Prévoir l'avenir d'un projet open source (Libres conseils 4/42)

Traduction Framalang : ga3lig, Coco, Aa, Lignusta, goofy, jcr83, peupleLa (relectures), Sylvain, CoudCoud, lamessen + Julius22

Préparez-vous pour le futur : l'évolution des équipes dans le logiciel libre et *open source*

par Felipe Ortega

Felipe Ortega est chercheur et chef de projet à Libresoft, un groupe de recherche de l'Université Rey Juan Carlos [1] *en Espagne. Il développe de nouvelles méthodologies pour analyser les communautés collaboratives ouvertes (comme les projets de logiciels libres, Wikipédia et les réseaux sociaux). Il a mené des recherches approfondies sur le projet Wikipédia et sa communauté de contributeurs. Felipe participe activement à la recherche, la promotion et l'éducation/formation sur le logiciel libre, plus particulièrement dans le cadre du Master « Logiciel libre » de l'URJC* [2]*. C'est un fervent défenseur de l'ouverture*

des ressources éducatives, du libre accès aux publications scientifiques et de l'ouverture des données scientifiques.

Dans son célèbre essai *La Cathédrale et le Bazar* [1], Eric S. Raymond souligne l'une des plus importantes leçons que chaque développeur doit apprendre : « Un bon logiciel commence toujours par un développeur qui gratte là où ça le démange ». Vous ne pouvez comprendre à quel point cette phrase est vraie avant d'avoir vous-même vécu la situation. En fait, la majorité des programmeurs de logiciels libres et *open source* (si ce n'est tous) est certainement passée par cette étape où il faut mettre les mains dans le cambouis sur un tout nouveau projet, ou en rejoindre un, désireux d'aider à le rendre meilleur. Cependant, de nombreux développeurs et autres participants dans les communautés libres et *open source* (rédacteurs de documentation, traducteurs etc.) négligent généralement une autre importante leçon soulignée par Raymond plus loin dans son essai : « Quand un programme ne vous intéresse plus, votre dernier devoir à son égard est de le confier à un successeur compétent ». C'est le thème central que je veux traiter ici. Vous devez penser à l'avenir de votre projet, et aux nouveaux arrivants qui un jour prendront le relais et continueront de le faire avancer.

Le relais entre les générations

Tôt ou tard, de nombreux projets libres et *open source* devront faire face à un relais générationnel. Les anciens développeurs en charge de la maintenance du code et des améliorations finissent par quitter le projet et sa communauté pour des raisons diverses et variées. Il peut s'agir de problèmes personnels, d'un nouveau travail qui ne laisse pas assez de disponibilités, d'un nouveau projet qui démarre, ou du passage à un autre projet qui semble plus attirant… la liste peut être assez longue.

L'étude du relais générationnel (ou renouvellement des développeurs) dans les projets de logiciel libre et *open source* reste un domaine émergent qui nécessite davantage de recherches pour améliorer notre compréhension de ces situations. En dépit de cela, certains chercheurs ont déjà collecté des preuves objectives qui mettent en lumière certains processus. Pendant l'OSS 2006 (NdT : Conférence sur l'Open Source System [4]), mes collègues Jesús González-Barahona et Gregorio Robles présentèrent un travail intitulé « Le renouvellement des contributeurs dans les projets de logiciel libre ». Dans cette présentation, ils

exposèrent une méthodologie pour identifier les développeurs les plus actifs – généralement connus comme les développeurs principaux – à différents moments, pendant toute la durée d'un projet donné. Ils appliquèrent ensuite cette méthode à l'étude de 21 gros projets, en particulier *Gimp* [5], *Mozilla* [6] et *Evolution* [7]. En bref, ils ont découvert qu'on peut distinguer trois types de projets en fonction du taux de renouvellement des développeurs.

- Les projets avec des dieux du code : ces projets reposent en grande partie sur le travail de leurs fondateurs et le relais générationnel est très faible, voire nul. *Gimp* se classe dans cette catégorie.
- Les projets avec de multiples générations de codeurs : des projets comme Mozilla montrent clairement un modèle de renouvellement des développeurs, avec de nouveaux groupes actifs qui prennent en main la gestion du développement et de la maintenance du code des mains mêmes du noyau des anciens contributeurs.
- Les projets composites : *Evolution* appartient à une troisième catégorie de projets ; il connaît un certain taux de renouvellement, mais celui-ci n'est toutefois pas aussi marqué que pour les projets de la catégorie précédente, parce qu'atténué par la rétention de certains des principaux contributeurs au cours de l'histoire du projet.

Cette classification nous amène à une question évidente : quel est le modèle le plus fréquemment rencontré dans les projets de logiciels libre et *open source* ? Pour tout dire, les résultats de l'analyse menée sur l'échantillon de 21 projets lors de ces travaux établissent clairement cette conclusion : ce sont les projets à multiples générations, ainsi que les projets composites qui sont les plus communément rencontrés dans l'écosystème des projets libres et *open source*. Seuls *Gnumeric* et *Mono* ont montré un modèle distinct avec une forte rétention d'anciens développeurs, ceci indiquant que les personnes contribuant à ces projets auraient de plus fortes raisons de continuer leurs travaux sur le long terme.

Cependant ce n'est pas le cas le plus courant. Au contraire, cette étude donne plus de légitimité au conseil suivant : nous devons préparer, à plus ou moins long terme, le transfert de notre rôle et de nos connaissances au sein du projet vers les futurs contributeurs qui rejoignent notre communauté.

Le fossé de connaissances

Toute personne faisant face à un changement significatif dans sa vie doit s'adapter à de nouvelles conditions. Par exemple, quand vous quittez votre emploi pour un autre, vous vous préparez à une période pendant laquelle il vous faudra vous intégrer dans un autre groupe de travail, dans un nouveau lieu. Heureusement, au bout d'un moment vous aurez pris vos marques dans ce nouvel emploi. Mais parfois vous aurez gardé de bons amis de votre ancien boulot, et vous vous reverrez après avoir bougé. Parfois alors, en discutant avec des anciens collègues, vous pourrez savoir ce qu'il est advenu avec la personne recrutée pour vous remplacer. Cela ne se produit que rarement dans les projets *open source*.

Le revers du relais générationnel dans un projet libre peut apparaitre sous une forme très concrète, à savoir un fossé de connaissances. Quand un ancien développeur quitte le projet, et particulièrement s'il avait une expérience approfondie dans cette communauté, il laisse derrière lui ses connaissances aussi bien concrètes qu'abstraites, qui ne sont pas forcément transmises aux nouveaux venus.

