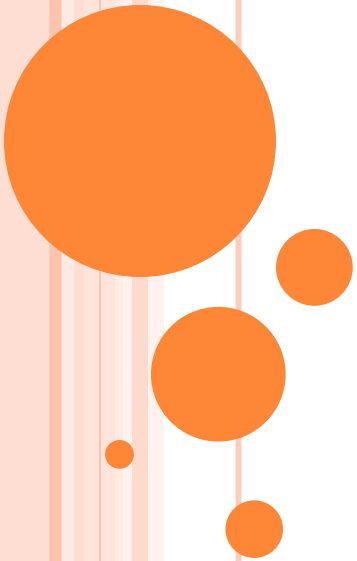


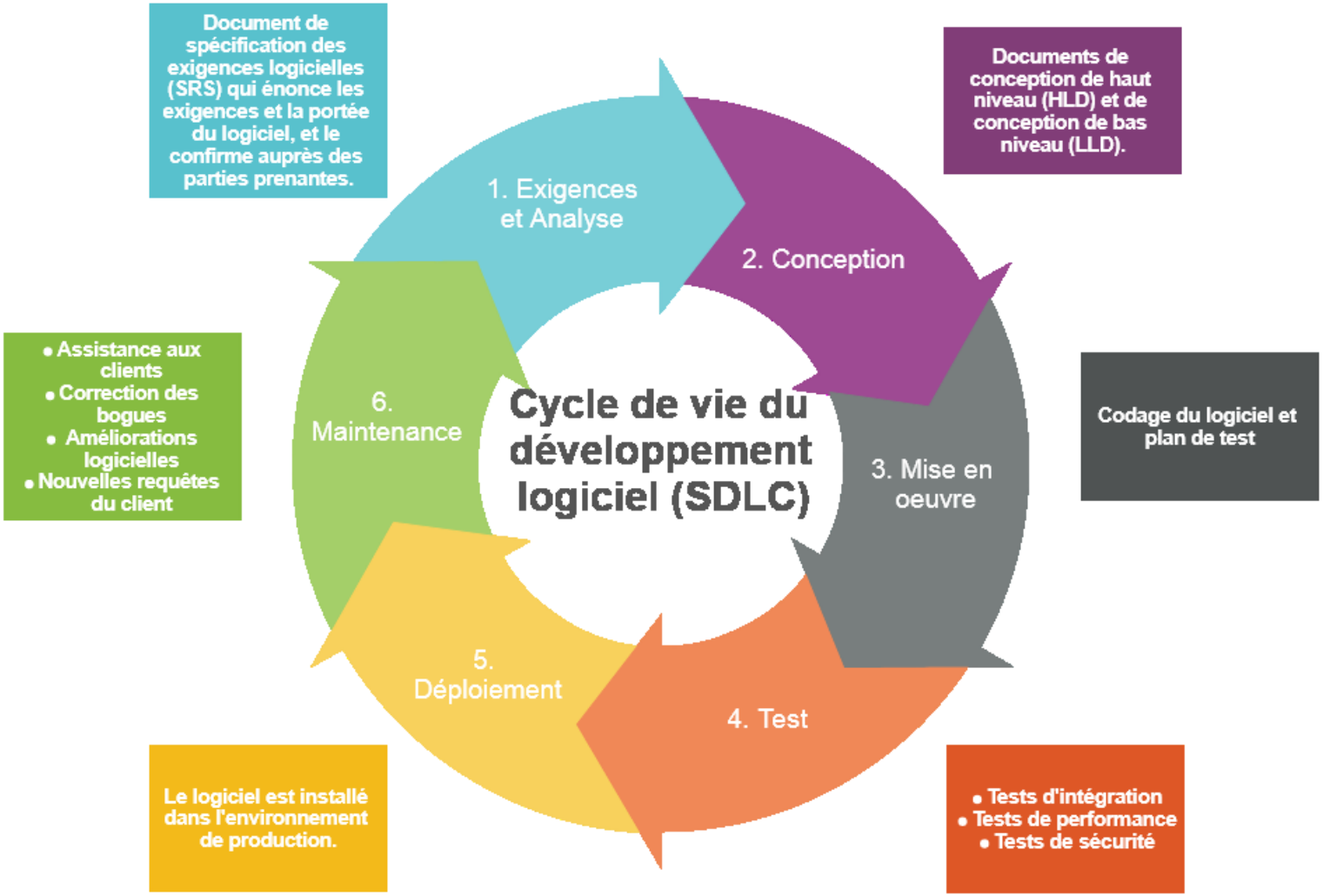
Chapitre 2 : Tester tout au long du cycle de vie du projet

