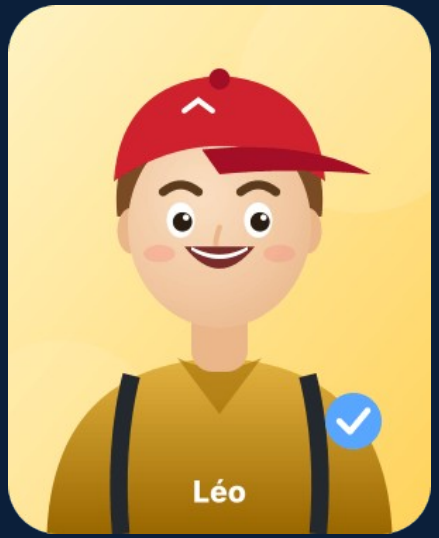
HEF-150 · MISSION 04 / 10

MVP et périmètre



Un MVP n'est pas une version bâclée du produit : c'est la plus petite expérience qui permet d'apprendre ce dont on a le plus besoin.

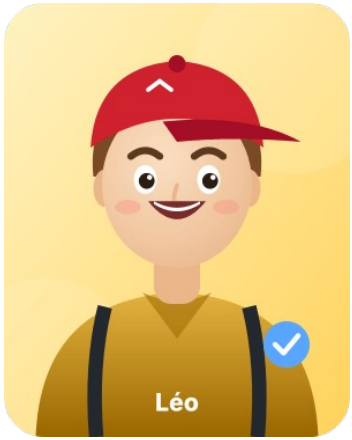


Badge : Sculpteur de périmètre

105 min · avancé

L'histoire

L'équipe de HashBox veut lancer un « espace formateurs » : création de cours, quiz, corrections automatiques, statistiques, messagerie, paiement, certificats. Estimation : huit mois. Sam demande : « Quelle est l'hypothèse la plus risquée ? » Réponse après discussion : *les formateurs indépendants accepteront-ils de publier leurs cours sur HashBox ?* « Pour le savoir, dit Sam, il n'est pas nécessaire de construire huit mois de fonctionnalités. Il suffit qu'un formateur puisse publier un cours, et que des membres le suivent. Le reste est du



« Un MVP se mesure à ce qu'il permet d'apprendre, pas à ce qu'il contient, ajoute Maya. Et "minimal" ne veut pas dire "mauvais" : ce que l'on livre doit marcher de bout en bout, même s'il est petit. »

L'histoire (suite)

Tu apprends à définir un MVP par l'hypothèse qu'il teste, à cartographier un parcours utilisateur en histoires, à découper une première version qui couvre le parcours de bout en bout, à choisir un périmètre sous budget, et à rendre les critères d'acceptation exécutables.

Ta mission

À la fin, je serai capable de…

1. définir un MVP par l'hypothèse qu'il teste et ses critères de succès
2. construire une cartographie d'histoires (story map) et en extraire une tranche mince de bout en bout
3. utiliser MoSCoW et le squelette ambulant (walking skeleton) pour cadrer un périmètre
4. choisir un périmètre sous contrainte de budget et de dépendances (optimisation)
5. écrire des critères d'acceptation exécutables (Étant donné / Quand / Alors)

Défi d'échauffement

Défi d'échauffement

Pour savoir si les gens veulent une voiture électrique, un constructeur doit-il construire une usine ? Compare avec : montrer un prototype, ouvrir une liste de précommandes, louer des modèles existants. Que faut-il avoir au minimum pour qu'on puisse dire que le test a *vraiment* mesuré l'envie d'acheter ? Une voiture sans roues, c'est un MVP ?

Le concept

Qu'est-ce qu'un MVP ?

MVP = *Minimum Viable Product* : la version d'un produit qui permet de **valider une hypothèse** avec le **moins d'effort**, auprès de **vrais utilisateurs**.

