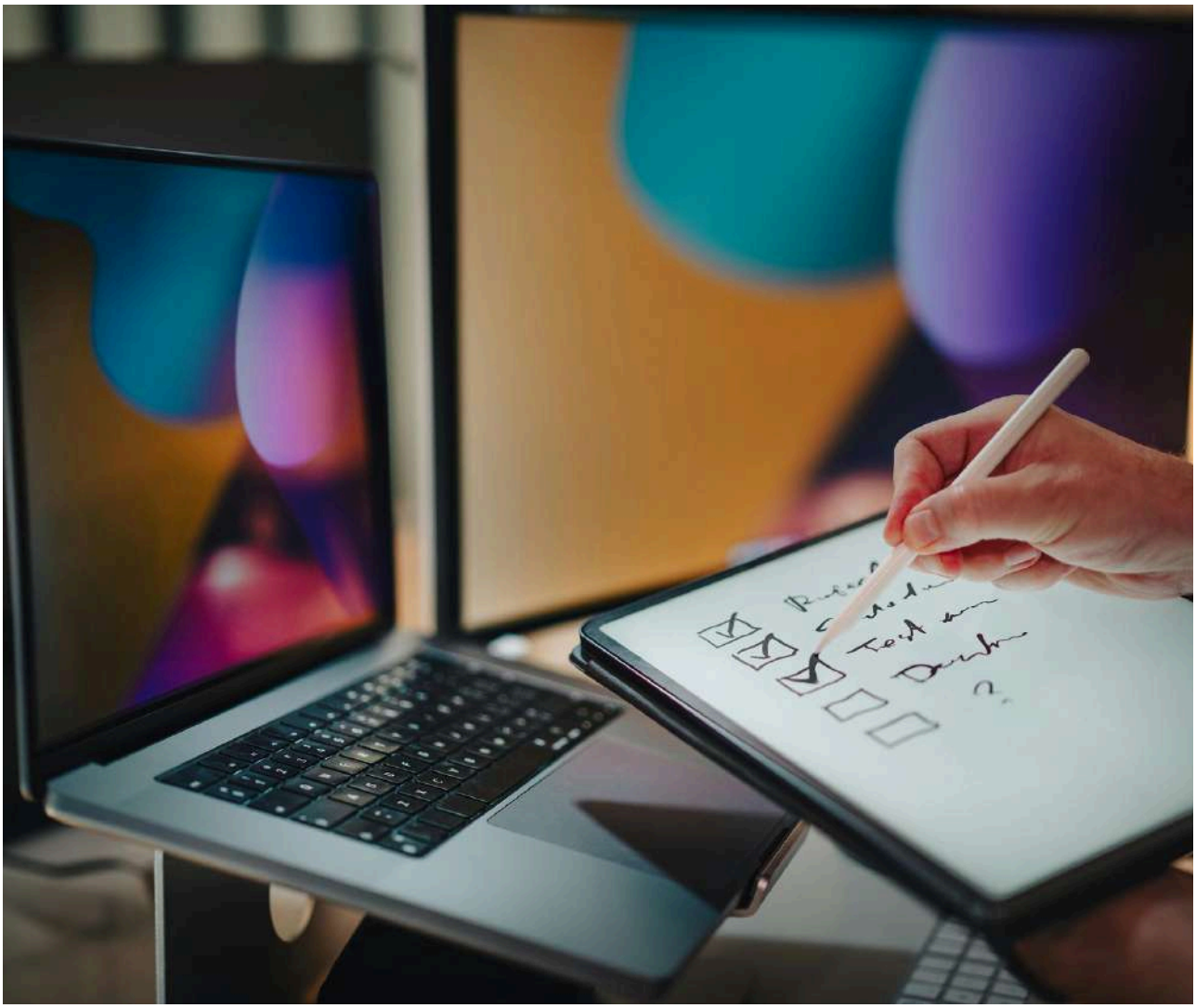
HS NUMÉRO 1 : NOVEMBRE 2025

QUALITÉ & TESTS

| | | |
|---|---|---|
| DOSSIER SUR LA STRATEGIE DE TESTS | AVIS D'EXPERT : MARC HAGE CHAHINE | CLOUDNETCARE |

# LE MAG DU TESTEUR

HORS SÉRIE



# LETTRE DE L'EDITEUR

Ce hors-série du Mag du Testeur marque une étape un peu particulière : celle où la technique rencontre la stratégie.

Parce que tester ne consiste pas seulement à exécuter des cas, mais à construire une démarche réfléchie, mesurée et alignée avec les objectifs du projet.

Dans ce numéro, je vous propose de prendre un peu de hauteur :

- comment bâtir une stratégie de test solide et réaliste ?
- comment articuler les niveaux de test, les risques et les contraintes d'équipe ?
- et surtout, comment tirer parti d'outils comme CloudNetCare pour automatiser intelligemment, sans perdre la maîtrise de la qualité ?

Ce hors-série est à la fois un guide, une boîte à idées et une invitation à repenser notre façon d'aborder la qualité.

Plus que jamais, le test doit être perçu comme un levier de décision, pas comme une simple étape de validation.

Bonne lecture. Et comme toujours, vos retours, vos retours d'expérience et vos critiques sont les bienvenus.

Parce que le Mag du Testeur, c'est avant tout une aventure collective, portée par des professionnels passionnés et curieux.

Un grand merci à Lionel Gravereau et Marc Hage Chahine pour leur aide et participation à ce numéro.


**Fanny Velsin**

# SOMMAIRE


5 Dossier stratégie de test

6 Documents

7 Stratégie de test, à quoi ça sert ?

10 SLA, SLO, SLI : quelles différences ?

11 Quel angle adopter ?

12 Métro des stratégies

28 Avis d'expert : stratégie de test agile par Marc Hage Chahine

40 Cas concret : stratégie de test de performance

45 Cloudnetcare

46 Mécanisme et compatibilité

47 Comment le mettre en place ?

48 Présentation de l'interface

50 Cas concret : site d'ecommerce

60 Et l'IA dans tout ça ?

65 Conclusion

**Nous serions ravis de lire vos impressions et suggestions pour les prochains numéros :**

**feedback@lemagdutesteur.fr**

# DOSSIER

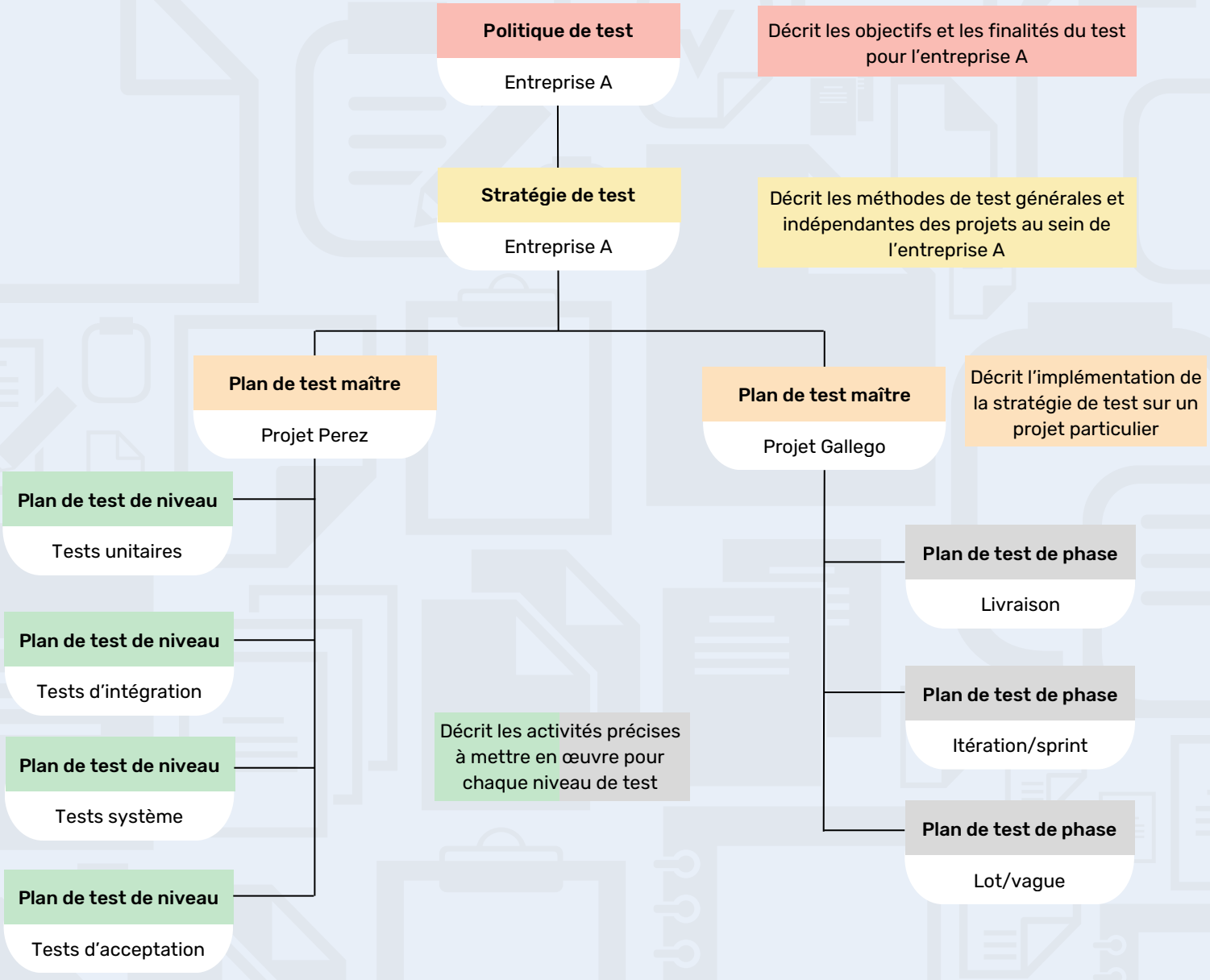
# STRATÉGIE DE TEST

# DOCUMENTS

Conformément au syllabus ISTQB, la documentation de gestion des tests est généralement produite dans ce cadre. Les intitulés et périmètres peuvent varier selon les organisations et les projets, mais on retrouve le plus souvent le schéma ci-dessous.

**Exemple** : au sein de l'entreprise A, deux projets (Perez et Gallego) disposent chacun d'un plan de test maître et de leurs plans de test de niveau propres.

La forme de ces documents varie selon le contexte : regroupés en un seul document, séparés, parfois implicites ou intégrés à une méthodologie existante. Les organisations plus grandes et formelles documentent généralement davantage ; les structures plus petites et moins formelles, moins. Ce syllabus décrit chaque type séparément ; en pratique, c'est le contexte organisationnel et projet qui guide leur bon usage.



# STRATEGIE DE TEST, A QUOI CA SERT ?

La qualité ne se “rattrape” pas en fin de projet : elle se conçoit, se gouverne et se mesure. Une bonne stratégie de test n’est pas une liste de cas de test : c’est un cadre de décision qui dit quoi valider, quand, avec qui, avec quels moyens et surtout pourquoi.

Dans l’ISTQB, la stratégie de test est un document organisationnel, aligné sur la politique qualité, qui décrit comment on teste “en général” (approches, niveaux, types, techniques, standards, outils, gouvernance).

**Avant la moindre exécution, clarifiez :**

- Objectifs qualité (ex. : zéro régression critique sur le parcours d’achat).
- Périmètre : fonctionnel (ce que le système doit faire) vs non-fonctionnel (performance, sécurité, accessibilité, UX). Une stratégie efficace couvre les deux, explicitement.
- Risques & hypothèses : sur quoi mise-t-on (données, intégrations, environnements) et que fait-on si ça tombe ?
- Exigences mesurables (SLA) & indicateurs clefs (KPI) qui serviront de critères de sortie.

**Qui rédige ?**
Test Manager/QA Lead avec PO, Archi, SecOps, Ops.

**Qui valide ?**
Sponsor produit/IT + responsable qualité.

**Quand la mettre à jour ?**
À chaque bascule de contexte : nouvelle release majeure, changement d’archi, arrivée d’une contrainte réglementaire, incident majeur post-prod.

**Combien de pages ?**
Aussi court que possible, assez long pour être actionnable (souvent 4–8 pages + annexes).

**Où la trouver ?**
Wiki/Repo versionné, même place que le Master Test Plan.

**Différence “Stratégie” vs “Plan” ?**
La stratégie fixe les approches & règles du jeu de l’organisation ; le plan applique ces règles à un projet donné (ressources, jalons, risques spécifiques).

## C’est quoi un RACI ?

Un RACI est une matrice qui clarifie qui fait quoi sur chaque activité / décision :

- R : Responsible (Responsable d’exécution) : celles/ceux qui réalisent le travail. Il peut y en avoir plusieurs.
- **A** : Accountable (Autorité / redevable) : la personne qui porte la décision/le livrable et tranche. Une seule personne par ligne.
- C : Consulted (Consulté·e) : on les sollicite avant l’action pour avis d’expert ; échanges bilatéraux.
- I : Informed (Informé·e) : on les tient au courant après l’action/décision ; diffusion unidirectionnelle.

**Variantes fréquentes : RASCI (ajoute Support), CAIRO, DACI. L’idée reste la même : rendre visible la responsabilité et la décision.**

**Ci dessous, un exemple sous forme de tableau d’un extrait de RACI pour une anti-régression automatisée (tous les rôles et activités ne sont pas présents) :**

## Pourquoi c’est important pour une stratégie de test ?

Parce qu’une stratégie n’est pas qu’un document : c’est un mécanisme de décision et d’exécution.

**Une bonne matrice RACI :**

- Accélère les décisions (on sait qui tranche : « A » unique).
- Évite les trous dans la raquette (chaque activité a au moins un « R »).
- Rend auditable la gouvernance (utile en réglementaire et pour l’alignement avec la politique qualité).
- Protège l’indépendance du test (tu peux déclarer explicitement qui vérifie quoi à chaque niveau).
- Fluidifie l’onboarding (les nouveaux comprennent tout de suite le qui-fait-quoi).
- Débloque l’outillage (qui choisit/finance les outils, qui en assure l’exploitation).
- Stabilise le reporting (qui produit, qui lit, qui agit).

| Activité | Product Owner | Test Manager | Dev Lead | DevOps |
|---|---|---|---|---|
| Définir la pyramide de tests & périmètre auto | C | R | A | I |
| Choisir les techs & outils (UI, API, visuel, perf) | I | A | C | R |
| Politique de flakiness & temps de réparation | I | R | C | A |
| Stratégie données & environnements | I | A | C | R |
| Reporting (qui lit quoi, quand) | A | R | I | I |

## Retour de terrain (Lionel GRAVEREAU)
## Non-fonctionnel : penser tôt, tester tôt

**Sur plusieurs projets, j'ai observé que traiter les tests non fonctionnels (performance, sécurité, résilience, etc.) dès l'amont et les activer très tôt (micro-bench* côté dev, perf smoke** en CI, puis campagnes plus complètes avant la pré-prod) évite des reprises coûteuses.**

**Dans des contextes sensibles comme par exemple un logiciel de trading avec un objectif de réponse < 2 secondes, attendre la fin des tests fonctionnels pour mesurer la performance peut conduire à réinterroger des choix d'architecture.**

**En définissant les objectifs (SLO) et en mesurant régulièrement tout au long du cycle, les équipes détectent plus tôt les goulets d'étranglement et convergent plus vite vers la cible, sans avoir un travail massif à réaliser.**

**L'idée n'est pas de chercher des responsables, mais de sécuriser collectivement le produit : objectifs clairs, mesures rapides, boucles de feedback continues, y compris pour des services SaaS.**

### *Micro-bench

Un micro-benchmark mesure le temps d'exécution d'un petit bout de code (fonction, algo, requête SQL spécifique) en isolation, côté dev.

**Objectif** : vérifier qu'un calcul/parseur/tri respecte un budget de temps (ex. < 5 ms) avant même d'assembler le reste.

**Bonnes pratiques** : warm-up, plusieurs itérations, machine stable, comparer les ordres de grandeur (pas prédire la prod).

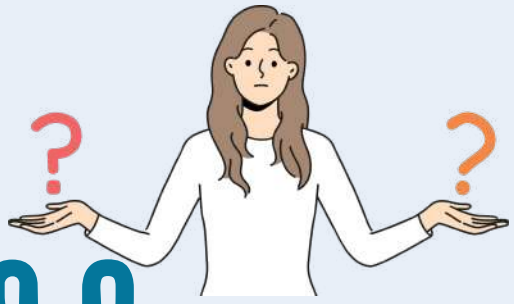
### **Perf smoke

Un perf smoke test est un test de performance ultra-court et léger intégré à la CI.

**Objectif** : détecter vite une régression flagrante (latence qui explose, erreurs) sur 1–3 endpoints clés, sans faire une vraie campagne de charge.

**Typique** : 30–120 s, 1–10 VUs, seuils simples (p95 < X ms, error rate < Y %). Si ça dépasse, on fail la pipeline.

# SLA, SLO, SLI : QUELLES DIFFERENCES ?



### SLA (Service Level Agreement)

C'est l'engagement externe/contractuel (avec remèdes/ pénalités).

**Exemple** : disponibilité mensuelle garantie 99,5 % Sinon vous devrez par exemple donner un crédit si cette garantie n'est pas atteinte. Dans d'autres cas, cela peut être des pénalités financières.

Le SLA s'appuie sur des SLO, mais reste souvent un peu moins strict pour limiter le risque contractuel.

### SLO (Service Level Objective)

C'est la cible interne appliquée à un SLI sur une période.

**Exemple** : p95 < 300 ms sur /checkout pendant 28 jours, disponibilité 99,9 % sur 30 jours.

Donne l'error budget (la part d'écart "tolérable").

Exemple : SLO 99,9 %/30 j ⇒ 0,1 % d'indispo ≈ 43 min 12 s au total.

### SLI (Service Level Indicator)

C'est ce qu'on mesure.

**Exemple** : la latence p95 du /checkout, taux d'erreurs 5xx, disponibilité minute par minute.

### C'est quoi le p95 ?

p95 = 95e percentile.

Ça veut dire que 95 % des mesures sont plus rapides (ou plus petites) que cette valeur, et 5 % sont plus lentes (ou plus grandes).

Si le p95 du /checkout est 300 ms, cela signifie que 95 % des requêtes de /checkout ont répondu en moins de 300 ms. Les 5 % restantes ont dépassé 300 ms.

Avec un autre exemple : sur 100 clients qui passent à la caisse, du plus rapide au plus lent, le p95, c'est le 95e client dans cette liste triée. Tous avant lui ont été plus rapides, ceux après plus lents.

Il existe également p90, p95, p99, ...

**Quand utiliser p90, p95, p99 ?**

- p90 : vue "confort" globale
- p95 : standard produit (équilibre entre confort et sévérité).
- p99 : exigeant/critique (on regarde presque les pires cas).

# QUEL ANGLE ADOPTER ?

## Boite blanche :



On conçoit des tests en connaissant la structure interne (code, chemins, conditions). L'objectif est de couvrir des éléments structurels (instructions, branches, conditions).

Majoritairement unitaires et intégration bas niveau, mais peut compléter un contrôle fonctionnel (p. ex. forcer des branches).

**Vous devez choisir la boite blanche si**

- **Vous devez assurer qu'un algorithme exécute tous ses chemins.**
- **Vous sécurisez une régression fine sur du code critique.**
- **Vous testez à des seuils de couverture (instruction/ décisions/ conditions/ chemins)**

## Boite grise :

On teste le comportement fonctionnel en s'aidant d'informations internes partielles (schéma BDD, logs, séquence d'appels, feature flags) pour mieux cibler les scénarios et détecter des défauts d'intégration.

Fréquent en end-to-end, API, microservices, sécurité logique.

**Vous devez choisir la boite grise si :**

- **Vous voulez prioriser en fonction de l'architecture (flux sensibles, caches).**
- **Vous testez des interactions (file de messages, BDD, services tiers).**
- **Vous exploitez logs/traces pour générer des scénarios réalistes.**

## Boite noire :

On teste le comportement observable d'un composant/système sans connaître l'interne (code, structures).

On se base sur spécifications, règles métier, maquettes, API contract, etc.

