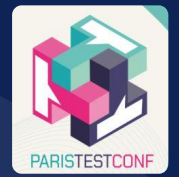
Comment une conception logicielle influence votre stratégie de test

Paris Test Conf 2023 - Christophe Breheret-Girardin



Martin Fowler
Développeur, auteur, speaker

“

L'architecture est vraiment importante.
Les choses qui sont ***difficiles à changer*** sont ***l'architecture initiale***, la culture et les compétences de l'équipe.
C'est pourquoi il est important de bien faire les choses ***dès le départ***.

”

Naim BenCharif
QA Analyst

> “Tester, c’est diriger les bugs *vers la sortie*”

> “Tester, c’est diriger l’équipe *vers la qualité*”

Julien Pessarossi-Langlois
Test Manager

Christophe Breheret-Girardin

+ 20 ans d'expérience

PARISTESTCONF

@CHRISTOPHE-BREHERET-GIRARDIN

@CHRISTOPHEB_G

@CHRISTOPHEBG

- Coach craft
- Auditeur
- Formateur
- Co-leader de tribu

OCTO Technology
Part of Accenture

- Conférencier
- Créateur de contenu
- Auteur

- Développeur
- Tech-lead
- Architecte
- Directeur technique

Le livre blanc : “Culture Test” volume 1

OCTO Technology
Part of Accenture

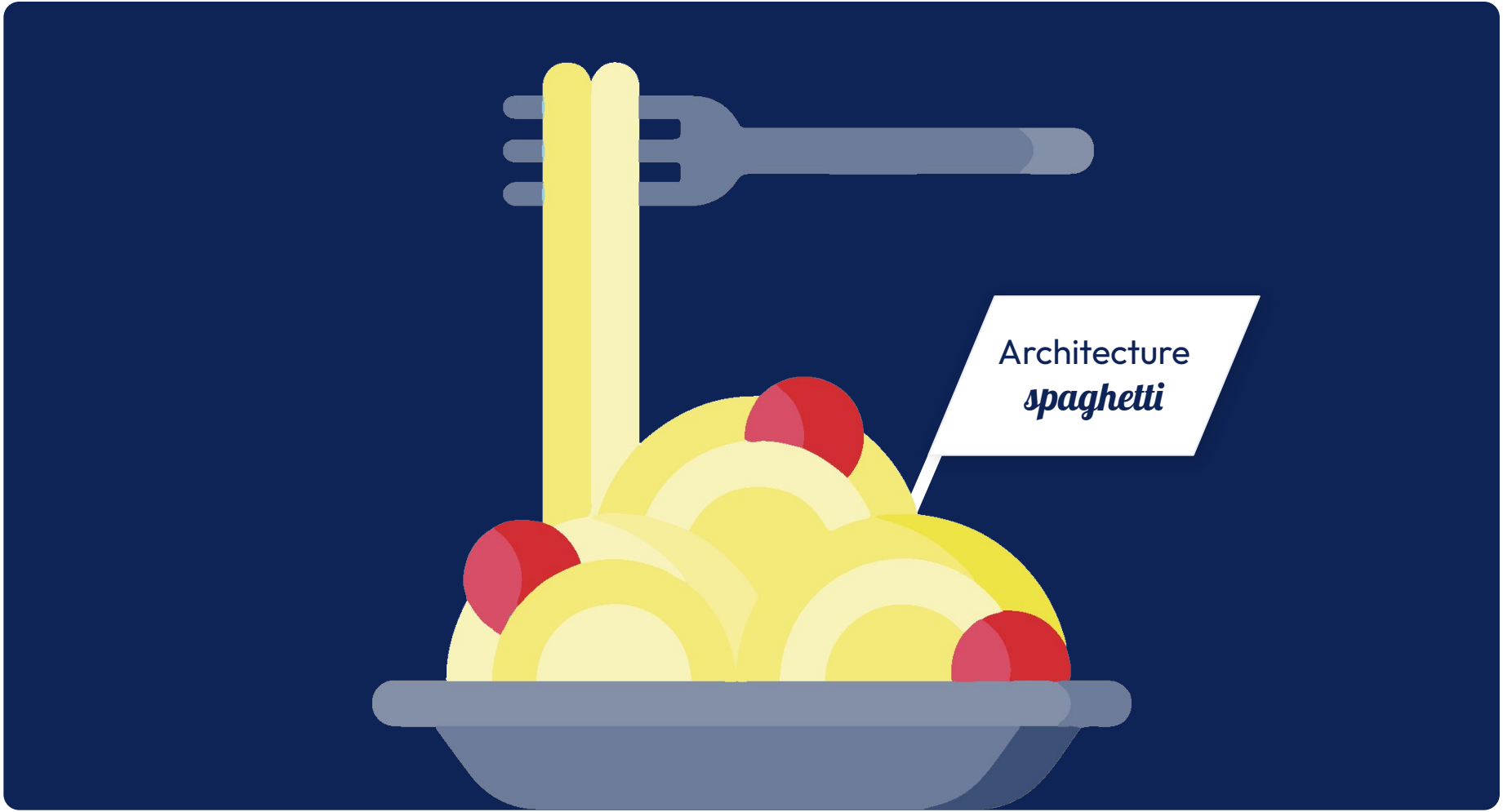
Co-écrit par :
Sylvie Ponthus
Stéphane Bedeau
Christophe Breheret-Girardin

Version gratuite PDF et epub à venir ici

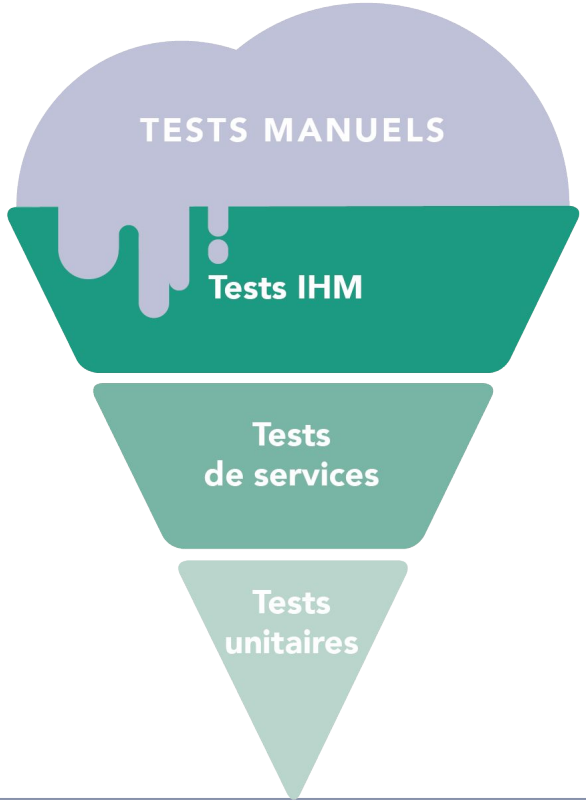
Sommaire

01

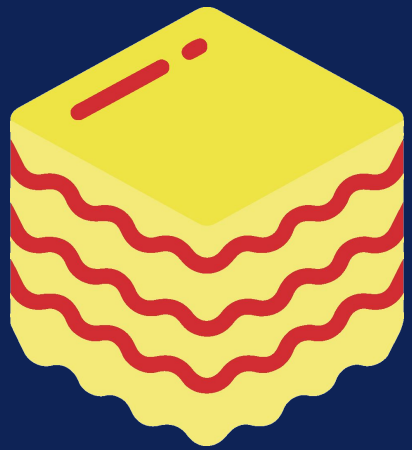
Un peu d’histoire



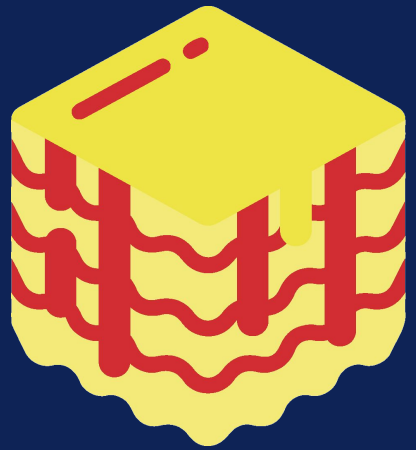
Stratégie de tests



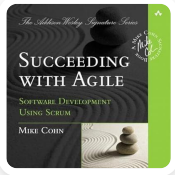
Anti-pattern : répartition des tests « *Ice Cream Cone* »



