Caliopen, la messagerie libre sur la rampe de lancement

Le projet Caliopen, lancé il y a trois ans, est un projet ambitieux. Alors qu'il est déjà complexe de créer un nouveau logiciel de messagerie, il s'agit de proposer un agrégateur de correspondance qui permette à chacun d'ajuster son niveau de confidentialité.

Ce logiciel libre mûrement réfléchi est tout à fait en phase avec ce que Framasoft s'efforce de promouvoir à chaque fois que des libristes donnent aux utilisateurs et utilisatrices plus d'autonomie et de maîtrise, plus de sécurité et de confidentialité.

Après une nécessaire période d'élaboration, le projet Caliopen invite tout le monde à tester ***la version alpha*** *et à faire remonter les observations et suggestions. La première version grand public ce sera pour dans un an environ.*

Vous êtes curieux de savoir ce que ça donne ? Nous l'étions aussi, et nous avons demandé à ***Laurent Chemla****, qui bidouillait déjà dans l'Internet alors que vous n'étiez même pas né·e, de nous expliquer tout ça, puisqu'il est le père tutélaire du projet Caliopen, un projet que nous devons tous soutenir et auquel nous pouvons contribuer.*

Bonjour, pourrais-tu te présenter brièvement ?

J'ai 53 ans, dont 35 passés dans les mondes de l'informatique et des réseaux. Presque une éternité dans ce milieu – en tous cas le temps d'y vivre plusieurs vies (« pirate », programmeur, hacktiviste, entrepreneur…). Mais ces temps-ci je suis surtout le porteur du projet Caliopen,

même si je conserve une petite activité au sein du CA de la Quadrature du Net. Et je fais des macarons.

Le projet Caliopen arrive ce mois-ci au stade de la version alpha, mais comment ça a commencé ?

Jérémie Zimmermann est venu me sortir de ma retraite nîmoise en me poussant à relancer un très ancien projet de messagerie après les révélations de Snowden. Ça faisait déjà un petit moment que je me demandais si je pouvais encore être utile à la communauté autrement qu'en publiant quelques billets de temps en temps, alors j'ai lancé l'idée en public, pour voir, et il y a eu un tel retour que je n'ai pas pu faire autrement que d'y aller, malgré ma flemme congénitale.

Quand tu as lancé le projet publiquement (sur une liste de diffusion il me semble) quelle était la feuille de route, ou plutôt la « bible » des spécifications que tu souhaitais voir apparaître dans Caliopen ?

Très vite on a vu deux orientations se dessiner : la première, très technique, allait vers une vision maximaliste de la sécurité (réinventer SMTP pour protéger les métadonnées, garantir l'anonymat, passer par du P2P, ce genre de choses), tandis que la seconde visait à améliorer la confidentialité des échanges sans tout réinventer. Ça me semblait plus réaliste – parce que compatible avec les besoins du grand public – et c'est la direction que j'ai choisi de suivre au risque de fâcher certains contributeurs.

J'ai alors essayé de lister toutes les fonctionnalités (aujourd'hui on dirait les « *User Stories* ») qui sont apparues dans les échanges sur cette liste, puis de les synthétiser, et c'est avec ça que je suis allé voir Stephan Ramoin, chez Gandi, pour lui demander une aide qu'il a aussitôt accepté de donner. Le projet a ensuite évolué au rythme des échanges que j'ai pu avoir avec les techos de Gandi, puis de façon plus approfondie avec Thomas Laurent pendant la longue étape durant

laquelle nous avons imaginé le design de Caliopen. C'est seulement là, après avoir défini le « pourquoi » et le « quoi » qu'on a pu vraiment commencer à réfléchir au « comment » et à chercher du monde pour le réaliser.

La question qui fâche : quand on lit articles et interviews sur Caliopen, on a l'impression que le concept est encore super flou. C'est quoi *l'elevator pitch* pour vendre le *MVP* de la *start-up* aux *business angels* des internets digitaux ? (en français : tu dis quoi pour convaincre de nouveaux partenaires financiers ?)

Ça fait bien 3 ans que le concept de base n'a pas bougé : un agrégateur de correspondances qui réunit tous nos échanges privés (emails, message Twitter ou Facebook, messageries instantanées…), sous forme de conversations, définies par ceux avec qui on discute plutôt que par le protocole utilisé pour le faire. Voilà pour ton *pitch*.

Ce qui est vrai c'est qu'en fonction du public auquel on s'adresse on ne présente pas forcément le même angle. Le document qui a été soumis à BPI France pour obtenir le financement actuel fait 23 pages, très denses. Il aborde les aspects techniques, financiers, l'état du marché, la raison d'être de Caliopen, ses objectifs sociétaux, ses innovations, son design, les différents modèles économiques qui peuvent lui être appliqués… ce n'est pas quelque chose qu'on peut développer en un article ou une interview unique.

Si j'aborde Caliopen sous l'angle de la vie privée, alors j'explique par exemple le rôle des indices de confidentialité, la façon dont le simple fait d'afficher le niveau de confidentialité d'un message va influencer l'utilisateur dans ses pratiques: on n'écrit pas la même chose sur une carte postale que dans une lettre sous enveloppe. Rien que sur ce sujet, on vient de faire une conférence entière (à Paris Web

et à BlendWebMix) sans aborder aucun des autres aspects du projet.

Si je l'aborde sous l'angle technique, alors je vais peut-être parler d'intégration « verticale ». Du fait qu'on ne peut pas se contenter d'un nouveau Webmail, ou d'un nouveau protocole, si on veut tenir compte de tous les aspects qui font qu'un échange est plus ou moins secret. Ce qui fait de Caliopen un ensemble de différentes briques plutôt qu'une unique porte ou fenêtre. Ou alors je vais parler de la question du chiffrement, de la diffusion des clés publiques, de TOFU et du RFC 7929…

Mais on peut aussi débattre du public visé, de design, d'économie du Web, de décentralisation… tous ces angles sont pertinents, et chacun peut permettre de présenter Caliopen avec plus ou moins de détails.

Caliopen est un projet complexe, fondé sur un objectif (la lutte contre la surveillance de masse) et basé sur un moyen (proposer un service utile à tous), qui souhaite changer les habitudes des gens en les amenant à prendre réellement conscience du niveau d'exposition de leur vie privée. Il faut plus de talent que je n'en ai pour le décrire en quelques mots.

