemble des codes informatiques et des designs des pièces mécaniques y ont été téléversés.

GitHub, basé sur la technologie Git, offre les fonctionnalités essentielles pour un travail collaboratif efficient (contrôle de code et de version, mode déconnecté, *commit*, historique, wiki, gitbook, etc). Par exemple la taille de stockage par projet est limitée et il y est aussi compliqué pour un projet logiciel autour de données massives d'utiliser ce seul outil de collaboration entre développeurs. Mais comme cela n'était pas rédhibitoire pour le projet Poppy, nous avons maintenu ce choix. Deux responsables ont été chargés de l'administration de la forge, Matthieu Lapeyre et Pierre Rouanet. Ce sont eux qui décident d'accepter ou non les modifications de code proposées par la communauté.

Plusieurs outils de gestion de projet ont été évalués (Trello, Pivotal Tracker, Iscrum, Waffle, Asana, etc.) et c'est finalement Slack, un service en ligne, que l'équipe a retenu.

Une liste de diffusion a été créée pour supporter les fonctions de partage et de communication avec la communauté. Cette liste est également destinée à diffuser une lettre d'information. Elle a été alimentée en amont, durant la phase d'analyse du contexte. Mais nous avons créé plusieurs listes de diffusions institutionnelles : poppy-project@inria.fr est utilisée pour que des personnes extérieures au projet puissent contacter l'équipe principale par exemple pour des invitations à des conférences, proposer une collaboration, etc. Nous avons aussi créé un alias poppy-strategy@inria.fr. Il s'agit d'une liste de discussion privée destinée aux membres de l'équipe principale et traitant des aspects stratégiques du projet (on ne parle pas de code dans cette liste, les discussions autour du code se font sur d'autres canaux, forum et forge Github). L'équipe a récemment créé un alias poppy-

education@inria.fr pour toutes demandes relatives à la branche éducation du projet : invitations aux conférences, propositions de collaborations, etc. Bien sûr, le forum (décrit dans le paragraphe suivant) est souvent privilégié et d'ailleurs nous n'hésitons pas à le rappeler lorsque nous recevons des courriels qui mériteraient une discussion plus large - avec la communauté - ou qui relèvent de questions de type « support ».

Le choix du forum s'est porté sur Discourse (c'est son nom) qui est un outil *open source* de type Stack Exchange et qui propose les mêmes (ou presque) fonctions que ce dernier. Les fonctionnalités principales sont : le contrôle des râleurs ou personnes ne suivant pas les règles de bonnes conduites, la création semi-automatique de FAQ, la possibilité d'archiver, de répondre par courriel, un système de badge afin de donner plus de rôles aux meilleurs *posteurs*, la possibilité de suivre la dynamique du forum (nombre de membres, nombres de post échangés, possibilité de savoir si les membres se répondent entre eux, etc.) et enfin cet outil dispose d'une interface simple et ergonomique pour les utilisateurs, augmentant grandement son attractivité.

Le forum est hébergé par le service informatique du centre de recherche Inria de Bordeaux, mais nous aurions très bien pu choisir un hébergeur externe. Nous avons privilégié un hébergement par Inria pour les raisons suivantes : la maîtrise des données, la plus grande souplesse pour l'administration, une mise en place très rapide, un coût – apparent – plus intéressant, la facilité de migration des données en cas de changement d'outil ou d'hébergeur et enfin le fait que l'équipe bénéficie ainsi de l'expertise des services informatiques d'Inria, ce qui est un plus. L'administration du forum est effectuée par Adrien Dumez, Matthieu Lapeyre et Théo Segonds.

Un compte Twitter a été créé (@poppy_project) ainsi qu'une chaîne sur Dailymotion (puis quelques mois plus tard sur Youtube). Le responsable de ces canaux de communication est Matthieu Lapeyre.

2.4 Promotion

Une liste de contacts a été créée dès les premières étapes du projet et constamment enrichie depuis. En parallèle, le site web, le code et le forum ont été publiés. Enfin, une vidéo de type teaser a été créée pour faire connaître le projet, en espérant créer un buzz.

Le tournage et le montage du clip ont été réalisés par l'équipe multimédia Inria. La diffusion à travers les différents canaux de communication du projet a très bien fonctionné et permis de passer d'une cinquantaine de membres à plus de 250 en moins de trois semaines. Cependant, sur le moyen terme, la vidéo a produit un effet inattendu : bien que très apprécié et largement relayé, le clip donne l'impression, par sa haute qualité, au commun des mortels qu'il s'agit d'un projet haut de gamme qui lui est inaccessible (cela en dépit du fait qu'il soit *open source* et *open hardware*). Pour corriger cette perception nous avons alors publié plusieurs vidéos « amateur » prises lors de nos déplacements avec nos smartphones. Cela a permis de décontracter l'ambiance en donnant l'image d'un projet plus accessible. La communauté s'est alors mise à publier toutes sortes de petites vidéos centrées sur le projet Poppy, amorçant ainsi une campagne de promotion sur le projet sans même que nous en soyons les acteurs.

Un autre aspect qui a permis d'accroître l'aspect *fun* et sympathique du projet (et donc de faciliter l'adhésion et le relais d'information par des tiers) est le look des robots Poppy. C'est un aspect important du projet bien souvent oublié par les équipes de développement logiciel : l'interface graphique, le design, le look des applications *open source* et *open hardware* sont des facteurs importants dans l'acte d'adhésion à une solution ou à un projet par les utilisateurs. Or les gens adorent Poppy, il trouvent ses structures élégantes et sa tête, qui n'a rien d'humain, « trop mignonne ». Cette sorte de charisme de Poppy a très probablement été un facteur clé de son succès et a certainement facilité le relais des messages de promotion sur la toile et aussi le passage à l'acte (décision de « construire un Poppy »).

Après plusieurs mois d'existence, une vie sociale s'est organisée autour du projet. Cela apparaît sur la page du site vitrine qui affiche l'ensemble des informations publiées sur le projet Poppy à travers le monde. Cette dernière est alimentée par de multiples sources très dynamiques. Les infos affichées proviennent des réseaux sociaux, de blogs, de webzines, etc. et l'on peut y voir l'ensemble des tweets, des vidéos, des photos, provenant de différents laboratoires, universités, particuliers, artistes, écoles, fablabs, et entreprises, au fur et à mesure de leurs publications sur le web.

Le forum est également très actif, à tel point que les informations institutionnelles passent en priorité sur ce canal de communication où elles

sont davantage vues que sur la liste de diffusion pourtant pressentie pour ce type de flux.

Cependant, au-delà de la communication via les médias numériques l'équipe a une vie intense dans le vrai monde : organisation d'ateliers, participation à des conférences, cela au niveau local (fablabs et *hackerspaces*, universités, écoles, associations, café science, réunion de famille) ou plus loin en France, eu Europe (Portugal, Belgique, Suisse, etc.) ou sur d'autres continents (Inde, Japon, etc.). Lorsqu'ils se déplacent, les membres de la Poppy team n'ont pas seulement leur robot *Poppy humanoïd* dans leur sac de sport, mais aussi des flyers qu'ils distribuent en faisant les hommes-sandwich sur les événements.

Il est important de mentionner le fait que, surtout durant les premiers mois mais cela continue, la promotion du projet Poppy s'est effectuée principalement de bouche à oreille. Les visiteurs du laboratoire de l'équipe Flowers qui venaient voir où en était le projet, prenaient des photos ou des vidéos qu'ils relayaient spontanément sur les réseaux sociaux. D'autres personnes, par exemple des chercheurs Inria venus pour évaluer de possibles collaborations, ont aussi contribué à amorcer la diffusion du projet. Comme la plupart d'entre eux enseignent en école ou à l'université ils ont souhaité monter des projets autour de Poppy avec leurs étudiants. Chacune de ces rencontres faisait augmenter nos fichiers de contacts en produisant un effet de levier, via les élèves et les étudiants. Lorsque nous rencontrons quelqu'un qui manifeste un intérêt quelconque (académique, professionnel, développeur, commercial, etc.) pour le Poppy, notre politique est de systématiquement lui conseiller de rejoindre le forum.

En somme, l'importante notoriété du projet Poppy n'est pas due à une action de promotion ou une autre mais plutôt à l'utilisation systématique de multiples vecteurs de communication, en fait de tous ceux qui nous tombaient sous la main, jusqu'à aller faire l'homme sandwich sur les salons où nous étions présents, en arpentant les stands et les allées. D'autre part cette notoriété ne résulte pas des efforts de telle ou telle personne mais des actions de tous les membres de l'équipe, menées tout le temps, relayées ensuite par les membres du forum, par les partenaires, par les collègues, la famille et les amis. C'est l'accumulation de toutes ces modestes actions qui a permis de promouvoir l'existence du projet et d'augmenter le nombre de membres.

2.5 Animation et collaboration

Le forum est un service crucial dans la stratégie d'animation de la communauté, et l'habitat numérique du projet Poppy en a fait sa pièce centrale. En effet, l'objectif des contributions externes est de faciliter les participations et d'accroître le développement des différentes ressources du projet. Pour cela le forum est utilisé comme le lieu de discussion et d'échange privilégié entre l'équipe principale et la communauté. C'est là que sont proposées les idées et que se produisent les interactions qui mènent à de nouveaux contenus, de nouvelles connaissances, et qui aboutissent au lancement de nouveaux projets.

