13 points que les gens détestent sur la documentation de votre projet libre

Qu'il s'agisse de son code ou de son utilisation, la faiblesse de la documentation d'un logiciel libre est souvent montrée du doigt.

Voici, selon Andy Lester, 13 défauts ou lacunes communément rencontrés, qui sont autant d'écueils que l'on peut contourner avec un minimum d'efforts aujourd'hui pour gagner demain un temps précieux.

13 choses que les gens détestent sur vos documentations open source

13 Things People Hate about Your Open Source Docs

Andy Lester – 10 janvier 2013 – SmartBear Blog
(Traduction : Lamessen, calou, Shanx, sinma, Asta + anonymes)

La plupart des développeurs open source aiment penser à la qualité du logiciel qu'ils développent, mais la qualité de la documentation est souvent laissée de côté. Il est rare de voir vanter la documentation d'un projet, et pourtant elle a un impact direct sur sa réussite. Sans une bonne documentation, les utilisateurs n'utiliseront pas votre projet, ou ils n'y prendront pas de plaisir. Les utilisateurs comblés sont ceux qui diffusent des infos à propos de votre projet – ce qu'ils ne font qu'après avoir compris comment il fonctionne. Et ils apprennent cela à partir de la documentation du projet.

Malgré tout, de trop nombreux projets ont une documentation décevante. Et cela peut être décevant de plusieurs manières.

Les exemples que je donne ci-dessous sont purement arbitraires, je ne veux pas cibler un projet en particulier. Ce sont seulement ceux que j'ai utilisés récemment, cela ne veut pas dire qu'ils représentent les pires atrocités. Chaque projet a commis au moins quelques-uns de ces péchés. Que vous soyez utilisateur ou développeur, à vous d'évaluer à quel point votre logiciel préféré est ou non coupable, et comment vous pouvez aider à y remédier le cas échéant.

1. Le manque d'une bonne introduction ou d'un README/LISEZ-MOI

Le README/LISEZ-MOI est la première impression que les utilisateurs potentiels ont de votre projet. Si le projet est sur GitHub, le README/LISEZ-MOI est automatiquement affiché sur la page d'accueil du projet. Si vous l'avez mal rédigé,

ils peuvent ne jamais revenir.

Vous voulez capter l'attention du lecteur et l'encourager à continuer la découverte de votre projet ? Le README/LISEZ-MOI devrait alors au moins expliquer :

- ce que le projet fait
- pour qui il est fait
- sur quel matériel ou plateforme il tourne
- toutes les dépendances majeures, comme « Requiert Python 2.6 et libxml »
- comment l'installer, ou un accompagnement de chaque étape à la suivante.

Tout cela doit pouvoir être compris par quelqu'un qui n'a jamais entendu parler de votre projet, et peut-être même jamais imaginé un projet pouvant s'en rapprocher. Si le projet possède un module calculant la distance de Levenshtein, ne partez pas du principe que n'importe qui lisant votre README/LISEZ-MOI sait ce que c'est. Expliquez que la distance de Levenshtein est utilisée pour comparer deux chaînes de caractères, et ajoutez quelques renvois vers des explications plus poussées pour celui qui aimerait approfondir le sujet.

Ne décrivez pas votre projet par rapport à un autre projet, comme « NumberDoodle est comme BongoCalc, mais meilleur ! » Ça n'est d'aucune aide pour quelqu'un qui n'a jamais entendu parlé de BongoCalc.

2. La documentation non disponible en ligne

Bien que je n'ai pas lu d'études à ce sujet, je serais prêt à parier que 90% des recherches de documentation sont faites avec Google et un navigateur sur Internet. La documentation de votre projet doit être en ligne, et disponible. Partant de là, il serait embarrassant que la documentation de mon propre projet, ack, ne soit pas disponible à l'endroit où la majorité des gens vont la chercher. Mon hypothèse est basée sur ma propre expérience, à savoir que si je veux connaître le

fonctionnement d'un outil en ligne de commande, je vais vérifier sa page man.