Il reste un intérêt pour les mails ? On a l'impression que tout passe par les webmails ou encore dans des applis de communication sur mobile, non ?

Même si je ne crois pas à la disparition de l'email, c'est justement parce qu'on a fait le constat qu'aujourd'hui la

correspondance numérique passe par de très nombreux services qu'on a imaginé Caliopen comme un agrégateur de tous ces échanges.

C'est un outil qui te permet de lire et d'écrire à tes contacts sans avoir à te préoccuper du service, ou de l'application, où la conversation a commencé. Tu peux commencer un dialogue avec quelqu'un par message privé sur Twitter, la poursuivre par email, puis par messagerie instantanée… ça reste une conversation: un échange privé entre deux humains, qui peuvent aborder différents sujets, partager différents contenus. Et quand tu vas vouloir chercher l'information que l'autre t'a donné l'année passée, tu vas faire comment ?

C'est à ça que Caliopen veut répondre. Pour parler moderne, c'est l'*User Story* centrale du projet.

C'est quoi exactement cette histoire de niveaux de confidentialité ? Quel est son but ?

Il faut revenir à l'objectif principal du projet : lutter contre la surveillance de masse que les révélations d'Edward Snowden ont démontrée.

Pour participer à cette lutte, Caliopen vise à convaincre un maximum d'utilisateurs de la valeur de leur vie privée. Et pour ça, il faut d'abord leur montrer, de manière évidente, que leurs conversations sont très majoritairement *espionnables*, sinon espionnées. Notre pari, c'est que quand *on voit* le risque d'interception, on réagit autrement que lorsqu'on est seulement *informé* de son existence. C'est humain : regarde l'exemple de la carte postale que je te donne plus haut.

D'où l'idée d'associer aux messages (mais aussi aux contacts, aux terminaux, et même à l'utilisateur lui-même) un niveau de confidentialité. Représenté par une icône, des couleurs, des chiffres, c'est une question de design, mais ce qui est

important c'est qu'en *voyant* le niveau de risque, l'utilisateur ne va plus pouvoir faire semblant de l'ignorer et qu'il va accepter de changer – au moins un peu – ses pratiques et ses habitudes pour voir ce niveau augmenter.

Bien sûr, il faudra l'accompagner. Lui proposer des solutions (techniques, comportementales, contextuelles) pour améliorer son « score ». Sans le culpabiliser (ce n'est pas la bonne manière de faire) mais en le récompensant – par une meilleure note, de nouvelles fonctionnalités, des options gratuites si le service est payant… bref par une *ludification* de l'expérience utilisateur. C'est notre piste en tous cas.

Et c'est en augmentant le niveau global de confidentialité des échanges qu'on veut rendre plus difficile (donc plus chère) la surveillance de tous, au point de pousser les états – et pourquoi pas les GAFAM – à changer de pratiques, eux aussi.

Financièrement, comment vit le projet Caliopen ? C'était une difficulté qui a retardé l'avancement ?

Sans doute un peu, mais je voudrais quand même dire que, même si je suis bien conscient de l'impression de lenteur que peut donner le projet, il faut se rendre compte qu'on parle d'un outil complexe, qui a démarré de zéro, avec aucun moyen, et qui s'attaque à un problème dont les racines datent de plusieurs dizaines d'années. Si c'était facile et rapide à résoudre, ça se saurait.

Dès l'instant où nous avons pris conscience qu'on n'allait pas pouvoir continuer sur le modèle du bénévolat, habituel au milieu du logiciel libre, nous avons réagi assez vite : Gandi a décidé d'embaucher à plein temps un développeur *front end*, sur ses fonds propres. Puis nous avons répondu à un appel à projet de BPI France qui tombait à pic et auquel Caliopen

était bien adapté. Nous avons défendu notre dossier, devant un comité de sélection puis devant un panel d'experts, et nous avons obtenu de quoi financer deux ans de développement, avec une équipe dédiée et des partenaires qui nous assurent de disposer de compétences techniques rares. Et tout ça est documenté sur notre blog, depuis le début (tout est public depuis le début, d'ailleurs, même si tous les documents ne sont pas toujours faciles à retrouver, même pour nous).

Et finalement c'est qui les partenaires ?

Gandi reste le partenaire principal, auquel se sont joints Qwant et l'UPMC (avec des rôles moins larges mais tout aussi fondamentaux).

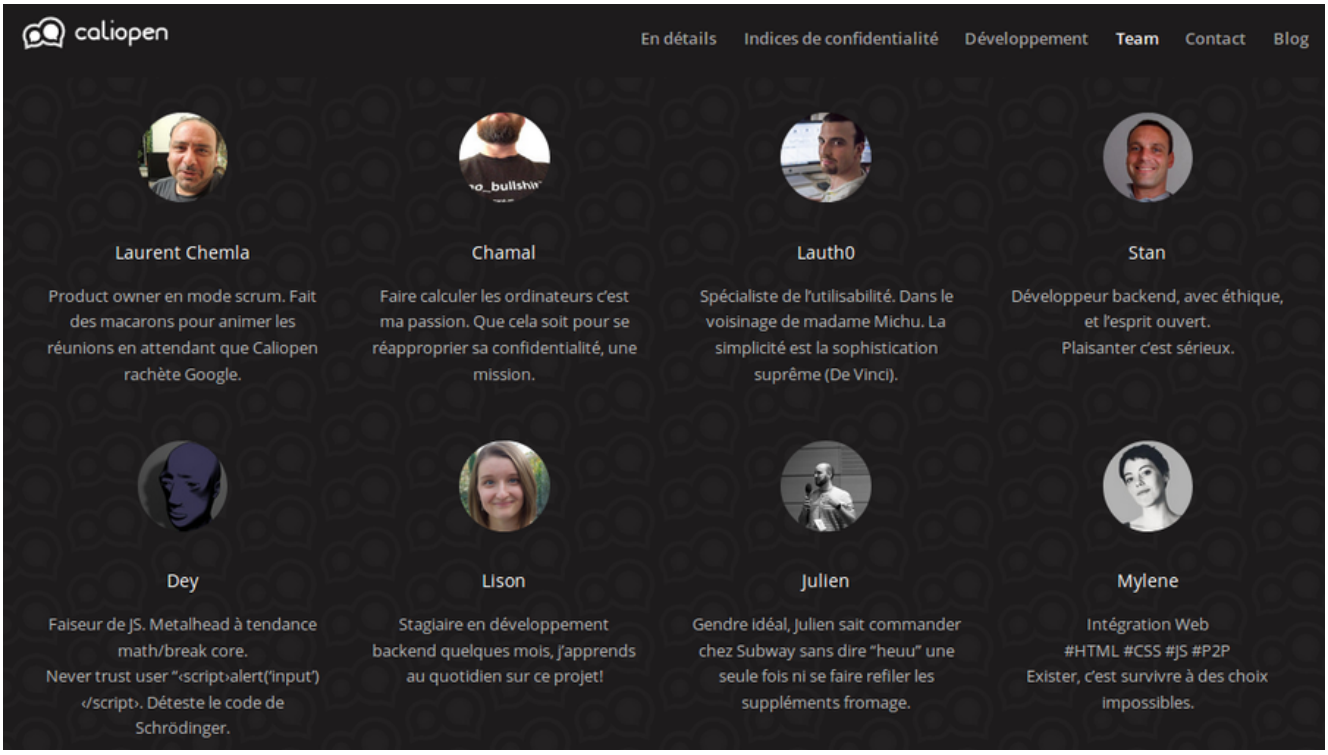
Quel est le modèle économique ? Les développeurs (ou développeuses, y'en a au fait dans l'équipe ?) sont rémunérés autrement qu'en macarons ? Combien faudra-t-il payer pour ouvrir un compte ?