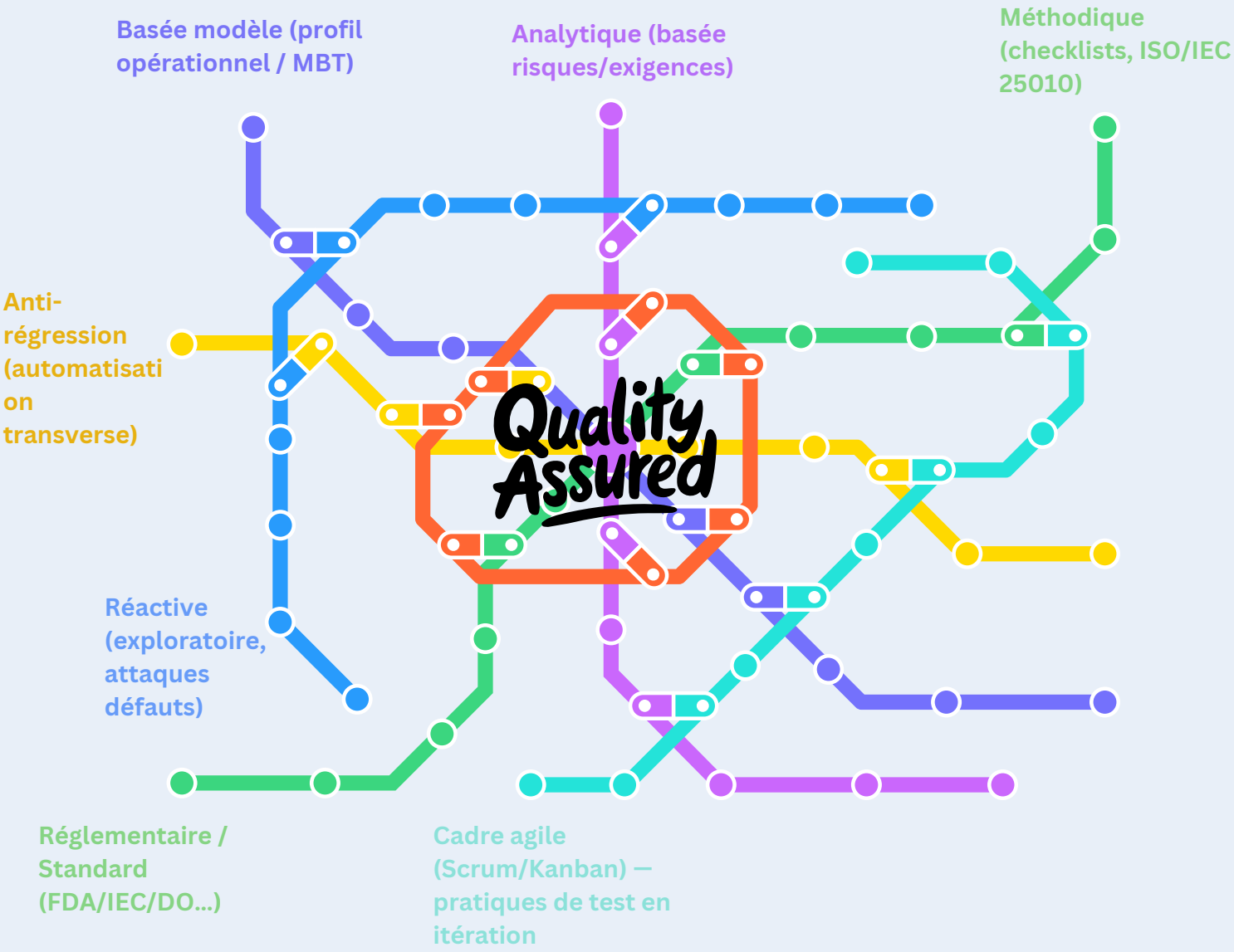
Typique des tests fonctionnels système et acceptation.

**Vous devez choisir boite noire si :**

- **Vous validez des exigences, des règles métier ou des contrats d'API.**
- **Vous faites de l'acceptation avec des users ou Product Owner.**
- **Vous travaillez par cas d'usage avec données et rôles variés.**

# METRO DES STRATEGIES

Il existe plusieurs stratégies, voyons ensemble quelques unes d'entre elles.




Des stratégies différentes peuvent être nécessaires court/long terme et selon le modèle de développement.

## Analytique (basée risques/exigences)

L'idée est de relier directement les tests aux risques métiers et aux exigences formalisées.

L'objectif est de mettre ses ressources là où le risque est le plus élevé (ex. sécurité des paiements, disponibilité d'un service critique). On parle de risk-based testing (**RBT**).

**Quand l'utiliser** :

- Lors d'une conformité réglementaire attendue (banque, médical, assurance...).
- Lors d'exigences fonctionnelles détaillées et validées (par ex. backlog stable, cahier des charges clair).
- Lors d'enjeux business critiques comme une régression pourrait générer des pertes directes de CA ou une sanction légale.
- Lors de projets avec deadline figée et qu'on ne peut pas tout tester, donc on priorise selon les risques.

**Livrables** :

- Matrice risques : tableau croisant les exigences, leurs risques, et les tests associés (voir prochaine page)
- Rapports de couverture par criticité : ex. 100 % des risques "critiques" couverts, 70 % des risques "moyens", 20 % des risques "faibles"
- Rapports par exigence : état de test lié à chaque exigence (validation, défauts, statut).


**Métriques** :

- % de couverture des risques élevés (objectif : 100 %)
- Nombre de défauts par classe de risque (faible, moyen, élevé)
- Tendance résiduelle des risques : y a-t-il encore des zones critiques non validées ?
- Délai moyen de correction pour un risque élevé.


**Limites**

- Nécessite une base d'exigences fiable et stable. Si les exigences changent trop, la stratégie perd son sens.
- Risque de se concentrer uniquement sur le "papier" et pas sur la réalité terrain.

Ci-dessous, une matrice d'évaluation des risques (Risk Assessment Matrix), qui est l'outil de base utilisé en RBT pour :

- croiser la probabilité d'occurrence d'un risque (fréquent, probable, occasionnel…)
- avec la sévérité de ses conséquences (catastrophique, critique, marginal, négligeable).

Le résultat donne une priorité de test :

- Elevé en rouge qui doit absolument être testé (test automatique, test précoce, forte couverture).
- Sérieux/médium en orange/jaune qui doit être couvert mais avec effort proportionné.
- faible/ Eliminé en vert/bleu qui peut être testé manuellement, échantillonné, ou parfois ignoré.

| **Probabilité\ Gravité** | **Catastrophique (pertes majeures, sécurité critique, légalité)** | **Critique (impact fort sur business, réputation)** | **Marginal (inconfort utilisateur, perte mineure)** | **Négligeable (impact cosmétique, faible gêne)** |
|---|---|---|---|---|
| Fréquente (se produit souvent) | Elevé | Elevé | Sérieux | Médium |
| Probable (peut survenir régulièrement) | Elevé | Elevé | Sérieux | Médium |
| Occasionnelle (rare mais possible) | Elevé | Sérieux | Médium | Faible |
| Rare (peu probable) | Sérieux | Sérieux | Médium | Faible |
| Très improbable | Médium | Médium | Médium | Faible |
| Eliminé (impossible, déjà maîtrisé) | Eliminé | | | |

## Basée modèle (profil opérationnel / MBT)

L'idée est de modéliser le système ou ses usages pour générer des tests de manière systématique.

On ne part pas seulement des exigences écrites, mais on formalise un modèle (diagramme d'états, transitions, profils d'utilisation, flux de charge).

L'objectif est d'explorer toutes les combinaisons possibles (chemins, états, transitions), de vérifier que les SLO sont respectés sous différents scénarios d'usage, de simuler le réel plutôt que de tester seulement sur papier.

On parle de **MBT** (model based testing)

**Quand l'utiliser :**

- Quand l'application présente une forte variabilité d'usage (nombreux parcours utilisateurs possibles).
- Quand il y a une montée en charge importante ou des scénarios extrêmes à couvrir.
- Quand on cherche la robustesse : systèmes critiques (paiement, trading, aviation, médical).

**Livrables :**

- Modèles formalisés
  - Diagramme d'états-transitions (UML, BPMN, GraphWalker).
  - Profils de charge utilisateurs (ex. navigation catalogue vs. paiement)
  - Arbre de décision ou diagramme de flux.
- Scénarios dérivés automatiquement
  - Générés à partir du modèle (tous les chemins, transitions critiques)
  - Permettent d'éviter les oublis humains.
- Oracles
  - Références automatiques qui définissent le comportement attendu du système
  - Peuvent être implémentés directement dans l'outil de test.

**Métriques** :

- % des états ou transitions explorées par les tests.
- Respect des SLO/SLI : latence, débit, disponibilité sous différents profils.
- Nombre de défauts détectés sous profils extrêmes (stress tests).
- Évolution du modèle : effort de mise à jour (combien d'adaptations par release).

**Retour de terrain (Marc Hage Chahine)**

**De mon expérience avec Yest, j'ai constaté que son utilisation exige une manière de penser différente : il faut raisonner en termes de workflow produit plutôt qu'en tests centrés sur une fonctionnalité.**

**C'est le principal point que j'ai dû travailler pour adopter efficacement Yest. Et je pense que cette différence de logique explique en grande partie les difficultés, voire les rejets, rencontrés lors de l'implémentation du BPMN (modèle de processus métier et notation)**

**Limites**

- Effort initial élevé : construire un modèle juste et complet prend du temps.
- Maintenance lourde : le modèle doit évoluer à chaque changement d'architecture ou de fonctionnalités.
- Besoin de compétences spécifiques : tous les testeurs ne maîtrisent pas la modélisation formelle.
- Risque de créer un modèle théorique déconnecté du réel si on ne l'alimente pas avec de la donnée terrain (logs, analytics, usage réel).

## Méthodique (checklists, ISO/IEC 25002)

Cette approche consiste à s'appuyer sur des référentiels stables et des checklists de qualité pour encadrer le test et sécuriser la maintenance dans le temps.

Ici, l'idée n'est pas d'analyser les risques ou de modéliser le comportement du système, mais d'utiliser des critères établis et répétables pour garantir que les points critiques soient toujours vérifiés.

C'est une stratégie de constance et de rigueur, idéale pour les produits qui évoluent fréquemment ou qui sont déjà bien maîtrisés.

Elle sert à maintenir un niveau de qualité durable, surtout dans les contextes où le produit vit plusieurs années (application SaaS, site e-commerce, logiciel interne...).

**Quand l'utiliser** :

- Quand le produit est déjà en production et que la plupart des risques sont connus.
- Quand on fait face à des releases fréquentes et à un besoin de vérifications rapides mais systématiques.
- Quand l'organisation dispose d'un référentiel qualité (ISO/IEC 25002, normes internes, guides UI/UX, critères RGAA, etc.).
- Quand le test doit être reproductible et facilement transférable à d'autres équipes (nouveaux testeurs, prestataires...).
- Quand la stabilité et la traçabilité priment sur l'exploration.

- Quand il y a beaucoup de paramètres à vérifier. Le risque d'oubli d'un élément est alors assez élevé surtout si les vérifications sont fréquentes et manuelles

**Qu'est ce que ISO/IEC 25002?**

**La norme ISO/IEC 25002:2024 fait partie de la série 25000 (SQuaRE – Systems and software Quality Requirements and Evaluation).**

**Elle insiste sur le fait que tout modèle de qualité doit être contextualisé : les caractéristiques à privilégier, leur pondération, les seuils d'acceptabilité varient selon le domaine, le public ciblé, les contraintes techniques ou réglementaires.**

**La norme reconnaît que les modèles de qualité ne doivent pas être figés : ils doivent évoluer avec le système, les usages, les retours terrain, les nouvelles contraintes. ITEH Standards+1**

**Chaque version ou itération du logiciel peut nécessiter des ajustements dans les poids, seuils, ou même les caractéristiques retenues.**

**Retour de terrain (Marc Hage Chahine)**
**Cette approche est souvent utilisée en complément d'une autre, notamment dans le cadre d'une DoD ou d'une DoR.**

**Je la recommande lorsqu'il y a de nombreux paramètres à vérifier, car le risque d'oublier un élément devient important, surtout lorsque les vérifications sont fréquentes et effectuées manuellement.**

**Livrables :**

- Checklists qualité
  - Par page, fonctionnalité ou parcours utilisateur.
  - Incluent les critères fonctionnels et non fonctionnels (affichage, sécurité, performance, compatibilité, accessibilité).

Par exemple :
  - Vérifier que les liens sont cliquables
  - Mesurer le temps de chargement < 2s
  - S'assurer que les formulaires sont accessibles clavier

- Tableaux de suivi par critère qualité (ISO/IEC 25002 : fiabilité, performance, sécurité, maintenabilité, portabilité, compatibilité, accessibilité…).

- Documentation associée à la checklist (ex. comment évaluer l'ergonomie, la sécurité, les messages d'erreur…). Cela permet d'uniformiser la méthode d'évaluation entre plusieurs testeurs.

**Métriques suivies**

- % d'items couverts dans la checklist (ex. 95 % des critères passés = bonne couverture)
- Nombre d'écarts récurrents par critère (ex. ergonomie = 4 anomalies sur 10 sprints)
- Tendance de la dette qualité : combien de points de checklist restent non conformes au fil du temps
- Temps moyen de correction par écart (utile pour planifier la maintenance)
- Indice de conformité globale : pourcentage de conformité par domaine (performance, sécurité, accessibilité, etc.).

**Limites**

- Si utilisée seule, elle ne garantit pas la couverture des scénarios critiques : **il faut la compléter par un peu d'analytique (RBT) pour cibler les priorités.**
- Risque de devenir trop mécanique : les testeurs cochent les cases sans se poser la question du "pourquoi".
- Nécessite de mettre à jour les checklists régulièrement pour ne pas tester des critères obsolètes.
- Moins efficace pour découvrir des anomalies nouvelles ou émergentes (ce n'est pas une approche exploratoire).

## Réactive (exploratoire, attaques défauts)

La stratégie réactive repose sur le principe que le test n'est pas uniquement une exécution planifiée, mais aussi un apprentissage en continu.

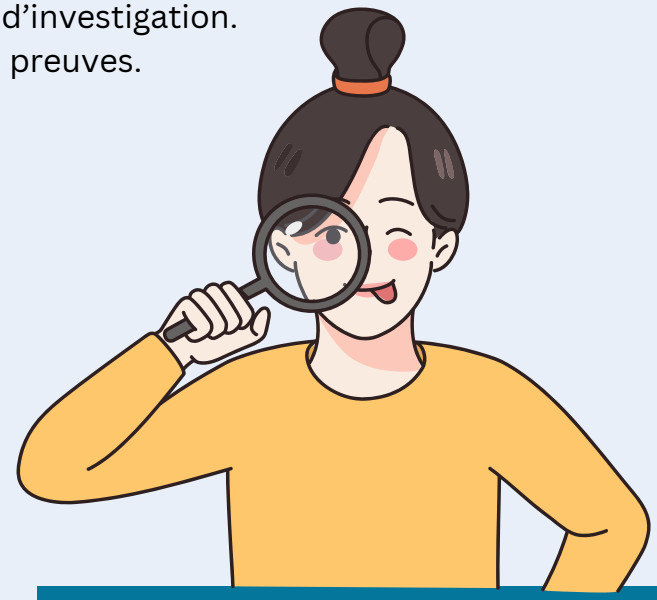
Elle s'appuie sur l'idée que, dans un contexte d'incertitude, de changements rapides ou de manque de documentation, le meilleur moyen de comprendre le système est… **de le tester en explorant.**

L'objectif est double :

1. Apprendre en testant. Chaque action de test produit une connaissance nouvelle sur le produit.
2. Ajuster en temps réel. L'approche est adaptative, réactive à ce que découvre le testeur (un défaut, une incohérence, une surprise).

C'est une approche empirique : on n'a pas tous les cas de test écrits à l'avance, on observe, on hypothétise, on creuse.

On parle souvent de tests exploratoires structurés, parfois aussi appelés attack-based testing ou defect-based testing, selon le degré d'investigation.
preuves.



### Attack-Based Testing

Aussi appelé testing par attaques ciblées est une approche proactive, inspirée du raisonnement d'un "attaquant" ou d'un utilisateur malintentionné.

Le testeur cherche volontairement à casser le système en exploitant ses failles, ses points faibles, ses zones sensibles.

L'idée est de simuler les comportements imprévus et vérifier la résilience du produit.

### Defect-Based Testing

Aussi appelé testing basé sur les défauts connus, cette approche part du constat que les défauts ont tendance à se reproduire dans les mêmes zones ou selon les mêmes schémas.

Elle consiste donc à analyser les historiques de bugs pour concevoir de nouveaux tests ciblés sur ces zones fragiles.

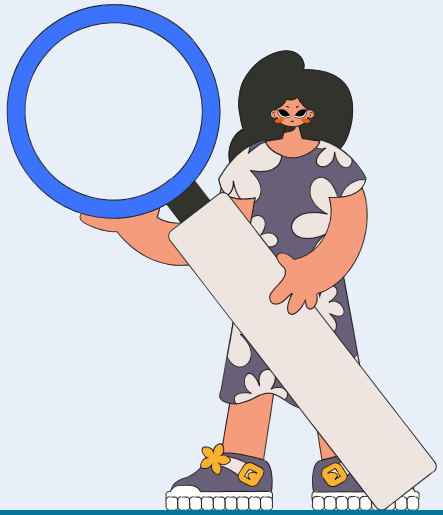
**Quand l'utiliser :**

- Quand la documentation est incomplète ou obsolète.
- Quand le périmètre est flou, instable, ou en évolution constante (MVP, POC, startup, innovation).
- Quand il faut valider un prototype rapidement, sans perdre du temps à formaliser des cas de test trop tôt.
- Quand les exigences sont implicites (attentes utilisateurs non écrites).
- Quand le produit a une forte incertitude technique ou fonctionnelle : nouvelles techno, refonte majeure, forte dette technique.
- **En complément d'autres stratégies : par exemple après une exécution automatisée, pour chercher des anomalies "non prévues".**

**Livrables :**

- Contenu de session
  - Objectif clair : "Explorer la fonctionnalité X sous l'angle Y".
  - Durée limitée (souvent 45 à 90 minutes).
  - Contexte : version, environnement, données.
  - Hypothèses : ce que l'on veut vérifier ou mieux comprendre.

- Journal de session
  - Notes prises pendant le test : actions réalisées, comportements observés, anomalies suspectées.
  - Captures d'écran ou vidéo, logs, commentaires.
  - Structure libre, mais souvent guidée (SBTM – Session-Based Test Management).
- Synthèse de session
  - Récapitulatif des défauts trouvés.
  - Insights : quelles zones sont instables, quelles hypothèses se sont révélées vraies/fausses.
  - Recommandations pour les tests futurs.



**Métriques :**

- Taux de trouvailles par session : nombre de défauts significatifs détectés / session.
- Temps-vers-insight : délai entre le début d'une session et la découverte du premier point critique.
- Taux de reproductibilité : proportion des défauts exploratoires qui ont pu être reproduits et confirmés.
- Regroupement de défaut : parties du produit où les anomalies sont concentrées (identifiées sur plusieurs sessions).
- Taux de couverture implicite : estimation des fonctionnalités explorées (ex. 80 % des flux critiques couverts).

**Limites**

- non reproductible et non automatisable.
- Le résultat dépend du testeur : exige des testeurs expérimentés et curieux, capables d'observer, d'inférer et de structurer leurs découvertes.
- Difficile à tracer et auditer : l'absence de cas de test formalisés peut poser problème dans les environnements **réglementés**.
- Risque de couverture inégale si les sessions ne sont pas bien cadrées ou si les hypothèses sont trop vagues.
- Peut être perçue comme "peu sérieuse" dans des organisations trop centrées sur la documentation. Il faut donc la légitimer par des rapports clairs.

## Réglementaire / Standard (FDA/IEC/DO…)

Cette approche vise à garantir et démontrer la conformité du produit logiciel aux normes et réglementations applicables à son domaine.

L'objectif n'est pas seulement de tester que le logiciel "fonctionne", mais de prouver qu'il a été développé, vérifié et validé selon un processus documenté, traçable et **auditable**.