Comment je m'en suis aperçu ? Les utilisateurs m'écrivaient pour me poser des questions dont les réponses se trouvaient dans la FAQ. Ce qui m'a ennuyé : ils ne lisaient pas ma FAQ. Il se trouve qu'ils avaient cherché sur le site internet, mais je n'avais pas mis la FAQ à cet endroit. C'est une erreur facile à faire. Je suis proche du projet et je n'ai jamais eu besoin d'utiliser moi-même la FAQ, je n'avais donc pas remarqué qu'elle n'était pas présente en ligne. Beaucoup de problèmes sont dus à ce piège : les auteurs ne se mettent pas à la place des utilisateurs.

3. La documentation disponible uniquement en ligne

Le revers de ce problème est d'avoir la documentation disponible uniquement en ligne. Certains projets ne distribuent pas la documentation avec les livrables du projet, ou incluent une version médiocre de la documentation.

Le moteur de recherche Solr, par exemple, a un excellent wiki qui sert à la documentation du projet. Malheureusement, la documentation liée au téléchargement comporte 2200 pages de Javadoc d'API auto-générées. Au final, la seule documentation pour l'utilisateur est une unique page de tutoriel.

Le langage PHP n'est distribué avec aucune documentation. Si vous voulez la documentation, vous devez aller sur une page séparée pour les obtenir. Pire, seule la documentation du cœur est disponible au téléchargement, sans les annotations utiles des utilisateurs (voir « Ne pas accepter les remarques des utilisateurs » plus bas), et ce n'est pas le même format facile à parcourir que celui qui est disponible en ligne.

Les projets open source ne peuvent pas supposer que les utilisateurs ont accès à Internet quand ils ont besoin de la documentation. Le mode avion existe toujours. De toute façon, vous ne souhaitez pas que l'utilisateur dépende uniquement du

fait que votre site web est disponible ou non. Au moins à deux reprises durant les derniers mois, le wiki de Solr était indisponible au beau milieu de ma journée de travail alors que je recherchais des informations sur un problème de configuration épineux.

Un projet qui fait les choses bien est Perl et son dépôt de module CPAN. La documentation pour chaque module est disponible soit à search.cpan.org ou metacpan.org dans un format hypertexte facile à lire. Pour la consultation hors-ligne, la documentation de chaque module est intégrée dans le code lui-même, et quand le module est installé sur le système d'un utilisateur, la documentation locale est créée sous forme de pages man. Les utilisateurs peuvent aussi utiliser « perldoc Module::Name » pour obtenir la documentation depuis le shell. En ligne ou hors-ligne : c'est votre choix.

4. La documentation non installée avec le paquet

Ce problème est généralement une erreur des paquageurs, pas des auteurs du projet. Par exemple, sur Ubuntu Linux, la documentation du langage Perl est séparée, ce sont des paquets optionnels pour le langage lui-même. L'utilisateur doit savoir qu'il doit explicitement installer la documentation de la même façon que le langage principal ou il n'y aura pas accès quand il en aura besoin. Ce compromis de quelques mégabites d'espace disque au détriment de la documentation à portée de main de l'utilisateur dessert tout le monde.

5. Le manque de captures d'écran

Il n'y a pas de meilleur moyen d'obtenir l'attention potentielle d'un utilisateur, ou d'illustrer un usage correct, qu'avec des captures d'écran judicieuses. Une image vaut mieux qu'un long discours, c'est encore plus important sur Internet parce que vous ne pouvez obtenir d'un lecteur de lire plus de quelques centaines de mots en tout.

Les captures d'écran accompagnant le texte sont inestimables

pour guider l'utilisateur voulant faire les choses au mieux. Une capture d'écran lui permet de comparer visuellement ses résultats à ceux de la documentation et va le rassurer d'avoir exécutée la tâche correctement ou l'aidera à trouver facilement ce qui ne va pas.