Je ne suis pas sûr qu'on puisse parler de « modèle économique » pour un logiciel libre : après tout chacun pourra en faire ce qu'il voudra et lui imaginer tel ou tel modèle (économique ou non d'ailleurs).

Une fois qu'on a dit ça, on peut quand même dire qu'il ne serait pas cohérent de baser des services Caliopen sur l'exploitation des données personnelles des utilisateurs, et donc que le modèle « gratuité contre données » n'est pas adapté. Nous imaginons plutôt des services ouverts au public de type *freemium*, d'autres fournis par des entreprises pour leurs salariés, ou par des associations pour leurs membres. On peut aussi supposer que se créeront des services pour adapter Caliopen à des situations particulières, ou encore qu'il deviendra un outil fourni en Saas, ou vendu sous forme de *package* associé, par exemple, à la vente d'un nom de domaine.

Bref : les modèles économiques ce n'est pas ce qui manque le plus.

L’équipe actuelle est salariée, elle comporte des développeuses, et tu peux voir nos trombinettes sur https://www.caliopen.org



L’équipe de Caliopen

Trouver des développeurs ou développeuses n’est jamais une mince affaire dans le petit monde de l’*open source*, comment ça s’est passé pour Caliopen ?

Il faut bien comprendre que – pour le moment – Caliopen n’a pas d’existence juridique propre. Les gens qui bossent sur le projet sont des employés de Gandi (et bientôt de Qwant et de l’UPMC) qui ont soit choisi de consacrer une partie de leur temps de travail à Caliopen (ce que Gandi a rendu possible) soit été embauchés spécifiquement pour le projet. Et parfois nous avons des bénévoles qui nous rejoignent pour un bout de chemin

Le projet est encore franco-français. Tu t’en félicites (cocorico) ou ça t’angoisse ?

J’ai bien des sujets d’angoisse, mais pas celui-là. C’est un problème, c’est vrai, et nous essayons de le résoudre en allant, par exemple, faire des conférences à l’étranger (l’an dernier au FOSDEM, et cette année au 34C3 si notre soumission est acceptée). Et le site est totalement trilingue (français, anglais et italien) grâce au travail (bénévole) de Daniele Pitrolo.

D’un autre côté il faut quand même reconnaître que bosser au quotidien dans sa langue maternelle est un vrai confort dont il n’est pas facile de se passer. Même si on est tous conscients, je crois, qu’il faudra bien passer à l’anglais quand l’audience du projet deviendra un peu plus internationale, et nous comptons un peu sur les premières versions publiques pour que ça se produise.

Et au fait, c’est codé en quoi, Caliopen ? Du JavaScript surtout, d’après ce qu’on voit sur GitHub, mais nous supposons qu’il y a pas mal de technos assez pointues pour un tel projet ?

Sur GitHub, le code de Caliopen est dans un *mono-repository*, il n’y a donc pas de paquet (ou dépôt) spécifique au front ou au back. Le client est développé en JavaScript avec la librarie ReactJS. Le backend (l’API ReST, les workers …) sont développés en python et en Go. On n’a pas le détail mais ce doit être autour de 50% JS+css, 25% python, 25% Go. L’architecture est basée sur Cassandra et ElasticSearch.

Ce n’est pas que l’on utilise des technos pointues, mais plutôt qu’ on évite autant que possible la dette technique en intégrant le plus rapidement possible les évolutions des langages et des librairies que l’on utilise.

Donc il faut vraiment un haut niveau de compétences pour contribuer ?

Difficile à dire. Si on s’arrête sur l’aspect développement pur, les technos employées sont assez grand public, et si on a

suivi un cursus standard on va facilement retrouver ses habitudes (cf. https://github.com/kamranahmedse/developer-roadmap).

Effectivement quelqu'un qui n'a pas l'habitude de développer sur ces outils (docker, Go, webpack, ES6+ …) risque d'être un peu perdu au début. Mais on est très souvent disponibles sur IRC pour répondre directement aux questions.

Néanmoins nous avons de « simples » contributions qui ne nécessitent pas de connaître les patrons de conception par cœur ou de devoir monter un cluster; par exemple proposer des corrections orthographiques, de nouvelles traductions, décrire des erreurs JavaScript dans des issues sur github, modifier un bout de css…

Ou même aider la communauté sur https://feedback.caliopen.org/ ou sur les réseaux sociaux, tout ça en fait partie.

Et bien sûr les alpha-testeurs sont bienvenus surtout s'ils font des retours d'expérience.

Qu'est-ce qui différencie le projet Caliopen d'un projet comme Protonmail ?

Protonmail est un Gmail-like orienté vers la sécurité. Caliopen est un agrégateur de correspondance privée (ce qui n'est rien-like) orienté vers l'amélioration des pratiques du grand public via l'expérience utilisateur. Protonmail est

centralisé, Caliopen a prévu tout un (futur) écosystème exclusivement destiné à garantir la décentralisation des échanges. Et puis Caliopen est un logiciel libre, pas Protonmail.

