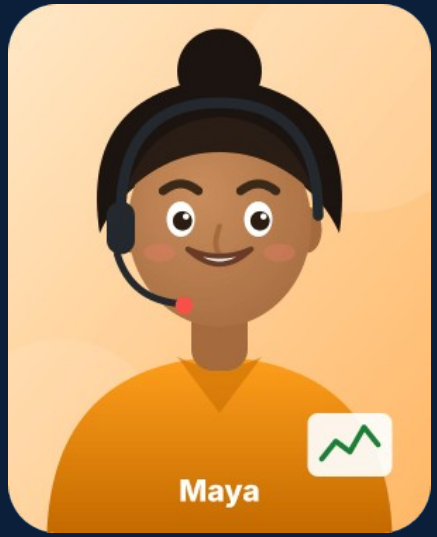
HEF-160 · MISSION 10 / 10

Workflow d'ingénierie assistée par l'IA

L'IA est un multiplicateur de capacité. Un multiplicateur de zéro donne zéro ; un multiplicateur sans contrôle multiplie aussi les erreurs. Il faut un flux de travail qui garde le jugement humain au centre.



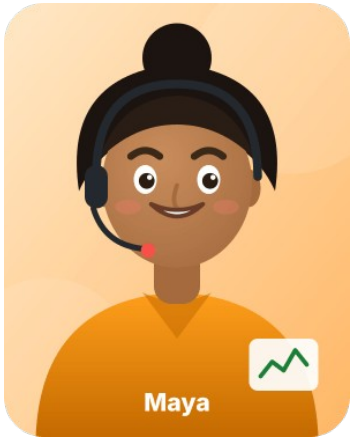


Badge : Ingénieur augmenté

140 min · avancé

L'histoire

Un an après les premiers essais, l'équipe de HashBox fait le bilan. Certaines tâches vont trois fois plus vite : tests de caractérisation, scripts de migration, documentation. D'autres sont plus lentes qu'avant : celles où l'on passait des heures à comprendre une sortie fausse mais convaincante. Un développeur junior, qui ne programme plus qu'avec l'assistant, n'arrive plus à déboguer sans lui. Une fonctionnalité intégrée sans vraie revue a provoqué un incident de facturation.



« L'IA n'est ni un miracle ni un danger en soi, conclut Maya. Elle amplifie ce que l'équipe fait déjà : une bonne pratique va plus vite, une mauvaise pratique va plus vite aussi. Il faut un workflow : où l'on l'emploie, quelles preuves on exige avant d'intégrer, ce qu'on trace, comment on garde et on développe ses compétences — et comment on mesure si ça paie vraiment. »

L'histoire

L'histoire (suite)

Dans cette dernière mission du cursus, tu assembles tout : spécification, génération, vérification indépendante, revue, traçabilité, mesure du retour réel, protection des compétences. Tu construis un assistant d'ingénierie avec périmètre, outils autorisés, évaluations, journal et dossier de validation — et tu fais le bilan de tout le parcours HEF.

Ta mission

À la fin, je serai capable de…

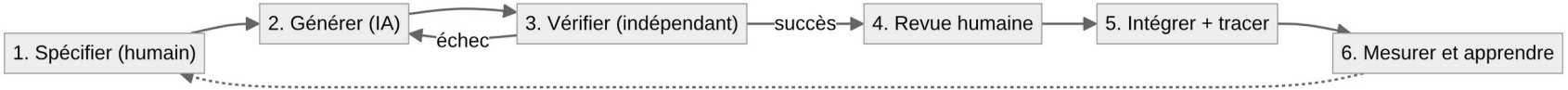
1. décrire un workflow assisté par IA : spécifier, générer, vérifier, revue, intégrer, apprendre
2. définir des portes de qualité qui empêchent l'intégration d'une sortie non vérifiée
3. tracer la provenance de chaque contribution assistée (modèle, prompt, vérifications, relecteur)
4. modéliser l'économie réelle : temps gagné, coût de vérification, défauts échappés
5. préserver et développer ses compétences face à l'assistance, et décider quand ne pas utiliser l'IA

Défi d'échauffement

Fais la liste de cinq tâches de ton travail. Pour chacune, note (1 à 5) : à quel point la production est coûteuse pour toi, à quel point vérifier le résultat est bon marché, et à quel point une erreur non détectée serait grave. Dans quel cas l'IA est-elle une excellente idée ? Dans quel cas est-elle risquée ? Et si la tâche est celle qui te fait progresser, que perds-tu en la déléguant ?

