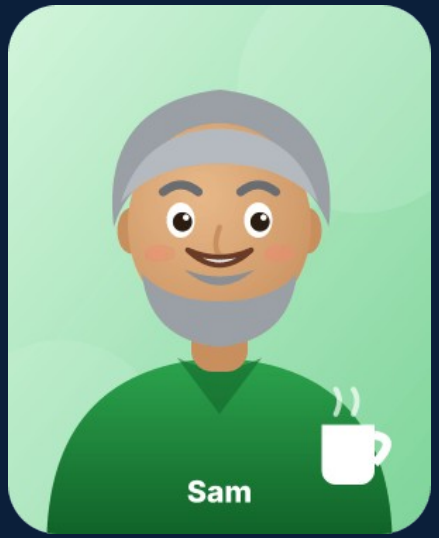
HEF-150 · MISSION 07 / 10

Priorisation et roadmap

Dire oui à tout, c'est ne rien faire bien. Prioriser, c'est choisir avec des critères explicites — et accepter de dire non, avec des raisons.





Badge : Stratège des choix

105 min · avancé

L'histoire

Lors de la réunion trimestrielle, vingt-trois demandes sont sur la table : des idées de l'équipe, des demandes de clients importants, un souhait personnel de la direction. L'équipe a la capacité d'en traiter six. Le responsable commercial insiste : « Le client Dupont-Formation veut l'export PDF, sinon il part. » Le directeur : « Mon idée est stratégique. » La discussion dure trois heures et se termine par : « On fera un peu de tout. »

> *« Un peu de tout, c'est rien en entier, dit Maya. Et la dernière personne à parler a gagné. Il faut des critères explicites, des estimations avec leur incertitude, une capacité réelle, et un moyen de vérifier que le classement ne dépend pas d'un chiffre inventé. »*



L'histoire

L'histoire (suite)

Tu apprends les méthodes de priorisation (RICE, ICE, WSJF, coût du retard), à tester la robustesse d'un classement face aux incertitudes des estimations, à ordonner des travaux pour minimiser le coût du retard, et à construire une feuille de route « maintenant / ensuite / plus tard » sous contraintes de capacité et de dépendances.

Ta mission

À la fin, je serai capable de…

1. calculer des scores RICE, ICE et WSJF et comparer les classements
2. évaluer la robustesse d'un classement quand les estimations sont incertaines (Monte Carlo)
3. appliquer la règle de Smith (coût du retard par durée) pour ordonner des travaux
4. construire une feuille de route sous capacité et dépendances
5. reconnaître les biais de priorisation (HiPPO, dernier arrivé, client bruyant)

Défi d'échauffement

Défi d'échauffement

Tu as 3 heures pour accomplir : (a) préparer une présentation importante pour demain matin (5 heures de travail), (b) répondre à 30 e-mails (1 heure), (c) réparer le robinet qui fuit (30 minutes, l'eau coule en permanence), (d) planifier tes vacances (2 heures, rien ne presse). Quel ordre et pourquoi ? Quelle information utilises-tu implicitement (urgence, importance, durée, coût de ne pas faire) ?

Le concept

Pourquoi prioriser explicitement

Sans critères	Avec critères
le plus bruyant gagne	le plus **utile** gagne
décisions incohérentes	cohérence entre trimestres
non défendables	argumentées, révisables

Aucune formule ne décide à ta place. Les scores servent à **structurer la discussion**, à rendre les hypothèses visibles, et à comparer des options hétérogènes.

RICE

Le concept

RICE (suite)

```
RICE = (Portée × Impact × Confiance) / Effort
```

Facteur	Définition	Échelle typique
Portée (*Reach*)	combien de personnes par période	nombre (ex. 2 000 membres / trimestre)
Impact	effet par personne	0,25 (minime), 0,5, 1, 2, 3 (massif)
Confiance	certitude des estimations	50 %, 80 %, 100 %
Effort	personnes-mois	nombre

La **confiance** pénalise les paris spéculatifs : un gain énorme mais incertain peut passer derrière un gain modeste certain.

Le concept

ICE

`ICE = Impact × Confiance × Facilité` (chacun sur 10) : plus rapide, plus grossier.

