Bonjour tout le monde

Test Driven Development

→ Devenu

Behavior Driven Development

approche technique

Pour objectif d'être compréhensible

Cycle de vie

→ Besoins utilisateur (fonctionnel)

→ Écrire des specs et la validation

(cachet UML)

Coder

Tester (recette) → Intégration

End-to-End fonctions.

TDD

→ Debord d'écrire les tests

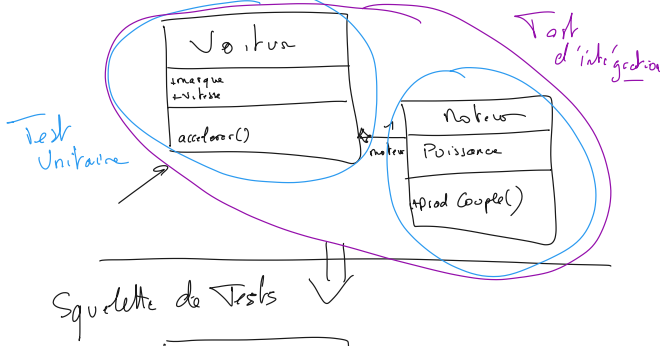
① Analyse l'UML → Diag de classes

squelette de code ② | squelette de Tests ④

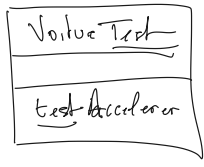
② implémenter le code

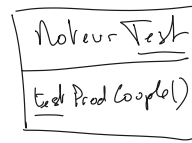
④ implémenter les Tests

③

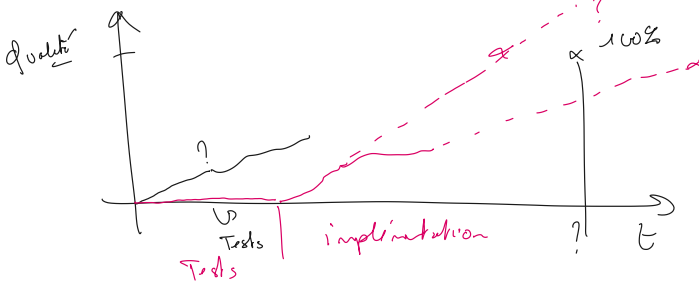


Squelette de Tests





classes de Test Unitaires



Sécurité

Sécurité

→ Failles (exploit)

→ Dépendances (packages tiers)

ex.. log4j (log4shell)

↳ version?

Common Vulnerability & Exposures.

https://cve.mitre.org/

https://nvd.nist.gov/

Tests via des logiciels (compare les libs avec la liste des CVEs)

→ SonarQube / SonarLint

→ Snyk https://rules.sonarsource.com/

https://cwe.mitre.org/

→ OS (hardening)

https://learn.cisecurity.org/benchmarks

→ Developpeur

→ Voir Référentiels de bonne pratiques

Web.. OWASP / Top 10

général: CWE Common Weakness Exposures.

https://owasp.org/Top10/fr/

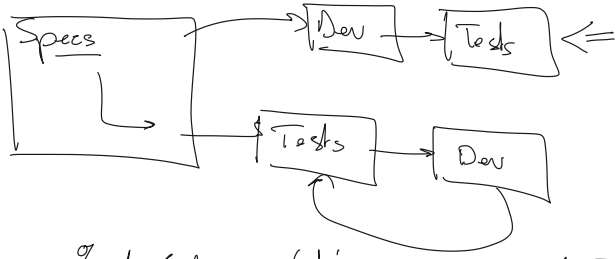
https://cwe.mitre.org/

Types de tests :

• Accessibility testing
• Acceptance testing
• Black box testing : tester un système inconnu.
• White-box testing : teste les entrées sorties d'un logiciel bien connu, concernant le design, l'utilisabilité, la sécurité, la stabilité.
• End to end testing : tester un workflow fonctionel
• Functional testing : s'assurer qu'il fait exactement ce qui était prévu.
• Interactive testing : eq tests manuels
• Interface tests
• Integration testing : test en environnement équivalent à la prod dans son ensemble.
• Load testing
• Non functional testing : catégorie de tests d'interface (conformité), accessibilité, performance, ...
• Performance testing : recherche des meilleurs métriques, benchmark.
• Single user performance testing
• Non-Regression testing
• Sanity testing : contrôle que les bugs ciblés sont bien corrigés
• Security testing : contrôle face aux menaces
• Smoke testing : valider les fonctions critiques d'un système, envers la stabilité.
• Stress testing : test selon des conditions exceptionnelles de charge, au dela des specs.
• Unit testing - Implémenter le pattern Facade
• Property-based Testing : valider les contraintes appliquées aux contraintes des propriétés.