Le concept

Le workflow



Description du schéma : six étapes en boucle : l'humain spécifie ; l'IA génère ; une vérification indépendante accepte ou renvoie en génération ; une revue humaine valide ; la contribution est intégrée avec traçabilité ; on mesure et on apprend pour améliorer les spécifications suivantes.

Étape	Qui	Produit	Porte de qualité
Spécifier	humain	comportement attendu, cas limites, contraintes, critères d'acceptation, tests d'acceptation	spécification écrite avant la génération

Le concept

Le workflow (suite)

Étape	Qui	Produit	Porte de qualité
Générer	IA (avec outils autorisés)	proposition de code, de test, de documentation	périmètre respecté, contexte maîtrisé
Vérifier	outils + tests indépendants (mission 8)	rapport de vérification	échelle de vérification adaptée à l'enjeu
Revue	humain	décision argumentée	le relecteur peut **expliquer** chaque ligne
Intégrer	humain + pipeline	commit avec provenance	CI verte (HEF-120), revue de sécurité si besoin
Apprendre	équipe	mesures, rétrospective, mise à jour des prompts / des règles	indicateurs suivis

Le concept

Où l'IA excelle, où elle est risquée

Très adaptée	Prudence	À éviter sans expert
tests de caractérisation, scripts jetables, transformations de format, documentation, exploration d'options, explication de code existant, revue en complément	logique métier subtile, concurrence, sécurité, performance critique	décisions juridiques / réglementaires, conception de systèmes critiques, tout ce qu'on ne sait pas vérifier

Critère central : **vérifier coûte-t-il bien moins que produire ?** Et **que coûte une erreur non détectée ?**

Les portes de qualité

Aucune contribution assistée ne passe sans :

Le concept

Les portes de qualité (suite)

1. **spécification** écrite et acceptée ;
2. **tests indépendants** de la génération (mission 8) ;
3. **analyse statique / sécurité** (HEF-100) ;
4. **revue humaine** avec explication de la logique ;
5. **CI verte** (HEF-120) ;
6. **traçabilité** : fiche de provenance.

Une porte qui **échoue** renvoie à l'étape précédente ; elle n'est jamais « contournée pour aller plus vite ».

La traçabilité (provenance)

Le concept

La traçabilité (provenance) (suite)

Pour chaque contribution : **identifiant**, **auteur humain responsable**, **modèle et version**, **date**, **prompt (ou référence)**, **sources fournies**, **vérifications passées** (avec résultats), **relecteur**, **empreinte** du code livré. Utilité : enquêter après un incident, auditer, comprendre une dérive, répondre aux obligations (licences, conformité).

L'économie réelle

```
gain net par tâche = temps_sans_IA − (temps_prompt + temps_vérification + temps_co
   rrection) − coût_défauts_échappés
```

- Si l'IA est correcte avec probabilité `p` : le temps de vérification est **payé à chaque fois**, le temps de correction `(1 − p)` fois.
- **Les défauts échappés** (non détectés) coûtent cher plus tard : plus la vérification est légère, plus ils s'accumulent.

Le concept

L'économie réelle (suite)

- Le **taux d'acceptation** de la première proposition n'est pas le bon indicateur : mesurer le **temps de cycle de bout en bout**, les **défauts découverts après intégration**, le **coût total par tâche réussie**.

Le **point d'équilibre** : l'IA vaut la peine tant que p dépasse un seuil dépendant du rapport (coût de vérification + correction) / (coût de production).