Valeur contre effort

Une grille 2×2 : **gains rapides** (valeur haute, effort bas), **projets majeurs** (haute/haut), **à éviter** (basse/haut), **bricoles** (basse/basse).

Le coût du retard et WSJF

Le **coût du retard** (*cost of delay*, CoD) : ce que coûte **chaque semaine** de retard d'un travail (revenu perdu, risque, client perdu, délai d'une opportunité). **WSJF** (*Weighted Shortest Job First*) :

Le concept

Le coût du retard et WSJF (suite)

```
WSJF = coût du retard / durée
```

Règle de Smith (ordonnancement) : pour minimiser le **coût total du retard** quand on traite des travaux l'un après l'autre, on les ordonne par **coût du retard par unité de durée décroissant**. Faire d'abord ce qui est court et coûte cher à attendre.

Le coût du retard se décompose : valeur utilisateur/métier, **criticité temporelle** (échéance fixe, fenêtre de marché), **réduction de risque / création d'opportunité**.

Kano

Les fonctionnalités se classent selon la satisfaction qu'elles procurent :

Type	Si présent	Si absent
Indispensable (*must-be*)	neutre	très insatisfait

Le concept

Kano (suite)

Type	Si présent	Si absent
Proportionnelle (*performance*)	plus = mieux	moins = pire
Attrayante (*delighter*)	ravi	neutre
Indifférente	rien	rien

Les attrayantes d'aujourd'hui deviennent les indispensables de demain.

Robustesse d'un classement

Les estimations sont **incertaines** (portée ±30 %, effort ±50 %). Pour savoir si un classement est fiable, on **simule** : on tire les estimations dans leurs fourchettes, on recalcule le classement, et on regarde la **fréquence** à laquelle chaque élément est dans le « top k » :

- élément toujours dans le top 3 : décision **solide** ;

Le concept

Robustesse d'un classement (suite)

- élément qui oscille : **investir dans l'information** (mission 3) avant de trancher.

La feuille de route

Style	Principe
Dates précises	engage sur des livraisons ; faux sentiment de certitude
Maintenant / Ensuite / Plus tard	engage sur l'**ordre** et les **résultats visés**, pas sur les dates ; réexaminé régulièrement
Par résultats	thèmes liés à des métriques (mission 1)

Une bonne feuille de route : **résultats** plutôt que sorties, **capacité réelle** (jamais 100 % : prévoir 20 à 30 % d'imprévu et de dette), **dépendances** explicites, **hypothèses** à tester, **ce que l'on ne fera pas**.

Le concept

Capacité et dépendances

Planifier = répartir des travaux (effort, dépendances, priorité) sur des périodes de **capacité fixe**. Un travail ne démarre que si ses dépendances sont terminées. L'ordonnancement par liste (prochain travail prioritaire dont les dépendances sont satisfaites) donne une bonne feuille de route de départ.

Les biais de priorisation

Biais	Remède
HiPPO (opinion du plus haut salaire)	critères publics, estimations par plusieurs personnes
Récence (le dernier arrivé gagne)	revue périodique de tout le backlog
Client bruyant	mesurer la demande réelle (combien de clients ?)
Biais de projet favori	notation à l'aveugle, avocat du diable

Le concept

Les biais de priorisation (suite)

Biais	Remède
Coût irrécupérable	décider sur l'avenir, pas sur ce qu'on a déjà dépensé
Illusion de précision	intervalles plutôt que valeurs uniques

🧠 ***Astuce*** *: pour chaque élément, écris la* ***phrase de refus*** *que tu donnerais s'il n'était pas retenu. Si elle est difficile à écrire, c'est qu'il a de bonnes chances d'avoir sa place.*

