Introduction aux tests logiciels

Fabian SERIN 27/09/2023

Qui suis-je ?

Fabian SERIN

fabian.formations@gmail.com

Présentation SLB

https://www.welcometothejungle.com/fr/companies/schlumberger

Organisation des deux jours

Jour 1

Matin:

Fondamentaux des tests

Tester pendant le cycle de vie du logiciel

Après midi:

Démos et Travaux Dirigés

Jour 2

Matin:

Récapitulatif

Gestion des tests

Après midi:

Suite Gestion des tests / Fin TD

360

QCM

Évaluation

TD:

Note de participation sur 5 points

Note de réussite sur 15 points

(envoyer les productions à fabian.formations@gmail.com)

QCM:

37 Questions (40 points)

1 bonne réponse: +1

Pas de réponse: 0

Mauvaise réponse: 0

ISTQB

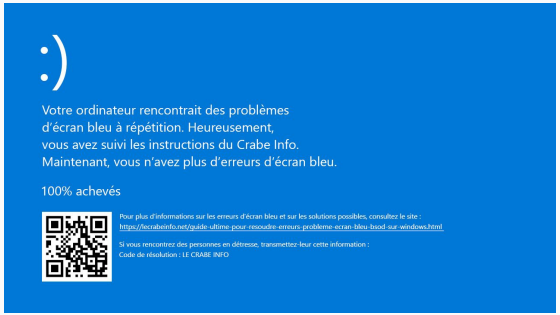
International Software Testing Qualifications Board https://www.istqb.org/

Comité Français des tests logiciels CFTL https://www.cftl.fr/

Fondamentaux des tests

Contexte des tests

exemples de bugs



et aussi

Origine des défauts

Erreur

Défaut (Bug) dans le code, les spécifications ou activité connexe

Défaillance

L'origine de l'erreur est multiple, manque de temps, de compétence, de communication, ...

L'introduction du défaut n'est pas systématique
La défaillance n'est pas systématique

Objectif: Qualité

Test et qualité

La qualité logicielle est une appréciation globale d'un logiciel, basée sur de nombreux indicateurs:

- La complétude des fonctionnalités,
- la correction et précision des résultats,
- la fiabilité,
- la tolérance de pannes,
- la facilité et la flexibilité de son utilisation,
- la simplicité,
- l'extensibilité,
- la compatibilité et la portabilité,
- la facilité de correction et de transformation,
- la performance,
- la cohérence et l'intégrité des informations qu'il contient

Ces indicateurs sont définis par des exigences (requirements), des spécifications.
La qualité devient le degré par lequel un logiciel / composant atteint les exigences spécifiées.

Que sont les tests ?

Les tests mesurent les écarts entre le logiciel et les exigences

Les tests se basent sur les fonctionnalités

Les tests se basent sur les non-fonctionnalités

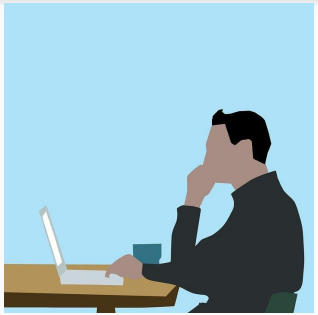
Les tests sont estimés, planifiés, analysés, maintenus

Un test cherche les défaillances et les défauts

Les tests améliorent la qualité

Créer les tests	Exécution des tests	Résoudre les défauts	Amélioration de la qualité
A partir des spécifications fonctionnelles ou non fonctionnelles, écrire des suites de tests	Jouer les suites de tests, manuels, automatiques Trouver ou non des défaillances	A partir des défaillances trouvées, faire une analyse des causes. Corriger les causes directes (défauts) et dans le cadre de rétrospectives corriger les sources des erreurs	Amélioration de la confiance dans les spécifications, composants. Garantie que les cas testés fonctionnent dans le contexte de tests.

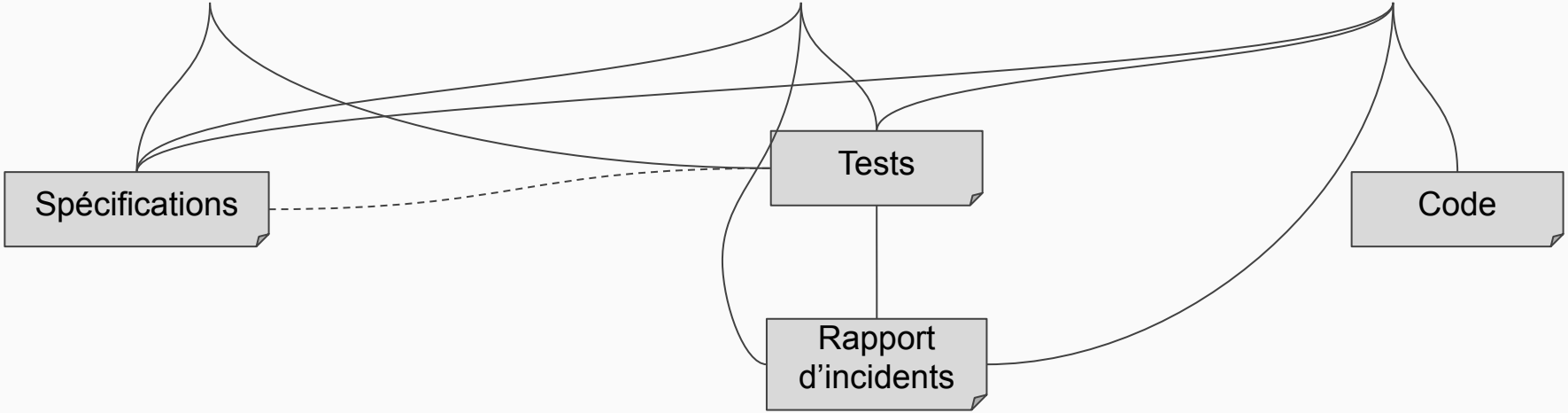
Acteurs des tests



Product owner

Testeur

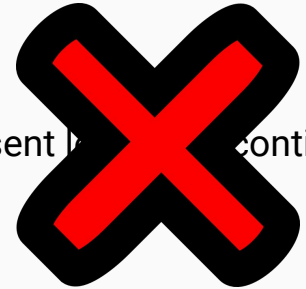
Développeur



Les 7 principes

1.Les tests montrent la présence de défauts, par leur absence

Si les tests passent l[illegible]contient zéro défaut



Les tests peuvent prouver la présence de défauts, mais ne peuvent pas en prouver l’absence. Les tests réduisent la probabilité que des défauts restent cachés dans le logiciel mais, même si aucun défaut n’est découvert, ce n’est pas une preuve que tout est correct

