Passer de l’exercice scolaire à la maintenance des paquets (Libres conseils 28/42)

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

Traduction Framalang : satanas_g, Sphinx, Sky, Julius22, peupleLà, lamessen, goofy

Du débutant au professionnel

Jonathan Riddell

Jonathan Riddell est développeur KDE et Kubuntu, actuellement employé par Canonical. Quand il n’est pas devant un ordinateur, il fait du canoë sur les rivières d’Écosse.

Il y avait un bogue dans le code. Un bien méchant en plus : un plantage sans enregistrement des données. C’est bien là le problème dès qu’on regarde le code, on trouve des trucs à réparer. C’est facile de s’impliquer dans le logiciel libre ; le plus dur est d’en sortir. Après le premier bogue réparé, il y en a d’autres, et de plus en plus, tous à portée de main. Les corrections de bogues mènent à l’ajout de fonctionnalités, ce qui mène à la maintenance de projet, ce qui mène à faire fonctionner une communauté.

Tout a commencé en lisant Slashdot, cette masse d’actualité geek et technique peu filtrée avec des commentaires de quiconque peut recharger assez vite pour être en haut de liste. Chaque actualité était intéressante et excitante, apportait un éclairage nouveau sur le monde de la technologie qui finissait par me fasciner. Je n’avais plus à accepter ce qui m’était donné par de grandes entreprises de logiciels, je

pouvais voir là, dans la communauté du logiciel libre, le code se développer devant moi.

En tant qu'étudiant, il était possible de finir les exercices donnés par les professeurs très rapidement. Mais les exercices ne sont pas des programmes terminés. Je voulais savoir comment appliquer les compétences basiques qu'ils m'avaient données dans le monde réel en écrivant des programmes résolvant des problèmes réels pour les gens. J'ai donc recherché du code, qui n'était pas difficile à trouver, il se trouvait là, sur Internet, en fait. En regardant le code des programmes que j'utilisais de plus près, j'y ai décelé de la beauté. Non pas parce que le code était parfaitement soigné ou bien structuré, mais parce que je pouvais le comprendre avec les concepts que j'avais déjà appris. Ces classes, méthodes et variables étaient bien en place, me permettant de résoudre les problèmes pertinents. Le logiciel libre est le meilleur moyen de franchir le pas entre savoir comment finir ses exercices de cours et comprendre comment de vrais programmes sont écrits.

Tous les étudiants en informatique devraient travailler sur du logiciel libre comme sujet de leur mémoire. Sinon, vous avez de grandes chances d'y passer six mois à un an pour qu'il finisse au sous-sol d'une bibliothèque sans être jamais plus consulté. Seul le logiciel libre permet d'exceller en faisant ce qui va de soi : vouloir apprendre comment résoudre des problèmes intéressants. À la fin de mon projet, des programmeurs de la NASA utilisaient mon outil de création de diagrammes en **UML** (NdT : langage de modélisation unifié) et il reçut des prix au cours de réceptions somptueuses. Avec le logiciel libre, on peut résoudre de vrais problèmes pour de vrais utilisateurs.

La communauté des développeurs est remplie de personnes formidables, passionnées et dévouées à leur travail, sans espoir autre de récompense qu'un programme d'ordinateur couronné de succès. La communauté des utilisateurs est également incroyable. Il est satisfaisant de savoir qu'on a

aidé quelqu'un à résoudre un problème. Et j'apprécie les messages de remerciement que je reçois.

Après avoir écrit un logiciel utile, il faut le mettre à la disposition du plus grand nombre. Le code source ne va pas fonctionner pour la plupart des gens, il doit être compilé. Avant d'être impliqué, je trouvais que le fait de compiler était une manière un peu paresseuse de contribuer au logiciel libre. Vous vous attirez la plus grande partie de la reconnaissance sans rien avoir à coder. C'est, quelque part, quelque chose d'injuste. De même, la gestion de la communauté nécessaire pour porter un projet de logiciel libre peut aussi être vue comme une façon de s'attirer la reconnaissance sans faire de code.

Les utilisateurs dépendent beaucoup des *packagers* (NdT : les « empaqueteurs » qui préparent et maintiennent les paquets logiciels). Il est nécessaire que leur travail soit à la fois rapide, pour satisfaire ceux qui veulent la dernière version, et fiable, pour ceux qui veulent la stabilité (autant dire tout le monde). La partie la plus délicate, c'est que cela implique de travailler avec les logiciels des autres, qui sont toujours « cassés ». Une fois que le logiciel est lâché dans la nature, commencent à emerger des problèmes qui n'étaient pas repérables sur l'ordinateur de l'auteur. Il est possible que le code ne puisse pas être compilé avec une version de compilateur différente, peut-être que la licence n'est pas claire et ne permet pas de le copier, peut-être que la gestion des versions est incohérente et qu'une mise à jour mineure est incompatible, ou encore que la taille de l'écran est différente, les environnements de bureau peuvent aussi l'affecter, quelquefois, des bibliothèques tierces nécessaires ne sont pas encore à jour. De nos jours, le logiciel doit pouvoir tourner sur différentes architectures. Les processeurs 64 bits ont occasionné pas mal de problèmes quand ils sont devenus courants. Aujourd'hui, ce sont les processeurs ARM qui déjouent les calculs des codeurs. Les *packagers* doivent régler

tous ces problèmes pour donner aux utilisateurs quelque chose qui fonctionne de façon fiable.

