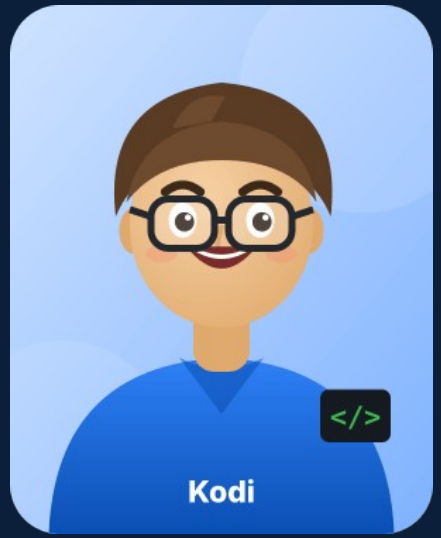
HEF-150 · MISSION 05 / 10

UX et parcours

Un produit utile mais pénible à utiliser est un produit abandonné. L'expérience se conçoit, se mesure et se corrige avec méthode — pas avec du goût seulement.





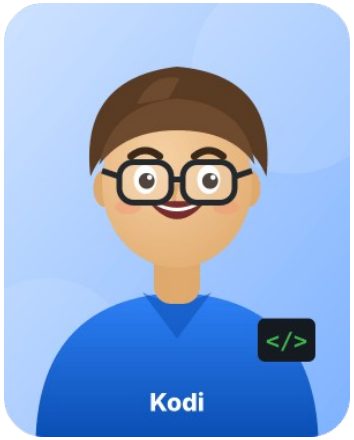
Badge : Ami de l'utilisateur

105 min · avancé

L'histoire

Le formulaire d'inscription de HashBox compte 14 champs. 62 % des visiteurs qui l'ouvrent l'abandonnent. L'équipe décide de le « moderniser » avec des couleurs plus tendance. Le texte gris clair sur fond blanc plaît beaucoup au designer. Une utilisatrice malvoyante écrit au support : « Je ne vois pas les étiquettes des champs. » Une autre : « J'ai tapé mon e-mail, ça m'a répondu "format invalide" sans me dire lequel était attendu. »

« L'expérience utilisateur n'est pas une question de goût, dit Maya. Il y a des principes, des mesures et des erreurs classiques. On peut calculer un contraste, mesurer la charge d'un choix, compter où les gens abandonnent, et demander à cinq personnes d'essayer. »



L'histoire (suite)

Tu apprends à cartographier un parcours et mesurer ses abandons (entonnoir), à appliquer les heuristiques d'utilisabilité, à calculer le contraste de couleurs (accessibilité), à modéliser le temps de décision (Hick) et de pointage (Fitts), à mener et analyser un test d'utilisabilité (SUS, taux de réussite), et à détecter des défauts d'accessibilité dans du HTML.

Ta mission

À la fin, je serai capable de…

1. décrire un parcours utilisateur et mesurer ses abandons étape par étape
2. appliquer les heuristiques de Nielsen à un écran
3. calculer un rapport de contraste WCAG et vérifier une palette
4. utiliser les lois de Hick et de Fitts pour raisonner sur le choix et le pointage
5. analyser un test d'utilisabilité : taux de réussite avec intervalle, score SUS

Défi d'échauffement

Défi d'échauffement

Tu entres dans un bâtiment inconnu pour trouver un guichet. Les panneaux sont écrits en gris clair sur gris moyen, les portes ne montrent pas si l'on pousse ou si l'on tire, et le guichet attend que tu connaisses ton numéro de dossier sans te l'avoir dit. Liste ce qui te ferait abandonner, dans l'ordre. Lesquels de ces défauts pourrais-tu mesurer avec des chiffres ?

Le concept

Le parcours utilisateur

Un **parcours** décrit les étapes qu'une personne traverse pour accomplir une tâche : **déclencheur → étapes → résultat**. Pour chaque étape : ce que la personne **fait**, **pense**, **ressent**, les **points de friction**.
Entonnoir (*funnel*) : proportion de personnes qui atteignent chaque étape. Le **taux d'abandon** entre deux étapes désigne où investir.

```
Visite → Page d'inscription → Formulaire commencé → Formulaire envoyé → E-mail con
   firmé → Premier cours
```

Le plus grand abandon relatif est souvent la meilleure opportunité (mission 1).