Le forum retenu dispose d'outils permettant très facilement de poser des questions et d'y répondre, de recueillir des avis externes, d'exprimer des demandes, de publier des nouvelles comme le développement de nouveaux projets par les contributeurs externes. Dans ce foisonnement, les idées qui paraissent les plus intéressantes, les meilleurs sujets de discussion ou ceux qui sont les plus débattus, peuvent être mis en avant sur le site vitrine. De même sont mis en avant certains projets des contributeurs externes, etc. Du coup le site principal présente en permanence un flux continu d'informations vivantes, issu du forum donc en relation directe avec l'activité de la communauté et les sujets du moment. Ce mécanisme est utilisé dans d'autres projets tel que Arduino, qui utilise également le forum comme source de contenu pour les sites vitrines. Compte tenu de la dimension internationale du projet Poppy, la langue privilégiée sur le forum est l'anglais, mais les posts écrits en français ne sont pas bloqués pour autant. On essaie cependant, lorsqu'un post parait particulièrement intéressant ou qu'il a une portée générale, « d'éduquer » l'auteur du post en lui demandant gentiment de le réécrire en anglais pour que tout le monde puisse en profiter.

Le deuxième effet de levier est d'utiliser le forum comme un moyen de recueillir les avis sur un sujet (technique ou non) de manière approfondie (à plusieurs, on est plus forts). Un membre a par exemple ouvert le sujet du prix et des performances des moteurs embarqués dans la solution Poppy. Produits par le fabricant Robotis, leurs prix vont de 80 à 250 Euros, ce qui représente un budget de plus de 5 000 Euros pour les 25 moteurs intégrés dans le robot *Poppy Humanoid*.

Ce post est devenu la discussion du forum la plus active et la deuxième la plus consultée après celle initiée par un autre post intitulé, « alternative to Solidworks » traitant du logiciel hors de prix permettant le design et la conception des pièces mécaniques. Ce post a lancé le sujet le plus consulté puisqu'il dépasse désormais les 35 000 vues et arrive en tête des résultats de recherche sur Google.

Voilà des informations qui aident à la prise de décision quant à l'orientation du projet. C'est ainsi que l'équipe a décidé de lancer un projet de servomoteurs *open source*, de créer des cartes électroniques spécifiques à la robotique, de faire migrer les plans de conception vers des formats permettant l'utilisation de logiciels de CAO ouverts, ou encore de créer de nouvelles formes de robots Poppy. Finalement, le forum permet à l'équipe de maintenir une feuille de route en bonne adéquation avec les besoins de la communauté et en phase avec la réalité du terrain. De même, les posts qui témoignent des difficultés rencontrées lors de l'installation ou de la construction des robots sont suivis avec grand intérêt car ils permettent la remise en cause et l'amélioration du projet. À côté de cette dimension utilitaire, les posts d'encouragement sont fort appréciés car très bons pour le moral de l'équipe.

Un troisième effet de levier est produit par la disposition d'une large base de contacts, active, réactive et mobilisable, ce qui facilite beaucoup les actions de promotion et d'appels à contributions.

En somme le forum joue, pour la communauté, un rôle de catalyseur (producteur de contenus et de connaissances, animation, rencontres et interactions). Pour autant, cela n'arrive pas tout seul et l'animation du forum demande un effort continu, de la motivation et du temps. Chaque jour, les membres de l'équipe et quelques contributeurs externes doivent y consacrer au moins une heure, quittes à la prendre sur leur temps de *coding*. Un jour, nous avons reçu un courriel du créateur du projet robotique InMoov nous présentant le projet HEOL, basé sur la technologie Poppy, dont nous n'avions pas connaissance jusque-là. Nous avons profité de l'occasion pour inviter l'auteur de HEOL à nous le présenter sur le forum. Il a été félicité par les membres de la communauté et des discussions se sont rapidement engagées sur des points techniques et sur l'orientation de son projet. Il a même eu plusieurs conseils sur des choix de conception pour sa version du robot. Aujourd'hui des kits HEOL sont proposés à la

vente. Cette anecdote montre que le fait d'adopter une attitude ouverte et collaborative, y compris avec des projets que certains pourraient considérer comme concurrents ou même comme de la copie, peut avoir un impact très positif.

Le second volet important dans la stratégie d'animation de la communauté concerne la collaboration avec des équipes et organismes externes, plus particulièrement avec des laboratoires de recherche et des écoles : l'équipe Flowers (mais aussi des membres de la communauté en général) ont régulièrement des collaborations avec les écoles du primaire et secondaire, les lycées, les écoles d'ingénieurs et universités. Ensemble, ils mettent en œuvre et promeuvent des projets basés sur la technologie des robots Poppy.

La liste des ateliers, initiatives, collaborations et conférences auxquels l'équipe et les auteurs de ce guide ont été conviés, ou qu'ils ont organisés, serait trop longue à décrire ici. Il faut retenir que l'ensemble des membres du projet ont d'abord passé beaucoup de temps à la promotion du projet puis qu'ils ont progressivement mis en place, en fonction des moyens disponibles, des ateliers et collaborations pour attirer l'attention puis pour obtenir une certaine réputation et enfin pour accroître le nombre de membres de manière significative. Pour atteindre une masse critique de membres permettant la délégation de rôles et tâches et pour produire une dynamique, il a fallu à l'équipe de la patience et de la ténacité. Les invitations à des conférences et ateliers ne sont pas arrivées d'un coup d'un seul, et les publications dans des journaux et magazines grand public non plus. Cela a demandé un travail d'arrache-pied, sans compter ses heures : les soirs, les week-ends, tout était bon à prendre pour aller « évangéliser », faire découvrir, faire connaître, dans des ateliers, lors de soirées dans des *hackerspaces*, durant des week-ends portes ouvertes, dans des écoles, etc.

2.6 Résultats

Ces efforts ont été récompensés par une belle croissance de la communauté Poppy qui désormais compte ses membres dans près de 80 pays différents. Plus de la moitié de la communauté est constituée d'européens (53%), un cinquième vient d'Amérique du Nord suivi dans une moindre

mesure par l'Asie et l'Amérique Latine. Le forum voit défiler environ 900 membres, le compte twitter est suivi par 1 500 personnes, et nous recensons à ce jour 18 développeurs sur la bibliothèque Pypot qui a été téléchargée plus de 50 000 fois.

Plusieurs reportages sur le robot Poppy et les recherches de l'équipe Flowers ont été publiés sur des chaînes de télévision françaises et européennes, plus d'une centaine d'articles ont été publiés dans des journaux, dont certains de grande audience. Il est à noter qu'une partie de ces articles ont été rédigés par des membres de la communauté sans que l'équipe ait à s'y consacrer, ce qui démontre la valeur que peut apporter à un tel projet une communauté dynamique, motivée et impliquée.

Plusieurs collaborations sont en cours, et un nombre important des membres de la communauté produisent des revenus grâce à l'activité qu'ils mènent autour du projet (enseignements, vacations, animation de soirées, formations, projets artistiques utilisant les robots, vente de kits, prestation de conseils et/ou de support, etc). Nous recensons à ce jour plus d'une vingtaine de laboratoires de recherche qui travaillent autour du projet Poppy à travers le monde. Une branche Éducation a été créée récemment grâce à un financement de la région (FEDER) auquel l'Inria participe en fournissant un budget et des ressources humaines (4 ingénieurs). En France, les régions Aquitaine, Provence-Alpes-Côte d'Azur, Rhônes-Alpes-Auvergne et Nord Pas de Calais ont contacté les responsables du projet Poppy Education afin de monter des projets similaires pour les collèges, lycées et dans l'enseignement supérieur sur leurs territoires respectifs. Plusieurs responsables de l'Éducation Nationale soutiennent également le projet depuis Paris en encourageant les différentes académies à adopter la technologie Poppy dans l'enseignement des sciences du numériques. Des réflexions sont en cours pour d'autres financements de type non-marchand (dont on constate, en dépit du fait qu'ils n'aient pas de dimension commerciale, que leur impact est tout à fait considérable).

Un partenariat a été initié avec le réseau de Fablabs Inmediats qui compte huit fablabs français. Ce réseau a proposé aux communautés de ses huit membres de construire chacun un robot Poppy Humanoid durant un weekend. L'objectif du réseau Inmediats est que chaque fablab dispose de son robot Poppy pour proposer ensuite toutes sortes d'animations : pour le public le temps d'un week-end, d'une soirée, d'aller dans les écoles ou

d'accueillir des sorties de classes, d'accompagner des artistes dans leur travail créatif, de former des salariés d'entreprises, etc. Ce projet a pour eux été l'occasion de mutualiser l'effort d'acquisition des compétences pour le montage, le démarrage et la programmation des robots. Pour cela l'équipe Poppy a accueilli quelques semaines avant le week-end de montage deux représentants de chaque fablab pour les former et les préparer. Plusieurs Fablabs Espagnols ont emboîté le pas et sont à présent en train de construire des robots Poppy. Le but sera aussi de mutualiser et proposer des activités autour des robots Poppy aux particuliers et aux entreprises.

Compte tenu de tous ces efforts menés collectivement, le but est de trouver des moyens pour pérenniser le projet. Une solution pour cela est de faire émerger un écosystème. Mais la question qui se pose est de savoir comment parvenir à accroître la communauté pour lui faire atteindre une masse critique pour qu'un écosystème s'amorce et que le projet puisse perdurer sans qu'il soit nécessaire de maintenir une structure centrale chargée d'injecter en continu des ressources humaines et financières dans le projet pour qu'il continue à vivre.

2.7 Quelques mots sur la valorisation

Pour contribuer à ce mouvement de pérennisation, différentes actions ont été engagées ou soutenues, en espérant que leur ensemble permettra de rendre le projet autonome. Un partenariat a été mis en place avec l'entreprise bordelaise Génération Robots pour la production et la commercialisation de kits de robots Poppy. Le groupe Dassault Systèmes utilise les robots Poppy dans son centre de formation en Île-de-France.