Le risque zéro n'existe pas
Les tests ne peuvent pas affirmer qu’un logiciel est sans défaut

Les 7 principes

2.Les tests exhaustifs sont impossibles

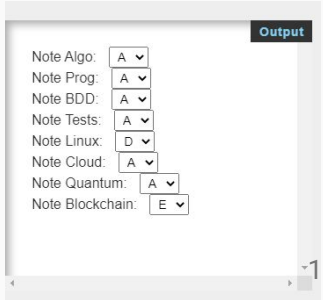
Je veux que les tests te les cas possibles

Tout tester (toutes les combinaisons d'entrées et de préconditions) n'est pas faisable sauf pour des cas triviaux. Plutôt que de chercher à faire tests exhaustifs, l'analyse des risques, des techniques de test et des priorités devraient être utilisées pour cibler les efforts de tests

Exemples:

- transaction bancaire
- formulaire (15 champs, 5 valeurs possibles)

5^{15}=30 517 578 125 cas
Si 1 cas en 1 ms → 353 jours



Les 7 principes

3.Tester tôt économise du temps et de l’argent

On fera les t[illegible]n

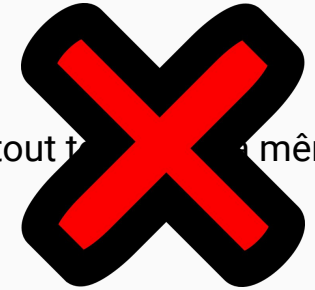
Pour détecter tôt les défauts, des activités de tests doivent être lancées le plus tôt possible dans le cycle de vie de développement du logiciel. Tester tôt dans le cycle de vie du développement logiciel permet de réduire ou d'éliminer des changements coûteux.

Tester dès la phase de spécifications
Une erreur dans les specs qui se propage dans les plans de tests, dans le codage et dans la validation sera très coûteuse à réparer.

Les 7 principes

4.Regroupement des défauts

Il faut tout t même façon



Un petit nombre de modules contient généralement la plupart des défauts. Des regroupements prévisibles de défauts constituent un élément important de l'analyse des risques utilisée pour cibler l'effort de test.

Principe de pareto: 80% des effets sont dues à 20% des causes

Les 7 principes

5.Le paradoxe du pesticide

Plus je teste [illegible] est

Si les mêmes tests sont répétés de nombreuses fois, le même ensemble de cas de tests finira par ne plus détecter de nouveaux défauts.

Pour détecter de nouveaux défauts, il peut être nécessaire de modifier les tests existants et les données de test existantes, ainsi que de rédiger de nouveaux tests.

Les tests doivent être remis en cause, pensés différemment, avec une autre approche

Les 7 principes

6.Les tests dépendent du contexte

Un code qui “march[illegible]e dans tous les cas

Les tests sont effectués différemment dans des contextes différents.

Exemple de contexte :
environnement (linux, windows)
contraintes de budget
versions des serveurs, des applications tierces
type d’application (gestion de bdd, aéronautique, site web)

Exemple: Ariane 5

Les 7 principes

6.Les tests dépendent du contexte

Les 7 principes

7.L’absence d’erreur est une illusion

Tous les tests passent et [illegible] fonctions sont ok donc tout va bien

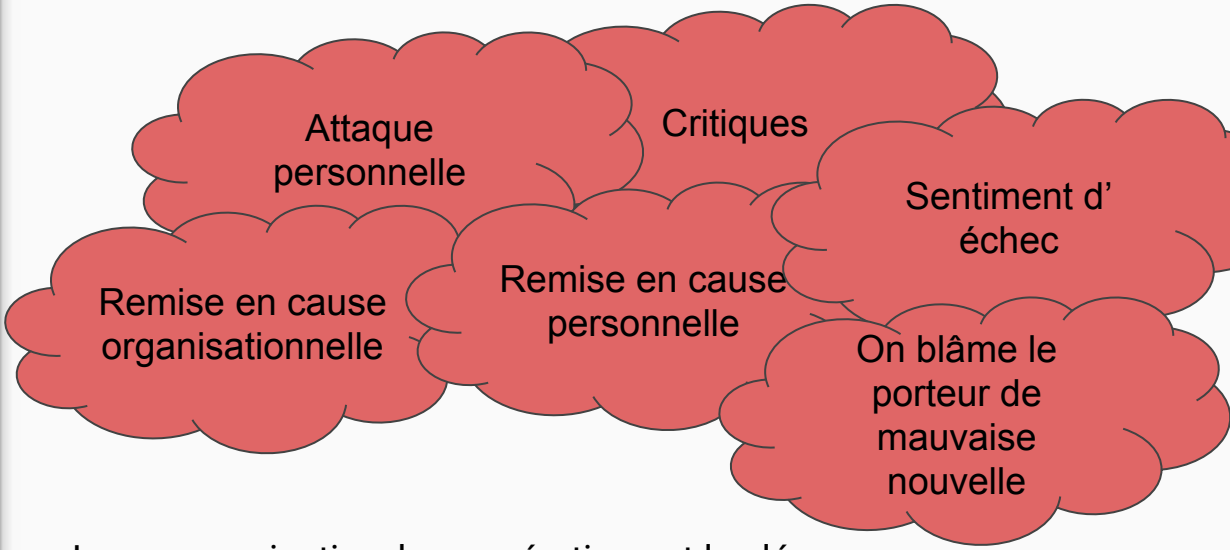
Il est illusoire de s'attendre à ce que le simple fait de trouver et de corriger un grand nombre de défauts garantisse la réussite d’un système.