Chaque exigence, chaque test, chaque défaut doit être justifié, relié et conservé dans un référentiel afin qu'un auditeur ou un organisme certificateur puisse suivre le raisonnement complet, du besoin initial jusqu'à la preuve de validation.

C'est une stratégie où le test devient une activité de conformité, avec un niveau élevé de rigueur documentaire, de relecture et de validation indépendante.

**Quand l'utiliser :**

- Dans les domaines réglementés : médical (FDA 21 CFR Part 820 / IEC 62304), aéronautique (DO-178C), ferroviaire (EN 50128), automobile (ISO 26262), défense, nucléaire, etc.
- Quand le produit doit obtenir une certification externe (autorisation de mise sur le marché, homologation, audit qualité).
- Quand l'entreprise est soumise à des audits qualité réguliers (internes, clients ou autorités).

- Quand la sécurité, la fiabilité ou la sûreté de fonctionnement sont des critères de conformité.
- Quand le projet fait partie d'une chaîne critique (dispositif médical connecté, cockpit avionique, système de freinage…).

**Livrables** :

- Plan de vérification et de validation (V&V Plan)
  - Définit les objectifs de vérification (conformité aux spécifications) et de validation (conformité au besoin utilisateur).
  - Liste les activités prévues : revues, inspections, tests unitaires, d'intégration, système, d'acceptation.
  - Spécifie les rôles, responsabilités et critères d'acceptation.
- Matrice de traçabilité
  - Relie chaque exigence à un ou plusieurs cas de test.
  - Lie également les anomalies identifiées aux exigences concernées.
  - Permet de démontrer la complétude (rien n'a été oublié) et la traçabilité bidirectionnelle (du besoin au test et inversement).

- Revues indépendantes et rapports d'audit
  - Chaque document clé (spécification, plan, rapport) doit être revu et signé par une personne indépendante de l'équipe projet.
  - Les rapports d'audit ou de validation finale constituent des preuves officielles de conformité.
- Les comportements attendus du logiciel sont définis de manière normative ou mathématique. Par exemple : pour un algorithme médical, les valeurs attendues sont définies par un modèle statistique validé.

**Métriques** :

- Taux de conformité documentaire : % de documents validés et signés selon le plan qualité.
- Complétude de la traçabilité : % d'exigences couvertes par au moins un test (souvent visé = 100 %).
- Nombre d'écarts d'audit : non-conformités mineures/majeures détectées lors des inspections.
- Taux de correction des non-conformités : combien d'écarts ont été corrigés et revalidés avant la certification.
- Taux de relecture indépendante : proportion de livrables ayant fait l'objet d'une revue tierce.

**Limites**

- Formalisme élevé : le poids documentaire peut ralentir le développement.
- Risque de perdre le lien avec la valeur métier si les équipes se concentrent uniquement sur la conformité et non sur l'usage.
- Exige un haut niveau de discipline organisationnelle et une culture qualité mature.
- Le testeur doit souvent jongler entre la technique et le juridique (preuve de conformité).
- L'innovation (par exemple en IA ou en DevOps) peut être freinée par la rigidité des processus normatifs.

## Anti-régression

L’objectif de cette stratégie est de sécuriser les évolutions du produit à travers un filet de sécurité automatisé capable de détecter toute régression fonctionnelle, technique ou visuelle avant la mise en production.

Elle consiste à construire un ensemble de tests automatisés multi-niveaux (UI, API, visuel, performance) qui s’exécutent de façon continue dans la pipeline CI/CD.

Chaque modification du code déclenche une série de vérifications rapides et fiables pour garantir que ce qui fonctionnait hier fonctionne encore aujourd’hui.

Cette approche transforme le test en système de surveillance permanent, capable de détecter les impacts indirects des changements dans des environnements complexes ou multi-équipes. Très pratique quand on travaille avec plusieurs API de différentes équipes par exemple.

**Quand l’utiliser** :

- Quand les releases sont fréquentes (quotidiennes, hebdomadaires ou à la demande).
- Quand il existe une forte dette de régression due à l’accumulation de corrections manuelles.
- Quand plusieurs équipes travaillent sur le même produit, souvent en parallèle.

- Quand le périmètre fonctionnel est vaste, et que la répétition manuelle des tests est trop coûteuse.
- Quand la qualité attendue doit être démontrée en continu (ex. SaaS, applications critiques, CI/CD industrialisée).

**Livrables typiques :**

- Suites de tests automatisés intégrées à la CI/CD
  - Tests API, UI, visuels, contractuels, performance, sécurité légère.
  - Organisées par modules, risques ou types de scénarios (filtres, paiement, authentification…).
  - Exécution planifiée à chaque commit, merge request ou nightly build.
- Politique de gestion du flakiness
  - Identification et suivi des tests instables.
  - Définition des règles : rerun autorisé, exclusion temporaire, correction obligatoire.
  - Suivi d’un indicateur “taux de tests flakys”.

- Seuils de qualité / Gates
  - Définis dans la pipeline CI/CD :
    - taux de réussite minimum (ex. 95 % des tests doivent passer),
    - performance max (temps de réponse < 2 s),
    - seuil d'erreurs tolérées (0 erreurs critiques).
  - Si les seuils ne sont pas atteints, le déploiement est bloqué.
- Rapports automatisés
  - Historique des exécutions, tendances, anomalies récurrentes.
  - Intégration dans un tableau de bord qualité (Allure, Grafana, Jenkins).

**Métriques suivies**

- Durée moyenne de pipeline : temps d'exécution total des tests automatisés (objectif : < 30 min).
- Taux de flakiness : % de tests échouant de manière aléatoire sans lien avec le code.
- MTTR (Mean Time To Repair) : temps moyen pour corriger une régression après détection.
- Taux de faux positifs/négatifs : proportion d'erreurs signalées à tort / non détectées.
- Taux de couverture régression : part du périmètre critique couvert par les tests automatisés.
- Ratio de tests maintenus vs obsolètes : indicateur de la santé du patrimoine automatisé.

**Limites**

- **Coûts d'entretien élevés : la maintenance des tests peut devenir une charge si elle n'est pas planifiée dans chaque sprint.**
- **Risque d'effet "illusion de sécurité" : 100 % vert ≠ 100 % testé ; certaines zones non couvertes peuvent encore contenir des défauts.**
- **Faux positifs/négatifs : une mauvaise stabilité des tests (flakiness) nuit à la confiance dans les résultats.**
- **Besoin de priorisation : tous les tests ne méritent pas d'être automatisés. Il faut cibler selon le risque et la fréquence d'usage.**
- **Infrastructure nécessaire : pipelines, environnements isolés, données stables.**

## Cadre agile (Scrum/Kanban), pratiques de test en itération

Dans une approche agile, le test n'est pas une phase distincte, mais une activité continue intégrée au flux de développement.

L'objectif est de garantir la qualité au fil de l'eau, à travers :

- des boucles de feedback rapides,
- des revues fréquentes,
- et une collaboration constante entre testeurs, développeurs, PO et métiers.

Le but du test agile n'est pas seulement de "vérifier après coup", mais de prévenir les défauts dès la conception.

La qualité devient un effort collectif, matérialisé dans la Definition of Ready (DoR), la Definition of Done (DoD) et les User Acceptance Criteria (UAC).

**Quand l'utiliser** :

- Quand le projet suit un cadre itératif ou incrémental (Scrum, Kanban, Scrumban, SAFe, etc.).
- Quand les releases sont fréquentes (parfois quotidiennes).
- Quand le time-to-market est prioritaire et la valeur métier doit être livrée vite.
- Quand les équipes sont autonomes, cross-fonctionnelles et collaboratives (testeurs intégrés à l'équipe).
- Quand le produit évolue en continu (ex. : plateforme SaaS, produit numérique, API exposée).

**Livrables** :

- Critères d'acceptation (User Acceptance Criteria)
  - Définis dès la création des User Stories
  - Servent de base à la validation automatique
- Definition of Ready (DoR)
  - Liste des prérequis avant qu'une User Story puisse être développée. Par exemple : story estimée, critères d'acceptation clairs, jeux de données disponibles, dépendances identifiées.
- Definition of Done (DoD)
  - Ensemble des validations pour considérer une story "terminée". Cela inclut les tests unitaires passés, la couverture code > 80 %, les tests automatisés exécutés, les revues de code effectuées, le déploiement en environnement d'intégration, ...
- Revues collectives (3 Amigos, Sprint Review)
  - Échanges entre Dév / Test / PO pour aligner compréhension et validation.
  - Retours d'utilisateurs et démonstrations en fin de sprint.

- Statuts de tests dans les boards
    - Chaque US suit un statut clair : non testé, échec, succès, bloqué, skip.
    - Ces statuts sont mis à jour à chaque exécution automatisée dans le pipeline CI/CD

**Métriques :**

- Lead time / Cycle time : délai entre le début et la validation d'une story (DoR → DoD)
- Stabilité du build : taux de builds verts/rouges par sprint
- Débit de défauts par sprint : nombre de bugs détectés / résolus par itération
- Taux de réussite des tests automatisés : couverture des tests d'acceptation et de régression
- Taux de stories "Done mais non validées" : alerte sur la dérive "done = dev only"
- Feedback loop : temps moyen entre commit et résultat de test (objectif < 15 min en CI/CD)


**Limites**

- Risque de confondre "Done" avec "Développé" : le test peut être négligé s'il n'est pas intégré dès la planification.
- Perte d'indépendance du testeur : en équipe intégrée, la validation peut manquer de recul critique.
- Pression temporelle élevée : le testeur doit équilibrer vitesse et rigueur.
- Risque de "test de surface" si le sprint est trop court pour des tests en profondeur.
- Nécessite une culture qualité partagée et une communication fluide pour fonctionner.

# AVIS D'EXPERT

## Une stratégie de test en Agile

Pour commencer je souhaite préciser que ce que je vais présenter peut servir dans un contexte de stratégie ou de Plan de test maître au sens ISTQB.

La différence se fait principalement sur le contenu avec plus de spécificités (notamment au niveau des risques) sur le plan de test maître… Usuellement appelé stratégie de test dans nos organisations.

### Le rôle d'une stratégie de test

On parle régulièrement de stratégie de test sans pour autant savoir quel est son réel intérêt.

Une stratégie de test est un document qui définit comment et pourquoi sont organisées les activités liées à la qualité d'un service numérique.

Ce document a de multiples intérêts comme :

- Centraliser les informations liées à la qualité
- Appréhender le contexte lié à l'organisation ou au service numérique
- Homogénéiser les pratiques

**Marc Hage Chahine**

Marc Hage Chahine (s'il faut encore le présenter) est une figure reconnue dans le domaine des tests logiciels et de la qualité logicielle en France.

- Il est co-fondateur / animateur du blog La Taverne du Testeur
- Il intervient comme orateur dans des conférences sur les tests, notamment pour présenter des sujets comme la qualité durable, l'automatisation, l'intelligence artificielle dans les tests… Ou comme la JFTL (Journée Française des Tests Logiciels)
- Il participe aussi à la conception d'outils ludiques ou pédagogiques dans le domaine du test, comme "Bugs End", un serious game lancé par K-Lagan avec Julien Cahu, pour promouvoir les compétences en test logiciel.

https://lataverneduteste ur.fr/2025/06/16/pourquoi-bugs-end/

- Cadrer les activités
- Donner du sens aux activités
- Former (je me sers souvent de ce document lors de l'intégration de nouvelles personnes) …

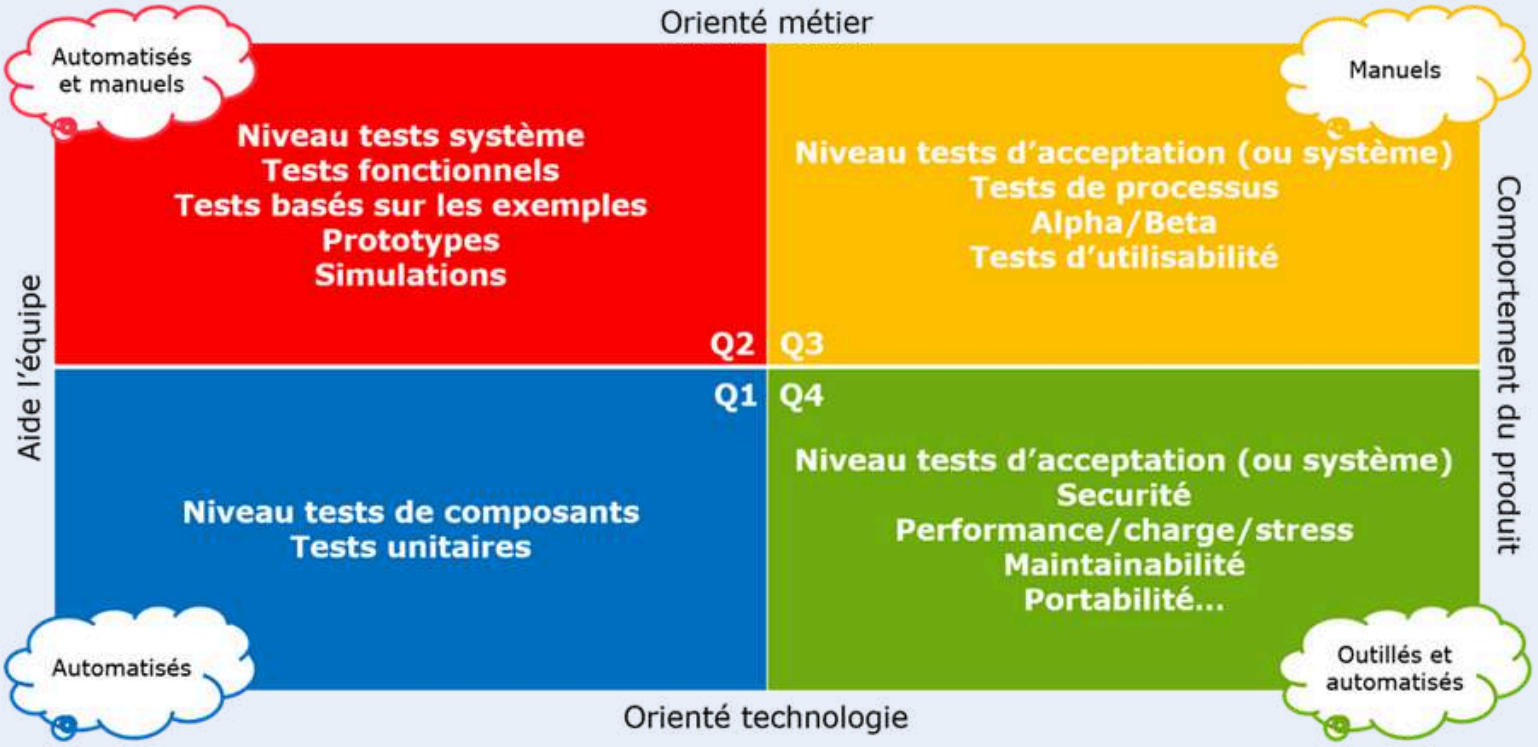
Il est vrai qu'il existe d'autres solutions qui permettent de répondre à ces besoins.

Néanmoins, la stratégie de test a le mérite de tout regrouper en un seul document. Cela s'avère très pratique à condition que ce dernier soit : connu, partagé, à jour et adopté par l'ensemble des acteurs intervenant sur le produit.

## Intérêt d'une stratégie de test en Agile

Lorsque l'on pense stratégie de test on pense souvent à un document lié au cycle en V avec différentes phases.

Le terme « Plan de niveau », faisant référence aux niveaux de test n'aide pas non plus. En effet, même si les niveaux de test ont toute leur place en Agile le terme « Niveau » ne fait pas Agile. En Agile on a tendance à penser aux Quadrants des tests.



Pour en revenir à la stratégie de test en Agile, l'intérêt de ce document est le même que dans les méthodologies de développement dîtes traditionnelles. Néanmoins le contexte Agile étant différent de celui d'un cycle en V il faut adapter la stratégie de test en conséquence.

**Adapter la stratégie de test à l'Agile**

Que l'on soit en Agile ou en Agile à l'échelle on travaille sur un produit. L'axe de travail est différent de celui d'un cycle en V.

Les éléments liés au cycle en V comme :

- Les phases de test et d'intervention de test
- Les kick off et autres réunions qui cadencent les projets
- Les plannings avec le contenu des release définis très en amont
- L'organisation en fonction des différents projets…

Tous ces éléments n'ont plus leur place.

Néanmoins, le rôle d'une stratégie de test n'est pas de répondre aux points mentionnés ci-dessus mais bien ceux énoncés dans la première partie de l'article. Parce qu'au final Agile ou pas il reste important de savoir :

- Pourquoi on teste
- Quels sont les tests à prioriser dans notre contexte
- Comment répartir les activités
- Quels sont les outils de test à notre disposition
- Qui fait quoi
- Comment sont organisés les tests…

Ces informations doivent être partagées et la stratégie de test permet justement de les regroupe en un point.

En cela un document de stratégie de test a toute sa place en Agile… Il faut cependant bien réfléchir à son contenu.

Voici un contenu qui peut servir d'exemple et/ou de template.

## Exemple de contenu d'une stratégie de test en Agile

Voici un exemple de contenu d'une stratégie de test. Il va de soi qu'il n'est pas directement applicable à votre contexte. Néanmoins il devrait vous aiguiller dans vos réflexions

**Une présentation du document**
Cette partie est là pour rappeler l'objectif du document.

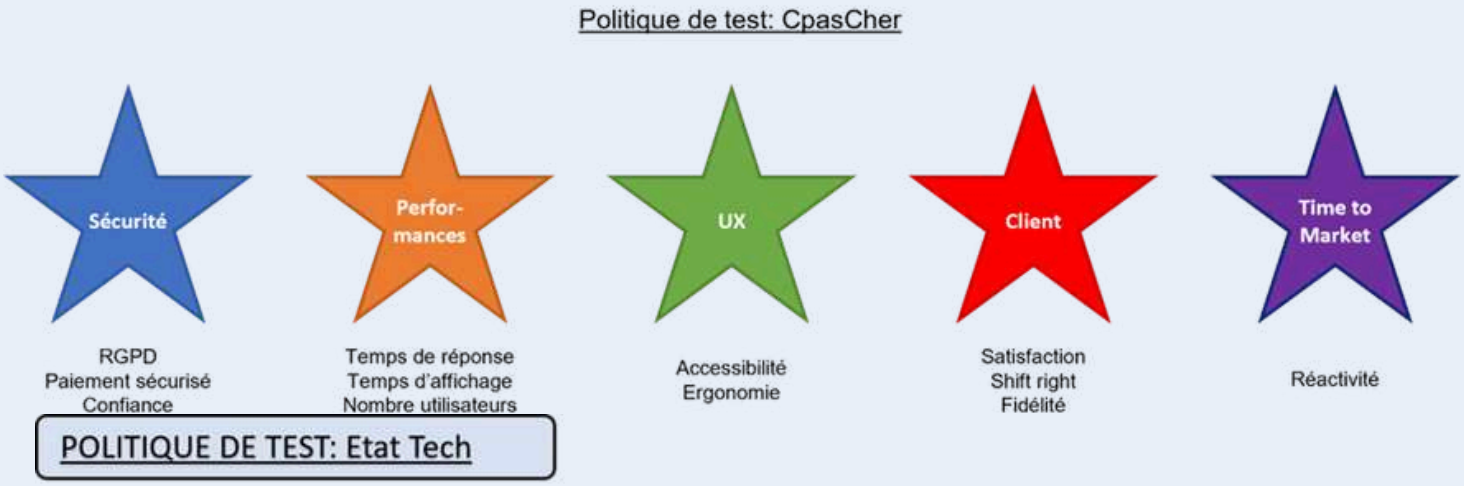
On peut également se servir de cette partie pour maintenir à jour l'historique des modifications apportées au document… Car oui, une stratégie de test se doit de vivre. Le contexte évolue, la stratégie de test doit évoluer avec.