Mais au-delà de ces différences techniques et philosophiques, ce sont surtout deux visions différentes, et peut-être complémentaires, de la lutte contre la surveillance de masse: Protonmail s'attaque à la protection de ceux qui sont prêts à changer leurs habitudes (et leur adresse email) parce qu'ils sont déjà convaincus qu'il faut faire certains efforts pour leur vie privée. Caliopen veut changer les habitudes de tous les autres, en leur proposant un service différent (mais utile) qui va les sensibiliser à la question. Parce qu'il faut bien se rendre compte que, malgré son succès formidable, aujourd'hui le nombre d'utilisateurs de Protonmail ne représente qu'à peine un millième du nombre d'utilisateurs de Gmail, et que quand les premiers échangent avec les seconds ils ne sont pas mieux protégés que M. Michu.

Maintenant, si tu veux bien imaginer que Caliopen est aussi un succès (on a le droit de rêver) et qu'il se crée un jour disons une dizaine de milliers de services basés sur noooootre proooojet, chacun ne gérant qu'un petit dixième du nombre d'utilisateurs de Protonmail… Eh ben sauf erreur on équilibre le nombre d'utilisateurs de Gmail et – si on a raison de croire que l'affichage des indices de confidentialité va produire un effet – on a significativement augmenté le niveau global de confidentialité.

Et peut-être même assez pour que la surveillance de masse devienne hors de prix.

Est-ce que dans la future version de Caliopen les messages seront chiffrés de bout en bout ?

À chaque fois qu'un utilisateur de Caliopen va vouloir écrire à un de ses contacts, c'est le protocole le plus sécurisé qui sera choisi par défaut pour transporter son message. Prenons un exemple et imaginons que tu m'ajoutes à tes contacts dans Caliopen : tu vas renseigner mon adresse email, mon compte Twitter, mon compte Mastodon, mon Keybase… plus tu ajouteras de moyens de contact plus Caliopen aura de choix pour m'envoyer ton message. Et il choisira le plus sécurisé par défaut (mais tu pourras décider de ne pas suivre son choix).

Plus tes messages auront pu être sécurisés, plus hauts seront leurs indices de confidentialité affichés. Et plus les indices de confidentialité de tes échanges seront hauts, plus haut sera ton propre indice global (ce qui devrait te motiver à mieux renseigner ma fiche contact afin d'y ajouter l'adresse de mon email hébergé sur un service Caliopen, parce qu'alors le protocole choisi sera le protocole intra-caliopen qui aura un très fort indice de confidentialité).

Mais l'utilisateur moyen n'aura sans doute même pas conscience de tout ça. Simplement le système fera en sorte de ne pas envoyer un message en clair s'il dispose d'un moyen plus sûr de le faire pour tel ou tel contact.

Est-ce qu'on pourra (avec un minimum de compétences, par exemple pour des CHATONS) installer Caliopen sur un serveur et proposer à des utilisateurs et utilisatrices une messagerie à la fois sécurisée et respectueuse ?

C'est fondamental, et c'est un des enjeux de Caliopen. Souvent quand je parle devant un public technique je pose la question : « combien de temps mettez-vous à installer un site Web en partant de zéro, et combien de temps pour une messagerie complète ? ». Et les réponses aujourd'hui sont bien sûr diamétralement opposées à ce qu'elle auraient été 15 ans plus tôt, parce qu'on a énormément travaillé sur la facilité d'installation d'un site, depuis des années, alors qu'on a totalement négligé la messagerie.

Si on veut que Caliopen soit massivement adopté, et c'est notre objectif, alors il faudra qu'il soit – relativement – facile à installer. Au moins assez facile pour qu'une entreprise, une administration, une association… fasse le choix de l'installer plutôt que de déléguer à Google la gestion du courrier de ses membres. Il faudra aussi qu'il soit facilement administrable, et facile à mettre à jour. Et tout ceci a été anticipé, et analysé, durant tout ce temps où tu crois qu'on n'a pas été assez vite !

On te laisse le dernier mot comme il est de coutume dans nos interviews pour le blog…

À lire tes questions j'ai conscience qu'on a encore beaucoup d'efforts à faire en termes de communication. Heureusement pour nous, Julien Dubedout nous a rejoints récemment, et je suis sûr qu'il va beaucoup améliorer tout ça.

- Devenir alpha-testeur
- Les fonctionnalités de Caliopen
- roadmap et bugreport

- GitHub de Caliopen

Faire un test à la fois vous met sur la bonne voie (Libres conseils 15/42)

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

Traduction Framalang : lamessen, Sky, Kalupa, ga3lig, Wxcafe, goofy, Astalaseven, Slystone, okram, KoS, Lycoris, peupleLa, Julius22

La Voie des tests conduit à la Lumière

Jonathan “Duke” Leto

Jonathan Leto, dit « Le Duc » est un développeur de logiciel, un mathématicien dont les travaux sont publiés, un ninja de Git et un passionné de cyclisme qui vit à Portland, en Oregon. C’est l’un des développeurs principaux de la machine virtuelle Parrot et le fondateur de Leto Labs LLC.

Lorsque j’ai commencé à m’impliquer dans le logiciel libre et open source, je n’avais aucune idée de ce que pouvaient être les tests ni de leur importance. J’avais travaillé sur des projets personnels de programmation auparavant, mais la première fois que j’ai réellement travaillé sur un projet collaboratif (c’est-à-dire en faisant un peu de *commit*) c’était pour Yacas, acronyme de Yet Another Computer Algebra

System, (NdT : encore un autre logiciel de calcul algébrique similaire à Mathematica).

À ce stade de mon parcours, les tests ne venaient qu'après coup. Mon méta-algorithme général était : bidouiller du code > voir si ça fonctionne> écrire (éventuellement) un test simple pour démontrer que ça fonctionne. Un test difficile à écrire n'était généralement jamais écrit.

C'est le premier pas sur la voie qui mène à la Lumière grâce aux tests. Vous savez que les tests sont probablement une bonne idée, mais vous n'en avez pas vu clairement les bénéfices, alors vous vous contentez de les écrire de temps en temps.