Compétences, apprentissage, responsabilité

Risque	Parade
Atrophie : on ne sait plus faire	tâches importantes d'abord à la main ; explain-back ; débogage sans IA régulier
Sur-confiance : le texte convaincant endort la vigilance	préférer des preuves (tests) à la lecture rapide

Le concept

Compétences, apprentissage, responsabilité (suite)

Risque	Parade
Compétence de relecture perdue	pratiquer la relecture critique ; relire avec une grille
Apprentissage superficiel (débutants)	exiger l'explication ; utiliser l'IA comme tuteur (questions, indices) plutôt que générateur de réponses
Responsabilité diluée	un humain nommé répond de chaque changement

Un principe : **ne délègue pas ce que tu veux apprendre ; délègue ce que tu sais vérifier.**

Quand ne pas utiliser l'IA

- données réglementées ou secrets non masquables (mission 9) ;
- tâche dont tu ne peux pas vérifier le résultat ;
- tâche à forte exigence de compréhension profonde (pour apprendre) ;

Le concept

Quand ne pas utiliser l'IA (suite)

- quand le coût total (spécifier + vérifier) dépasse celui de faire soi-même.

Conventions d'équipe

Un document court : outils autorisés, données autorisées, portes de qualité, modèle de fiche de provenance, rôles de revue, formation, procédure d'incident, indicateurs suivis, revue trimestrielle.

Le cursus HEF en une phrase

Tout le parcours — penser, programmer, structurer, déployer, tester, sécuriser, mesurer, décider — converge ici : **utiliser des outils puissants sans déléguer le jugement**. Une sortie générée non vérifiée ne constitue pas une preuve de compétence.

Le concept

Le cursus HEF en une phrase (suite)

🧠 ***Astuce*** *: à la fin de chaque semaine, relis trois contributions assistées et demande-toi pour chacune : l'aurais-je écrite pareil ? ai-je compris ce que j'ai accepté ? qu'est-ce que j'ai appris ?*

Exemple pas à pas

Problème : simuler un workflow assisté par IA avec portes de qualité et provenance sur 200 tâches, mesurer l'économie réelle (temps gagné, défauts échappés) selon trois politiques de vérification, trouver le point d'équilibre en fonction de la fiabilité de l'IA, et générer les fiches de provenance.

```
import hashlib, json, random
from dataclasses import dataclass, field, asdict

# ---------------- 1. fiche de provenance ----------------
@dataclass
class Provenance:
    tache: str; auteur_humain: str; modele: str; version_modele: str; prompt_ref:
   str
    verifications: dict = field(default_factory=dict); relecteur: str = ""; emprei
   nte: str = ""; statut: str = "en cours"

def empreinte(code): return hashlib.sha256(code.encode()).hexdigest()[:12]

# ---------------- 2. le workflow avec portes ----------------
```

Exemple pas à pas (suite)

```
class Workflow:
    PORTES = ["specification", "tests_independants", "analyse_statique", "revue_hu
   maine", "ci"]
    def __init__(self): self.journal = []
    def integrer(self, prov, code, resultats):
        manque = [p for p in self.PORTES if not resultats.get(p)]
        prov.verifications = {p: bool(resultats.get(p)) for p in self.PORTES}
        if manque: prov.statut = f"REFUSÉ (portes non passées : {manque})"
        elif not prov.relecteur or prov.relecteur == prov.auteur_humain: prov.stat
   ut = "REFUSÉ (relecteur indépendant requis)"
        else: prov.statut = "INTÉGRÉ"; prov.empreinte = empreinte(code)
        self.journal.append(asdict(prov)); return prov.statut
w = Workflow()
ok = w.integrer(Provenance("T-1", "Maya", "modele-x", "2025-05", "prompts/migratio
   n.md", relecteur="Sam"), "def migrer(): ...", {p: True for p in Workflow.PORTES
   })
sans_tests = w.integrer(Provenance("T-2", "Kodi", "modele-x", "2025-05", "prompts/
   facture.md", relecteur="Sam"), "def facturer(): ...", {**{p: True for p in Work
   flow.PORTES}, "tests_independants": False})
```