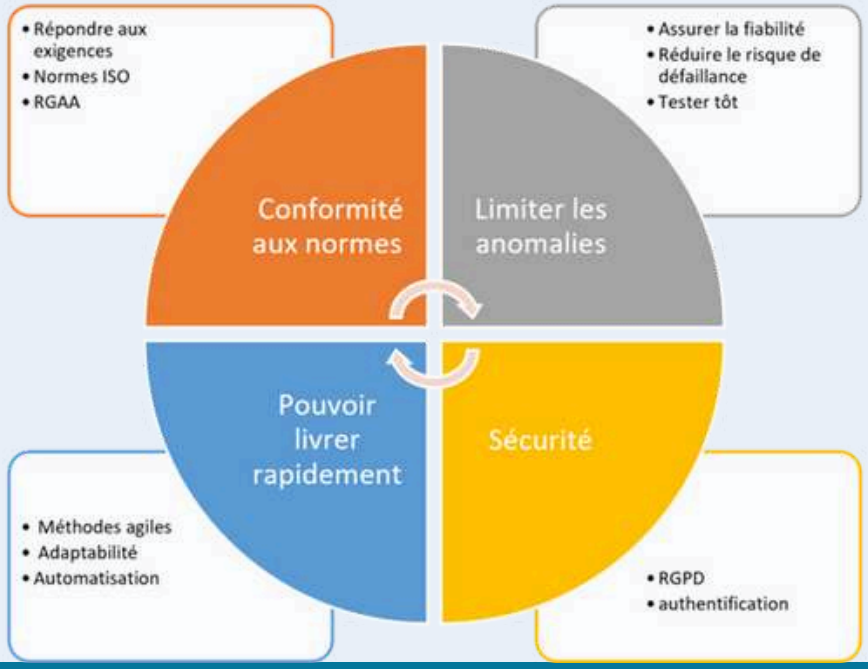
**La politique de test – Pourquoi teste-t-on ?**
Ce n'est pas forcément quelque chose que l'on trouve souvent mais c'est pour moi le point essentiel de toute stratégie et de tout travail lié à la qualité.

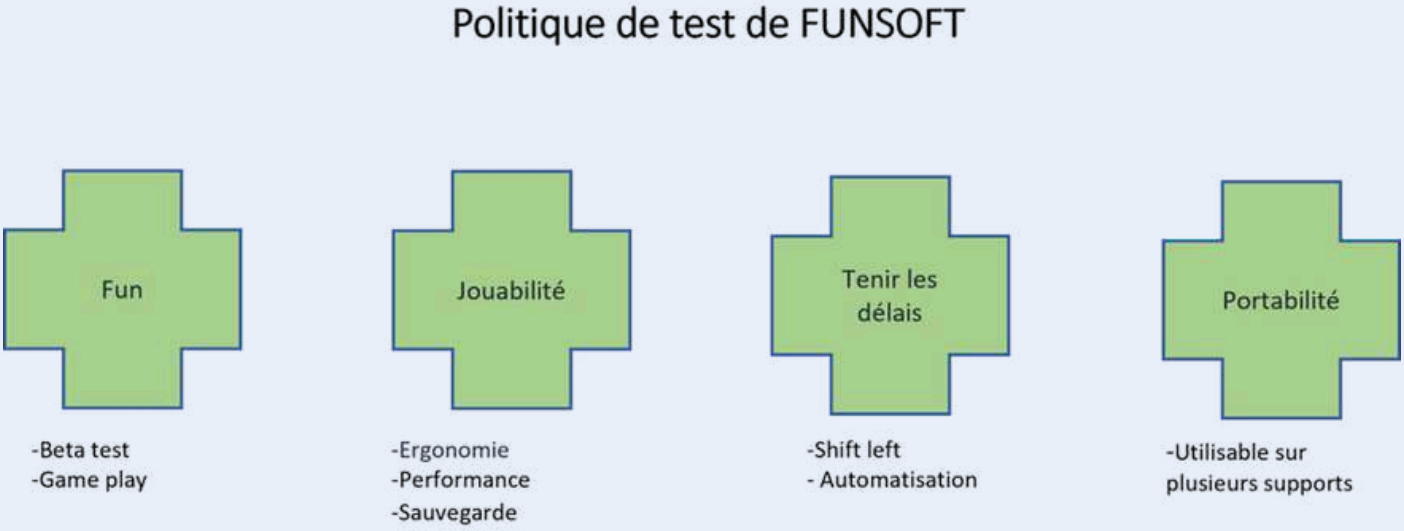
Il faut connaitre ses objectifs. Quelle qualité souhaitons-nous ? Quels sont les axes à privilégier ?

C'est les fondations de toute stratégie réussie. Et, selon les organisations, les raisons de tester sont drastiquement différentes ce qui fait que les tests à effectuer le seront également.

Ci-dessous, des résultats de politique de test que j'avais donné en exercice à des testeurs en reconversion :






Comme vous pouvez le constater, en voyant ces objectifs, on imagine tout de suite des approches et des techniques de test différents.

### Le contexte et présentation du produit

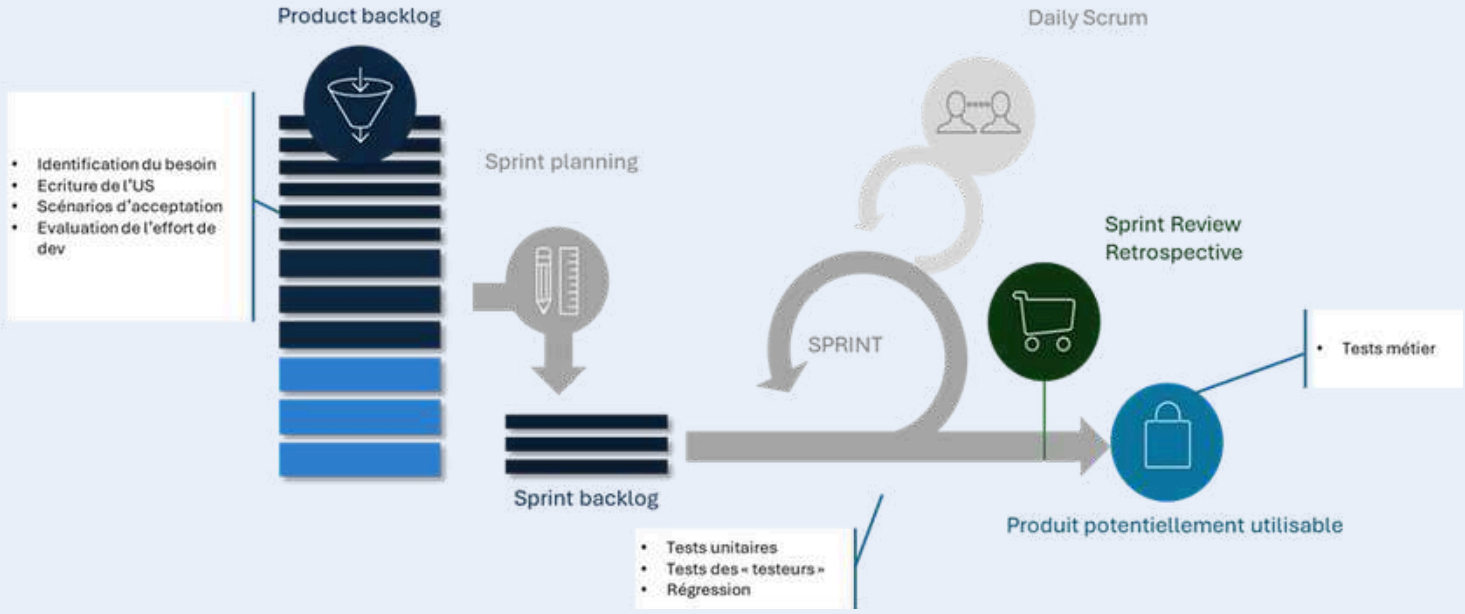
Cette partie est là pour présenter le produit, ses objectifs, ses utilisateurs et tout élément permettant de comprendre rapidement le contexte et l'environnement dans lequel le service numérique est développé.

### Les processus de test

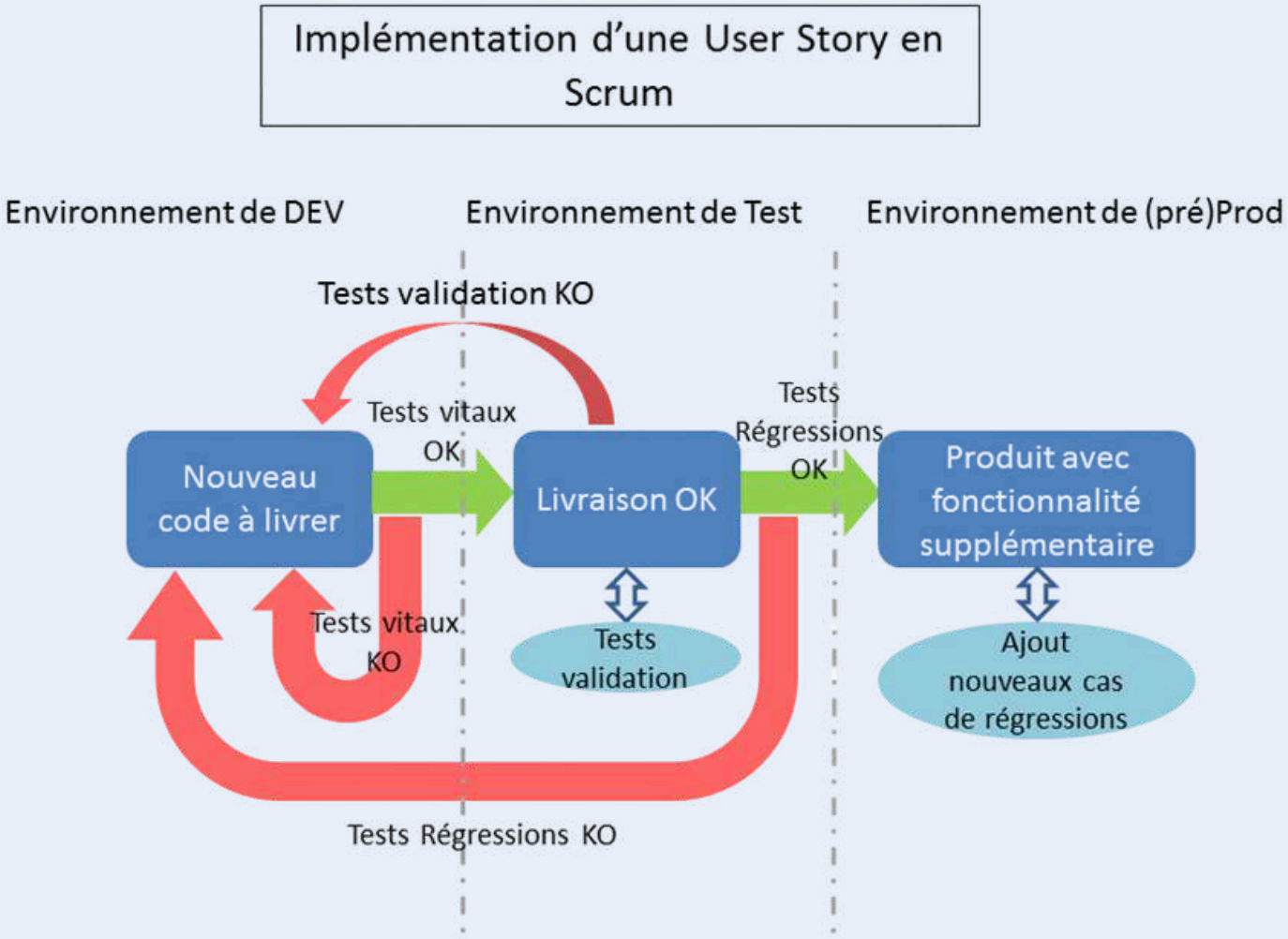
Que teste-t-on ?
Quand le teste-t-on ?

Il est essentiel de le définir.

Si l'on travaille en Scrum on peut avoir quelque chose qui ressemble à cela :



Ou encore quelque chose comme cela qui séquence des campagnes de tests pour valider une User Story :

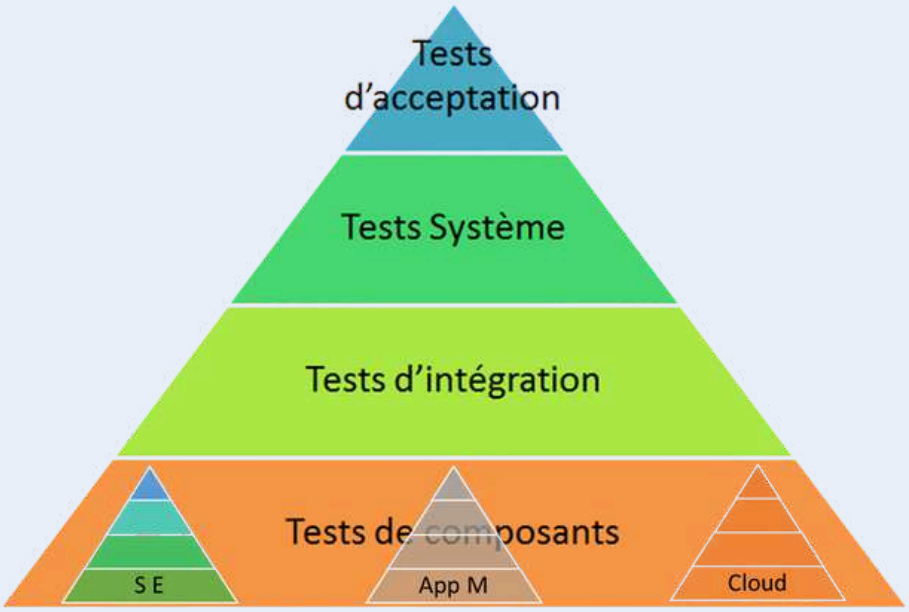


On peut aussi imaginer un tableau qui décrit les campagnes :

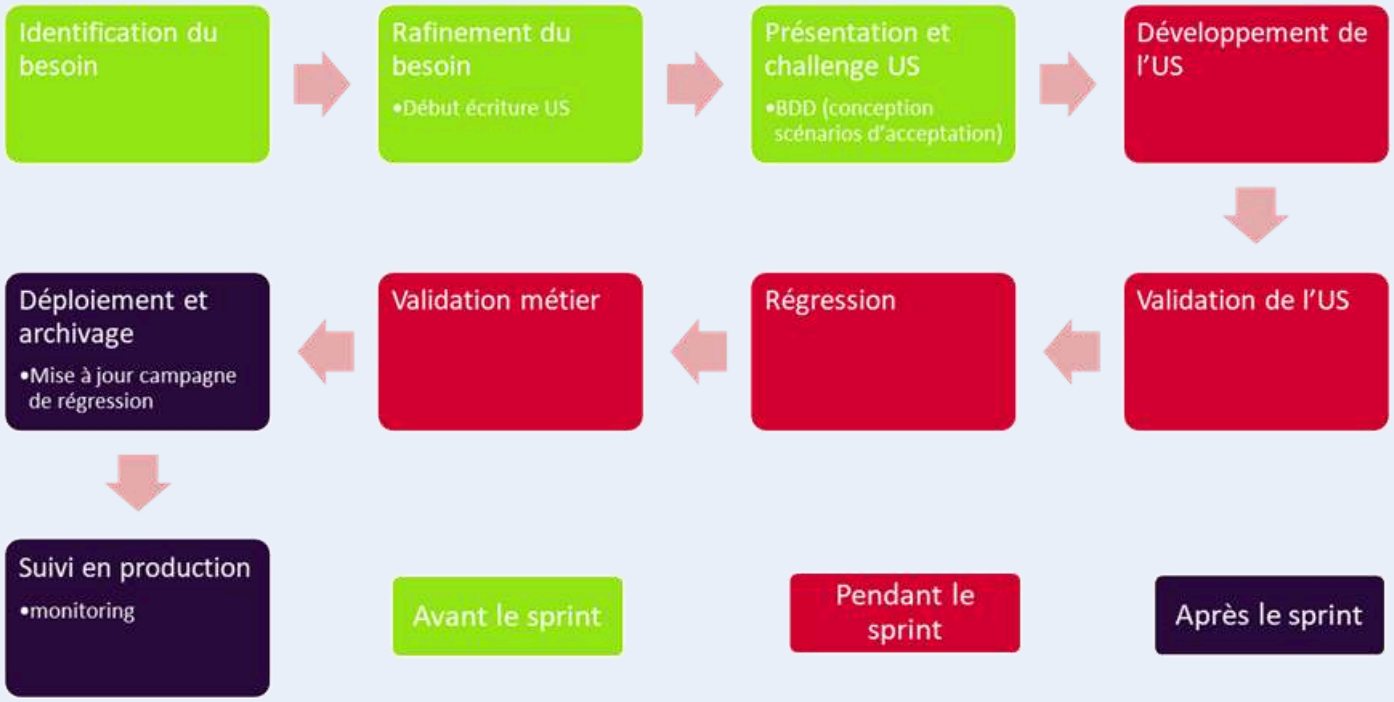
| | Tests vitaux | Régression | Validation | Acceptation |
|---|---|---|---|---|
| **Fréquence** | Très élevée (CI/CD) | Moyenne<br>1 fois par US | (très) faible<br>Une spécifique par US | (très) faible<br>Une spécifique par US |
| **But** | Pas de bug critique, l'application est "en vie" | Pas de régression majeure introduite dans le produit | Pas de bug majeur détectée<br>Valide le comportement de l'US | Vérifie que ce qui a été développé correspond au besoin |
| **Caractéristiques** | Courte peu sophistiquée ou variée | Plus sophistiquée que la campagne de tests vitaux<br>**Attention au paradoxe des pesticides**<br>Plusieurs phases possible | Varié<br>Vérification en profondeur | Exécution manuelle |
| **Périmètre** | Produit<br>Basé sur les « épiques » | Produit<br>Basé sur les « features » | US<br>Basée sur l'US en développement | US et produit<br>Validation du comportement de l'US dans une utilisation « produit » |
| **Contenu** | Tests unitaires (TU)<br>Qualité de code<br>Tests systèmes essentiels | Tests sur le comportement du produit (ex: tests issus du BDD) | Tests BDD<br>Tests exploratoires<br>Tests scriptés spécifiques à l'US | |
| **Automatisation** | 100% | Autant que possible (80%+) | Uniquement les futurs tests de régression (10%-) | 0% |
| **Durée** | Moins de 15 minutes | Moins de 2 heures | Moins de ½ journée (dépend de l'US) | Moins de 1 heure |
| **Evolution** | A chaque merge:<br>Nouveaux TU et TU mis à jour<br>Occasionnellement:<br>MaJ et ajouts tests essentiels | A chaque US ajoutée au produit:<br>MaJ tests de régression<br>Nouveaux tests de régression | Toujours différents avec les tests exploratoires<br>Spécifique à chaque US (BDD et tests scriptés liés) | Différents à chaque campagne |

Ceci n'est évidemment pas exhaustif, on peut avoir des parties spécifiques sur des campagnes spécifiques liées à de l'accessibilité, la sécurité ou tout autre sujet.