Si je pouvais ouvrir un trou de souris dans l'espace-temps et donner à mon moi plus jeune un conseil plein de sagesse sur les tests, ce serait :

> ***« Certains tests sont plus importants, sur le long terme, que le code qu'ils testent. »***

Il y a sans doute quelques personnes qui pensent en ce moment même que je mets mon casque de protection psychique (NdT : il s'agit d'un chapeau pour se protéger contre la manipulation à distance du cerveau) quand je m'assois pour écrire du code. Comment les tests pourraient-ils être plus importants que le code qu'ils testent ? Les tests sont la preuve que votre code marche réellement ; ils vous montrent le chemin vers l'écriture d'un code propre et vous apportent aussi la souplesse qui vous permettra de modifier le code tout en sachant que les fonctionnalités seront toujours là. En effet, plus votre code source grossit, plus vos tests sont importants, car ils vous permettent de changer une partie dudit code en sachant que le reste fonctionnera.

Une autre raison essentielle qui justifie l'écriture de tests est la possibilité de spécifier que quelque chose est un

souhait explicite et non une conséquence imprévue ou un oubli. Si vous avez un cahier des charges, vous pouvez utiliser des tests pour vérifier qu'il est respecté, ce qui est très important, voire indispensable dans certaines industries. Un test, c'est comme quand on raconte une histoire : l'histoire de votre conception du code et de la façon dont il devrait fonctionner. Soit le code change et évolue, soit il mute en code infectieux (1).

Très souvent, vous écrirez des tests une première fois pour ensuite remettre totalement en cause votre réalisation voire la réécrire à partir de zéro. Les tests survivent souvent au code pour lesquels ils ont été conçus à l'origine. Par exemple, un jeu de tests peut être utilisé quel que soit le nombre de fois où votre code est transformé. Les tests sont en fait l'examen de passage qui vous permettra de jeter une ancienne réalisation et de dire « cette nouvelle version a une bien meilleure structure et passe notre jeu de tests ». J'ai vu cela se produire bien des fois dans les communautés Perl et Parrot, où vous pouvez souvent me voir traîner. Les tests vous permettent de changer les choses rapidement et de savoir si quelque chose est cassé. Ils sont comme des propulseurs pour les développeurs.

Les charpentiers ont un adage qui dit quelque chose comme ça :

> *« Mesurer deux fois, couper une fois. »*

Le code serait la coupe, le test serait la mesure.

La méthode de développement basée sur les tests fait économiser beaucoup de temps, parce qu'au lieu de vous prendre la tête à bricoler le code sans but défini, les tests précisent votre objectif.

Les tests sont aussi un très bon retour d'expérience. Chaque fois que vous faites une nouvelle passe de test, vous savez que votre code s'améliore et qu'il a une fonctionnalité de

plus ou un bogue de moins.

Il est facile de se dire « je veux ajouter 50 fonctionnalités » et de passer toute la journée à bricoler le code tout en jonglant en permanence entre différents travaux. La plupart du temps, peu de choses aboutiront. La méthode de développement basée sur les tests aide à se concentrer sur la réussite d'un seul test à la fois.

Si votre code échoue devant ne serait-ce qu'un seul test, vous avez pour mission de le faire réussir. Votre cerveau se concentre alors sur quelque chose de très spécifique, et dans la plupart des cas cela produit de meilleurs résultats que passer constamment d'une tâche à une autre.

La plupart des informations relatives au développement basé sur les tests sont très spécifiques à un langage ou à une situation, mais ce n'est pas une obligation. Voilà comment aborder l'ajout d'une nouvelle fonctionnalité ou la correction d'un bogue dans n'importe quel langage :

1. Écrivez un test qui fait échouer votre code, mais qui, selon vous, sera passé quand la fonctionnalité sera implémentée ou que le bogue sera corrigé. Mode expert : pendant l'écriture du test, pensez à l'exécuter de temps en temps, même s'il n'est pas encore fini, et tentez de deviner le message d'erreur effectif que le test renverra. À force d'écrire des tests et de les faire tourner, cela devient plus facile ;
2. Bidouillez le code ;
3. Exécutez le test. S'il marche, passez au point 4, sinon retournez au point 2 ;
4. C'est fini ! Dansez le sirtaki.

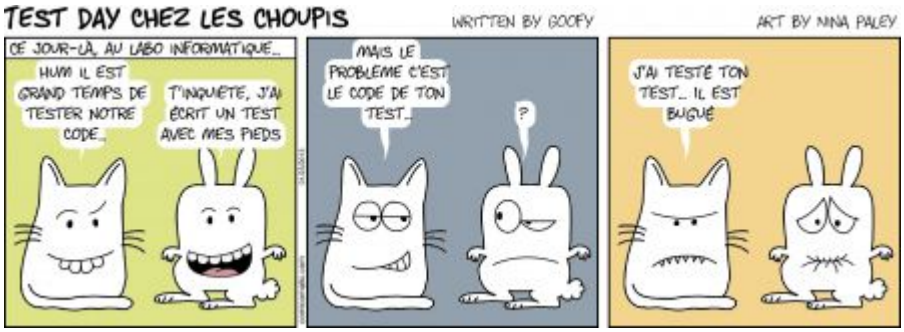
Cette méthode fonctionne pour n'importe quel type de test et n'importe quel langage. Si vous ne deviez retenir qu'une seule chose de ce texte, souvenez-vous des étapes ci-dessus.

Voici maintenant quelques directives plus générales de

conduite de tests qui vous serviront bien et fonctionneront dans n'importe quelle situation :

1. Comprendre la différence entre ce qu'on teste et ce qu'on utilise comme un outil pour tester autre chose ;
2. Les tests sont fragiles. Vous pouvez toujours écrire un test qui contrôle la validité d'un message d'erreur. Mais que se passera-t-il si le message d'erreur change ? Que se passera-t-il quand quelqu'un traduira votre code en catalan ? Que se passera-t-il lorsque quelqu'un exécutera votre code sur un système d'exploitation dont vous n'avez jamais entendu parler ? Plus votre test est résistant, plus il aura de valeur.

Pensez à cela quand vous écrivez des tests. Vous voulez qu'ils soient résistants, c'est-à-dire que les tests, dans la plupart des cas, ne devraient avoir à changer que quand les fonctionnalités changent. Si vous devez modifier vos tests régulièrement, sans que les fonctionnalités aient changé, c'est que vous faites une erreur quelque part.



Types de tests

Bien des personnes sont perdues quand on leur parle de tests d'intégration, tests unitaires, tests d'acceptation et autres tests à toutes les sauces. Il ne faut pas trop se soucier de ces termes. Plus vous écrirez de tests, mieux vous en distinguerez les nuances et les différences entre les tests deviendront plus apparentes. Tout le monde n'a pas la même définition de ce que sont les tests, mais c'est utile d'avoir des termes pour décrire les types de tests.