Il n’y a pas que le fonctionnel qui compte du point de vue de l’utilisateur.
Les attentes des utilisateurs sont traduites en spécifications
Les tests et le dev se basent sur les spécifications
Le logiciel fonctionne bien quand il conforme aux spécifications
Le logiciel est utilisable quand il est conforme aux attentes des utilisateurs.

Les tests et la psychologie

But d’un développeur:
Faire un logiciel sans défauts

But d’un testeur:
Trouver des défauts



La communication, la coopération est la clé
Les équipes de test et de dev doivent travailler ensemble

But commun: Améliorer la qualité

Tester pendant le cycle de vie du logiciel

Rappel sur les modèles de cycle

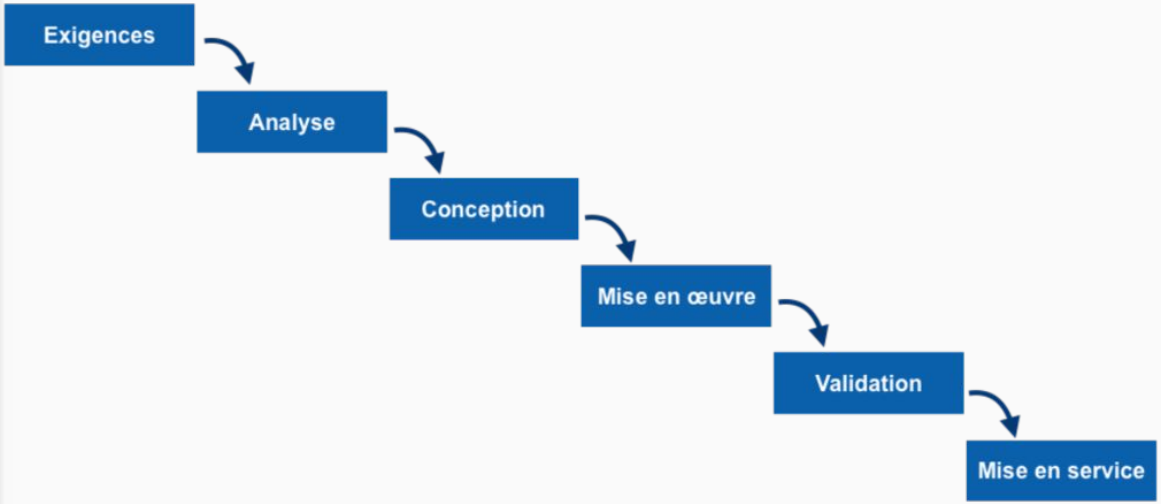
Modèle en cascade

Modèle en V

Modèle itératif

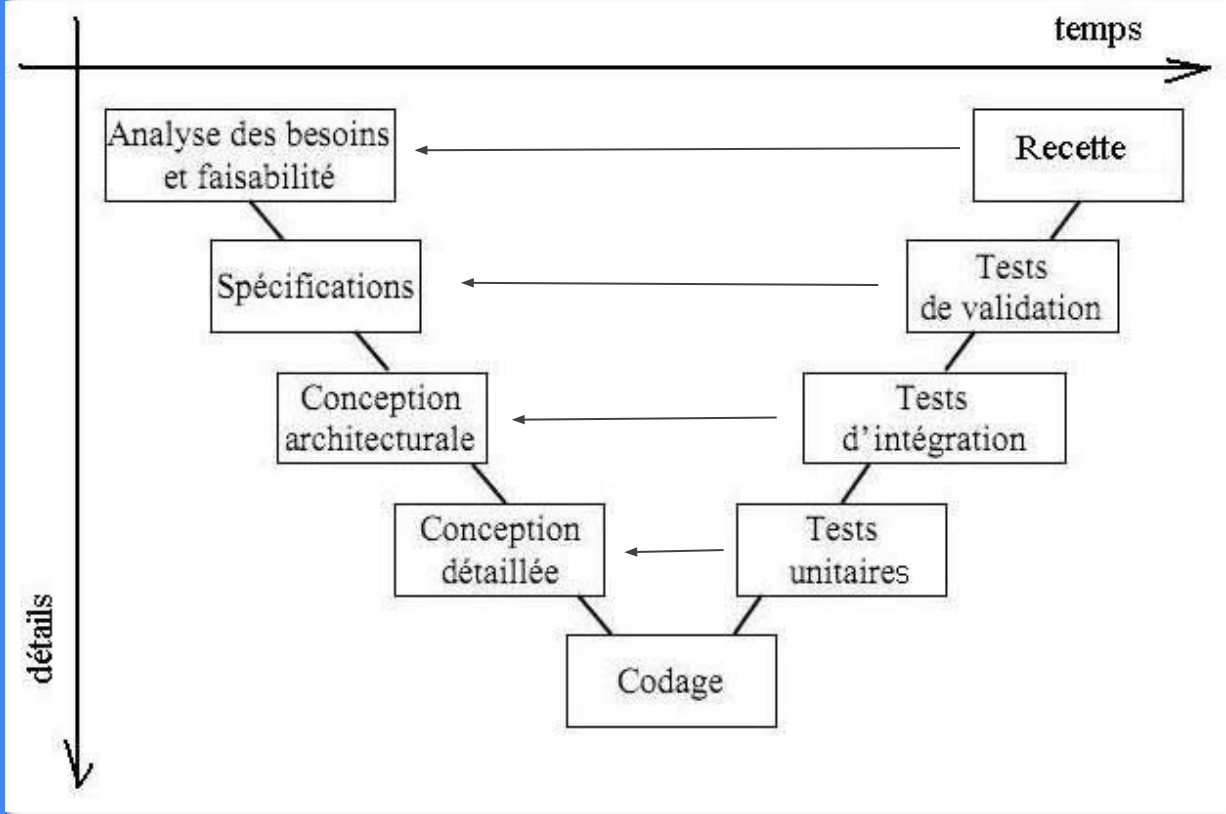
Rappel sur les modèles de cycle

Modèle en cascade



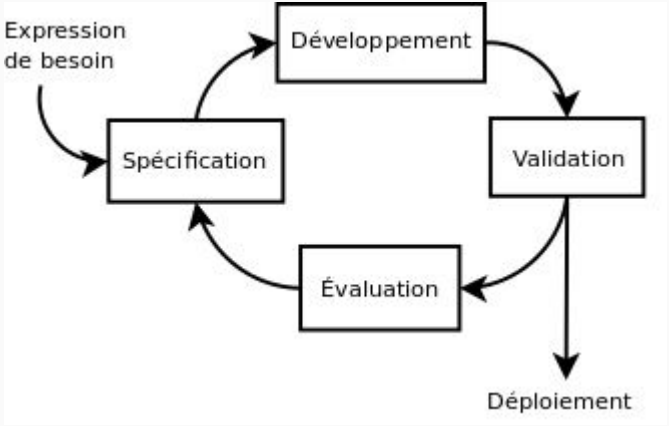
Rappel sur les modèles de cycle

