Les tests unitaires

esprit
Se former autrement

Bureau E204

PLAN DU COURS

- Introduction
- Tests Unitaires
- Utilisation de JUNIT
- Place à la Pratique

INTRODUCTION

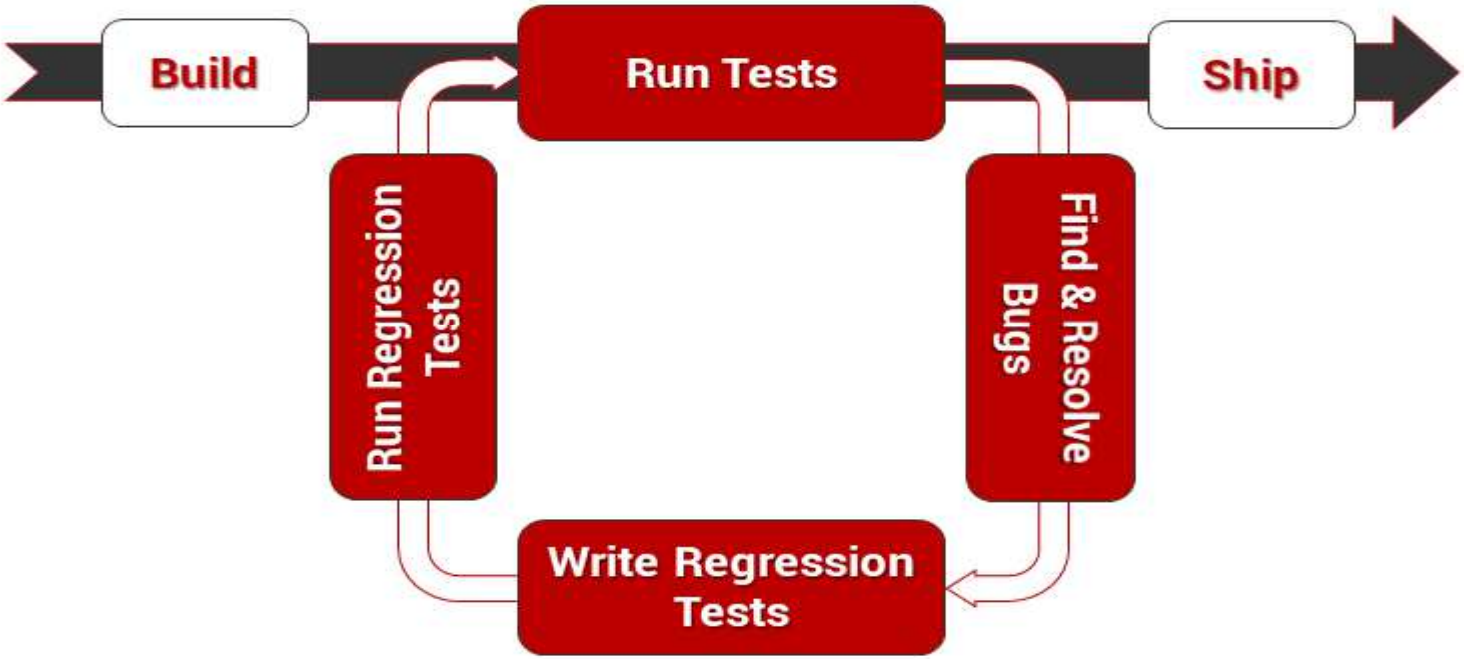
- Il existe différents niveaux de test :
  - Test unitaire
  - Test d'intégration
  - Test de charge
  - Test fonctionnel
  - Test sécurité
  - ….

Test d'intégration

- L'intégration, c'est assembler plusieurs composants logiciels élémentaires pour réaliser un composant de plus haut niveau.
- **Exemple**: Une classe Client et une classe Produit pour créer un module de commande sur un site marchant, c'est de l'intégration !
- **Un test d'intégration** vise à s'assurer du bon fonctionnement de la mise en œuvre conjointe de plusieurs unités de programme, testés unitairement au préalable.