Plusieurs hôpitaux de la région bordelaises utilisent des robots Poppy Torso dans le cadre du projet Cherry, une belle histoire qui montre comment la robotique peut avoir un impact utile sur notre société : l'association Poppy Cherry [1] a modifié et installé des robots pour des enfants hospitalisés. Les robots lisent des histoires, font des devoirs et plein d'autres choses encore. Cette expérience fait des émules en Europe et des demandes de commercialisation à plus grande échelle sont en cours de discussion. Au-delà d'une illustration de l'amorçage d'un écosystème

1. `https://projetcherry.wordpress.com/`.

l'exemple du projet Poppy montre toute l'utilité du détour par la sphère non-marchande pour faire émerger des débouchés dans la sphère marchande. Le problème que l'on rencontre souvent, et auquel il faut prendre garde, est celui des *businessmen* qui, trop pressés de gagner trois sous se précipitent vers le marché et qui du coup passent à côté des grosses opportunités qu'offre la notoriété et la diffusion que permet d'acquérir un passage par la sphère non-marchande.

Le succès du projet Poppy ne doit pas cependant cacher l'ensemble des actions peu visibles (voire invisibles comme la rationalisation des processus), menées au quotidien par l'équipe : tenir une gouvernance bien ordonnée, s'assurer de la mise à disposition de ressources de qualité (ex. la documentation), scruter et rebondir sur les sujets intéressants qui émergent du forum et les publier sur le site principal, mettre à jour du site web. Il faut aussi penser à proposer une architecture de publication donnant de la visibilité aux contributions diverses (artistes, professeurs, chercheurs, entreprises, etc.) en nommant les membres qui se sont impliqués, car la reconnaissance est un ressort important des actions de contribution.

Actuellement l'équipe principale Poppy est prête à prendre son envol avec le projet de création d'une entreprise dont les modèles de revenus seront basés sur les valeurs de l'*open source* et l'*open hardware*. Une nouvelle page du projet Poppy est en train de s'écrire mais ceci est une autre histoire.

C

Interview de Pascal Chevrel (Mozilla)

— Peux-tu te présenter ? Quels sont ta formation et ton parcours ?

Je m'appelle Pascal Chevrel, j'ai quarante-quatre ans, je n'ai pas une formation d'informaticien au départ. J'ai une formation en langues étrangères appliquées au droit et à l'économie puis en négociation commerciale internationale.

— Comment es-tu arrivé à rejoindre le projet Mozilla ?

J'ai rejoint le projet Mozilla en 2000, j'ai toujours été passionné par l'informatique. J'ai découvert le logiciel libre essentiellement à travers le projet Mozilla même si je connaissais Linux. Je ne contribuais pas à d'autres projets libres. J'étais surtout passionné par le Web. J'avais découvert le navigateur Netscape en 1995 lors d'un séjour Erasmus. À la disparition de Netscape, j'ai très vite compris que Mozilla pouvait devenir une alternative au navigateur Internet Explorer. L'offre Microsoft ne me convenait pas. L'évolution que prenait alors le Web me paraissait importante et j'avais très envie de participer à l'aventure. C'est pour ces différentes raisons que j'ai aidé Mozilla en tant que bénévole pendant environ six années. Il n'y avait pas de structure, pas de revenus, pas d'employés à l'époque. Il n'y avait pas non plus de possibilité d'embauche à l'époque (je ne cherchais pas non plus à me faire employer par le projet).

J'ai commencé à travailler avec Tristan Nitot au début des années 2000 et nous avons créé des communautés à grande échelle. Pour exemple, en 2003 nous avons créé Mozilla Europe. J'ai amené mes compétences en langues et mon côté un peu plus social que la majorité des informaticiens de l'époque.

En 2003 j'ai créé le site Mozilla Europe en 27 langues. Ce projet d'internationalisation, Mozilla n'arrivait pas à le faire, ils ne trouvaient pas les compétences nécessaires. Quand Mozilla Firefox est sorti en 2004, il a eu un succès retentissant ! L'équipe de développement de Mozilla a très vite eu besoin d'un site web traduit en plusieurs langues, c'était vital pour le passage à l'échelle et la pérennisation du projet. Le hic était que Mozilla « Monde » n'avait pas de sites multilingue (rappelez-vous que Mozilla n'était pas à l'époque aussi bien structuré qu'aujourd'hui). Lorsqu'ils ont découvert que Tristan et moi-même avions créé une association en Europe et créé un site web correspondant à leurs attentes, ils m'ont contacté pour que je travaille avec eux. Mozilla ne savait pas comment parler à un traducteur :-) je connais à la fois la culture informatique et la culture des langues. Mon rôle était de créer et animer des communautés de traducteurs (mais pas que, hein ! un peu de code aussi !).

J'ai commencé comme prestataire en 2006 pour Mozilla, car il n'y avait pas de structure pour m'embaucher en Europe. Deux années plus tard, Mozilla France est créé et je deviens alors l'un des premiers employés en Europe. Tristan quant à lui était le président de l'association Mozilla Europe. Il tenait le rôle du personnage public, la voix de Mozilla, la personne qui parle à la presse, l'étendard, il est très charismatique.

— Quelles sont les raisons qui t'ont poussé à contribuer activement sur le projet ?

Je croyais aux valeurs du projet (rendre meilleur le Web) et c'était pour moi une activité en plus de mon travail. Mon travail la journée consistait à vendre des meubles, travail bien éloigné de mes études en langues. Je voulais continuer à pratiquer ma passion pour les langues étrangères et la découverte des autres cultures (surtout l'anglais et l'espagnol). Il y a donc un peu d'égoïsme de ma part : j'ai utilisé cette activité pour entretenir mes connaissances et me rendre utile.

D'ailleurs, je suis aussi l'un des fondateurs des communautés francophones et hispanophones des utilisateurs Firefox et des traducteurs. J'ai

d'abord monté la communauté francophone, puis j'ai montré à quelques personnes hispanophones qu'il était possible de faire la même chose.

— Quel est ton rôle en ce moment au sein de l'organisation Mozilla ?

Je fais beaucoup de développement en ce moment, PHP, Python, du BASH. Je maintiens plusieurs outils utilisés par les traducteurs. Je maintiens pour exemple un outil de contrôle qualité des traductions multilangues. Cet outil était maintenu par une personne qui est partie il y a quelques années et il m'a donné les clefs du projet. Je construis aussi pas mal de tableaux de bord afin de suivre les avancées des différents projets à travers le monde. Je suis aussi très content car Mozilla m'emploie en tant qu'ingénieur alors que mes études sont loin de ce métier. Je fais aujourd'hui moins d'animation et conséquemment moins de recrutements. J'ai dû amener environ une centaine de traducteurs au total. On compte aujourd'hui environ 150 traducteurs (stables) dans la communauté Mozilla.

— Qu'apprécies-tu le plus dans ce travail ?

Ce que j'aime le plus en ce moment c'est la technique. J'ai beaucoup aimé faire le *community manager* ces dernières années mais je crois en avoir fait le tour. Je préfère donc aujourd'hui la technique.

— Quelle est ta démarche pour attirer de nouveaux membres ?

Souvent j'utilise la méthode de l'opportuniste ! Je ne me déplace pas forcement pour voir les personnes en réel, j'utilise beaucoup Internet et cela marche très bien. Je vais rechercher des personnes qui ont un profil Mozilla (et qui ont des capacités de traducteurs). J'ai une démarche de recruteur, parmi les utilisateurs des outils Mozilla. Je regarde les forums pour détecter des profils intéressants. Je vais aussi chercher quelles sont les organisations du libre pour les pays où je souhaite créer une communauté. Je regarde les activités qui sont déjà en place, j'identifie les points de contacts, les forums locaux, etc. Le but est d'identifier des personnes qui pourraient contribuer activement. Mes recherches peuvent aller plus loin, je vais regarder le Wikipédia d'un pays en particulier, surtout la page dédiée à Mozilla. Je contacte la personne qui a écrit (dans sa langue) cette page. Dans d'autres cas, je peux aussi aller chercher des traducteurs chez les projets copains ; j'ai demandé un jour au traducteur de Wordpress en Portugais de nous aider. Ces personnes ont des connaissances en langues

mais aussi ont des connaissances du Web (ils savent utiliser un Wiki, un CMS, etc.) et cerise sur le gâteau, ils connaissent le Libre.

Ces profils sont rares, il y a peu de formation. Il faut que la personne ait une culture de l'informatique en priorité, c'est important. J'évite les experts dans un domaine (les spécialistes), ils ne s'adaptent pas bien aux projets du Libre. Il est plus facile d'améliorer les compétences en langue d'une personne que de lui apprendre l'informatique.

— Peux-tu décrire le mode de fonctionnement de la communauté des traducteurs (bénévoles) ?

Nous avons plusieurs communautés de traducteurs bénévoles (communautés de communautés). Il est important de noter que les communautés de traducteurs n'ont pas le même fonctionnement, ni la même culture que la communauté des documentalistes de Mozilla ou des administrateurs des forums. Ces personnes n'ont pas les mêmes intérêts, elles ne s'impliquent pas dans le projet pour les mêmes raisons. Un codeur par exemple sera plus intéressé par le projet Mozilla au sens large, un traducteur en langue bretonne est intéressé par la défense de la langue bretonne avant tout (ce n'est pas Firefox en lui-même qui l'attire en premier lieu). Notre rôle est de permettre à toutes ces personnes de participer, de contribuer au projet (ce qu'elles ne pourraient pas faire avec des logiciels fermés tels qu'Internet Explorer).