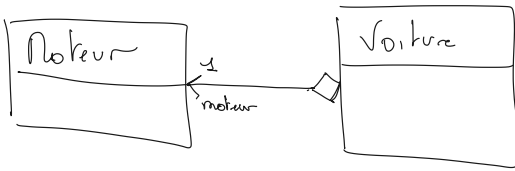
3 Ways to Test
There are 3 ways you can do testing.
• Manual testing is the most hands-on type of testing and is employed by every team at some point. Of course, in today's fast-paced software development lifecycle, manual testing is tough to scale.
• Automated testing uses test scripts and specialized tools to automate the process of software testing.
Continuous testing goes even further, applying the principles of automated testing in a scaled, continuous manner to achieve the most reliable test coverage for an enterprise. Keep reading to learn more about the differences between automated testing vs. manual testing and how continuous testing fits in.

La Couverture de Code



→ % de Code exécutés au moment des Tests

→ Statistiques → nb d'entrées de Méthode (hotspots) | Temps moyen connu / Méthode



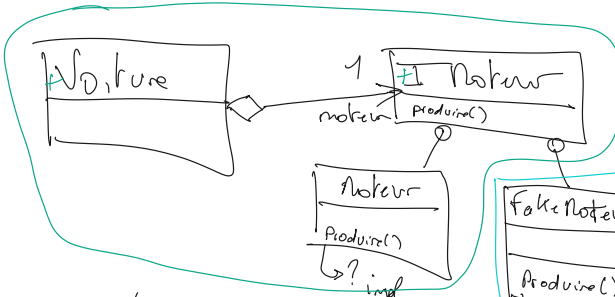
TU Moteur → aucune dépendance de la Voiture

Voiture → dépends du Moteur !

S. pbm Moteur = pbm Voiture.

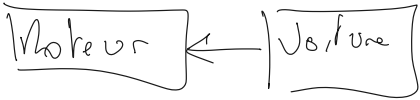
Implémentation des TU

Composant Principal

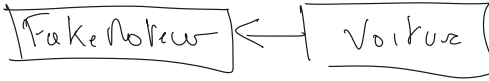


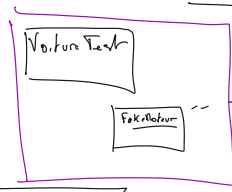
Diag d'instance

Dev



Test





Patterns d'archi

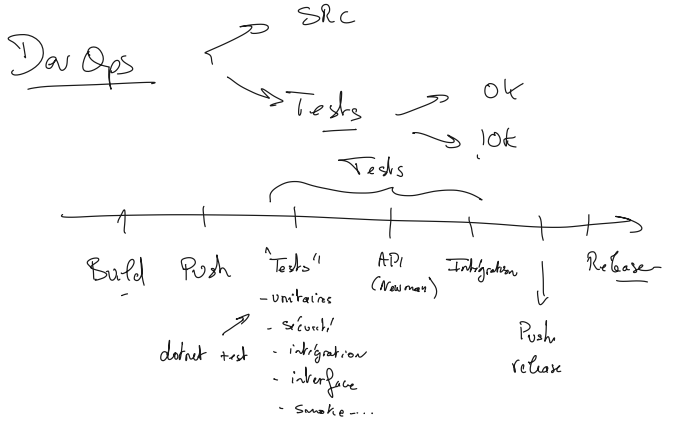
MVC → Asp.net MVC
ORM → EF, nHibernate
AOP → Aspect oriented Prog.
IoC → Inversion of Control

• ("**test doubles**" : copie des classes dont on connait le comportement) : (des doublures, comme au ciné)
  ○ **Dummy**
  ○ Classe qui renvoie toujours null en membres et ne fait rien du tout.
  ○ **Stub**
  § Est une classe qui réponds exactement à ce que j'attends (forme de dépendance)
  § **Spy**
  ○ Est un stub qui enregistre ce qui a été exécuté pour contrôler.
  ○ **Fake**
  § Forme de mini implémentation de la classe, maquette
  § Plus intelligent que le Stub. Mais réussira toujours (car simple)
  § Exemple : La classe Fake implémente une base de données inMemory ou un tableau, qui fonctionnera toujours, et l'écriture s'y fera vraiment.
  § Il n'y aura jamais d'erreurs.
  ○ **Mock** : Objet plus intelligent qu'un Fake ou Stub, dans le sens ou il peut décider de lever des exceptions lui-même si le contenu passé est incorrect.