Idée reçue	Réalité
« La version 1 avec peu de fonctions »	Une **expérience** conçue pour apprendre
« Un produit de mauvaise qualité »	Il doit être **viable** : ce qu'il fait doit fonctionner
« Une seule fois »	Une suite de MVP successifs, chacun testant une hypothèse
« Toujours du code »	Peut être manuel, un prototype, une page, un service concierge

Le concept

Qu'est-ce qu'un MVP ? (suite)

Le skateboard, pas la roue : pour tester « les gens veulent se déplacer vite », on livre un **skateboard** (qui transporte déjà), pas une roue (inutile seule), puis une trottinette, une bicyclette… Chaque étape est **utilisable** de bout en bout.

Le MVP défini par l'hypothèse

```
Hypothèse la plus risquée → critère de succès → fonctionnalités MINIMALES nécessai
   res pour la tester
```

Tout le reste est **hors périmètre** pour l'instant. Le MVP est jugé sur la **réponse** qu'il apporte, pas sur son volume.

La cartographie d'histoires (story map)

On organise le travail selon le **parcours** de l'utilisateur :

Le concept

La cartographie d'histoires (story map) (suite)

```
Colonne du haut (colonne vertébrale) : les ACTIVITÉS, de gauche à droite dans l'or
   dre du parcours
Dessous : les ÉTAPES / histoires, par ordre de nécessité décroissante (haut = indi
   spensable)
Une « tranche » horizontale = une version qui couvre TOUT le parcours
```

Activité	Créer un cours	Publier	Être trouvé	Être payé
Tranche 1 (squelette)	formulaire minimal	publier	liste simple	virement manuel
Tranche 2	modèles de cours	planifier	recherche	paiement en ligne
Tranche 3	quiz, vidéos	versions	recommandations	statistiques de ventes

La **première tranche** est un **squelette ambulant** : la plus petite implémentation qui traverse **toute** l'architecture et **tout** le parcours (même avec des étapes manuelles).

Le concept

Prioriser avec MoSCoW

Catégorie	Sens
Must (doit)	sans cela, la version est inutile ou illégale
Should (devrait)	important, mais contournable à court terme
Could (pourrait)	souhaitable, si le temps le permet
Won't (pas cette fois)	explicitement exclu — pour l'instant

Règle de prudence : les **Must** ne devraient pas dépasser ~60 % de la capacité, pour garder de la marge face aux imprévus. La liste des **Won't** est aussi précieuse que celle des Must : elle évite le « petit ajout ».

Le concept

Couper sans casser

Techniques pour réduire un périmètre :

- **remplacer l'automatique par du manuel** (concierge) ;
- **réduire les variantes** (un seul moyen de paiement) ;
- **réduire l'ampleur** (un segment, une région) ;
- **reporter la qualité de finition** (pas la correction) ;
- **acheter ou réutiliser** plutôt que construire (mission 6).

Définition de terminé et critères d'acceptation

Chaque histoire a des **critères d'acceptation** précis, au format :

Le concept

Définition de terminé et critères d'acceptation (suite)

```
Étant donné [contexte]
Quand [action]
Alors [résultat observable]
```

Ces critères deviennent des **tests exécutables** (HEF-110) : l'équipe sait exactement quand une histoire est terminée.

Choisir un périmètre sous budget

Avec une capacité limitée et des **dépendances** entre histoires (on ne peut pas « payer » sans « publier »), choisir l'ensemble d'histoires qui maximise la **valeur** sous contrainte de **budget** est un problème d'optimisation (sac à dos avec dépendances). La valeur doit être définie **par rapport à l'hypothèse** à tester, pas par préférence.

Le concept

Après le MVP

Un MVP se termine par une **décision** (mission 10) : persévérer (élargir le périmètre), pivoter, arrêter. Planifier d'avance ce qu'on apprendra, quand, et comment on le saura.

