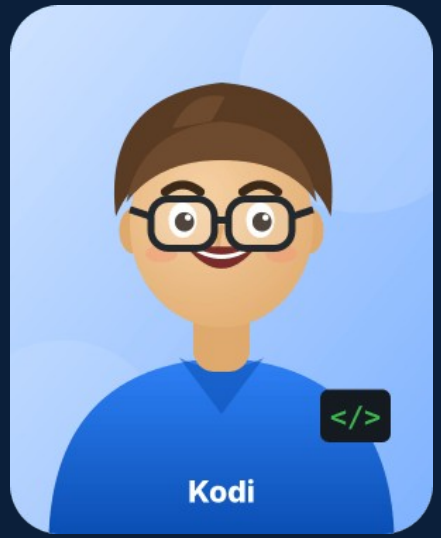
HEF-150 · MISSION 09 / 10

Qualité, dette et delivery

Aller vite aujourd'hui en négligeant la qualité, c'est emprunter : les intérêts se paient en lenteur demain. Savoir quand emprunter, quand rembourser et comment mesurer le flux, c'est piloter.





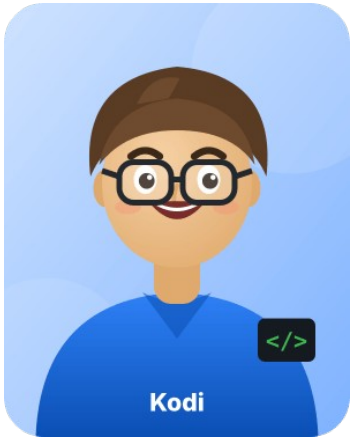
Badge : Gardien du rythme

110 min · avancé

L'histoire

Pendant le lancement de l'espace formateurs, HashBox livre en trois semaines grâce à des raccourcis : tests incomplets, copier-coller de code, configuration en dur. Six mois plus tard, la moindre modification prend trois fois plus de temps qu'avant. Chaque nouvelle fonctionnalité casse deux anciennes. Les développeurs disent « on n'ose plus y toucher ». La direction demande : « Pourquoi livrons-nous moins vite alors que l'équipe a grandi ? »

« C'est la dette technique, explique Sam. Elle n'est pas une honte : parfois on emprunte exprès pour apprendre vite. Mais comme toute dette, elle porte des intérêts. Si on ne les mesure pas, on découvre un jour qu'on ne fait plus que les payer. Il faut aussi regarder le flux du travail : combien de choses sont en cours en même temps, combien de temps chacune met, et quand on aura fini. »



L'histoire (suite)

Tu apprends à modéliser la dette technique et ses intérêts, à choisir quoi rembourser, à mesurer le flux (débit, temps de cycle, travail en cours), à comprendre pourquoi limiter le travail en cours accélère, et à prévoir une date de fin avec une simulation de Monte Carlo.

Ta mission

À la fin, je serai capable de…

1. définir la dette technique, ses quadrants et son intérêt, et la modéliser
2. comparer des stratégies de remboursement par simulation sur plusieurs trimestres
3. mesurer débit, temps de cycle et travail en cours, et appliquer la loi de Little
4. expliquer et simuler l'effet d'une limite de travail en cours (WIP)
5. prévoir une date de fin avec une simulation de Monte Carlo à partir de l'historique

Défi d'échauffement

Défi d'échauffement

Un restaurateur ne nettoie jamais les hottes pour gagner du temps. Au début, aucun problème. Après six mois, la cuisine est plus lente, risque un incendie, et le contrôle sanitaire ferme l'établissement une semaine. Qu'est-ce qui a été « emprunté » ? Quels sont les « intérêts » ? Dans quel cas ne pas nettoyer était un choix raisonnable ? Comment décider quand il faut passer des heures de nettoyage plutôt que des heures de service ?

Le concept

Qualité interne et externe

Qualité	Visible par	Exemples
Externe	les utilisateurs	bugs, performance, utilisabilité
Interne	l'équipe	lisibilité, tests, architecture, documentation

La qualité interne n'est pas un luxe : elle détermine la **vitesse de changement** à long terme. Il n'y a pas de compromis durable entre vitesse et qualité interne — il y a un **emprunt**.