Les communautés diffèrent aussi selon le lieu géographique et leur âge. En France, nous avons des personnes qui ont la quarantaine, alors qu'en Amérique du sud, l'âge moyen des contributeurs est plus proche des 25 ans. Je fais attention à l'âge et la maturité des contributeurs. En effet, une personne plus âgée est plus à même de mener un projet à terme, alors qu'une personne jeune pourrait être passionnée par le projet pendant un mois puis partir sur une autre passion (la guitare ?). Il y a aussi d'autres avantages à privilégier des profils adultes, ils ont une voiture, ils peuvent parfois avancer des petites sommes d'argent pour les pizzas, etc.

Je passais 80% de mon temps à animer les communautés, 20% sur du code. J'ai inversé la tendance depuis quelques mois…

Les contributeurs Mozilla peuvent aussi changer de « travail » au sein du projet assez facilement, ils évoluent en fonction des avancées technologiques du projet et de leurs propres désirs.

— Comment sont prises les décisions au sein de ces communautés ?

Il n'y pas de *leader* dans la communauté francophone ! Les décisions sont prises de manière collégiale. Par ailleurs la communauté francophone est très influente. Les associations en France sont très nombreuses contrairement à beaucoup d'autres pays dans le monde. L'esprit associatif est très fort en France.

En Espagne, les prises de décision sont faites par un *leader* désigné par la communauté. Nous sommes face à une culture du chef. On retrouve cette même culture en Asie mais ils évitent les conflits, il faut donc souvent deviner quand ils ne sont pas d'accord (rire).

Parfois la culture du métier peut influencer la manière de travailler et de prendre les décisions : les codeurs ont mondialement à peu près tous la même culture du code.

— Peux-tu décrire le cycle de vie d'un bout de code Mozilla depuis l'idée de départ jusqu'à l'insertion du code correspondant dans la branche principale ?

En fait, lorsque l'on me demande de faire un développement, ou lorsque j'ai une idée de développement, je laisse tout tomber pour ne faire que ce projet.

J'ai souvent à faire face à des problèmes très concrets et je dois avoir alors une approche pragmatique pour le résoudre. Mon équipe travaille en mode agile (*eXtreme Programming*) et Mozilla laisse de l'autonomie à ses équipes. On évite de réinventer la roue, on dépose tout sur GitHub.

Je produis une preuve de concept (souvent cela me prend deux jours) et je le diffuse à des collègues pour qu'ils la testent. Si la mayonnaise prend (je sonde sur une semaine) alors on va faire suivre le bout de code à l'équipe responsable de la mise en production des codes. Ils suivent des procédures classiques de tests et de mise en production. Cela va très vite, entre le moment de la demande et la mise en production, il peut se passer entre dix et quinze jours maximum.

Si la mayonnaise ne prend pas, on enlève la fonction et le code associé (on veut éviter les bouts de codes morts).

Attention, il y a des départements chez Mozilla, orientés recherche, qui prendront plusieurs années avant de pousser leur code en production. Ils

s'occupent de projets sur le long terme qui demandent plus de tests, de réflexions et de qualités… Ils ne suivent pas les mêmes procédures et n'ont pas les mêmes pressions. Ils vont souvent au bout des choses afin de proposer des architectures, des algorithmes, de nouvelles fonctionnalités qui répondent au mieux aux problématiques du Web et qui sont fiables.

— Peux-tu me décrire vos pratiques en termes de propriété intellectuelle sur le projet ?

Les copyrights appartiennent à chacun des contributeurs. Le logo, et le nom Mozilla, appartiennent à Mozilla mais c'est à peu près tout.

Ah si… une règle importante chez Mozilla : éviter d'utiliser des codes fermés, uniquement des projets libres. Peu importe si la licence est contaminante ou pas, il faut en priorité utiliser du Libre.

La fonction est plus importante que le code !

J'utilise souvent la licence Mozilla (MPL2.0). On va souvent choisir des licences plus permissives que d'autres selon l'exploitation qui sera faite par des tiers. Un exemple : certains projets Mozilla doivent s'intégrer dans des projets BSD ou Apache, on va alors choisir une licence de sortie plus permissive.

Je me pose souvent la question suivante : est-ce que je fais du code pour qu'il s'intègre dans un projet bien déterminé (tel que Firefox) ? Ou est-ce que je suis en train de développer un code qui a vocation à être intégré de manière la plus large possible ? Auquel cas, la licence du code devra pouvoir s'adapter à de telles situations (certains codes de Mozilla doivent pouvoir être intégrés dans différentes distributions GNU Linux, Debian, BSD et autres), Mozilla privilégiera alors une licence plus permissive. Aujourd'hui (2016), je pense que nous allons vers de plus en plus de licences permissives (ce n'est pas un mal).

Pour Mozilla, le logiciel libre est un outil de pérennisation, de collaboration. Le projet de Mozilla a pour but d'améliorer le Web. Mozilla est très orienté logiciel libre mais il ne s'interdit pas de discuter certaines fois la possibilité d'intégrer un logiciel fermé dans leur suite logicielle. Il faut que cela se justifie par, entre autres, un besoin.

— Avez-vous une architecture de participation particulière ?

Nous avons plusieurs outils et plusieurs lieux (je parle des communautés de traducteurs).

Site vitrine : nous avons deux sites vitrines (les codes de ces sites web sont déposés sur Github).

Pour le code : « *The repository is king* ». Le dépôt des traductions (Mercurial) est la référence. Nous avons des personnes qui vont aider les traducteurs à intégrer leur travail sur le dépôt de référence, les aider à utiliser les outils de traduction, etc.

Communication : pour la communication, nous utilisons notre propre serveur IRC (environ 1 400 salons différents). Nous utilisons aussi beaucoup les listes de diffusions.

On met à disposition des wikis où les contributeurs déposent des informations sur leurs projets de traductions, leurs contacts, leurs avancements, leurs règles de fonctionnement, etc.

Les équipes sont libres de choisir leurs manières de travailler, l'équipe Mozilla propose des outils pour qu'ils puissent travailler comme ils le souhaitent. Mozilla leur demande juste de remplir quelques informations qui permettent de créer les tableaux de bord (suivi des projets au niveau mondial).

Si une équipe fonctionne mal, Mozilla intervient en les redirigeant vers des guides de « bonnes pratiques ».

— Avant de terminer cette interview, peux-tu nous parler rapidement des activités d'animations ?

Plein d'activités sont organisées, beaucoup ! Ce serait trop long à décrire mais par exemple nous organisons des meetings (hackathon) tous les mois entre les traducteurs dans différents pays. Mozilla incite toutes les initiatives visant à donner du rythme, donner un sentiment d'appartenance au projet. Mozilla peut aider à organiser et à donner un budget (petit).

— Cette interview est terminée, merci Pascal.

Merci à vous :)

Interview de Stefano Zacchiroli (Debian)

— Peux-tu te présenter, exposer ton parcours et comment tu en es arrivé à contribuer au projet Debian ?

Je m'appelle Stefano Zacchiroli, je suis maître de conférence en informatique à l'université Paris VII actuellement en délégation à l'Inria. Côté communautaire, cela fait un peu plus de 15 ans que je suis actif dans le domaine du logiciel libre (comme activiste et développeur) et la plupart de mes énergies ont été dédiées à la communauté Debian. J'ai commencé comme utilisateur, contributeur, puis développeur. Finalement, je suis devenu *leader* du projet Debian (DPL – *Debian Project Leader*) au niveau mondial et pendant 3 ans (élu trois fois pour un mandat de 1 an). J'ai aussi d'autres rôles dans diverses organisations internationales comme notamment l'OSI (*Open Source Initiative*) où je suis membre du conseil d'administration (*Board Director*).

— Ton rôle, tes activités au sein de la communauté et du projet Debian ?

Aujourd'hui, après avoir été DPL, je suis redevenu développeur « simple ». Je suis particulièrement actif dans le développement de certaines parties logicielles qui composent l'infrastructure Debian. Je suis le responsable de la maintenance (*maintainer*) du code de la plateforme (et des outils associés) de sources.debian.net. Il s'agit d'un site web où vous

pouvez parcourir tous les codes sources de Debian passés et présents. Il est possible de faire des recherches, pointer des lignes de code, etc. Dans le passée j'ai été le responsable de la maintenance de plusieurs paquets autour des langages OCaml et Python, et de l'éditeur Vim. J'étais très focalisé sur le développement d'outils pour l'assurance qualité de Debian. Aujourd'hui, je suis davantage concentré sur le développement d'infrastructures.

— Qu'est-ce que tu apprécies le plus dans le projet ?

C'est un projet très particulier – même pour le domaine du logiciel libre – car il est très vieux (23 ans, il date de 1993). À l'époque il n'était pas évident de savoir ce que voulait dire par exemple « développer en ligne » (c'est plus courant aujourd'hui). Tout était à inventer, et au-delà de ses contributions techniques, Debian (il s'agit d'un système d'exploitation très réputé et largement utilisé à travers le monde, Wikipédia, Google.com, etc.) comporte une particularité importante : un volet social et politique. Cette dimension était très en avance pour son temps (dans les années 1990 à 2000). Par exemple, comment définir une communauté en ligne et comment s'organise-t-on ?

