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

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