La dette technique (Ward Cunningham)

La **dette technique** : choix de conception ou de réalisation qui accélère aujourd'hui et **ralentit les changements futurs**. Comme une dette financière :

Le concept

La dette technique (Ward Cunningham) (suite)

- le **principal** : coût pour corriger (en jours) ;
- l'**intérêt** : ralentissement **à chaque** modification touchant la zone concernée (en jours par mois) ;
- on peut **emprunter volontairement** (pour apprendre vite) mais il faut **rembourser**.

Quadrant de Fowler :

	Délibérée	**Involontaire**
Prudente	« nous livrons maintenant et nous nettoierons ; plan de remboursement »	« maintenant nous savons comment nous aurions dû faire »
Imprudente	« pas le temps de concevoir »	« c'est quoi une couche d'abstraction ? »

La dette **prudente** est un outil ; l'**imprudente** est un risque.

Le concept

Mesurer l'intérêt

Un **registre de dette** liste chaque élément avec : description, zone du code, **principal** (jours pour corriger), **intérêt** (jours perdus par mois), risque, propriétaire. On priorise par `intérêt / principal` (rendement du remboursement) et par proximité du code qu'on modifie souvent.

Signaux : temps de cycle qui croît, taux de bugs de régression, nombre de fichiers touchés par changement, peur de modifier, temps d'intégration d'un nouveau, couverture de tests, complexité.

Stratégies de remboursement

Stratégie	Principe
Règle du scout	laisser le code un peu plus propre qu'on ne l'a trouvé
Budget fixe	15 à 25 % de la capacité à chaque cycle

Le concept

Stratégies de remboursement (suite)

Stratégie	Principe
Refonte ciblée	rembourser les éléments à plus fort intérêt
Gel + grande refonte	souvent risqué et coûteux
Strangler (étrangleur)	remplacer progressivement derrière une façade

Le flux de travail

Un système de travail s'étudie par :

Mesure	Définition
Débit (*throughput*)	éléments terminés par période
Temps de cycle	du début du travail à la fin d'un élément
Travail en cours (*WIP*)	éléments commencés, non terminés

Le concept

Le flux de travail (suite)

Mesure	Définition
Âge du travail en cours	depuis combien de temps un élément est en cours

Loi de Little : `WIP moyen = débit × temps de cycle` (HEF-130). À débit constant, **réduire le WIP réduit le temps de cycle**. Un système avec trop de travail en cours passe son temps à changer de contexte et à attendre.

Limiter le travail en cours

Une **limite de WIP** (kanban) : on ne commence une nouvelle chose que lorsqu'une autre est terminée. Effets : moins de changements de contexte, **retour plus rapide**, goulots visibles, finir avant de commencer.

Le concept

Prévoir : Monte Carlo à partir de l'historique

« Quand aurons-nous fini ces 40 éléments ? » Au lieu d'une estimation unique, on utilise l'**historique de débit** (éléments terminés par semaine : 3, 5, 2, 4, …). On **rééchantillonne** ces semaines au hasard des milliers de fois pour simuler le futur : on obtient une **distribution** de dates de fin (50 %, 85 %, 95 % de confiance). On répond « 85 % de chances d'avoir fini en 11 semaines ou moins ».

Qualité du livrable : Définition de terminé

Une **définition de terminé** (*Definition of Done*) commune : code relu, tests automatiques verts, documentation à jour, déployable, métriques en place, aucune régression de budget. Sans elle, « terminé » veut dire « presque » et la dette s'accumule en silence.

Le concept

Livrer en continu

Lien avec DORA (HEF-120) : **petits lots**, intégration continue, déploiements fréquents, retour arrière rapide. La dette et la fréquence de déploiement sont liées : plus il est difficile de déployer, plus on regroupe, plus c'est risqué.

🧠 ***Astuce*** *: transforme « la dette technique ralentit » en chiffres : combien de jours ont été perdus ce mois-ci à cause de tel module ? Une dette qu'on sait chiffrer se négocie ; une dette vague se subit.*

Exemple pas à pas

Problème : simuler l'effet de la dette sur la vitesse de livraison avec différentes politiques de remboursement, vérifier la loi de Little et l'effet d'une limite de WIP, et prévoir une date de fin par Monte Carlo.

```
import random, statistics

# ---------------- 1. la dette réduit la vitesse : trois politiques sur 24 sprints
    ----------------
def simuler_dette(part_remboursement, sprints=24, capacite=20.0, dette_initiale=0.
   0, graine=1):
    rng = random.Random(graine)
    dette, fonctionnalites = dette_initiale, 0.0
    historique = []
    for s in range(sprints):
        vitesse = capacite / (1 + 0.03 * dette)                    # chaque poi
   nt de dette ralentit un peu (intérêt)
        remboursement = min(dette, part_remboursement * vitesse)
        production = (1 - part_remboursement) * vitesse
```