Dans le cas d'Agile à l'échelle on peut avoir d'autres propositions, basée sur des objectifs et qui montre l'imbrication des équipes come dans ce schéma que j'avais conçu en 2019 :



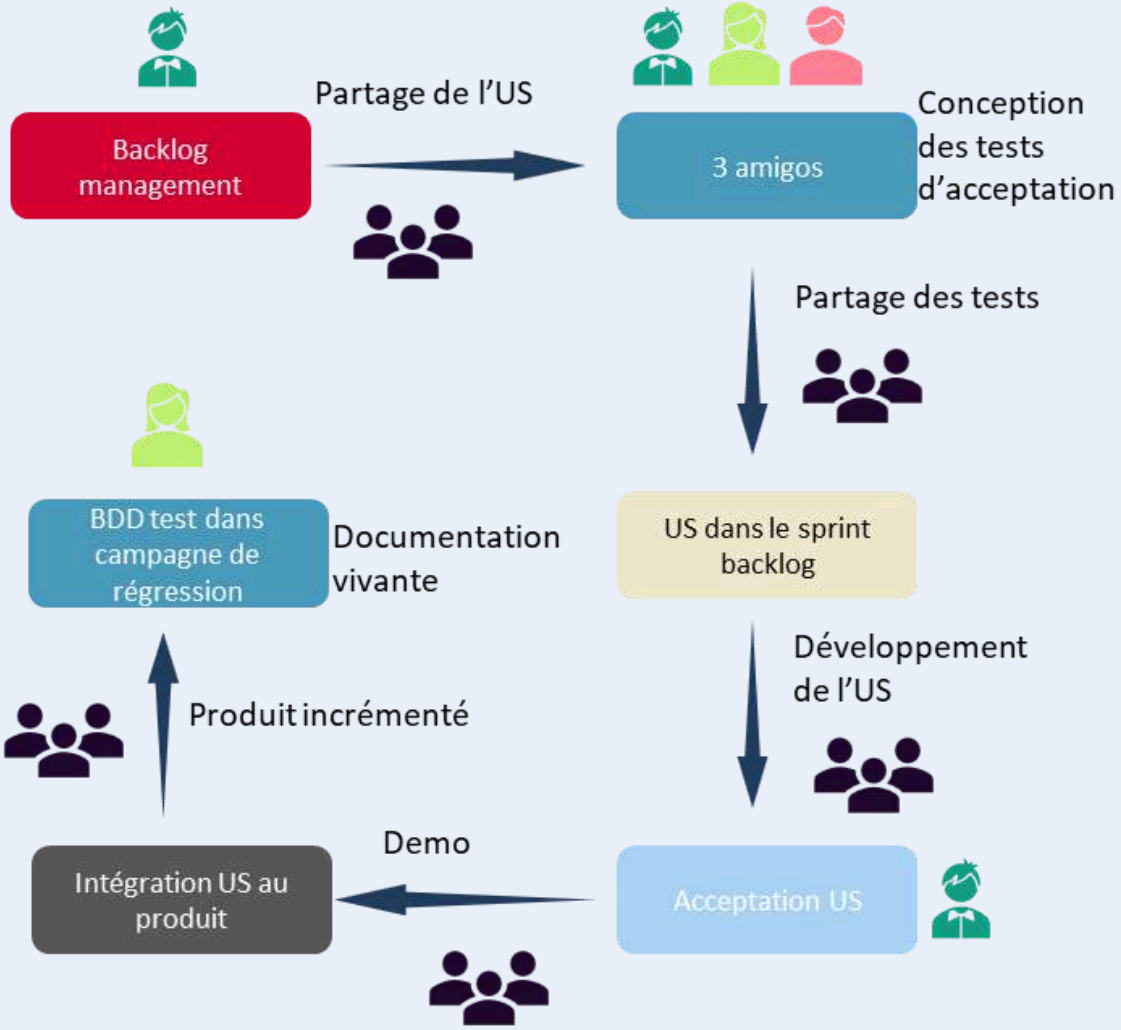

Dans cette partie on peut imagine la présence d'un schéma standard de validation des US :




## Les pratiques de test

C'est ici que l'on met les techniques utilisées.
On peut penser, en Agile à :

- du BDD




- L'organisation des exigences et la gestion de la régression en fonction
- Les différentes techniques de conception de tests
- Les couvertures recherchées
- Les éléments de testware attendus
- La mises en place de tests exploratoires avec un modèle de charte

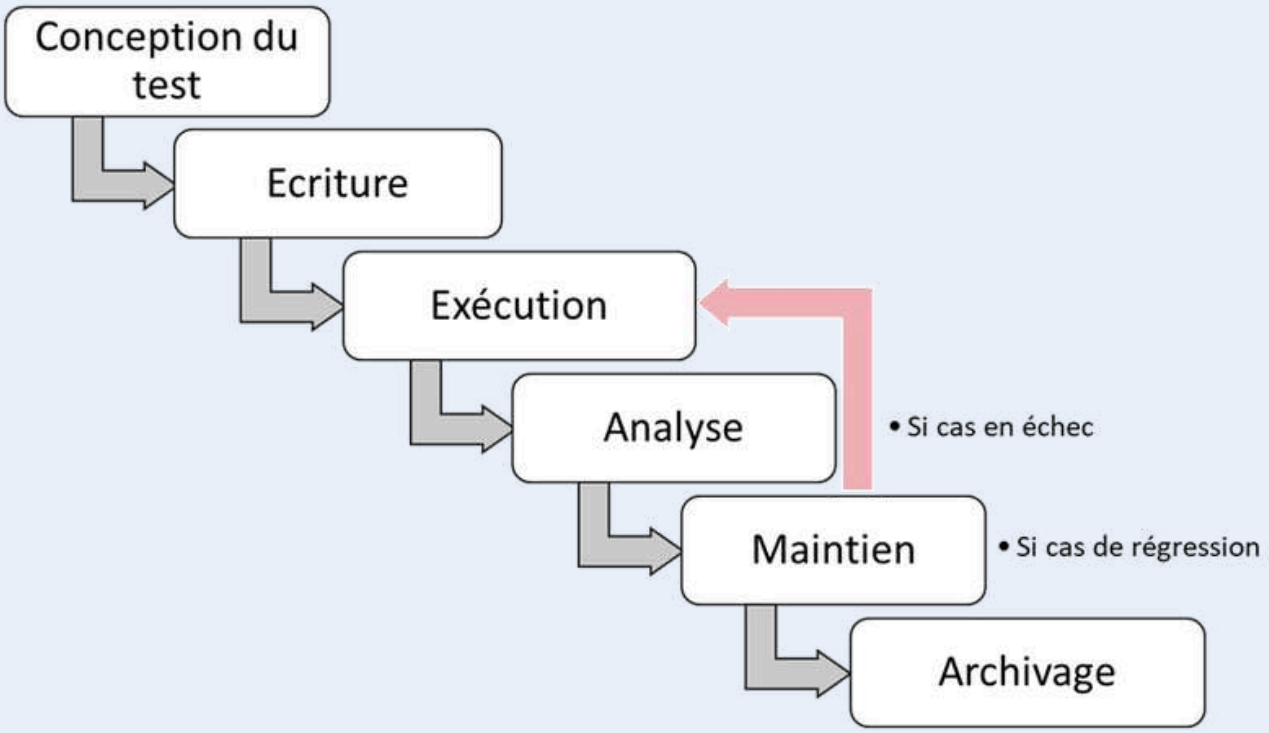
## La gestion des bugs

Que seraient le test et la qualité sans bug ?

Il est primordial de savoir bien gérer ces bugs.

Pour cela je trouve qu'il est intéressant d'avoir dans sa stratégie un Workflow des bugs mais aussi un template de fiche de bugs.

Exemple de workflow basique :




## Les rôles et responsabilités

Je tiens à rappeler que oui, en Agile, l'équipe est globalement responsable de ce qu'elle livre et donc de la qualité de ses déploiements.

Néanmoins, je ne me suis encore jamais retrouvé dans un contexte qui permette de mettre simplement « équipe » dans les rôles et responsabilités.

De plus, un Service numérique n'est rarement le fruit d'une seule équipe isolée.
Voici ce qu'un tableau de rôle et responsabilité peut donner :

| Nom | Rôle principal | Spécificités |
|---|---|---|
| **Testeur** | Coach test/qualité dans les équipes agiles | • Membre de l'équipe Agile<br>• Met en œuvre la stratégie de test et l'adapte à l'équipe<br>• Membre de l'équipe de test |
| **Dev** | Développement des US | • Membre de l'équipe Agile<br>• Gestion des tests Q1<br>• Gestion de la CI/CD |
| **Test Manager** | Référent méthodologique test | • Référent méthodologique des testeurs<br>• Définit la stratégie de test sur les produits<br>• Peut être intégré à une équipe Agile |
| **Automaticien** | Implémente l'automate de test | • Initie et délivre l'automate de test à l'équipe Agile<br>• Support à l'équipe Agile sur l'automatisation |
| **Expert test** | Aide ponctuelle à la demande<br>Définit la stratégie de test globale | • Référent des test lead / test manager<br>• Formation et animation de la communauté |
| **Product Owner** | Assure le lien entre le métier et l'équipe Agile | • Représentant métier dans l'équipe Agile<br>• Validation finale des US et responsable de la campagne d'acceptation |

### La gestion de la connaissance

La manière dont on gère les connaissances du produit mais aussi les formations proposées et nécessaires ainsi que les référents.

### La stratégie d'automatisation (souvent un document spécifique dont on fait juste mention)

Donne le cadre de l'automatisation sur le produit ou au sein de l'organisation.

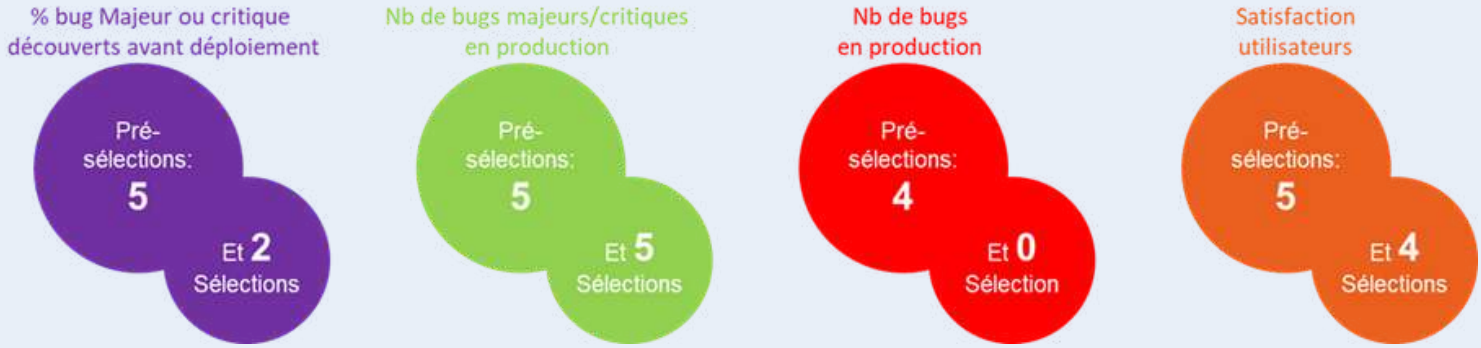
### L'organisation du centre de test (pour les grandes organisations uniquement)

Comment les différents testeurs interagissent entre eux et se servent de leurs expériences respectives même s'ils ne sont pas dans la même équipe Agile.

### Les indicateurs

La liste des indicateurs utilisés.
Leurs description et calculs.
Le nombre d'indicateurs possible est extrêmement important. Il est important de se poser et de bien les sélectionner puis de bien rappeler leur objectif pour s'assurer qu'ils continuent à répondre à ce dernier tout au long de leur utilisation.

D'ailleurs, il est intéressant de voir que si l'on se penche sur le sujet, certains indicateurs « usuels » pris par défaut n'ont que peu de valeur s'ils sont seuls. Je pense notamment au nombre de bugs ou encore un % de tests automatisés :




### Les outils

Donne les outils utilisés dans le cadre des activités liés au test et à la qualité. Cela englobe aussi des outils non spécifiques au test comme les outils de suivi du développement, potentiellement certains outils de gestion de versions…

On peut également proposer les manières d'utiliser les outils avec des modes opératoires.

### La stratégie de test : un document vivant

Après avoir fait tout ce travail de centralisation, avoir réfléchi au pourquoi et au comment on est généralement assez fier de son résultat et on présente sa stratégie en grande pompe.

En général, les équipes s'en servent beaucoup au début pour initialiser leurs processus.

Ce document est ensuite trop souvent mis de côté et non maintenu.

J'observe souvent ces éléments qui sont un signe de manque de maintenance de la stratégie de test :

- Les équipes font évoluer leurs pratiques et ne mettent pas à jour la stratégie.

La stratégie se trouve alors en décalage avec la réalité et il devient très compliqué de pouvoir s'y fier.

On arrive d'ailleurs assez rapidement à un point où le document n'est plus présenté et les nouveaux arrivants ne connaissent même pas son existence.

- Le contexte évolue… Mais pas le document

Dans ce cas on peut arriver au premier point avec des équipes qui doivent s'adapter ou à une situation potentiellement plus problématique : les équipes continuent à s'appuyer sur la stratégie qui n'est plus adaptée. Elles peuvent alors se retrouver à faire des tests peu utiles tout en omettant de faire des tests nécessaires.

Le document va alors à l'encontre de son rôle initial en empêchant d'avoir des processus liés à la qualité efficaces et efficients.

Enfin, je ne pouvais pas parler de stratégie de test Agile sans rappeler que ce document, encore plus que dans les cycles traditionnels, doit être connu, compris et adopté par tous les membres des différentes équipes.

Tout le monde contribue à la qualité, tout le monde est concerné par la stratégie de test… Et ce encore plus en Agile où chaque membre de l'équipe est appelé à faire du « dépassement de fonction » et d'aider l'équipe autant qu'il le peut à chaque instant.

# CAS CONCRET


## STRATEGIE DE TEST DE PERFORMANCE

### Contexte

Nous avons entrepris une refonte complète de notre plateforme digitale car l'objectif est de répondre aux enjeux de disponibilité et de scalabilité lors des périodes critiques telles que le Black Friday, les soldes saisonnières et les campagnes marketing massives.

L'ancien système monolithique montrait ses limites, notamment par des lenteurs de navigation et des interruptions de service en période de forte affluence.

Le nouveau système, basé sur une architecture microservices et hébergé dans le cloud, doit garantir une expérience utilisateur fluide, stable et compétitive.

### Description de l'application

La plateforme repose sur un frontend développé en ReactJS, déployé sur un CDN pour maximiser la rapidité d'affichage. Le backend est constitué de microservices Java/Spring Boot communiquant entre eux via des API REST et GraphQL. Les données sont stockées dans une base PostgreSQL en haute disponibilité, avec Redis utilisé pour le cache. L'ensemble est orchestré par un cluster Kubernetes déployé sur Google Cloud Platform.

Les parcours utilisateurs considérés comme critiques incluent la consultation du catalogue, l'ajout d'articles au panier, le passage de commande et le paiement, ainsi que le suivi des commandes. Ces parcours représentent la valeur business principale et nécessitent une attention particulière lors des tests de performance.

### Parties prenantes

Plusieurs parties prenantes sont impliquées dans ce projet

La direction technique attend que la nouvelle solution soit disponible et scalable en toute circonstance.

Les équipes marketing souhaitent garantir un parcours utilisateur fluide afin de maximiser le taux de conversion, particulièrement lors des campagnes à fort trafic.

L'équipe QA, en charge des tests de performance, doit démontrer la capacité du système à tenir la charge et fournir des recommandations d'optimisation.

Enfin, l'équipe DevOps assure la supervision et l'automatisation de la scalabilité dans le cloud.

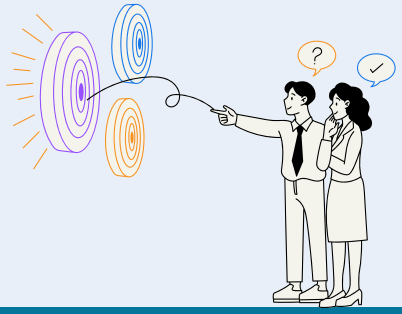
## Criticité de la performance

La performance est directement liée au chiffre d'affaires de ShopNow. Un ralentissement de deux secondes sur les pages critiques peut générer une baisse de conversion estimée à 15 %. Chaque minute d'indisponibilité lors d'un pic de trafic entraîne une perte estimée à 25 000 €. Dans un secteur hautement concurrentiel, où les acteurs majeurs offrent déjà des temps de réponse inférieurs à une seconde, il est impératif de se hisser à ce niveau pour rester compétitif.

Le taux de conversion correspond au pourcentage de visiteurs qui accomplissent une action attendue sur un site ou une application. Il est donc directement lié au retour sur investissement (ROI) de votre site web. En effet, plus le taux de conversion est élevé, plus le nombre de clients ou prospect est important, et donc plus votre chiffre d'affaires est élevé

## Objectifs de performance et SLA

Les objectifs fixés pour cette campagne de tests sont les suivants : un temps de réponse moyen inférieur à 1,5 seconde pour les pages catalogue et inférieur à 1 seconde pour la page de paiement. Le débit doit atteindre 200 transactions par seconde lors des pics de charge, avec une disponibilité garantie de 99,95 %. Le taux d'erreur toléré est fixé à moins de 1 % des requêtes.



## Indicateur de performance

Les indicateurs suivis incluront le nombre de transactions par seconde, les temps de réponse moyens et au 95e percentile, le taux d'erreurs HTTP, l'utilisation des ressources système (CPU, RAM, I/O), le temps de réponse des requêtes en base de données, ainsi que le comportement du mécanisme d'auto-scaling de Kubernetes.

## Périmètre des tests

Les tests couvriront les parcours critiques (catalogue, panier, paiement), les API internes et externes, ainsi que la capacité du cluster Kubernetes à gérer la montée en charge. Les tests d'endurance porteront sur huit heures continues afin de vérifier la stabilité. En revanche, les tests mobiles natifs et les scénarios non-critiques tels que les outils d'administration interne ne font pas partie du périmètre immédiat.

## Modèle de charge et scénarios d'usage

Le modèle de charge simule jusqu'à 100 000 utilisateurs simultanés. La répartition des scénarios est la suivante : 60 % de navigation sur le catalogue, 25 % d'ajout au panier, 10 % de paiement et 5 % de suivi de commande.

Deux variantes de charge seront testées : une journée normale avec 10 000 utilisateurs et un scénario Black Friday multipliant ce volume par dix.

## Données de test

Les données utilisées comprendront un million de produits issus d'un jeu anonymisé de la production. Elles incluent les informations relatives aux produits, aux utilisateurs et aux commandes. Une distribution aléatoire sera appliquée pour simuler le comportement réel. Un rafraîchissement quotidien des données est prévu afin de garantir leur cohérence.

## Environnement de test

L'environnement de test est une réplique de la production à 80 % des ressources. Il intègre un frontend ReactJS sur CDN, des microservices Java orchestrés sur Kubernetes, une base PostgreSQL en haute disponibilité et un cache Redis. Les conditions réseau sont simulées avec une latence moyenne de 50 ms en Europe et de 150 ms depuis les États-Unis.

## Outils et méthodes

Les outils sélectionnés pour les tests de charge sont Gatling et K6, avec injection cloud pour simuler un trafic réaliste. Le monitoring sera assuré par Dynatrace, Grafana et la stack ELK pour l'analyse des logs. Les méthodologies appliquées incluront les tests de charge pour établir une baseline, les stress tests pour identifier les limites, les tests d'endurance et les tests de montée en charge progressive.

## Critères d'entrée

Les tests ne pourront débuter qu'une fois l'application stabilisée, sans bugs bloquants identifiés. Les jeux de données devront être disponibles et validés, l'environnement de test opérationnel et les solutions de monitoring actives.

## Critères de sortie

La campagne de tests sera considérée comme terminée si les SLA sont atteints sur 95 % des scénarios, si la stabilité est confirmée en endurance, si aucune fuite mémoire n'est observée et si la documentation complète des résultats est fournie.

## Risques, hypothèses et dépendances

Les principaux risques concernent la différence potentielle entre l'environnement de test et la production, ainsi que le budget cloud limité pour l'exécution des tests massifs. Les hypothèses reposent sur une répartition homogène du trafic et sur des scénarios réalistes. Les dépendances identifiées incluent notamment l'API de paiement externe et les services de livraison tiers.

## Planification et ressources

Les données utilisées comprendront un La campagne de tests s'étalera sur quatre semaines : une semaine pour la préparation et le développement des scripts, deux semaines pour l'exécution itérative et une semaine pour l'analyse et la production du rapport final. Les rôles impliqués incluent un test manager, un performance engineer, un DevOps et un DBA.

## Rapports et communication

Les résultats seront suivis en temps réel via les dashboards Grafana et Dynatrace. Des rapports hebdomadaires synthétiques seront diffusés aux parties prenantes, complétés par un rapport final détaillé en fin de campagne. Les destinataires incluent le CTO, l'équipe QA, les DevOps et le département marketing.

## Processus de correction et optimisation

Les anomalies seront enregistrées dans Jira, priorisées selon leur impact business et leur sévérité.
Les correctifs appliqués par l'équipe de développement feront l'objet de retests systématiques. Cette boucle de tuning se poursuivra jusqu'à obtention de résultats conformes aux attentes.

## Revue et validations

Un tableau de validation accompagnera le document, recensant les approbations du CTO, du QA Manager, du responsable DevOps et du chef de projet côté métier. Chaque validation sera datée et assortie de commentaires si nécessaire.