Architecture *en* couches



La pyramide idéale des tests **Mike Cohn** - 2009



Temps d'exécution +

Coût +

sécurité, charge, tests métier

Interface graphique, de bout en bout, Tests fonctionnels

Intégration, Tests d'API

Tests unitaires

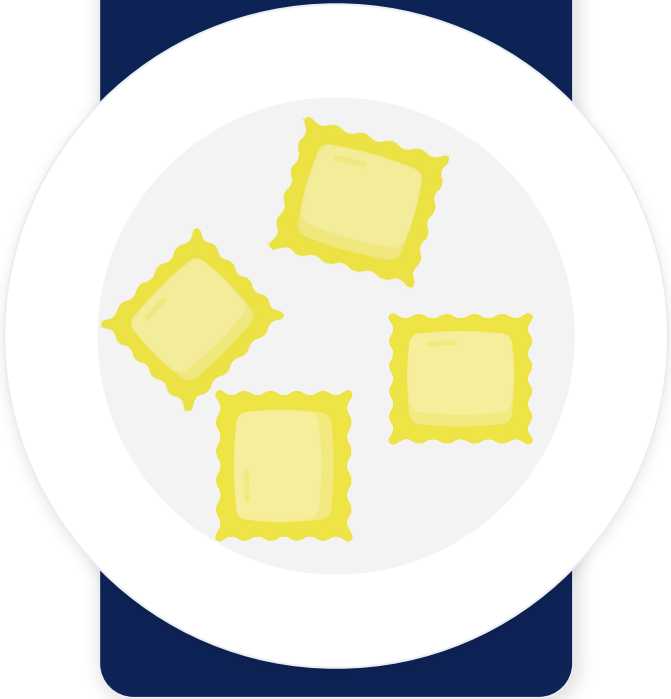
+ Quantité de tests

+ Investissement du développeur

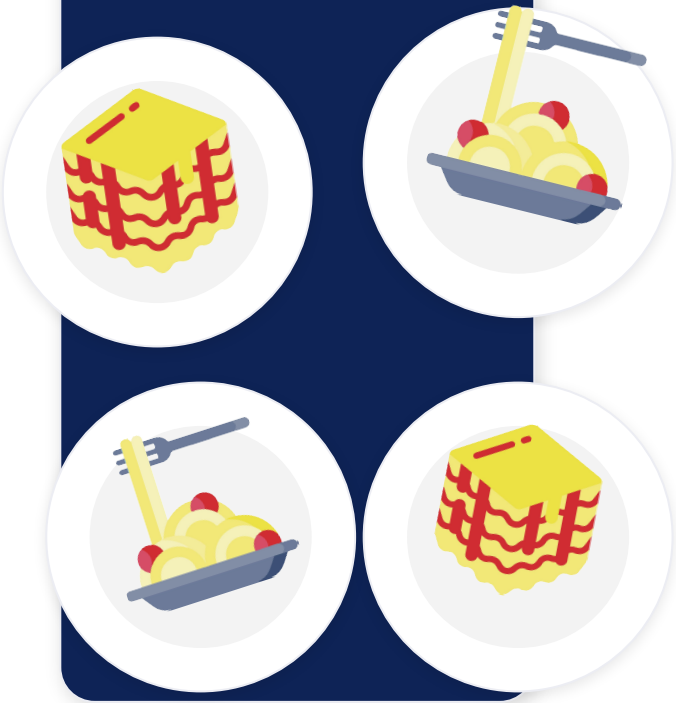
La seule et unique mesure de qualité ... ou pas

Les microservices

Division en
“petits” services



La réalité



02

Objectif

Automatiser les tests

Tester
c'est coder

Avec apport
de valeur

Le quadrant de test

Orienté métier

Automatisés et manuels

Tests système

Tests fonctionnels
Tests basés sur les exemples
Prototypes
Simulations

Tests d'acceptation (ou système)

Tests de processus
Alpha/Beta
Tests d'utilisabilité

Manuels

Aide l'équipe

Comportement du produit

Tests de composants

Tests Unitaires

Tests d'acceptation (ou système)

Performance / charge / stress
Maintenabilité
Portabilité...

Automatisés

Outillés et automatisés

Orienté technologie

La typologie applicative

Toute les parties de l'application ne se ressemblent pas.
Certaines amènent :

- Un aspect **business**
- Un aspect **différenciant**
- Un aspect **essentiel** sans en être le coeur
- Un aspect **nécessaire** sans être critique
- Etc.

Faut-il la **même architecture** ?
Faut-il la **même stratégie de test** ?

Place du *QA* ?

Place du *QA* ?

Dans *l'équipe* !

Friedrich Nietzsche
Philosophe, poète, auteur

> “
> La bonne idée est *comme une flèche*
> lancée par-dessus la colline :
> *un autre la ramasse* et la tire plus loin
> ”

03

Le coeur du business

Cas d'usage

Coeur *métier* de votre organisation

Aspect *différenciant*

Modèle métier *complexe*

Exemples :

Gestion des promotions par type de client

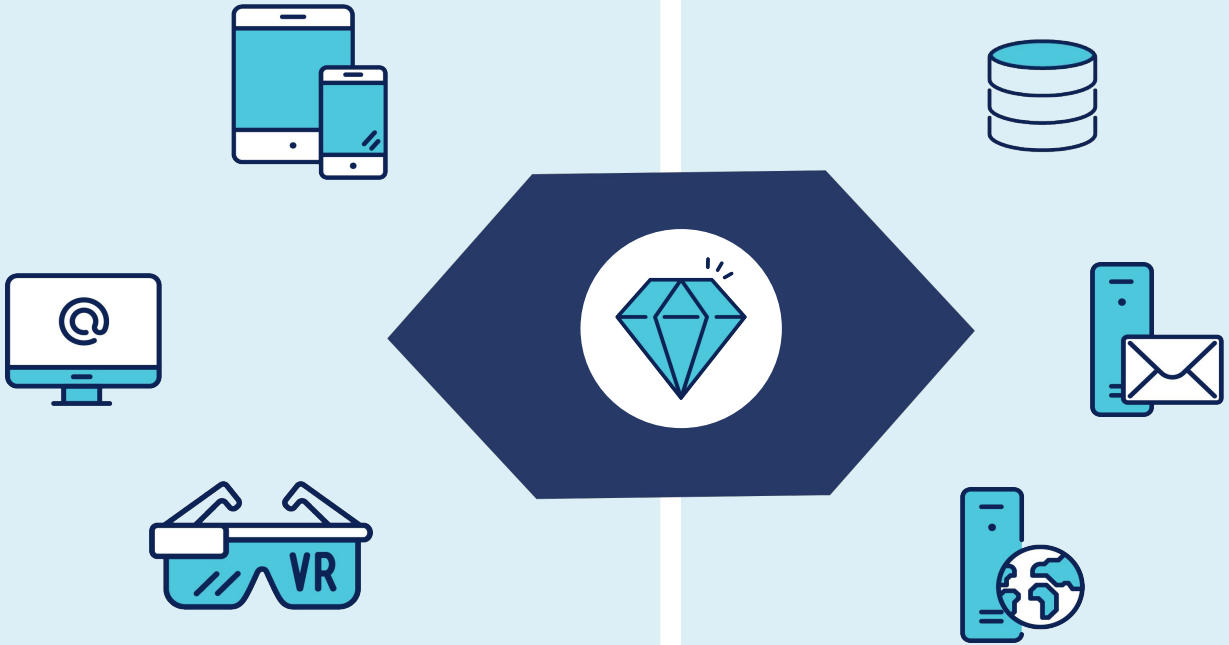
Calcul de l'impôt sur les revenus

Planning temps réel des chauffeurs de métro

Architecture hexagonale par Alistair Cockburn

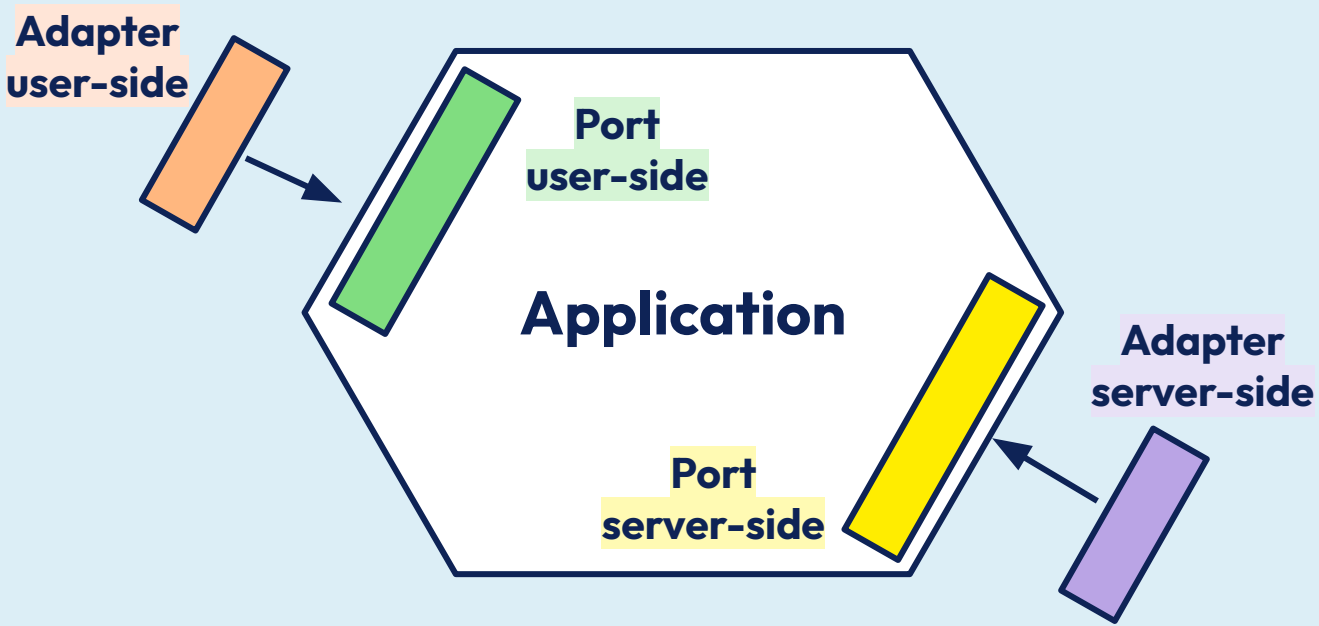
Blog d'Alistair Cockburn

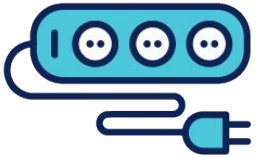
Isoler la valeur



Pourquoi un hexagone ?