Exemple pas à pas (suite)

```
        dette += 0.30 * production * rng.uniform(0.8, 1.2) - remboursement
     # chaque point livré en crée un peu de nouveau (raccourcis, complexité)
        dette = max(dette, 0.0)
        fonctionnalites += production
        historique.append((fonctionnalites, dette, vitesse))
    return historique
for politique in (0.0, 0.10, 0.20, 0.40):
    h = simuler_dette(politique)
    print(f"remboursement {politique:4.0%} | fonctionnalités cumulées en 24 sprint
   s : {h[-1][0]:6.1f} | dette finale {h[-1][1]:5.1f} | vitesse finale {h[-1][2]:4
   .1f}")
h0, h20, h40 = simuler_dette(0.0), simuler_dette(0.20), simuler_dette(0.40)
assert h20[-1][0] > h0[-1][0] and h20[-1][2] > h0[-1][2]
   # sur 24 sprints, consacrer 20 % au remboursement livre PLUS
assert h0[5][0] >= h20[5][0] - 1e-9
   # …mais à court terme, ne rien rembourser livre plus

# ---------------- 2. loi de Little et limite de travail en cours ----------------
def kanban(wip_max, jours=3000, arrivees_par_jour=0.9, travailleurs=3, graine=3):
```

Exemple pas à pas (suite)

```
    rng = random.Random(graine)
    attente, en_cours, termines, temps_cycle = [], [], 0, []
    wip_total = 0
    for jour in range(jours):
        if rng.random() < arrivees_par_jour: attente.append(jour)
               # une demande arrive
        while attente and (wip_max is None or len(en_cours) < wip_max):
                # on démarre tant que la limite le permet
            en_cours.append({"debut": jour, "reste": rng.choice([1, 2, 3, 4, 6, 8]
), "arrivee": attente.pop(0)})
        wip_total += len(en_cours)
                  # WIP mesuré APRÈS les démarrages, AVANT la progression du jour
        if en_cours:
            partage = travailleurs / len(en_cours)
                 # la capacité est PARTAGÉE : plus de travail en cours = chaque él
ément avance moins
            for t in en_cours: t["reste"] -= partage * (0.6 if len(en_cours) > tra
vailleurs else 1.0)      # coût de changement de contexte
            for t in [t for t in en_cours if t["reste"] <= 0]:
```

Exemple pas à pas

Exemple pas à pas (suite)

```
                termines += 1; temps_cycle.append(jour - t["debut"] + 1); en_cours
    .remove(t)
     return termines / jours, statistics.mean(temps_cycle), wip_total / jours, stat
    istics.quantiles(temps_cycle, n=20)[18]
print("limite de WIP | débit/jour | temps de cycle moyen | p95 | WIP moyen | débit
     × temps de cycle")
resultats = {}
for limite in (None, 12, 6, 3):
     d, tc, w, p95 = kanban(limite); resultats[limite] = (d, tc, w)
     print(f"  {str(limite):10s} |   {d:5.2f}    |       {tc:6.1f} j        | {p95:
    5.1f} |  {w:5.2f}    |   {d * tc:5.2f}")
     if limite is not None: assert abs(w - d * tc) / w < 0.15
    # loi de Little vérifiée (régime stable)
assert resultats[3][1] < resultats[None][1] / 2                                  #
     limiter le WIP réduit le temps de cycle

# ---------------- 3. prévision Monte Carlo ----------------
HISTORIQUE = [3, 5, 2, 4, 4, 1, 6, 3, 4, 5, 2, 3]
     # éléments terminés par semaine, 12 semaines
```

