Parcours A — Testing Agent

Adservio | Dr Olivier Vitrac

2026-02-03

Contents

1 Parcours A — Testing Agent

"Du vibe coding à la falsification"

1.1 Contexte & Enjeu

Les agents IA génèrent du code avec une facilité déconcertante. Cette facilité crée un piège : **la confiance non vérifiée**.

Un développeur qui “vibe code” avec un agent produit du code qui *semble* fonctionner. Mais : - Les cas limites sont ignorés - Les contrats implicites sont violés - Les erreurs silencieuses s’accumulent

Le test est l’antidote. Un agent qui **écrit, exécute, et interprète des tests** apprend à confronter ses hypothèses à la réalité.

Enjeu pour Adservio : Développer une culture où l’agent est un outil de *vérification*, pas seulement de *génération*.

1.2 Objectif Pédagogique

À l’issue de ce parcours, vous serez capables de :

1. **Concevoir** une boucle agent → code → test → analyse → correction
2. **Identifier** les points où un agent génère de la fausse confiance
3. **Instrumenter** un agent pour qu’il produise des traces exploitables
4. **Distinguer** un test qui passe d’un test qui *prouve* quelque chose

---

1.3 Cahier des Charges Minimal

1.3.1 Périmètre

Choisissez un repo Adservio réel (ou <your-repo> en fallback) contenant : - Une API publique ou CLI documentée - Au moins 10 fonctions/méthodes testables - Une documentation (même partielle)

1.3.2 Fonctionnalités de l’Agent

L’agent doit être capable de :

1. **Découvrir** l’API/CLI du projet
   - Lister les points d’entrée publics
   - Identifier les signatures et types
2. **Générer** des tests
   - Tests unitaires (fonction isolée)
   - Tests d’intégration (scénario complet)
   - Couverture des cas nominaux ET limites
3. **Exécuter** les tests localement
   - Lancer pytest (ou équivalent)
   - Capturer stdout/stderr
   - Parser les résultats
4. **Analyser** les échecs
   - Identifier la cause probable
   - Distinguer : bug code vs bug test vs environnement
5. **Itérer**
   - Proposer un fix (code ou test)
   - Ré-exécuter pour valider

1.3.3 Scope Minimal Viable

- 5 tests générés minimum
- Au moins 1 échec analysé et corrigé
- Trace complète de la session

1.4 Contraintes Non-Négociables

Contrainte	Justification
Tests exécutés localement	Pas de simulation, pas de “ça devrait marcher”
Échecs expliqués	Un test rouge sans analyse = livrable invalide
Pas de mock universel	Les mocks doivent être justifiés et documentés
Trace structurée	JSON ou format parseable, pas de texte libre
Un livrable sans trace exploitable est considéré comme non livré	Règle d’or Adservio

1.4.1 Ce qui est Interdit

- Simuler l’exécution des tests
- Ignorer un échec (“on verra plus tard”)
- Générer des tests triviaux (assert True)
- Faire tourner l’agent sans supervision humaine finale

1.5 Auditabilité & Traçabilité

Toute action de l’agent doit produire une trace exploitable par un humain : décisions, appels d’outils, hypothèses, erreurs, sorties intermédiaires.

1.5.1 Format de Trace Attendu

```
{
  "session_id": "uuid",
  "timestamp": "ISO8601",
  "steps": [
    {
      "step_id": 1,
      "action": "discover_api",
      "input": {"repo": "path/to/repo"},
      "output": {"endpoints": ["func_a", "func_b", "..."]},
      "reasoning": "Analyse des fichiers __init__.py et public exports",
      "status": "success"
    },
    {
      "step_id": 2,
      "action": "generate_test",
      "input": {"target": "func_a", "strategy": "boundary"},
      "output": {"test_code": "def test_func_a_empty_input(): ..."},
      "reasoning": "Cas limite : entrée vide",
      "status": "success"
    },
    {
      "step_id": 3,
      "action": "run_tests",
      "input": {"test_file": "test_generated.py"},
```

```
      "output": {"passed": 3, "failed": 1, "error": "AssertionError at line 12"},
      "reasoning": null,
      "status": "partial"
    }
  ]
}
```