Exemple pas à pas (suite)

```
auto_revue = w.integrer(Provenance("T-3", "Kodi", "modele-x", "2025-05", "prompts/
   x.md", relecteur="Kodi"), "def f(): ...", {p: True for p in Workflow.PORTES})
for s in (ok, sans_tests, auto_revue): print(" ", s)
assert ok == "INTÉGRÉ" and sans_tests.startswith("REFUSÉ (portes") and auto_revue.
   startswith("REFUSÉ (relecteur")

# ---------------- 3. l'économie : 200 tâches, trois politiques de vérification --
   --------------
def simuler(politique, fiabilite, n=200, graine=1):
    rng = random.Random(graine); temps = 0.0; echappes = 0; cout_defauts = 0.0
    PARAM = {   # (coût de vérification en minutes, probabilité de détecter un déf
   aut)
        "aucune": (0, 0.0), "légère (relecture rapide)": (6, 0.5), "complète (test
   s indépendants + revue)": (18, 0.95)}
    cout_verif, detection = PARAM[politique]
    for _ in range(n):
        production = rng.uniform(30, 90)                          # minutes pour f
   aire soi-même
        gain = production                                          # temps économi
```

Exemple pas à pas (suite)

```
    sé si l'IA est correcte
         temps_ia = 8 + cout_verif                                  # écrire le pro
    mpt + vérifier
         correct = rng.random() < fiabilite
         if correct:
             temps += production - temps_ia
         else:
             if rng.random() < detection:                           # défaut détect
    é : on corrige à la main (en plus du temps perdu)
                 temps += production - (temps_ia + 0.6 * production)
             else:                                                  # défaut échapp
    é : il coûte cher plus tard (investigation, correction, incident)
                 temps += production - temps_ia; echappes += 1; cout_defauts += rng
    .uniform(120, 480)
     return temps / 60, echappes, cout_defauts / 60            # heures gagnées, dé
    fauts échappés, heures perdues plus tard

print("fiabilité de l'IA | politique de vérification                | temps gagné
    | défauts échappés | coût différé | GAIN NET (heures)")
```

Exemple pas à pas (suite)

```
nets = {}
for fiab in (0.6, 0.8, 0.95):
    for pol in ("aucune", "légère (relecture rapide)", "complète (tests indépendan
   ts + revue)"):
        t, e, c = simuler(pol, fiab); nets[(fiab, pol)] = t - c
        print(f"      {fiab:.0%}        | {pol:41s} |  {t:7.1f} h  |       {e:3d}
          |  {c:6.1f} h  |  {t - c:7.1f}")
assert nets[(0.6, "complète (tests indépendants + revue)")] > nets[(0.6, "aucune")
   ]            # si l'IA est peu fiable, ne pas vérifier est désastreux
assert nets[(0.95, "aucune")] > nets[(0.95, "complète (tests indépendants + revue)
   ")]       # honnête : avec une IA très fiable, vérifier lourdement coûte plus q
   ue ça ne rapporte… ici

# ---------------- 4. point d'équilibre : à partir de quelle fiabilité l'IA vaut-e
   lle la peine ? ----------------
def gain_net(fiabilite, politique="complète (tests indépendants + revue)"):
    t, e, c = simuler(politique, fiabilite, n=400, graine=2); return t - c
seuil = next((p / 100 for p in range(5, 100, 5) if gain_net(p / 100) > 0), None)
print(f"avec une vérification complète, l'IA devient rentable pour cette tâche à p
```

Exemple pas à pas

Exemple pas à pas (suite)

```
   artir d'une fiabilité d'environ {seuil:.0%}")
assert seuil is not None and 0.2 <= seuil <= 0.7
print("sans vérification, le gain net à 80 % de fiabilité est de", round(gain_net(
   0.8, "aucune"), 1), "h ; avec vérification complète :", round(gain_net(0.8), 1)
   , "h")

# ---------------- 5. ne pas déléguer ce que l'on veut apprendre : un exemple de r
   ègle ----------------
def doit_utiliser_ia(tache):
    raisons = []
    if tache["donnees_reglementees"]: raisons.append("données réglementées : non")
    if not tache["verifiable"]: raisons.append("résultat non vérifiable à bas coût
    : non")
    if tache["objectif_apprentissage"]: raisons.append("tâche d'apprentissage : fa
   ire à la main d'abord, utiliser l'IA comme tuteur")
    if tache["cout_verif"] >= tache["cout_production"]: raisons.append("vérifier c
   oûte autant que produire : inutile")
    return (not raisons), raisons or ["adapté"]
for t in ({"nom": "tests de caractérisation", "donnees_reglementees": False, "veri
```