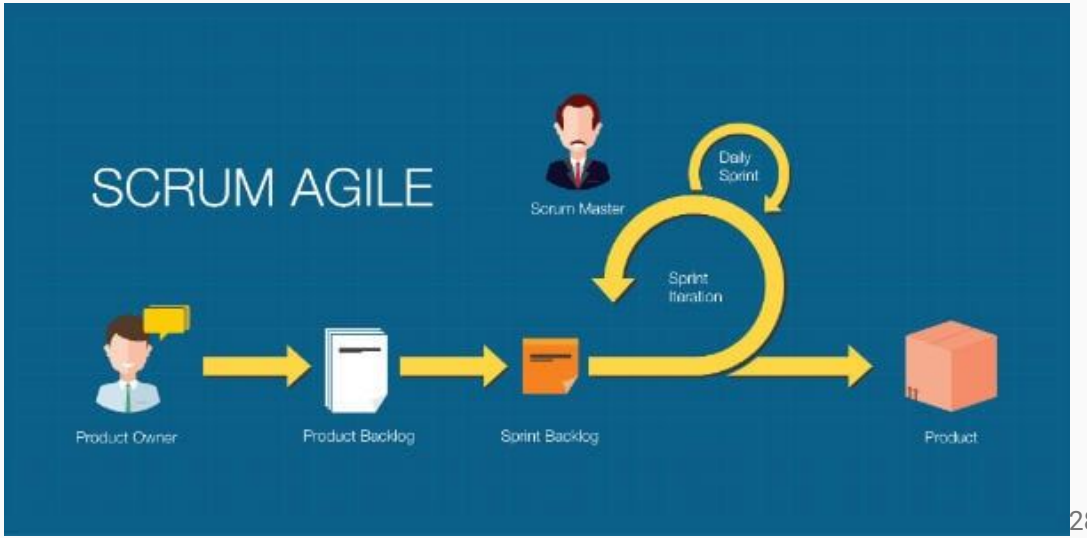
Modèle en V



Rappel sur les modèles de cycle

Modèle en itératif





Niveaux des tests

Niveaux de tests

Composants (unitaires)

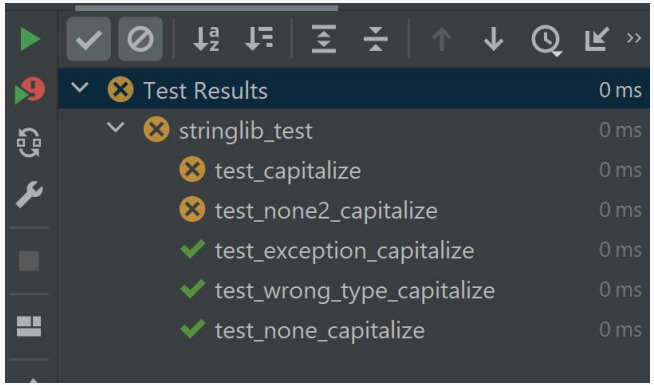
Test de composants pouvant être testés individuellement
Objectifs: trouver des défauts dans le composant

Souvent écrits et exécutés par le développeur avant, pendant ou après le développement.

Test statiques ou dynamiques

Ces tests servent aussi de test de non régression et de maintenance, fonctionnels et non fonctionnels, statiques et dynamiques.

Et peuvent faire appel à des mocks (pilotes ou bouchons).



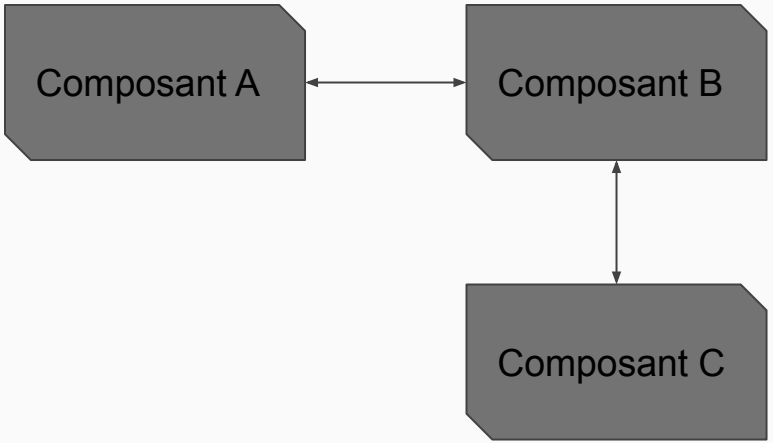
Niveaux de tests

Intégration

v

Test des interactions entre les composants
Objectifs: trouver des défauts dans les interactions entre composants

Fonctionnels ou non fonctionnels, concerne les API, les types de données attendus, etc.