🧠 ***Astuce*** *: pour chaque fonctionnalité candidate, demande : « si je la retire, puis-je encore tester l'hypothèse la plus risquée ? » Si oui, elle n'est pas dans le MVP.*

Exemple pas à pas

Problème : à partir d'une cartographie d'histoires de l'espace formateurs, extraire la première tranche de bout en bout, vérifier la règle MoSCoW sur la capacité, choisir un périmètre optimal sous budget avec dépendances, et exécuter des critères d'acceptation.

```
import itertools, unittest, io

# ---------------- 1. cartographie d'histoires ----------------
# activité (colonne vertébrale) → histoires (nom, valeur pour tester l'hypothèse «
    les formateurs publient », effort en jours, dépendances)
CARTE = {
    "Créer un cours": [("formulaire minimal", 9, 3, []), ("modèles de cours", 4, 5
   , ["formulaire minimal"]), ("quiz intégrés", 5, 8, ["formulaire minimal"])],
    "Publier":        [("publication simple", 9, 2, ["formulaire minimal"]), ("pla
   nification", 3, 3, ["publication simple"])],
    "Être trouvé":    [("liste des cours", 8, 2, ["publication simple"]), ("recher
   che", 6, 5, ["liste des cours"]), ("recommandations", 4, 13, ["recherche"])],
    "Être payé":      [("virement manuel", 7, 1, ["publication simple"]), ("paieme
   nt en ligne", 8, 13, ["publication simple"]), ("statistiques de ventes", 3, 8,
```

Exemple pas à pas

Exemple pas à pas (suite)

```
   ["paiement en ligne"])],
}
HISTOIRES = {nom: dict(activite=a, valeur=v, effort=e, deps=d) for a, lst in CARTE
   .items() for nom, v, e, d in lst}

def tranche_mince(carte):
    # la première tranche : pour CHAQUE activité, l'histoire la plus précieuse par
    jour d'effort (et ses dépendances)
    choisies = set()
    for activite, lst in carte.items():
        meilleure = max(lst, key=lambda h: h[1] / h[2])
        choisies.add(meilleure[0])
    def fermeture(noms):
        res, pile = set(), list(noms)
        while pile:
            n = pile.pop()
            if n in res: continue
            res.add(n); pile.extend(HISTOIRES[n]["deps"])
        return res
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    return fermeture(choisies)

premiere = tranche_mince(CARTE)
effort = sum(HISTOIRES[n]["effort"] for n in premiere)
activites_couvertes = {HISTOIRES[n]["activite"] for n in premiere}
print("première tranche :", sorted(premiere), f"→ {effort} jours")
assert activites_couvertes == set(CARTE) and effort <= 12                  # elle
   traverse TOUT le parcours, pour moins de 12 jours
assert "paiement en ligne" not in premiere and "virement manuel" in premiere
     # paiement manuel (concierge) au lieu d'automatique

# ---------------- 2. MoSCoW et capacité ----------------
CAPACITE = 20
MOSCOW = {"formulaire minimal": "Must", "publication simple": "Must", "liste des c
   ours": "Must", "virement manuel": "Must",
          "recherche": "Should", "modèles de cours": "Should", "quiz intégrés": "C
   ould", "paiement en ligne": "Should", "planification": "Could", "statistiques d
   e ventes": "Won't", "recommandations": "Won't"}
must = sum(HISTOIRES[n]["effort"] for n, c in MOSCOW.items() if c == "Must")
```

Exemple pas à pas (suite)

```
print(f"Must : {must} jours sur une capacité de {CAPACITE} = {must / CAPACITE:.0%}
    (prudence : ≤ 60 %)")
assert must / CAPACITE <= 0.60

# ---------------- 3. choisir le périmètre optimal sous budget, avec dépendances -
   ---------------
def meilleur_perimetre(budget):
    noms = list(HISTOIRES)
    meilleur = (0, set(), 0)
    for r in range(1, len(noms) + 1):
        for combo in itertools.combinations(noms, r):
            s = set(combo)
            if any(d not in s for n in s for d in HISTOIRES[n]["deps"]): continue
            # dépendance manquante : invalide
            e = sum(HISTOIRES[n]["effort"] for n in s)
            if e > budget: continue
            v = sum(HISTOIRES[n]["valeur"] for n in s)
            if v > meilleur[0]: meilleur = (v, s, e)
    return meilleur
```