1.5.2 Niveaux de Trace

- **Niveau 1 (minimum)** : actions + entrées/sorties
- **Niveau 2 (attendu)** : + reasoning + timestamps
- **Niveau 3 (bonus)** : + métriques, + replay possible

1.6 Livrables Attendus

Livrable	Format	Description
Script agent	Python	Code exécutable de l'agent
Tests générés	Python/pytest	Suite de tests produite par l'agent
Trace d'exécution	JSON	Log structuré de la session
Rapport d'analyse	Markdown	Synthèse : succès, échecs, apprentissages

1.6.1 Structure Suggérée

```
testing_agent/
├── agent.py              # Script principal
├── generated_tests/
│   └── test_*.py         # Tests générés
├── traces/
│   └── session_*.json    # Traces d'exécution
└── report.md             # Rapport final
```

1.7 Critères d'Évaluation

Critère	Poids	Indicateurs
Couverture	20%	Nombre et diversité des tests générés
Exécution réelle	25%	Tests effectivement lancés, résultats capturés
Analyse des échecs	25%	Qualité du diagnostic, pertinence des corrections
Traçabilité	20%	Complétude et exploitabilité de la trace
Itération	10%	Capacité à corriger et ré-exécuter

1.7.1 Barème

- **Insuffisant** : Tests générés mais non exécutés, ou exécutés sans trace
- **Passable** : Tests exécutés, échecs listés mais non analysés

- **Satisfaisant** : Boucle complète avec analyse et trace
- **Excellent** : Itérations multiples, trace exemplaire, insights originaux

---

1.8 Pièges & Anti-Patterns

1.8.1 Piège 1 : Le test qui teste le mock

L'agent génère un mock, puis teste que le mock retourne ce qu'il a configuré. → **Solution** : Exiger au moins 50% de tests sans mock.

1.8.2 Piège 2 : Le test tautologique

```python
def test_add():
    assert add(2, 2) == add(2, 2)  # Toujours vrai
```

→ **Solution** : Vérifier que chaque test a une valeur attendue *externe*.

1.8.3 Piège 3 : L'échec ignoré

"Le test échoue mais c'est probablement un problème d'environnement." → **Solution** : Tout échec doit être diagnostiqué ou explicitement classé comme "hors scope".

1.8.4 Piège 4 : La confiance dans le vert

Tous les tests passent → "le code est correct". → **Solution** : Analyser *ce que* les tests vérifient. Couverture ≠ qualité.

1.8.5 Piège 5 : L'agent bavard

L'agent génère 50 tests triviaux pour "faire du volume". → **Solution** : Évaluer la *diversité* des cas couverts, pas le nombre.

1.9 Retour Critique & Limites de l'Agent

À compléter **obligatoirement** à la fin du parcours :

1. **Qu'est-ce que l'agent a *semblé* bien faire mais n'a pas réellement fait ?**
2. **Où lui avez-vous fait confiance à tort ?**
3. **Quelle hypothèse *vous* avez faite qui s'est révélée fausse ?**
4. **Qu'interdiriez-vous à cet agent en production ?**
5. **Qu'est-ce qui vous a le plus surpris ?**

1.10 Questions Croisées (Préparation Demo Day)

1.10.1 Vers le parcours B (RAG)

- Comment un index RAG sur la documentation améliorerait-il la génération de tests ?
- Votre agent détecte-t-il les incohérences entre code et documentation ?

1.10.2 Vers le parcours C (MCP)

- Quels outils exposeriez-vous via MCP pour industrialiser votre agent ?
- Comment séparer le raisonnement (choix des tests) de l'exécution (lancement pytest) ?

1.10.3 Intégration Future

- Votre trace est-elle compatible avec un système de monitoring centralisé ?
- Comment enchaîner votre agent avec un agent de correction automatique ?

1.11 Ressources

- pytest documentation
- Claude Code CLI
- Property-based testing avec Hypothesis

Parcours A — Testing Agent — Adservio Workshop