Exemple pas à pas

Problème : calculer RICE, ICE et WSJF pour 8 candidats, tester la robustesse du classement RICE par simulation, vérifier la règle de Smith par énumération, construire une feuille de route « maintenant / ensuite / plus tard » sous capacité et dépendances.

```
import itertools, random
from collections import Counter

CANDIDATS = {   # portée (membres/trimestre), impact (0,25-3), confiance, effort (
   personnes-mois), coût du retard (k€/mois)
    "export PDF":                  dict(portee=300,  impact=1.0, confiance=0.8, ef
   fort=1.0, cod=8),
    "rappels par e-mail":          dict(portee=3000, impact=0.5, confiance=0.8, ef
   fort=1.0, cod=14),
    "parcours « par où commencer »": dict(portee=2500, impact=1.0, confiance=0.5,
   effort=3.0, cod=20),
    "mode sombre":                 dict(portee=1500, impact=0.25, confiance=1.0, e
   ffort=0.5, cod=2),
    "classement des membres":      dict(portee=1000, impact=0.5, confiance=0.5, ef
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    fort=2.0, cod=3),
     "application mobile":          dict(portee=2000, impact=2.0, confiance=0.5, ef
    fort=8.0, cod=18),
     "recherche améliorée":         dict(portee=3500, impact=0.5, confiance=0.8, ef
    fort=2.0, cod=12),
     "tableau de bord formateurs":  dict(portee=150,  impact=2.0, confiance=0.8, ef
    fort=3.0, cod=9),
}
def rice(c): return c["portee"] * c["impact"] * c["confiance"] / c["effort"]
def wsjf(c): return c["cod"] / c["effort"]
print("candidat                          |  RICE  | WSJF")
classement_rice = sorted(CANDIDATS, key=lambda n: -rice(CANDIDATS[n]))
for n in classement_rice: print(f"{n:32s} | {rice(CANDIDATS[n]):6.0f} | {wsjf(CAND
    IDATS[n]):5.1f}")
assert classement_rice[0] == "rappels par e-mail" and classement_rice[-1] == "tabl
    eau de bord formateurs"

# ---------------- robustesse du classement RICE ----------------
def top_k_frequence(k=3, essais=4000, graine=1):
```

Exemple pas à pas (suite)

```
    rng, compteur = random.Random(graine), Counter()
    for _ in range(essais):
        scores = {}
        for n, c in CANDIDATS.items():
            portee = c["portee"] * rng.uniform(0.7, 1.3)
            impact = c["impact"] * rng.uniform(0.6, 1.4)
            confiance = min(1.0, c["confiance"] * rng.uniform(0.8, 1.2))
            effort = c["effort"] * rng.uniform(0.7, 1.8)                    # l
   'effort est sous-estimé plus souvent que sur-estimé
            scores[n] = portee * impact * confiance / effort
        for n in sorted(scores, key=scores.get, reverse=True)[:k]: compteur[n] +=
   1
    return {n: compteur[n] / essais for n in CANDIDATS}
freq = top_k_frequence()
print("présence dans le top 3 quand les estimations varient :")
for n, f in sorted(freq.items(), key=lambda kv: -kv[1]): print(f"  {f:5.0%}  {'#'
   * int(f * 30)} {n}")
assert freq["rappels par e-mail"] > 0.9 and 0.15 < freq["parcours « par où commenc
   er »"] < 0.6 and freq["tableau de bord formateurs"] < 0.05           # certains
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    choix sont solides, d'autres dépendent des estimations

# ---------------- règle de Smith : minimiser le coût total du retard ------------
   ----
TRAVAUX = {n: (c["effort"], c["cod"]) for n, c in CANDIDATS.items() if n in ("expo
   rt PDF", "rappels par e-mail", "mode sombre", "recherche améliorée", "classemen
   t des membres")}
def cout_total(ordre):
    t, total = 0.0, 0.0
    for n in ordre:
        duree, cod = TRAVAUX[n]; t += duree; total += cod * t                 # c
   haque travail attend jusqu'à sa fin : coût = coût du retard × temps d'achèvemen
   t
    return total
optimal = min(itertools.permutations(TRAVAUX), key=cout_total)
smith = tuple(sorted(TRAVAUX, key=lambda n: -TRAVAUX[n][1] / TRAVAUX[n][0]))
naif = tuple(sorted(TRAVAUX, key=lambda n: TRAVAUX[n][0]))                    # «
   le plus court d'abord »
print(f"coût du retard : optimal (énumération) {cout_total(optimal):.1f} | Smith {
```