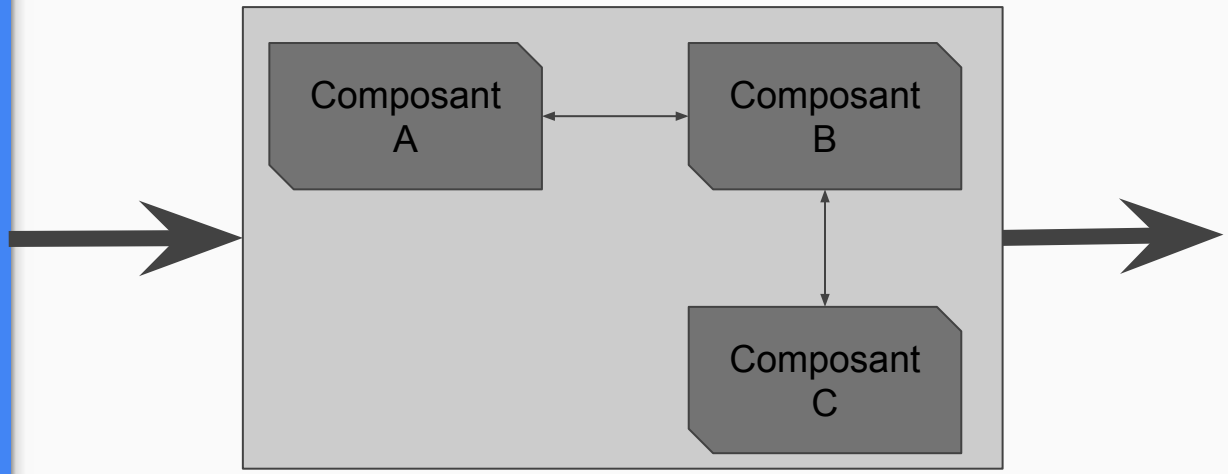
Niveaux de tests

Système

Test du système ou produit complet par rapport aux spécifications

Objectifs: trouver des défauts dans le produit final fonctionnels ou non fonctionnels

Il peut s'agir de tester les résultats de calcul, les temps de réponse etc.



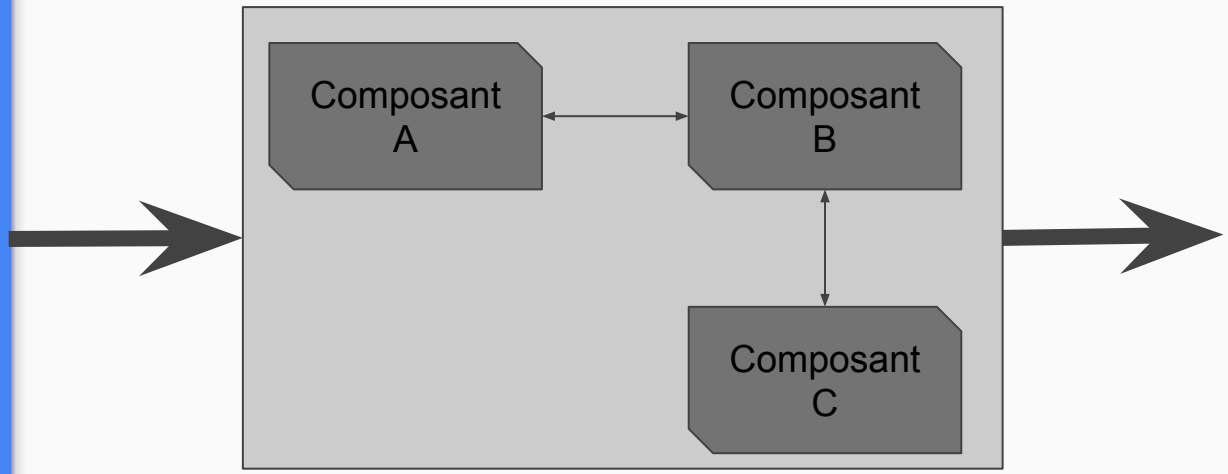
Niveaux de tests

Acceptation

Test du système complet dans son contexte par rapport aux spécifications

Objectifs: Valider que le système ou produit fonctionne comme prévu

Ce type de test ne doit pas en principe trouver de défauts, il sert à donner confiance dans le logiciel



Types de tests

Types de tests

fonctionnels

Une fonctionnalité est un **QUOI**
Par exemple l'application calcule les moyennes pondérées des notes des étudiants.

A quels niveaux s'appliquent ces tests ?

TOUS

Types de tests

non fonctionnels

Le non fonctionnel est ce qui est attendu par l'utilisateur mais qui n'est pas un QUOI, c'est plutôt un **COMMENT**

Par exemple:

- la performance, le logiciel doit donner les résultats rapidement sans donner l'impression de se figer
- l'API doit répondre en une seconde maximum
- le logiciel doit être facile à utiliser
- le code doit être documenté et facile à maintenir
- le site web ne doit pas exposer ses mots de passe

A quel niveau se font les tests non fonctionnels ?

Tous

bien que souvent on pense d'abord à les faire au niveau système

Types de tests

boite noire

Les tests sont écrits sans avoir connaissance des détails d'implémentation, uniquement des spécifications.

Types de tests

boite blanche

Les tests boîte blanche sont basés sur la connaissance des détails d'implémentation et d'architecture.

Par exemple on peut avoir des mesures de complexité du code, de sa modularité, de la présence ou non de dépendance cyclique, de la couverture des tests

Coverage report: 21%

Module ↑	statements	missing	excluded	coverage
simple__init__.py	0	0	0	100%
simple\second_exercise.py	11	0	0	100%
simple\sort.py	59	59	0	0%
simple\stringlib.py	6	1	0	83%
Total	**76**	**60**	**0**	**21%**

coverage.py v5.5, created at 2021-03-14 20:34 +0100

```
--------------------------------------
Your code has been rated at 7.84/10
```

Types de tests

confirmation

Lorsqu’un bug est détecté, la première chose est de le reproduire sous forme d’un test.

Une fois le bug corrigé, on fait tourner à nouveau les tests pour confirmer que ce bug n’est plus présent.

Types de tests

non régression

Les tests de non régressions sont très utiles lorsque le code, les composant évoluent:

- par exemple dans les modèles itératifs
- mais aussi dans le cycle de vie d'une application qui évolue sur plusieurs versions

Un test de non régression va détecter qu'une spécification fonctionnelle ou non fonctionnelle n'est plus respectée

Un test de confirmation, une fois dans les suites de tests peut être considéré comme un test de non régression.