Le concept

Les dix heuristiques d'utilisabilité de Nielsen

#	Heuristique	Exemple
1	**Visibilité de l'état du système**	barre de progression, confirmation
2	**Correspondance avec le monde réel**	vocabulaire de l'utilisateur
3	**Contrôle et liberté**	annuler, revenir en arrière
4	**Cohérence et standards**	mêmes mots, mêmes gestes
5	**Prévention des erreurs**	confirmations, contraintes
6	**Reconnaître plutôt que se rappeler**	choix visibles
7	**Flexibilité et efficacité**	raccourcis
8	**Design esthétique et minimaliste**	pas d'information superflue

Le concept

Les dix heuristiques d'utilisabilité de Nielsen (suite)

#	Heuristique	Exemple
9	**Aider à reconnaître, diagnostiquer, corriger les erreurs**	message clair, solution proposée
10	**Aide et documentation**	accessible, courte

Une **évaluation heuristique** : 2 ou 3 personnes parcourent l'écran en cherchant les violations, notent la **gravité** (cosmétique, mineure, majeure, bloquante).

Accessibilité

Environ 15 % de la population vit avec un handicap. L'accessibilité (a11y) bénéficie à tous (soleil sur l'écran, bruit, main occupée). Les **WCAG** (niveaux A, AA, AAA) définissent des critères :

Le concept

Accessibilité (suite)

Critère	Exemple
Contraste	texte normal : rapport ≥ **4,5:1** (AA) ; grand texte ≥ 3:1
Alternatives textuelles	`alt` sur les images porteuses de sens
Étiquettes de formulaire	`<label for>` associé à chaque champ
Clavier	tout est utilisable sans souris, focus visible
Structure	titres hiérarchisés (h1 → h2 → h3)
Pas seulement la couleur	une erreur ne se signale pas seulement en rouge
Zones tactiles	≥ 44 × 44 px

Rapport de contraste : $(L_1 + 0{,}05) / (L_2 + 0{,}05)$, où L est la **luminance relative** (0 noir, 1 blanc) des deux couleurs, L_1 la plus claire.

Le concept

Lois de l'interaction

- **Loi de Hick** : le temps de décision croît avec le logarithme du nombre de choix : $T = a + b \cdot \log_2(n + 1)$. Réduire les options accélère la décision.
- **Loi de Fitts** : le temps pour atteindre une cible croît avec la distance D et décroît avec sa largeur W : $T = a + b \cdot \log_2(2D / W)$. Les cibles importantes doivent être **grandes** et **proches**. Les bords de l'écran sont infiniment « profonds ».
- **Charge cognitive** : environ 4 à 7 éléments en mémoire de travail. Découper un formulaire long en étapes.
- **Effet de position en série** : on retient le début et la fin d'une liste.

Les formulaires, source principale d'abandon

Le concept

Les formulaires, source principale d'abandon (suite)

Bonne pratique	Effet
Moins de champs (ne demander que le nécessaire)	chaque champ coûte des conversions
Étiquettes **au-dessus** des champs, visibles	lisibilité, accessibilité
Validation **en ligne**, messages précis	corrige tôt
Types de champ adaptés (e-mail, nombre)	clavier mobile adapté
Afficher les règles avant l'erreur (mot de passe)	prévention

Mesurer l'utilisabilité

Un **test d'utilisabilité** : 5 à 8 personnes accomplissent des tâches sur le produit, pendant que l'on observe sans guider. Mesures :

Le concept

Mesurer l'utilisabilité (suite)

Mesure	Définition
Taux de réussite	part des tâches accomplies sans aide (avec intervalle de confiance)
Temps sur tâche	médiane, pas moyenne
Erreurs	nombre et gravité
Satisfaction	questionnaire **SUS** (10 questions, score 0 à 100 ; ≈ 68 = moyenne, > 80 = bon)

Score SUS : pour les questions impaires, contribution = réponse − 1 ; paires : 5 − réponse ; somme × 2,5.