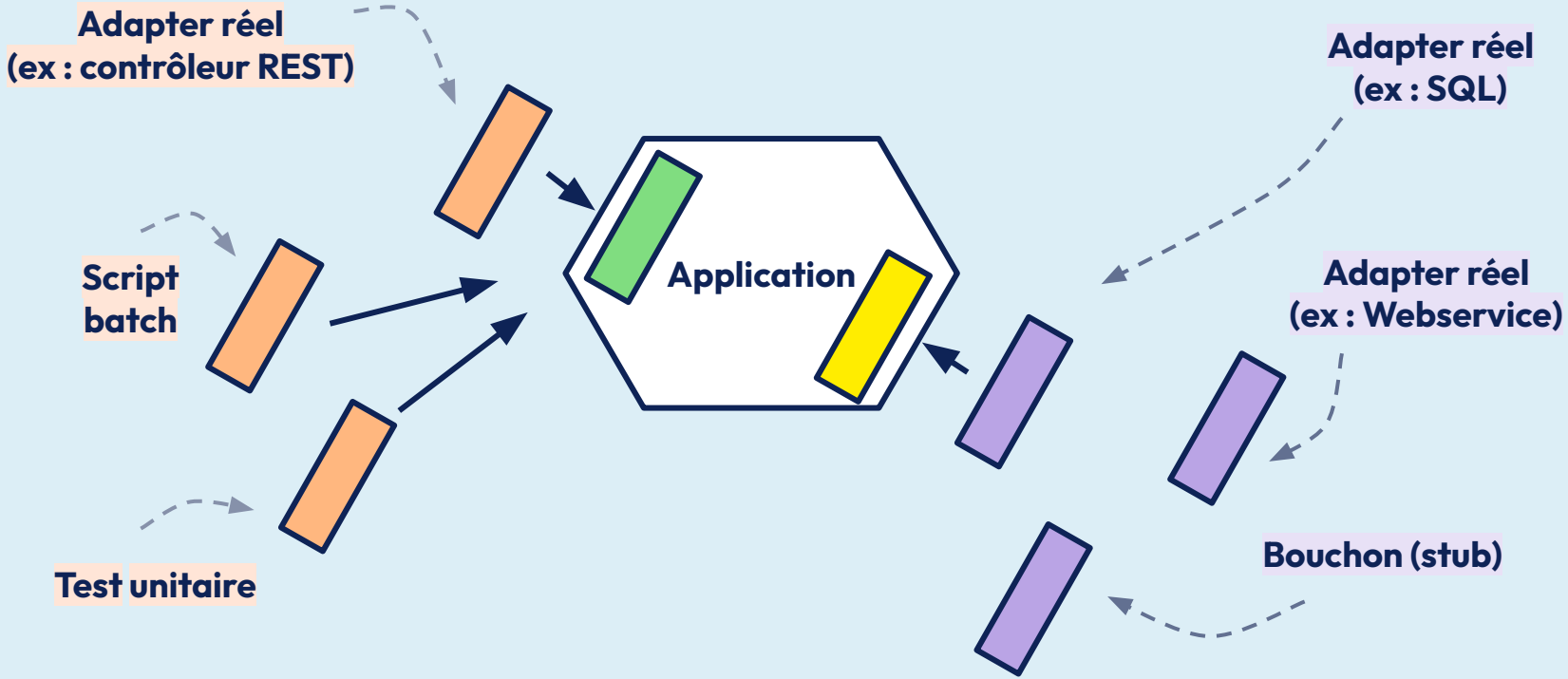
Architecture “port et adapter”



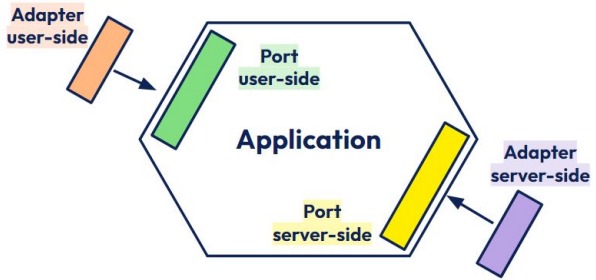


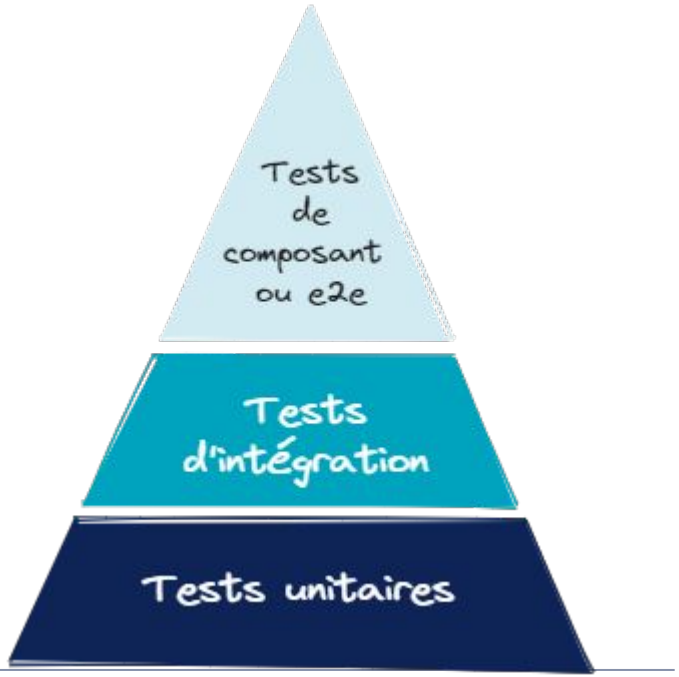
Formation
Design applicatif
d’Octo Academy