Exemple pas à pas

Exemple pas à pas (suite)

```
    fiable": True, "objectif_apprentissage": False, "cout_verif": 10, "cout_product
    ion": 60},
           {"nom": "analyse juridique", "donnees_reglementees": False, "verifiable"
    : False, "objectif_apprentissage": False, "cout_verif": 90, "cout_production":
    60},
           {"nom": "premier parcours de récursivité", "donnees_reglementees": False
    , "verifiable": True, "objectif_apprentissage": True, "cout_verif": 10, "cout_p
    roduction": 40}):
     print(f"  {t['nom']:34s} → {doit_utiliser_ia(t)}")
assert doit_utiliser_ia({"donnees_reglementees": False, "verifiable": True, "objec
    tif_apprentissage": False, "cout_verif": 10, "cout_production": 60})[0]
```

Lecture :

1. Le **workflow** refuse l'intégration d'une contribution dont une porte n'est pas passée (ici, absence de tests indépendants) ou dont le relecteur est aussi l'auteur humain. Chaque contribution acceptée reçoit sa **fiche de provenance** avec l'empreinte du code.

Exemple pas à pas (suite)

2. L'**économie réelle** montre que le gain dépend de la **fiabilité de l'IA** et de la **politique de vérification**. Avec une IA à 60 %, ne pas vérifier est désastreux : les défauts échappés coûtent plus que ce qu'on a gagné. Avec une IA à 95 %, dans cette simulation, la vérification lourde coûte plus qu'elle ne rapporte — mais c'est justement cette fiabilité qu'il faut **mesurer** avant de la supposer (mission 7).
3. Le **point d'équilibre** donne la fiabilité minimale à partir de laquelle l'IA vaut la peine pour *cette* tâche et *cette* vérification.
4. La **règle de décision** écrit explicitement quand ne pas utiliser l'IA : données réglementées, résultat non vérifiable, tâche d'apprentissage, vérification aussi coûteuse que la production.

À toi, avec aide

Une équipe mesure sur 50 tâches assistées :

- temps moyen sans IA : 60 minutes ;
- temps moyen pour rédiger le prompt : 6 minutes ;
- temps moyen de vérification : 14 minutes ;
- 80 % des propositions sont correctes ; pour les 20 % incorrectes, correction en 25 minutes de plus ;
- 4 défauts échappés sur 50 tâches, qui ont coûté chacun en moyenne 3 heures de correction plus tard.

1. Calcule le temps moyen par tâche avec l'IA (hors défauts échappés), puis le gain moyen par tâche.
2. Intègre le coût des défauts échappés : quel est le gain net moyen par tâche ?

À toi, avec aide (suite)

3. Si l'on passait à une vérification de 24 minutes qui ne laisse échapper que 1 défaut sur 50 : le gain net est-il meilleur ?
4. À partir de quelle durée de vérification (avec 4 défauts échappés) le gain devient-il nul ?

À toi, avec aide · réponse

Indices progressifs

Indice 1

Temps IA = prompt + vérification + 20 % × correction.

Indice 2

Coût des défauts par tâche = (4 × 180 minutes) / 50.

Indice 3

Avec la vérification plus longue : 24 minutes et 1 défaut.

Réponse

```
sans_ia = 60
def temps_ia(verif): return 6 + verif + 0.20 * 25
def gain_net(verif, defauts): return sans_ia - temps_ia(verif) - defauts * 180 / 5
   0
print(f"vérification 14 min : temps {temps_ia(14):.1f} min, gain brut {sans_ia - t
   emps_ia(14):.1f}, gain net {gain_net(14, 4):.1f} min")
print(f"vérification 24 min : temps {temps_ia(24):.1f} min, gain net {gain_net(24,
    1):.1f} min")
verif_nulle = next(v for v in range(1, 120) if gain_net(v, 4) <= 0)
print("gain nul à partir d'une vérification de", verif_nulle, "minutes")
assert round(temps_ia(14), 1) == 25.0 and round(gain_net(14, 4), 1) == 20.6 and ga
   in_net(24, 1) > gain_net(14, 4)
```

