9ème édition de la

Soirée du Test Logiciel Sophia - Antipolis

15 septembre 2026

17h à 23h

ETSI, Valbonne

Soirée du test logiciel

Sophia Antipolis

Green QA. Tester mieux pour consommer moins.

Comment supprimer le gaspillage dans nos pratiques de test sans sacrifier la qualité.

- Analyse d'impact
- Pipelines sobres
- Tests basés sur les risques
- Indicateurs de valeur

Et si 80 % de vos tests ne servaient à rien ?

– SILENCE –

CONTEXTE

L'obsession de la quantité n'a pas rendu nos tests meilleurs.

Pendant des années, la réponse à chaque problème qualité a été la même : plus.

LE RÉFLEXE HABITUEL

- Plus de pipelines
- Plus de parallélisation
- Plus de machines
- Plus de tableaux de bord
- Plus d'indicateurs
- Plus de tests

→ résultat réel

CE DONT ON PARLE AUJOURD'HUI

Le **Green QA** ne consiste pas seulement à mesurer l'empreinte carbone.

Il consiste avant tout à **supprimer le gaspillage** : à tester mieux, pas davantage.

CONSTATS

Ce que les chiffres nous disent

Sources — Green Software Foundation

3–4%

des émissions mondiales de CO_2 sont générées par le secteur informatique **autant que l'aviation civile.**

80%

des tests de non-régression **ne détectent aucun défaut pendant des mois** ; et pourtant s'exécutent en boucle à chaque modification de code.

< 5%

des équipes qualité **mesurent activement l'empreinte carbone** de leurs activités de test.

UN TEST D'INTERFACE UTILISATEUR

5 à 15 g de CO_2 par exécution
= 25 km en train ou 130 m en avion

EFFET MULTIPLICATEUR - 1 000 TESTS LOURDS SUR UN AN

≈ 37 tonnes de CO_2
= 18 allers-retours Paris — New York

Conférence

DIAGNOSTIC — LE PRINCIPE FONDATEUR

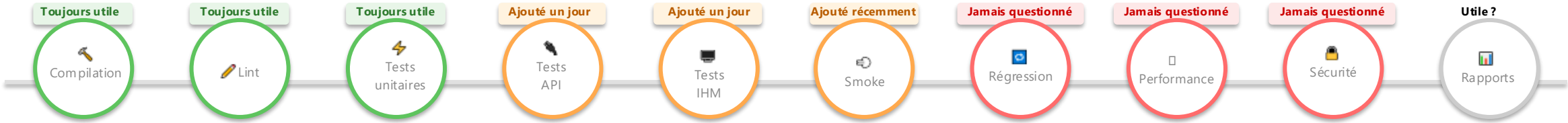
Le logiciel le plus écologique est celui qui ne s'exécute pas inutilement.

Avant de mesurer l'empreinte de nos tests, posons-nous la question de leur existence même.

DIAGNOSTIC

Un pipeline typique de bout en bout

Chaque étape a été ajoutée un jour. Combien ont été remises en question ?



Combien de ces étapes existent parce qu'elles apportent de la valeur ?
Parce qu'elles ont détecté un problème récemment. Parce qu'elles protègent quelque chose de critique.

Combien existent parce qu'on les a ajoutées un jour…
… sans jamais les remettre en question ? C'est le vieillissement silencieux du pipeline.

LA VRAIE MALADIE – PARTIE 1

Le retour sur investissement de l'automatisation qu'on ne calcule jamais

Sur vos tests automatisés, **combien ont détecté un défaut en production cette année ?**

Combien ont bloqué un déploiement à tort ?

Combien **n'ont absolument rien fait** ?

🚫 Difficile à mesurer , donc on ne mesure pas

🚫 "Plus de tests = plus de sécurité" semble évident

🚫 Les outils ne facilitent pas ce calcul

↳ LA CONSÉQUENCE DIRECTE

On automatise massivement

La suite grossit dans le temps ->sans jamais maigrir

Un test qui ne détecte jamais rien a un **retour sur investissement négatif**, et un coût carbone réel

Ce n'est pas un filet de sécurité.
C'est une charge.

LA VRAIE MALADIE – PARTIE 2

Les 5 pathologies d'un pipeline obèse

Des symptômes que chaque équipe reconnaît

Test bloat

La suite grossit à chaque sprint, ne maigrit jamais. Aucune stratégie de décommissionnement.