Sommaire

- Impact du cycle de vie du développement logiciel sur le test
- Le test en tant que moteur du développement de logiciels
- DevOps et Tests
- Approche ShiftLeft
- Rétrospectives et amélioration de Processus
- Niveau de test
- Type de test

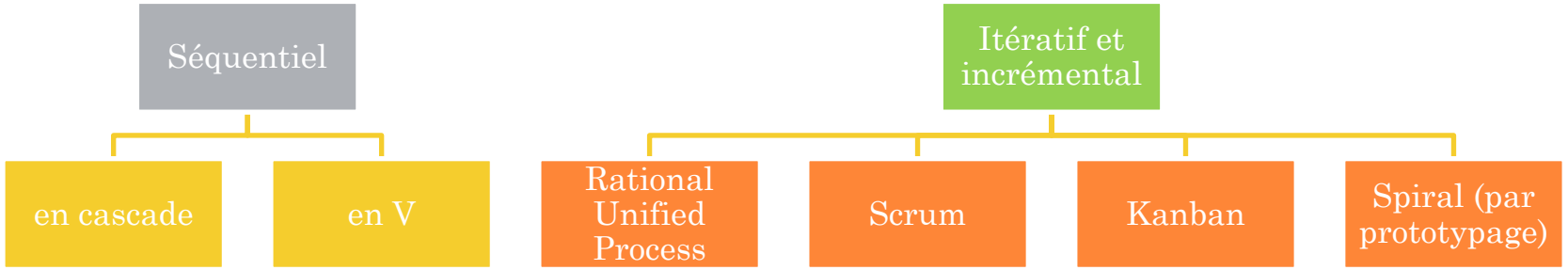
Impact du cycle de vie du développement logiciel sur le test

- Le modèle de cycle de vie du développement logiciel représente le processus de développement logiciel de façon abstraite (haut niveau).



Impact du cycle de vie du développement logiciel sur le test

Types de modèles de développement



Un enchaînement linéaire (séquentiel) d'activités visant à livrer un logiciel intégrant toutes les fonctionnalités prévues.

Élaboration des exigences, conception, développement et test du système par étapes, pour fournir plusieurs versions avec des fonctionnalités qui s'enrichissent progressivement (incrémental).

Impact du cycle de vie du développement logiciel sur le test

- Le choix du cycle de vie du développement logiciel a une incidence sur:

- Le périmètre et le calendrier des activités de test (par exemple, les niveaux de test et les types de test).
- Le niveau de détail de la documentation des tests.
- Le choix des techniques de test et de l'approche de test.
- Le degré d'automatisation des tests.
- Le rôle et les responsabilités d'un testeur.