Cinq personnes révèlent ~85 % des problèmes d'utilisabilité majeurs d'un écran (Nielsen), à condition de **répéter** après correction.

Le concept

Intervalle d'un taux de réussite sur un petit échantillon

Avec 5 personnes dont 4 réussissent (80 %), l'intervalle de confiance est large. On utilise l'**intervalle de Wilson** :

```
centre = (p + z²/2n) / (1 + z²/n)      demi-largeur = z·√(p(1−p)/n + z²/4n²) / (1 + z²/n)
```

Pour 4 sur 5 : environ [38 % ; 96 %]. Un test à 5 personnes **trouve des problèmes**, il ne **mesure pas** précisément un taux.

🧠 ***Astuce*** *: fais un test avec cinq personnes avant d'ajouter quoi que ce soit. Le premier problème que tu verras sera presque toujours plus grave que ceux que tu imaginais.*

Exemple pas à pas

Problème : mesurer un entonnoir d'inscription, calculer des contrastes WCAG, appliquer les lois de Hick et de Fitts, noter un test d'utilisabilité (SUS, Wilson), et détecter automatiquement des défauts d'accessibilité dans du HTML.

```
import math
from html.parser import HTMLParser

# ---------------- 1. entonnoir d'inscription ----------------
ETAPES = [("Visite de la page d'inscription", 10000), ("Formulaire commencé", 6200
   ), ("Formulaire envoyé", 3800), ("E-mail confirmé", 2300), ("Premier cours déma
   rré", 1700)]
print("étape                              | personnes | conversion depuis l'étape
   précédente | cumul")
for i, (nom, n) in enumerate(ETAPES):
    prec = ETAPES[i - 1][1] if i else n
    print(f"{nom:34s} | {n:9d} | {n / prec:6.1%}                                  | {
   n / ETAPES[0][1]:5.1%}")
abandons = [(ETAPES[i][0], 1 - ETAPES[i][1] / ETAPES[i - 1][1]) for i in range(1,
```

Exemple pas à pas (suite)

```
    len(ETAPES))]
pire = max(abandons, key=lambda x: x[1])
print("plus fort abandon relatif :", pire[0], f"({pire[1]:.0%})")
assert pire[0] == "E-mail confirmé" and abs(pire[1] - 0.395) < 0.01
assert round(ETAPES[-1][1] / ETAPES[0][1], 2) == 0.17

# ---------------- 2. contraste WCAG ----------------
def luminance(hexa):
    r, g, b = (int(hexa[i:i + 2], 16) / 255 for i in (1, 3, 5))
    def lin(c): return c / 12.92 if c <= 0.03928 else ((c + 0.055) / 1.055) ** 2.4
    return 0.2126 * lin(r) + 0.7152 * lin(g) + 0.0722 * lin(b)
def contraste(a, b):
    la, lb = sorted((luminance(a), luminance(b)), reverse=True)
    return (la + 0.05) / (lb + 0.05)
PALETTE = {"texte gris clair sur blanc": ("#aaaaaa", "#ffffff"), "texte gris foncé
    sur blanc": ("#595959", "#ffffff"), "blanc sur bleu": ("#ffffff", "#0b5cad"),
           "rouge sur blanc (erreur)": ("#ff0000", "#ffffff"), "noir sur blanc": (
   "#000000", "#ffffff"), "orange sur blanc": ("#ff8800", "#ffffff")}
print("combinaison                    | rapport | AA texte normal (≥ 4,5)")
```