# OUTIL

# CLOUDNETCARE

# CLOUDNETCARE

## Présentation de la plateforme

CloudNetCare est une plateforme SaaS **100% française**. Elle a été créer en 2009 par les fondateurs que nous verrons page suivante.

Elle propose trois offres complémentaires :

- Tests fonctionnels
- Tests de montée en charge
- Monitoring UX.

Née d'une rencontre en mai 2009, CloudNetCare passe, dès 2010, de l'assistance informatique à l'édition d'une plateforme SaaS de test.

L'idée fondatrice est de fournir aux équipes un outil unifié pour vérifier la non-régression, éprouver la performance sous charge et surveiller en continu l'UX (sans empiler licences et expertises rares).

Développée par une R&D en France, la plateforme s'est consolidée autour de scénarios réutilisables qui s'exécutent sur navigateurs réels, avec données injectées, re-run auto, diff visuel, rapports et intégration CI.

01

**2009**

Rencontre des fondateurs, idée d'une plateforme de test "cloud-first".

02

**2011**

Première présentation au forum de Bargento qui a permis de trouver des gros clients

**Fermes mobiles physiques hébergées en France, via OVHcloud**
**Fermes de VM et navigateurs en France, via Microsoft Azure**

**Mais les deux offrent la possibilité d'exécuter des tests ailleurs comme en Asie. Ce fut le cas pour l'un de leurs clients.**

# MECANISME ET COMPATIBILITE

CloudNetCare repose sur un principe d'automatisation « sans code », où la logique métier du test est reproduite à travers une interface intuitive.

Ici, aucune compétence en langage de programmation n'est requise : les parcours de test sont définis sous forme de séquences d'actions qui imitent fidèlement le comportement d'un testeur humain.

Ces parcours s'appuient sur des sélecteurs (par exemple des XPath ou des identifiants CSS) et sur un ensemble de commandes prédéfinies représentant les interactions les plus courantes :

- Se positionner sur un champ de formulaire
- Renseigner une valeur
- Cliquer sur un bouton ou un lien
- Vérifier la présence ou le contenu d'un élément à l'écran
- Comparer le résultat obtenu à la valeur attendue

Chaque étape du parcours est ainsi traduite en instructions compréhensibles par la plateforme, qui les exécute sur différents environnements ou navigateurs.

Cette approche permet à un testeur fonctionnel, même sans compétences techniques, de concevoir et d'automatiser des scénarios complets tout en gardant une maîtrise fine des validations.

### Réversibilité et Portabilité

Un des points forts de CloudNetCare réside dans la réversibilité de ses parcours.

Contrairement à d'autres outils « no-code » souvent fermés, les scénarios créés sur la plateforme restent exploitables en dehors de celle-ci. À tout moment, l'utilisateur peut exporter ses tests pour les exécuter dans un autre cadre, sans dépendance technique à CloudNetCare.
Les exports proposés couvrent plusieurs formats standards :

- Selenium (.json) : pour réutiliser le scénario dans des environnements Selenium IDE ou l'intégrer dans des pipelines CI/CD existants.
- Cypress (.js) : pour bénéficier de la puissance du framework JavaScript et enrichir le script avec des assertions, des hooks ou des intégrations spécifiques.

**Cette portabilité ouvre la porte à une véritable stratégie hybride : commencer par prototyper et stabiliser les parcours via l'interface CloudNetCare, puis les intégrer progressivement dans une suite d'automatisation plus complète au sein du pipeline d'intégration continue.**
**Ainsi, CloudNetCare ne se positionne pas comme une alternative fermée, mais plutôt comme un accélérateur de mise en place qui s'intègre harmonieusement dans un écosystème de test existant.**

# COMMENT LE METTRE EN PLACE ?

Deux options s'offrent à vous :

1. Vous pouvez gérer, créer, exécuter vos tests en toute autonomie

2. Confier toute la gestion à l'équipe de CloudNetCare (certifiée ISTQB) - c'est eux qui vont mettre en place vos tests et planifier les exécutions avec votre aide (cas de test, accès aux sites/applications à tester, ...).
Vous pouvez également gérer, créer et exécuter vos tests à tout moment.

# PRESENTATION DE L'INTERFACE


## Présentation de l'inteface

Voyons maintenant ensemble tout ce que CloudNetCare peut vous apporter et en quoi cette solution est différente des autres.

Elle est disponible à 100% dans le cloud. Cette seule interface permet de simuler les sites internet, les applications web et les applications mobiles.

L'interface se présente comme cela :

## Présentation de l'interface

Dès l'ouverture, la page **d'accueil** met à disposition une bibliothèque de vidéos didactiques.

Ces tutoriels couvrent l'ensemble des usages possibles : création de parcours Web ou mobiles, intégration de Selenium, utilisation de Jenkins, gestion des plans de tests ou encore mise en place de non-régressions graphiques.

On y trouve également un bouton permettant de lancer immédiatement des parcours et tests d'exemple.

La **section parcours** permet de créer et modifier les scénarios qui simulent le comportement des utilisateurs :

- enregistrement d'un parcours Web via le Web Recorder
- intégration de scripts issus de Selenium IDE
- configuration de parcours sur mobile

Nous verrons ces cas en détails dans les cas concrets.

L'espace **tests automatisés** regroupe les fonctionnalités destinées à l'organisation et à l'exécution :

- Vue d'ensemble : état global des campagnes et résultats.
- Plans de tests : regroupement des cas par fonctionnalité, release ou sprint
- Plans d'exécution : préparation des suites de tests à lancer
- Historique : traçabilité et consultation des exécutions passées
- Bug Tracking : suivi des anomalies identifiées
- Masques (Visual testing) : gestion des zones dynamiques pour fiabiliser les comparaisons visuelles

Enfin, le bandeau supérieur permet de gérer son profil et son compte, tandis que l'ensemble de l'interface favorise une navigation intuitive : on passe facilement de la création de scénarios à leur exécution, puis au suivi des résultats et des anomalies.

# CAS CONCRET

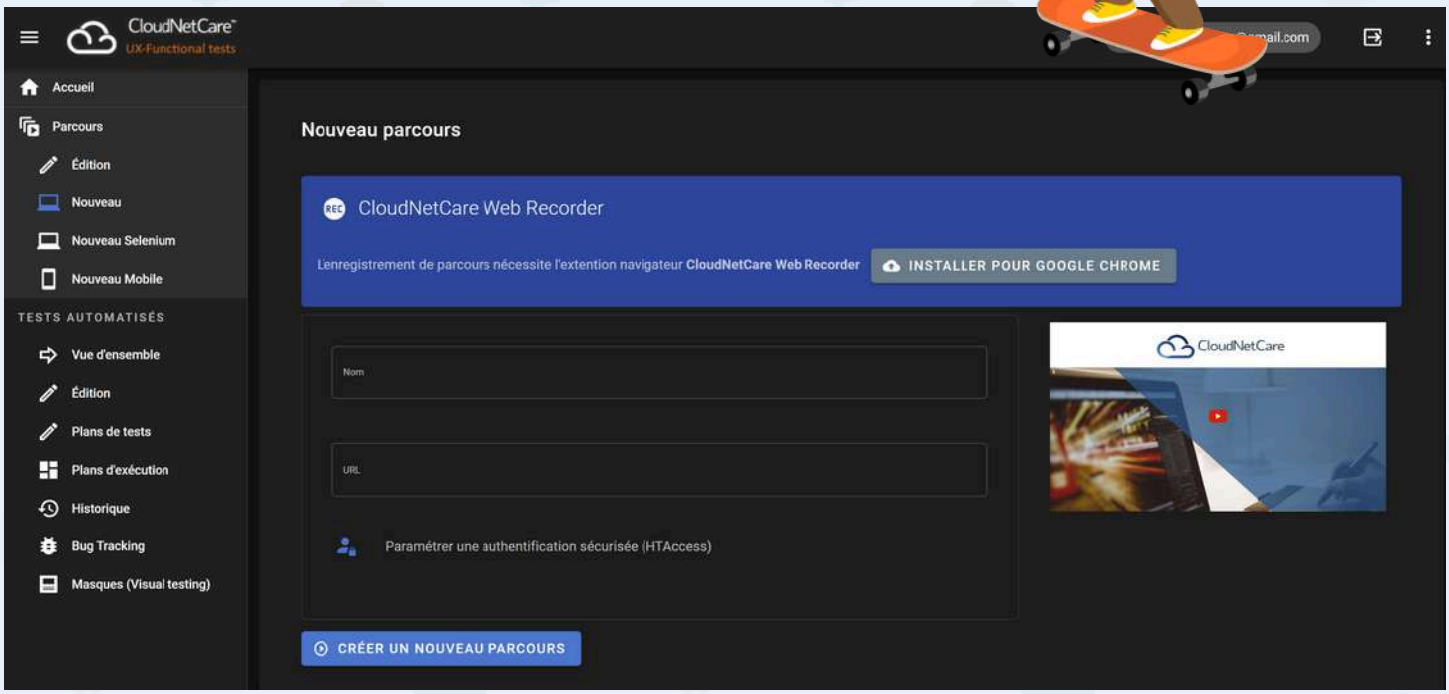

## SITE D'ECOMMERCE

Créons ensemble un cas concret d'un e-commerce : https://skateboardshop.fr.

Créons tout d'abord nos scénarios avec le web recorder.

En premier, il faut vous connecter : ouvrez CloudNetCare, se connecter avec son compte utilisateur.

Ensuite, cliquez sur la section parcours dans le menu latéral puis sur "Nouveau" :



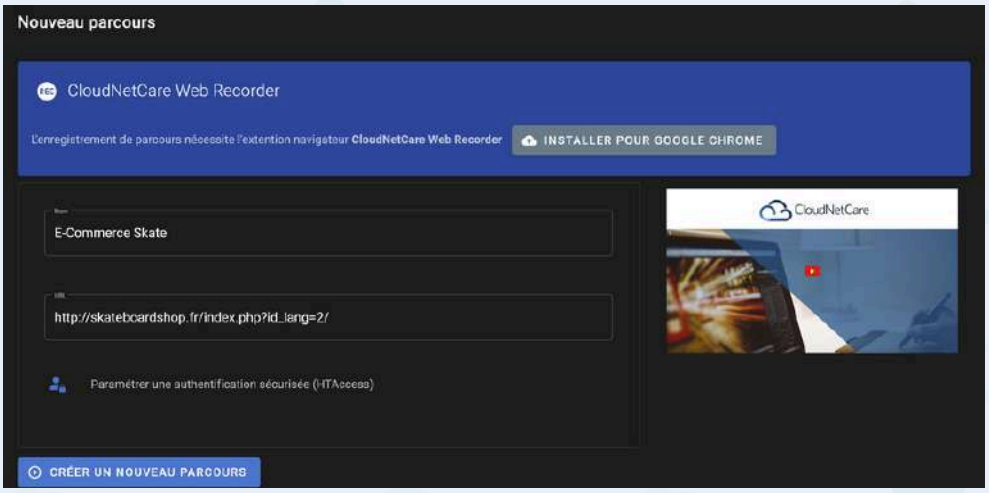

Cliquez sur installer pour google chrome (cela installera le web recorder). A noter que dans mon cas, j'ai google chrome.


Cliquez sur ajouter à google chrome puis sur ajouter l'extension.

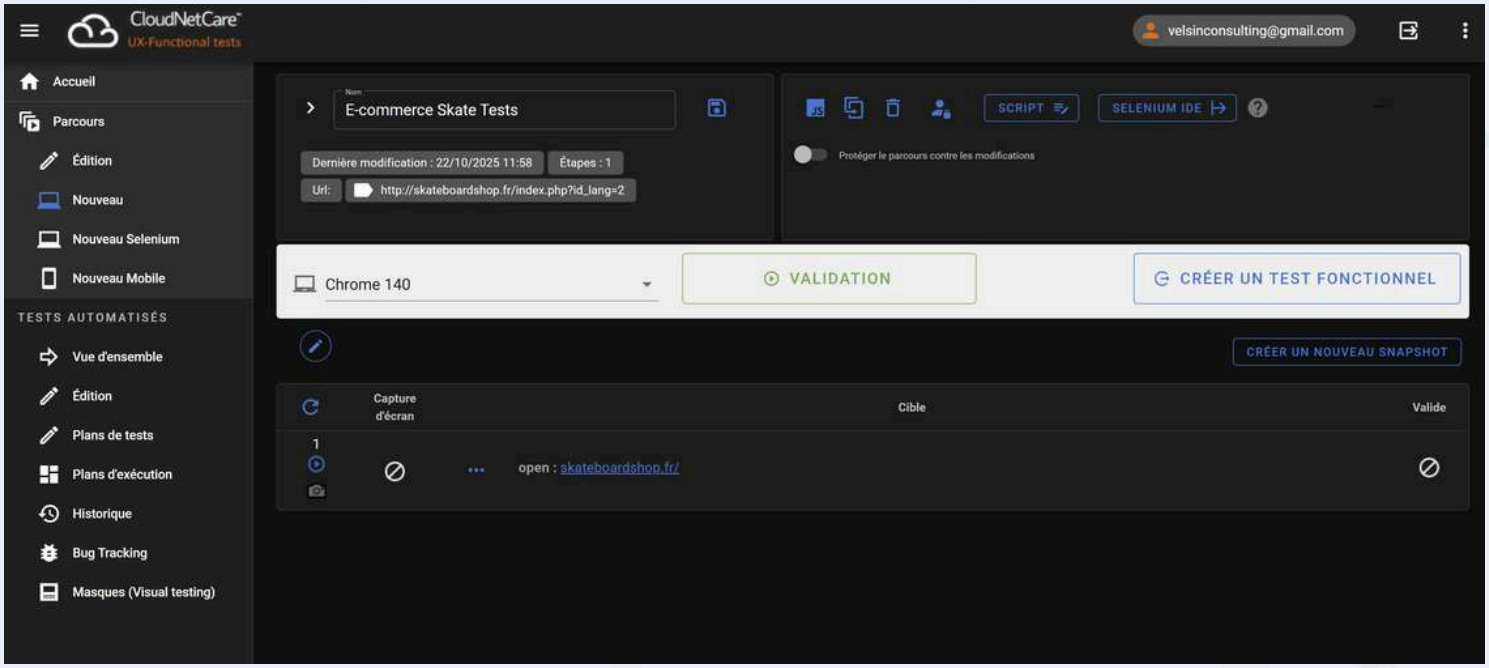
**L'extension ne collecte et n'utilise pas vos données en dehors de l'utilisation des interactions pour CloudNetCare.**

Ensuite, remplissez le nom et l'URL de l'application que vous voulez tester. Dans mon cas, j'ai mis "E-Commerce Skate" comme nom et l'url : http://skateboardshop.fr/index.php?id_lang=2/

Puis cliquez sur créer un nouveau parcours.

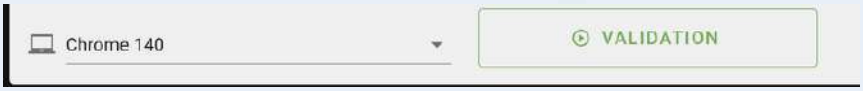



Puis cliquez sur créer un nouveau parcours.



Vous retrouvez votre nouveau parcours. Pour le moment, on voit que l'on ne fait que ouvrir le lien vers notre site :


Vous pouvez choisir votre navigateur sur lequel exécuter les tests (cf à droite)

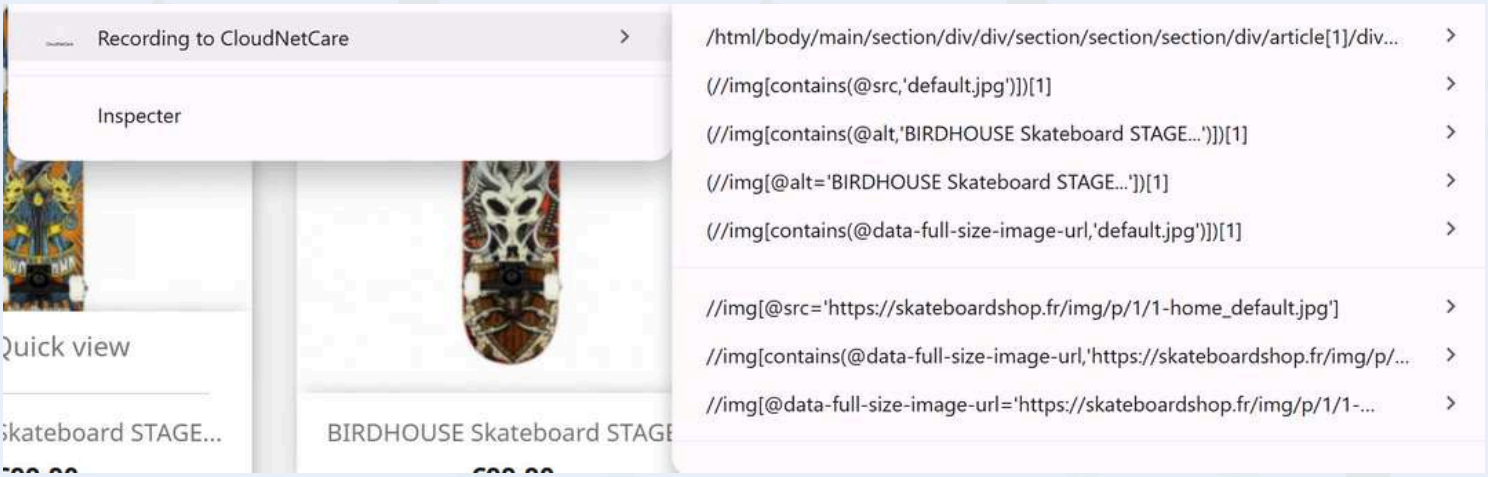



Pour ajouter une nouvelle étape, il faut survoler l'étape d'ouverture et cliquer sur le plus. Vous pouvez également éditer l'étape, la dupliquer, copier dans le presse-papier et la supprimer


Après avoir installé l'extension Chrome, vous devez avoir une icone en plus pour enregistrer.


Cliquez dessus.

Le navigateur ouvre la page du site Skateboard shop. Si vous faites un clic droit sur le premier skate board, vous allez avoir la proposition d'enregistrement :



Chaque action effectuée sur le site (clic, saisie, navigation, survol, validation, etc.) est traduite en étapes de test automatisables avec un XPath ou un sélecteur CSS associé.

Cela permet à la fois :

- de gagner du temps dans la création de tests,
- de réduire les erreurs humaines dans la rédaction des XPaths,
- et de maintenir la cohérence entre les tests front et les objets de la page.

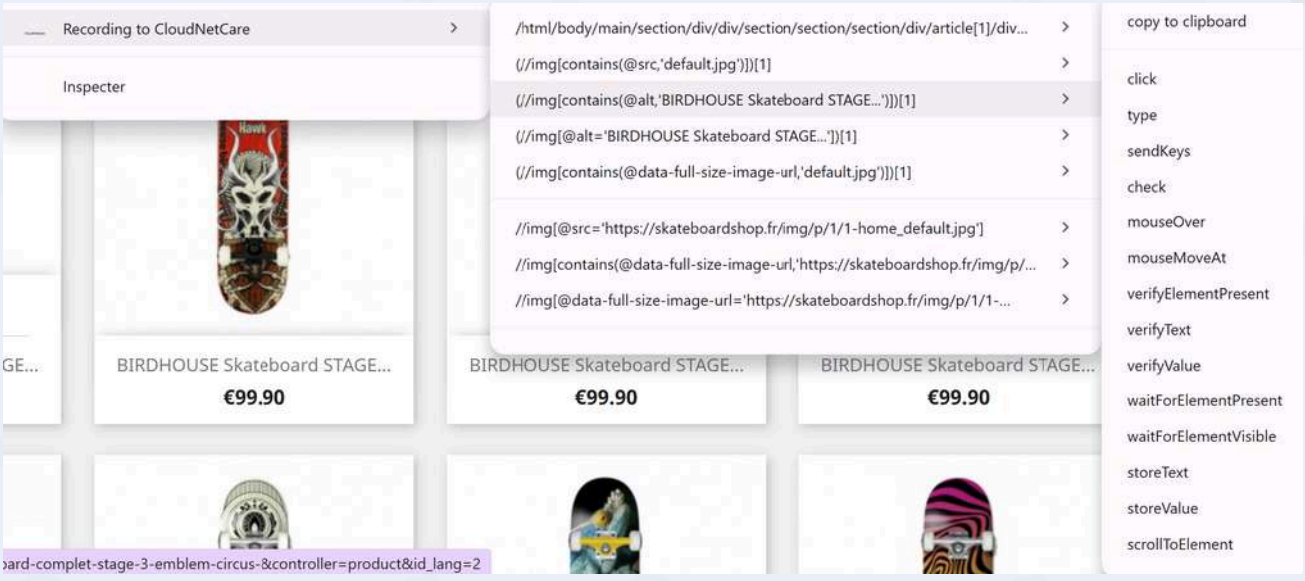
Ici, l'image du skateboard est détectée avec plusieurs XPaths possibles.