Il est de plus en plus commun de trouver des vidéos sur le site internet d'un projet pour en donner un aperçu, et c'est génial. Tout autant que le fait d'avoir une vidéo pour chaque étape d'un processus complexe. Le projet Plone, par exemple, a un site entier dédié aux tutoriels vidéos. Cependant, les vidéos ne peuvent pas remplacer les captures d'écran. Un utilisateur veut voir rapidement l'allure des captures d'écran sans s'arrêter devant une vidéo. Les vidéos n'apparaissent également pas dans une recherche *Google Image*, à l'inverse des captures d'écran.

6. Le manque d'exemples réalistes

Pour les projets basés sur du code, l'analogue des captures d'écran sont de bons et solides exemples du code en action. Ces exemples ne devraient pas être abstraits, mais directement issus du monde réel. Ne créez pas d'exemples bateaux plein de « nom de la démo ici » et lorem ipsum. Prenez le temps de créer des exemples signifiants avec une histoire d'utilisateur qui représente la façon dont votre logiciel résout un problème.

Il y a de bonnes raisons de vous embêter avec des problèmes de maths en classe. Ils permettent d'appliquer ce que vous avez appris.

Disons que j'ai écrit un module d'un robot Web, et que j'explique la méthode follow_link. Je pourrais montrer la définition de la méthode ainsi :

```
$mech->follow_link( text_regex => $regex_object, n => $link_index );
```

Mais admirez à quel point cela devient évident en ajoutant de la réalité dans l'exemple.

```
# Suit le 2e lien où la chaîne de caractères « download » apparait
$mech->follow_link( text_regex => qr/download/, n => 2 );
```

Les noms des variables $regex_object et $link_index sont maintenant compréhensibles par le lecteur.

Bien entendu, vos exemples ne doivent pas être aussi brefs. Comme Rich Bowen du projet Apache le souligne, « Un exemple correct, fonctionnel, testé et commenté l'emporte sur une page de prose, à chaque fois. »

Montrez autant que possible. L'espace n'est pas cher. Créez une section dédiée aux exemples dans la documentation, ou même un livre de cuisine. Demandez aux utilisateurs d'envoyer du code qui fonctionne, et publiez leurs meilleurs exemples.

7. Liens et références inadéquats

Vous avez les hyperliens. Utilisez-les.

Ne pensez pas, parce que quelque chose est expliquée dans une certaine partie de la documentation, que le lecteur a déjà lu cette partie, ou bien qu'il sait où elle se trouve. Ne vous contentez pas de signaler que cette partie du code manipule des objets frobbitz. Expliquez brièvement lors du premier usage de ce terme ce qu'est un objet frobbitz, ou donnez le lien vers la section du manuel l'expliquant. Encore mieux, faites les deux !

8. Oublier les nouveaux utilisateurs

Il arrive trop souvent que l'écriture de la documentation soit rédigée à partir du point de vue de son auteur, alors que es nouveaux utilisateurs ont besoin de documentation d'introduction pour les aider.

L’introduction devrait être une page séparée de la documentation, idéalement avec des exemples qui permettent à l’utilisateur de réussir quelques manipulations avec le logiciel. Pensez à l’excitation que vous ressentez quand vous commencez à jouer avec un nouveau logiciel et qu’il vous permet de faire quelque chose de cool. Faites que ça arrive aux nouveaux utilisateurs également.

Par exemple, un package graphique pourrait présenter une série de captures d’écran qui montrent comment ajouter des données dans un fichier, comment faire intervenir le grapheur, et ensuite montrer les graphes obtenus. Une bibliothèque de codes pourrait montrer quelques exemples d’appels à la bibliothèque, et montrer le résultat obtenu. Pensez simplicité. Offrez une victoire facile. Le texte devrait introduire les termes aux endroits appropriés, avec des liens vers une documentation plus détaillée sur le long terme.

Un document de démarrage séparé donne à l’utilisateur une compréhension rapide du logiciel. Il garde aussi les explications d’introduction en dehors de la partie principale de votre documentation.