Volume ↑ · Signal ↓

Over-Testing

Régression complète à chaque commit, même pour des changements mineurs. Consommation maximale, valeur marginale.

Calcul ↑ · Valeur ↓

Env Sprawl

Des environnements de test actifs h24. Des machines allumées la nuit pour rien.

Coût ↑ · Utilité ↓

Flaky Tests

Des tests instables souvent relancés automatiquement plusieurs fois. On traite le symptôme, pas la cause. Triple consommation, zéro confiance.

Bruit ↑ · Confiance ↓

Data Waste

Bases de données de test à répliquées dans chaque environnement, jamais nettoyées. Les données et artefacts également.

Stockage ↑ · Pertinence ↓

LA VRAIE MALADIE – PARTIE 3

Les indicateurs de façade qui alimentent le gaspillage

INDICATEUR DE FAÇADE

"95 % de couverture de code"
Ne dit pas si les tests détectent quoi que ce soit

"800 tests automatisés"
Un chiffre qui grossit sans jamais maigrir

"Pipeline au vert"
Peut cacher 600 tests qui ne testent rien d'utile

"Durée de build : 45 min"
On s'y est habitué. C'est devenu la norme.

→ remplacer par

INDICATEUR DE DÉCISION

"% des zones à risque couvertes"
On cible ce qui compte pour l'activité

"Défauts détectés / tests exécutés"
La densité de détection : signal vs bruit

"Taux d'échappement en production"
L'indicateur qui mesure vraiment la qualité

"Tendance de durée historique"
Doit baisser, sinon on accumule du gaspillage

On mesure ce qui se mesure facilement , pas ce qui nous aide à décider.

LEVIERS D'ACTION

Conférence

LES LEVIERS D'ACTION

Optimiser ce qu'on exécute

Levier 1 — Exécution sélective, ordonnancement et infrastructure sobre

Analyse d'impact | Ordonnancement intelligent | Environnements éphémères | Architecture sobre

LEVIER 1 - TECHNIQUE 1

Analyse d'impact - Exécuter uniquement les tests concernés

TESTS UNITAIRES - PRINCIPE

1 Instrumenter le code

Activer la collecte de couverture (Jest, JaCoCo, Coverlet...) pour savoir quel test couvre quel module.

2 Détecter les fichiers modifiés

À chaque merge, le pipeline identifie la liste exacte des fichiers modifiés dans le commit.

3 Croiser les deux

Seuls les tests dont le code source a été modifié sont lancés. Les autres sont ignorés.

4 Régression complète repositionnée

La suite complète reste exécutée uniquement avant les livraisons. Pas à chaque commit.

🖥️ TESTS D'INTERFACE — APPROCHE PRAGMATIQUE

- Construire un fichier de correspondance explicite: test -> composants applicatifs couverts
- Le pipeline lit ce fichier et sélectionne les tests dont les composants ont été modifiés
- Playwright permet également une collecte de couverture navigateur native V8) pour les applications front
- Limites : Les effets de bord entre composants nécessitent toujours une régression complète avant livraison

✅ CE QU'ON OBSERVE EN SORTIE

→ Moins de tests exécutés à chaque commit de code
→ Durée de pipeline réduite significativement
→ Retour développeur beaucoup plus rapide
→ Consommation d'infrastructure réduite
→ Couverture globale préservée

LEVIER 1 - RETOUR D'EXPERIENCE

Ce qu'on a mis en place sur notre périmètre

ORCHESTRATION PAR BRANCHE ET PAR ÉQUIPE

Dépôt de code

↓

Lecture du nom de branche
feature/equipe-A-Souscription

↓

Orchestrateur central
Quelle équipe ? Quel périmètre ?

↓

Identification de l'équipe impactée
Équipe Contractualisation

↓

Lancement ciblé des tests de l'équipe concernée

AFFINAGE PAR MARQUEURS ET FONCTIONNALITÉS

- Les tests sont marqués par fonctionnalité / équipe métier
- L'orchestrateur combine le nom de la branche avec la cartographie des dépendances pour affiner la sélection
- Un composant partagé modifié déclenche automatiquement les tests de toutes les équipes qui en dépendent
- La cartographie est maintenue par les équipes – C'est un document vivant.