Exemple pas à pas (suite)

```
for nom, (c1, c2) in PALETTE.items():
    r = contraste(c1, c2); print(f"{nom:30s} | {r:5.2f}:1 | {'oui' if r >= 4.5 els
   e 'NON'}")
assert round(contraste("#000000", "#ffffff")) == 21 and contraste("#aaaaaa", "#fff
   fff") < 3 and contraste("#595959", "#ffffff") >= 4.5 and contraste("#0b5cad", "
   #ffffff") >= 4.5

# ---------------- 3. lois de Hick et de Fitts ----------------
def hick(n, a=0.2, b=0.15): return a + b * math.log2(n + 1)
def fitts(distance, largeur, a=0.1, b=0.12): return a + b * math.log2(2 * distance
    / largeur + 1)
print("choix | temps de décision (Hick) :", {n: round(hick(n), 2) for n in (2, 4,
   8, 16, 32)})
print("bouton 40 px à 600 px →", round(fitts(600, 40), 2), "s | bouton 120 px à 60
   0 px →", round(fitts(600, 120), 2), "s | 120 px à 200 px →", round(fitts(200, 1
   20), 2), "s")
assert hick(16) < 2 * hick(2) and fitts(600, 120) < fitts(600, 40) and fitts(200,
   120) < fitts(600, 120)
# un menu de 32 options découpé en 2 niveaux de 6 est-il plus rapide ?
```

Exemple pas à pas (suite)

```
print(f"32 choix d'un coup : {hick(32):.2f} s | 2 niveaux de 6 : {2 * hick(6):.2f}
    s")
assert hick(32) < 2 * hick(6)              # (un seul niveau reste plus rapide : l
   a structure doit aussi aider à CHOISIR, pas seulement à réduire le nombre)

# ---------------- 4. test d'utilisabilité : SUS et intervalle de Wilson ---------
   -------
def sus(reponses):                          # 10 réponses de 1 à 5
    total = sum((r - 1) if i % 2 == 0 else (5 - r) for i, r in enumerate(reponses)
   )
    return total * 2.5
PARTICIPANTS = [[4, 2, 4, 2, 4, 1, 5, 2, 4, 2], [3, 3, 3, 4, 3, 3, 3, 3, 3, 4], [5
   , 1, 5, 1, 5, 1, 5, 1, 5, 1], [2, 4, 2, 4, 2, 4, 2, 4, 2, 4], [4, 2, 3, 2, 4, 2
   , 4, 2, 4, 2]]
scores = [sus(p) for p in PARTICIPANTS]
print("scores SUS :", scores, "→ moyenne", sum(scores) / len(scores))
assert scores[2] == 100.0 and scores[3] == 25.0

def wilson(succes, n, z=1.96):
```

Exemple pas à pas (suite)

```
    p = succes / n
    centre = (p + z * z / (2 * n)) / (1 + z * z / n)
    demi = z * math.sqrt(p * (1 - p) / n + z * z / (4 * n * n)) / (1 + z * z / n)
    return centre - demi, centre + demi
bas, haut = wilson(4, 5)
print(f"4 réussites sur 5 : intervalle de confiance à 95 % = [{bas:.0%} ; {haut:.0
   %}]")
bas20, haut20 = wilson(16, 20); print(f"16 réussites sur 20 : [{bas20:.0%} ; {haut
   20:.0%}]")
assert bas < 0.45 and haut > 0.95 and (haut20 - bas20) < (haut - bas)

# ---------------- 5. détection de défauts d'accessibilité dans du HTML ----------
   ------
class Audit(HTMLParser):
    def __init__(self):
        super().__init__(); self.problemes, self.labels_for, self.champs, self.niv
   eaux = [], set(), [], []
    def handle_starttag(self, balise, attrs):
        a = dict(attrs)
```

Exemple pas à pas

Exemple pas à pas (suite)

```
        if balise == "img" and "alt" not in a: self.problemes.append("image sans a
   ttribut alt")
        if balise == "label" and "for" in a: self.labels_for.add(a["for"])
        if balise == "input" and a.get("type") not in ("hidden", "submit"): self.c
   hamps.append(a.get("id"))
        if balise in ("h1", "h2", "h3", "h4"): self.niveaux.append(int(balise[1]))
        if balise == "button" and not a.get("type"): pass
    def fin(self):
        for c in self.champs:
            if c is None or c not in self.labels_for: self.problemes.append(f"cham
   p « {c} » sans étiquette associée")
        for prec, cur in zip(self.niveaux, self.niveaux[1:]):
            if cur > prec + 1: self.problemes.append(f"saut de niveau de titre (h{
   prec} → h{cur})")
        return self.problemes
HTML = '<h1>Inscription</h1><h3>Vos informations</h3><img src="logo.png"><label fo
   r="nom">Nom</label><input id="nom" type="text"><input id="email" type="email"><
   input type="password">'
audit = Audit(); audit.feed(HTML)
```

