Article – Point de vue

Traiter la dette technique : sortir du malentendu

Le chiendent du SI ? Une métaphore pour repenser la soutenabilité

"La dette technique, c'est comme les mauvaises herbes : on n'en vient jamais à bout."

Cette image est juste à plusieurs niveaux. L'idée d'un système d'information sans dette est une illusion, tout comme celle d'un jardin sans mauvaises herbes. La notion même de "mauvaise" herbe est discutable : les permaculteurs expliquent qu'il s'agit souvent de bio-indicateurs.

Le pissenlit, par exemple, pousse sur une terre trop tassée pour de l'herbe. Ses racines fragmentent le sol et préparent la venue d'autres espèces.
Les "mauvaises herbes" révèlent un déséquilibre, mais participent aussi à sa régulation.

Et si la dette technique était de cet ordre ?

Un signal plus qu'un problème. Un indicateur de ce qui, dans le système, s'est durci, s'est déconnecté ou a cessé d'évoluer.

Cet article propose une remise à plat pratique :

➜ Clarifier ce dont on parle
➜ Proposer une façon de voir la dette
➜ Proposer une façon de la piloter à l'échelle d'un portefeuille applicatif

La dette technique, une métaphore utile… mais piégeuse

Le terme "dette technique" est devenu un raccourci commode pour désigner tout ce qui freine un produit ou un système : du code mal conçu à la dépendance à un prestataire. Mais la métaphore a été appauvrie en chemin.

À l'origine, Ward Cunningham (1992)

Il y voyait un emprunt sur le futur : **un compromis temporaire pour livrer plus vite, à condition de "payer les intérêts" en améliorant ensuite le code.**

À l'usage on voit que c'est devenu un jugement moral implicite : la dette serait un défaut de qualité ou de professionnalisme des équipes technique, une faute à corriger.

Résultat :

On oscille entre deux extrêmes : le déni ("on verra plus tard") et la croisade ("refaisons tout proprement").

Dans les deux cas, on oublie l'essentiel ➜ la dette est le témoin de nos arbitrages. Elle fait partie de la vie du système.

Redéfinir la dette pour faire ressortir les enjeux

Ainsi, la dette n'est plus uniquement un problème de code, mais un problème de soutenabilité. Elle révèle notre (in-)capacité à faire évoluer le SI sans s'épuiser.

> Pour sortir de la culpabilisation et remettre l'enjeu business au cœur des décisions sur la dette technique, nous proposons de la définir comme **l'ensemble des résultats de décisions et d'actions (ou d'inactions) du passé qui rendent le système d'information difficile, coûteux ou risqué à maintenir et à faire évoluer.**

On peut la décomposer en quatre familles qui interagissent :

1. Fonctionnelle

Perte de compréhension du produit, documentation obsolète, "sédimentation fonctionnelle" (on empile des couches sans nettoyer).

2. Technique

Dépendances non maintenues, frameworks vieillissants, dette de testabilité ou de performance.

3. Organisationnelle

Dette de gouvernance, équipes projet tournantes, TMA off-shore, absence de propriétaire produit.

4. RH / Compétences

Dépendance à quelques sachants, turnover, perte d'attractivité des techno anciennes.

La dette devient ainsi un thermomètre organisationnel autant que technique.

Elle parle de flux, de décisions, de connaissance, de culture.

Un sujet d'entreprise

La dette fait partie du produit. Elle est la trace de la vie du système : de ses évolutions, de ses arbitrages, de ses contraintes. Vouloir l'éliminer, c'est comme vouloir figer le vivant.

Le sujet n'est donc pas moral ("bonne" ou "mauvaise" dette) mais stratégique :

➜ Jusqu'où notre dette est-elle soutenable ?
➜ Quelles zones de notre parc risquent de nous immobiliser demain ?
➜ Où faut-il investir pour préserver la capacité à livrer ?

Comment la dette s'accumule (et pourquoi elle persiste)

> La barque prend l'eau, il faut écoper, mais il faudrait peut-être aussi revoir l'étanchéité. Traiter la dette commence par comprendre ce qui la nourrit.

Pourquoi la dette technique s'accumule-t-elle ? Les facteurs sont multiples :

Pression court terme

On privilégie des livraisons rapides au détriment de la capacité à modifier demain (tests, architecture, automatisation). On « travaille à crédit » de notre productivité future.

Disparition de la connaissance

Stagnation fonctionnelle, docs périmées, départs non compensés ; la compréhension vivante du produit s'érode.

Décisions organisationnelles

TMA ou externalisation complète sans garde-fous, équipes qui tournent, "petit noyau" d'experts sur-sollicités.

Outillage et pratiques

Peu de tests, CI/CD fragile, dette de design, refactoring différé – on repousse la charge.

Sans compter les fausses solutions qui recréent de la dette ailleurs :

Réécriture magique : on remplace une dette par une autre (nouvelle techno, nouvelle équipe) sans traiter les causes.

Contre-feux organisationnels : Au-delà de tout ceci, dans de nombreuses équipes, il n'existe pas de culture craft, de conscience des enjeux, ni de rituels explicites ou d'espace pour entretenir le code : refactoring, revue de dépendances, culture "tidy-first" et revues de code croisées sont inconnues.