Exemple pas à pas (suite)

```
   cout_total(smith):.1f} | le plus court d'abord {cout_total(naif):.1f}")
assert abs(cout_total(smith) - cout_total(optimal)) < 1e-9 and cout_total(naif) >=
    cout_total(smith)

# ---------------- feuille de route sous capacité et dépendances ----------------
ELEMENTS = {"A. rappels par e-mail": (1.0, []), "B. recherche améliorée": (2.0, []
   ), "C. parcours « par où commencer »": (3.0, ["A. rappels par e-mail"]),
            "D. tableau de bord formateurs": (3.0, []), "E. export PDF": (1.0, ["D
   . tableau de bord formateurs"]), "F. application mobile": (8.0, ["B. recherche
   améliorée", "C. parcours « par où commencer »"])}
PRIORITE = ["A. rappels par e-mail", "C. parcours « par où commencer »", "B. reche
   rche améliorée", "D. tableau de bord formateurs", "E. export PDF", "F. applicat
   ion mobile"]
def planifier(elements, priorite, capacite_par_trimestre=4.0, reserve=0.25):
    utilisable = capacite_par_trimestre * (1 - reserve)                          #
   25 % réservés à l'imprévu et à la dette
    reste = {n: e for n, (e, _) in elements.items()}; fait, trimestres = set(), []
    while len(fait) < len(elements):
        libre, courant = utilisable, []
```

Exemple pas à pas (suite)

```
        progres = True
        while progres:
            progres = False
            for n in priorite:
                if n in fait or n in courant: continue
                if all(d in fait for d in elements[n][1]) and reste[n] <= libre +
   1e-9:
                    courant.append(n); libre -= reste[n]; progres = True; break
        if not courant: raise RuntimeError("élément plus gros que la capacité d'un
    trimestre : à découper")
        fait |= set(courant); trimestres.append(courant)
    return trimestres
try:
    planifier(ELEMENTS, PRIORITE)
except RuntimeError as e:
    print("alerte :", e)
ELEMENTS["F. application mobile"] = (3.0, ["B. recherche améliorée", "C. parcours
   « par où commencer »"])           # on découpe en une version initiale
for i, t in enumerate(planifier(ELEMENTS, PRIORITE), 1): print(f"  {['Maintenant',
```

Exemple pas à pas (suite)

```
    'Ensuite', 'Plus tard', 'Plus tard (2)', 'Plus tard (3)'][min(i - 1, 4)]:14s}:
    {t}")
plan = planifier(ELEMENTS, PRIORITE)
assert plan[0][0] == "A. rappels par e-mail" and next(i for i, t in enumerate(plan
   ) if "F. application mobile" in t) > next(i for i, t in enumerate(plan) if "C.
   parcours « par où commencer »" in t)
```

Lecture :

1. **RICE** et **WSJF** ne classent pas pareil : RICE valorise la portée (le tableau de bord des formateurs, qui touche peu de monde, est dernier), WSJF le coût du retard par unité d'effort (l'export PDF, petit et urgent, remonte). Les différences sont des **sujets de discussion**, pas des erreurs.
2. La simulation de **robustesse** sépare les choix solides (« rappels par e-mail » est dans le top 3 dans 98 % des tirages ; le tableau de bord n'y est jamais) des choix dont la place dépend des estimations (le parcours « par où commencer » n'y est que dans environ 28 % des cas) : sur ces derniers, mieux vaut **acquérir de l'information** avant de trancher.

Exemple pas à pas (suite)

3. La **règle de Smith** donne exactement l'ordre optimal trouvé par énumération des permutations : on traite d'abord ce qui coûte cher à retarder par unité de travail. L'ordre « le plus court d'abord » est moins bon.
4. La **feuille de route** réserve 25 % de capacité, respecte les dépendances et signale qu'un élément plus gros qu'un trimestre (l'application mobile à 8 mois-personnes) doit être **découpé**.

À toi, avec aide

À toi, avec aide

Voici cinq candidats :

Candidat	Portée	Impact	Confiance	Effort	CoD (k€/mois)
A	800	2	80 %	4	10
B	4 000	0,5	100 %	2	6
C	200	3	50 %	1	12
D	1 500	1	80 %	3	5
E	600	1	100 %	0,5	3

1. Calcule RICE et WSJF, et classe selon chacun.
2. Quels éléments changent de place entre les deux classements ? Pourquoi ?
3. L'équipe a 4 personnes-mois : quel ensemble retiens-tu ? Argumente.