Nous avons une règle chez Ubuntu selon laquelle les paquets avec des tests unitaires doivent inclure ces mêmes tests dans le processus de la création des paquets. Souvent, ils échouent et l'auteur du logiciel nous dit que les tests sont uniquement à son usage. Malheureusement, quand il s'agit de logiciel, il n'est jamais assez fiable de le tester soi-même, il doit aussi être testé par d'autres. Un test unique est rarement suffisant, il faut une approche à plusieurs niveaux. Les tests unitaires du programme original devraient être le point de départ, ensuite, le *packager* les teste sur son propre ordinateur, il faut ensuite que d'autres personnes les testent aussi. L'installation automatique et les tests de mise à jour peuvent être scriptés assez correctement sur les services d'informatique dans le nuage. L'envoyer dans la branche de développement d'une distribution permet d'effectuer plus de tests avant de le voir distribué en masse quelques mois après. À chaque étape, des problèmes peuvent être et seront découverts, ils devront être corrigés, puis ces correctifs eux-mêmes devront être testés. Il n'y a donc pas forcément à écrire beaucoup de code, mais il y a pas mal de travail pour passer le logiciel de 95 % à 100 % prêt. Ces 5 % sont la partie la plus difficile, un lent et délicat processus qui demande une grande attention pendant tout son cours.

Vous ne pouvez pas faire de paquets sans une bonne communication avec les développeurs en amont. Quand des bogues se produisent, il est vital de pouvoir trouver la bonne personne à laquelle parler rapidement. Il est important d'apprendre à bien les connaître comme des amis et des collègues. Les conférences sont vitales pour cela, car rencontrer quelqu'un apporte beaucoup plus de contexte à un message sur une liste de diffusion qu'une année entière de messages.

Une des faces cachées du monde du logiciel libre réside dans

la communication par les canaux IRC privés utilisés par les principaux membres d'un projet. Tous les grands projets en ont. Quelque part, Linus Torvalds a un moyen de discuter avec Andrew Morton et les autres sur ce qui est bon et sur ce qui est mauvais dans Linux. Ils sont plus sociaux que techniques et, quand on en abuse, ils peuvent être très antisociaux pour la communauté en général. Mais pour les moments où on a besoin d'un canal de communication rapide sans bruit parasite, ils fonctionnent bien.

Tenir un blog est un autre moyen de communication important dans la communauté du logiciel libre. C'est notre principale méthode pour promouvoir à la fois le logiciel que nous produisons et nous-mêmes. Non pas que ce soit utilisé éhontément pour de l'auto-promotion (il est inutile de prétendre que vous sauverez des vies avec votre blog…), mais parler de votre travail sur le logiciel libre aide à construire une communauté. Cela peut même vous valoir de trouver un travail ou d'être reconnu dans la rue.

Ces histoires venant de Slashdot, à propos de développements de nouvelles technologies, ne concernent pas des personnalités éloignées que vous ne rencontrerez jamais comme dans la presse *people*. Elles concernent des personnes qui ont trouvé un problème et qui l'ont résolu en utilisant l'ordinateur qu'elles avaient en face d'elles. Pendant quelques années, j'ai édité le site d'informations de KDE, trouvant les personnes qui résolvaient des problèmes, créaient des idées novatrices, s'acharnaient longuement à améliorer un logiciel jusqu'à ce qu'il soit d'une qualité suffisante, et j'en parlais au monde entier. Je n'ai jamais été à court d'histoires à raconter ni de personnes à présenter à tout le monde.

Mon dernier conseil est de conserver de la diversité. Il existe une telle richesse de projets intéressants à explorer, qui vous permettent d'apprendre et de progresser. Mais une fois que vous avez atteint une position de responsabilité, il

peut être tentant d’y rester. Après avoir aidé à créer une communauté pour Kubuntu, je repars temporairement vers un travail sur Bazaar, un projet très différent, orienté sur les développeurs plutôt que sur des utilisateurs novices en technologies. Je peux à nouveau apprendre comment le code devient une réalité utile, comment une communauté communique, comment la qualité est maintenue. Ce sera un défi amusant et j’ai hâte de m’y attaquer.