Exemple pas à pas (suite)

```
for p in audit.fin(): print("  ⚠️", p)
assert len(audit.problemes) == 4
```

Lecture :

1. L'**entonnoir** : seulement 17 % des visiteurs démarrent un cours. Le plus fort abandon relatif désigne l'étape à améliorer en premier (ici la confirmation de l'e-mail, ≈ 39 % de perte, talonnée par l'envoi du formulaire à ≈ 39 %).
2. Le **contraste** : le gris clair #aaa sur blanc n'atteint que ≈ 2,3:1 (échec AA), le gris foncé #595959 passe (≥ 4,5), et le **rouge pur** sur blanc (4,0:1), souvent employé pour les erreurs, échoue lui aussi de peu.
3. **Hick/Fitts** : doubler les options n'ajoute que l'équivalent d'un bit de décision ; agrandir ou rapprocher une cible accélère le pointage.

Exemple pas à pas (suite)

4. **SUS** : un score de 100 (excellent) et un de 25 (mauvais) coexistent chez 5 personnes — un test à 5 personnes **révèle des problèmes**. L'**intervalle de Wilson** sur 4 réussites sur 5 est très large (≈ 38 % à 96 %) ; sur 16 sur 20, il se resserre.
5. L'audit HTML trouve quatre défauts : image sans `alt`, deux champs sans étiquette, saut de titre h1 → h3.

À toi, avec aide

À toi, avec aide

Voici une palette de boutons d'une application :

Usage	Texte	Fond
Bouton principal	`#ffffff`	`#3b82f6`
Bouton secondaire	`#3b82f6`	`#ffffff`
Bouton désactivé	`#9ca3af`	`#e5e7eb`
Message d'erreur	`#dc2626`	`#fef2f2`

1. Calcule le rapport de contraste de chacun et dis lesquels respectent le niveau AA (4,5:1 pour le texte normal).
2. Pour ceux qui échouent, propose une couleur de texte ou de fond corrigée la plus proche qui passe (cherche par assombrissement).
3. Un bouton désactivé a-t-il besoin de respecter le contraste ? Justifie.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Utilise la fonction `contraste` de l'exemple.

Indice 2

Pour corriger, assombris le texte en réduisant chaque composante RVB d'un pas jusqu'à passer 4,5.

Indice 3

Les composants inactifs sont exemptés de l'exigence de contraste WCAG, mais ils doivent rester compréhensibles.

Réponse

```
def luminance(h):
    r, g, b = (int(h[i:i + 2], 16) / 255 for i in (1, 3, 5))
    f = lambda c: c / 12.92 if c <= 0.03928 else ((c + 0.055) / 1.055) ** 2.4
    return 0.2126 * f(r) + 0.7152 * f(g) + 0.0722 * f(b)
def contraste(a, b):
    la, lb = sorted((luminance(a), luminance(b)), reverse=True); return (la + 0.05
   ) / (lb + 0.05)
def assombrir(h, fond, cible=4.5):
    r, g, b = (int(h[i:i + 2], 16) for i in (1, 3, 5))
    while contraste(f"#{r:02x}{g:02x}{b:02x}", fond) < cible and (r, g, b) != (0,
   0, 0):
        r, g, b = max(r - 3, 0), max(g - 3, 0), max(b - 3, 0)
    return f"#{r:02x}{g:02x}{b:02x}"
PAIRES = {"principal": ("#ffffff", "#3b82f6"), "secondaire": ("#3b82f6", "#ffffff"
   ), "désactivé": ("#9ca3af", "#e5e7eb"), "erreur": ("#dc2626", "#fef2f2")}
for nom, (t, f) in PAIRES.items():
    r = contraste(t, f)
    ligne = f"{nom:11s} {r:4.2f}:1 {'OK' if r >= 4.5 else 'ÉCHEC'}"
    if r < 4.5 and nom != "désactivé":
```

