Tests automatisés

Principes & mise en application (TypeScript, Node.js, Jest, React Testing Library)

Arnaud Renaud

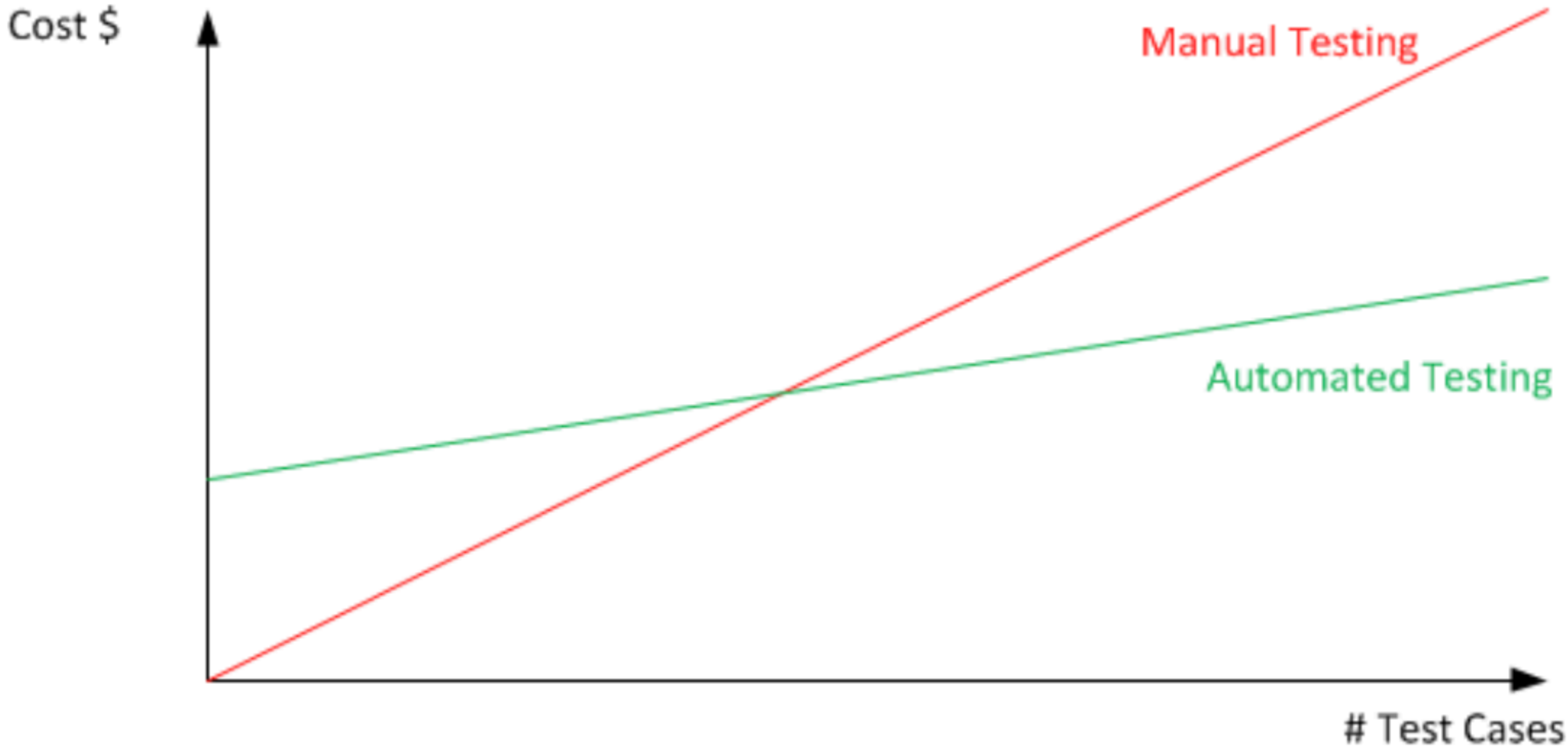
Pourquoi automatiser les tests ?

Une application devient rapidement trop complexe pour jongler mentalement entre tous ses cas d'usage.

Batterie de tests automatisés

Système de gestion de la complexité :

- Décharge mentale pour le développeur
- Garantie du fonctionnement d'une application complexe
- Spécifications intégrées et vérifiées automatiquement
- Ne nécessite pas d'interface


Cost $
Manual Testing
Automated Testing
Test Cases

Quels cas tester ? Où s'arrêter ?

Un tableau vide ? *null* ? *undefined* ?

- Se limiter aux cas réalistes
- Restreindre les cas en typant les paramètres

Périmètres de test

- Tests unitaires
- Tests d'intégration
- Tests de bout en bout

Tests unitaires

Contrôle du comportement d'un composant isolé (fonction ou méthode) dans différents cas d'usage.

Ici, on ne teste pas le comportement de ses dépendances internes.

Exemple : tester un algorithme de calcul.

Tests d'intégration

Contrôle du comportement d'une chaîne de composants.