Un test est un adapter

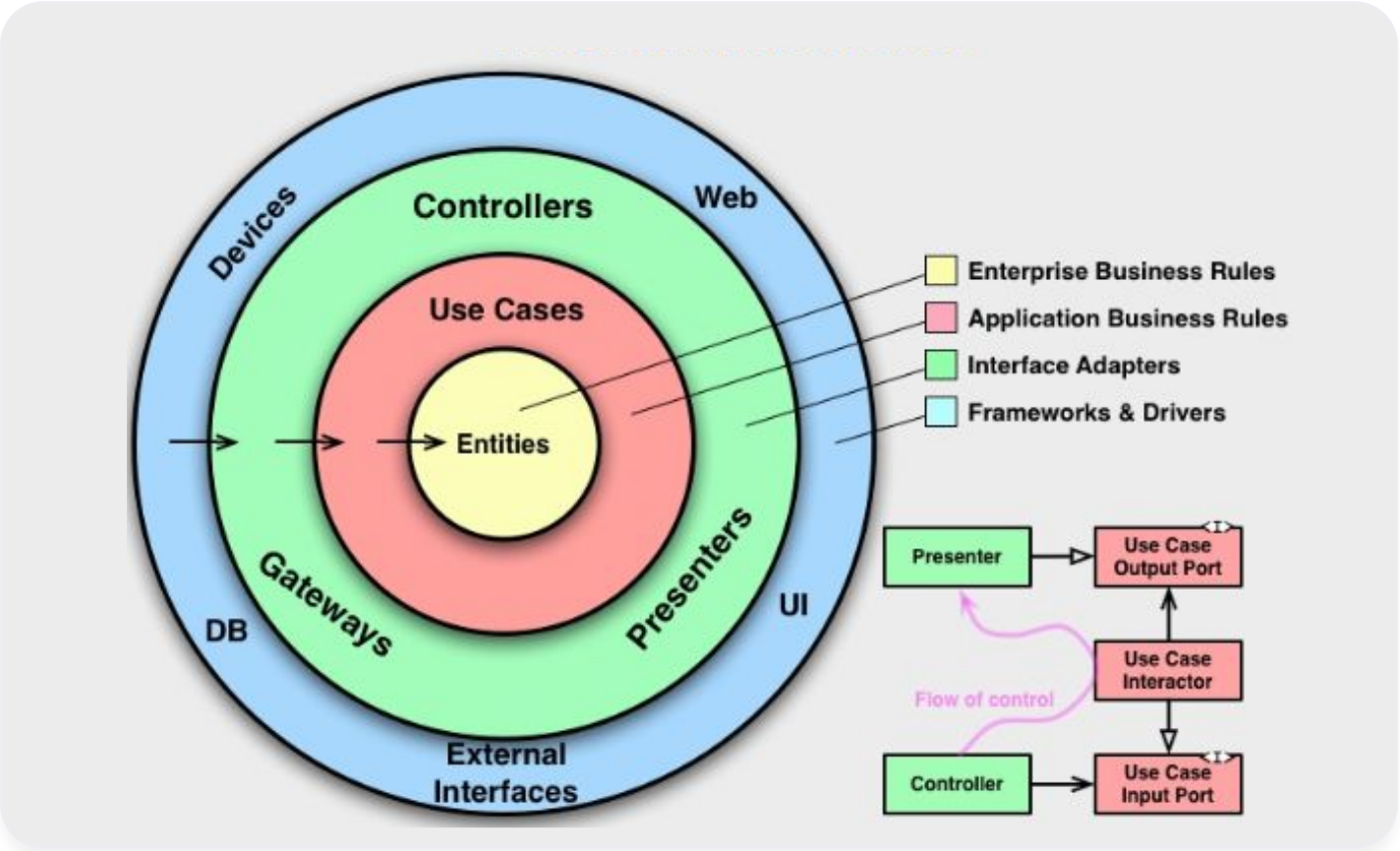


La stratégie de tests





D'autres patterns d'architecture



The *clean* architecture

Blog de Martin C. Martin

Architectures hexagonale & clean : *bonnet blanc, blanc bonnet ?*

LE COMPTOIR OCTO

DUCK CONF

Vidéo de 30mn

04

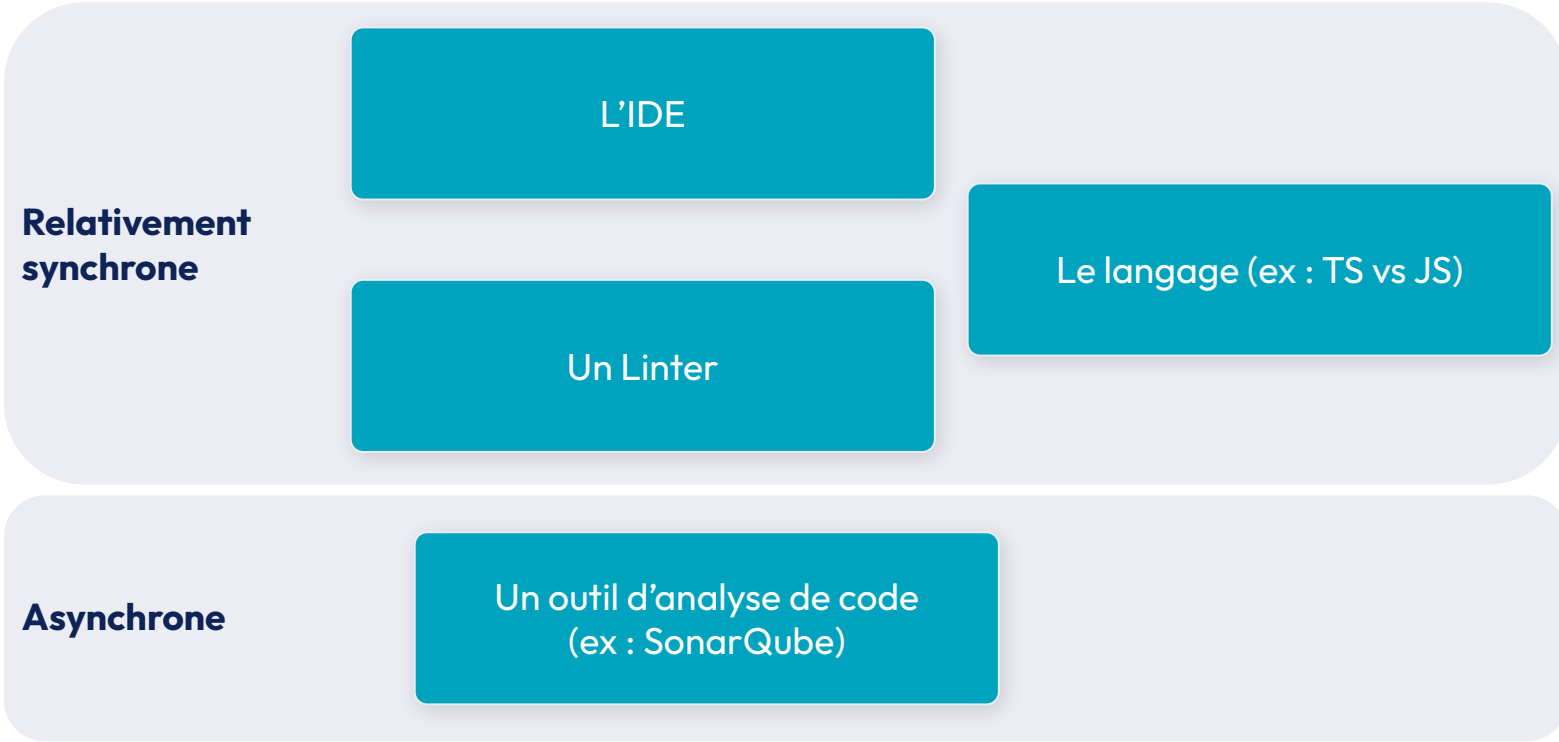
Et la partie statique ?

Définition

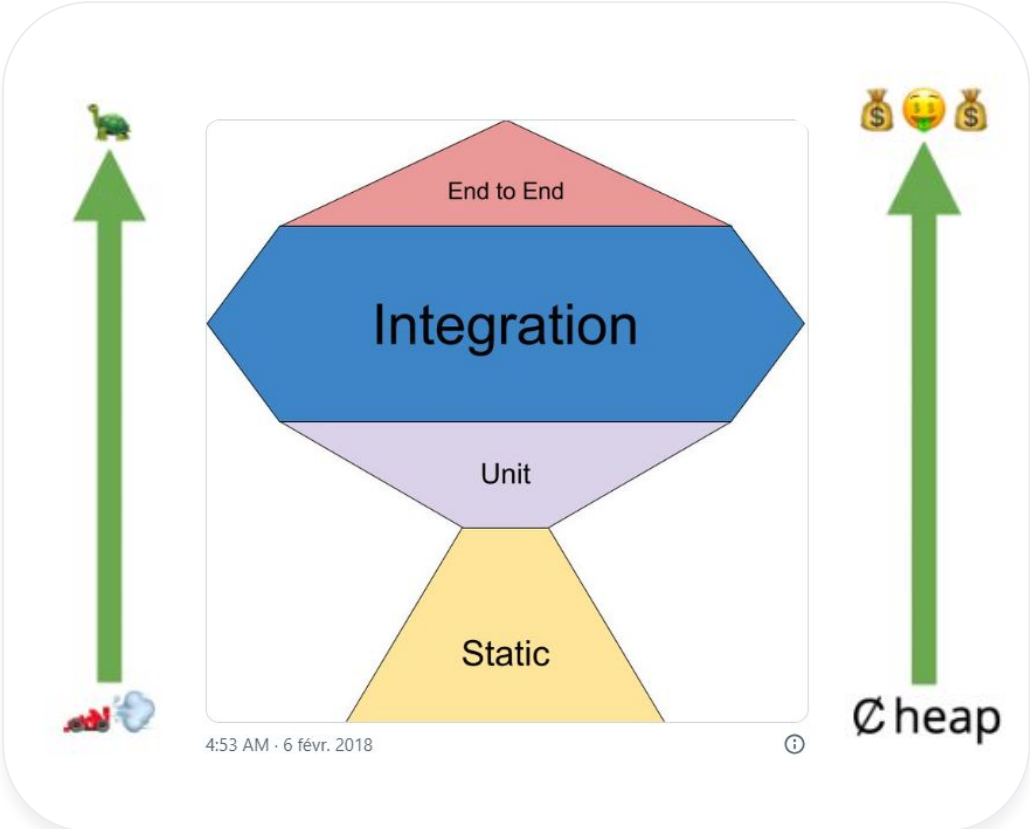
Les lignes
de code

Avant
l'exécution

Comment tester ?



Stratégie de test



The testing trophy and testing classification

There is a better way

05

Périmètre essentiel, non critique

Cas d'usage

Utile mais
pas le coeur
de votre système

Peut contenir
quelques règles métier

Structure de
données *simple*

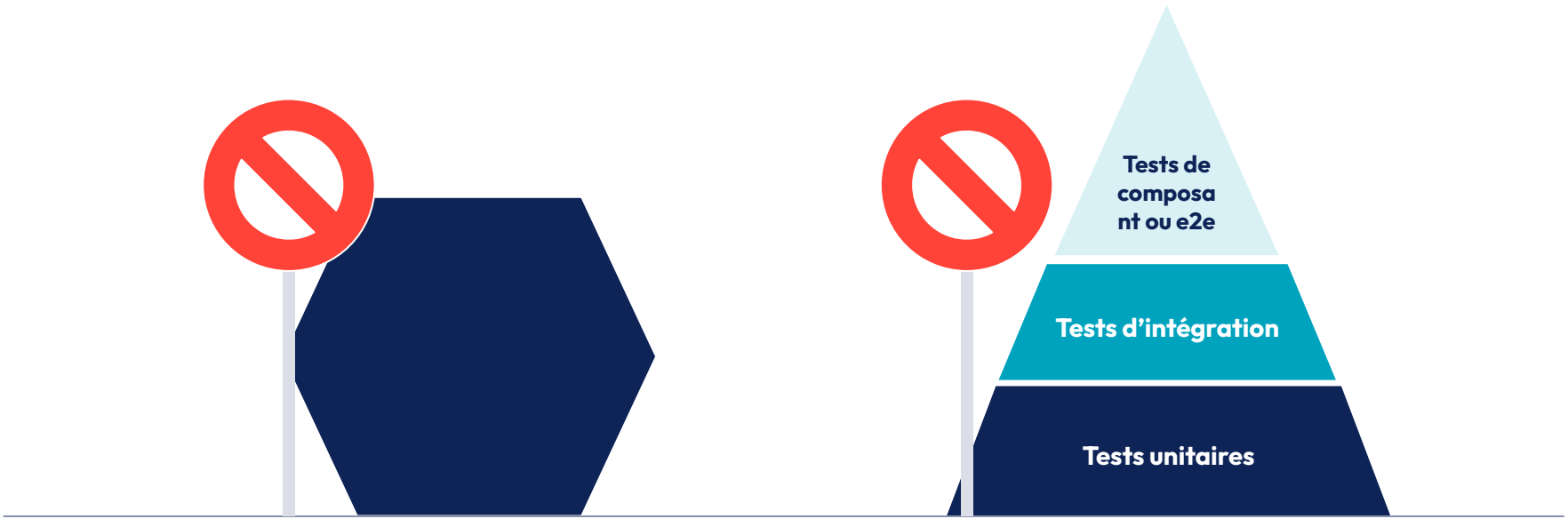
Exemples :