Types de tests

maintenance

La maintenance d’un logiciel consiste en :

- la mise à jour du moteur de base de données
- la mise à jour de version de librairies
- le passage de windows à mac os ou à Linux
- le passage de Chrome à Edge
- etc….

Les tests de maintenance permettent de détecter des défaillances fonctionnelles et non fonctionnelles.

TDD

TDD

Méthode de développement qui consiste à développer par étapes en écrivant les tests avant les fonctionnalités.

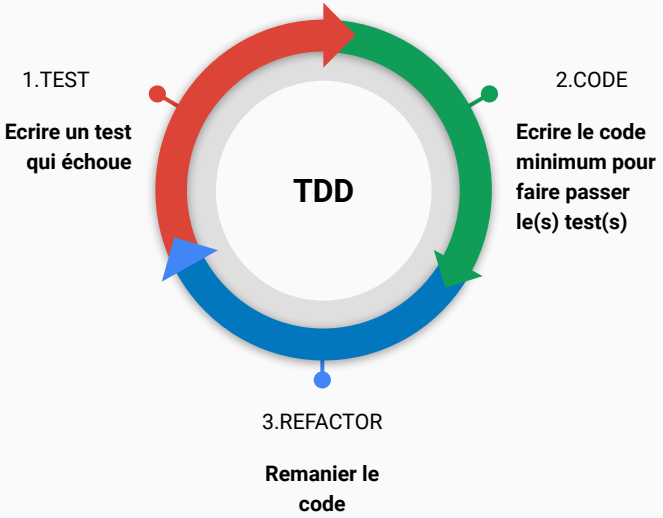
1 écrire un test

2 vérifier que ce test échoue

3 écrire le minimum de code pour que le test réussisse

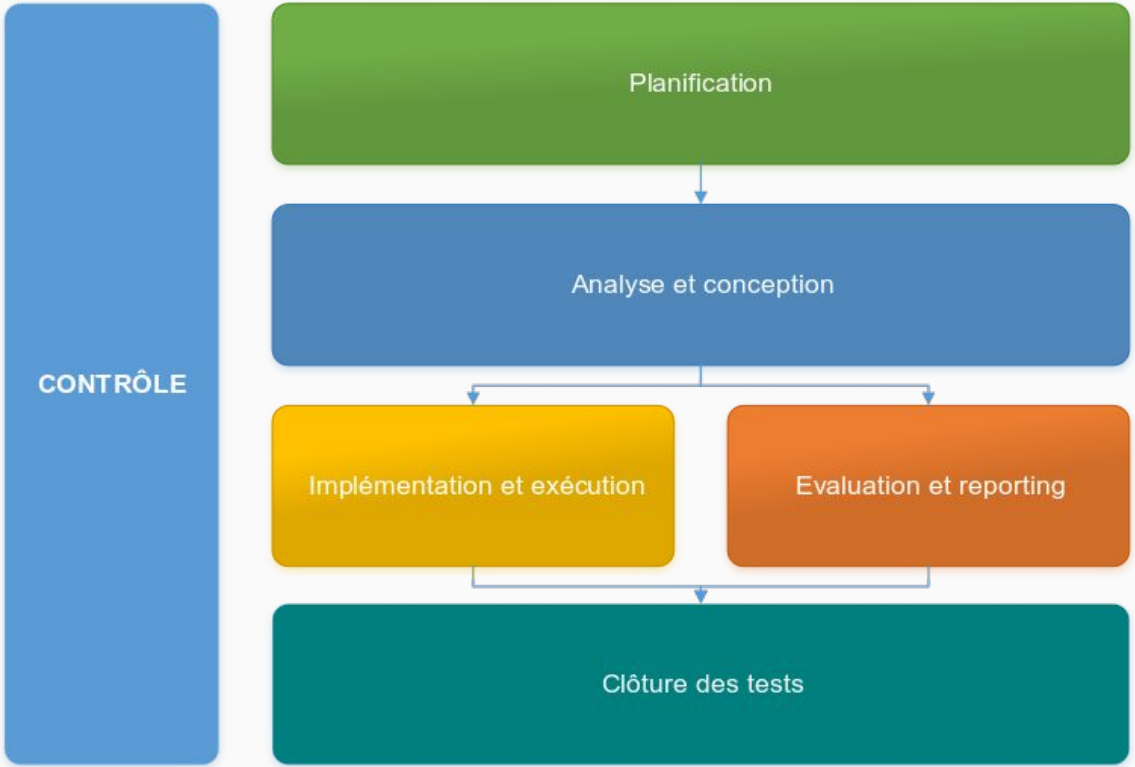
4 vérifier que le test passe

5 refactorer le code - l'améliorer sans changer le comportement (les tests passent toujours)



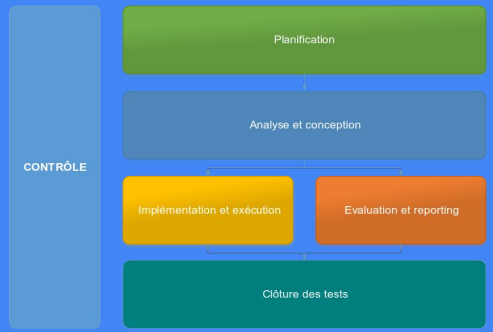
Gestion des tests

Gestion des tests



Gestion des tests

Planification / Plan de test

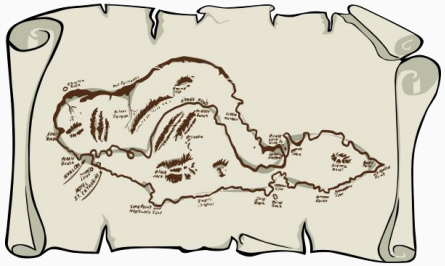


Approche des tests

Périmètre et Objectifs des tests

Ressources nécessaires

Métriques pour le contrôle des tests



But et contenu rapports de test

Planification dans le temps

Définir le budget

Definition of ready / done

Gestion des tests

Planification / Plan de test

Approches de test possibles:

- Analytique (ex analyse de risque)
- Basée sur des modèles
- Méthodique (taxonomie des types de défaillances, norme interne)
- Conforme à un processus ou une norme externe
- Dirigée ou consultative (recommandations d'experts métiers ou techniques)
- Anti-régréssions
- Réactive (tests conçus en fonction des résultats de tests antérieurs)

Gestion des tests

Planification / Plan de test

Définition of Ready
Conditions permettant à l'activité de test de démarrer

- Disponibilité d'exigences testables
- Disponibilité d'éléments de test (dépendances à des tests précédents)
- Disponibilité de l'environnement de test
- Disponibilité des outils de tests et autres ressources

Attention, l'activité de test peut démarrer même si les conditions ne sont pas satisfaites mais —> Augmentation du risque / temps / coût

Définition of Done
Conditions qui permettent de terminer l'activité de test

- Les tests planifiés ont été exécutés
- Un niveau de couverture est atteint
- Le nombre de défaut non résolus est inférieur à un seuil
- Les niveaux de fiabilité, performance, … sont suffisants

Attention même si les conditions de sortie ne sont pas satisfaites on peut arrêter les activités de test: dépassement budget, planning, pression pour livrer, …

Gestion des tests

Métrique: couverture

Coverage for **simple\stringlib.py** : 83%

6 statements | 5 run | 1 missing | 0 excluded

```
1  def basic_capitalize(data: str) -> str:
2      return data.capitalize()
3
4
5  def safe_capitalize(data: str) -> str:
6      if isinstance(data, str):
7          return basic_capitalize(data)
8      return data
```

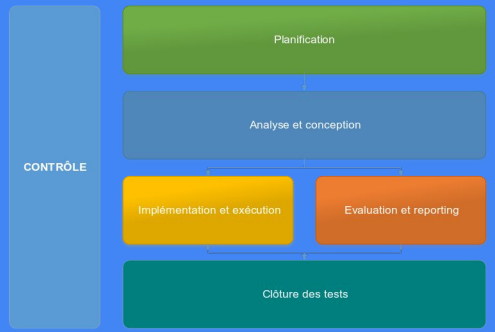
Couverture de spécifications:
Indique si telle ou telle spécification est couverte. Cette tâche est souvent manuelle

Couverture de code:
Indique si telle ou telle ligne de code est couverte par des tests. Peut être calculé automatiquement

Attention une couverture de code à 100% ne veut pas dire que tous les cas sont testés

Gestion des tests

Contrôle

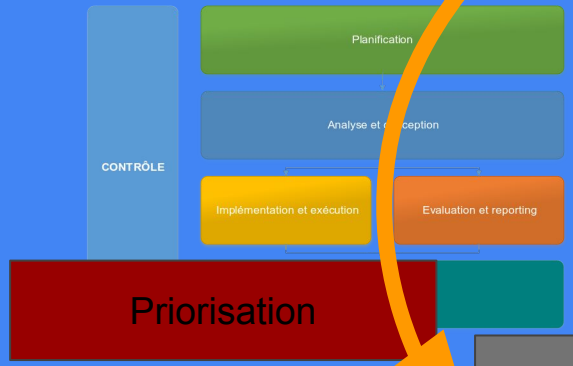


Tout au long de la procédure de tests sont effectués des contrôles:

- déviation par rapport à la planification
- variations dans les résultats
- mesure d'avancement (effectué / prévu)
- estimation des retards
- mise en place de mesures correctives
- ...

Gestion des tests

Analyse et conception



Priorisation

Traçabilité bidirectionnelle

Plan de test : Objectifs de test, approche de test

Documents utilisateurs

Spécifications

Normes / Standards

Documents contractuels

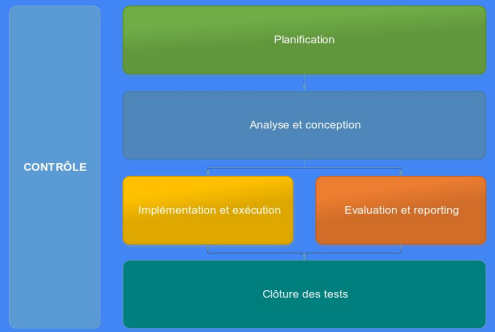
Procédures de test

Conditions de test

Données de test

Gestion des tests

Implémentation et exécution



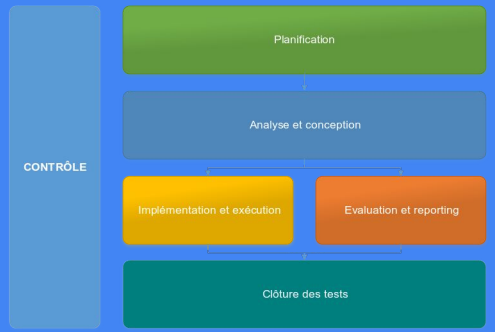
Les testeurs/développeurs implémentent les tests pour couvrir les fonctionnalités et les cas de tests. Ceci comprend les différents niveaux et types de tests et les outils de tests.

Une fois les tests implémentés ils sont exécutés soit directement (si possible) soit en suivant le calendrier de tests.

Certains tests automatiques peuvent être exécutés en continu, à chaque modification de code par des pipelines de test.

Gestion des tests

Évaluation et reporting



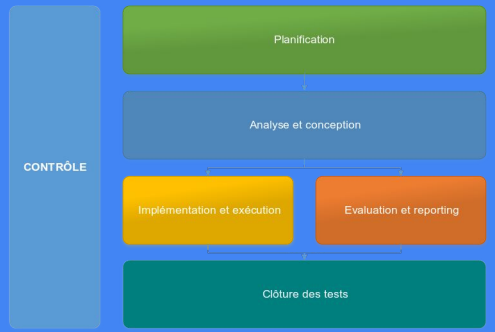
Rapports de test:
Pour les tests managers:
suivre l'état d'avancement des tests, la qualité du logiciel, la couverture fonctionnelle

Pour les développeurs
Pour chaque défaillance trouvée il faut documenter la défaillance (date du test, environnement, version du logiciel, données et procédure pour reproduire le défaut).