1. Temps moyen avec l'IA : 6 + 14 + 0,2 × 25 = **25 min** ; gain brut **35 min** par tâche (58 %).
2. Les défauts échappés coûtent 4 × 180 / 50 = **14,4 min** par tâche : gain net **20,6 min**.
3. Vérification de 24 min avec 1 seul défaut : temps 35 min, coût des défauts 3,6 min, gain net **21,4 min** — **légèrement meilleur** malgré une vérification plus longue.

Réponse (suite)

4. À 4 défauts échappés, le gain devient nul à partir de **35 minutes** de vérification. Moralité : la vérification n'est pas un coût à minimiser, c'est un **investissement à optimiser**, et on ne peut le faire qu'en **mesurant** défauts échappés et temps.

À toi, en solo

Simulateur d'organisation assistée par IA. Simule une équipe de 6 personnes sur 12 mois (1 pas = 1 semaine, 10 tâches par personne et par semaine tirées au hasard avec graine fixée) :

1. chaque tâche a un coût de production, un coût de vérification, une gravité d'erreur ; l'IA a une fiabilité qui dépend du **type de tâche** (par exemple 95 % documentation, 85 % tests, 65 % logique métier, 55 % performance) ;
2. politique A : « toujours utiliser l'IA, vérifier légèrement » ; politique B : « utiliser l'IA selon la règle de décision de la mission (vérifiable, coût de vérif < production, pas d'apprentissage), vérifier selon la gravité » ; politique C : « jamais d'IA » ;
3. chaque personne a un niveau de **compétence** qui progresse avec la pratique manuelle et **s'érode** avec l'usage excessif sans pratique (paramètres à définir) ; une compétence basse réduit la capacité à détecter les erreurs ;
4. mesure sur 12 mois : tâches terminées, heures économisées, défauts échappés et leur coût, compétence moyenne finale, **coût total de possession** ;

À toi, en solo (suite)

5. cherche le paramétrage de la politique B qui maximise le gain net sans que la compétence moyenne passe sous 70 % du niveau initial ;
6. synthèse chiffrée et recommandation d'équipe (règles, formation, indicateurs).

À toi, en solo · réponse

Pistes de correction

```
import random
FIABILITE = {"documentation": 0.95, "tests": 0.85, "metier": 0.65, "performance":
   0.55}
class Personne:
    def __init__(self): self.competence = 1.0
    def detection(self): return min(0.97, 0.45 + 0.5 * self.competence)
def tache(rng):
    t = rng.choice(list(FIABILITE)); prod = rng.uniform(30, 120)
    return dict(type=t, prod=prod, verif=prod * (0.25 if t in ("documentation", "t
   ests") else 0.6), gravite=rng.choice([1, 1, 2, 5]), apprentissage=rng.random()
   < 0.1)
def utiliser(politique, t):
    if politique == "A": return True
    if politique == "C": return False
    return t["verif"] < t["prod"] * 0.5 and not t["apprentissage"] and FIABILITE[t
   ["type"]] >= 0.8
def simuler(politique, graine=1, semaines=48, personnes=6, effort_verif=1.0):
    rng = random.Random(graine); equipe = [Personne() for _ in range(personnes)]
    heures = defauts = 0; cout_def = 0.0; faites = 0
```

À toi, en solo · réponse

Pistes de correction (suite)

```
    for _ in range(semaines):
        for p in equipe:
            utilisations = 0
            for _ in range(10):
                t = tache(rng); faites += 1
                if utiliser(politique, t):
                    utilisations += 1
                    verif = t["verif"] * (0.5 if politique == "A" else effort_veri
f * (0.7 + 0.1 * t["gravite"]))
                    det = p.detection() * (0.55 if politique == "A" else 1.0)
                    if rng.random() < FIABILITE[t["type"]]: heures += (t["prod"] -
 8 - verif) / 60
                    elif rng.random() < det: heures += (t["prod"] - 8 - verif - 0.
6 * t["prod"]) / 60
                    else: heures += (t["prod"] - 8 - verif) / 60; defauts += 1; co
ut_def += t["gravite"] * rng.uniform(1.0, 4.0)
            # compétence : progresse avec la pratique manuelle, s'érode avec l'usa
ge massif
            manuel = 10 - utilisations
```