Un exemple évident, est le code source. Comme dans toute production intellectuelle bien faite – du moins, c'est ce à quoi on pourrait s'attendre, non ? – les développeurs laissent une marque personnelle chaque fois qu'ils écrivent du nouveau code. Parfois, vous avez l'impression d'avoir une dette éternelle envers le programmeur qui a écrit ce code élégant et soigné qui parle de lui-même et qui est facile à maintenir. D'autres fois, la situation est inverse et vous bataillez pour comprendre un code très obscur sans un seul commentaire ni indice pour vous aider.

C'est ce que l'on a essayé de mesurer en 2009, dans une recherche présentée à l'HICSS 2009 (NdT : Hawaii International Conference on System Sciences [8]). Le titre en est « Utiliser l'archéologie logicielle pour mesurer les pertes de connaissances provoquées par le départ d'un développeur ». Au cas où vous vous poseriez la question, cela n'a rien à voir avec des histoires de fouet, de trésors, de temples, et autres aventures palpitantes. Ce qui a été mesuré, entre autres choses, c'est le pourcentage de code orphelin laissé par les développeurs ayant quitté des projets libre et/ou *open source*, et qu'aucun des développeurs actuels n'a encore repris. Pour cette étude, nous avons choisi quatre projets (*Evolution,*

GIMP, Evince et *Nautilus*) pour tester la méthode de recherche. Et nous sommes arrivés à des résultats assez intéressants.

Evolution présentait une tendance plutôt inquiétante car le taux de code orphelin augmentait au cours du temps. En 2006, près de 80 % de l'ensemble des lignes de code avaient été abandonnées par les précédents développeurs et étaient restées intouchées par le reste de l'équipe. À l'opposé, *Gimp* affichait un modèle tout à fait différent, avec une volonté claire et soutenue par l'équipe de développement de réduire le nombre de lignes orphelines. Souvenons-nous au passage que *Gimp* avait déjà été qualifié de projet des dieux du code et bénéficiait donc d'une équipe de développement bien plus stable pour surmonter cette tâche harassante.

Cela signifie-t-il que les développeurs de *Gimp* avaient une bien meilleure expérience que ceux d'*Evolution* ? Honnêtement, on n'en sait rien. Néanmoins, on peut prévoir un risque clair : plus le taux de code orphelin est élevé, plus l'effort à fournir pour maintenir le projet est important. Que ce soit pour corriger un bogue, développer une nouvelle fonctionnalité ou en étendre une préexistante, il faut faire face à du code que l'on n'a jamais vu auparavant. Bien sûr les programmeurs d'exception existent, mais peu importe à quel point l'on est merveilleux, les développeurs de *Gimp* ont un avantage certain ici, puisqu'ils ont quelqu'un dans l'équipe qui a une connaissance précise de la majorité du code à maintenir. De plus, ils travaillent à réduire la portion de code inconnu au cours du temps.

C'est comme à la maison

Ce qui est intéressant, c'est que certains projets parviennent à retenir les utilisateurs sur des périodes bien plus longues qu'on aurait pu s'y attendre. Là encore, nous pouvons trouver des preuves empiriques justifiant cette déclaration. Pendant l'OSS 2005 [9], Michlmayr, Robles et González-Barahona présentèrent des résultats pertinents concernant cet aspect. Ils étudièrent la persistance de la participation des responsables de logiciels sur Debian en calculant ce qu'on appelle la demi-vie. C'est le temps nécessaire à une population donnée de développeurs principaux pour perdre la moitié de sa taille initiale. Le résultat fut que la demi-vie estimée des responsables Debian était approximativement de 7 ans et demi. En d'autres termes, l'étude ayant été menée sur une période de six ans et demi (entre juillet 1998 et décembre 2004), donc depuis Debian 2.0 jusqu'à

Debian 3.1 (versions stables uniquement), plus de 50 % des responsables de Debian 2.0 contribuaient encore à Debian 3.1.

Debian a créé une sorte de procédure très formelle pour accepter de nouveaux codeurs logiciels (aussi connus sous le nom de développeurs Debian) qui inclut l'acceptation du Contrat Social Debian et la démonstration d'une bonne connaissance de la Politique Debian. Résultat, on peut s'attendre à avoir des contributeurs très engagés. C'est en effet le cas, puisque les auteurs de l'étude ont constaté que les paquets délaissés par les anciens développeurs étaient généralement repris par d'autres développeurs de la communauté. C'est seulement dans le cas où le paquet n'était plus utile qu'il a été abandonné. Je pense que nous pouvons tirer quelques conclusions de ces travaux de recherche :

1. Passez du temps à développer les principales lignes directrices de votre projet. Cela peut commencer par un seul et court document, qui détaille simplement des recommandations et des bonnes pratiques. Cela peut évoluer à mesure que le projet grandit, et permettre aux nouveaux arrivants tant de saisir rapidement les valeurs principales de votre équipe, qu'à comprendre les traits principaux de votre méthodologie.
2. Forcez-vous à suivre des standards de codage, des bonnes pratiques et un style élégant. Documentez votre code. Insérez des commentaires pour décrire les sections qui seraient particulièrement difficiles à comprendre. Ne pensez pas que c'est du temps perdu. En fait, vous faites preuve de pragmatisme en investissant du temps dans l'avenir de votre projet.
3. Dans la mesure du possible, lorsque le moment vient pour vous de quitter le projet, essayez d'avertir les autres de cette décision longtemps à l'avance. Assurez-vous qu'ils comprennent quelles parties essentielles du code nécessiteront un nouveau développeur pour le maintenir. Idéalement, si vous formez une communauté, préparez au moins une procédure simple afin d'automatiser la transition, et assurez-vous de n'oublier aucun point important avant que la personne ne quitte le projet (particulièrement si celle-ci est un développeur clé).
4. Gardez un œil sur la quantité de code orphelin. Si celle-ci augmente trop rapidement, ou si elle atteint une trop grande proportion de votre projet, cela indique clairement que vous allez avoir des problèmes dans peu de temps, en particulier si le nombre de rapports de bogues augmente ou si vous envisagez une nouvelle implémentation de votre code avec de

fortes modifications.
5. Assurez-vous de toujours laisser assez d'astuces et de commentaires pour qu'à l'avenir un nouvel arrivant puisse s'approprier votre travail.

J'aurais voulu savoir que vous arriviez (avant de partir)

Je reconnais que ce n'est pas très facile de penser à ses successeurs lorsque vous programmez. La plupart du temps, vous ne vous rendez tout simplement pas compte que votre code pourrait à la fin être repris par un autre projet, réutilisé par d'autres personnes ou que vous pourriez éventuellement être remplacé par un autre, qui continuera votre travail après vous. Cependant, le plus remarquable atout des logiciels libres et *open source* est précisément celui-là : le code sera réutilisé, adapté, intégré ou exporté par quelqu'un d'autre. La maintenance est une composante essentielle de l'ingénierie logicielle. Mais cela devient primordial dans le cas des logiciels libres et *open source*. Ce n'est pas seulement une question de code source. Cela concerne aussi les relations humaines et la netiquette. C'est quelque chose qui va au-delà du bon goût. *Quod severis metes* (« On récolte ce que l'on a semé »). Souvenez-vous en. La prochaine fois, vous pourriez être le nouveau venu qui viendra combler le vide de connaissance laissé par un ancien développeur.