Réponse (suite)

```
        if luminance(t) > luminance(f): ligne += f"  → fond assombri : {assombrir(
    f, t)}"            # texte clair : on assombrit le fond
        else: ligne += f"  → texte assombri : {assombrir(t, f)}"
                       # texte foncé : on assombrit le texte
    print(ligne)
assert contraste("#ffffff", "#3b82f6") < 4.5 and contraste("#3b82f6", "#ffffff") <
    4.5 and contraste("#dc2626", "#fef2f2") < 4.5
```

Le bouton principal et le bouton secondaire (≈ 3,7:1) échouent : il faut **assombrir le bleu** (par exemple #2563eb, ≈ 5,2:1). Le message d'erreur (rouge sur fond rosé, ≈ 4,4:1) échoue de peu : un rouge un peu plus foncé (#b91c1c) passe. Le bouton **désactivé** est **exempté** des critères de contraste de WCAG, mais il doit rester **identifiable** comme désactivé et ne pas être la seule information (étiquette, explication de la raison) : l'accessibilité ne se réduit pas au calcul.

À toi, en solo

Analyse d'un test d'utilisabilité. Tu as mené un test avec 8 personnes sur 4 tâches. Les données (à générer avec une graine fixée) : pour chaque personne et chaque tâche, réussite (oui/non), temps (secondes, loi log-normale), nombre d'erreurs ; plus un questionnaire SUS par personne.

1. calcule, par tâche : taux de réussite avec intervalle de Wilson, temps médian et p90, nombre moyen d'erreurs ;
2. calcule le score SUS moyen avec un intervalle bootstrap et situe-le (moyenne 68) ;
3. classe les tâches par **priorité de correction** = (1 − taux de réussite) × gravité × fréquence d'usage (valeurs fournies) ;
4. simule l'effet de « corriger la tâche prioritaire » : le taux passe de x à y ; recalcule le score de succès global ;
5. estime combien de personnes il faudrait pour que l'intervalle de Wilson d'un taux de 80 % ait une largeur inférieure à 20 points ;
6. rédige le rapport : 3 constats, preuves chiffrées, recommandation.

À toi, en solo · réponse

Pistes de correction

```
import math, random, statistics

def wilson(s, n, z=1.96):
    p = s / n; c = (p + z * z / (2 * n)) / (1 + z * z / n)
    d = z * math.sqrt(p * (1 - p) / n + z * z / (4 * n * n)) / (1 + z * z / n); re
   turn c - d, c + d
def percentile(v, p):
    v = sorted(v); return v[min(len(v) - 1, math.ceil(p / 100 * len(v)) - 1)]

rng = random.Random(3)
TACHES = {"trouver un cours": (0.95, 20, 0.3, 1, 0.9), "s'inscrire": (0.60, 60, 1.
   2, 3, 0.7), "reprendre un cours": (0.80, 25, 0.6, 2, 0.8), "contacter le suppor
   t": (0.50, 80, 1.5, 2, 0.2)}
# (probabilité de réussite, temps typique, erreurs moyennes, gravité 1-3, fréquenc
   e d'usage)
donnees = {t: [(rng.random() < p, rng.lognormvariate(math.log(temps), 0.4), max(0,
    round(rng.gauss(err, 0.7)))) for _ in range(8)] for t, (p, temps, err, g, f) i
   n TACHES.items()}
priorites = {}
```

À toi, en solo · réponse

Pistes de correction (suite)

```
for t, lignes in donnees.items():
    s = sum(1 for ok, _, _ in lignes if ok); bas, haut = wilson(s, 8)
    temps = [x for _, x, _ in lignes]; erreurs = sum(e for _, _, e in lignes) / 8
    gravite, frequence = TACHES[t][3], TACHES[t][4]
    priorites[t] = (1 - s / 8) * gravite * frequence
    print(f"{t:22s} réussite {s}/8 [{bas:4.0%};{haut:4.0%}] | temps médian {statis
   tics.median(temps):5.1f}s p90 {percentile(temps, 90):5.1f}s | erreurs {erreurs:
   3.1f} | priorité {priorites[t]:.2f}")
prioritaire = max(priorites, key=priorites.get)
print("tâche à corriger en premier :", prioritaire)
assert prioritaire in ("s'inscrire", "reprendre un cours", "contacter le support")

def sus(r): return sum((x - 1) if i % 2 == 0 else (5 - x) for i, x in enumerate(r)
   ) * 2.5
scores = [sus([min(5, max(1, round(rng.gauss(4 if i % 2 == 0 else 2, 0.9)))) for i
    in range(10)]) for _ in range(8)]
boot = sorted(statistics.mean(rng.choices(scores, k=8)) for _ in range(2000))
print(f"SUS moyen {statistics.mean(scores):.1f}, IC95 bootstrap [{boot[50]:.1f} ;
   {boot[1950]:.1f}] (moyenne du secteur : 68)")
```