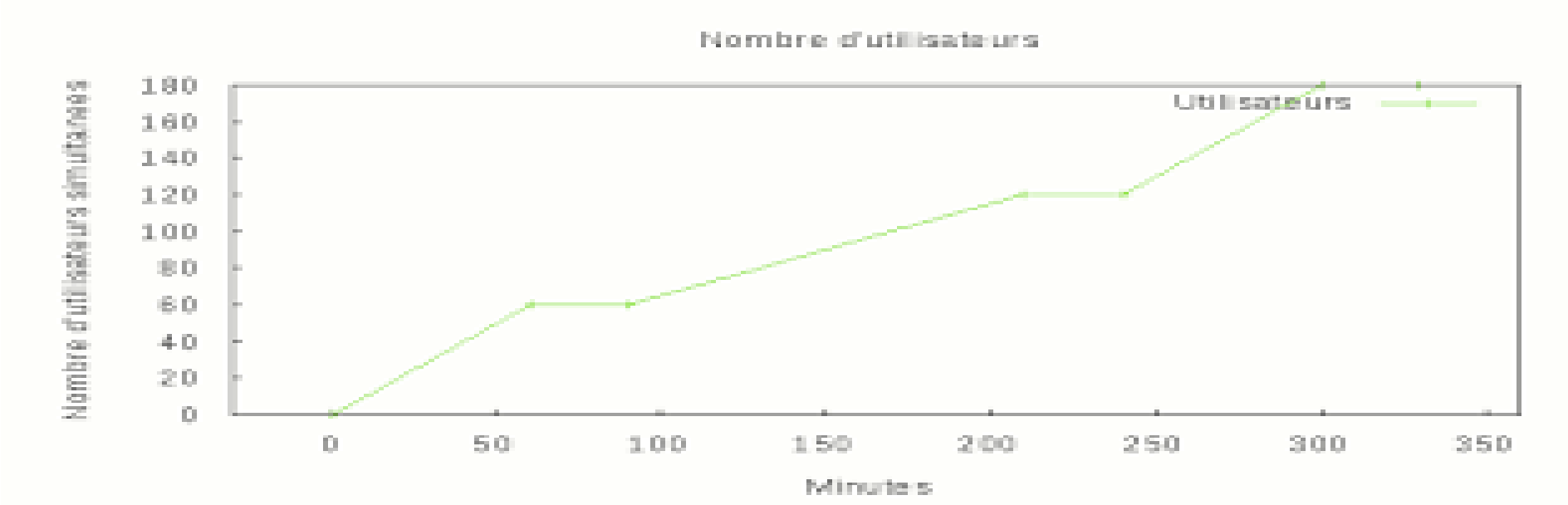
Test de régression

- **Les tests de régression** sont les tests exécutés sur un programme préalablement testé mais qui a subit une ou plusieurs modifications.



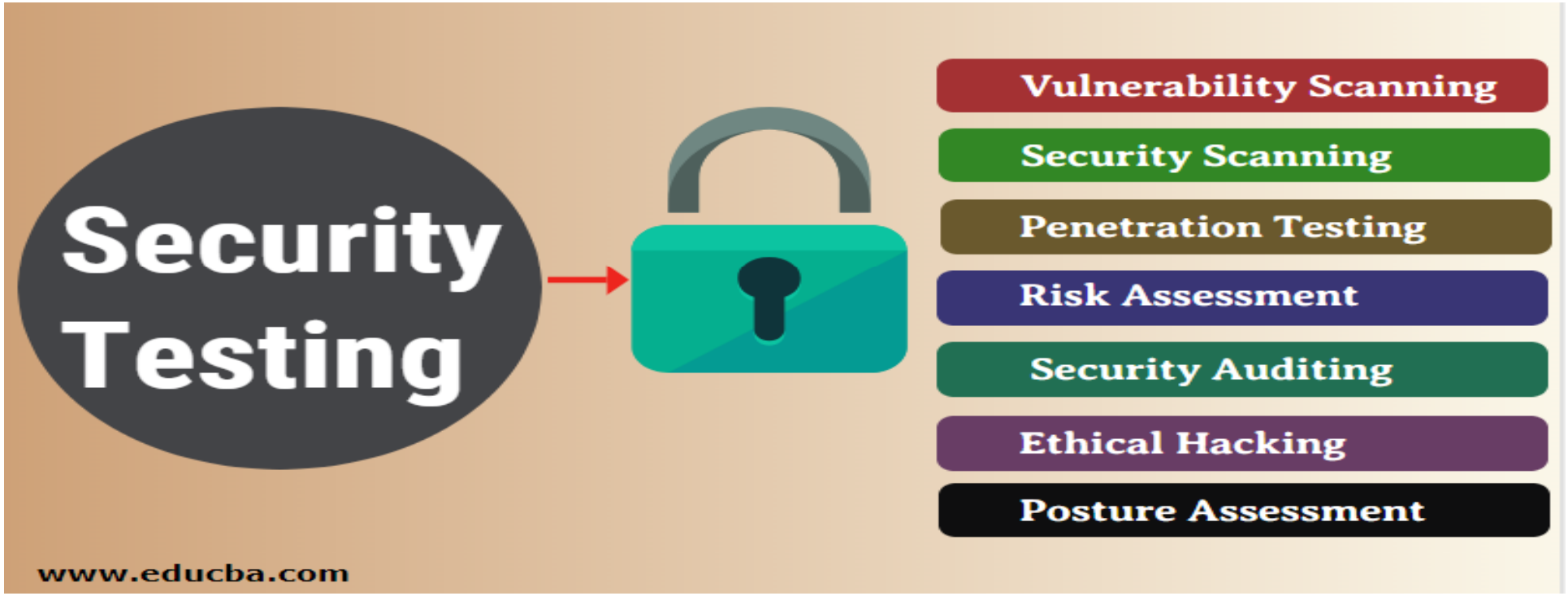
Test de montée en charge

- **Test de montée en charge** (Test de capacité) : il s'agit d'un test au cours duquel on va simuler un nombre d'utilisateurs sans cesse croissant de manière à déterminer quelle charge limite le système est capable de supporter.