Ici, on teste le comportement des dépendances internes concernées.

Exemple : vérifier qu'une méthode produit la bonne écriture en base de données.

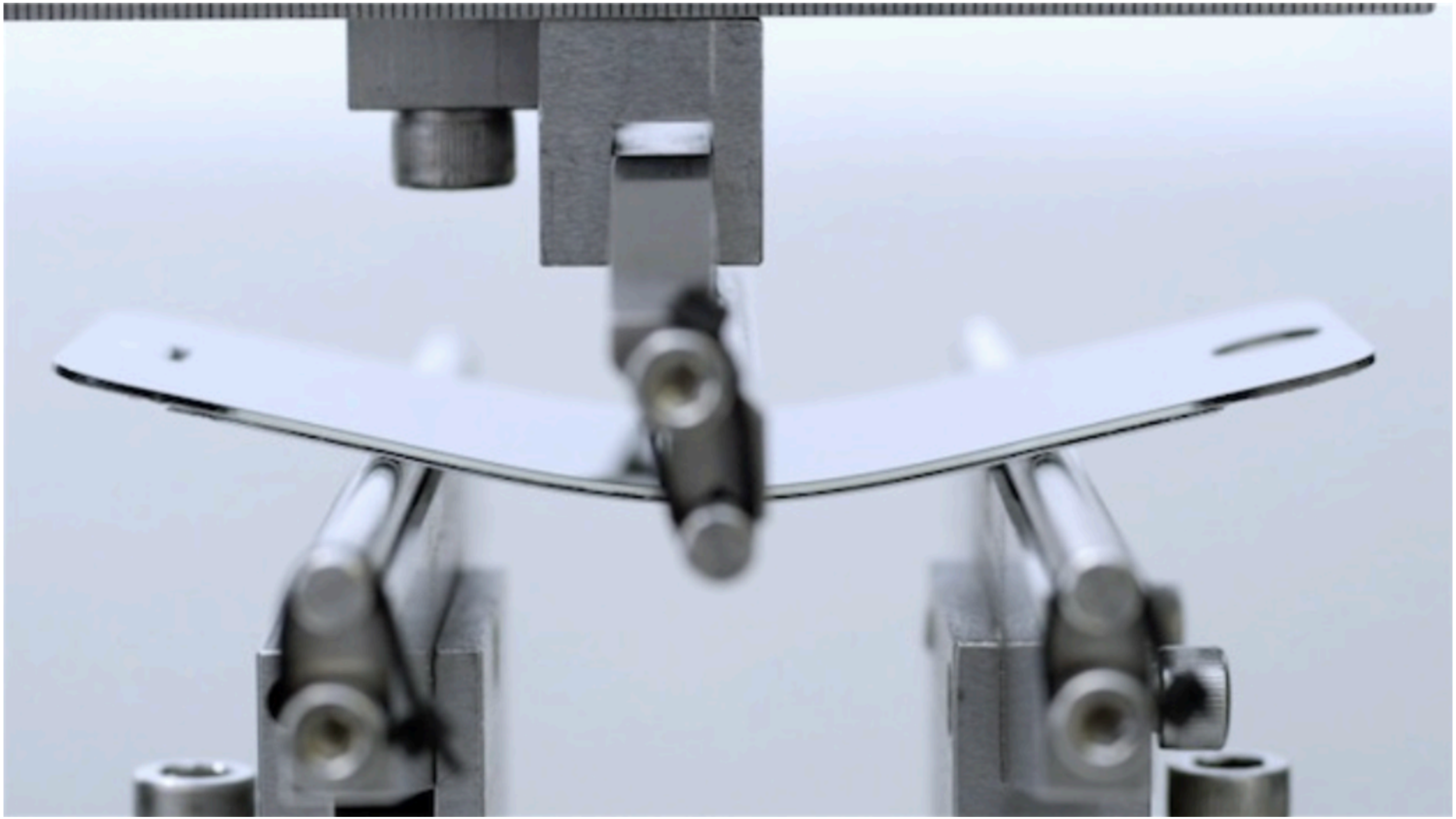
Tests de bout en bout

Contrôle du comportement de la chaîne complète des composants d'une fonctionnalité de l'application.

Ici, on se place du point de vue de l'utilisateur, sans importer de code dans les tests.

Exemple : en soumettant un formulaire web, vérifier que le bon message de succès ou d'erreur est affiché à l'utilisateur.