CE QUE ÇA CHANGE CONCRÈTEMENT

- On ne lance plus « tout » par précaution : On lance ce qui est pertinent par conception
- Les équipes sont responsables de leur périmètre de test (meilleure appropriation)
- Les tests transverses restent déclenchés sur les composants partagés

LEVIER 1 - TECHNIQUE 2

Ordonnancement intelligent du pipeline

DÉCLENCHEUR	CATÉGORIES AUTORISÉES	OBJECTIF
⚡ **Chaque dépôt de code**	Tests unitaires	Retour immédiat sur la logique — moins de 5 minutes
🔀 **Chaque demande de fusion**	Tests unitaires · Tests d'intégration	Valider les contrats entre composants avant fusion
🌿 **Fusion sur la branche principale**	Unitaires · Intégration · Contrôles essentiels IHM	Filet de sécurité avant propagation
🌙 **Nuit (planifié)**	Unitaires · Intégration · IHM complets	Couverture large sans impacter les développeurs
🚀 **Avant livraison**	Tout — sans exception	Régression complète — la seule fois où tout tourne
🗓️ **Hebdomadaire**	Performance · Sécurité	Tests coûteux sur infrastructure dédiée

⚡ **Retour développeur accéléré**
Un dépôt de code donne un résultat en minutes, pas en heures

🌱 **Consommation réduite**
Les tests lourds s'exécutent moins souvent mais au bon moment

🎯 **Signal de qualité renforcé**
Quand un test échoue, c'est pris au sérieux : il ne tourne pas en boucle inutilement

LEVIER 1 - TECHNIQUE 3

Environnements éphémères - Zéro machine allumée pour rien

1. **Auditer les environnements existants**
   Mesurer le taux d'utilisation réel de chaque environnement. Identifier les environnements actifs mais inutilisé.

2. **Définir ce qui peut devenir éphémère**
   Les environnements par demande de fusion et les environnements fonctionnels sont les premiers candidats.

3. **Automatiser le cycle de vie**
   Création automatique à l'ouverture d'une pull request. Destruction automatique à la fermeture ou après inactivité.

4. **Mutualiser les ressources partagées**
   Les services communs (bases de données, services tiers) sont partagés entre environnements pour éviter la duplication.

✅ **CE QU'ON OBSERVE EN SORTIE**

- → Consommation d'infrastructure réduite significativement
- → Chaque commit est testé dans un environnement isolé et propre
- → Suppression des environnements « zombies » oubliés
- → Meilleure reproductibilité des résultats de test

⚠️ **ÉVITER LA SUR-PARALLÉLISATION**

Multiplier les exécutions en parallèle sur des machines virtuelles sous-dimensionnées peut annuler les gains. Adapter les ressources de calcul aux exigences réelles des tests : ni trop, ni trop peu.

LEVIER 1 - ARCHITECTURE SOBRE

Principes d'architecture pour des pipelines économes

Détection rapide des échecs

Exécuter en priorité les tests rapides et à forte valeur ajoutée. En cas d'échec critique, interrompre immédiatement l'exécution plutôt que de gaspiller des ressources sur les étapes suivantes.

Dimensionnement adapté des ressources

Adapter les ressources de calcul aux exigences réelles des tests. Un test unitaire n'a pas besoin de la même infrastructure qu'un test de performance.

Données de test synthétiques

Générer des données à la volée plutôt que de copier d'importants ensembles de données de production.

Nettoyage rigoureux des artefacts

Les captures d'écran d'un test réussi il y a six mois sont inutiles. Archiver uniquement les échecs et les données de diagnostic critiques. Définir des politiques de conservation.

Conférence

LES LEVIERS D'ACTION

Optimiser ce qu'on choisit de tester

Levier 2 — Priorisation par le risque et élimination du bruit

Exécution sélective

Tests basés sur les risques

Élimination des tests instables

LEVIER 2 - TECHNIQUE 1

Tests basés sur les risques - Tester ce qui compte vraiment

LA MATRICE DE PRIORISATION

CRITICITÉ HAUTE · DÉFAUTS FRÉQUENTS Tests approfondis à chaque livraison	CRITICITÉ HAUTE · DÉFAUTS RARES Tests réguliers contrôles essentiels
CRITICITÉ FAIBLE · DÉFAUTS FRÉQUENTS Tests ciblés surveillance renforcée	CRITICITÉ FAIBLE · DÉFAUTS RARES Contrôles légers avant livraison seulement

Prioriser les tests en fonction de leur criticité pour l'activité et de la densité historique des défauts. Concentrer les tests approfondis sur les zones à haut risque et effectuer des contrôles plus légers sur les composants stables et à faible risque.

