La dette technique expliquée !

Xavier Blanc – Expert Scientifique, ProMyze – Professeur, Université de Bordeaux

Ce « white paper » explique clairement ce qu'est la « dette technique » !

Qu'est-ce que la dette ?

Originalement la dette technique était une métaphore pointant du doigt « le code mal écrit qu'on écrira mieux plus tard » [Cun 92]. On comprend bien alors cette métaphore car le fait de reporter l'écriture propre d'un code engendre un coût qui augmentera avec le temps. Enfin ce report « à plus tard » aura peut-être même des conséquences très négatives dans le futur.

Le concept de dette technique n'a cessé d'évoluer et englobe maintenant aussi bien la documentation, les tests, l'architecture et évidemment le code source. Ce qui caractérise la dette est (1) qu'elle est invisible, c'est-à-dire qu'elle n'a pas d'impact immédiat sur les fonctionnalités du logiciel, et (2) que son effet négatif n'est que potentiel et n'apparaîtra que plus tard, si jamais il apparait. La dette est alors généralement définie comme étant « le résultat invisible et potentiellement négatif que les actions du passé peuvent avoir sur le futur » [Kru 12].

Cette définition relativement vague nécessite quelques exemples concrets :

- L'exemple le plus simple de dette dans le code source est celui d'une condition 'if' qui a été codée mais sans que le cas 'else' n'ait été traité. On voit fréquemment ce genre de dette avec parfois un message de type 'TODO' dans le code. C'est une dette technique car le développeur a reporté l'écriture du code 'else', pas forcément pour de mauvaises raisons d'ailleurs. Si ce développeur attend trop il sera alors plus compliqué pour lui, et encore plus pour un autre développeur, de coder cette partie manquante. De plus, il n'est pas impossible que cette portion de code ne devienne la source d'un bogue important dans le futur.
- La dette dans la documentation vient de portions de logiciels non documentées. Le temps passant il sera alors de plus en plus couteux de rédiger ces documentations manquantes. Enfin, le manque de documentation pourra avoir un impact négatif sur la maintenance de l'application.
- Les tests peuvent être source de dette mais il ne faut pas confondre la dette dans les tests et le manque de tests qui n'est absolument pas une dette technique. Ne pas tester est visible et a un impact immédiat en cas de non détection de bogue, ce n'est donc pas une dette. Une dette dans les tests est par exemple le fait de ne pas chainer les tests dans leur exécution alors que cela ferait gagner du temps. Plus il y aura de nouveaux tests dans le futur plus ils mettront du temps à s'exécuter. La dette risquera alors de couter cher en temps de calcul. Pire elle rendra peut-être même l'exécution de certains tests impossible par manque de temps.
- La dette d'architecture est la plus compliquée car les décisions d'architecture ne sont quasiment jamais reportées à plus tard. Par contre, il arrive souvent qu'une décision d'architecture prise à un moment ne doive être remise en question plus tard. C'est le cas notamment de l'exploitation de bibliothèques ou de frameworks. On voit trop souvent des projets trainer des frameworks techniques obsolètes qui ont une faible plus-value. Changer ces framework constitue alors un cas classique de dette technique d'architecture.

Quel est le coût de la dette ?

Qui dit dette, dit coût de la dette. Sur ce sujet il est important de bien identifier les différents coûts de la dette qui permettront alors de bien mesurer l'importance d'une dette.

Le *coût principal* de la dette est celui de la correction de la dette. En effet la dette est principalement causée par le fait de reporter des actions de développement. Le coût de ces actions ne fait qu'augmenter. Plus on attend, plus il sera difficile de corriger la dette et plus le coût principal sera important. Certains modèles prédictifs permettent d'estimer ce coût en fonction de l'action à réaliser [Cur12][Let12]. Ce sont ces modèles prédictifs qui permettent à des outils tels que Sonar ou CAST de quantifier le coût de la dette.

Le *coût risqué* de la dette est celui qui serait causé par l'impact négatif d'une dette. Ce coût est malheureusement impossible à calculer de manière générale car il dépend essentiellement du logiciel qui porte la dette. Certaines approches proposent d'estimer le niveau de risque de la dette sans pouvoir calculer son coût risqué. Elles proposent alors des niveaux de risque en fonction des différents types de dette et de leur importance dans un logiciel. Par exemple, ces approches considèrent que si au moins 80% du code d'une application est documenté, cela ne correspond pas à un risque important d'une dette de documentation mais plutôt à un risque modéré. Ces niveaux permettent alors d'estimer grossièrement le niveau du risque mais pas son coût !