À toi, en solo · réponse

Pistes de correction (suite)

```
n = next(n for n in range(5, 500) if (lambda b, h: h - b < 0.20)(*wilson(round(0.8
    * n), n)))
print("personnes nécessaires pour un intervalle de Wilson de moins de 20 points à
   80 % :", n)
assert n > 50
```

Avec **8 personnes**, les intervalles de Wilson sont très larges : le test **révèle des problèmes** (ici surtout l'inscription), il ne mesure pas un taux avec précision. Pour **quantifier** un taux de réussite à ± 10 points, il faudrait plus de 60 personnes : c'est le rôle des données d'usage en production (entonnoir), combinées à des tests qualitatifs répétés après correction.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Décider d'un design « au goût »	Contraste insuffisant, formulaires pénibles	Principes + mesures (contraste, entonnoir, test à 5 personnes)
Trop de champs dans un formulaire	Chaque champ coûte des conversions	Ne demander que le nécessaire, découper
Signaler les erreurs par la couleur seule	Illisible pour les daltoniens, ambigu	Texte + icône + couleur
Conclure d'un test à 5 personnes sur un taux	Intervalle énorme	Les tests qualitatifs trouvent des problèmes ; les données mesurent

Mini-boss

Mini-boss

Audit UX d'un parcours d'inscription. Pour le parcours d'inscription de HashBox (5 écrans HTML minimalistes que tu écris) :

1. construis un **entonnoir** à partir d'un journal d'événements simulé (10 000 sessions, graine fixée) et identifie le plus fort abandon ;
2. écris un **auditeur d'accessibilité** (contraste de la palette CSS, alt, étiquettes, hiérarchie de titres, tailles de cibles tactiles) ; applique-le à la version initiale (au moins 8 défauts) puis à la version corrigée (0 défaut critique) ;
3. évalue le formulaire avec une **checklist heuristique** (10 critères) : note chaque violation par gravité ;
4. modélise le temps de remplissage (Hick pour les menus, Fitts pour les boutons, frappe) et compare 14 champs en une page contre 3 étapes ;
5. simule un test d'utilisabilité (8 personnes) avant et après correction : taux de réussite avec Wilson, temps médian, SUS ;

Mini-boss

Mini-boss (suite)

6. conclus par un tableau avant/après et un plan de 5 corrections priorisées par (gain attendu × facilité).

Mini-boss · réponse

Pistes de correction

Un bon travail : entonnoir calculé depuis des événements, auditeur HTML/CSS avec luminance WCAG, checklist Nielsen avec gravité, modèle de temps cohérent (la version en étapes réduit la charge mais ajoute des clics : calculer l'équilibre), simulation de test avec intervalles, tableau avant/après, plan priorisé (par exemple : réduire à 5 champs, étiquettes visibles, contraste ≥ 4,5, messages d'erreur précis, validation en ligne).

Quiz éclair

Quiz éclair

1. Le rapport de contraste minimal AA pour un texte normal est :

A. 2:1
B. 10:1
C. 4,5:1

2. Selon la loi de Fitts, atteindre une cible est plus rapide si elle est :

A. plus grande et plus proche
B. plus petite et plus lointaine
C. cachée dans un menu

3. Un test d'utilisabilité avec 5 personnes sert surtout à :

A. mesurer précisément un taux de réussite

Quiz éclair (suite)

B. découvrir des problèmes d'utilisabilité majeurs
C. remplacer les données d'usage

4. 🔁 *Révision mission 04* : Un MVP est défini avant tout par :

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

5. 🔁 *Révision mission 04* : La première tranche d'une cartographie d'histoires doit :

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

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Où, dans ton produit, les gens abandonnent-ils le plus, et le sais-tu par des données ou par intuition ?
2. Quand as-tu regardé quelqu'un utiliser ton produit pour la dernière fois, sans l'aider ?