Exemple pas à pas (suite)

```
def prevoir(reste, essais=10000, graine=7):
    rng, durees = random.Random(graine), []
    for _ in range(essais):
        fait, semaines = 0, 0
        while fait < reste:
            fait += rng.choice(HISTORIQUE); semaines += 1
        durees.append(semaines)
    durees.sort()
    return {p: durees[int(p / 100 * essais) - 1] for p in (50, 85, 95)}
p = prevoir(40)
moyenne = statistics.mean(HISTORIQUE)
print(f"40 éléments à terminer — moyenne historique {moyenne:.1f}/semaine : estima
   tion naïve = {40 / moyenne:.1f} semaines ; Monte Carlo : 50 % → {p[50]} sem., 8
   5 % → {p[85]} sem., 95 % → {p[95]} sem.")
assert p[50] <= p[85] <= p[95] and p[85] > 40 / moyenne                          #
    l'engagement à 85 % est plus prudent que la moyenne

# ---------------- 4. registre de dette : quoi rembourser avec un budget limité --
   --------------
```

Exemple pas à pas

Exemple pas à pas (suite)

```
DETTES = {"tests d'intégration manquants (paiement)": (6, 4.0), "module de factura
   tion copié-collé": (10, 5.0), "configuration en dur": (2, 1.5),
          "documentation absente de l'API": (3, 0.8), "dépendances obsolètes": (4,
    1.2), "schéma de base sans index": (3, 3.5), "script de déploiement manuel": (
   5, 3.0)}   # (principal en jours, intérêt en jours perdus par mois)
def rendement(d): return d[1] / d[0]
budget, choisies, reste = 12, [], 12
for nom, d in sorted(DETTES.items(), key=lambda kv: -rendement(kv[1])):
    if d[0] <= reste: choisies.append(nom); reste -= d[0]
gain = sum(DETTES[n][1] for n in choisies)
print(f"à rembourser avec {budget} jours : {choisies} → intérêt économisé : {gain:
   .1f} jours par mois (retour en {sum(DETTES[n][0] for n in choisies) / gain:.1f}
    mois)")
assert "schéma de base sans index" in choisies and "documentation absente de l'API
   " not in choisies
```

Lecture :

Exemple pas à pas (suite)

1. À court terme (5 premiers sprints), ne rien rembourser livre plus. Mais sur **24 sprints**, consacrer ~20 % de la capacité au remboursement livre **plus de fonctionnalités au total** : la vitesse ne s'effondre pas.
2. La **loi de Little** est vérifiée dans les régimes stables : `WIP moyen ≈ débit × temps de cycle`. **Sans limite**, tout démarre en même temps, la capacité est dispersée, le débit s'effondre (0,31 par jour) et le temps de cycle explose (plus de 800 jours). **Limiter le WIP de 12 à 3** divise le temps de cycle par près de sept et fait même **monter** le débit : on **finit** avant de commencer.
3. La prévision **Monte Carlo** donne une distribution : « 50 % en N semaines, 85 % en M semaines ». L'engagement à 85 % est plus prudent que la division naïve par la moyenne.
4. Le **registre de dette** classé par rendement (intérêt économisé / principal) sélectionne d'abord les corrections peu coûteuses à fort intérêt (index de base, déploiement manuel) et laisse la documentation à plus tard.

À toi, avec aide

Voici les temps de cycle (en jours) de 20 éléments terminés :

```
[2, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 7, 8, 9, 10, 12, 14, 18, 21, 30]
```

1. Calcule la moyenne, la médiane, p85 et p95.
2. Quel est le niveau de service attendu (*Service Level Expectation*) : « 85 % des éléments sont terminés en N jours ou moins » ? Pourquoi est-ce plus utile qu'une moyenne ?
3. Une équipe a un débit de 5 éléments par semaine. Selon la loi de Little, quel est son WIP moyen si le temps de cycle moyen est celui calculé en 1 ?
4. Si la limite de WIP est ramenée à la moitié, que prévois-tu pour le temps de cycle (à débit constant) ?

À toi, avec aide · réponse

Indices progressifs

Indice 1

Utilise `statistics.mean`, `median` et un calcul de percentile.

Indice 2

WIP = débit × temps de cycle (convertis le débit par jour : 5/7 ou 5/5 jours ouvrés).

Indice 3

À débit constant, diviser le WIP par deux divise le temps de cycle par deux.