Le *coût réel* correspond au coût réellement payé par une entreprise pour sa dette. Il est très facile à calculer pour le passé et est difficilement estimable pour le futur. Pour le passé, il suffit de sommer le coût principal qui a été payé par l'entreprise (le coût des réparations) avec le coût risqué qu'il a fallu malheureusement payer si des impacts négatifs se sont déroulés (le coût des dégâts). Seules les entreprises qui ont un historique de coûts réels peuvent prétendre estimer leur futur coût réel.

Ces trois coûts résument le compromis de la dette. Tout industriel souhaite en effet minimiser le coût réel de la dette. Le challenge est donc de payer le moins possible du coût principal en se préservant du coût risqué. Pour faire face à ce compromis, il faut alors mettre en place une réelle stratégie de la dette.

Quelle stratégie employer ?

Il existe 4 stratégies possibles pour faire face à la dette technique.

La stratégie de *la dette non assumée* consiste à ne pas payer le coût principal et voir même à ne pas l'estimer. L'issue fatale de cette stratégie est alors de devoir payer le coût risqué de la dette au fur et à mesure que les effets négatifs ne surviennent. Cette stratégie est malheureusement adoptée par de nombreuses entreprises et les rapports catastrophiques de règlements exorbitants sont nombreux. Il est à noter qu'il n'existe pas d'étude permettant de connaître le coût risqué moyen, ni le temps moyen avant qu'un effet négatif ne survienne. De notre point de vue, cette stratégie est suicidaire.

La stratégie de *la dette par niveau de risque* consiste à payer régulièrement une partie du coût principal en se focalisant sur les niveaux de risque de la dette. Cette stratégie se base sur les approches qui permettent d'identifier ces niveaux de risque et vise des seuils à atteindre. A titre d'exemple, il s'agira de commenter suffisamment le logiciel afin d'atteindre le fameux seuil des 80%. De notre point de vue cette stratégie n'est pas optimale car elle n'est que faiblement en relation avec le coût risqué. De plus cette stratégie n'est pas facile à mettre en place car elle souffre d'un manque de visibilité quant aux

objectifs à atteindre (pourquoi 80% de documentation?). Dans les pires cas, nous avons même observé des comportements déviants visant uniquement à atteindre le seuil en réparant très sommairement la dette (ajoutant des commentaires sans aucun sens pour atteindre le seuil de 80%).

La stratégie de *la dette par analyse du coût risqué* consiste à payer régulièrement une partie du coût principal de la dette mais en se focalisant sur les parties à fort coût risqué. Cette stratégie nécessite une analyse du coût risqué qui ne peut se faire que par des experts de la dette et des experts du logiciel qui porte la dette. Cette stratégie est la stratégie que nous conseillons.

La stratégie de *la non dette* consiste à payer l'intégralité du coût principal. Cette stratégie est forcément la moins risqué mais n'est peut-être pas la plus optimale car une grosse partie de la dette n'aura certainement jamais de conséquence négative. Payer l'intégralité du coût principal est certainement trop important.

En conclusion

Retenons que la dette technique représente tous les éléments logiciels (code, test, doc, archi, …) non terminés ou dépassés et qui sont du coup imparfaits. Il faudra donc très certainement les corriger sous peine de souffrir gravement de leurs défauts.

La dette technique ne doit pas être sous-estimée car son coût (coût risqué) peut être exorbitant. Nous conseillons à minima de réaliser une estimation du coût principal la dette.

Enfin, nous soutenons que la stratégie de la dette par analyse du coût risqué est la plus adaptée à une gestion optimale de la dette. Cette stratégie permet de mieux corriger la dette en traitant celle qui a le plus fort coût risqué.

Pour terminer, même si la dette est attachée à un logiciel, elle est avant tout l'affaire de tous les membres de l'équipe de développement. Une gestion de la dette ne peut pas se faire sans intégrer les aspects humains dans le management.

Références :

[Cun 92] Cunningham, Ward. "The WyCash Portfolio Management System." In Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum), 29–30. OOPSLA '92. New York, NY, USA: ACM, 1992.

[Cur 12] Curtis, Bill, Jay Sappidi, and Alexandra Szynkarski. "Estimating the Principal of an Application's Technical Debt." IEEE Software 29, no. 6 (2012): 34–42.

[Kru 12] Kruchten, Philippe, Robert L. Nord, and Ipek Ozkaya. "Technical Debt: From Metaphor to Theory and Practice." IEEE Software 29, no. 6 (2012): 18–21.

[Let 12] Letouzey, J. L., and M. Ilkiewicz. "Managing Technical Debt with the SQALE Method." IEEE Software 29, no. 6 (November 2012): 44–51.