Les développeurs à partir de ce rapport peuvent analyser la défaillance et trouver le défaut dans le code. Une fois trouvé, ils peuvent le corriger.

Gestion des tests

Clôture



Lorsque les tests atteignent la definition of done, ou que le budget se réduit, que les dates de livraison approchent, que le niveau de qualité requis est atteint les tests sont clôturés.

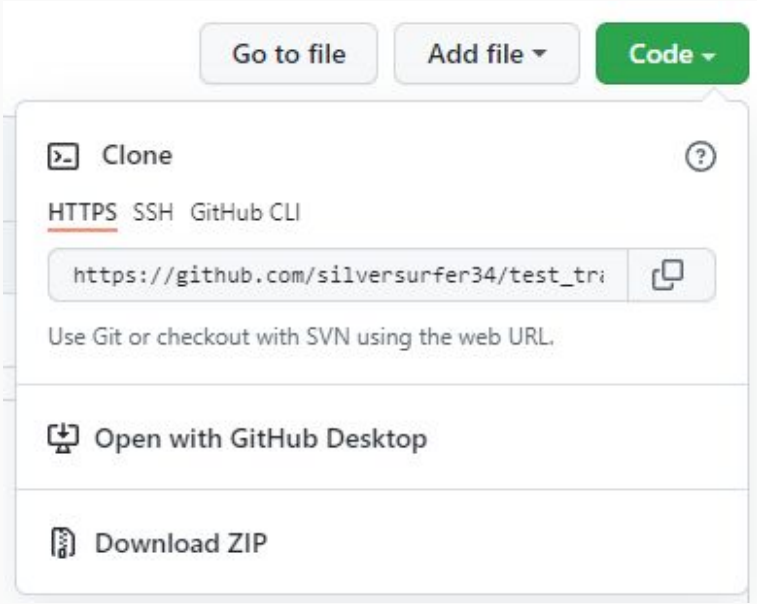
La clôture des tests requiert un accord entre les différents membres de l'équipe. Tout le monde doit être d'accord sur le niveau de qualité du logiciel avant de clôturer les tests.
Les défaillances non corrigées doivent être mineures et documentées.
Les objectifs de tests doivent être atteints, sinon les différences comprises et acceptées.

Les tests, résultats de tests, documents de tests, données, environnements de test, etc… sont sauvegardées et archivées

Démos et Travaux dirigés

Prérequis

Clone or download this git repository
https://github.com/silversurfer34/test_training_2022



Setup Python virtual env
python -m venv venv
windows: venv\script\activate
linux: source venv/bin/activate
pip install -r requirements.txt -r requirements_dev.txt

Démos - TD

Test de performance

- unitaire (timeit)

Démo de timeit sur des algorithmes de tris sur des tableaux de 10_000 valeurs ou plus

TD: Utilisation de timeit sur des cas d'utilisation:

- Tableau aléatoire
- Tableau déjà trié (croissant)
- Tableau trié (croissant) mais le plus petit élément est à la fin
- Tableau trié à l'envers (décroissant)
- Tableau trié (croissant) mais le plus grand élément est au début

Fournir le code, les résultat de mesure et vos conclusions (quel est le meilleur algorithme dans chacun des cas, quel est le meilleur/pire cas de test pour chaque algo)

Prérequis

Install Docker
https://www.docker.com/get-started

Démos - TD

Test de performance

- intégration(locust)

Démo de locust avec serveur local et plusieurs API

TD

test unitaires (composant) fonctionnels

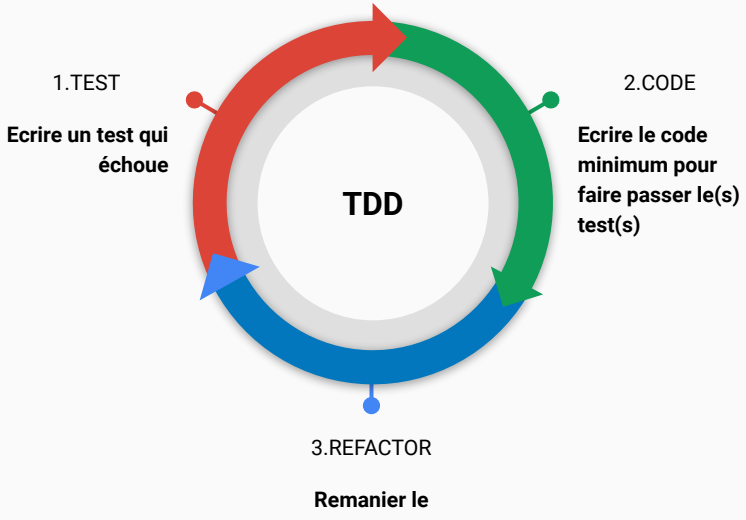
Code coverage

Exercice 1(stringlib):

- Sur un exemple ajouter des tests et calculer la couverture de code
- fournir le code, les tests et le résultat du coverage

Exercice 2 TDD (hello world and calculator):

- fournir le code, les tests et le résultat du coverage



Prérequis

Installer Postman desktop application
https://www.postman.com/

TD

test d’intégration fonctionnel

Récupérer swagger.json sur https://petstore.swagger.io/

Importer swagger.json dans postman

Créer les tests demandés, penser à sauver et exporter la collection postman.
Fournir la collection postman

Prérequis

Installer selenium web driver sous chrome
https://sites.google.com/chromium.org/driver/

Installer selenium ide
https://www.selenium.dev/selenium-ide/

Prérequis

Fournir les tests selenium (side / python)

360

360

Vos retours sur la formation.

- adéquation par rapport à votre situation professionnelle actuelle / future
- Dans quelle délai comptez-vous mettre en pratique ce que vous avez appris dans le cadre projet / études / pro ?

- Contenu cours
- TD
- Pistes d’améliorations

QCM 1h

Certaines questions nécessitent plusieurs réponses