2 - C'EST QUOI UN TEST AUTOMATISÉ ?

R4.02 - Qualité de développement

Institut Universitaire de Technologie de Bayonne et du Pays Basque

Pantxika Dagorret - Damien Urruty

BUT Informatique 2022 / 2023 - Semestre 4

TESTS AUTOMATISÉS

- Du code...
- ... qui permet de vérifier le comportement d'un autre morceau de code...
- ... et qui doit forcément réussir ou échouer (résultat clair)
- Un test automatisé se suffit à lui-même (pas besoin d'intervention humaine)

TESTS AUTOMATISÉS

- Il existe beaucoup de types de tests différents
- Attention : certains termes représentent parfois le même type de test
- Attention : les définitions de chaque type de test peuvent varier d'une personne à l'autre

TESTS AUTOMATISÉS

- L'important avant d'écrire les tests est de réfléchir aux cas d'utilisations
- Qu'est censé faire ce code/composant/produit dans une situation donnée ?
- Peut-il y avoir des scénarios alternatifs, des cas d'erreur ?

TESTS AUTOMATISÉS - EXERCICE

COMBIEN DE SCÉNARIOS POSSIBLES POUR CETTE FONCTION ?

```
// example: 1101 va retourner 13
int binaireVersDecimal(String binaire)
```

STRUCTURE D'UN TEST AUTOMATISÉ:

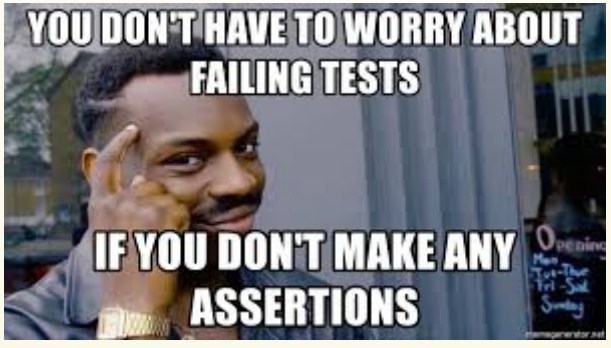
- Etant donné *(Given/Arrange)* un état initial
- Lorsque *(When/Act)* on réalise quelque chose
- Alors *(Then/Assert)* on vérifie l'état final

ASSERTIONS

S'UTILISENT LORS DE L'ÉTAPE THEN/ASSERT

VÉRIFICATION D'UNE CONDITION, D'UN ÉTAT PARTICULIER

AU MOINS UNE ASSERTION PAR CAS DE TEST !



TYPES DE TESTS

- Tests unitaires
- Tests d'intégration / service
- Tests end-to-end
- Tests de fumée *(Smoke test)*
- Tests de performance
- ...

TESTS UNITAIRES

- Vérifient le bon fonctionnement d'un morceau de code (unité)
- Ecrits avec du code

TESTS UNITAIRES - CARACTÉRISTIQUES

- S'exécutent très rapidement (quelques ms par test -> plusieurs secondes au total)
- Ecrits pour les programmeurs, par les programmeurs
- En général testent de la technique mais peuvent tester du fonctionnel
- Test un petit montant de code : en général méthode/fonction ou classe
- Pas de consensus sur ce qui est une "unité"

EXEMPLE (EN LANGAGE NATUREL) : TWITTER

- Etant donné un grand nombre, 30000000
- Lorsqu'on le condense
- Alors on s'attend à obtenir 30M

EXEMPLE (EN PSEUDO CODE)

```
// GIVEN
float nombre = 30000000

// WHEN
int resultat = condenser(number)

// THEN
assert resultat == 30M
```

TESTS D'INTÉGRATION/SERVICE

- Vérifient le bon assemblage de plusieurs composants
- Par composants, on entend plusieurs classes/modules
- Ecrits pour les programmeurs (souvent par les programmeurs)

EXEMPLE (EN LANGAGE NATUREL) : TWITTER

- Etant donné un compte qui a bloqué l'utilisateur
- Lorsque l'utilisateur veut afficher les tweets de ce compte
- Alors une erreur survient

EXEMPLE (EN PSEUDO CODE)

```
// GIVEN
var utilisateur = new Utilisateur();
var compte = new Compte();
compte.bloque(utilisateur);

// WHEN
Try<> result = utilisateur.getTweets(compte)

// THEN
assert result is error
```