• **Shim** : Classes qui encapsulent des Dll System, afin de s'en prémunir.
  ○ Exemple : Tester le bug de l'an 2000. On intercepte l'appel système vers notre code qui engendre le bug, qui sera alors testé.
• **Fixtures**
Jeux de données pour les tests

Exercice :
Ecrivez l'implémentation du diagramme de classes concernant la voiture et la doublure moteur.
Implémentez la doublure du moteur pour pouvoir implémenter des tests
L'accélération de 10 doit, à travers le moteur, incrémenter la vitesse de 10.
L'implémentation normale du moteur est beaucoup plus complexe (état du moteur, carburant, etc... dont on se passe pour les tests).
L'objectif étant de tester la voiture, pas le moteur.

https://docs.microsoft.com/fr-fr/visualstudio/test/using-shims-to-isolate-your-application-from-other-assemblies-for-unit-testing?view=vs-2022&tabs=csharp



Projet

Projet de Tests → Tests Unitaires

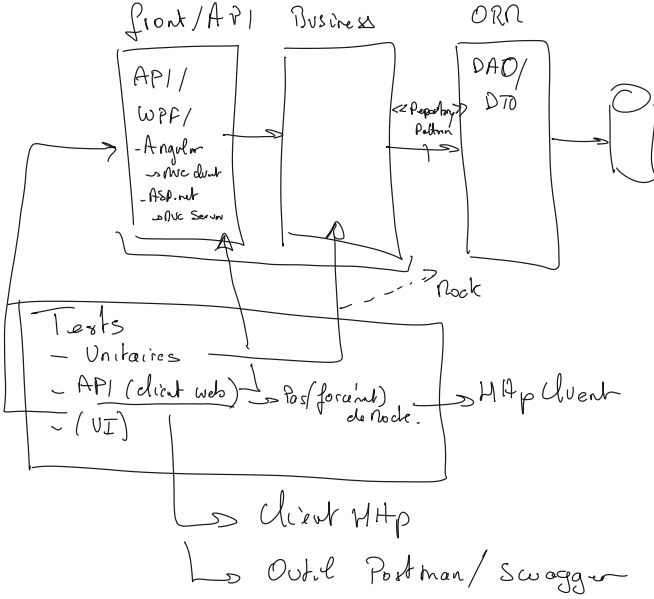
→ Suites de Tests
↳ une Piste de Tests unitaires (dans un certain ordre)

→ Tests de sécurité

→ interface UI (si dispo)

→ Intégration (sans Mocks)

etc ...


front / API
Business
ORM
API /
WPF /
- Angular
-> MVC client
- ASP.net
-> MVC Server
<< Repository Pattern >>
DAO /
DTO
Mock
Tests
- Unitaires
- API (client web)
-> Post (forcément de Mock)
-> HTTP Client
- (UI)
Client HTTP
Outil Postman / Swagger

Web Services → Service SOAP XML ~ 2000

→ XML

↓

DOM / Parser

Trés lourd (CPU)

mobiles

JSON

↓

Sérialisation / Désérialisation

→ API Rest → JSON XML etc...

Cadre / normes ?

W3C → Protocole

Swagger →
- → outils
- → éditeurs
- → bonnes pratiques

→ décrire une API (l'équivalent du WSDL)

"Specs swagger → v1 "
v2

→ "Open API v3" → écriture en ligne

→ outils → Tester .....