Tests unitaires



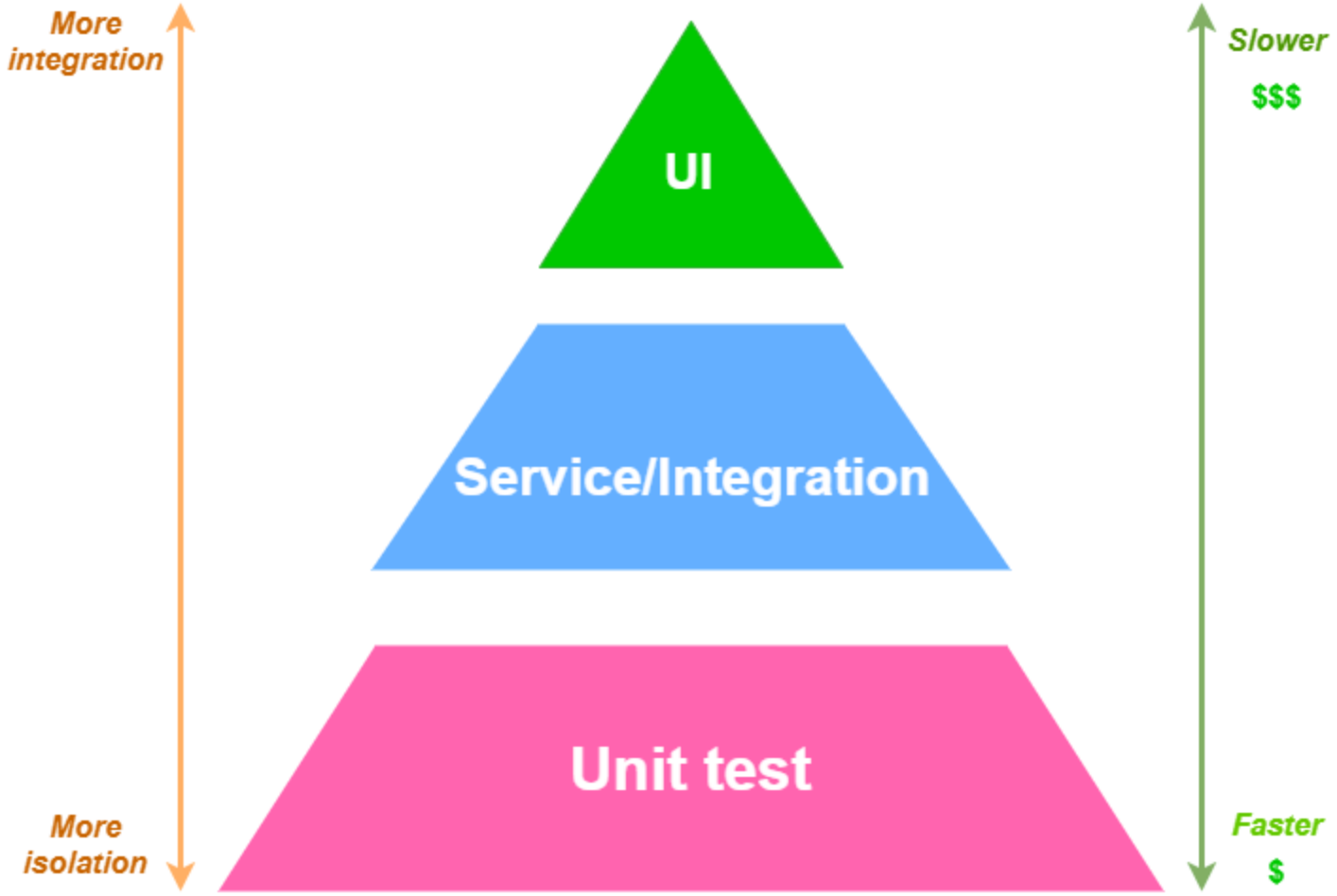
Tests d'intégration

Tests de bout en bout

Tests d'intégration et de bout en bout : jusqu'où aller ?

⚠️ Ne pas tester tous les cas : c'est la prérogative des tests unitaires, qui sont les plus précis.

Pyramide des tests



Fonctions pures et impures, environnement réel ou simulé

Fonction pure

Une fonction pure n'a pas d'effet de bord : elle ne change aucun état, se contente de retourner un résultat qui dépend de ses paramètres d'entrée.

Exemple : un algorithme de calcul.

Tester une fonction pure revient à contrôler sa valeur de retour.

Fonction impure

Une fonction impure a des effets de bord : elle modifie l'état environnant.

Exemple : envoyer un email ou modifier l'état d'un objet.

Pour tester une fonction impure, il faut contrôler l'état des objets appelés ou modifiés.

Environnement réel ou simulé ?

Si l'on contrôle l'environnement, on peut l'utiliser réellement pour le mettre dans différents cas.

Exemple : une base de données.

Sinon, on le remplace par une simulation (un bouchon, ou *mock*) : cela permet d'avancer dans les tests même si le service n'est pas implémenté.

Exemple : une API externe (simuler sa réponse en succès ou en erreur).

Pratiques de test

Test-driven development (TDD)

Écrire d'abord les tests, puis les laisser nous guider dans l'implémentation.

- L'interface (entrées et sorties) doit être clairement définie à l'avance
- Adapté en cas d'implémentation difficile avec des cas nombreux (quand on ne sait pas par où commencer l'implémentation)

Couverture de tests

Statistiques donnant une vue d'ensemble sur la proportion de nos fonctions et méthodes testées, et si tous leurs chemins sont parcourus par les tests.