Un référentiel

Un écran d'administration

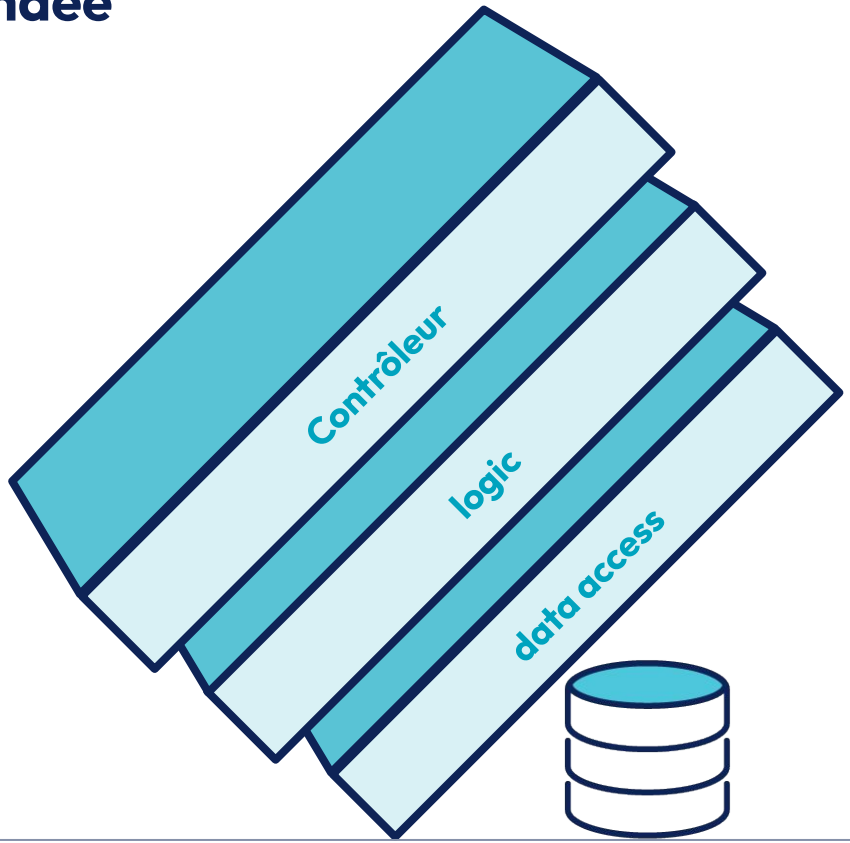
Un brouillon de dossier d'inscription

Architecture et stratégie de test non adaptées

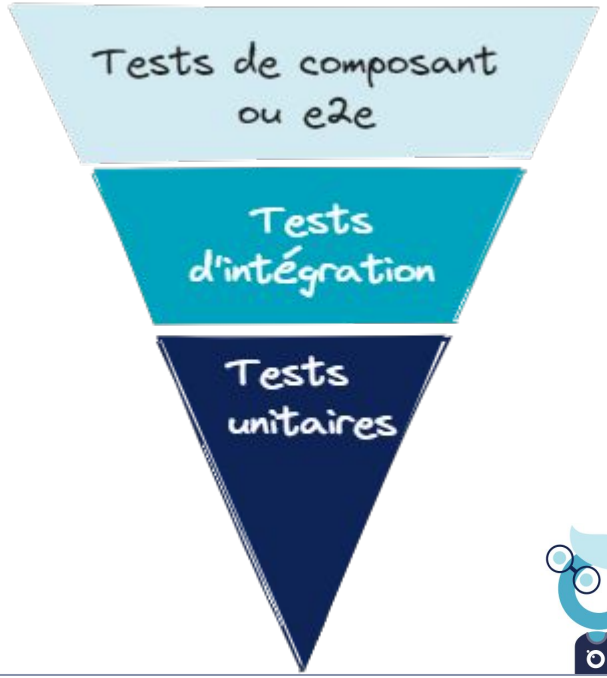


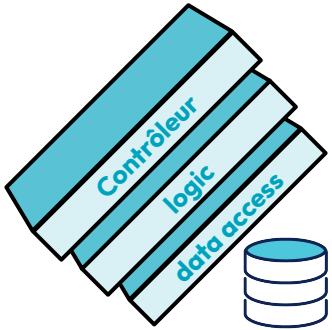
Architecture recommandée

Transaction
script



Stratégie de test recommandée





06

Périmètre orienté données

Cas d'usage

Interaction forte avec une source de données

Peut contenir *quelques règles métier*

Structure de données *complexe*

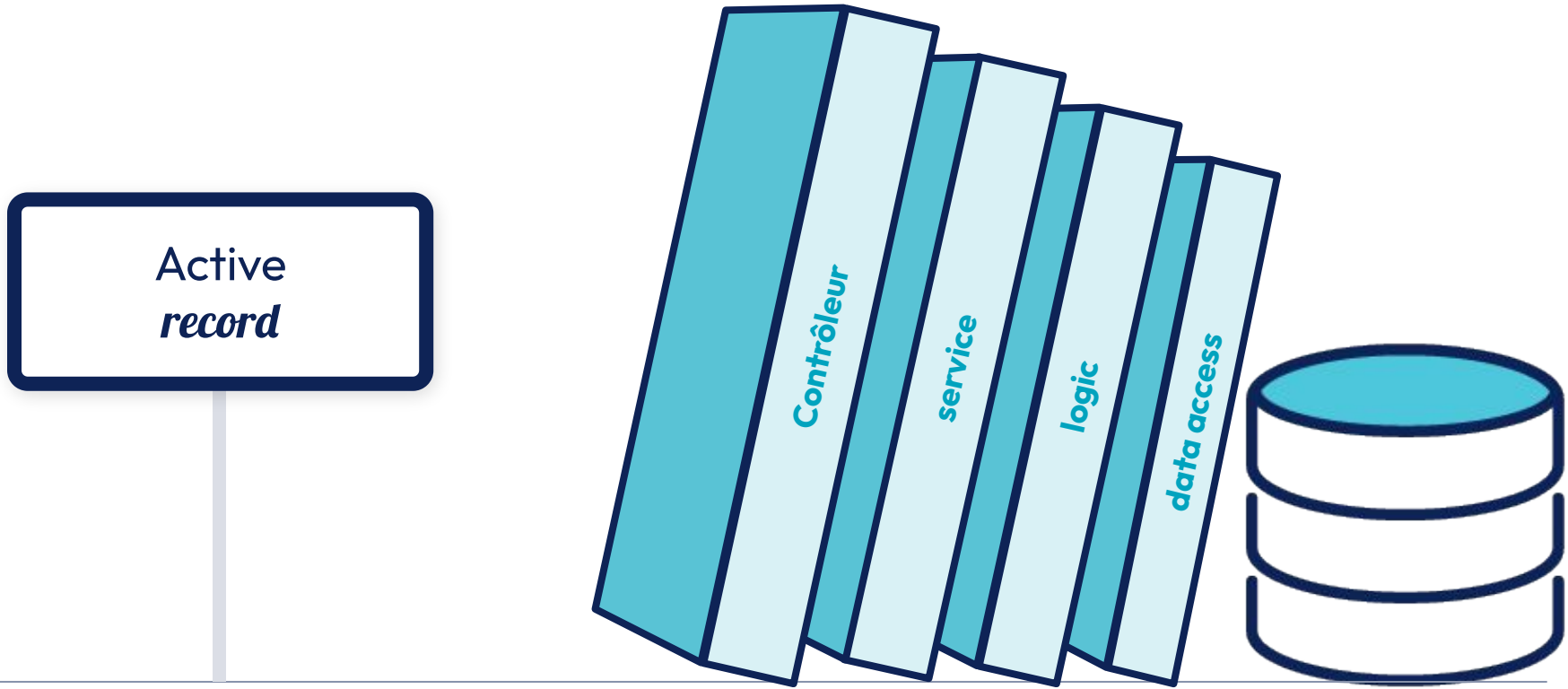
Exemples :