Debian est une organisation très *bottom-up*, presque anarchique selon certains points de vue. Cependant, le projet s'est structuré au fil des années, Debian s'est doté d'un manifeste et d'une constitution qui décide de ce que l'on fait, qui décide, quelles sont les valeurs qui vont permettre d'aider à faire à la fois des choix techniques, politiques, légaux, etc. Sur les aspects légaux, Debian est devenu aussi l'un des acteurs réputés dans le domaine, par exemple dans la décision si une licence est considérée « libre » ou pas. Donc ce sont toutes des innovations qui sont très particulières et qui font de Debian un cas d'étude très riche dans le domaine du logiciel libre.

Un dernier point très important pour le projet est le volet de l'indépendance. Le projet n'a pas été créé par une entreprise, ni pour une entreprise. Le projet n'est pas contrôlé par une entreprise spécifique, même en termes de fonds. Debian est plutôt éloigné des intérêts du commerce. Si l'on regarde les autres distributions les plus connues, une telle indépendance est plutôt rare.

— Peux-tu nous décrire le fonctionnement du projet, la gouvernance Debian ?

Il faut d'abord se pencher sur le côté quasi-anarchique du projet. En effet, les membres du projet peuvent choisir sur quoi ils souhaitent travailler. Ils décident quoi faire et comment le faire. Ils doivent par contre respecter les standards de qualité imposés par le projet Debian. En revanche, comme nous avons des milliers de développeurs qui s'auto-organisent, il est difficile de faire rapidement des changements à grande échelle. Cela peut prendre très longtemps. Imposer des changements techniques qui auront des impacts sur des milliers de paquets Debian prend beaucoup de temps, beaucoup de discussions. Il faut convaincre les développeurs de l'approche.

Cependant, et malgré ces discussions ouvertes, il peut y avoir une minorité de collaborateurs qui travailleront quand même comme ils l'entendent. Debian est un projet qui s'auto-organise et forcément l'un des inconvénients est peut-être le temps pour faire ces changements techniques. On pourrait croire qu'au fil des années, le projet, parti d'un bazar, est devenu une sorte de cathédrale. Il n'en est rien, Debian est un projet composé de sous-communautés par sous-domaines (jeux vidéos, traductions, desktop, etc.). Chacune de ces sous-communautés travaille selon ses propres règles (mais doit respecter le code qualité de Debian). Le tout forme un bazar de bazars ! Nous n'avons pas fait le passage du bazar vers la cathédrale comme cela a été le cas pour un grand nombre de projets de logiciels libres lors de ces dernières années. Chacun de ces paradigmes apporte son lot d'avantages et d'inconvénients.

Je prends souvent l'image des tentes pour décrire le projet Debian. Considérer le projet comme un ensemble de tentes… Il y a une tente principale et plein d'autres tentes dispersées sur le terrain. Chacune a ses propres règles. Cependant, ces dernières doivent respecter quelques règles émises par la tente principale (règles construites avec la collaboration de la communauté). En effet, le projet Debian a été très influencé par les communautés de standards (IETF, W3C, etc), et pousse fortement à la normalisation. Il y a donc des standards techniques que tous les projets (tentes) doivent respecter. Par exemple, vous devez respecter la Debian (*Technical*) *Policy* qui explique comment un paquet Debian doit se comporter d'un point de vue technique (la composition du nom, où doivent être ins-

tallés les fichiers d'un certain type, comment doivent se passer les mises à jour, etc). Il s'agit d'une spécification technique mais qui est accompagnée d'outils pratiques. On utilise par exemple des outils d'intégration continue permettant de voir si les packages proposés par les sous-communautés respectent les règles. Un groupe de personnes est responsable de la maintenance de ce document. Il s'agit donc là d'une gouvernance plutôt technique. Pour les aspects sociaux (comme p.ex. les conflits) nous avons au sein du projet un comité (*Debian Technical Committee*) qui peut agir en médiateur sur des conflits et le cas échéant trancher. Ils agissent parfois comme une sorte de tribunal (interne).

La Constitution Debian (document construit et voté initialement par les membres du projet en 1995) fixe les règles de constitution des différents corps de gouvernance de Debian, leur responsabilités, et comment sont élus les membres de chacun, pour les cas dans lesquels une élection est nécessaire. C'est le document de référence, et lorsqu'une nouvelle personne est proposée pour rejoindre le projet Debian, ce dernier doit accepter la constitution. À l'intérieur de ce cadre les différentes équipes s'autogouvernent.

Il faut reconnaître qu'après près de 23 ans d'existence, la plupart des conflits se sont résolus par le simple fait de faire des réunions de discussions entre les différents protagonistes. En somme, les conflits ont toujours ou presque été résolus sans avoir besoin de faire intervenir les grands gourous du projet Debian. Pour l'anecdote, je crois avoir compté 10 « gros » conflits sur 20 ans ! C'est très, très peu.

— **Peux-tu me dire comment cela est possible ?**

La procédure de « recrutement » aide grandement. En effet, lorsqu'une personne veut rejoindre Debian en qualité de membre officiel du projet, nous vérifions que les principes de base du projet Debian sont acceptés et comprises par la personne. Au delà de cet accord sur les principes de base, la granularité des paquets aide à créer des zones d'autonomie non conflictuelles. Les conflits remontent donc très peu, car ces choix ne dépassent pas les limites de chaque zone (sorte de silo du paquet). L'auteur original de Debian, Ian Murdock (mort en 2015), décrit souvent le projet Debian de la manière suivante : « Le système de gestion des paquets Debian n'a pas été créé à la base pour gérer du logiciel mais pour gérer de la collaboration ».

Quand je pense que ces idées datent de 23 ans, c'est assez impressionnant !

— Comment faites-vous pour gérer la croissance du projet, attirer de nouveaux membres, de nouvelles contributions ?

Une *release* de Debian contient au peu près 40 000 paquets. Ce qui implique beaucoup de personnes et beaucoup de zones autonomes. En créant ces zones d'autonomie, nous avons permis d'attirer un grand nombre de contributeurs mais cela apporte aussi un inconvénient : il arrive que nous passions beaucoup de temps et d'énergie à trouver si tel ou tel paquet est toujours bien maintenu par au moins une personne. En effet, celle-ci peut avec le temps et l'âge avoir d'autres centres d'intérêts, car la plupart des contributeurs Debian sont bénévoles, sans avoir informé le projet. Il faut donc s'assurer que la (dernière) personne derrière un paquet n'a pas disparu. Dans ce cas, il faut trouver un nouveau responsable de la maintenance du code. « *There is no free meal !* ».

— Peux-tu décrire le cycle de vie d'un bout de code Debian et nous dire quand vous décidez de publier le code à un large public (*Official Release*) ?

Une distribution (un système d'exploitation et ses extensions) est par sa nature un objet logiciel très particulier. Notre rôle principal est de faire de l'intégration. La plupart du code que l'on intègre est produit par des personnes externes que nous appelons les *upstreams*. Nous nous assurons que les codes que nous intégrons respectent les conventions de la *Debian Policy* et qu'ils n'ont pas d'interactions malveillantes entre eux, c-à-d. qu'ils sont compatibles entre eux. Nous développons aussi nos propres logiciels, comme l'installateur, le gestionnaire de paquets, ou l'infrastructure de Debian.

Cela dit, le cycle de vie commence par la *release* d'un nouveau logiciel qui a vocation a être intégré dans Debian. Ce logiciel proposé par un *upstream* n'a pas été créé sous le contrôle de Debian. Donc l'arrivée de nouvelles versions ou d'un nouveau logiciel n'est pas sous le contrôle Debian, il en arrive tout au long de l'année.

Les responsables de la maintenance des paquets en Debian ont des outils pour s'apercevoir qu'il y a une nouvelle version *upstream* des logiciels

qu'ils maintiennent en Debian. Ces outils peuvent être à faible technologie comme une annonce envoyée sur une liste ou des outils qui vérifient périodiquement un dépôt sur un site web (où sont publiées les nouvelles versions).

Ensuite, vient la phase d'intégration dans Debian de ce nouveau logiciel. Le logiciel passe par une phase de test. Nous regardons si ce nouveau paquet pourrait remplacer certains « anciens » paquets de la distribution. Si nous effectuons des modifications (correction de bugs principalement), nous demandons à la personne *upstream* de rapporter ces modifications dans sa branche principale. Une fois le report effectué, nous supprimons les *patchs* correctifs que le responsable de la maintenance du code en Debian a écrit.

Une fois que le code *upstream* a été *packagé* par Debian, il est automatiquement re-compilé pour les différentes architectures informatiques supportées par Debian (une dizaine environ x86, amd, etc.). Le paquet entre par default dans une version de Debian qui est nommée *unstable* ; c'est la version en cours de développement. Cette version est souvent utilisée par les développeurs et les personnes qui veulent avoir la toute dernière version d'un logiciel. Après une semaine passée dans *unstable*, re-compilé sur les différentes architectures et si personne n'a signalé de bugs sérieux sur ce paquet, alors il passe automatiquement dans une version qui s'appelle *testing* – sorte de *staging area*, c'est une branche qui reçoit les paquets qui ont déjà été testées (avec succès). Tous les deux ans, Debian gèle le passage des paquets entre *unstable* et *testing*. Les développeurs travaillent alors à nettoyer tous les bugs sérieux qui auraient pu s'accumuler entre-temps. Une fois cette phase terminée, ils produisent une version dite stable. Une version stable est donc produite environ tous les deux ans. En gros l'approche aux *releases* de Debian est un mélange de *testing* automatiques (recompilation, intégration continue, etc) et de *user testing*.

Lorsqu'il s'agit de corrections qui touchent au domaine de la sécurité, Debian n'attend pas deux années pour apporter le correctif dans la branche stable, évidemment, nous proposons des mises à jours ponctuelles pour ce genre des problèmes.