À toi, avec aide · réponse

Réponse

```
import math, statistics
tc = [2, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 7, 8, 9, 10, 12, 14, 18, 21, 30]
pct = lambda p: sorted(tc)[math.ceil(p / 100 * len(tc)) - 1]
moy, med = statistics.mean(tc), statistics.median(tc)
print(f"moyenne {moy:.1f} j, médiane {med} j, p85 {pct(85)} j, p95 {pct(95)} j")
debit = 5 / 5                                          # 5 éléments par semaine de
    5 jours ouvrés
wip = debit * moy
print(f"WIP moyen d'après Little : {wip:.1f} éléments ; à moitié → temps de cycle
   ≈ {moy / 2:.1f} j")
assert (round(moy, 1), med, pct(85), pct(95)) == (8.8, 6.0, 14, 21)
```

1. Moyenne **8,8 j** (tirée vers le haut par quelques longs éléments), médiane **6 j**, **p85 = 14 j**, **p95 = 21 j**.
2. SLE : « **85 % des éléments sont terminés en 14 jours ou moins** ». Cette phrase prévient l'attente d'un utilisateur ; la moyenne n'engage à rien, car la distribution est très asymétrique (queue longue).

À toi, avec aide · réponse

Réponse (suite)

3. WIP ≈ **8,8 éléments** (1 élément/jour × 8,8 jours).
4. À débit constant, un WIP de ~4 éléments ramènerait le temps de cycle autour de **4 à 5 jours**. En pratique le débit peut même augmenter (moins de changements de contexte).

À toi, en solo

Optimiseur de portefeuille de dette. Avec un registre de 15 éléments de dette (générés avec graine fixée : principal 1 à 12 jours, intérêt 0,2 à 6 jours par mois, zone du code, risque), et un budget de remboursement de 20 jours :

1. résous exactement le problème du **sac à dos** (programmation dynamique) pour maximiser l'intérêt économisé ;
2. compare avec la stratégie gloutonne par rendement et avec « ce qui énerve le plus l'équipe » (ordre aléatoire) ;
3. ajoute une contrainte de **dépendance** (rembourser A avant B) et de **fenêtre d'opportunité** (on remboursera plus facilement la dette d'un module quand on le modifie déjà : bonus de 30 % sur le rendement pour les modules touchés ce trimestre) ;
4. simule 12 mois : à chaque mois, intérêt payé (jours perdus) et capacité regagnée par les remboursements ; compare avec ne rien faire ;

À toi, en solo (suite)

5. calcule le moment du retour sur investissement (mois à partir duquel le cumul de jours économisés dépasse les jours investis).

À toi, en solo · réponse

Pistes de correction

```
import random

def registre(graine=1, n=15):
    rng = random.Random(graine)
    return {f"D{i}": dict(principal=rng.randint(1, 12), interet=round(rng.uniform(
   0.2, 6), 1), module=rng.choice("ABCDE")) for i in range(n)}
def sac_a_dos(dettes, budget):
    noms = list(dettes); dp = [(0.0, [])] * (budget + 1)
    dp = [(0.0, frozenset())] + [(0.0, frozenset())] * budget
    meilleur = [(0.0, frozenset()) for _ in range(budget + 1)]
    for nom in noms:
        p, v = dettes[nom]["principal"], dettes[nom]["interet"]
        for b in range(budget, p - 1, -1):
            val, ens = meilleur[b - p]
            if val + v > meilleur[b][0]: meilleur[b] = (val + v, ens | {nom})
    return max(meilleur, key=lambda x: x[0])
def glouton(dettes, budget):
    reste, choisies = budget, []
    for nom, d in sorted(dettes.items(), key=lambda kv: -kv[1]["interet"] / kv[1][
```

À toi, en solo · réponse

Pistes de correction (suite)

```
   "principal"]):
        if d["principal"] <= reste: choisies.append(nom); reste -= d["principal"]
    return sum(dettes[n]["interet"] for n in choisies), choisies
def hasard(dettes, budget, graine):
    rng, reste, total = random.Random(graine), budget, 0.0
    ordre = list(dettes); rng.shuffle(ordre)
    for nom in ordre:
        if dettes[nom]["principal"] <= reste: reste -= dettes[nom]["principal"]; t
   otal += dettes[nom]["interet"]
    return total

D = registre(); B = 20
opt, ens = sac_a_dos(D, B); glo, _ = glouton(D, B)
moy_hasard = sum(hasard(D, B, g) for g in range(300)) / 300
print(f"intérêt économisé par mois : optimal {opt:.1f} | glouton {glo:.1f} | ordre
    au hasard {moy_hasard:.1f} (moyenne de 300 tirages)")
assert opt >= glo - 1e-9 and glo > moy_hasard

# simulation sur 12 mois
```