Tests unitaires contre tests d'intégration

Les tests unitaires et les tests d'intégration couvrent un large spectre. Les tests unitaires testent de petits segments de code et les tests d'intégration vérifient comment ces segments se combinent. Il revient à l'auteur du test de décider ce que comprend une unité, mais c'est le plus souvent au niveau d'une fonction ou d'une méthode, même si certains langages appellent ces choses différemment.

Pour rendre cela un peu plus concret, nous établirons une analogie sommaire en utilisant des fonctions. Imaginez que f(x) et g(x) soient deux fonctions qui représentent deux unités de code. Pour l'aspect concret, supposons qu'elles représentent deux fonctions spécifiques du code de base de votre projet libre et open source.

Un test d'intégration affirme quelque chose comme la composition de la fonction, par exemple $f(g(a)) = b$. Un test d'intégration consiste à tester la façon dont plusieurs choses s'intègrent ou travaillent ensemble, plutôt que la façon dont chaque partie fonctionne individuellement. Si l'algèbre n'est pas votre truc, une autre façon de comprendre est de considérer que les tests unitaires ne testent qu'une partie de la machine à la fois, tandis que les tests d'intégration s'assurent que la plupart des parties fonctionnent à l'unisson. Un bel exemple de test d'intégration est le test de conduite d'une voiture. Vous ne vérifiez pas la pression atmosphérique, ni ne mesurez le voltage des bougies d'allumage. Vous vous assurez que le véhicule fonctionne globalement.

La plupart du temps, il est préférable d'avoir les deux. Je commence souvent avec les tests unitaires puis j'ajoute les tests d'intégration au besoin puisqu'on a besoin d'éliminer d'abord les bogues les plus basiques, puis de trouver les

bogues plus subtils issus d'un emboitement imparfait des morceaux, à l'opposé de pièces qui ne fonctionnent pas individuellement. Beaucoup de gens écrivent d'abord des tests d'intégration puis se plongent dans les tests unitaires. Le plus important n'est pas de savoir lequel vous écrirez en premier, mais d'écrire les deux types de tests.

Vers la Lumière

La méthode de développement basée sur les tests est un chemin, pas un aboutissement. Sachez apprécier le voyage et assurez-vous de faire une pause pour respirer les fleurs si vous êtes égaré.

(1) Équivalent approché du terme *bitrot* qui en argot de codeur désigne ce fait quasi-universel : si un bout de code ne change pas mais que tout repose sur lui, il « pourrit ». Il y a alors habituellement très peu de chances pour qu'il fonctionne tant qu'aucune modification ne sera apportée pour l'adapter à de nouveaux logiciels ou nouveaux matériels.

Comic strip réalisé avec le Face-O-Matic de Nina Paley et Margot Burns

---

Remue-ménage dans le triage (Libres conseils 14/42)

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

Traduction Framalang : lamessen, Sky, Kalupa, ga3lig, goofy, Astalaseven, okram, KoS, Lycoris, 4nti7rust, peupleLa + Julius22

Penser/Classer

Andre Klapper

Dans la vraie vie, Andre Klapper est maître ès débogage. Pendant sa pause déjeuner ou sa sieste, il travaille à divers trucs sur GNOME (bugsquad, équipe de release, traduction, documentation, etc.), ou Maemo, étudie ou mange de la crème glacée.

Au tout début, je n'avais qu'une seule et unique question : comment imprimer seulement une partie du courriel que j'ai reçu dans Evolution, le client de messagerie GNOME ? J'ai donc demandé sur la liste de diffusion.

Ça faisait exactement un an que j'étais passé sous Linux, frustré de ne pouvoir faire fonctionner mon modem après avoir réinstallé un OS propriétaire plutôt populaire à l'époque.

La réponse à ma question fut : « impossible ». Des petits génies auraient parcouru le code, l'auraient compilé, l'auraient bidouillé pour qu'il se comporte comme voulu, puis auraient soumis un correctif joint au rapport de bogue. Bon. Comme vous l'aurez deviné, je n'étais pas un petit génie. Mes talents de programmeur sont plutôt limités, donc sur le moment je suis resté coincé sur une solution de contournement plutôt lourde pour mon impression. La réponse que j'avais reçue sur la liste de diffusion signalait également que cette fonctionnalité était prévue, et qu'on avait complété pour moi un rapport de bogue – sans préciser où, mais je m'en fichais, j'étais content d'apprendre qu'il était prévu de corriger mon problème prochainement.

Il se peut que je sois resté abonné à la liste de diffusion par simple paresse. Certains mentionnaient le rapporteur de bogues de temps en temps, souvent comme une réponse directe aux demandes de fonctionnalités, alors j'y ai finalement jeté un coup d'œil. Mais les rapporteurs de bogue, en particulier Bugzilla, sont d'étranges outils avec beaucoup d'options complexes. Un domaine que vous préférez normalement éviter à moins que vous ne soyez masochiste. Ils contiennent maints tickets décrivant des bogues ou des demandes de fonctionnalités émanant d'utilisateurs et de développeurs. Il semblait également que ces rapports aient été en partie utilisés pour planifier les priorités (appeler cela « gestion de projet » aurait été un euphémisme ; moins d'un quart des problèmes qui étaient planifiés pour être résolus ou implémentés dans une version spécifique étaient réellement corrigés au bout du compte).

Au-delà d'une vision intéressante sur les problèmes du logiciel et sur la popularité de certaines demandes, ce que j'ai découvert, c'est beaucoup de choses incohérentes et pas

mal de bruit, comme des doublons ou des rapports de bogues manquant d'éléments pour pouvoir être traités correctement. J'ai eu envie de nettoyer un peu en « triant » les rapports de bogues disponibles. Je ne sais pas bien ce que cela vous dit sur mon état d'esprit – ajouter ici des mots-clés bidon pour une caractérisation aléatoire, comme *organisé*, *persévérant et intelligent*. C'est assez ironique quand on pense à mon père qui se plaignait toujours du bordel dans ma chambre. Donc à cette époque lointaine de modems commutés, j'avais pour habitude de rassembler mes questions et de les faire remonter sur IRC une fois par jour afin de mitrailler de questions le responsable des bogues d'Evolution, qui était toujours accueillant, patient et soucieux de partager son expérience. Si jamais à l'époque il y avait un guide de triage qui couvrait les savoirs de base pour la gestion des bogues et qui exposait les bonnes pratiques et les pièges les plus courants, je n'en avais pas entendu parler.