— L'organisation Debian comprend plusieurs sous-communautés, comment faites-vous pour être sûr qu'un responsable de la maintenance est toujours actif sur le paquet qui lui a été confié ?

Nous avons une initiative qui s'appelle *Missing in Actions* (MIA) qui a pour but de vérifier si un développeur est toujours actif. Nous pouvons vérifier cela à plusieurs niveaux : tous les *commits*, les téléversements des paquets sont publics. On peut aussi trouver l'information sur l'activité du responsable de la maintenance grâce aux votes (est-ce qu'il a participé aux derniers votes, etc.). On peut donc facilement vérifier si un développeur n'a pas fait un *commit* depuis plusieurs mois, voire aucun depuis deux ans.

Si c'est le cas, une équipe va gentiment le contacter et lui demander s'il est toujours actif. Sans réponse, ou s'il confirme être inactif, alors son paquet est mis en état « orphan » (orphelin). On publie une information disant que ce paquet n'a plus de responsable et une autre personne de la communauté peut alors se déclarer intéressée pour prendre la suite (devenir le nouveau *maintainer*). Si personne ne prend la relève, nous pouvons décider, après un certain laps de temps, de retirer ce paquet de la distribution. En effet, au fil du temps, des bugs vont s'accumuler ce qui va nuire à l'ensemble du projet (le paquet n'est plus au niveau de qualité requis).

— Comment Debian peut avoir confiance dans le travail que les contributeurs produisent ?

Il s'agit principalement de faire de la revue de code. C'est rendu possible car le code source est disponible (que ce soit pour les modifications faites par les responsables de la maintenance ou pour les codes provenant des *upstreams*). Si un bout de code malveillant a été ajouté, il sera identifiable très rapidement. Nous avons donc à la fois une immense confiance dans le travail des développeurs Debian grâce à leur *track record*, mais nous (et nous tous d'ailleurs) pouvons aussi facilement vérifier leur travail.

— Peux-tu me décrire vos pratiques en termes de Propriété intellectuelle sur le projet ? Est ce qu'un nom de marque est déposé ? Qui possède les droits et comment sont gérés les apports des contributeurs externes ?

Côté droits des marques, Debian a des marques déposées : le nom, les logos de la distribution et les logos de notre conférence principale (Deb-

Conf). Cependant, il n'y pas une seule entité légale dénommée « Debian ». Ce qui existe ce sont plusieurs entités autour du monde qui sont des *trusted organisations*, des organisations à but non lucratif auxquelles Debian a formellement décidé de faire confiance afin de garder ses biens. Ces biens peuvent être de l'argent (qui provient de donations) ou de la propriété intellectuelle (notamment les marques). Historiquement, la *trusted organisation* la plus importante pour Debian a été la *Software in the Public Interest* (SPI) qui a été créée par Debian en 1997) pour ses propres besoins. On a très vite compris que ce modèle de gestion intéressait d'autres projets orientés logiciels libres. L'organisation SPI a été alors généralisée afin d'aider d'autres projets. Aujourd'hui SPI organisation gère les biens de plus de 30 projets logiciels libres.

Il y a aussi les droits des auteurs, le *copyright*. Debian ne force aucun développeur à donner ses droits d'auteur ; par défaut chaque développeur qui participe à Debian garde son propre droit d'auteur sur les contributions qu'il a faites. Debian a ajouté récemment la possibilité pour les développeurs qui le souhaitent de céder leur droits d'auteurs à l'organisation *Software Freedom Conservancy*, qui est spécialisée dans la gestion de droit d'auteur dans le domaine du logiciel libre, et notamment dans l'application des licences de type *copyleft* comme la GPL. Si un développeur devait se « défendre » seul contre des violations de licence comme la GPL, cela lui coûterait beaucoup de temps (et je ne suis pas sûr qu'il connaisse toujours bien les procédures à suivre). Cet accord avec Software Freedom Conservancy permet donc de pouvoir faire valoir les droits et les licences choisies par le développeur par une organisation de confiance.

Il existe plusieurs autres organisations auprès desquelles nous transférons nos biens : Debian France (Loi 1901), Debian CH en Suisse, etc. Nous avons réparti nos biens sur plusieurs organisations car si, pour des raisons techniques ou politiques, une organisation devait disparaître, les biens de Debian ne serait pas mis en défaut dans leur ensemble. Nous avons réparti les risques.

— Comment faites-vous pour garantir que ce que vous intégrez dans Debian est légal ?

Nous avons une équipe de reviewers dédiée à cela : les *ftp-masters* ! (les maîtres de l'archive). Lorsqu'un nouveau paquet est apporté intégré

en Debian, il est mis dans une zone de *staging*. Cette zone n'est pas publique. Les personnes de l'équipe vont vérifier le paquet : est-ce que le développeur responsable du paquet a fait une liste exhaustive de toutes les licences qui existent dans ce paquet et les a synthétisées dans un fichier nommé *Debian copyright* (tous les utilisateurs retrouvent ce fichier dans leur distribution) ?

Ils vérifient qu'aucune licence n'a été oubliée et vérifient que les licences sont compatibles avec les requis de liberté de Debian. Cette étape de review manuelle obligatoire est faite selon une approche pragmatique et heuristique, lors de l'arrivée d'un nouveau paquet en Debian. Pour le reste nos utilisateurs peuvent aussi nous remonter des erreurs de licence ou d'autres problèmes de nature légale. L'équipe va alors re-vérifier le paquet (et s'il le faut, retirer celui-ci de la distribution).

— Avez-vous une architecture de participation particulière ? un site web ? un lieu pour partager et collaborer sur la base de code ? un lieu pour discuter ?

Nous avons une réflexion importante sur notre infrastructure de participation. Cependant, comme Debian est un projet qui date de 1993, à cette époque les outils n'étaient aussi performants qu'aujourd'hui. Comparée à certains outils qui, en 2016, font tout, notre infrastructure peut paraître un peu ancienne.

Nous avons plusieurs outils qui ont été développés en grande partie par les membres Debian il y a désormais très longtemps. Notre architecture de collaboration est composé principalement d'une forge (*fusionforge*) et d'un gestionnaire de bug développé par Debian (il se nomme *debbugs*). Cet outil a été adopté par d'autres projets (ex. : GNU Emacs). Les logiciels de gestion de l'archive Debian, et notamment *dak*, ont aussi été développés par l'équipe Debian. Nous disposons d'un réseau de serveurs de téléchargement (miroirs) afin de distribuer les *releases* Debian à travers le monde.

Pour la communication, nous utilisons un gestionnaire de listes de diffusion standard. Nous utilisons aussi beaucoup le réseau IRC (OFTC). Le site web – site vitrine – est écrit en langage WML.

Si l'on regarde les technologies utilisées dans le projet Debian, je pense qu'elles font vintage ! Cette suite d'outils numériques est cependant assez standardisée et bien maîtrisée par les développeurs Debian.

Comme je le disais en début d'interview, nous avons chez Debian une culture des standards. Il y a un compromis entre « documenter comment les choses fonctionnent » (provient de la culture du standard) et « imposer l'utilisation de certains outils ». Il y a toujours des tensions entre les deux, nous essayons de trouver le bon équilibre. Si l'on compare Debian à d'autres distributions (distributions souvent plus structurées - à la manière d'une cathédrale), on retrouve bien plus d'uniformité sur leurs manières de travailler que dans notre projet.

— Peux-tu me donner une description des règles et outillages que vous avez mis en place pour la gestion de projet ?

Nous n'avons pas vraiment d'outils pour le suivi de projet, par contre, nous avons une équipe (la *Debian release team*) qui va se charger du suivi des *releases*. Cette équipe a beaucoup de pouvoir. Elle décide, par exemple, de la date de mise en *freeze* d'une version de Debian, de la date de release d'une nouvelle version stable, quels bugs doivent être considères comme bloquants pour une *release*, ... et aussi du nom de chaque *release* !

Pour ce qui est de la feuille de route, c'est plutôt le DPL qui organise les discussions « politiques » les plus importants pour le projet.

— Quels sont les profils de vos contributeurs ?

Les projets de logiciels libres émergent souvent à travers l'idée de « *scratch your own itch* ». Debian étant un système d'exploitation, il est donc normal qu'au démarrage du projet, les membres aient été composés principalement d'administrateurs systèmes (*sysadmins*). L'évolution du projet pendant les 5 à 10 dernières années a poussé les membres de la communauté à réfléchir sur les profils des membres du projet. Nous avons compris que Debian avait besoin de différents profils de contributeurs. C'est aussi important car Debian est indépendant, autonome, nous devons faire quasiment tout nous-mêmes.

Par exemple, nous organisons, chaque année, une conférence, la DebConf, avec entre 300 et 400 participants chaque année. Cette conférence est entièrement gérée par des bénévoles. Nous devons donc faire la mise

en place et la maintenance du site web (site vitrine), faire la communication par nous-mêmes, etc. Il est donc important de diversifier les profils des contributeurs. Afin d'accueillir ces nouveaux profils, nous avons compris qu'il ne fallait pas être obligatoirement un développeur. Donc, si une personne contribue régulièrement et sur la durée, si cette personne comprend et accepte les valeurs et les règles du projet, elle peut alors demander à devenir développeur Debian. Elle n'a pas besoin d'apporter une contribution spécifiquement technique.

Nous avons aujourd'hui d'un côté des *sysadmins* et des développeurs et de l'autre coté, des graphistes, des traducteurs, des *community managers*, etc.

— Comment faites-vous pour attirer de nouveaux membres ?