[1] http://www.urjc.es

[2] http://www.urjc.es/estudios/masteres_universitarios/ingenieria/software_libre/index.htm

[3] http://www.linux-france.org/article/these/cathedrale-bazar/cathedrale-bazar.html

[4] http://oss2006.org

[5] logiciel de création graphique, http://www.gimp.org

[6] https://www.mozilla.org/fr/firefox/fx/

[7] logiciel de messagerie, http://projects.gnome.org/evolution

[8] archives de la conférence, http://www.informatik.uni-trier.de/~ley/db/conf/hicss/hicss2009.html

[9] http://oss2005.case.unibz.it

Les projets open source s'épanouissent à l'air libre (Libres conseils 3/42)

3. Hors du labo, au grand air

La croissance des communautés open source autour des projets universitaires

par Markus Kroëtzsch

Markus Kroëtzsch est post-doctorant au Département des Sciences Informatiques de l'Université d'Oxford. Il a soutenu son doctorat en 2010 à l'Institut d'Informatique Appliquée et Méthodes Formelles de description du Karlsruhe Institute of Technology. Ses recherches portent sur le traitement automatique de l'information, depuis les fondements de la représentation de la connaissance formelle jusqu'à leurs domaines d'application, tel le Web Sémantique. Il est le développeur principal de la plate-forme Semantic MediaWiki, application de Web sémantique, co-éditeur des spécifications W3C OWL2, administrateur principal du portail communautaire semanticweb.org, et co-auteur de l'ouvrage Foundations of Semantic Web Technologies.

Au sein des universités, les chercheurs développent de grandes quantités de logiciels, que ce soit pour valider une hypothèse, pour illustrer une nouvelle

approche, ou tout simplement comme outil en appui à une étude. Dans la plupart des cas, un petit prototype dédié fait le travail, et il est déployé rapidement tandis que l'enjeu de la recherche évolue. Cependant, de temps à autres, une nouvelle approche ou une technologie émergente a le potentiel de changer complétement la manière de résoudre un problème. Ce faisant, cela génère de la réputation professionnelle, du succès commercial, et la gratification personnelle d'amener une nouvelle idée à son plein potentiel. Le chercheur qui a fait une découverte de ce genre est alors tenté d'aller au-delà du prototype vers un produit qui sera réellement utilisé. Il est alors confronté à une toute nouvelle série de problèmes pratiques.

La peur de l'utilisateur

Dans l'un de ses célèbres essais sur l'ingénierie logicielle, Frederick P. Brooks, Jr. permet de se faire une bonne idée des efforts liés à la maintenance d'un vrai logiciel et nous met en garde contre l'utilisateur :

« Le coût total de la maintenance d'un programme largement utilisé est habituellement de 40% ou plus de son coût de développement. De façon surprenante, ce coût est fortement influencé par le nombre d'utilisateurs. Plus il y a d'utilisateurs, plus il y a de bugs » [1]

Bien que les chiffres soient probablement différents dans le contexte actuel, la remarque reste fondamentalement vraie. Elle pourrait même avoir été confirmée par l'usage généralisé de la communication instantanée. Pire encore : davantage d'utilisateurs ne produit pas seulement davantage de bugs ; en général, ils expriment aussi plus de demandes. Qu'il s'agisse d'une véritable erreur, d'une demande de fonctionnalité, ou tout simplement d'une mauvaise compréhension du fonctionnement du logiciel, les demandes de l'utilisateur lambda ne ressemblent en rien à un rapport de bug précis et technique. Et chaque demande requiert l'attention des développeurs et occupe un temps précieux qui n'est plus disponible pour écrire du code.

Avec son esprit d'analyse, le chercheur anticipe sur ce problème. Dans sa lutte naturelle pour éviter un avenir sombre dans le service client, il peut carrément développer de la peur envers l'utilisateur. Dans le pire des cas, cela peut le mener à prendre des décisions qui vont à l'encontre du projet dans son ensemble ; sous

des formes plus légères, cela peut tout de même mener le chercheur à cacher des produits logiciels brillants à ses utilisateurs potentiels. Plus d'une fois, j'ai entendu des chercheurs dire : « Nous n'avons pas besoin de plus de visibilité : nous recevons déjà suffisamment d'emails ! ». Il est vrai que parfois la communication pour un outil logiciel nécessite un effort supérieur à celui que peut fournir un chercheur sans laisser tomber son emploi principal.

Pourtant, cette issue tragique aurait bien souvent pu être évitée. Brooks pouvait difficilement l'anticiper. Quand il a écrit ses essais, le fait est que les utilisateurs étaient des clients et que la maintenance logicielle faisait partie du produit qu'ils achetaient. Il fallait trouver un équilibre entre l'effort de développement, la demande du marché, et le prix. C'est toujours le cas pour de nombreux produits logiciels commerciaux de nos jours, mais ça a peu de choses à voir avec la réalité du développement à petite échelle de l'*open source*. Les utilisateurs habituels de l'*open source* ne paient pas pour le service qu'ils reçoivent. Leur attitude n'est donc pas celle d'un client exigeant, mais bien plus souvent celle d'un partisan enthousiaste et reconnaissant. Transformer cet enthousiasme en un soutien plus que nécessaire n'est pas négligeable pour réussir dans l'art de la maintenance d'un logiciel open source : l'intérêt croissant de l'utilisateur doit aller de pair avec une contribution croissante.

Reconnaitre que les utilisateurs de logiciels *open source* ne sont pas que « des consommateurs qui ne paient pas » est une notion importante. Mais cela ne doit pas mener à surestimer leur potentiel. Le contre-pied optimiste de la peur irrationnelle de l'utilisateur est la croyance que des communautés actives croissent naturellement avec pour seule base la licence choisie pour publier le code. Cette grave erreur de jugement est bizarrement toujours aussi commune, et a scellé le destin de bien des tentatives de création de communautés ouvertes.

Semer et récolter

Le pluriel d' « utilisateur » n'est pas « communauté ». Si l'un peut s'accroître en nombre, l'autre ne grandit pas d'elle-même, ou alors elle grandit sans direction et surtout sans fournir le soutien espéré au projet. La mission du responsable de projet qui cherche à profiter de l'énergie brute des utilisateurs ressemble à celle d'un jardinier qui doit préparer un terrain fertile, planter et arroser les semis, et peut-être élaguer les pousses non désirées avant de pouvoir récolter les fruits.

Par rapport aux récompenses, l'investissement global est minime, mais il est essentiel de faire les bonnes choses, au bon moment.

