Dette technique et nos développements

Comme (presque) tous les jours, je parcours quelques sites internet pour faire ma veille technologique. J'suis tombé sur un article qui m'a parlé et j'aimerais profiter de remettre une compresse sur la dette technique. Dans douze jours, je ne serai plus responsable du Service Développement Digital et une 30aine d'applications seront désormais dans le Service Non-Vie de Patricio de Oliveira Lino. Il faut que je profite des douze derniers jours pour essayer de passer le message (encore une fois).

Le saviez-vous ?

> *La dette technique est une métaphore pour des raccourcis ou des hacks que nous faisons et qui rendent plus difficile le changement et la maintenance.*

cf Steve c/o Ardalis

Dans certains cas, il est nécessaire de déclarer banqueroute, et il ne sera jamais possible de rembourser la dette. Ce cas doit être rare et nous n'allons pas nous concentrer là-dessus.

Arrêter l'hémorragie.

Supposons que vous travaillez dans un projet avec une propension à créer de la dette technologique. La première action que vous devez faire est d'arrêter immédiatement l'hémorragie afin **d'éviter de créer de la dette supplémentaire**. Des petits changements au quotidien peuvent faire la différence sur le long terme. Dites-vous que la dette technique augmente à chaque fois que vous rajoutez du code non (automatiquement) testé, difficile à tester, fortement couplé, etc... La première étape est d'arrêter cette pratique. Le nouveau code doit suivre des standards, doit être testé automatiquement, est *well-factored* (SOLID, YAGNI, etc). Même si vous travaillez sur une énorme codebase, il est quand même possible d'écrire du nouveau code correctement designé. Maintenez le code legacy en rajoutant du beau code, et pas en changeant le code existant.

Créer de la dette technique coûte de l'argent à l'entreprise, et lorsqu'elle n'est pas sous contrôle, nous en sommes responsables.

Refactorer tout en rajoutant de la valeur.

Le but même du refactoring est de se débarrasser de la dette technique. Il y a plusieurs techniques que vous pouvez utiliser pour identifier ce qui contribue à la dette technique. Voici deux cours Pluralsight qui traitent du problème: Refactoring for C# Developers et Refactoring Fundamentals. La question suivante ressort très régulièrement : quand faut-il refactorer ?

Les clients et parties prenantes attendent des équipes de développement qu'elles fournissent des nouvelles fonctionnalités qui sont visibles. Si nous cessons d'apporter de la valeur et que nous passons notre temps à *nettoyer / refactorer* le code, je ne pense pas que nous aurons le soutien de nos collègues très longtemps (et, je le rappelle, ce sont eux qui ont le porte-monnaie dans les mains, pas nous). De plus, il n'est pas certain que *refactorer pour refactorer* soit vraiment ce qui apporte le plus de valeur pour l'entreprise. Dans nos codebases, il y a des zones que nous touchons rarement et qui ne valent pas la peine d'être refactorées. Toutefois, les zones sur lesquelles nous travaillons régulièrement (car il y a des nouvelles fonctionnalités demandées par le métier qui la touche), alors cela vaut la peine de passer du temps pour l'améliorer.

En résumé:

- refactorer pour refactorer, ça ne vaut probablement pas grand chose,
- la zone de la codebase n'est jamais touchée, alors ça ne vaut probablement pas la peine de refactorer,
- des nouvelles fonctionnalités – demandées par le métier – touche des zones sensibles de la codebase, alors il faut se poser la question si on peut réduire la dette technique.

Posez-vous la question dans quel "quadrant" se trouve la dette de votre projet ?

Les métriques, c'est toujours bon.

Une bonne manière de se rendre compte si nous améliorons notre qualité est d'avoir des métriques et de les observer sur le temps. Je ne souhaite pas rentrer dans les détails (i.e. les avantages et les inconvénients) de toutes les métriques que je vais donner, l'importance est de se rendre compte qu'on s'améliore sur le temps :

- Le nombre de tests automatiques qui passent,
- La fréquence des déploiements,
- La fréquence de création de nouveaux bugs,
- La fréquence des rollbacks dans les déploiements,
- Plus toutes les métriques données par SonarQube (code smell, mauvaises pratiques, couverture de tests, duplications, etc).

Pour conclure.

Le remboursement de la dette technique ne passe pas uniquement par les deux programmes de transformations. Il passe aussi par l'hygiène de travail que vous pouvez avoir.
Prenez un peu de recul et posez-vous la question : comment est-ce que nous pouvons nous améliorer et en faire bénéficier l'entreprise ?

Article fortement ~~copié~~ inspiré de https://ardalis.com/attacking-technical-debt.