Tout projet de logiciel libre a besoin de renouveler ses membres périodiquement, pour garantir un bon de turnover. Du fait de la nature des profils des contributeurs, principalement des bénévoles, ces personnes peuvent à un moment partir, avoir d'autres centres d'intérêts, et vont disparaître (pour peut être revenir plus tard). Nous avons donc mis en place une équipe dédiée (son nom : Outreach). Cette équipe a pour but de réfléchir sur la façon d'attirer de nouveaux contributeurs. Il y a plusieurs phases dans le cycle de vie du contributeur : tout d'abord la personne est plutôt « passive », elle utilise la distribution Debian (c'est un utilisateur). Par la suite, cette même personne va commencer à faire des rapports de bugs. Comme tout est public, lorsqu'un rapport de bugs est ajouté, n'importe qui peut le regarder et se dire « est-ce que je peux corriger moi-même ce bug ? ». Un bug n'est pas forcément relié à un bout de code, il peut aussi s'agir d'un problème de mauvaise traduction, etc.

Afin d'attirer de nouveaux membres, nous participons périodiquement aussi au « Google Summer of Code » et à l'initiative « Outreachy ». L'objectif principal de notre participation à ces deux grands événements (sous forme de stages effectués à distance sur plusieurs mois) est de garder, par la suite, le / les contributeur(s).

J'ai aussi remarqué que certains contributeurs vont partir vers des projets connexes à Debian ou, à l'inverse, que des membres actifs de projets connexes vont venir se joindre au projet Debian (exemple : des développeurs Ubuntu rejoignent le projet Debian pour être plus près de la source de développement).

— Vous collaborez avec beaucoup d'autres projets, je crois même que vous avez quelques relations avec la Linux Foundation ?

Nous n'avons pas de lien direct avec la Linux Foundation mais ils sponsorisent, à travers leur *Core Infrastructure Initiative* (CII), le projet Debian de *reproducible builds* (garantir que la même version d'un code source compilé, son binaire, est identique bit à bit). Cela est important pour des raisons de tests, mais aussi pour les utilisateurs qui peuvent vérifier que le code exécutable qu'ils utilisent sur leur machine correspond au code source publiquement disponible et « reviewable » (par ex., pour l'absence de *backdoors*, failles de sécurités, etc.). Spécifiquement CII paie deux développeurs Debian afin d'attendre l'objectif de 100 % des paquets Debian *reproducible buildable*. Aujourd'hui nous sommes à près de 80 % de paquets *reproducible buildable* dans l'archive Debian.

— Pour terminer, pourrais-tu me décrire le modèle de financement Debian ?

Une des responsabilités du DPL est de gérer les finances du projet. Les développeurs, membres et contributeurs, ne sont pas des employés de Debian. Bien sûr certains d'entre eux sont payés par leurs entreprises pour travailler spécifiquement sur le projet Debian. Mais ce ne sont pas des employés Debian. Ces entreprises ont compris l'importance pour eux de participer (et faire ainsi progresser le projet selon leurs besoins).

C'est important car nous n'avons pas à gérer les salaires des employés. Cela facilite grandement la gestion du budget. Par contre, nous avons deux ou trois sources principale de dépenses.

La première est l'organisation de la conférence DebConf (coût : 200 000 à 400 000 dollars US). Cette conférence est auto-financée car les sponsors spécifiques à la conférence donnent assez d'argent pour couvrir les frais.

La seconde dépense est la maintenance des serveurs (environ 100 machines physiques, sans compter les serveurs miroirs). Cela représente entre 20 000 à 30 000 dollars US par an des maintenance, remplacements, contrats de garantie, etc.

La troisième source de dépenses est l'organisation des voyages des développeurs. En effet, pour faciliter la collaboration entre contributeurs, nous organisons des réunions physiques, le temps d'un week-end ou d'une

semaine, sous forme d'un *code camp*. Debian aide ces personnes dans la logistique et la prise en charge de leurs voyages.

En source de revenus, à la fois les utilisateurs et les entreprises donnent de l'argent assez régulièrement. C'est la principale source de revenu. Les entreprises ont des profils divers (informatique, automobile, santé, petites ou moyennes et grandes entreprises, etc). Elles donnent ce qu'elles veulent, les donations ne sont pas basées sur la taille de l'entreprise. Pas de *fees*, que des donations !

— **Cette interview est terminée, merci Stefano.**

Merci à toi !

Glossaire

(source Wikipedia)

API *Application Programming Interface*, interface de programmation, est un ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.

Big data Les *big data*, littéralement les « grosses données », ou mégadonnées (recommandé), parfois appelées données massives, désignent des ensembles de données qui atteignent un tel volume qu'ils en deviennent difficiles à travailler avec des outils classiques de gestion de bases de données ou de gestion de l'information.

Blog Site web sur lequel un internaute tient une chronique personnelle ou consacrée à un sujet particulier.

Post Le terme *post* est largement utilisé par la communauté Internet du monde entier pour désigner un message « posté » (émis, envoyé) sur un *forum* de discussion ou sur toute plate-forme d'échange ou sur un site web de type blog (on parle alors de *blog post*).

Board, Conseil d'administration (CA) Ensemble des membres du projet (à travers une association) élus pour définir la stratégie, diriger les débats, prendre les décisions, garantir la propriété intellectuelle et les valeurs, distribuer les ressources, coordonner les activités, etc.

Bug En informatique, un *bug* est un défaut de conception d'un programme informatique à l'origine d'un dysfonctionnement.

CLA ou *Contributor License Agreement* Il s'agit d'un contrat de licence entre le projet et les contributeurs. Il s'agit d'un document qui explicite les règles sur la propriété intellectuelle du projet, son code et les ressources associées (tutoriels, documentation, images) et les contributions apportées par une personne ou une entité n'appartenant pas à l'équipe principale et donc non détentrice des droits sur le code.

Copyright Assignment Il s'agit d'un document explicitant les règles de transfert de copyright (*when a copyright is transferred from the owner to another person or party* — source thelawdictionary.org.

Core team, équipe principale Noyau central au projet, il s'agit de l'équipe principale qui développe et dirige le projet.

Commit Il s'agit principalement de propager un code dans une ou plusieurs branches du code du projet.

Community manager Lire le chapitre correspondant ;-)

Consortium Un consortium (du latin signifiant « partenariat » ou « association ») est une collaboration temporaire entre plusieurs acteurs à un projet ou programme dans le but d'obtenir un résultat. En France, le consortium n'a pas de statut juridique ; le terme peut donc être parfois utilisé abusivement. Il reste toutefois possible, et souhaitable notamment au titre de la propriété intellectuelle, d'encadrer le consortium par un contrat.

CMS Un CMS (*Content management system*) est un logiciel dénommé en français par système de gestion de contenu (SGC). Il s'agit d'un logiciel destiné à la conception et à la mise à jour dynamique de sites Web ou d'applications multimédia.

Download server Un serveur de téléchargement est un ordinateur permettant à un utilisateur de se connecter à celui-ci via Internet ; il peut alors télécharger les fichiers du serveur ou téléverser ses fichiers sur le serveur.

Easy task list Liste des tâches du projet faciles à réaliser pour un débutant.

Fork, Forker Un fork, en français fourche ou encore embranchement, est un nouveau logiciel créé à partir du code source d'un logiciel existant. Cela suppose que les droits accordés par les auteurs le permettent : ils doivent autoriser l'utilisation, la modification et la redistribution du code source. C'est pour cette raison que les *forks* se produisent facilement dans le domaine des logiciels libres.

IDE En programmation informatique, un environnement de développement « intégré » (abrégé EDI en français ou IDE en anglais, pour *Integrated Development Environment*) est un ensemble d'outils conçus pour augmenter la productivité des programmeurs qui développent des logiciels. Il comporte un éditeur de texte destiné à la programmation, des fonctions qui permettent, par pression sur un bouton, de démarrer le compilateur ou l'éditeur de liens ainsi qu'un débogueur en ligne, qui permet d'exécuter ligne par ligne le programme en cours de construction. Certains environnements sont dédiés à un langage de programmation en particulier.

IRC Acronyme de *Internet Relay Chat*. Canal de discussion en temps réel via un outil dédié ou un navigateur web.

Issue tracker, bug tracker Système informatique de prise en compte et de suivi des remontées utilisateurs sur les dysfonctionnements possibles de l'application.

Flyer Tract publicitaire, un prospectus au format papier qui est distribué dans des lieux de passage pour trouver des prospects. Les tracts sont souvent utilisés pour promouvoir des soirées ou événements. Ses dimensions peuvent être variables mais sont les plus souvent inférieures au format A4 ; le plus utilisé restant le A5 avec une impression recto-verso.

Forge, Code management system Système de gestion de développement collaboratif de logiciel. Considéré comme un assemblage de scripts/de services, une forge comprend plusieurs éléments tels que système de gestion des versions (par exemple, via Git ou Mercurial), gestionnaire de listes de discussion (et/ou de forums), outil de suivi des bugs, gestionnaire de documentation (souvent sur le principe du wiki), gestion des tâches, traduction en ligne.

Forum Espace de discussion publique (ou au moins ouvert à plusieurs participants). Les discussions y sont archivées, ce qui permet une

communication asynchrone (c'est ce qui différencie les forums de la messagerie instantanée tel que l'IRC).