⚠ ÉLIMINER LES TESTS INSTABLES

Un test qui échoue de manière aléatoire n'est pas un filet de sécurité : c'est du bruit. Chaque relance automatique consomme des ressources sans apporter de valeur. Identifier, corriger ou supprimer les tests instables est un levier d'économie immédiat et mesurable.

PERSPECTIVE — MODÉLISATION PRÉDICTIVE

La matrice de risque peut être construite manuellement dans un premier temps. À plus long terme, une modélisation légère , croisant l'historique des défauts, la complexité du code et la fréquence des modifications peut automatiser et affiner cette priorisation de manière continue.

MESURER

Indicateurs et outils - Piloter dans le temps

LES 3 INDICATEURS À SUIVRE ABSOLUMENT

1. **Densité de détection**
   Défauts détectés / nombre de tests exécutés. Doit augmenter dans le temps. Si ce ratio baisse, la suite grossit plus vite que sa valeur.

2. **Efficience du pipeline**
   Durée de pipeline / défauts détectés. Surveiller la **croissance de la durée des tests** — si elle augmente sans amélioration de la détection, on accumule du gaspillage.

3. **Taux de tests dormants**
   Pourcentage de tests sans détection depuis 90 jours. Au-delà de 30 %, une campagne de décommissionnement s'impose.

🛠️ OUTILS DISPONIBLES

Greenspector
Répond à la question "comment mesurer" : analyse l'empreinte des applications et des tests en conditions réelles.

Cloud Carbon Footprint
Outil libre qui connecte vos comptes hébergement et traduit la consommation en kilowattheures et équivalent CO_2.

Kit Carbon Aware - Green Software Foundation
Traduit l'utilisation abstraite des ressources d'intégration continue en impact environnemental tangible et mesurable.

📊 IDÉATION - TABLEAU DE BORD MINIMAL

Un suivi simple suffit pour commencer : mis à jour chaque semaine, automatisable ensuite via plateforme de déploiement continue.

Tests exécutés | Durée pipeline | Défauts détectés | Tests dormants %

Consommation kWh

CONCLUSION

Vers une culture d'ingénierie qualité responsable

LES TROIS PILIERS DE CETTE CULTURE

🎯 **Intégrer l'éco-efficacité dans la stratégie de test**
La maîtrise des coûts et la réduction du gaspillage font partie de la stratégie qualité pas seulement de la stratégie environnementale.

🤝 **En partenariat avec les équipes d'exploitation**
L'optimisation de l'infrastructure de test est facilitée quand les équipes qualité et les équipes ops travaillent ensemble sur la visibilité des ressources.

📚 **Former les équipes aux tests écoresponsables**
Ce n'est pas une contrainte supplémentaire , c'est une montée en compétence vers un état d'esprit d'ingénierie qualité.

🦾 L'ÉVOLUTION DU MÉTIER

Mettre en place une démarche de Green QA c'est adopter la posture de **l'ingénierie qualité** , celle qui consiste à produire le **juste test au bon moment**, pas le maximum de tests en permanence.

Ce n'est pas une question d'outils. C'est une question de **décisions éclairées par la donnée.**

> *La qualité logicielle durable ne consiste pas à exécuter plus de tests mais à exécuter* **les tests qui apportent le plus de valeur.**

AU-DELÀ DE CETTE CONFÉRENCE - UN FIL CONDUCTEUR

LES INDICATEURS DE FAÇADE
mesurent l'activité

↓

LES INDICATEURS DE VALEUR
mesurent l'impact

↓

LE GREEN QA
révèle le coût caché de cette activité

↓

L'INGÉNIERIE QUALITÉ
consiste à piloter ces arbitrages grâce à la donnée

9ème édition de la

Soirée du Test Logiciel Sophia - Antipolis

15 septembre 2026

17h à 23h

ETSI, Valbonne

Soirée du test logiciel

Sophia Antipolis

Joseph NGAN

- **Référent Technique de Test** – Test Lead
- **AXA France** depuis 4 ans
- Passionné par la qualité logicielle, les échecs et le football

linkedin.com/in/joseph-ngan

SCAN ME

Votre avis nous intéresse !

Merci de votre écoute !