9. Ne pas écouter les utilisateurs

Les développeurs doivent écouter les utilisateurs de la documentation. La chose évidente est d’écouter les suggestions et requêtes des personne qui utilisent activement votre logiciel. Quand un utilisateur prend le temps d’écrire un mail ou de poster quelque chose comme « ça aurait pu m’aider à mieux installer le programme s’il y avait eu une explication ou des liens au sujet des pilotes de la base de données », prenez ce message au sérieux. Pour chaque utilisateur vous envoyant un mail pour un problème, vous devez vous attendre à ce que dix utilisateurs silencieux aient le même problème.

Il est important d’écouter les problèmes des utilisateurs et d’en chercher les causes. S’ils ont souvent des problèmes pour

effectuer des mises à jour groupées de bases de données, la première chose à faire est d'ajouter une question à la FAQ (vous avez bien une FAQ, n'est-ce pas ?) qui traite de ces questions-là. Cependant, la question peut aussi indiquer que la section traitant des mises à jour de base de données n'est pas assez claire. Ou peut-être qu'il n'y a pas de référence à cette section depuis la vue d'ensemble introductive du document, avec pour conséquence que vos utilisateurs ne pensent jamais à lire le reste du manuel.

En plus d'aider plus de gens à découvrir à quel point votre projet est utile, ça diminuera aussi la frustration de la communauté déjà existante. Si votre liste de diffusion, forum ou canal IRC est remplie de personnes qui posent toutes les mêmes questions idiotes (ou pas si idiotes) au point que tout le monde devient lassé d'y répondre, sachez reconnaître que ce sont des questions récurrents pour la FAQ, et mettre les réponses à un endroit facile à trouver aidera tout le monde à se concentrer sur des choses plus amusantes.

Gardez aussi un œil sur les questions des forums externes. Consultez les sites comme StackOverflow régulièrement, et placez une Google Alert sur votre nom de projet pour être maintenu au courant des discussions concernant votre projet sur Internet.

10. Ne pas accepter les entrées des utilisateurs

Si votre projet a une base d'utilisateur assez grande, il peut être judicieux d'incorporer les commentaires des utilisateurs directement dans la documentation. Le meilleur exemple que j'ai pu voir est celui donné par PHP. Chaque page de la documentation permet aux utilisateurs authentifiés d'ajouter des commentaires sur la page, aidant ainsi à clarifier certains points ou ajoutant des exemples qui ne sont pas dans la documentation principale. L'équipe PHP laisse aussi le choix au lecteur de lire la doc avec ou sans les commentaires des autres utilisateurs.

Aussi pratique cela soit-il, cela nécessite de la maintenance. Les commentaires doivent être éliminés de temps en temps pour éviter la prolifération. Par exemple, la page de la documentation PHP sur comment lancer PHP depuis la ligne de commande inclut 43 commentaires d'utilisateurs qui remontent à 2001. Les commentaires écrasent la documentation principale. Les commentaires devraient être archivés ou supprimés, tout en incluant les points les plus importants dans la documentation principale.

Un wiki est également une bonne approche. Cependant, si votre wiki ne permet pas à l'utilisateur de télécharger toutes les pages en une seule grosse archive (cs. point n°3 ci-dessus), alors vos utilisateurs sont à la merci de votre connexion internet et du serveur hébergeant le projet.

11. Impossibilité de voir ce que fait le logiciel sans l'installer

Au minimum, chaque projet de logiciel nécessite une liste de fonctionnalités et une page de captures d'écran pour permettre au potentiel utilisateur intéressé de savoir pourquoi il devrait l'essayer. Aidez l'utilisateur, comparant les différents paquets à utiliser, à voir pourquoi cela vaut la peine de prendre le temps de le télécharger et de l'installer.

Les images sont un bon moyen de faire cela. Votre projet devrait avoir une page « Captures d'écran » qui montre des exemples de l'outil en action (cf. point n°5 ci-dessus). Si votre projet se résume uniquement à du code, comme une librairie, alors il devrait y avoir une page d'exemples montrant ce code utilisant le projet.

12. S'appuyer sur la technologie pour votre rédaction

Trop souvent, les auteurs de logiciels utilisent des systèmes de documentation automatisés pour faire leur travail. Ce système automatisé rend les choses plus facile à maintenir,

mais il ne supprime pas la nécessité d'un travail d'écriture humain.