Exemple pas à pas

Exemple pas à pas (suite)

```
for budget in (8, 12, 20):
    v, s, e = meilleur_perimetre(budget)
    print(f"budget {budget:2d} j : valeur {v:2d}, effort {e:2d} j → {sorted(s)}")
v8, s8, _ = meilleur_perimetre(8)
assert "formulaire minimal" in s8 and "publication simple" in s8
        # même avec 8 jours, le cœur du parcours est là
assert meilleur_perimetre(20)[0] > meilleur_perimetre(12)[0] > meilleur_perimetre(
   8)[0]

# ---------------- 4. critères d'acceptation EXÉCUTABLES ----------------
class Plateforme:
    def __init__(self): self.cours = {}
    def creer(self, formateur, titre):
        if not titre.strip(): raise ValueError("titre obligatoire")
        self.cours[titre] = {"formateur": formateur, "publie": False}
    def publier(self, titre):
        if titre not in self.cours: raise KeyError(titre)
        self.cours[titre]["publie"] = True
    def liste_publique(self): return sorted(t for t, c in self.cours.items() if c[
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    "publie"])

class CritèresAcceptation(unittest.TestCase):
    def test_un_formateur_publie_un_cours_visible_dans_la_liste(self):
        # Étant donné un formateur avec un cours créé / Quand il le publie / Alors
    le cours apparaît dans la liste publique
        p = Plateforme(); p.creer("ada", "Algorithmes"); self.assertEqual(p.liste_
   publique(), [])
        p.publier("Algorithmes"); self.assertEqual(p.liste_publique(), ["Algorithm
   es"])
    def test_un_cours_sans_titre_est_refuse(self):
        with self.assertRaises(ValueError): Plateforme().creer("ada", "  ")
    def test_publier_un_cours_inconnu_est_refuse(self):
        with self.assertRaises(KeyError): Plateforme().publier("Inconnu")
r = unittest.TextTestRunner(stream=io.StringIO()).run(unittest.defaultTestLoader.l
   oadTestsFromTestCase(CritèresAcceptation))
assert r.wasSuccessful()
print("critères d'acceptation exécutés :", r.testsRun, "verts")
```

Exemple pas à pas (suite)

Lecture :

1. La **tranche mince** choisit, pour chaque activité du parcours, l'histoire au meilleur rapport valeur/effort et ses dépendances : le parcours « créer → publier → être trouvé → être payé » est couvert de bout en bout pour **11 jours**, avec un **virement manuel** (concierge) au lieu du paiement en ligne (13 jours).
2. Les « Must » représentent 40 % de la capacité : de la marge reste pour l'imprévu.
3. L'optimisation sous budget montre que même avec **8 jours**, le cœur du parcours est inclus, et que la valeur croît (rendements décroissants) avec le budget.
4. Les **critères d'acceptation** sont des tests exécutables : l'équipe sait quand l'histoire est « terminée ».

À toi, avec aide

À toi, avec aide

Voici 8 fonctionnalités candidates pour un MVP de « révision par cartes mémoire », avec estimation d'effort (jours) et catégorie MoSCoW proposée par l'équipe :

Fonctionnalité	Effort	MoSCoW proposé
Créer une carte	2	Must
Réviser avec répétition espacée	5	Must
Importer depuis un fichier	4	Should
Statistiques détaillées	6	Should
Partage avec des amis	5	Could
Synchronisation multi-appareils	8	Could
Thèmes de couleurs	2	Could
Classement mondial	7	Won't

À toi, avec aide

À toi, avec aide (suite)

La capacité de l'équipe est de **15 jours**. Vérifie la règle des 60 %. Si elle n'est pas respectée, propose un découpage (réduction, remplacement manuel) pour l'atteindre. Quelle hypothèse ce MVP teste-t-il ?

À toi, avec aide · réponse

Indices progressifs

Indice 1

Somme des Must = 7 jours, soit 47 % de 15.

Indice 2

Pense à la répétition espacée : peut-on commencer par une version simplifiée ?

Indice 3

L'hypothèse la plus risquée est probablement « les gens révisent régulièrement avec cet outil ».