À toi, en solo · réponse

Pistes de correction (suite)

```
total_interets = sum(d["interet"] for d in D.values())
investi = sum(D[n]["principal"] for n in ens); gain_mensuel = opt
cumul_rien = cumul_rembourse = 0.0; retour = None
for mois in range(1, 13):
    cumul_rien += total_interets
    cumul_rembourse += total_interets - (gain_mensuel if mois > 1 else 0) + (inves
   ti if mois == 1 else 0)         # on investit les jours au mois 1
    if retour is None and cumul_rembourse < cumul_rien: retour = mois
print(f"12 mois : jours perdus sans rembourser {cumul_rien:.0f} | en remboursant {
   cumul_rembourse:.0f} (investis {investi} j au départ) | retour sur investisseme
   nt au mois {retour}")
assert cumul_rembourse < cumul_rien and retour is not None and retour <= 6
```

Le **sac à dos exact** dépasse (ou égale) le glouton et surpasse nettement un choix au hasard. Le **retour sur investissement** arrive en quelques mois : c'est l'argument pour défendre un budget de remboursement auprès de la direction — en jours économisés, pas en principes.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Ne jamais rembourser la dette	La vitesse s'effondre progressivement	Budget régulier (15-25 %), registre, priorisation par rendement
Commencer beaucoup de choses à la fois	Temps de cycle qui explose, changements de contexte	Limites de travail en cours : finir avant de commencer
Estimer une date avec la moyenne	Dates tenues une fois sur deux	Distribution Monte Carlo, engagement à 85 %
Une « grande refonte » qui gèle le produit	Risque et valeur retardée	Remboursement incrémental, étrangleur

Mini-boss

Rapport de santé du delivery de HashBox. Avec un jeu de données simulé de 60 éléments terminés sur 14 semaines (dates de début et de fin, taille, type : fonctionnalité / bug / dette) et un registre de 12 dettes :

1. calcule débit hebdomadaire, temps de cycle (moyenne, médiane, p85, p95), WIP moyen, âge des éléments en cours ; vérifie Little ;
2. calcule la part du temps consacrée aux bugs de régression et à la dette (indicateur de santé) et son évolution ;
3. propose un SLE et une limite de WIP ; simule l'effet de la limite (kanban) ;
4. prévois par Monte Carlo la fin de 30 éléments restants (50/85/95 %) ;
5. simule 24 sprints avec 3 politiques de dette et recommande un budget ;
6. sélectionne les dettes à rembourser avec un budget de 20 jours (sac à dos) et calcule le retour sur investissement ;

Mini-boss

Mini-boss (suite)

7. rédige la synthèse : 3 constats chiffrés, 3 décisions, 3 risques.

Mini-boss · réponse

Pistes de correction

Un bon rapport : tableau des mesures de flux, vérification de Little, SLE chiffré (par exemple « 85 % en 9 jours »), limite de WIP justifiée par simulation, prévision en distribution, comparaison des politiques de dette (le budget de 15-25 % l'emporte à 6-12 mois), portefeuille optimal et retour sur investissement, et synthèse qui relie les chiffres à des décisions (par exemple « limiter le WIP à 6, réserver 20 % à la dette, engager la livraison de l'espace formateurs à la date P85 »).

Quiz éclair

Quiz éclair

1. La dette technique peut se comparer à une dette financière parce que :

A. elle n'a aucun coût
B. elle accélère aujourd'hui mais impose des intérêts (ralentissement) sur chaque modification future
C. elle se rembourse toute seule

2. Selon la loi de Little, à débit constant, réduire le travail en cours :

A. augmente le temps de cycle
B. n'a aucun effet
C. réduit le temps de cycle moyen

3. Une prévision de Monte Carlo à partir de l'historique de débit donne :

Quiz éclair

Quiz éclair (suite)

A. une distribution de dates avec niveaux de confiance
B. une date unique certaine
C. aucune information

4. 🔁 *Révision mission 08* : Le paradoxe de Simpson désigne :

A. une erreur de calcul de moyenne
B. une tendance présente dans chaque sous-groupe qui s'inverse dans l'ensemble
C. une panne de serveur

5. 🔁 *Révision mission 08* : Le NPS se calcule comme :

A. moyenne des notes
B. nombre de réponses

Quiz éclair

Quiz éclair (suite)

C. % de promoteurs (9-10) − % de détracteurs (0-6)

Quiz éclair · réponse

Réponses

1. **B**.
2. **C**.
3. **A**.
4. **B**.
5. **C**.