Filtrage et consolidation de données

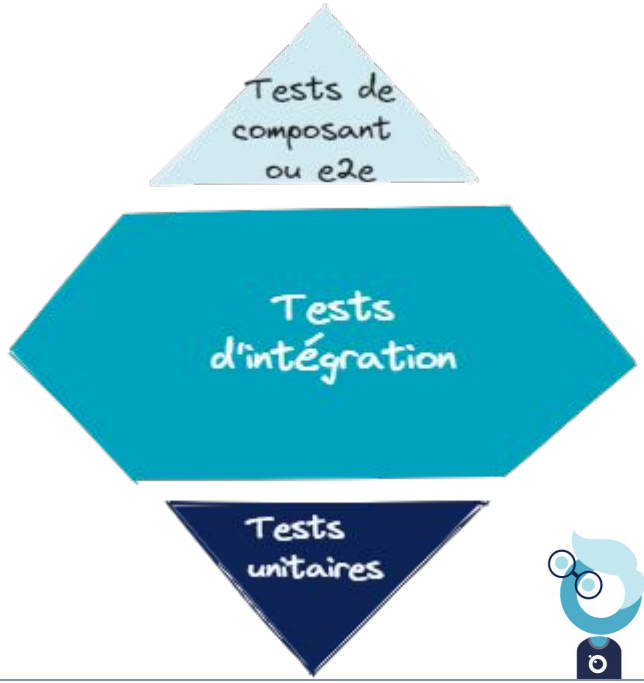
Génération de graphique

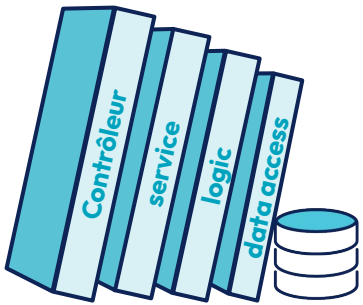
Génération de tableau de bord

Architecture recommandée



Stratégie de test recommandée





07

Besoin de lecture / écriture différent

Cas d'usage

Contraintes *différentes*

Eviter les *requêtes complexes*, se débarrasser *des jointures*

Deux *bases de données* optimisées

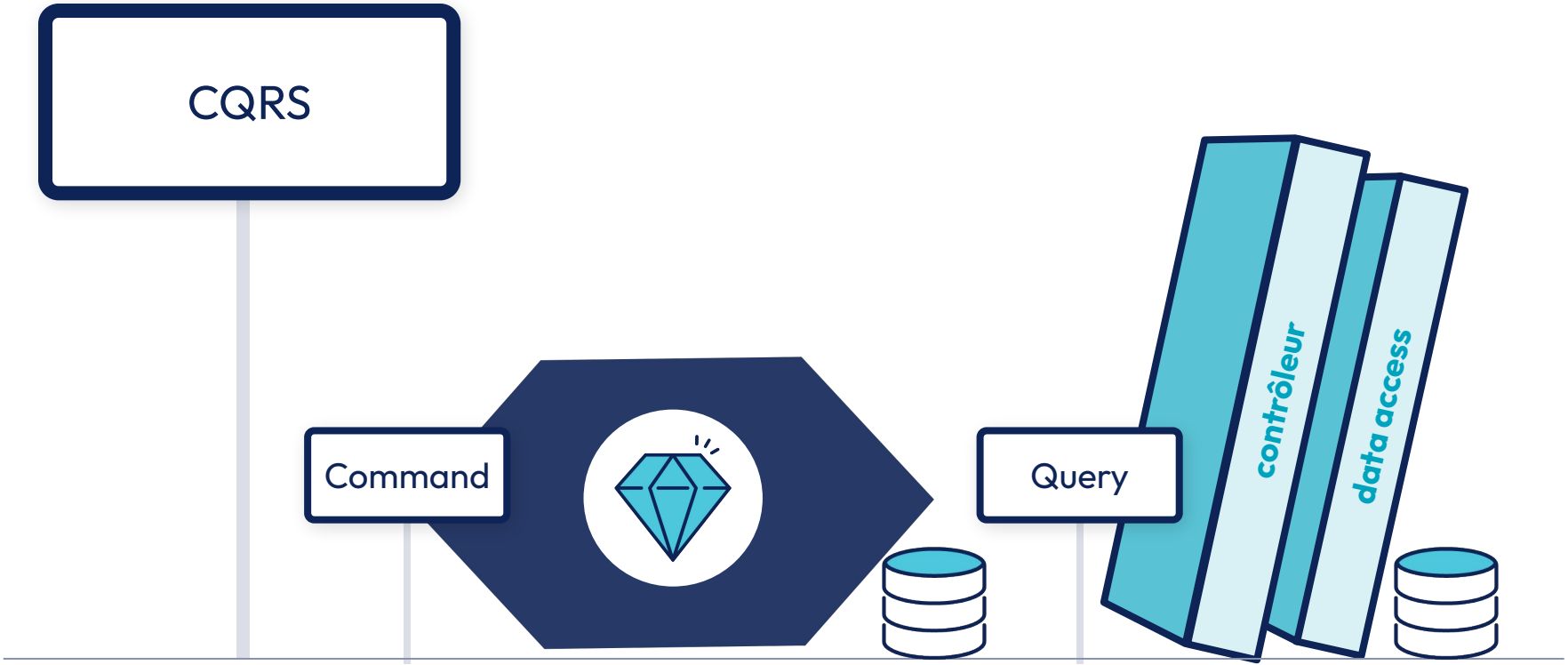
Exemples :