Voir avant d'agir : la cartographie du portefeuille

Sortir de l'immobilisme face à la dette demande un alignement au niveau de l'entreprise. Or la dette est invisible tant qu'elle n'est pas représentée. Une cartographie simple aide à rendre tangible la soutenabilité du parc applicatif.

Deux axes suffisent :

Criticité opérationnelle
– impact d'une panne ou d'un incident – et donc enjeu de maintenabilité.

Potentiel d'évolution
– fréquence et amplitude des changements prévisibles – et donc besoin de maintenabilité.

Potentiel d'évolution →

Prévention Fort potentiel / Faible criticité	**Correction** Fort potentiel / Forte criticité
Surveillance Faible potentiel / Faible criticité	**Contention** Faible potentiel / Forte criticité

Criticité opérationnelle →

Cette carte visuelle révèle des zones où la dette est maîtrisée ou supportable (faible criticité, peu d'évolutions) et où elle est dangereuse (forte criticité, fort besoin de changement).

Quatre politiques pour une gestion raisonnée

Toutes les dettes ne se remboursent pas de la même manière.

Une fois les produits cartographiés, on peut adopter une politique différenciée selon le quadrant :

Prévention – fort potentiel d'évolution, faible criticité

→ Entretenir, tester, investir dans la qualité.

Correction – fort potentiel et forte criticité

→ Prioriser le refactoring, allouer du budget technique.

Contention – faible potentiel, forte criticité

→ Stabiliser, sécuriser, documenter.

Surveillance – faible potentiel, faible criticité

→ Observer et laisser vieillir sans danger.

> Cette approche permet d'arbitrer les efforts : savoir où l'on nettoie, où l'on entretient, où l'on se contente de surveiller.

Traiter la dette, ce n'est pas "faire du ménage".

C'est piloter dans le temps la capacité à livrer. Ça se pilote au niveau du SI comme au niveau du produit, ou de l'équipe de dev.

Gérer la dette au quotidien : une discipline d'équipe agile

Les méthodes agiles et leur attachement à la livraison de valeur pour le client ont peut-être contribué involontairement à l'augmentation de la dette technique. Il s'agit pourtant d'un malentendu.

Je vois une erreur fréquente dans les équipes Scrum : considérer que le backlog de sprint est une version détaillée du sous-ensemble du backlog produit mis au planning.

Or, il n'y a pas de temps entre les sprints :

Tout ce qui est nécessaire pour maintenir indéfiniment le Scrum doit être pris en compte dans le sprint.

Ça veut dire différents types d'activités :

- Livrer les fonctionnalités prévues au planning (valeur court terme)
- Préparer les prochains sprints (découverte, analyse, cadrage)
- Réduire la dette technique (refactoring, test, documentation, outillage)
- Réduire la dette d'équipe (partage de connaissance, montée en compétence, culture craft)

Selon les phases du produit et les impératifs de livraison, la proportion de temps consacrée aux différentes activités variera — et devrait faire l'objet de discussions avec le PO — mais, de sprint en sprint, elle doit s'équilibrer à un niveau soutenable.

L'agilité d'une équipe se mesure aussi à sa capacité à livrer à chaque sprint un produit de qualité, et ceci nécessite un investissement dans la dynamique d'équipe, ses compétences, sa culture «tidy first» et sa connaissance du métier.

Passer à l'action : trois leviers concrets

1. Piloter et objectiver

- Cartographier la maintenabilité du SI
- La relier aux enjeux stratégiques business, en faire un sujet d'arbitrage au même titre que les fonctionnalités
- S'accorder au niveau entreprise ou produit sur la part d'allocation des ressources à la dette
- Faire de la maîtrise de la dette (= de la maintenabilité du SI) une responsabilité des équipes produit

2. Outiller et ritualiser

- Automatiser les contrôles de qualité (CI/CD, tests, métriques)
- Inclure la maintenabilité dans la definition of done
- Favoriser le refactoring continu plutôt que les grands chantiers annuels
- Mettre en place des revues périodiques de dette (technique, produit, RH)
- Suivre quelques indicateurs simples : maintenabilité, temps moyen de correction, volatilité du code, satisfaction développeur

3. Cultiver les compétences

- Organiser une rotation des collaborateurs build au support
- Identifier et résorber les dépendances critiques à quelques sachants («Truck factor»)
- Encourager la co-lecture, le binômage, la transmission
- Financer explicitement le temps d'apprentissage et de veille (c'est un investissement, pas un coût)

En conclusion : entretenir le vivant

Traiter la dette technique, ce n'est pas chasser les coupables ni viser un idéal de pureté logicielle.

> **C'est entretenir un SI vivant.**

Comme dans un jardin, les "mauvaises herbes" ne disparaissent jamais, mais un jardinier attentif sait lesquelles arracher, lesquelles observer, et lesquelles laisser faire leur travail.

De même, une DSI soutenable n'est pas celle qui a tout nettoyé, mais celle qui comprend sa dette, la pilote et la met au service de sa capacité à évoluer.

La dette technique n'est pas un problème à éliminer,

mais un système à comprendre et à piloter.