À toi, avec aide · réponse

Indices progressifs

Indice 1

RICE = portée × impact × confiance / effort ; WSJF = CoD / effort.

Indice 2

Regarde C : faible confiance mais coût du retard élevé et petit effort.

Indice 3

Un ensemble tient dans 4 personnes-mois en additionnant les efforts.

Réponse

```
C = {"A": (800, 2, 0.8, 4, 10), "B": (4000, 0.5, 1.0, 2, 6), "C": (200, 3, 0.5, 1,
    12), "D": (1500, 1, 0.8, 3, 5), "E": (600, 1, 1.0, 0.5, 3)}
rice = {n: p * i * c / e for n, (p, i, c, e, _) in C.items()}
wsjf = {n: cod / e for n, (_, _, _, e, cod) in C.items()}
print("RICE :", sorted(rice, key=rice.get, reverse=True), {n: round(v) for n, v in
    rice.items()})
print("WSJF :", sorted(wsjf, key=wsjf.get, reverse=True), {n: round(v, 1) for n, v
    in wsjf.items()})
assert sorted(rice, key=rice.get, reverse=True)[:2] == ["E", "B"] and sorted(wsjf,
    key=wsjf.get, reverse=True)[0] == "C"
```

RICE : E (1 200), B (1 000), D (400), A (320), C (300) → classement **E, B, D, A, C** ; WSJF : **C (12), E (6), B (3), A (2,5), D (1,7)**. **C** passe de dernier à premier : sa **portée** est petite (RICE faible) mais son **coût du retard** par unité d'effort est le plus élevé (une échéance ? un client important ? une fenêtre de marché ?). Avec 4 personnes-mois : **B (2) + E (0,5) + C (1) = 3,5** tient, avec une marge ; D et A restent pour plus tard. On vérifie la **confiance** de C (50 %) : avant de le construire, un test rapide de l'hypothèse (mission 3) est justifié.

À toi, en solo

Simulateur de comité de priorisation. Modélise un comité où des personnes biaisées donnent des notes :

1. 12 candidats avec valeur « réelle » cachée (tirée au hasard, graine fixée) ; 5 évaluateurs donnent des notes bruitées (bruit de 20 %) ;
2. un évaluateur « HiPPO » surpondère les idées du directeur (3 candidats gonflés de +40 %) ; un autre favorise ceux qui sont récents ;
3. compare trois méthodes d'agrégation : moyenne des notes, médiane, moyenne tronquée (on retire la note la plus haute et la plus basse) ; sans a priori sur le meilleur : mesure ;
4. mesure, sur 500 comités, la part des **vraies** quatre meilleures idées retenues par le top 4 choisi ;
5. montre l'effet d'une règle « notation à l'aveugle » (l'auteur n'est pas connu) et d'une règle « un évaluateur ne note pas ses propres idées » ;
6. conclus avec un tableau et une recommandation de processus.

À toi, en solo · réponse

Pistes de correction

```
import random, statistics

def comite(graine, aveugle=False, n_cand=12, n_eval=5):
    rng = random.Random(graine)
    reel = [rng.uniform(1, 10) for _ in range(n_cand)]
    du_directeur = set(rng.sample(range(n_cand), 3)); recents = set(rng.sample(ran
   ge(n_cand), 4))
    notes = [[] for _ in range(n_cand)]
    for e in range(n_eval):
        for c in range(n_cand):
            note = reel[c] * rng.uniform(0.65, 1.35)
         # bruit d'estimation
            if e == 0 and not aveugle and c in du_directeur: note *= 1.5
        # HiPPO : le directeur gonfle ses idées
            if e == 1 and c in recents: note *= 1.25
         # biais de récence
            notes[c].append(min(note, 14))
    return reel, notes
```

À toi, en solo · réponse

Pistes de correction (suite)

```
def agreger(notes, methode):
    if methode == "moyenne": return [statistics.mean(n) for n in notes]
    if methode == "médiane": return [statistics.median(n) for n in notes]
    return [statistics.mean(sorted(n)[1:-1]) for n in notes]
         # moyenne tronquée

def qualite(methode, aveugle, essais=500):
    total = 0
    for g in range(essais):
        reel, notes = comite(g, aveugle)
        agr = agreger(notes, methode)
        choisis = set(sorted(range(len(reel)), key=lambda c: -agr[c])[:4])
        vrais = set(sorted(range(len(reel)), key=lambda c: -reel[c])[:4])
        total += len(choisis & vrais) / 4
         # part des vraies meilleures idées retenues
    return total / essais

print("méthode            | avec biais connus | notation à l'aveugle")
res = {}
```