À toi, avec aide · réponse

Réponse

```
EFFORTS = {"Créer une carte": 2, "Réviser avec répétition espacée": 5, "Importer":
    4, "Statistiques": 6, "Partage": 5, "Synchronisation": 8, "Thèmes": 2, "Classe
   ment": 7}
MUST = ["Créer une carte", "Réviser avec répétition espacée"]
total_must = sum(EFFORTS[m] for m in MUST)
print(f"Must = {total_must} j sur 15 → {total_must / 15:.0%}")
assert total_must == 7 and total_must / 15 < 0.60
```

Les Must pèsent **7 jours, soit 47 %** de la capacité : la règle des 60 % est respectée, il reste de la marge pour l'imprévu. Si une marge supplémentaire était nécessaire, on pourrait simplifier la répétition espacée (planning fixe 1-3-7 jours au lieu d'un algorithme adaptatif) et reporter « Importer » (un fichier d'exemple fourni suffit).

Réponse (suite)

Hypothèse testée : *les personnes qui créent leurs cartes reviennent réviser régulièrement.* Mesure de succès possible : au moins 40 % des utilisateurs qui créent des cartes reviennent réviser dans les 7 jours. Le partage, les thèmes et le classement ne servent pas à tester cette hypothèse : ils restent hors périmètre (Could / Won't) tant qu'elle n'est pas validée.

À toi, en solo

Planificateur de périmètre avec programmation dynamique. Pour 30 histoires (générées avec graine fixée : valeur 1 à 10, effort 1 à 13, dépendances aléatoires formant un graphe acyclique) et un budget de 40 jours :

1. écris un solveur exact par **énumération des fermetures** pour de petites instances (≤ 14 histoires) et un solveur **glouton** (meilleur ratio valeur/effort, en ajoutant les dépendances manquantes) pour les grandes ;
2. compare les deux sur 200 petites instances aléatoires : écart de valeur moyen et maximal ;
3. sur l'instance de 30 histoires, calcule la valeur du glouton, puis tente une **amélioration locale** (échanges d'une histoire contre une autre) ;
4. ajoute la contrainte « la première tranche doit couvrir toutes les activités » et mesure son coût en valeur ;
5. montre graphiquement (texte) la courbe valeur contre budget de 5 à 60 jours.

À toi, en solo · réponse

Pistes de correction

```
import itertools, random

def instance(n, graine):
    rng = random.Random(graine); h = {}
    for i in range(n):
        deps = [f"h{j}" for j in range(i) if rng.random() < 0.15]
        h[f"h{i}"] = dict(v=rng.randint(1, 10), e=rng.randint(1, 13), deps=deps, a
   ct=i % 5)
    return h

def fermeture(noms, h):
    res, pile = set(), list(noms)
    while pile:
        n = pile.pop()
        if n in res: continue
        res.add(n); pile.extend(h[n]["deps"])
    return res

def valeur(s, h): return sum(h[n]["v"] for n in s)
```

À toi, en solo · réponse

Pistes de correction (suite)

```
def effort(s, h): return sum(h[n]["e"] for n in s)

def exact(h, budget):
    noms, meilleur = list(h), 0
    for r in range(len(noms) + 1):
        for combo in itertools.combinations(noms, r):
            s = set(combo)
            if any(d not in s for n in s for d in h[n]["deps"]) or effort(s, h) >
   budget: continue
            meilleur = max(meilleur, valeur(s, h))
    return meilleur

def glouton(h, budget):
    choisies = set()
    while True:
        meilleur, cand = 0, None
        for n in h:
            if n in choisies: continue
            ajout = fermeture([n], h) - choisies
```

À toi, en solo · réponse

Pistes de correction (suite)

```
            e = effort(ajout, h)
            if e == 0 or effort(choisies, h) + e > budget: continue
            ratio = valeur(ajout, h) / e
            if ratio > meilleur: meilleur, cand = ratio, ajout
        if not cand: break
        choisies |= cand
    return choisies

ecarts = []
for g in range(200):
    h = instance(12, g); b = 25
    e, gl = exact(h, b), valeur(glouton(h, b), h)
    ecarts.append((e - gl) / e if e else 0)
print(f"glouton contre exact sur 200 instances : écart moyen {sum(ecarts) / 200:.1
   %}, maximal {max(ecarts):.1%}")
assert sum(ecarts) / 200 < 0.12 and all(g >= 0 for g in ecarts)

h30 = instance(30, 7); base = glouton(h30, 40)
def ameliorer(s, h, budget):
```