Préparer le support technologique

Créer une communauté commence avant même que le premier utilisateur n'apparaisse. D'emblée, le choix du langage de programmation va déterminer le nombre de personnes qui pourront déployer et déboguer notre code. Objective Caml est sans doute un beau langage, mais si l'on utilise plutôt Java, la quantité d'utilisateurs et de contributeurs potentiels augmentera de plusieurs ordres de grandeur. Les développeurs doivent donc faire des compromis puisque la technologie la plus répandue est rarement la plus performante ou la plus élégante. Cela peut être une démarche particulièrement difficile pour des chercheurs qui privilégient souvent la supériorité formelle du langage. Quand je travaillais sur Semantic MediaWiki, on m'a souvent demandé pourquoi nous utilisions PHP alors que Java côté serveur serait tellement plus propre et performant. Comparons la taille de la communauté de Semantic MediaWiki et les efforts que demanderait le même développement basé sur du Java : voilà peut-être un début de réponse. Cet exemple illustre aussi que l'audience ciblée détermine le meilleur choix de la technologie de base. Le développeur lui-même devrait avoir le recul nécessaire pour prendre la décision la plus opportune.

Préparer minutieusement le terrain

Dans le même ordre d'idées, il faut créer un code lisible et bien documenté dès le début. Dans un environnement universitaire, certains projets logiciels rassemblent de nombreux contributeurs temporaires. Les changements dans les plannings et les projets des étudiants peuvent nuire à la qualité du code. Je me souviens d'un petit projet de logiciel à l'Université technique de Dresde qui avait été très bien maintenu par un assistant étudiant. Après son départ, on a constaté que le code était minutieusement documenté… en turc. Un chercheur ne peut être programmeur qu'à temps partiel, une discipline particulière est donc nécessaire pour mettre en œuvre le travail supplémentaire indispensable à l'élaboration d'un code accessible. En retour il y aura de bien meilleures chances par la suite d'avoir de bons rapports de bogues, des patchs utiles ou même des développeurs extérieurs.

Semer les graines des communautés

Les développeurs *open source* inexpérimentés considèrent souvent comme un grand moment la publication ouverte de leur code. En réalité, personne d'autre qu'eux ne la remarquera. Pour attirer aussi bien des utilisateurs que des contributeurs, il faut faire passer le mot. La communication publique d'un vrai projet devrait au moins inclure des annonces à chaque nouvelle version. Les listes de diffusion sont probablement le meilleur canal pour cela. Il faut un certain talent social pour trouver le juste équilibre entre le spam indésirable et la litote timide. Si un projet est motivé par la conviction sincère qu'il aidera les utilisateurs à résoudre de vrais problèmes, ce devrait être facile de lui faire une publicité convenable. Les utilisateurs feront vite la différence entre publicité éhontée et information utile. Bien évidemment, les annonces actives devront attendre que le projet soit finalisé. Cela ne concerne pas seulement le code, mais aussi la page d'accueil et la documentation d'utilisation basique.

Au cours de sa vie, le projet devrait être mentionné dans tous les endroits adéquats, y compris des sites web — à commencer par votre page d'accueil ! — des conférences, des papiers scientifiques, des discussions en ligne. On ne dira jamais assez le pouvoir du simple lien qui conduira un futur contributeur important sur le site du projet dès sa première visite. Les chercheurs ne doivent pas non plus oublier de publier leur logiciel en dehors de leur communauté universitaire proche. Les autres chercheurs sont rarement la meilleure base pour une communauté active.

Fournir des espaces pour grandir

Banal mais souvent négligé, le devoir des personnes qui maintiennent le projet est de fournir des espaces de communication afin que la communauté puisse se développer. Si un projet n'a pas de liste de discussion dédiée, alors toutes les demandes d'aide seront envoyées en message privé à la maintenance. S'il n'y a pas de bugtracker (NdT : logiciel de suivi de problèmes), les rapports de bogues seront moins nombreux et moins utiles. Sans un wiki éditable par tout un chacun pour la documentation utilisateur, le développeur est condamné à étendre et à réécrire la documentation en permanence. Si la version de développement du code source n'est pas accessible, alors les utilisateurs ne seront pas dans la capacité de tester la dernière version avant de se plaindre de problèmes. Si le

dépôt de code est intrinsèquement fermé, il n'est alors pas possible d'accueillir des contributeurs externes. Toute cette infrastructure est disponible gratuitement par l'intermédiaire d'un certain nombre de fournisseurs de service. Toutes les formes d'interaction ne sont pas forcément désirées, par exemple, il y a des raisons de garder fermé le cercle des développeurs. Mais il serait inconscient d'espérer le soutien d'une communauté sans même préparer des espaces de base pour celle-ci.

Encourager et contrôler la croissance

Les développeurs inexpérimentés sont souvent préoccupés par le fait que l'ouverture de listes de diffusions, de forums et de wikis pour les utilisateurs nécessitera une maintenance supplémentaire. C'est rarement le cas, mais certaines activités de base sont bien entendu indispensables. Cela commence par la mise en œuvre rigoureuse des usages de communication publique. Les utilisateurs ont besoin d'apprendre à poser des questions publiquement, à consulter la documentation avant de poser des questions, et à rapporter les bogues à l'aide du bugtracker plutôt que par e-mail. J'ai tendance à rejeter toutes les demandes d'aide privées, ou à répondre via des listes publiques. Cela garantit au passage que les solutions sont disponibles sur le web pour les futurs utilisateurs qui les chercheront. Dans tous les cas, les utilisateurs doivent être remerciés explicitement pour toutes les formes de contributions : il faut beaucoup d'enthousiasme et des gens bien intentionnés pour construire une communauté solide.

Quand on atteint un certain nombre d'utilisateurs, une aide mutuelle commence à se mettre en place entre eux. C'est toujours un moment magique pour un projet et c'est un signe évident qu'il est sur la bonne voie. Dans l'idéal, les responsables du projet devraient continuer d'apporter leur aide pour les questions délicates, mais à un moment donné certains utilisateurs vont faire preuve d'initiative dans les discussions. Il est important de les remercier (personnellement) et de les impliquer d'avantage dans le projet. À l'inverse, les évolutions malsaines doivent être stoppées dès que possible, en particulier les comportements agressifs qui peuvent être un véritable danger pour le développement de la communauté. De même, l'enthousiasme le mieux intentionné n'est pas toujours productif et il faut parfois savoir dire non — gentiment mais clairement – pour éviter les dérapages possibles.

Le futur est ouvert

Construire une communauté initiale autour d'un projet contribue fortement à transformer un prototype de recherche en un logiciel open source. Si ça porte ses fruits, il existe de nombreuses options pour le développer en fonction des buts fixés par le mainteneur du projet et la communauté. Voici quelques indications :

Persévérer dans l'expansion du projet et de sa communauté open source, augmenter le nombre de personnes ayant des droits de contributions « directs », en réduisant la dépendance à son origine universitaire. Par ce biais, vous impliquez plus fortement la communauté – notamment au travers d'événements dédiés –;et vous pourrez établir un soutien organisationnel.

