Zeste de savoir

Agit avec prudence — 97 choses à savoir quand on est un programmeur

20 mars 2019

Table des matières

1. Introduction

Introduction

Il existait, il y a de cela deux-trois ans, un site dont le nom était « 97 Things Every Programmer Should Know » mais qui aujourd'hui semble avoir disparu (bien qu'on puisse trouver le Gitbook). Ce site était composé de **conseils divers et variés sur la programmation**, écrits par différents programmeurs et librement diffusé sous licence Creative Commons Attribution Non Commercial Share Alike 3.0 . Je me permets donc de partager ici avec la communauté de Zeste de Savoir des traductions de ces différents articles. N'hésitez pas à commenter.

L'article traduit aujourd'hui est Agit avec prudence par Seb Rose.

2. Agit avec prudence

> Peu importe ce que tu entreprends, agis avec prudence et considère les conséquences.
>
> *Anon*

Même si le planning semble confortable quand l'itération démarre, on sait qu'à un moment ou un autre, on sera sous pression. Dans ces cas-là, on est face au dilemme entre « faire bien » et « faire vite » et bien souvent, c'est ce dernier qui l'emporte et l'on s'empresse de le justifier en arguant qu'on repassera plus tard faire les choses proprement. Et quand cette promesse est faite, à soi-même, à son équipe ainsi qu'au client, on pense sincèrement la tenir. Mais bien trop souvent, l'itération suivante vient avec son lot de problèmes qui focalisent l'attention. Ce travail sans cesse reporté est connu sous le nom de **dette technique** et n'est pas notre ami. C'est ce que Martin Fowler nomme, dans sa taxonomie de la dette technique , la dette technique délibérée, en opposition à la dette technique accidentelle.

La dette technique fonctionne comme un emprunt : sur le court-terme c'est avantageux, mais il faut **payer des intérêts** jusqu'à ce qu'il soit remboursé. Ces raccourcis dans le code rendent l'ajout de fonctionnalités ou la refactorisation plus difficiles. Ce sont des nids à problèmes et

rendent les cas de tests complexes. Plus on la laisse de côté, plus la situation empire. Avec le temps, quand on cherche enfin à corriger un problème, on peut se retrouver avec tout un tas de choix de design pas tout à fait correct venus se superposer au problème d'origine et qui rendent le refactoring et la correction beaucoup plus complexes. En fait, bien souvent, c'est quand les choses sont à tel point catastrophique qu'on se décide à régler le problème. Mais cette correction est tellement dure à implémenter qu'on ne veut pas, ou ne peut pas, prendre le risque ni le temps de le faire.

Il y a forcément des moments où l'on accepte de subir cette dette pour tenir les délais ou pour implémenter une partie d'une fonctionnalité. Essaye de ne pas te retrouver dans ce cas. Bien sûr, si la situation l'exige vraiment, alors vas-y. Mais, MAIS, tu dois suivre cette dette technique à la trace et la rembourser le plus rapidement possible, sous peine de voir les choses rapidement se dégrader. Dès que tu acceptes ce compromis, note-le sur une *task card*, ou créé une issue pour t'assurer qu'elle ne soit pas oubliée.

Si la dette est réglée à l'itération suivante, le coût aura été faible. Par contre, **laisser la dette traîner ne va faire qu'augmenter les intérêts**, qui doivent être suivis pour garder à l'esprit le coût. Par ce moyen, on souligne le coût de la dette technique sur la *business value* du projet et ça permet de mieux prioriser son remboursement. Le calcul de la dette et la trace des intérêts dépendront du projet, mais tu dois la suivre.

Rembourse la dette technique **dès que possible**. Ne pas le faire serait imprudent.

3. Mon mot à moi

Dans mes deux premiers jobs, c'était typiquement le cas. Il s'agissait d'applications embarquées avec des bases de code vieilles de 10 et 20 ans, respectivement. Les hacks, les raccourcis et les trucs pas propres étaient légions. Chaque fois qu'on s'y collait, il y avait soit trop de changements d'un coup, soit pas assez de temps parce qu'une super nouvelle fonctionnalité de la mort qui tue devenait ultra-prioritaire ASAP trop urgent. Dans les deux cas, aucune évaluation et aucun traçage de la dette technique n'étaient fait. Cela était donc invisible aux yeux des décideurs.

Plus précisément, dans le premier cas, il y avait des parties trop fortement couplées, des fonctions de plusieurs centaines de lignes parfois, des cas particuliers qui se multiplient. L'ajout de tests unitaires était ainsi compliqué et lent.

? Votre parole

Et vous, qu'en pensez-vous ?