Le nombre de signalements baissa de 20% en quelques mois, bien que ce ne fût bien évidemment pas grâce à une unique personne qui faisait le tri des tickets. Il y avait manifestement du travail en attente, comme diminuer le nombre des tickets attribués aux développeurs pour qu'ils puissent mieux se concentrer, parler avec eux, définir les priorités, et répondre aux commentaires non-traités de certains utilisateurs. L'open source accueille toujours bien les contributions une fois que vous avez trouvé votre créneau.

Bien plus tard, j'ai pris conscience qu'il y avait de la documentation à consulter. Luis Villa, qui fut probablement le premier des experts en bogues, a écrit un essai titré « Pourquoi tout le monde à besoin d'un expert en bogue » et la majorité des équipes anti-bogues sur les projets *open source* ont publié au même moment des guides sur le triage qui ont aidé les débutants à s'impliquer dans la communauté. De nombreux développeurs ont débuté leur fantastique carrière dans l'open source en triant les bogues et ont ainsi acquis

une première expérience de gestion de projet logiciel.

Il y a aussi de nos jours des outils qui peuvent vous épargner beaucoup de temps quand arrive l'abrutissant travail de triage. Du côté serveur, l'extension « stock answers » de GNOME fournit les commentaires courants et fréquemment usités afin de les ajouter aux tickets en un clic pendant que, du côté client, vous pouvez faire tourner votre propre script GreaseMonkey ou l'extension Jetpack de Matej Cepl, appelée « bugzilla-triage-scripts » [2].

Si vous êtes un musicien moyen ou médiocre mais que vous aimez tout de même la musique par-dessus tout, vous pouvez toujours y travailler en tant que journaliste. Le développement de logiciels possède également ce genre de niches qui peuvent vous donner satisfaction, au-delà de l'idée première d'écrire du code. Cela vous prendra un peu de temps pour les trouver, mais ça vaut la peine d'y consacrer vos efforts, votre expérience et vos contacts. Avec un peu de chance et de talent, cela peut même vous permettre de gagner votre vie dans le domaine qui vous intéresse personnellement… et vous éviter de finir pisse-code.

[1] http://tieguy.org/talks-files/LCA-2005-paper-html/index.html

[2] https://fedorahosted.org/bugzilla-triage-scripts

Crédit photo : Doug DuCap Food and Travel (CC BY-NC-SA 2.0)

Tests contre Bogues : une

guerre sans fin (Libres conseils 13/42)

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

Traduction Framalang : Floxy, ga3lig, goofy, Astalaseven, Slystone, okram, KoS, Lycoris, 4nti7rust, peupleLa, Luc Didry, + Julius22

Même en multipliant les regards, les bogues ne sautent pas aux yeux.

Ara Pulido

Ara Pulido est ingénieure d'essais pour Canonical, d'abord comme membre de l'équipe assurance qualité d'Ubuntu (QA team), et maintenant dans le cadre de l'équipe de certification du matériel. Même si elle a commencé sa carrière en tant que développeuse, elle a vite découvert que ce qu'elle aimait vraiment, c'était tester les logiciels. Elle est très intéressée par les nouvelles techniques d'analyse et tente d'utiliser son savoir-faire pour améliorer Ubuntu.

Les tests maison ne suffisent pas

Je me suis impliquée dans le logiciel libre dès le début de mes études à l'Université de Grenade. Là-bas, avec des amis, nous avons fondé un groupe local d'utilisateurs de Linux et organisé plusieurs actions pour promouvoir le logiciel libre. Mais, depuis que j'ai quitté l'université, et jusqu'à ce que je commence à travailler chez Canonical, ma carrière

professionnelle s'est déroulée dans l'industrie du logiciel propriétaire, d'abord comme développeuse puis comme testeuse.

Lorsque l'on travaille au sein d'un projet de logiciel propriétaire, les ressources pour tester sont très limitées. Une petite équipe reprend le travail initié par les développeurs avec les tests unitaires, utilisant leur expérience pour trouver autant de bogues que possible afin de mettre à disposition de l'utilisateur final un produit aussi abouti que possible. Dans le monde du logiciel libre, en revanche, tout est différent.

Lors de mon embauche chez Canonical, hormis la réalisation de mon rêve d'avoir un travail rémunéré au sein d'un projet de logiciel libre, j'ai été émerveillée par les possibilités des activités de test dans le cadre d'un tel projet. Le développement du produit s'effectue de manière ouverte, et les utilisateurs ont accès au logiciel dès son commencement, ils le testent et font des rapports de bogues dès que c'est nécessaire. C'est un nouveau monde rempli de beaucoup de possibilités pour une personne passionnée par les tests. Je voulais en profiter au maximum.

Comme beaucoup de personnes, je pensais que les tests « maison », c'est-à-dire l'utilisation par soi-même du logiciel que l'on envisage de mettre à disposition, était l'activité de test la plus importante qu'on puisse mener dans le logiciel libre. Mais si, selon la formule de Raymond dans La cathédrale et le bazar « avec suffisamment d'observateurs, tous les bogues sautent aux yeux », alors comment se fait-il qu'avec ses millions d'utilisateurs Ubuntu comporte encore des bogues sérieux à chaque nouvelle version ?

La première chose dont je me suis aperçue quand j'ai commencé à travailler chez Canonical c'est que les activités de test organisées étaient rares ou inexistantes. Les seules sessions de test qui étaient d'une certaine façon organisées se présentaient sous la forme de messages électroniques envoyés à

une liste de diffusion, manière de battre le rappel pour tester un paquetage dans la version de développement d'Ubuntu. Je ne pense pas que cela puisse être considéré comme une vraie procédure de test, mais simplement comme une autre forme de « test maison ». Cette sorte de test génère beaucoup de doublons, car un bogue facile à débusquer sera documenté par des centaines de personnes. Malheureusement le bogue potentiellement critique, vraiment difficile à trouver, a de bonnes chances de passer inaperçu en raison du bruit créé par les autres bogues, et ce même si quelqu'un l'a documenté.

En progrès

La situation s'améliore-t-elle ? Sommes-nous devenus plus efficaces pour les tests au sein des projets de développement libre ? Oui, j'en suis convaincue.