Créer une entreprise commerciale pour exploiter le projet, basée, par exemple, sur une double licence ou un *business model* de consultant. Des outils ayant fait leurs preuves et une communauté active sont des atouts majeurs dès le lancement d'une entreprise, et peuvent être bénéfiques dans chacune de vos stratégies d'entreprise sans abandonner le produit original open source.

Se retirer du projet. Il y a de nombreuses raisons pour qu'on ne puisse plus maintenir de lien avec le projet. Le fait d'avoir établi une communauté saine et ouverte maximise les chances pour que le projet continue de voler de ses propres ailes. Dans tous les cas, il est plus correct de faire une coupure nette que d'abandonner silencieusement le projet, en le tuant par une activité en chute libre au point qu'on ne trouve plus personne pour le maintenir.

Le profil de la communauté sera différent selon qu'on opte pour telle ou telle stratégie de développement. Mais dans tous les cas, le rôle du chercheur évolue en fonction des objectifs du projet. Le scientifique initial et le programmeur pourront prendre le rôle de gestionnaire ou de directeur technique. En ce sens, la différence principale entre un projet de logiciel open source d'importance et la recherche perpétuelle d'un prototype n'est pas tant la quantité de travail mais le type de travail nécessaire pour y arriver. Cela compte pour beaucoup dans sa réussite. Une fois qu'on l'a compris, il ne reste plus qu'à réaliser un super logiciel.

[1] Frederick P. Brooks, Jr.: The Mythical Man-Month. Essays on Software Engineering. Anniversary Edition. Addison-Wesley, 1995.

Tous les autres pourraient avoir tort, mais c’est peu probable (Libres conseils 2/42)

Tous les autres pourraient avoir tort, mais c’est peu probable

par Evan Prodromou

Evan Prodromou est le fondateur de Wikitravel, StatusNet et du réseau social Open Source Identi.ca. Il participe aux logiciels Open Source depuis 15 ans en tant que développeur, écrit de la documentation et se distingue à l’occasion comme agitateur. Il vit au Québec, à Montréal.

La plus importante caractéristique du fondateur d’un projet Open Source, dans les premières semaines ou premiers mois avant de lancer son logiciel dans le monde, c’est une obstination de tête de mule face à l’écrasante évidence des faits. Si votre logiciel est si important, pourquoi personne ne l’a-t-il déjà écrit ? Peut-être que ce n’est même pas possible. Peut-être que personne d’autre que vous ne veut ce que vous êtes en train de faire. Peut-être que vous n’êtes pas assez bon pour le faire. Peut-être que quelqu’un l’a déjà fait et que vous n’êtes simplement pas assez malin pour le trouver avec Google.

Garder la foi à travers cette longue et sombre nuit est difficile ; seules les têtes de cochons opiniâtres et bornées peuvent y parvenir. Et nous arrivons à l’application de nos opinions de programmeur les plus fortement défendues. Quel est le meilleur langage de programmation à utiliser ? L’architecture de l’application ? Les standards d’écriture du code ? La licence logicielle ? Le système de gestion de version ? Si vous êtes le seul à travailler (ou à connaître !) le projet, vous devez décider, de façon unilatérale.

Quand vous vous lancez finalement, malgré tout, cette détermination bornée et

cette forte opinion sont devenus préjudiciables, pas bénéfiques. Une fois que vous vous êtes lancé, vous aurez besoin de compétences tout à fait à l'opposé pour faire des compromis afin que votre logiciel soit davantage utile aux autres. Et beaucoup de ces compromis vous sembleront vraiment mauvais.

Il est difficile de recevoir des avis d'« étrangers » (c.à.d. des personnes qui ne sont pas vous). D'abord parce qu'ils se focalisent sur des choses si triviales, sans importance (votre convention de nommage des variables par exemple, ou l'emplacement de certain boutons). Ensuite parce qu'ils ont invariablement tort . Après tout, si ce que vous avez fait n'est pas la bonne manière de faire, vous ne l'auriez pas fait ainsi dès le départ. Si votre façon de faire n'était pas la bonne, pourquoi votre code serait si populaire ?

Mais « mauvais » est relatif. Si faire un *mauvais* choix rend votre logiciel plus accessible aux utilisateurs finaux, ou aux « développeurs en aval » ou aux administrateurs ou les empaqueteurs, est-ce que ce n'est pas finalement juste ?

Et la nature de ce genre de commentaires et de contributions est généralement négative. Les retours de la communauté sont principalement des réactions, ce qui implique qu'elles sont critiques. Avez vous déjà rapporté un bug qui disait « J'aime beaucoup l'organisation du module hashtable.c » ou « Bravo d'avoir supprimé ce sous-sous-sous-menu » ? Les gens font un retour d'expérience car ils n'aiment pas la façon dont fonctionne votre logiciel à un instant T. Et ils ne sont pas toujours très diplomates à ce moment-là.

Il est difficile de répondre de façon positive à ce genre de retour. Nous engueulons parfois les expéditeurs sur nos listes des discussions de développement, ou fermons les rapports d'anomalies avec un rictus et un WONTFIX (NdT : NESERAPASRESOLU, employé dans les rapports de bug). Pire encore, nous nous retirons dans notre cocon, ignorant les suggestions externes ou les retours d'expérience, câlinant notre confortable code qui sied parfaitement à nos idées préconçues et à nos marottes.

Si le logiciel n'est que pour vous-même, vous pouvez garder le code source et les infrastructures qui l'entourent comme terrain de jeu personnel. Mais si vous voulez que votre logiciel soit utilisé, qu'il compte pour les autres personnes, qu'il change (peut-être) le monde, alors vous allez devoir construire une saine et solide communauté d'utilisateurs, de contributeurs principaux, d'administrateurs et de

développeurs de modules. Les utilisateurs ont besoin d'avoir l'impression de posséder le logiciel, de la même façon que vous.

Il est difficile de se rappeler que chacune de ces voix dissidentes ne représente qu'une infime minorité. Imaginez tous les gens qui entendent parler de votre logiciel qui ne prennent jamais le temps de l'essayer. Ceux qui le téléchargent mais ne l'installent jamais. Ceux qui l'installent, restent bloqués, et l'abandonnent en silence. Et ceux qui veulent vous faire un retour, mais qui ne trouvent pas votre système de rapport de bugs, listes de mails de développeurs, canaux IRC ou adresses mails personnelles. Étant donné les difficultés à faire passer leur message, il y a probablement une centaine de personnes qui veulent que des modifications soient faites pour une seule qui parvient à transmettre le message. Donc, être à l'écoute de ces voix, quand elle parviennent à vous atteindre, est essentiel.