Billetterie pour concert

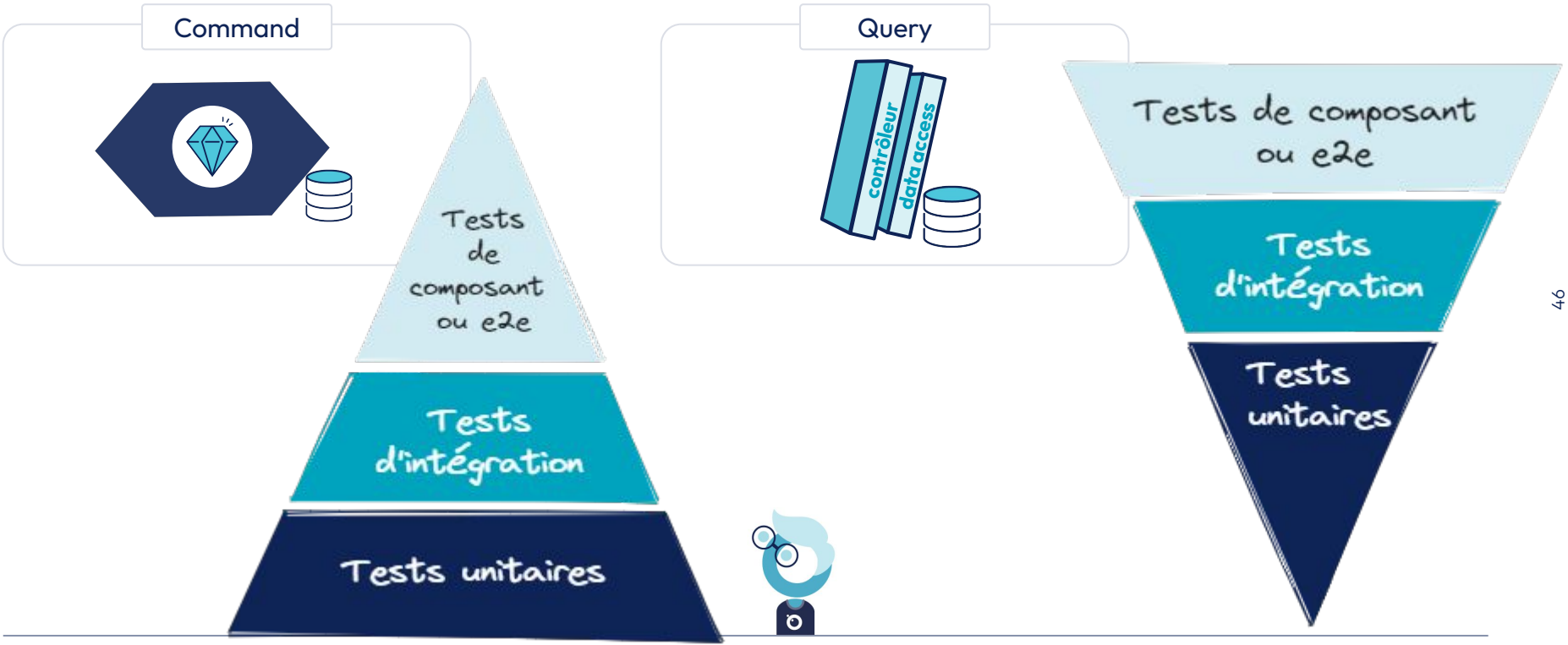
Déclaration de revenue

Service de géolocalisation de véhicule

Architectures préconisées



Stratégie de test recommandée



08

Conserver chaque changement d'état

Cas d'usage

Audit

Transaction financière

Rejouer / déjouer

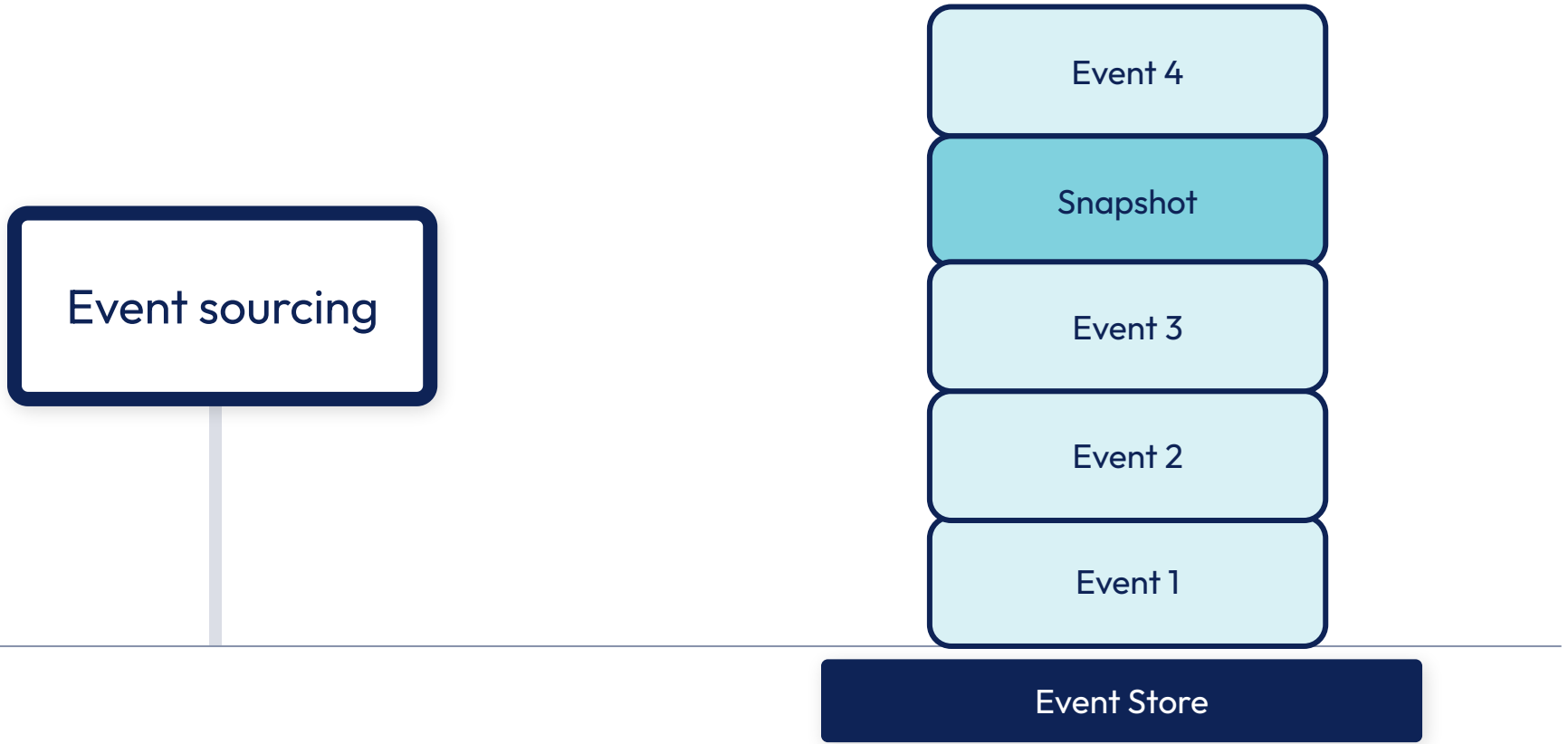
Exemples :

RATP et STIF

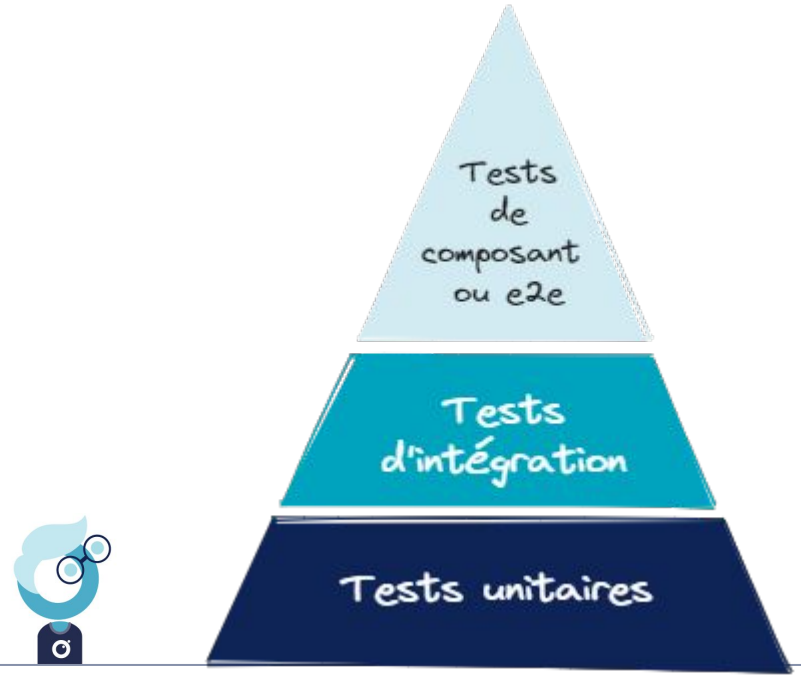
Retrait au DAB

Système de gestion de commande

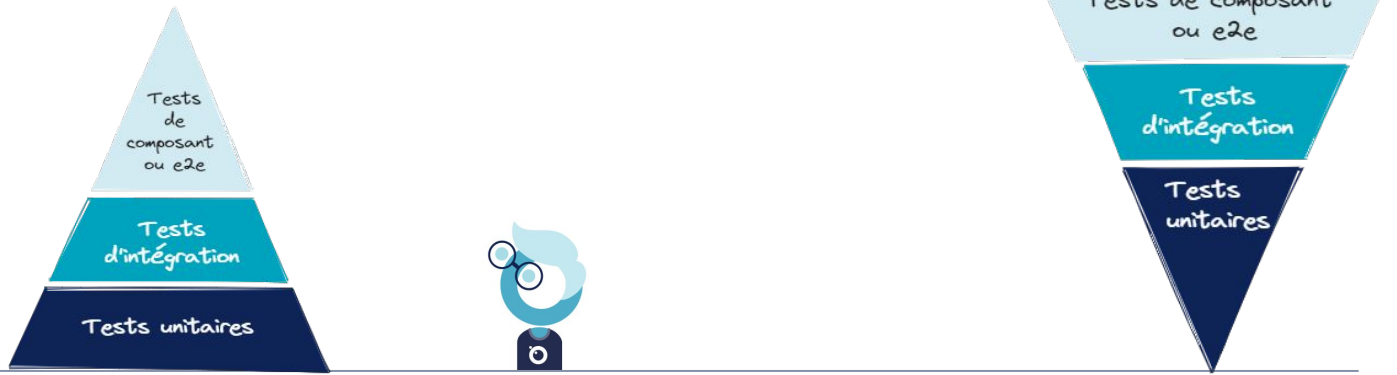
Architectures préconisées



Stratégie de test préconisée



Combiner les architectures



09

Les microservices

Cas d'usage

Découper ses équipes

Gagner en *autonomie*

Fluidifier la mise en production

Exemples :