À toi, en solo · réponse

Pistes de correction (suite)

```
            p.competence = max(0.2, min(1.0, p.competence + 0.0015 * manuel - 0.00
   2 * max(0, utilisations - 4)))
    comp = sum(p.competence for p in equipe) / len(equipe)
    return dict(faites=faites, heures=heures, defauts=defauts, cout_defauts=cout_d
   ef, net=heures - cout_def, competence=comp)
for pol in "ABC":
    r = simuler(pol); print(f"politique {pol} : heures économisées {r['heures']:7.
   0f} | défauts échappés {r['defauts']:4d} (coût {r['cout_defauts']:6.0f} h) | GA
   IN NET {r['net']:7.0f} h | compétence finale {r['competence']:.2f}")
res = {p: simuler(p) for p in "ABC"}
assert res["B"]["net"] > res["A"]["net"] and res["A"]["competence"] < 0.7 <= res["
   B"]["competence"]
meilleur = max(((e, simuler("B", effort_verif=e)) for e in (0.6, 0.8, 1.0, 1.3, 1.
   6)), key=lambda x: x[1]["net"] if x[1]["competence"] >= 0.7 else -1e9)
print(f"effort de vérification optimal pour B : x{meilleur[0]} → gain net {meilleu
   r[1]['net']:.0f} h, compétence {meilleur[1]['competence']:.2f}")
assert meilleur[1]["competence"] >= 0.7
```

Pistes de correction (suite)

Le simulateur met des **chiffres sur un débat** qui reste souvent idéologique : la politique « toujours utiliser » affiche beaucoup d'heures économisées, mais perd une partie en défauts échappés et **érode la compétence** de l'équipe, donc sa capacité à détecter les erreurs ; la politique « choisir selon la règle » donne un meilleur gain net et préserve la compétence. Les paramètres sont des **hypothèses** — la vraie leçon est de les **mesurer** dans ton équipe.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Intégrer parce que « ça a l'air bon »	Défauts échappés, incidents	Portes de qualité : spécification, tests indépendants, revue, CI
Mesurer seulement le temps gagné à la production	On ignore vérification et défauts échappés	Gain net de bout en bout
Déléguer ce qu'on veut apprendre	Atrophie des compétences	Faire d'abord à la main, IA comme tuteur
Pas de traçabilité	Impossible d'enquêter ou d'auditer	Fiche de provenance par contribution

Mini-boss

Mini-boss

Projet final HEF : l'assistant d'ingénierie de HashBox, avec dossier de validation. Conçois et implémente un assistant d'ingénierie (modèle simulé ou scripté) qui aide à corriger des défauts dans un petit projet, en réunissant les acquis du module et du cursus :

1. **périmètre explicite** (tâches permises / interdites, données autorisées par classification) et **outils autorisés** (schémas, rôles, plafonds, approbation) — missions 5 et 9 ;
2. **harnais d'agent** : budgets, stagnation, lecture seule des mesures, approbation, instantanés, retour arrière, trace — mission 6 ;
3. **contexte** : budget, faits épinglés, récupération documentaire avec citations et refus — missions 3 et 4 ;
4. **prompts** versionnés, schémas de sortie, banc de régression — mission 2 ;
5. **vérification indépendante** de chaque proposition : analyse statique, tests d'exemples, propriétés, différentiel, mutation — mission 8 ;

Mini-boss (suite)

6. **évaluation** : jeu de 30 tâches, exécutions multiples, intervalles, comparaison appariée de deux versions, contamination — mission 7 ;
7. **sécurité** : masquage, souillure, assainissement, banc d'attaques avec règle d'acceptation — mission 9 ;
8. **workflow** : portes de qualité, fiche de provenance, journal chaîné — mission 10 ;
9. **économie** : gain net mesuré et point d'équilibre ;
10. **dossier de validation** (5 pages) : ce qui est démontré, ce qui ne l'est pas, risques résiduels, plan de surveillance, règles d'usage, responsable ;
11. **bilan personnel** (20 lignes) : les trois idées du cursus HEF qui ont le plus changé ta façon de travailler, une habitude à garder, une à abandonner, et la prochaine spécialisation HashCode que tu veux explorer.