Le responsable de projet est chargé de maintenir la vision et la finalité du logiciel. Nous ne pouvons vaciller, en faisant des allers et retours basés sur tel ou tel courriel d'utilisateur pris au hasard. Et s'il y a un principe de base en jeu, alors, bien sûr, il est important de garder cette base stable. Personne d'autre que le responsable de projet ne peut le faire.

Mais nous devons réfléchir : y a-t-il des questions non fondamentales qui puissent rendre le logiciel plus accessible, plus facile d'utilisation ? Finalement, la mesure de notre travail est dans la façon dont nous touchons les utilisateurs, comment notre logiciel est utilisé, et ce pourquoi il est utilisé. À quel point notre idée personnelle importe-t-elle « vraiment » pour le projet et pour la communauté ? Quelle part est uniquement ce que le responsable aime, personnellement ? Si ces problèmes non essentiels existent, alors il faut réduire les désaccords, répondre aux demandes, et faire les changements. Le projet n'en sera que meilleur pour tout le monde.

Du code avant toute chose

Chaque jeudi à 21h, rendez-vous sur le framapad de traduction, le travail collaboratif sera ensuite publié ici même.

Libres conseils (1/42)

Traduction framalang peupleLa, tibs, Astalaseven, aKa, lerouge, Vilnus Atyx, liu qihao, Cyrille L., Khyvodul, jcr83, jcr83, Goofy, Gatien Bovyn, Antoine, lamessen, AlBahtaar + 2 anonymes

Première Partie, Idées et innovations

1. Du code avant toute chose

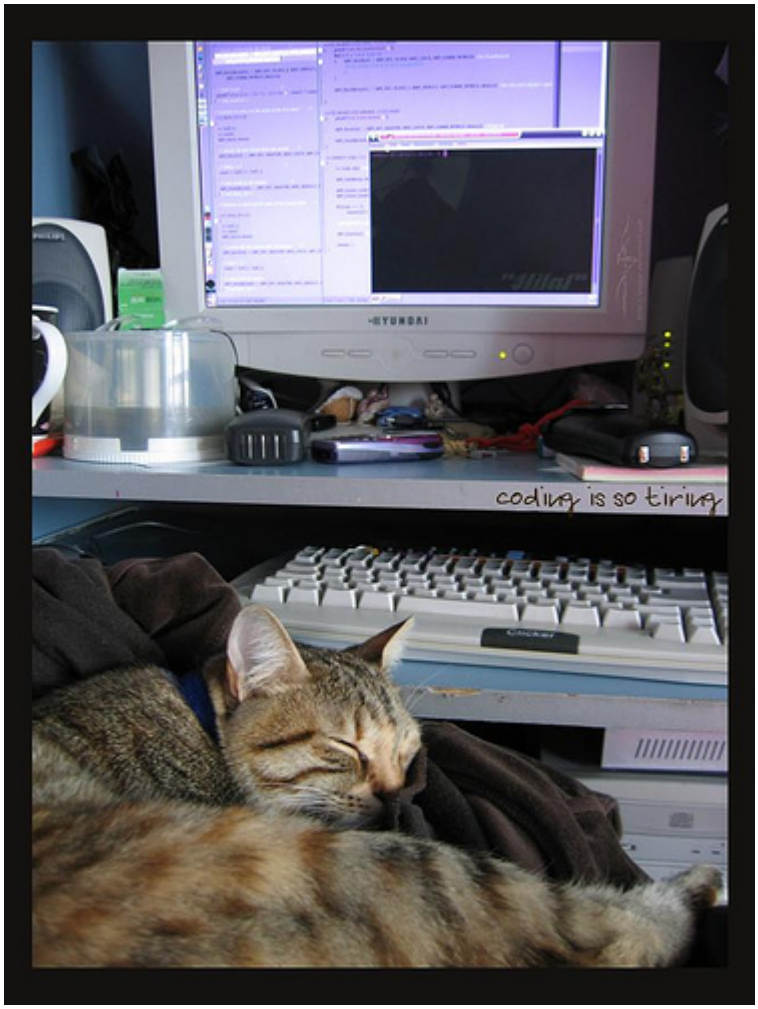
Armijn Hemel a utilisé des logiciels libres depuis 1994, lorsque son frère est revenu à la maison avec une pile de disquettes contenant l'une des premières versions de FreeBSD. Un an après, il migrait vers GNU/Linux, et depuis il n'utilise plus que des systèmes de type Unix, que ce soit chez lui, pour ses études à l'université d'Utrecht ou au travail. Depuis 2005, Armijn est membre du noyau dur de l'équipe de gpl-violations.org tout en possédant son propre cabinet de conseil (Tjaldur Software Governance Solutions) spécialisé dans la détection et la résolution de litiges nés de violations de licences GPL.

En 1999, je faisais tout juste mes premiers pas dans l'activisme du logiciel libre et Open Source. J'utilisais déjà Linux et FreeBSD depuis un certain nombre d'années, mais je n'étais encore qu'un simple utilisateur et je souhaitais apporter une contribution en retour. De mon point de vue, la meilleure manière de le faire était d'écrire du code. Étant donné que je ne trouvais aucun projet existant dans lequel j'aurais été à l'aise pour travailler, j'ai décidé de commencer mon propre projet. Avec le recul, je constate que plusieurs raisons m'ont poussé à débuter ce projet. L'une tenait à mes doutes sur le fait que mon code était d'une qualité suffisante pour être accepté dans un projet existant (je n'étais pas un programmeur brillant et d'ailleurs je ne le suis toujours pas), alors que pour un projet personnel, la question ne se pose pas. La seconde raison est certainement l'arrogance de la jeunesse.

Mon idée était de créer un logiciel de présentation, qui pourrait imiter la plupart des fonctionnalités les plus avancées (ou si vous préférez, les plus pénibles) de PowerPoint. À ce moment-là, OpenOffice.org n'existait pas et le choix était relativement limité : LaTeX et MagicPoint, qui sont davantage orientés vers le contenu textuel que sur les effets d'animation. Je voulais créer un logiciel multi-plateforme, et j'ai pensé que pour remplir cet objectif Java serait le meilleur choix.

Le concept était de faire un logiciel de présentation, écrit en Java, qui aurait intégré tous ces effets animés. Je me suis décidé et j'ai commencé le projet.

Toute l'infrastructure nécessaire était déjà disponible : une liste de diffusion, un site web, un système de gestion de versions (**cvs**). Mais il n'existait aucun code source qui aurait permis à des contributeurs potentiels de travailler directement . La seule chose en ma possession, c'était quelques idées de ce que je voulais faire, une démangeaison à soulager, et des slogans publicitaires séduisants. Je voulais en fait que beaucoup de gens rejoignent le projet pour que celui-ci devienne réellement un projet collaboratif.