Selon la stratégie d'automatisation, on peut :

- cibler l'attribut alt si son contenu est stable,
- cibler l'attribut src si le chemin d'image reste constant,
- ou préférer le data-full-size-image-url pour garantir une correspondance unique.

**Ce que l'on apprécie en tant que testeur automatisation :**

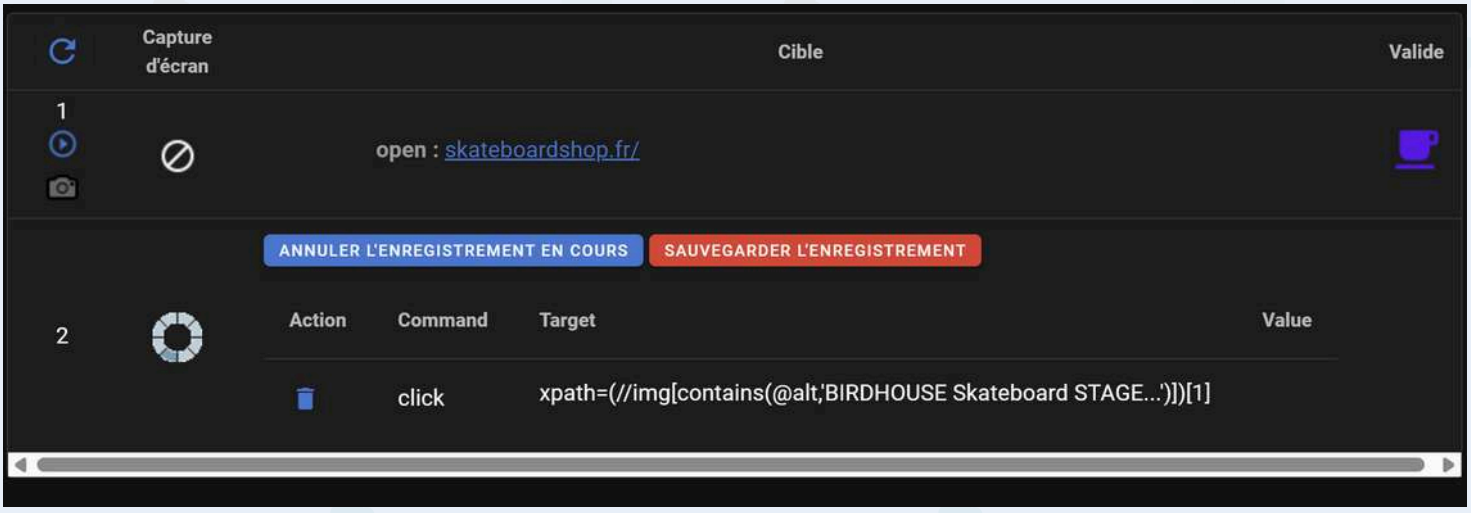
- **CloudNetCare ne se limite pas à l'enregistrement brut : il propose plusieurs sélecteurs intelligents, et on peut choisir le plus stable.**
- **L'IA intégrée peut, si activée, rechercher un nouvel XPath automatiquement si celui d'origine devient obsolète (par exemple, si un attribut change dans une nouvelle release).**
- **Cela améliore la résilience des tests E2E, et réduit le phénomène de tests flaky.**



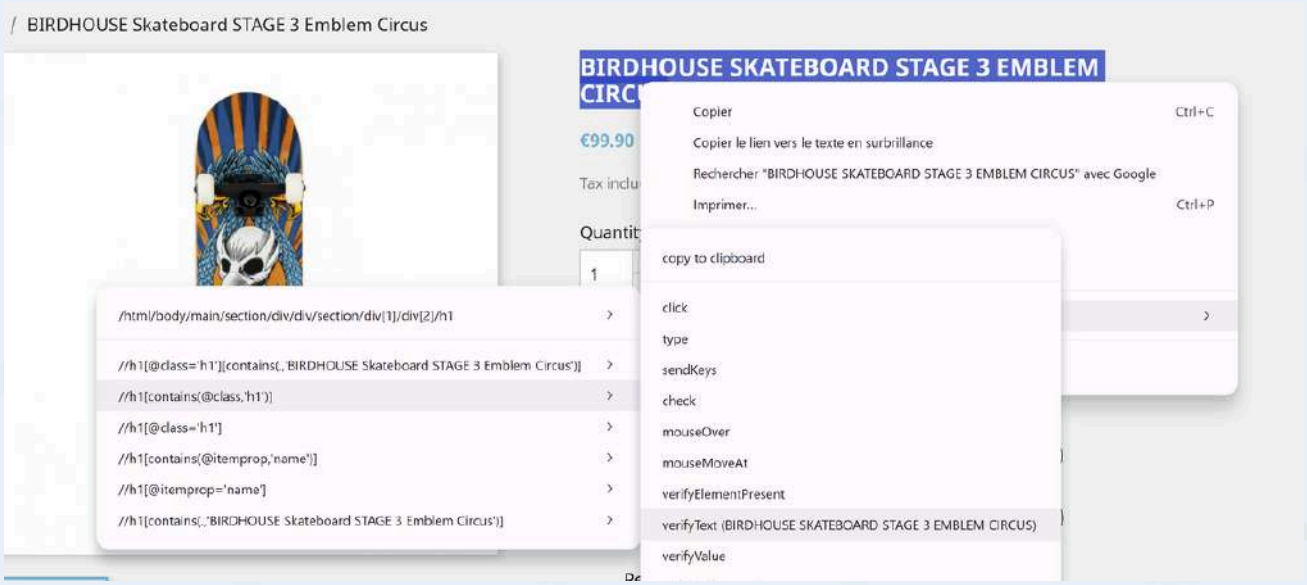
Pour éviter d'enregistrer des actions indésirables, celui-ci n'est pas automatique, c'est l'utilisateur qui décide quelle action doit être enregistrée et à quel moment.

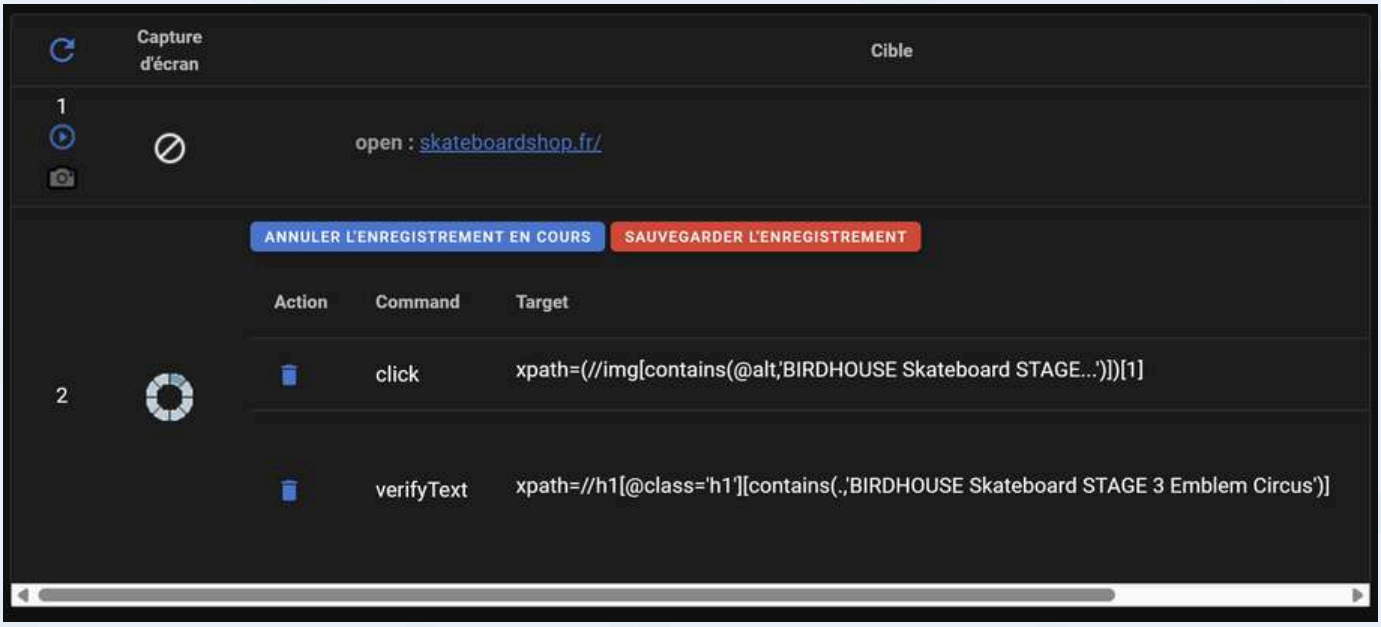
Par défaut, les actions simples sont directement accessibles (click, type, ...). Il existe un usage avancé basé sur les xPaths.


Cliquez sur “Click” sur le premier skate. Et là, l’étape évolue :




On continue l’enregistrement. Vérifions le titre par exemple !

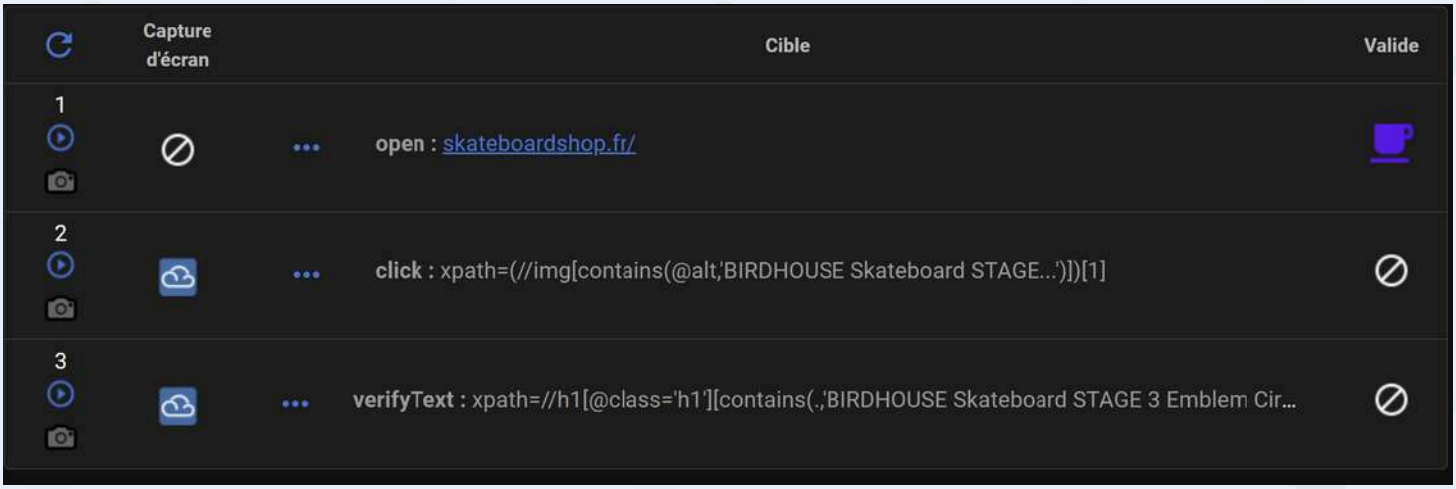





On peut vérifier tout les éléments de la page avant de faire tout son parcours utilisateur.

Si vous avez fini votre parcours, il faut stopper l'enregistrement dans le site de CloudNetCare en cliquant sur le bouton rouge "Sauvegarder l'enregistrement"

**En activant l'option iaverify, Il peut vérifier si l'image affichée correspond bien à la description attendue (« le site est en anglais », « le produit affiché est un skateboard », etc.)**




Nous avons bien nos 3 étapes :

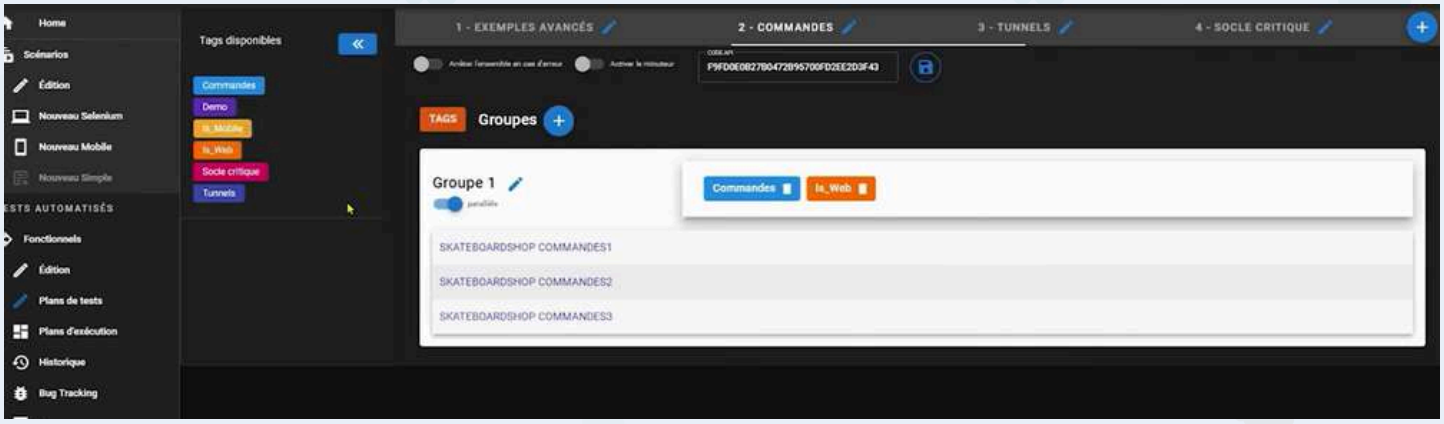
- aller sur le site skateboardshop.fr
- cliquer sur le premier skate
- vérifier que le titre de l'article correspond à notre attente.


Vous pouvez exécuter vos cas de tests sur les différents navigateurs (cela fonctionne pareil pour le mobile)
Puis cliquez sur Validation
Le script va être exécuté sur un navigateur dans le cloud.

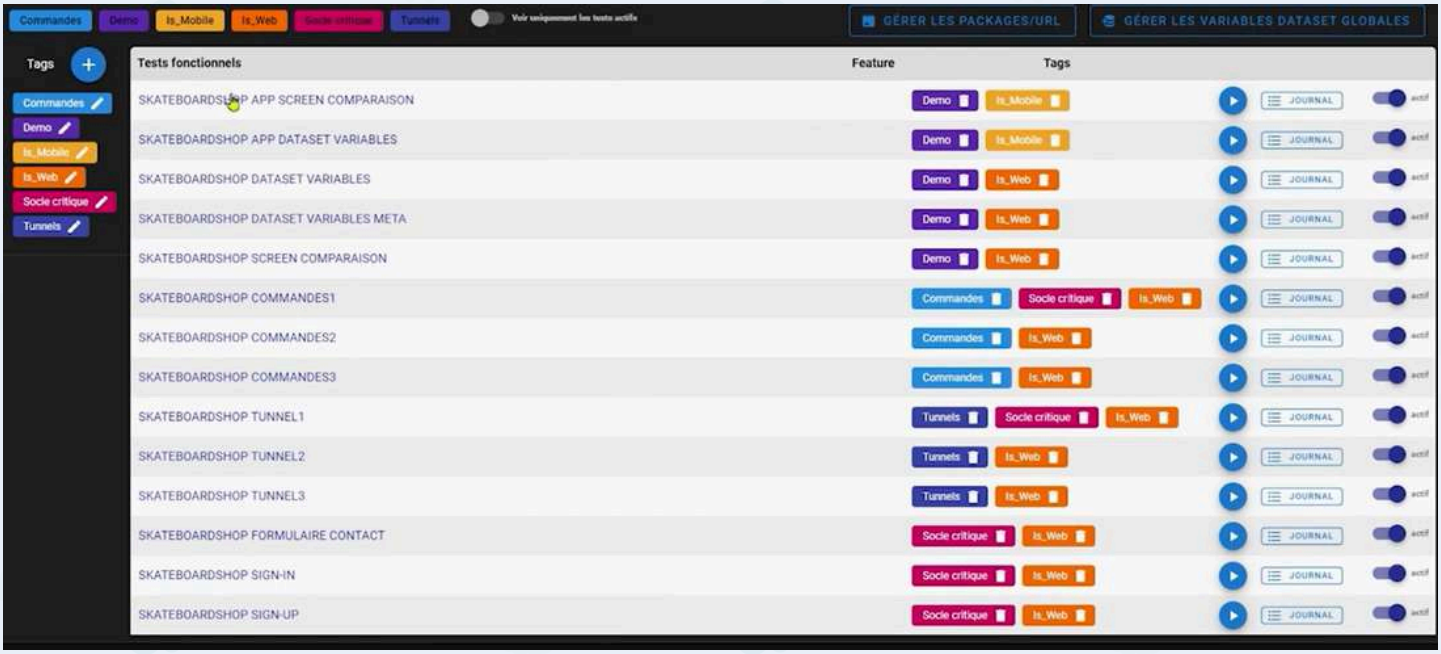

En temps réel, on peut voir les screens shots évolués.


## Tests plan

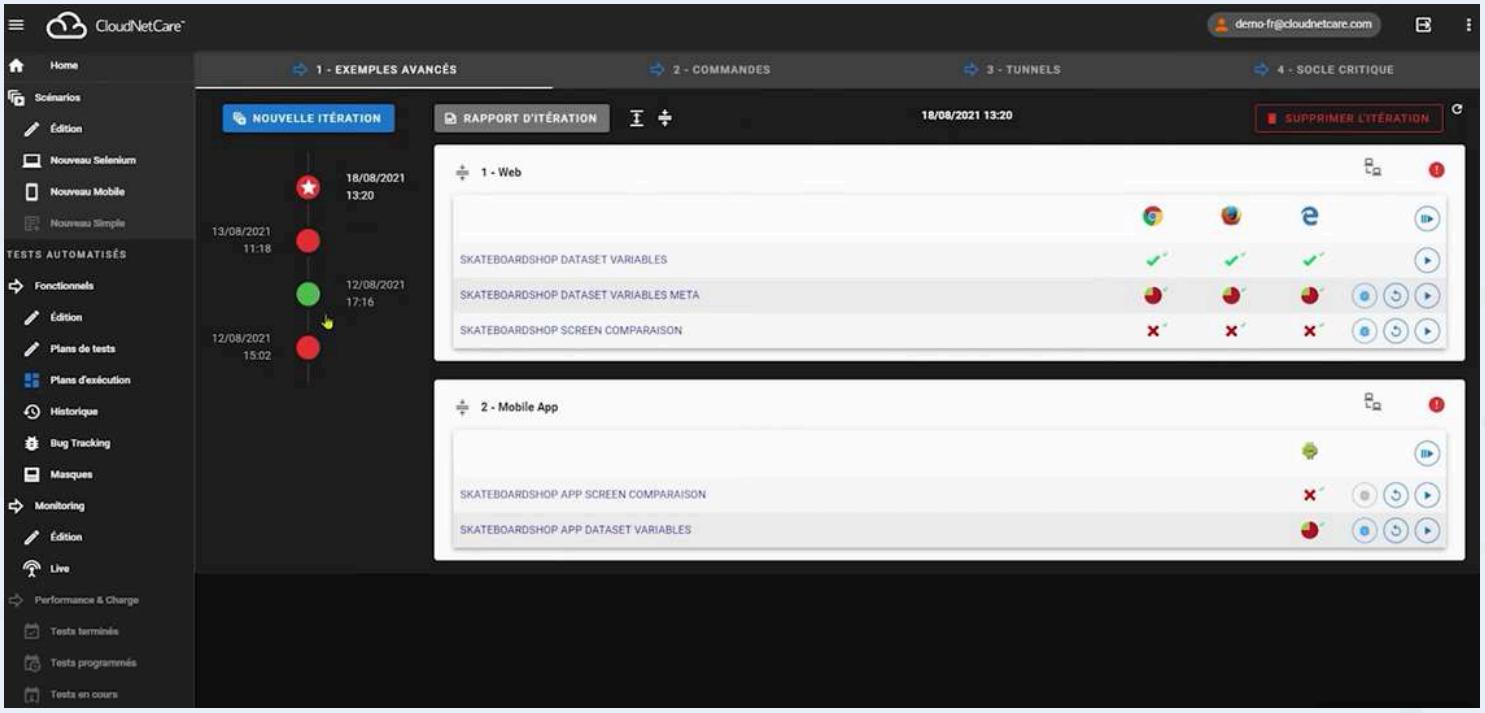
Vous pouvez organiser vos tests via des tests plan :




Dans les exemples avancés, vous pouvez les grouper, créer des tags, ...
Vous aurez la possibilité de les exécuter soit en série, soit en parallèle dans l'ensemble de ces groups



## Plan d'éxécution




L'image ci-dessus illustre une fonctionnalité souvent méconnue mais pourtant très utile de CloudNetCare : la gestion des itérations de tests automatisés.