Swagger

↓

Specs

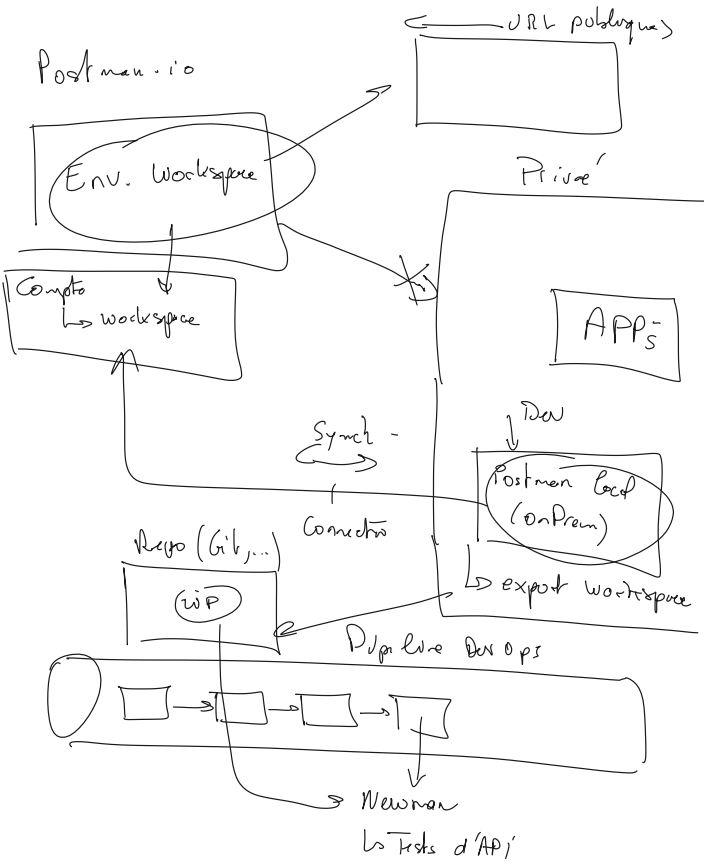
Postman

↓

Test d'API
- → Campagne de Tests
- → intégration pipeline NewMan

RAML

Specs


URL publique
Postman.io
Env. Workspace
Privé
Compte
↳ workspace
APPS
Dev
Synch -
Postman Local
(onPrem)
Connecté
export workspace
Repo (Git, ...)
WIP
Pipeline DevOps
Newman
↳ Tests d'API

Postman :

- Avoir une API (et créer un compte et obtenir une clef si nécessaire)
  - https://github.com/public-apis/public-apis
- Créer un Workspace
- Y intégrer un appel d'API
- Ecrire un test de validation
  - De code de retour 200
  - De contenu dans la réponse (utiliser les snippets)

Formateur : exporter le workspace et l'envoyer à newman pour consulter le code de retour (ok/pas ok) - intégré dans un pipe devops.

Mise en Pratique :

- Développement de Projets .net 5/6
  avec Projet Library et Projet de Test xUnit
- Implémentation de ≠ Tests :
  - Unitaire
  - Intégration
  - Sécurité
  - → Lancement Scénario

https://docs.microsoft.com/fr-fr/dotnet/core/testing/unit-testing-with-dotnet-test

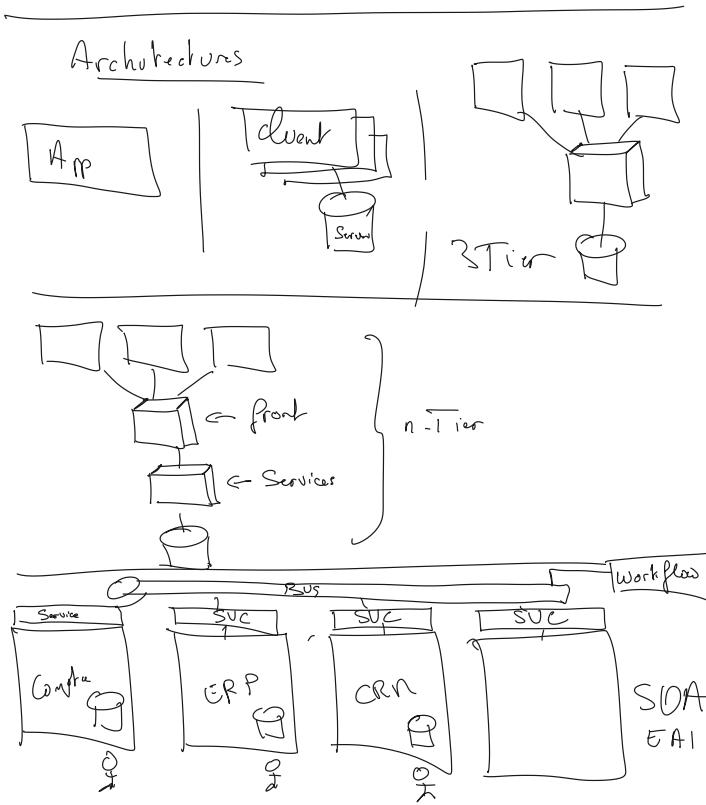
Exercice : spécifiez vos Tests, et lancer conditionnellement :
https://docs.microsoft.com/fr-fr/dotnet/core/testing/selective-unit-tests?pivots=xunit

Sélection d'une méthode :
dotnet test --filter DisplayName=Prime.UnitTests.Services.PrimeService_IsPrimeShould.IsPrime_InputIs1_ReturnTrue

Tests d'API Asp.Net
https://docs.microsoft.com/fr-fr/aspnet/web-api/overview/testing-and-debugging/unit-testing-controllers-in-web-api

Utilisation d'un client Http :
https://docs.microsoft.com/fr-fr/dotnet/csharp/tutorials/console-webapiclient

Exercice :
Implémenter des tests en implémentant un HttpClient :
https://docs.microsoft.com/fr-fr/dotnet/csharp/tutorials/console-webapiclient



>2000 → Internet 3V

Vitesse → Trop de débit

Volume → Trop de données (en taille)

Variabilité → Trop de modif de schéma

Relationnel → hors course

↳ Solutions NoSql (Big Data)

- Cassandra
- MongoDb
- Redis
- ElasticSearch
- Kafka
- etc....

