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.

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.