À travers cette interface, les équipes QA peuvent piloter leurs campagnes fonctionnelles aussi bien sur le Web que sur le Mobile, en centralisant les résultats, les comparaisons d'écrans et la gestion des datasets.

Chaque exécution de test est ici regroupée dans une itération : un lot d'exécutions correspondant à une version, un sprint ou une livraison donnée.
Sur la gauche, la timeline permet de visualiser la succession des itérations (avec leurs dates et statuts).

Un simple coup d'œil permet de savoir quelles campagnes sont en réussite ou en échec, grâce aux indicateurs colorés :

- 🟢 vert : tests passés avec succès
- 🔴 rouge : tests échoués

Cette approche facilite le suivi des non-régressions dans le temps et permet de documenter précisément les résultats de chaque version testée.

CloudNetCare met en avant une force rare : la centralisation Web + Mobile dans un même plan d'exécution.
Dans cet exemple, deux groupes de tests apparaissent :

- Web, contenant des scénarios
- Mobile App, pour la partie application Android du même produit.
-

Chaque scénario est exécuté sur différents navigateurs (Chrome, Firefox, Edge) ou environnements mobiles, permettant de vérifier la cohérence du rendu et du comportement fonctionnel.

Une approche visuelle des résultats
Les icônes de graphiques circulaires (camemberts partiellement rouges) représentent la répartition des tests réussis/ échoués au sein de chaque scénario.

Ce type de visualisation rend immédiatement compréhensible l'état de santé du produit : un test qui passe sur Firefox mais échoue sur Edge attire rapidement l'attention.

C'est une approche précieuse pour identifier les incohérences entre navigateurs et mieux cibler les correctifs à apporter.

Le bouton « Rapport d'itération » (en haut de la vue) permet de générer automatiquement un compte rendu complet :

- résumé des résultats,
- statistiques globales,
- captures d'écran comparatives,
- et métadonnées (environnements, datasets utilisés, etc.).

Ce rapport est particulièrement utile pour les équipes CI/CD : il peut être partagé à la fin de chaque build afin de suivre la qualité en continu.

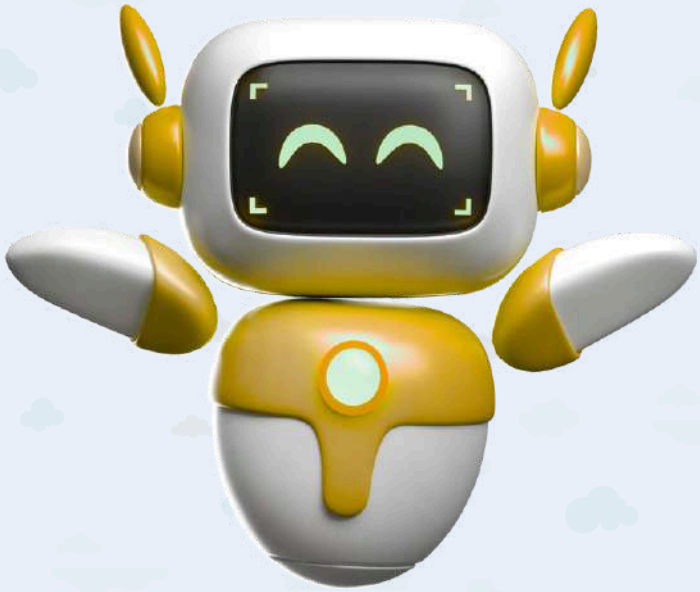
**Ce tableau de bord illustre bien la philosophie de CloudNetCare : rendre la qualité observable.**

**Au-delà des tests automatisés, la plateforme propose également :**

- **le monitoring continu (dans l'onglet Performance & Charge),**
- **l'intégration avec des systèmes de bug tracking,**
- **et la visualisation live des exécutions.**

**Grâce à cette centralisation, les testeurs, développeurs et chefs de projet peuvent collaborer autour d'une source unique de vérité, sans avoir à jongler entre différents outils.**

# ET L'IA DANS TOUT CA ?



CloudNetCare propose désormais une option IA, disponible en bêta, qu'il faut activer depuis son espace.

Cette fonctionnalité ouvre de nouvelles possibilités en matière de validation des tests. Elle repose sur un principe simple : au moment de l'ajout d'une étape dans un scénario, il est possible d'appeler l'intelligence artificielle.

3 bénéfices à l'utiliser :

- Vérification en langage naturel
- Génération automatique de cas de tests
- Auto-réparation des sélecteurs et descriptions

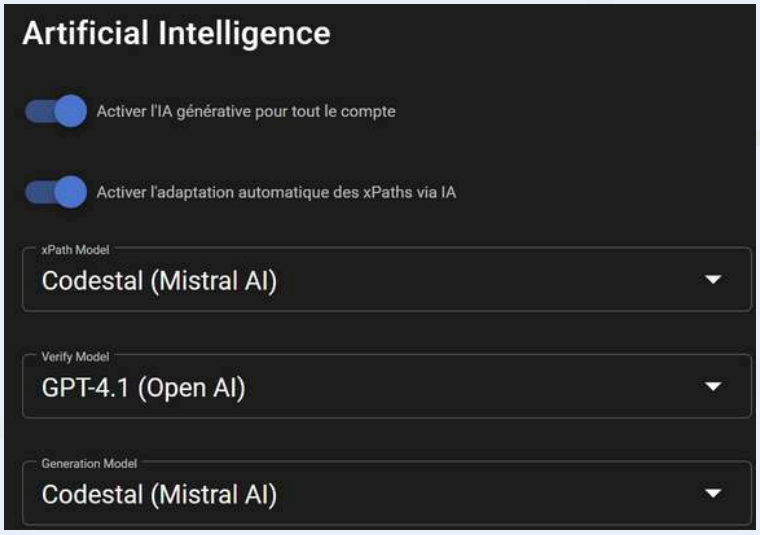
**Comme toute fonctionnalité en bêta, cette option doit être utilisée avec discernement. L'IA peut parfois donner une interprétation approximative ou dépendante du contexte.**

**L'IA ne cesse de s'améliorer.**

**Avec cette option IA avant coureuse, CloudNetCare permet d'ajouter une nouvelle dimension aux scénarios de test. Les testeurs peuvent désormais dialoguer avec l'outil en langage naturel et laisser l'IA décider de la validation d'un cas. Une piste prometteuse pour gagner en efficacité tout en simplifiant l'écriture des tests.**

## Comment activer l'option ?

Dans votre compte, vous devez aller dans les options puis, vous allez avoir ces choix :




L'option IA est incluse dans la licence CloudNetCare et peut être activée via un simple bouton.

Vous pouvez choisir ensuite les modèles IA adaptés à chaque usage :

- Verify Model (par défaut GPT-4.1 (OpenAI), idéal pour iaverify)
- xPath Model (par défaut Codestal (Mistral AI)), pour l'adaptation des sélecteurs
- Generation Model (par défaut Codestal (Mistral AI)), pour la création automatique de cas de tests.

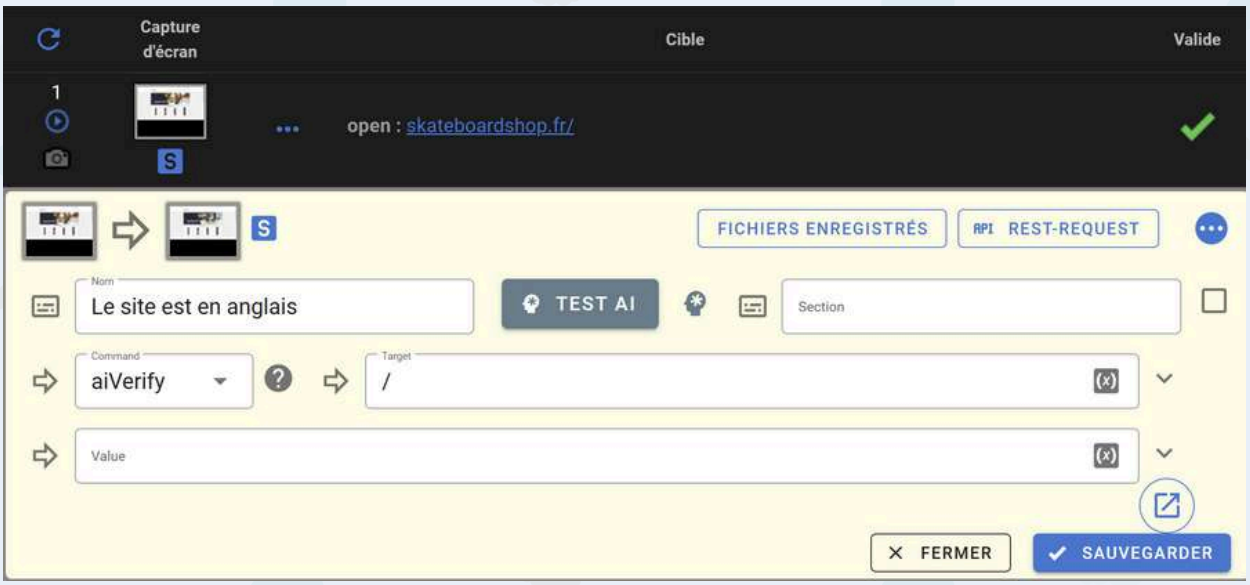
## Vérification avec iaverify


L'IA permet de valider une étape à partir d'une description en langage naturel.

Exemple :

- Étape : iaverify
- Description : "Le site est en anglais"

L'IA va analyser la page et répondre Oui ou Non pour valider le cas de test.




L'image ci-dessus illustre un scénario simple dans CloudNetCare.


Après avoir ouvert notre site internet. Nous ajoutons une étape avec la commande aiVerify et la description "Le site est en anglais".

Nous exécutons le test, et on peut voir que le retour est un succès : le site est en anglais.

## Auto-réparation des sélecteurs et descriptions

Dans un projet de test automatisé, un des efforts les plus lourds est la maintenance des sélecteurs (éléments DOM, identifiants, classes, XPaths…) et des descriptions de scénarios (libellés, étapes, données) à chaque évolution de l'application.

Quand l'interface change (refonte UI, nouveau framework, déplacement de composants), beaucoup de tests se cassent ou deviennent obsolètes, ce qui génère du coût, du délai et parfois des alertes « fausses régressions ».

L'auto-réparation (ou auto-heal) désigne la capacité d'une plateforme d'automatisation à détecter qu'un sélecteur ne fonctionne plus, à appliquer automatiquement ou semi-automatiquement une correction, et à prendre en compte la nouvelle localisation/structure du composant. Même principe pour une description de scénario qui ne correspond plus à la réalité métier : la plateforme peut suggérer une mise à jour ou alerter sur la dérive.

**Pourquoi c'est intéressant**

- **Réduction de la dette de maintenance : moins de scripts cassés après un déploiement UI.**
- **Gain de temps pour l'équipe qui passe moins de temps à corriger les tests.**
- **les tests automatisés restent fiables plus longtemps.**
- **Moins de « bruit » dans les rapports : les échecs dus à des changements d'UI ne doivent pas polluer le signal qualité.**

**Parce que rien n'est magique :**

- **Les heuristiques d'auto-réparation peuvent se tromper : un nouveau composant « visuellement proche » peut ne pas correspondre fonctionnellement. Il faudra prévoir une validation humaine.**
- **Une refonte majeure d'UI ou un framework changeant (ex. passage de Bootstrap à un design radicalement différent) peut dépasser la capacité d'auto-réparation.**
- **Il faut bien surveiller la dette technique : corriger automatiquement trop souvent peut masquer des fragilités dans les scénarios métier.**
- **Documentation et suivi restent importants : même corrigé automatiquement, il faut archiver la précédente version et documenter la modification pour les audits et traçabilité.**
- **Il faut prévoir un plan de test de « gros changements » où les tests sont reconstruits plutôt que réparés.**

# CAS CLIENT

Une société, bien connue de la mode, rencontrait un problème récurrent : pendant les pics d'affluence (soldes, campagnes marketing), l'expérience utilisateur de son site e-commerce ne permettait pas de transformer suffisamment de visites en achats.

Le DSI et la cheffe de projet avaient identifié que les tensions IT/serveur générées lors des pics étaient un facteur de stress important.

## Objectif

Mettre en place des tests de charge visant non seulement à solliciter l'infrastructure, mais à mesurer la qualité du ressenti de l'utilisateur pendant ces périodes critiques.

## Solution déployée

CloudNetCare a activé des scénarios de test reproduisant des parcours réalistes via de « vrais navigateurs » (et non uniquement des requêtes HTTP) :

- un internaute qui visite le site pour la première fois et parcourt le catalogue,
- un internaute qui crée un compte puis passe commande,
- un internaute déjà inscrit qui se connecte et passe commande.
- Cette approche permet de mesurer la qualité perçue côté utilisateur : temps de chargement, fluidité, passage à l'achat, en fonction de l'augmentation progressive du nombre de connexions simultanées.

## Bénéfices observés

- Les points de contention techniques ont pu être identifiés en amont et mesurés précisément.
- Les équipes dev et hébergement ont pu corriger les goulots d'étranglement grâce aux rapports de test.
- L'expérience utilisateur a été améliorée : les internautes pouvaient accéder au site et passer commande sans rupture majeure.
- L'équipe support était moins submergée par les réclamations clients pendant les pics, ce qui a permis un meilleur accompagnement des ventes.

# CONCLUSION ?

CloudNetCare s'impose comme une solution française complète et évolutive, réunissant tests fonctionnels (web/mobile), tests de performance et supervision UX au sein d'une même plateforme SaaS.

Ce positionnement "tout-en-un" répond à un besoin croissant des équipes QA : centraliser leurs activités de test sans multiplier les outils ni complexifier les intégrations CI/CD.

Cette mise en place peut se faire en toute autonomie ou avec l'aide de l'équipe CloudNetCare.

Grâce à son recorder intelligent, ses sélecteurs auto-corrigeants et sa brique d'intelligence artificielle capable de générer ou de réparer des tests automatiquement, CloudNetCare fait partie des rares plateformes à véritablement industrialiser la qualité de bout en bout, du test fonctionnel au monitoring.

Son ancrage en France et son hébergement sur des infrastructures locales (OVH et Azure France) en font une alternative crédible pour les organisations soucieuses de sécurité, conformité RGPD et souveraineté des données.

**L'équipe est très professionnelle et toujours disposées à aider et mettre en place la solution pour mieux satisfaire votre besoin.**

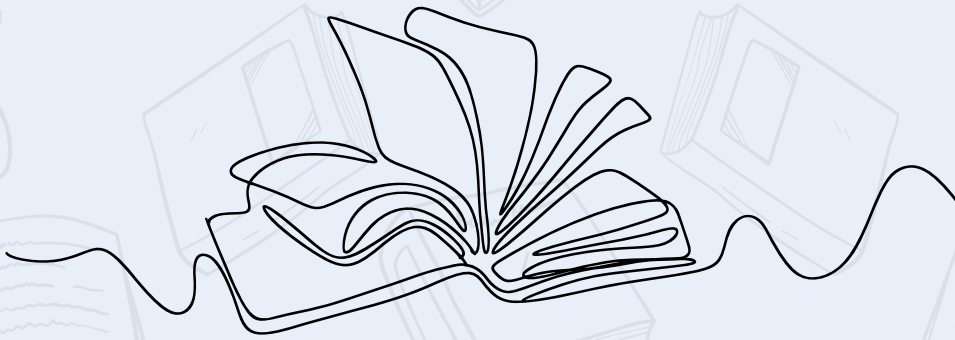
Mais le marché ne cesse d'évoluer. De nouveaux acteurs comme QANARY arrivent avec une approche no-code et IA-native, cherchant à simplifier encore davantage la création et la maintenance des tests automatisés. Ces solutions incarnent une nouvelle génération d'outils centrés sur la rapidité, l'autonomie et l'intelligence adaptative.

**Face à cette dynamique, CloudNetCare conserve une longueur d'avance sur la robustesse, la maturité et la profondeur fonctionnelle, là où d'autres misent sur la simplicité et l'expérimentation.**

**Un constat rassurant : la diversité des approches montre que le test logiciel entre dans une ère d'innovation continue, où les outils se complètent plus qu'ils ne s'opposent, au service d'une qualité plus intelligente, plus accessible, et surtout plus intégrée.**

# JEU CONCOURS

## DEFI DU TESTEUR



**Tentez de remporter un exemplaire du livre “Le test en mode Agile” de Christophe Moustier !**

5 dessins sont cachés dans le magazine (en plus de celle-ci), à vous de les trouver et d’indiquer les numéros des pages où ils se trouvent

**Image à trouver !**

### Comment participer ?

**Envoyez votre réponse par email avant le 22 novembre 2025 à 23h59 à : feedback@lemagdutesteur.fr.**
**Les deux premiers participants à répondre correctement gagneront.**

### Lot à gagner :

2 exemplaires du livre “Le test en mode Agile” (valeur : 45€) d’édition ENI.


- Le règlement complet est consultable ici ou à la demande par email:

- Concours gratuit – Une seule participation par personne – Données supprimées à l’issue du jeu

- Les gagnants seront contactés par mail

# PODCAST QG QUALITE

QG Qualité est un podcast francophone dédié à la qualité logicielle. Conçu et animé par SSID Testing Agency, pure-player de la qualification logicielle depuis plus de dix ans.

L'animateur est Romuald Lenormand, ancien journaliste, aujourd'hui automaticien chez SSID, qui orchestre chaque échange avec des expert(e)s de la qualité logicielle. Il est produit avec la contribution d'Élolia Drubay, responsable communication et image de la marque SSID.

Le podcast s'adresse à toutes celles et ceux qui souhaitent comprendre, améliorer ou simplement partager leur vision de la qualité dans les projets IT, des testeurs passionnés aux chefs de projets, en passant par les développeurs, les Product Owners et les curieux du monde de la tech.

**Quelques épisodes notables :**

- MARC HAGE CHAHINE - La Taverne du Testeur : focus sur l'un des pionniers de la qualité logicielle. Romuald reçoit Marc Hage Chahine pour évoquer son parcours, la création de la Taverne du Testeur et la manière dont la communauté QA francophone s'est structurée ces dernières années.

- Benjamin Butel - Ilek - La qualité logicielle à grande vitesse des start-ups. Avec Benjamin Butel, focus sur la réalité de la QA dans un contexte de croissance rapide et d'équipes en mouvement constant.

- NANCAIDAH TOURÉ-CHAUVIN - De la restauration à QA Lead et Qalisty : une reconversion inspirante. Nous vous parlions de son podcast lors du numéro précédent.

**Disponibilité** : QG qualité est disponible sur plusieurs plateformes de podcast, notamment :

- Youtube :
- Spotify :