Mini-boss · réponse

Pistes de correction

Un excellent projet : l'assistant est borné par un périmètre et des outils explicites, chaque proposition passe par une chaîne de vérification indépendante avant revue humaine, les évaluations ont des intervalles et une règle de décision écrite avant, le banc d'attaques ne montre aucune réussite pour la configuration finale, chaque contribution a une fiche de provenance et un journal chaîné, l'économie est chiffrée, le dossier de validation est honnête sur ce qui n'est pas prouvé, et le bilan personnel relie les idées du cursus (penser en preuves, mesurer avant d'optimiser, concevoir pour la panne, sécuriser par construction, décider sur l'avenir) à une pratique concrète.

Quiz éclair

Quiz éclair

1. Une contribution assistée par IA peut être intégrée seulement si :

A. elle a passé des vérifications indépendantes et une revue humaine, avec traçabilité
B. l'IA affirme qu'elle est correcte
C. elle est rapide à produire

2. Le gain net réel d'un assistant se calcule en tenant compte :

A. uniquement du temps de génération
B. du temps de prompt, de vérification, de correction et du coût des défauts échappés
C. du nombre de lignes produites

3. Pour préserver ses compétences face à l'assistance, on :

A. délègue tout par principe

Quiz éclair

Quiz éclair (suite)

B. évite de relire le code

C. fait d'abord à la main ce qu'on veut apprendre et sait expliquer ce qu'on accepte

4. 🔁 *Révision mission 09* : L'injection de prompt indirecte est une attaque où :

A. des instructions cachées dans une donnée lue par le système cherchent à le détourner

B. l'utilisateur écrit lui-même une consigne hostile

C. le serveur est saturé de requêtes

5. 🔁 *Révision mission 09* : La défense la plus solide contre l'exfiltration par un agent est de :

A. écrire un prompt plus long

B. casser au moins un côté de la triade (données privées, contenu non fiable, canal externe)

C. augmenter la température

Quiz éclair · réponse

Réponses

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

Je réfléchis

Je réfléchis

1. Quelle tâche de ton travail devrais-tu continuer à faire à la main pour rester compétent·e ?
2. Quelle règle d'équipe écrirais-tu demain pour que « ça passe les tests de l'IA » ne soit jamais un critère d'acceptation suffisant ?

À retenir

À retenir

1. Spécifier, générer, vérifier (indépendamment), revue humaine, intégrer avec traçabilité, apprendre.
2. Portes de qualité non négociables ; fiche de provenance par contribution.
3. Gain net = temps gagné − vérification − correction − défauts échappés.
4. Ne pas déléguer ce qu'on veut apprendre ; déléguer ce qu'on sait vérifier.
5. L'IA multiplie les capacités, jamais la responsabilité : celui qui intègre répond du résultat.

Félicitations

Félicitations

Tu as terminé **HEF-160 — IA et augmentation humaine**, et avec lui le **tronc commun HashCode Engineering Foundations**. Tu sais comprendre les limites d'un modèle, structurer prompts et contexte, construire un RAG qui cite ou refuse, borner des outils et des agents, évaluer avec rigueur, vérifier de façon indépendante le code généré, défendre un système contre l'injection, et intégrer l'IA dans un workflow où le jugement humain reste au centre.
Du premier raisonnement logique à l'assistant gouverné, le fil est le même : **penser en preuves, mesurer avant d'optimiser, concevoir pour la panne, sécuriser par construction, décider honnêtement**.

→ Et maintenant : choisis une **spécialisation HashCode** et applique ces fondations à un vrai projet. 🚀

MISSION ACCOMPLIE

+120 XP

Badge débloqué : Ingénieur augmenté

Fin du module HEF-160. Bravo !