TESTS END-TO-END

- Vérifient le bon fonctionnement du logiciel
- Ecrits pour le client et les programmeurs
- Exercent l'UI tel un utilisateur
- Font intervenir des composants tiers (web service)
- S'exécutent plus lentement (quelques secondes par test -> plusieurs minutes au total)

EXEMPLE (EN LANGAGE NATUREL) : TWITTER

- Etant donné un utilisateur abonné à un compte et étant sur la timeline sur l'application mobile
- Lorsqu'un nouveau tweet est publié sur ce compte
- Alors une notification indiquant le nouveau tweet devrait être affichée

EXEMPLE (EN PSEUDO CODE)

```
// GIVEN
click 'Se connecter'
saisir login
saisir mot de passe
s'abonner au compte XYZ

// WHEN
compte publie tweet

// THEN
assert ui.notifications contient tweet
```

TESTS DE FUMÉE

- Vérifient le logiciel dans les cas simples
- Le nom provient du hardware

TESTS DE PERFORMANCE

- Utilisation processeur
- Utilisation mémoire
- Utilisation disque
- Montée en charge

TESTS AUTOMATISÉS

QUI ÉCRIT LES TESTS AUTOMATISÉS ?

- Des testeurs logiciels (QA en anglais)
- Les développeurs eux-mêmes (de plus en plus)
- Certains disent le client (voir le format Gherkin et Cucumber)

COUVERTURE DE CODE

LA COUVERTURE DE CODE, C'EST QUOI ?

- Mesure qui permet de voir la proportion de code de production exécutée par les tests
- S'exprime en général en pourcentage
- Permet de s'assurer qu'un morceau de code est couvert par les tests
- Dans certains IDEs on peut voir la couverture pour chaque ligne de code

COUVERTURE DE TEST

100% N'EST PAS UN OBJECTIF

100% DE COUVERTURE NE VEUT PAS DIRE PAS DE BUG

UNE LIGNE DITE "COUVERTE" A ÉTÉ EXÉCUTÉE PAR AU MOINS UN TEST

COUVERTURE DE TEST

100% N'EST PAS UN OBJECTIF

100% DE COUVERTURE NE VEUT PAS DIRE PAS DE BUG

UNE LIGNE DITE "COUVERTE" A ÉTÉ EXÉCUTÉE PAR AU MOINS UN TEST

COUVERTURE DE TEST

TESTS UNITAIRES ET DÉPENDANCES

COMMENT TESTER CE CODE ?

```
public int printReceipt(String drink, boolean student, int amount
    // [...]
    USBPrinter printer = new USBPrinter("/dev/ttyUSB0");
    printer.print(drink, student, amount, price*amount);
    // [...]
}
```

DOUBLURE DE TEST *(TEST DOUBLE)*

POURQUOI ?

- Parfois on ne veut pas exécuter une partie de code pendant le test (envoi de mail, base de données, réseau, ...)
- On remplace donc l'implémentation réelle d'un module par une fausse implémentation
- Meilleure performance en utilisant une doublure
- On utilise le type de doublure le plus adapté en fonction des cas

COMMENT ?

- Il suffit d'implémenter une interface pour pouvoir avoir plusieurs implémentations
- Une dite de "production", une (ou plusieurs) de test

COMMENT ?

- Parfois il faut modifier le code de production pour faire émerger cette interface
- C'est ce qu'on appelle l'inversion de contrôle

BOUCHON *(STUB)*

- Permet de pré-configurer les futures réponses du module doublure
- Utile pour une base de données, un appel réseau, une saisie utilisateur...

ESPION *(SPY)*

- Implémentation qui enregistre les interactions avec le module
- Permet de récupérer les interactions passées

FACTICE *(DUMMY)*

- Doublure qui ne fait rien de particulier
- Utile pour créer une doublure minimale dont on ne se sert pas vraiment

FAUX *(FAKE)*

- Implémentation qui fonctionne mais qui prend des raccourcis
- Exemple : une base de données en mémoire

SIMULACRE *(MOCK)*

- Permet de pré-configurer les futures réponses du module doublé (comme le bouchon)...
- ... et de vérifier les interactions en fin de test (comme l'espion)
- Des bibliothèques peuvent aider comme par exemple Mockito

EN RÉALITÉ LA DISTINCTION EST RAREMENT FAITE ET ON PARLE DE "MOCKS" DANS TOUS LES CAS

AUTOMATISATION DES TESTS

- Servent comme filet de sécurité (harnais de test)
- Vérifient la non-régression
- Le plus souvent, exécution sur chaque "push"

AUTOMATISATION DES TESTS