Fondation, Association Une fondation est une personne morale de droit privé à but non lucratif, créée par un ou plusieurs donateurs, eux-mêmes pouvant être des personnes physiques ou morales, pour accomplir une œuvre d'intérêt général. La fondation se distingue de l'association par le fait qu'elle ne résulte pas du concours des volontés de plusieurs personnes pour œuvrer ensemble, mais de l'engagement financier et irrévocable des créateurs de la fondation, qu'il s'agisse de particuliers ou d'entreprises. Une fondation permet avant tout la mise d'argent privé à la disposition d'une cause publique. À l'inverse d'une association, une fondation ne comporte pas de membres. Elle est dirigée par un conseil d'administration, qui comprend les fondateurs, auxquels peuvent aussi s'ajouter des membres de droit et/ou des membres cooptés ou élus. Le terme fondation est également improprement employé pour traduire la notion de *foundation* anglo-saxonne, organisation à but non lucratif proche du modèle associatif français. Les fondations (*foundation*) américaines sont mi-publiques (car elles visent à l'intérêt général et n'obéissent pas aux règles du marché), mi-privées (par leur capital, leur fonctionnement et leurs méthodes).

Leader Il s'agit le plus souvent du membre le plus influent, le responsable « suprême » de l'équipe principale.

Mailing list Une liste de diffusion ou liste de distribution est une utilisation spécifique du courrier électronique qui permet le publipostage (mailing) d'informations aux utilisateurs qui y sont inscrits. Lorsque les membres peuvent répondre sur cette liste, on parle de liste de discussions.

Maintainer Un mainteneur est généralement une ou plusieurs personnes responsables de la maintenance d'un code source - souvent distribué, auprès d'un plus large public, sous la forme d'un paquet binaire.

Management office, bureau exécutif Il s'agit d'un petit groupe de personnes faisant partie d'un ensemble constitué, investi d'un pouvoir de surveillance, de décision (source cnrtl.fr).

Middleware En architecture informatique, un *middleware* (intergiciel) est un logiciel tiers qui crée un réseau d'échange d'informations entre différentes applications informatiques. Le réseau est mis en œuvre par l'utilisation d'une même technique d'échange d'informations dans toutes les applications impliquées à l'aide de composants logiciels.

Podcast Le podcasting est un moyen de diffusion de fichiers (audio, vidéo ou autres) sur Internet appelés « podcasts » (ou « balados » au Canada).

Release Dans le domaine informatique, il s'agit de la version d'un logiciel, par exemple *Release Candidate* - une version finale de test avant publication auprès d'un large public.

RFID La radio-identification, le plus souvent désignée par le sigle RFID (de l'anglais *radio frequency identification*), est une méthode pour mémoriser et récupérer des données à distance en utilisant des marqueurs appelés « radio-étiquettes » (« RFID tag » ou « *RFID transponder* » en anglais). Les radio-étiquettes sont de petits objets, tels que des étiquettes autoadhésives, qui peuvent être collés ou incorporés dans des objets ou produits et même implantés dans des organismes vivants (animaux, corps humain). Les radio-étiquettes comprennent une antenne associée à une puce électronique qui leur permet de recevoir et de répondre aux requêtes radio émises depuis l'émetteur-récepteur.

RSS feeds RSS (sigle venant de l'anglais *Really Simple Syndication*) est une famille de formats de données utilisés pour la syndication de contenu Web. Un produit RSS est une ressource du World Wide Web dont le contenu est produit automatiquement (sauf cas exceptionnels) en fonction des mises à jour d'un site Web. Les flux RSS sont des fichiers XML qui sont souvent utilisés par les sites d'actualité et les blogs pour présenter les titres des dernières informations consultables.

Webinar Un *Webinar* est une conférence en ligne (application Internet) qui offre la possibilité d'organiser des conférences, des réunions de travail ou des formations virtuelles avec des personnes distantes. C'est un outil de collaboration synchrone (tous les participants voient et entendent la même chose au même moment).

Wiki Un wiki est une application web qui permet la création, la modification et l'illustration collaboratives de pages à l'intérieur d'un site web. Il utilise un langage de balisage et son contenu est modifiable au moyen d'un navigateur web. C'est un outil de gestion de contenu, dont la structure implicite est minimale, tandis que la structure explicite émerge en fonction des besoins des usagers.

Workshop Atelier de travail et d'échange sur un sujet choisi à l'avance. Le mot colloque est un synonyme qui vous est peut-être plus familier. L'objectif du *workshop* est d'être un lieu de discussion où des spécialistes exposent leurs travaux devant un public composé aussi bien de personnes confirmées que de novices intéressés par le sujet (source Etudiants.ch).

Crédits

Voici la liste des personnes, articles et références qui nous ont permis d'écrire ce guide :

— Stéphane RIBAS & Michel CEZON, « How to build an OSS community from scratch », *ACM Special for Terena 25 years Anniversary*. 2011. tnc2011.terena.org.
— Pioneers of Change, *Building Communities of Practice : A summary guide*, 2003. pioneersofchange.net.
— Daniel W. RASMUS, *Best Practices : Starting and Driving Communities of Practice*, 2003, Planning Assumption. gigaweb.com.
— Stan GARFIELD, *Implementing a Successful KM Programme*. London : Ark Group. 2007.
— Mike ROWLAND, « B2B Communities - What works », Online Community Unconference. 2009.
— McDermott Consulting : Community of Practice
— The Cathedral and the Bazaar, Eric RAYMOND, 1997.
— Communities of practice : learning, meaning, an identity. E. WENGER, Cambridge University Press 1998.
— Cultivating Communities of Practice by E. WENGER, R. MCDERMOTT & W. M. SNYDER, 1998.
— The Community Round Table

— Exploiting Social Software to Build Open Source Communities by Imed HAMMOUDA, Timo AALTONEN and Petri SIRKKALA, Department of Software Systems, 2009.
— How to create an open source community, Jono BACON.
— 4th IEEE Services and Software & OSW 09 Sweden, « How to kill a community », Stephane RIBAS, Michel Cezon.
— Framework for Governance in Open Source Communities by C. LATEMANN, S. STIEGLITZ, Postdam University. 2005.
— FOSS Governance Fundamentals, 2008.
— fOSSa, the Free Open Source Software Academia Conference, Inria, 2009-2014.
— « Les modèles économiques du logiciel libre et leur évolution » in Histoires et cultures du libre, des logiciels partagés aux licences échangées, Collectif, Framabook, 2013, par Stéphane UBEDA, Patrick GUILLAUD et Stéphane RIBAS
— Open source software development methodology by OSS Watch, 2007-2011.
— A Proposal for Recognizing Reputation within Communities of Practice, C. CRUZ, M. GOUVÊA, C. MOTTA, F. SANTORO, UFRJ, UNIRIO, Brazil, 2008.
— Producing Open Source Software, Karl FOGEL.
— Project Management in Free Software Projects by Martin MICHLMAYR, 2010, lien 1 et lien 2
— Modèles économiques des logiciels open source et logiciels libres, Fabernovel Consulting, 2007.
— Reputation, trust and the dynamics of leadership in communities of practice, P. MULLER, 2006
— ROI et présence sur les médias sociaux, outils d'évaluation, translated from MITSLOAN Management Review Can you Measure the ROI of Your Social Media Marketing et Can you measure the ROI of your social media marketing

Nous remercions les personnes, souvent expertes du domaine, qui nous ont prodigué leur conseils avant et pendant la rédaction de ce guide :

— Adrian BOWYER (RepRap)
— Benjamin JEAN (Inno³ SAS)
— Bernard ESPIAU (Inria)

— Christophe MASUTTI (Framasoft)
— Dave NEARY (REDHAT)
— David CUARTIELLES (Arduino)
— Didier DONSEZ (Polytech Grenoble)
— Emmanuel GILLOZ (FoldaRap)
— Francois ELIE (ADULLACT)
— Gabrielle RUFATTI (ENGINEERING Group)
— Laurence GOUSSU (Inria)
— Maxence GUESDON (Inria)
— Mark ATWOOD (Openstack)
— Mark SOMERFIELD (Creative Innovation Lab)
— Martine COURBIN (Inria)
— Matteo CANTARELLI (Open Worm)
— Nadine PAOLUCCI (CEA)
— Nancy WHITE (Full Circle)
— Perrick PENET-AVEZ (NOPARKING)
— Ross GARDLER (Apache Foundation)
— Sebastien HEYMANN (GEPHI Consortium)
— Simon PHIPPS (OSI)
— Vincent QUINT (Inria)

Nous sommes également reconnaissants envers les membres de plusieurs communautés qui nous ont permis d'expérimenter et de mettre au point les méthodes décrites dans ce guide :

— Aspire
— Aversive++
— Ebiogenouest
— EZTrace
— I-Score
— MOOCLAB
— MAGNET
— MMGTools
— Natron
— PHARO
— Poppy-Project
— Qualipso
— Reador.net

— RMOD
— Sofa
— le réseaux interne social IRIS d'Inria
— et celles que nous avons omis de citer…

Nous remercions aussi notre institut de recherche qui a encouragé la rédaction de ce guide et qui a soutenu nos efforts pour comprendre les communautés. Cela nous a permis d'élaborer et de tester cette méthodologie susceptible de gérer les communautés de façon plus efficiente.

Enfin, nous remercions l'équipe de Framabook pour leur aide précieuse sans laquelle nous aurions eu du mal à propulser ce guide jusqu'à vous.

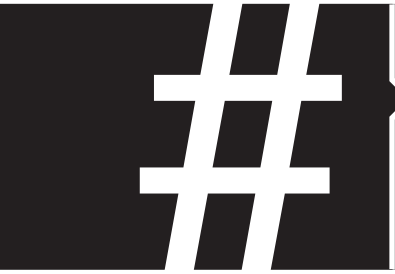


Table des matières