Pistes de correction (suite)

```
for m in ("moyenne", "médiane", "tronquée"):
    res[m] = (qualite(m, False), qualite(m, True)); print(f"{m:18s} |      {res[m]
   [0]:.3f}       |       {res[m][1]:.3f}")
assert all(res[m][1] > res[m][0] for m in res)
```

Résultat à lire honnêtement : avec un bruit **symétrique**, la **moyenne** reste la meilleure agrégation ; la médiane et la moyenne tronquée coûtent un peu de précision et ne protègent que contre un évaluateur aux notes **extrêmes**. Le vrai levier est le **processus** : la **notation à l'aveugle** (l'auteur n'est pas connu) améliore la part des vraies meilleures idées retenues pour **toutes** les méthodes, parce qu'elle supprime la surpondération des idées du directeur. Processus recommandé : estimations individuelles et indépendantes d'abord, notation à l'aveugle, agrégation simple, discussion centrée sur les **écarts** entre évaluateurs (c'est là que se trouve l'information), puis décision documentée.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Prendre un score pour une décision	Fausse précision, hypothèses cachées	Utiliser les scores pour structurer la discussion
Une seule valeur par estimation	Illusion de précision	Fourchettes et test de robustesse
Planifier à 100 % de capacité	Aucune marge pour l'imprévu et la dette	20 à 30 % de réserve
Accepter la demande du plus bruyant	Décision liée au statut ou au volume sonore	Critères publics, évaluation indépendante

Mini-boss

Mini-boss

Feuille de route du trimestre de HashBox. Pour 14 candidats (générés ou décrits par toi, avec portée, impact, confiance, effort, coût du retard, dépendances) :

1. calcule RICE, ICE et WSJF ; classe-les et explique les écarts ;
2. teste la robustesse du top 5 avec fourchettes d'incertitude (5000 tirages) ; identifie 3 éléments « solides » et 3 « fragiles » ;
3. pour les fragiles, propose l'**expérience la moins chère** qui lèverait l'incertitude (mission 3) ;
4. ordonne les 5 premiers par la règle de Smith et vérifie par énumération ;
5. construis une feuille de route « maintenant / ensuite / plus tard » sous capacité de 8 personnes-mois par trimestre avec 25 % de réserve, dépendances, découpage des éléments trop gros ;
6. rédige pour chaque élément **non retenu** la phrase de refus (raison, condition de réexamen) ;
7. simule un comité biaisé (HiPPO, récence) et montre comment ton processus protège la décision.

Mini-boss · réponse

Pistes de correction

Un bon travail : tableau des trois scores et des écarts expliqués; fréquences de présence dans le top 5; expériences proposées pour les fragiles (porte factice, entretien, prototype); ordre de Smith vérifié; feuille de route qui respecte la capacité et les dépendances; phrases de refus honnêtes et datées; simulation de comité montrant l'avantage de l'évaluation indépendante et de l'agrégation robuste.

Quiz éclair

Quiz éclair

1. Dans RICE, la confiance sert à :

A. pénaliser les estimations spéculatives
B. augmenter l'effort
C. mesurer la portée

2. La règle de Smith (WSJF) ordonne les travaux par :

A. durée croissante uniquement
B. coût du retard par unité de durée, décroissant
C. ordre alphabétique

3. Pour savoir si un classement est fiable face à l'incertitude des estimations, on :

A. augmente les poids

Quiz éclair

Quiz éclair (suite)

B. supprime les incertitudes

C. simule des variations des estimations et regarde la stabilité du classement

4. 🔁 *Révision mission 06* : Pour un produit en découverte avec une petite équipe, l'architecture recommandée est généralement :

A. un monolithe modulaire, avec possibilité d'extraire des services plus tard

B. quatorze microservices dès le premier jour

C. aucune structure

5. 🔁 *Révision mission 06* : Une décision « à sens unique » est une décision :

A. qu'on peut changer en cinq minutes

B. coûteuse à défaire, à prendre avec soin

Quiz éclair

Quiz éclair (suite)

C. prise par une seule personne

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Quelle demande a gagné ta dernière réunion de priorisation, et sur quel critère ?
2. Quel élément de ta roadmap n'a pas de phrase de refus claire pour son concurrent écarté ?