Je réfléchis

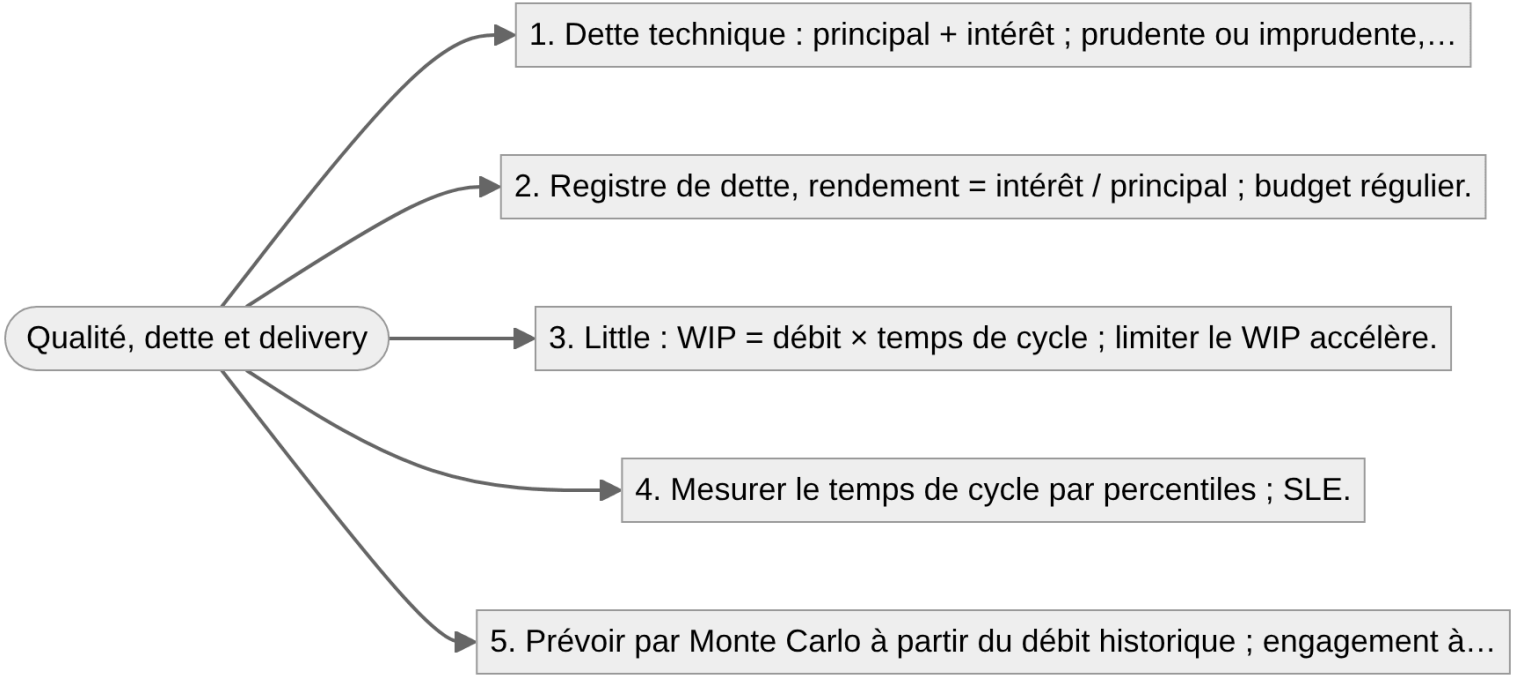
1. Quel module de ton produit « ralentit tout le monde » et peux-tu chiffrer ses intérêts en jours par mois ?
2. Combien d'éléments ton équipe a-t-elle en cours en ce moment, et combien finissent par semaine ?

À retenir

1. Dette technique : principal + intérêt ; prudente ou imprudente, délibérée ou non.
2. Registre de dette, rendement = intérêt / principal ; budget régulier.
3. Little : WIP = débit × temps de cycle ; limiter le WIP accélère.
4. Mesurer le temps de cycle par percentiles ; SLE.
5. Prévoir par Monte Carlo à partir du débit historique ; engagement à 85 %.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Gardien du rythme

Prochaine mission : Revue de décision produit