Application conséquente	Adresse des problématiques différentes	Équipe nombreuse

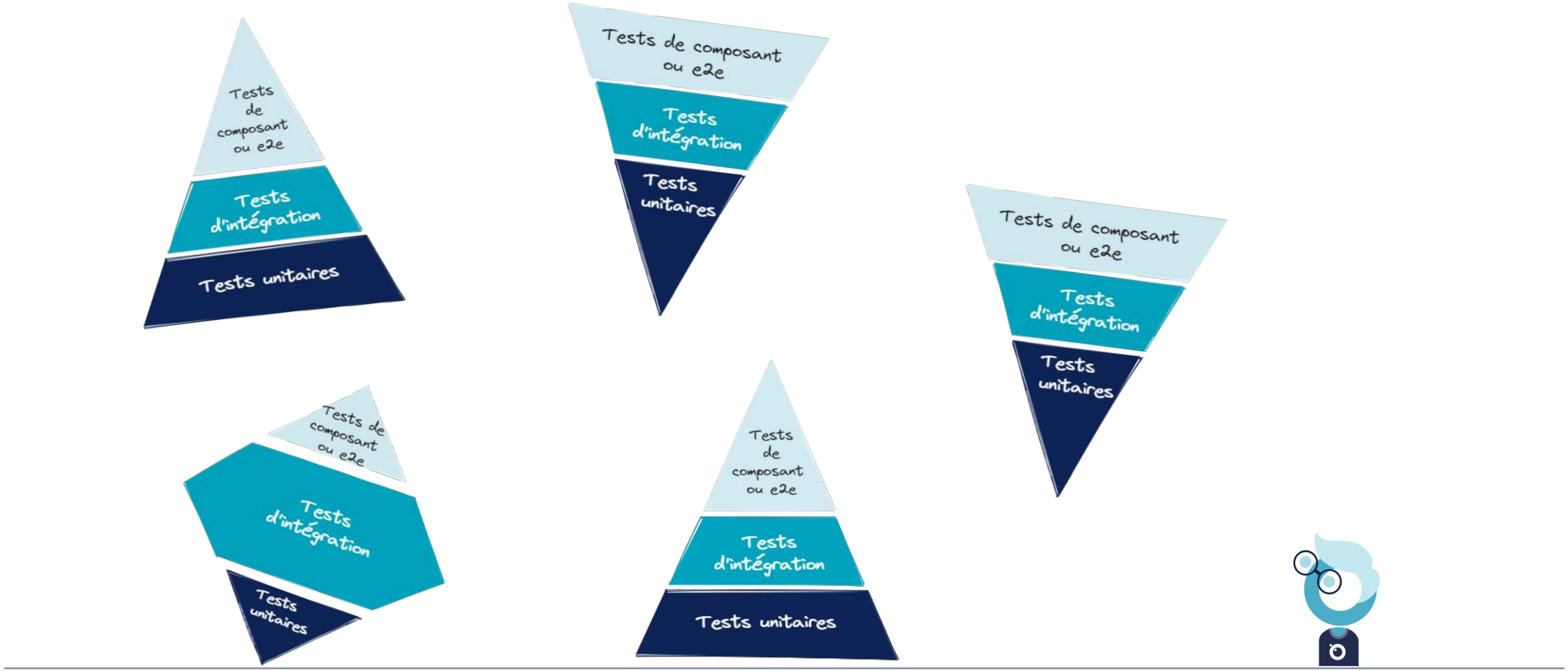
Stratégies unitaires

La stratégie se base sur :

- Une **conception modulaire**, permettant **l'autonomie réelle**
- **Découper en périmètre** applicatif (ex : event storming)
- Une **architecture** pour un **type de périmètre**
- Une **stratégie de test** par **périmètre**

Architectures préconisées au sein de chaque microservice

Stratégie de tests unitaires



There is a better way

Comment tester l'ensemble ?

Comment **évaluer** des **morceaux de l'application** pour être **confiant** sur le fait qu'ils **fonctionnent bien ensemble** ?

- Morceaux, développés potentiellement par des groupes de gens différents

Sans devoir attendre que tout soit **prêt** ?

- Ne pas perdre l'avantage des microservices en terme d'**autonomie**
- **Tester** la communication entre **producteurs** et **consommateurs** (aka **publisher/subscriber**)

Présence de risques

- Introduction de **dépendances**
- **Perte de productivité**, **feedbacks lents**
- **Casse facilement**
- Présence d'un **environnement d'intégration** qui **bloque** les mises en **production**

Stratégie de tests isolés

Contract Testing

- Contract : **interface** entre provider et consumer
  - **Consumer-Driven**
  - **Provider-Driven**

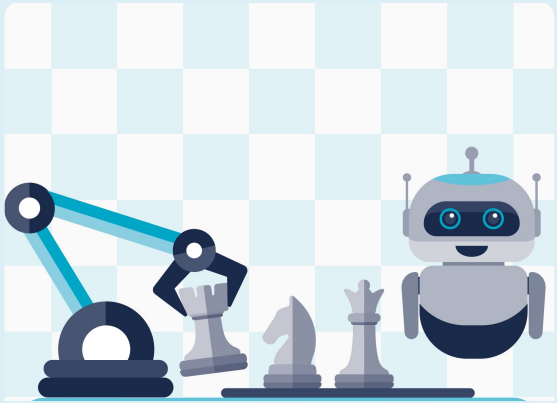
- **Tests automatisés isolés** à écrire de part et d'autre

- **Avantages**
  - Exécutions **indépendantes**, pas d'environnement d'intégration
  - **Feedbacks** rapides
  - **Moins complexe** à maintenir
- **Inconvénient** :
  - Garantir la **confiance** sur l'**ensemble de la chaîne ?**

Stratégie de tests collaboratifs

- **Consumer-driven contract** (ex : solution “Pact”) :
  - **Consumer test** : tests d’intégration sur un **Mock provider**
    - Chaque requête est enregistrée dans un fichier représentant le contrat
  - **Provider test :** à partir du **contrat**, un **consumer simulé** (ex : par Pact) rejoue **chaque demande** auprès du **provider**
    - Comparaison des réponses réelles et attendues
  - **Inconvénient**
    - **pas de développement parallèle** entre le consumer et le provider
- **Provider-driven** (ex : solution “Spring Cloud Contract”)
- **Bi-directionnel** (ex : solution “Pactflow”)
- **Mettre en place certains mécanismes**
  - Comment **partager les contrats ?**
  - **Alerte** en cas de **changement de contrat ?**

Contract-Driven Development



Contract-Driven Development

- API specification (first) : **contrat exécutable**

Ex: solution “Specmatic”

- **Contract as Stub** pour le consumer (à partir d’un contrat OpenAPI #NoCode)
- **Contract as Test** pour le provider (tests générés)

Autres avantages

- **Consumer-Driven ou Provider-Driven**
- **Vérification** de **compatibilité ascendante** (#NoCode)
- **Pas de paramétrage** serveur
- **Utilisable** pour projets **Brownfields**
- Etc.

Pact vs Patflow vs Specmatic

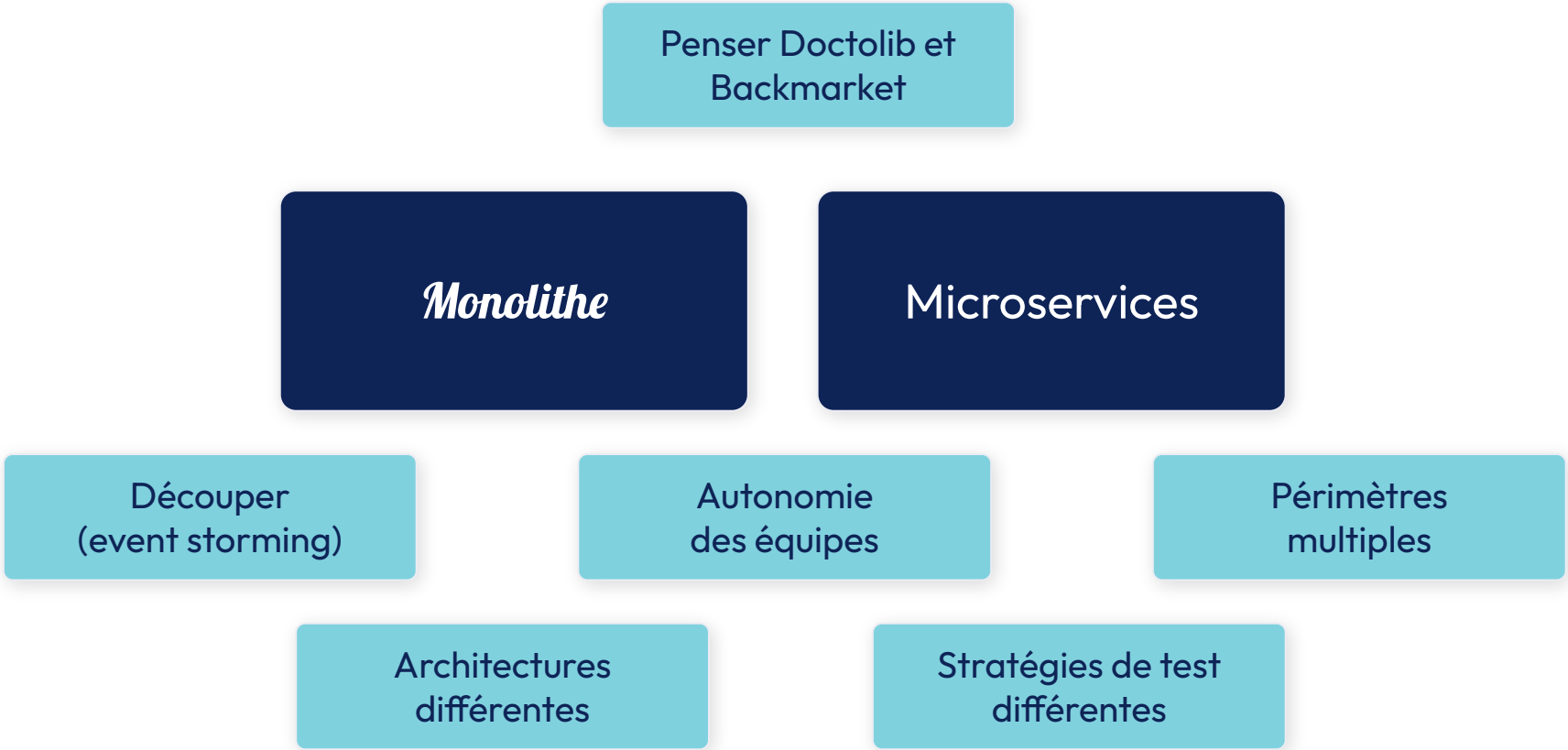
10

Conclusion

Fred Brooks
Computer architect, software engineer

> “There is no *silver bullet*”

Des conceptions logicielles qui cohabitent



Ralph Waldo Emerson
Essayiste, poète, philosophe

> “L’idéal de la vie n'est pas l'espoir de devenir parfait, c'est la ***volonté d'être toujours meilleur***”

Merci !

There is a Better Way

OCTO Technology
Part of Accenture