Test de sécurité

- Le test de sécurité est un type de test de logiciel qui vise à découvrir les vulnérabilités du système et à déterminer que ses données et ressources sont protégées contre d'éventuels intrus.



TEST UNITAIRE : DÉFINITION

- Un test unitaire est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel. Il s'agit d'un code.
- En POO, on teste au niveau des classes
- Pour chaque classe (MyClass), on a une classe de test (MyClassTest).

TEST UNITAIRE : QUELQUES RÈGLES

- Doit être isolé : il doit être indépendant
- N'est pas un test de bout en bout : il agit que sur une portion de code
- Doit être déterministe : le résultat doit être le même pour les mêmes entrées
- Est le plus petit et simple possible

TEST UNITAIRE : QUELQUES RÈGLES

- Ne teste pas d'enchainement d'actions
- Etre lancé le plus souvent possible : intégration continue
- Etre lancé le plus tôt possible : détection des bug plus rapide
- Couvrir le plus de code possible
- Etre lancé a chaque modification

TEST UNITAIRE : AVANTAGE ET INTÉRÊT

- Garantie la non régression
- Détection de bug plus facile
- Aide a isoler les fonctions
- Aide a voir l'avancement d'un projet (TDD)

*** Le test-driven development (TDD) ou en français développement piloté par les tests est une technique de développement de logiciel qui préconise d'écrire les tests unitaires avant d'écrire le code source d'un logiciel.**

TEST UNITAIRE : OUTIL DE TEST

PHP	JS	SQL	JAVA
PHPUnit SimpleTest	JSUnit	SQLUnit	JUnit

TEST UNITAIRE : CAS A TESTER

- Lors de l'utilisation de test unitaire on se doit de tester différents cas.
  - Cas en succès : fonctionnement normal
  - Cas d'erreur : test sur la gestion d'erreur

TEST UNITAIRE : LES RÉSULTATS

- Un test unitaire peux renvoyer 3 résultats différents :
  - Success : test réussi
  - Error : erreur inattendue a l'exécution
  - Failure : au moins une assertion est fausse

TEST UNITAIRE : MOCK

- Quelques fois un test a besoin d'un composant donné pour s'exécuter.
- Par exemple pour tester une fonctionnalité, nous avons besoin du retour d'un Web Service, qui n'a toujours pas été développé.
- Il est alors utile d'utiliser des bouchons (MOCK) pour isoler le test.
- De plus un bouchon permet de tester tout les cas (valeur correcte, erroné etc.)

UTILISATION DE JUNIT

- Il n'y a pas de limite au nombre de tests au sein de notre classe de test.
- On écrit au moins un test par méthode de la classe testée.
- Pour désigner une méthode comme un test, il suffit d'utiliser l'annotation **@Test** (a partir de JUnit4).

UTILISATION DE JUNIT

- Au sein des tests, on utilise des **assertions** pour valider ou non un test. Quelques assertions indispensable :

Assertion	Action
assertEquals()	Vérifie l'egalité entre deux entités
assertNotEquals()	Vérifie l'inégalité entre deux entités
assertFalse()	Vérifie que la valeur fourni en paramètre est fausse
assertTrue()	Vérifie que la valeur fourni en paramètre est vrai
assertNull()	Vérifie que la valeur fourni en paramètre est l'objet NULL
assertNotNull()	Vérifie que la valeur fourni en paramètre n'est pas l'objet NULL

UTILISATION DE JUNIT

```
package tn.esprit.spring;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;
@RunWith(SpringRunner.class)
@SpringBootTest
public class StockServiceImplTest {
@Autowired
IStockService stockService;
@Test
public void testAddStock() {
List<Stock> stocks = stockService.retrieveAllStocks();
int expected=stocks.size();
Stock s = new Stock();
s.setLibelleStock("stock test");
s.setQte(10);
s.setQteMin(100);
Stock savedStock= stockService.addStock(s);
assertEquals(expected+1, stockService.retrieveAllStocks().size());
assertNotNull(savedStock.getLibelleStock());
stockService.deleteStock(savedStock.getIdStock());
}
```

Travail à faire

- Optimiser le test d'ajout du stock en évitant la double récupération de la liste du stock lors du test.
- Implémenter un test permettant de vérifier que la fonctionnalité de suppression est bien opérationnelle.

JUNIT : PLACE A LA PRATIQUE

Développement d'un Projet qui implémente JUNIT

Ajout de JUNIT dans un Projet déjà existant

QUIZ