Pendant les derniers cycles de développement d'Ubuntu, nous avons commencé bon nombre de sessions de test. La gamme des objectifs pour ces sessions est large, elle comprend des domaines comme de nouvelles fonctionnalités de bureau, des tests de régression, des tests de pilotes X.org ou des tests de matériel d'ordinateur portable. Les résultats sont toujours suivis et ils s'avèrent vraiment utiles pour les développeurs, car ils leur permettent de savoir si les nouveautés fonctionnent correctement, au lieu de supposer qu'elles fonctionnent correctement à cause de l'absence de bogues.

En ce qui concerne les outils d'assistance aux tests, beaucoup d'améliorations ont été apportées :

- Apport(1) a contribué à augmenter le niveau de détail des bogues signalés concernant Ubuntu : les rapports de plantage incluent toutes les informations de débogage et leurs doublons sont débusqués puis marqués comme tels ; les utilisateurs peuvent signaler des bogues sur base de symptômes, etc.
- Le Launchpad(2), avec ses connexions en amont, a permis

d’avoir une vue complète des bogues – sachant que les bogues qui se produisent dans Ubuntu se situent généralement dans les projets en amont, et permet aux développeurs de savoir si les bogues sont en cours de résolution.
- Firefox, grâce à son programme et à son extension Test Pilot, mène des tests sans qu’on ait à quitter le navigateur(3). C’est, à mon sens, une bien meilleure façon de rallier des testeurs qu’une liste de diffusion ou un canal IRC.
- L’équipe Assurance Qualité d’Ubuntu teste le bureau en mode automatique et rend compte des résultats toutes les semaines(4), ce qui permet aux développeurs de vérifier très rapidement qu’il n’y a pas eu de régression majeure pendant le développement.

Cependant, malgré l’amélioration des tests dans les projets de logiciel libre il reste encore beaucoup à faire.

Pour aller plus loin

Les tests nécessitent une grande expertise, mais sont encore considérés au sein de la communauté du le logiciel libre comme une tâche ne demandant pas beaucoup d’efforts. L’une des raisons pourrait être que la manière dont on les réalise est vraiment dépassée et ne rend pas compte de la complexité croissante du monde du logiciel libre durant la dernière décennie. Comment est-il possible que, malgré la quantité d’innovations amenées par les communautés du logiciel libre, les tests soient encore réalisés comme dans les années 80 ? Il faut nous rendre à l’évidence, les scénarios de tests sont ennuyeux et vite obsolètes. Comment faire grandir une communauté de testeurs supposée trouver des bogues avérés si sa tâche principale est de mettre à jour les scénarios de test ?

Mais comment améliorer la procédure de test ? Bien sûr, nous

ne pouvons pas nous débarrasser des scénarios de test, mais nous devons changer la façon dont nous les créons et les mettons à jour. Nos testeurs et nos utilisateurs sont intelligents, alors pourquoi créer des scripts pas-à-pas ? Ils pourraient aisément être remplacés par une procédure de test automatique. Définissons plutôt une liste de tâches que l'on réalise avec l'application, et certaines caractéristiques qu'elle devrait posséder. Par exemple, « l'ordre des raccourcis dans le lanceur doit pouvoir être modifié », ou « le démarrage de LibreOffice est rapide ». Les testeurs trouveront un moyen de le faire, et créeront des scénarios de test en parallèle des leurs.

Mais ce n'est pas suffisant, nous avons besoin de meilleurs outils pour aider les testeurs à savoir ce qu'ils testent, où et comment. Pourquoi ne pas avoir des API (interfaces de programmation) qui permettent aux développeurs d'envoyer des messages aux testeurs à propos des nouvelles fonctionnalités ou des mises à jour qui doivent être testées ? Pourquoi pas une application qui nous indique quelle partie du système doit être testée ? en fonction des tests en cours ? Dans le cas d'Ubuntu, nous avons les informations dans le Launchpad (il nous faudrait aussi des données sur les tests, mais au moins nous avons des données sur les bogues). Si je veux démarrer une session de test d'un composant en particulier j'apprécierais vraiment de savoir quelles zones n'ont pas encore été testées ainsi qu'une liste des cinq bogues comptant le plus de doublons pour cette version en particulier afin d'éviter de les documenter une fois de plus. J'aimerais avoir toutes ces informations sans avoir à quitter le bureau que je suis en train de tester. C'est quelque chose que Firefox a initié avec Test Pilot, bien qu'actuellement l'équipe rassemble principalement les données sur l'activité du navigateur.

La communication entre l'amont et l'aval et vice-versa doit aussi être améliorée. Pendant le développement d'une

distribution, un bon nombre des versions amont sont également en cours de développement, et ont déjà une liste des bogues connus. Si je suis un testeur de Firefox sous Ubuntu, j'aimerais avoir une liste des bogues connus aussitôt que le nouveau paquet est poussé dans le dépôt. Cela pourrait se faire à l'aide d'une syntaxe reconnue pour les notes de versions, syntaxe qui pourrait ensuite être facilement analysée. Les rapports de bogue seraient automatiquement remplis et reliés aux bogues amont. Encore une fois, le testeur devrait avoir facilement accès à ces informations, sans quitter son environnement de travail habituel.

Les tests, s'ils étaient réalisés de cette manière, permettraient au testeur de se concentrer sur les choses qui comptent vraiment et font de la procédure de test une activité qualifiée ; se concentrer sur les bogues cachés qui n'ont pas encore été découverts, sur les configurations et environnements spéciaux, sur la création de nouvelles manières de casser le logiciel. Et, in fine, s'amuser en testant.

Récapitulons

Pour ce que j'en ai vu ces trois dernières années, les tests ont beaucoup progressé au sein d'Ubuntu et des autres projets de logiciels libres dans lesquels je suis plus ou moins impliquée, mais ce n'est pas suffisant. Si nous voulons vraiment améliorer la qualité du logiciel libre, nous devons commencer à investir dans les tests et innover dans la manière de les conduire, de la même façon que nous investissons dans le développement. Nous ne pouvons pas tester le logiciel du XXIe siècle avec les techniques du XXe siècle. Nous devons réagir. Qu'il soit open source ne suffit plus à prouver qu'un logiciel libre est de bonne qualité. Le logiciel libre sera bon parce qu'il est open source et de la meilleure qualité que nous puissions offrir.

1 http://wiki.ubuntu.com/Apport

2 http://launchpad.net

3 http://testpilot.mozillalabs.com

4 http://reports.qa.ubuntu.com/reports/desktop-testing/natty