→ La Normalisation → Merise

Relations entre Tables

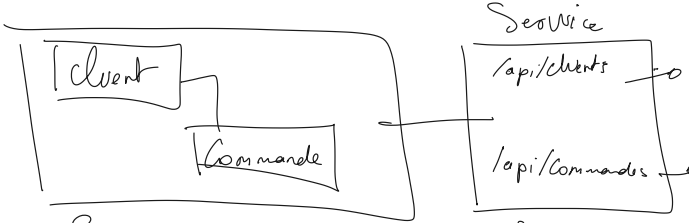
Jointures

Contraintes

↓

Un anti Pattern ! en Big Data

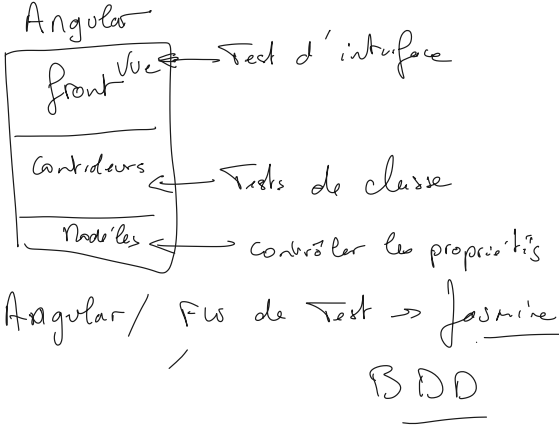
- Relationnelle



µService

Contraintes ?

Relation ?

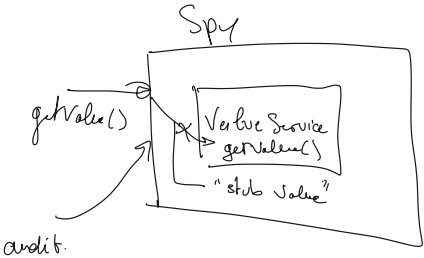


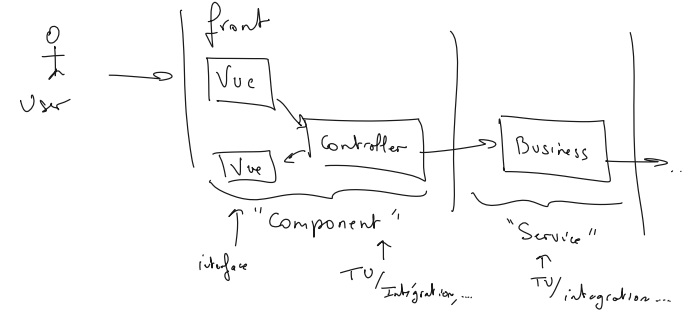
---

https://angular.io/guide/setup-local

Intégration de l'équivalent de Live Unit Testing et Test explorer pour VSCode / Angular : https://marketplace.visualstudio.com/items?itemName=lucono.karma-test-explorer
(merci Morgan)

Execution des tests hadoc (non continu) pour une boucle CI par ex :
npm run test -- --no-watch --no-progress
ou : ng test -- --no-watch --no-progress [--browsers=ChromeHeadlessCI]





Cas des Service :
https://angular.io/guide/creating-injectable-service

ng generate service heroes/hero

Implémentation d'un Stub :

```
beforeEach(() => {
    TestBed.configureTestingModule({});
    //service = TestBed.inject(MasterserviceService);
  });

it('#getValue should return \'stub value\' ( UnitTest with Stub )', () => {
    let masterService: MasterserviceService;

    // Préparation du Stub
    let valueServiceSpy: jasmine.SpyObj<ValueServiceService>;
    const spy = jasmine.createSpyObj('ValueServiceService', ['getValue']);
    TestBed.configureTestingModule({
      // Provide both the service-to-test and its (spy) dependency
      providers: [
        MasterserviceService,
        { provide: ValueServiceService, useValue: spy }
      ]
    });
    valueServiceSpy = TestBed.inject(ValueServiceService) as
jasmine.SpyObj<ValueServiceService>;
    const stubValue = 'stub value';
    valueServiceSpy.getValue.and.returnValue(stubValue);

    masterService = TestBed.inject(MasterserviceService);

    expect(masterService.getValue())
      .withContext('service returned stub value')
      .toBe(stubValue);
  });
```