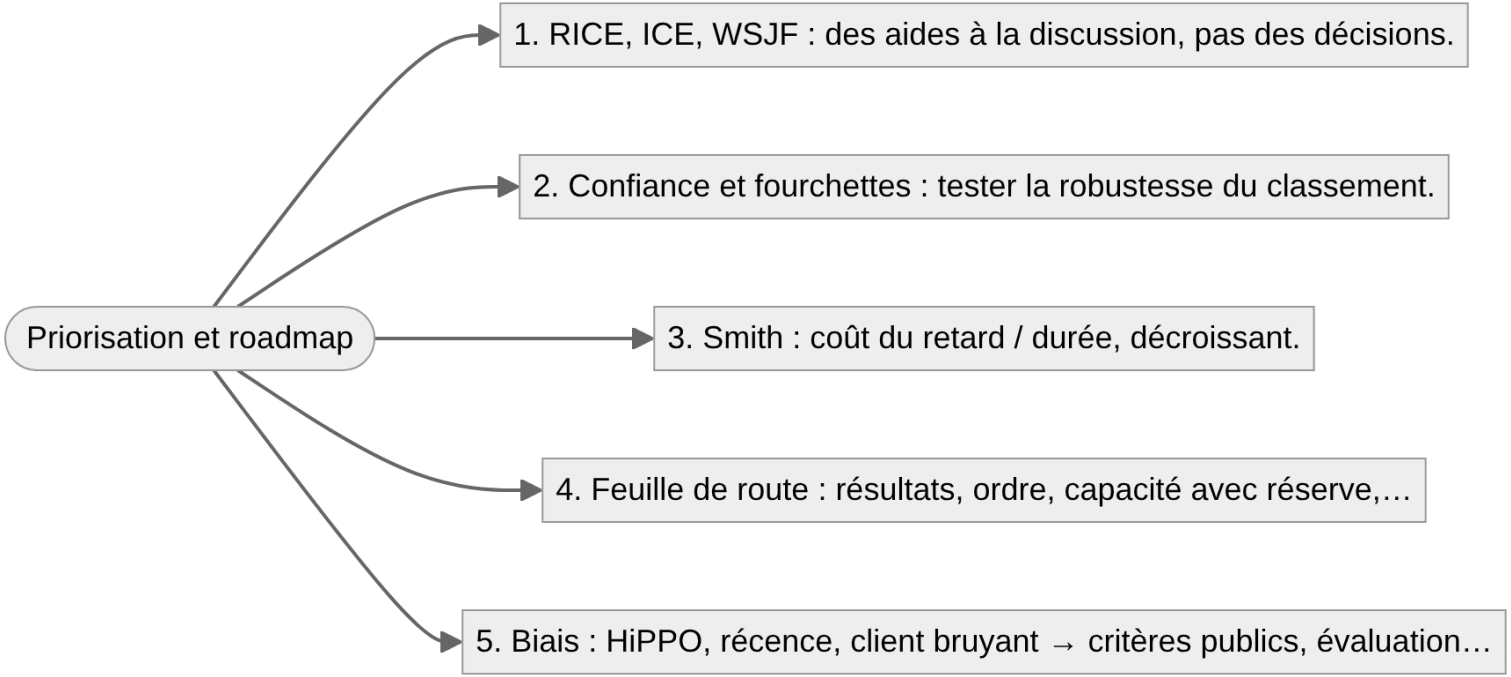
À retenir

À retenir

1. RICE, ICE, WSJF : des aides à la discussion, pas des décisions.
2. Confiance et fourchettes : tester la robustesse du classement.
3. Smith : coût du retard / durée, décroissant.
4. Feuille de route : résultats, ordre, capacité avec réserve, dépendances.
5. Biais : HiPPO, récence, client bruyant → critères publics, évaluation indépendante.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Stratège des choix

Prochaine mission : Analytics et feedback