Le pire des cas concerne le changelog, qui n'est rien de plus qu'un dump des messages de commit du système de gestion de version, mais sans un résumé qui l'explique. Un changelog devrait lister les nouvelles fonctionnalités, les problèmes résolus et les incompatibilités potentielles. Sa cible est l'utilisateur final. Un log de commit est pratique et simple à générer pour les personnes travaillant sur le projet, mais ce n'est pas ce dont l'utilisateur a besoin.

Jetez un œil à cette page de la documentation de Solarium, une interface PHP pour le moteur de recherche Solr. Tout d'abord, l'avertisemment prend la moitié supérieure de l'écran, ne donnant aucune information au lecteur. Ensuite, il n'y a vraiment rien de véritablement descriptif sur la page que la liste des noms de fonctions. Il n'y a aucune explication sur les différentes méthodes, ni de liens indiquant où trouver l'explication. Les pages générées automatiquement sont jolies, et elles pourraient ressembler à de la documentation, mais elles n'en sont pas.

13. Arrogance et hostilité vis-à-vis de l'utilisateur

L'attitude du type RTFM (Read The Freaking Manual) est mauvaise pour votre projet et votre documentation.

C'est le summum de l'arrogance que de croire que tous les problèmes qui ont trait au fait que quelqu'un ne sache pas utiliser votre logiciel sont de la faute de l'utilisateur.

Même s'il est probablement vrai que les utilisateurs peuvent trouver leurs réponses dans votre documentation mais ne le font pas, il est stupide de penser que c'est la faute de l'utilisateur. Peut-être votre documentation est-elle mal écrite, ou difficile à lire, ou présente mal à l'écran. Peut-être avez-vous besoin d'améliorer la section « Mise en route »

(lien #8 ci-dessus) qui explique ce que le logiciel a pour but de faire. Peut-être que certaines parties d'information nécessitent d'être répétées à de multiples endroits de la documentation.

N'oubliez pas que les nouveaux utilisateurs de votre logiciel peuvent arriver sur votre projet sans rien n'en savoir. Votre documentation doit faire de son mieux pour s'assurer que cette ignorance soit facilement résolue.

Synthèse

Je suis sûr que vous avez déjà eu affaire à quelques-uns de ces problèmes listés ci-dessous, et peut-être que pour d'autres vous n'y avez pas pensé. Faites-nous connaître les problèmes que vous avez rencontrés dans les commentaires ci-dessous, sachant qu'il ne s'agit pas de pointer du doigt certains projets en particulier.

Surtout, j'espère que si vous reconnaissez un problème dans la documentation de vos projets, vous prendrez la peine d'améliorer la situation. Heureusement, améliorer la documentation est une manière idéale de faire participer les nouveaux arrivants dans votre projet. On me demande souvent : « Comment puis-je commencer dans l'open source », et je recommande des améliorations dans la documentation comme une bonne manière de commencer.

Faites-en sorte que ce soit aussi facile que possible, pour les novices comme les plus anciens, de savoir où il est nécessaire de travailler la documentation. Créez une liste des tâches, par exemple dans votre système de suivi des bogues, qui explique ce qui a besoin d'être amélioré. Soyez précis dans ce que sont vos besoins. Ne vous contentez pas de dire que vous avez besoin d'exemples, sans plus de précision. Créez des tâches spécifiques, comme « ajoutez un code d'exemple sur le fonctionnement de la tâche X », « ajouter une capture d'écran du générateur de rapports » ou « ajouter des

informations de dépendances au fichier README/LISEZ-MOI ». Les contributeurs souhaitent aider mais trop souvent ils ne savent pas par où commencer.

La documentation n'est pas la partie la plus glamour d'un projet open source, et pour la plupart d'entre nous ce n'est pas amusant. Mais sans une bonne documentation, les utilisateurs ne sont pas servis comme ils pourraient l'être, et votre projet en souffrira sur le long terme.

Crédit photo : Rosalux Stiftung (Creative Commons By)