Introduction aux Logiciels Libres – TD n° 7

Produire de nouvelles versions d'un logiciel libre

Exercice 1 – Communiquer sur une nouvelle version

1. De quelles informations les utilisateurs peuvent-ils se servir lorsqu'ils souhaitent mettre à jour leur installation d'un logiciel libre ?
2. Quel est le mécanisme le plus commun de numérotation des versions ? Quel sens est attribué à chaque composant du numéro de version ?
3. Quels sont les six catégories de modifications recommandées par le standard "Keep a Changelog" ? Lesquelles sont susceptibles d'encourager les utilisateurs à mettre à jour leur logiciel ? Lesquelles sont susceptibles de les retenir de mettre à jour immédiatement ?
4. Quel usage les mainteneurs peuvent-ils avoir des notes de version du logiciel dont ils/elles sont responsables ?

Exercice 2 – Préparer une nouvelle version

1. Comment les mainteneurs d'un projet libre peuvent-ils procéder pour stabiliser une nouvelle version sans interrompre le développement de nouvelles fonctionnalités pour les versions ultérieures ?
2. Quelles sont les différentes stratégies possibles pour que les correctifs atteignent toutes les branches auxquels ils s'appliquent ? Quels sont leurs avantages et leurs inconvénients ?
3. Quel est le but de sortir des versions préliminaires (versions beta, *release candidates*) ?

Exercice 3 – Cycles de sortie de versions L'article "Why and How Should Open Source Projects Adopt Time-Based Releases ?", par Michlmayr, Fitzgerald et Stol, contient des extraits d'entretiens avec des mainteneurs de logiciels libres :

> In an open source environment, the feature-based strategy is just basically impossible unless you want to wait forever [...], which is what happens to a lot of projects. Some haven't had a release in five years. You cannot tell anybody to do anything.
>
> For developers, regular releases are like trains : if you miss one, you know that there will be another one in the not-too-distant future.
>
> It's a problem if you start packing too many features into a release. The features are there, but they are simply not stable.
>
> If you can keep the development more tightly linked to the actual usage, you will get much better feedback in terms of bugs and development direction.
>
> If you insist on doing all development work on [the main development branch], then yes, of course you have a problem because you need a certain amount of time to get anything done.

1. Qu'est-ce qui fait que la stratégie du cycle basé sur les fonctionnalités (*feature-based*) est plus difficile à utiliser dans le contexte d'un logiciel libre que pour un logiciel propriétaire ?
2. Quels sont les principaux risques associés à ce type de cycles / les bénéfices associés aux cycles basés sur le calendrier (*time-based*) ?
3. Pourquoi l'absence de régularité dans la sortie de nouvelles versions peut-elle décourager certains contributeurs ?
4. Quelles pratiques de développement sont-elles rendues nécessaires par l'adoption d'un cycle basé sur le calendrier ?
5. Pour quel type de projets l'adoption d'un cycle basé sur le calendrier serait-elle inutile ?