J’ai commencé par faire des plans (avec mes nouvelles connaissances en UML) et à les faire circuler. Rien ne s’est passé. J’ai essayé d’impliquer des contributeurs, mais créer une architecture de manière collaborative est très difficile (sans compter qu’a priori, ce n’est sûrement pas le meilleur moyen de créer un logiciel). Après un certain temps, j’ai laissé tomber et le projet est mort en silence, sans qu’une seule ligne de code ait été écrite. Chaque mois je recevais des messages par la liste de diffusion qui me rappelaient que ce projet avait un jour existé, j’ai donc demandé sa mise hors-ligne.

J’en ai tiré une leçon précieuse, quoiqu’un peu douloureuse : dès lors que vous annoncez quelque chose et que vous souhaitez que les gens s’impliquent dans votre projet, assurez-vous au moins qu’il y ait un minimum de code disponible. Peu importe qu’il ne soit pas complètement terminé ; il est acceptable même s’il est mal dégrossi (au début en tout cas). Mais au moins cela démontre l’existence d’une base sur laquelle des contributeurs peuvent travailler et ainsi l’améliorer. Dans le cas contraire, votre projet finira de la même manière que tant d’autres, dont le mien : aux oubliettes.

Finalement, j’ai trouvé mon créneau pour contribuer au progrès du logiciel libre et open source, en m’assurant que les fondements légaux de ceux-ci restent protégés par l’intermédiaire du projet gpl-violations.org. Rétrospectivement, je n’ai jamais utilisé — sans en éprouver de frustration d’ailleurs — les effets animés dans les logiciels de présentation. En fait, je les trouve de plus en plus irritants, ils nous distraient trop du contenu. Pour faire mes présentations, je suis un utilisateur heureux de LaTeX Beamer et occasionnellement — mais avec moins de plaisir — d’OpenOffice.org/LibreOffice.

Crédit photo vampir 42 (CC-BY-SA)

« Libres conseils », une, première !

Qui n’a pas son projet libre ?

Plus qu’une mode ou un engouement passager, c’est un véritable mouvement de

fond depuis quelques années : toute une communauté qui crée, échange, élabore, donne et reçoit des contributions, enfourche de nouveaux projets…

Fort bien, mais…

SourceForge récemment et Github aujourd'hui sont de véritables cimetières de projets libres et open source qui n'ont jamais trouvé d'audience, d'équipe de développement, de communauté active. Rien de bien tragique là-dedans. On peut estimer que ces plateformes sont pour beaucoup de libristes une sorte de terrain de jeu, de laboratoire, d'incubateur où le code et sa documentation s'expérimentent par à-coups, avec l'enthousiasme et l'énergie de ceux qui s'emparent d'un outil pour le mettre au service de leur créativité. Un excellent moyen d'apprendre en *faisant* finalement, à code ouvert. Et qu'importe alors l'absence d'aboutissement dans 80% des cas puisque c'est la démarche qui a été formatrice.

Cependant vous pouvez avoir envie de dépasser le stade du hobbyiste sympathique qui va bricoler son génial projet dans son coin. Vous pouvez avoir le désir de mettre toutes le chances de votre côté pour que le projet libre aboutisse vraiment, gagne en notoriété, entre dans une logique commerciale, vous procure amour, gloire et beauté.

C'est précisément l'intérêt du feuilleton dont vous allez déguster les épisodes semaine après semaine.

42 auteurs vous feront partager leur expérience, avec sérieux et humour, vous raconteront leurs ratages et leurs succès, vous diront comment éviter les uns et atteindre les autres. Des principes, des recommandations mais aussi des trucs et des ficelles, bref une ribambelle chatoyante de libres conseils.

Chaque semaine ou presque, l'équipe framalang vous proposera un nouvel épisode traduit du livre électronique en anglais Open Advice.

Chaque semaine — **top départ chaque jeudi soir à 21h** — une ou deux tranches du gâteau seront proposées à la traduction collaborative sur un framapad, donc en libre accès pour tous ceux qui souhaitent y contribuer. Participez à l'aventure !

La version que nous publierons ensuite ici même, comme dans le premier échantillon ci-dessous qui est une sorte de préambule, est un premier état de la

traduction (donc évidemment perfectible), l'étape suivante sera une révision générale de tous les articles pour les joindre en un Framabook à venir.

Eh oui ça se passe comme ça chez Frama !

Les traducteurs de ce premier round d'échauffement :

peupleLa, Astalaseven, Hideki, Vilnus Atyx, liu qihao, Cyrille L., Khyvodul, jcr83, Slystone, schap2, 4nti7rust, Goofy, Antoine, lamessen + 4 anonymes

Libres Conseils

Logiciels libres et open source : ce que nous aurions aimé savoir avant de commencer

Open Advice est une base de connaissances provenant d'une grande variété de projets de logiciels libres. Elle répond à des questions dont 42 contributeurs majeurs auraient aimé connaître les réponses lorsqu'ils ont débuté. Vous aurez ainsi une longueur d'avance quelle que soit la façon dont vous contribuez et quel que soit le projet que vous avez choisi.

Les projets de logiciels libres modifient le paysage du logiciel de façon impressionnante grâce à des utilisateurs dévoués et une gestion innovante. Chacun apporte quelque chose au mouvement à sa façon, avec ses capacités et ses connaissances. Cet engagement personnel et la puissance du travail collaboratif sur Internet donnent toute leur force aux logiciels libres et c'est ce qui a rassemblé les auteurs de ce livre.

Ce livre est la réponse à la question « Qu'auriez-vous aimé savoir avant de commencer à contribuer ? » Les auteurs offrent un aperçu de la grande variété de talents qu'il faut rassembler pour réussir un projet de logiciel : le codage bien sûr, mais aussi le design, la traduction, le marketing et bien d'autres compétences. Nous sommes là pour vous donner une longueur d'avance si vous êtes nouveau. Et si ça fait déjà un moment que vous contribuez, nous sommes là pour vous donner un aperçu d'autres domaines et projets.

pour les géants et ceux qui se tiendront sur leurs épaules [1]

Avant-propos

Ce livre parle de communauté et de technologies. Il est le fruit d'un travail collectif, un peu comme la technologie que nous construisons ensemble. Si c'est votre première rencontre avec notre communauté, vous pourrez trouver étrange de penser qu'une communauté puisse être le moteur qui propulse la technologie. La technologie n'est-elle pas l'œuvre des grands groupes industriels ? En fait, pour nous c'est presque l'inverse. Les auteurs de ce livre sont tous membres de ce que vous pourriez appeler la communauté du logiciel libre. Un groupe de personnes qui partagent l'idée fondatrice que les logiciels sont plus puissants, plus utiles, plus flexibles, mieux contrôlables, plus justes, plus englobants, plus durables, plus efficaces, plus sûrs et finalement simplement meilleurs quand ils sont fournis avec les quatre libertés fondamentales : la liberté d'utiliser, la liberté d'étudier, la liberté de partager et la liberté d'améliorer le logiciel.