Pistes de correction (suite)

```
    amelioration = True
    while amelioration:
        amelioration = False
        for sortant in list(s):
            for entrant in h:
                if entrant in s: continue
                t = (s - {sortant}) | fermeture([entrant], h)
                t = {n for n in t if all(d in t for d in h[n]["deps"])}
                if all(d in t for n in t for d in h[n]["deps"]) and effort(t, h) <
   = budget and valeur(t, h) > valeur(s, h):
                    s, amelioration = t, True; break
            if amelioration: break
    return s
meilleur = ameliorer(set(base), h30, 40)
print(f"30 histoires, 40 jours : glouton {valeur(base, h30)} → amélioration locale
    {valeur(meilleur, h30)}")
assert valeur(meilleur, h30) >= valeur(base, h30) and effort(meilleur, h30) <= 40
print("budget | valeur (glouton)")
for b in (5, 10, 20, 30, 40, 60):
```

À toi, en solo · réponse

Pistes de correction (suite)

```
    v = valeur(glouton(h30, b), h30); print(f"  {b:3d}  | {'#' * (v // 4)} {v}")
```

Le glouton est **proche** de l'optimum (quelques pourcents d'écart) et suffit pour un cadrage ; l'exact reste utile pour vérifier. La courbe montre les **rendements décroissants**, argument fort pour s'arrêter à un MVP plutôt que de remplir toute la capacité.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Confondre MVP et version bâclée	On teste la mauvaise chose avec un produit qui ne marche pas	Peu de fonctions, mais de bout en bout et correctes
Un MVP sans hypothèse	On ne sait pas ce qu'on apprend	Hypothèse la plus risquée + critère de succès
Couper une étape du parcours	L'utilisateur ne peut pas aller au bout	Tranche mince horizontale : tout le parcours
Pas de liste « Won't »	Le périmètre grossit par petits ajouts	Exclusions explicites et datées

Mini-boss

Mini-boss

Spécifier le MVP de HashBox « espace formateurs » avec critères exécutables.

1. écris l'hypothèse la plus risquée et le critère de succès chiffré ;
2. cartographie 4 activités et 14 histoires (valeur par rapport à l'hypothèse, effort, dépendances) ; extrais la tranche mince et le périmètre optimal pour un budget de 15 jours ;
3. classe en MoSCoW avec la règle des 60 % ; liste les « Won't » et la date de réexamen ;
4. implémente la tranche (une petite classe `Plateforme`) et écris au moins 8 critères d'acceptation Étant donné / Quand / Alors **exécutables** (dont 3 cas d'erreur) ;
5. propose pour chaque fonction non incluse un **contournement manuel** (concierge) ;
6. écris le plan de mesure : quelles données collecter, quand décider ;
7. écris les 3 raisons pour lesquelles ce MVP pourrait ne rien nous apprendre.

Mini-boss · réponse

Pistes de correction

Un bon travail : hypothèse chiffrée (par exemple « 10 formateurs indépendants publient chacun un cours payant en 6 semaines »), tranche mince couvrant création, publication, découverte, paiement manuel, listes MoSCoW cohérentes, classe `Plateforme` avec critères d'acceptation exécutés, concierge pour paiement et facturation, plan de mesure précis, et liste honnête des biais (formateurs recrutés dans notre réseau, enthousiasme du lancement, contournements manuels qui ne passent pas à l'échelle).

Quiz éclair

1. Un MVP est défini avant tout par :

A. l'hypothèse la plus risquée qu'il permet de tester
B. le nombre minimal de lignes de code
C. une version bon marché du produit final

2. La première tranche d'une cartographie d'histoires doit :

A. contenir toutes les fonctions importantes
B. couvrir tout le parcours de bout en bout, même avec des étapes manuelles
C. ne contenir que la première activité

3. La règle de prudence MoSCoW est que les Must représentent :

A. 100 % de la capacité

Quiz éclair

Quiz éclair (suite)

B. moins de 5 %
C. environ 60 % au plus de la capacité

4. 🔁 *Révision mission 03* : Une hypothèse falsifiable est une hypothèse :

A. que des observations peuvent réfuter, avec un critère fixé à l'avance
B. qu'on ne peut jamais tester
C. qui plaît à tout le monde

5. 🔁 *Révision mission 03* : S'arrêter dès que « p < 0,05 » pendant l'expérience :

A. n'a aucun effet
B. augmente fortement le taux de faux positifs
C. réduit les faux positifs

Quiz éclair · réponse

Réponses

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

Je réfléchis

Je réfléchis

1. Quelle fonctionnalité de ton produit actuel ne sert à tester aucune hypothèse ?
2. Quelle étape de ton parcours pourrais-tu faire à la main pour apprendre avant de l'automatiser ?