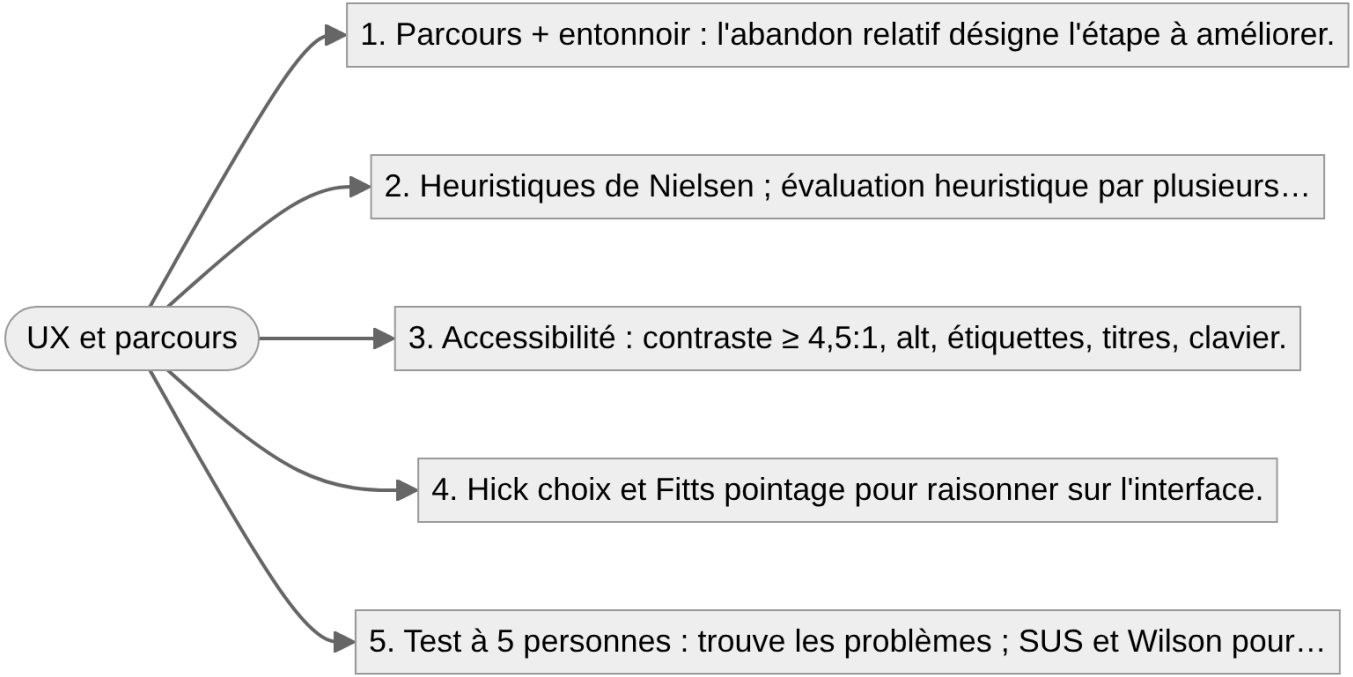
À retenir

À retenir

1. Parcours + entonnoir : l'abandon relatif désigne l'étape à améliorer.
2. Heuristiques de Nielsen ; évaluation heuristique par plusieurs personnes.
3. Accessibilité : contraste ≥ 4,5:1, alt, étiquettes, titres, clavier.
4. Hick (choix) et Fitts (pointage) pour raisonner sur l'interface.
5. Test à 5 personnes : trouve les problèmes ; SUS et Wilson pour mesurer prudemment.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Ami de l'utilisateur

Prochaine mission : Architecture orientée produit