Et bien qu'il y ait maintenant un nombre croissant de communautés qui ont appris à se passer de la proximité géographique grâce aux moyens de communication virtuels, c'est cette communauté qui en a été le précurseur.

En fait, Internet et la communauté du logiciel libre[2] suivaient des développements mutuellement dépendants. Au fur et à mesure qu'Internet

grandissait, notre communauté pouvait grandir en même temps. Mais sans les valeurs ni la technologie qu'apportait notre communauté, il ne fait aucun doute à mes yeux que jamais Internet n'aurait pu devenir ce réseau global reliant les personnes et les groupes du monde entier.

À ce jour, nos logiciels font fonctionner la majeure partie d'Internet, et vous devez en connaitre au moins quelques-uns, comme Mozilla Firefox, OpenOffice.org, Linux, et peut-être même Gnome ou KDE. Mais notre technologie peut aussi se cacher dans votre téléviseur, votre routeur sans fil, votre distributeur automatique de billets, et même votre radio, système de sécurité ou bataille navale. Elle est littéralement omniprésente.

Ils ont été essentiels dans l'émergence de quelques-unes des plus grandes sociétés que vous connaissez, comme Google, Facebook, Twitter et bien d'autres. Aucune d'entre elles n'aurait pu accomplir autant en si peu de temps sans le pouvoir du logiciel libre qui leur a permis de monter sur les épaules de ceux qui étaient là avant eux. Mais il existe également de nombreuses petites entreprises qui vivent de, avec, et pour le logiciel libre, dont la mienne, Kolab Systems. Le fait d'agir activement avec la communauté et dans un bon esprit est devenu un élément de succès essentiel pour nous tous. Et c'est aussi vrai pour les plus grosses, comme Oracle nous l'a involontairement démontré durant et après sa prise de contrôle de Sun Microsystems. Il est important de comprendre que notre communauté n'est pas opposée au commerce. Nous aimons notre travail, et beaucoup d'entre nous en ont fait leur métier pour gagner leur vie et rembourser leurs crédits. Donc quand nous parlons de communauté, nous voulons dire des étudiants, des entrepreneurs, des développeurs, des artistes, des documentalistes, des professeurs, des bricoleurs, des hommes d'affaires, des commerciaux, des bénévoles et des utilisateurs. Oui, des utilisateurs. Même si vous ne vous en êtes pas encore rendu compte ou n'avez jamais appartenu à une communauté, vous faites en réalité déjà partie de la nôtre. La question est de savoir si vous allez y participer activement. Et c'est cela qui nous différencie des poids lourds de la monoculture, des communautés fermées, des jardins clôturés de sociétés telles qu'Apple, Microsoft et d'autres. Nos portes sont ouvertes. Tout comme nos conseils. Et également notre potentiel. Il n'y a pas de limite à ce que vous pouvez devenir — cela dépend uniquement de votre choix personnel comme cela a été le cas pour chacun d'entre nous.

Donc si vous ne faites pas encore partie de notre communauté, ou si vous êtes

simplement curieux, ce livre offre un bon point de départ. Et si vous êtes déjà un participant actif, ce livre pourrait vous offrir un aperçu de quelques facettes et de quelques perspectives qui seront nouvelles pour vous.

En effet, ce livre contient d'importantes graines de ce savoir implicite que nous avons l'habitude de construire et de transférer à l'intérieur de nos sous-communautés qui travaillent sur diverses technologies. Ce savoir circule généralement des contributeurs les plus expérimentés vers les moins expérimentés. C'est pourquoi il semble tellement évident et naturel à ceux qui fréquentent notre communauté. Ce savoir et cette culture de la collaboration nous permettent de créer d'extraordinaires technologies avec de petites équipes du monde entier au-delà des différences culturelles, linguistiques et de nationalité. Cette manière de fonctionner permet de surpasser des équipes de développement bien plus grandes de certaines des plus grosses sociétés au monde. Tous les contributeurs de ce livre ont une expérience solide dans au moins un domaine, parfois plusieurs. Ils sont devenus des enseignants et des mentors. Au cours des quinze dernières années, j'ai eu le plaisir d'apprendre à connaître la plupart d'entre eux, de travailler avec beaucoup, et j'ai le privilège de compter certains parmi mes amis.

Comme l'a dit judicieusement Kévin Ottens pendant le Desktop Summit 2011 à Berlin, « construire une communauté, c'est construire de la famille et de l'amitié ».

C'est donc en réalité avec un profond sentiment de gratitude que je peux dire qu'il n'y a aucune autre communauté dont je préférerais faire partie, et je suis impatient de vous rencontrer à l'une ou l'autre des conférences à venir.

— Georg Greve

Zürich, Suisse, le 20 août 2011

Georg Greve a fondé la Free Software Foundation Europe (FSFE) en 2000 et en a été le président fondateur jusqu'en 2009. Durant cette période, il a été responsable du lancement et du développement de nombreuses activités de la FSFE, telles que les alliances, la politique ou les travaux juridiques. Il a intensivement travaillé avec de nombreuses communautés. Aujourd'hui, il poursuit ce travail en tant qu'actionnaire et PDG de Kolab Systems AG, une société qui se consacre entièrement aux logiciels libres. Pour ses actions en

faveur du logiciel libre et des standards ouverts, Georg Greve a été décoré de la croix fédérale du mérite (Bundesverdienstkreuz am Bande) par la République Fédérale d'Allemagne le 18 décembre 2009. Thank You! Merci !

Ce livre n'aurait pu voir le jour sans la participation de chaque auteur et des personnes suivantes, qui ont aidé à sa réalisation :

Anne Gentle (relecture)

Bernhard Reiter (relecture)

Celeste Lyn Paul (relecture)

Daniel Molkentin (mise en page)

Debajyoti Datta (site internet)

Irina Rempt (relecture)

Jeff Mitchell (relecture)

Mans Rullgard (relecture)

Noirin Plunkett (relecture)

Oregon State University Open Source Lab (hébergement du site internet)

Stuart Jarvis (relecture)

Supet Pal Singh (site internet)

Saransh Sinha (site internet)

Vivek Prakash (site internet)

Will Kahn-Greene (relecture)

* * * * * *

[1] Note des traducteurs : dédicace par allusion à *« Nous sommes des nains juchés sur les épaules de géants. »* Bernard de Chartres, XIIe siècle

[2] Note de l'auteur : pour moi, l'Open Source n'est que l'un des aspects de cette

communauté. Cet aspect particulier a trouvé son articulation en 1998, c'est-à-dire quelque temps après l'arrivée d'Internet. Mais n'hésitez pas à dire « Open Source » au lieu de « logiciel libre » si vous préférez ce terme.

Crédits photo **hellojenuine** (CC-BY-SA)