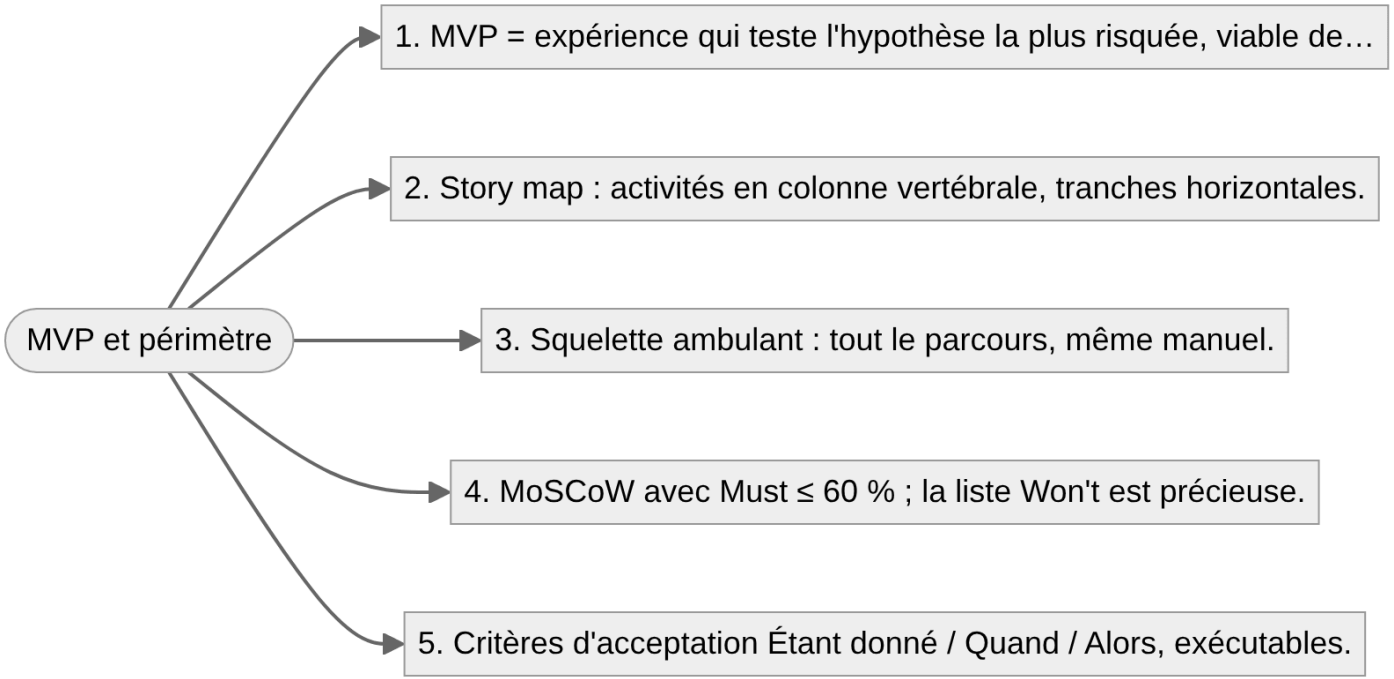
À retenir

À retenir

1. MVP = expérience qui teste l'hypothèse la plus risquée, viable de bout en bout.
2. Story map : activités en colonne vertébrale, tranches horizontales.
3. Squelette ambulant : tout le parcours, même manuel.
4. MoSCoW avec Must ≤ 60 % ; la liste Won't est précieuse.
5. Critères d'acceptation Étant donné / Quand / Alors, exécutables.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Sculpteur de périmètre

Prochaine mission : UX et parcours