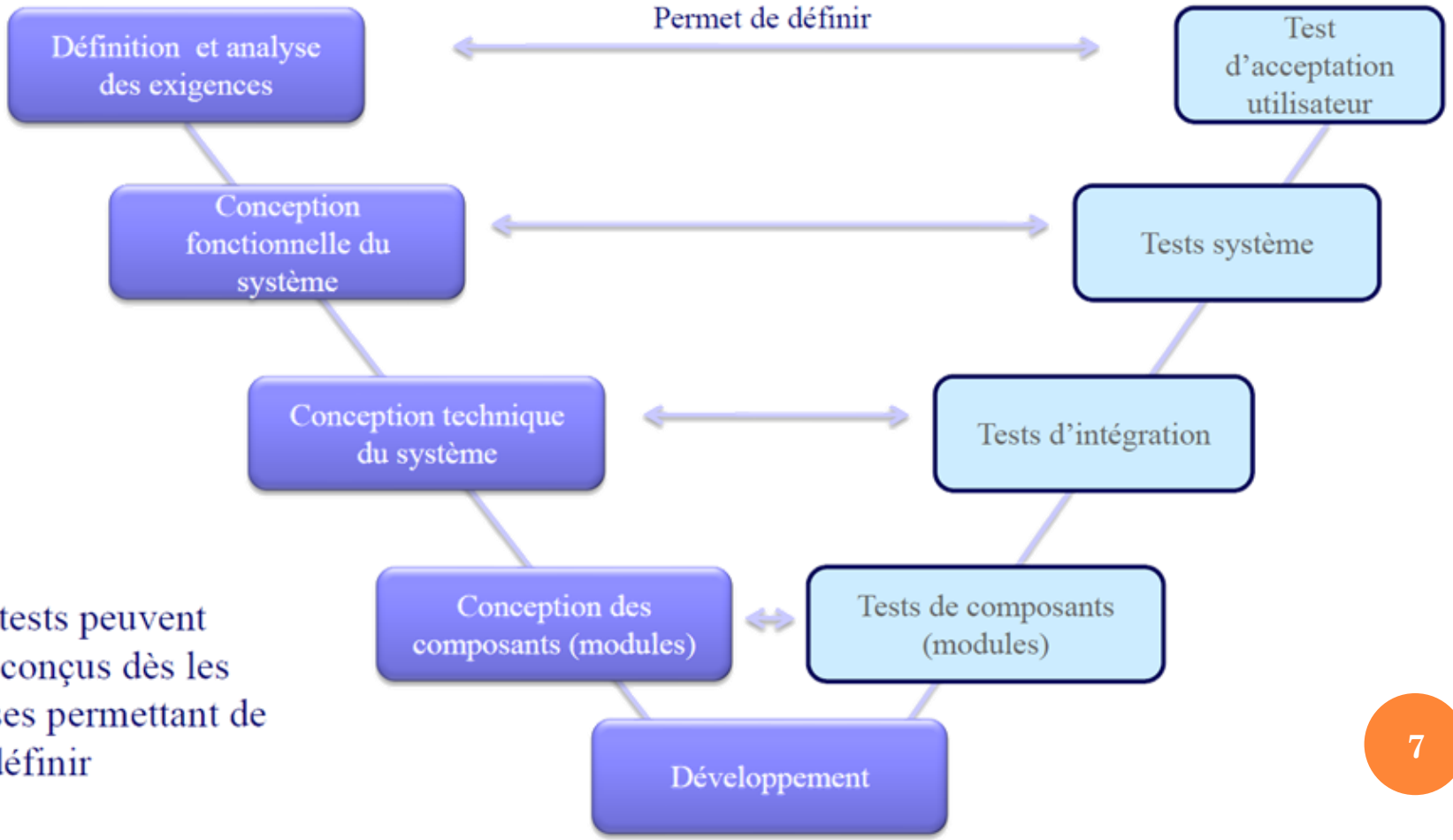
Impact du cycle de vie du développement logiciel sur le test

- Cycle séquentiel - En cascade (Waterfall model)

Les activités de test n'ont lieu qu'à la fin du projet

Impact du cycle de vie du développement logiciel sur le test

- Cycle séquentiel - En V



Les tests peuvent être conçus dès les phases permettant de les définir

Impact du cycle de vie du développement logiciel sur le test

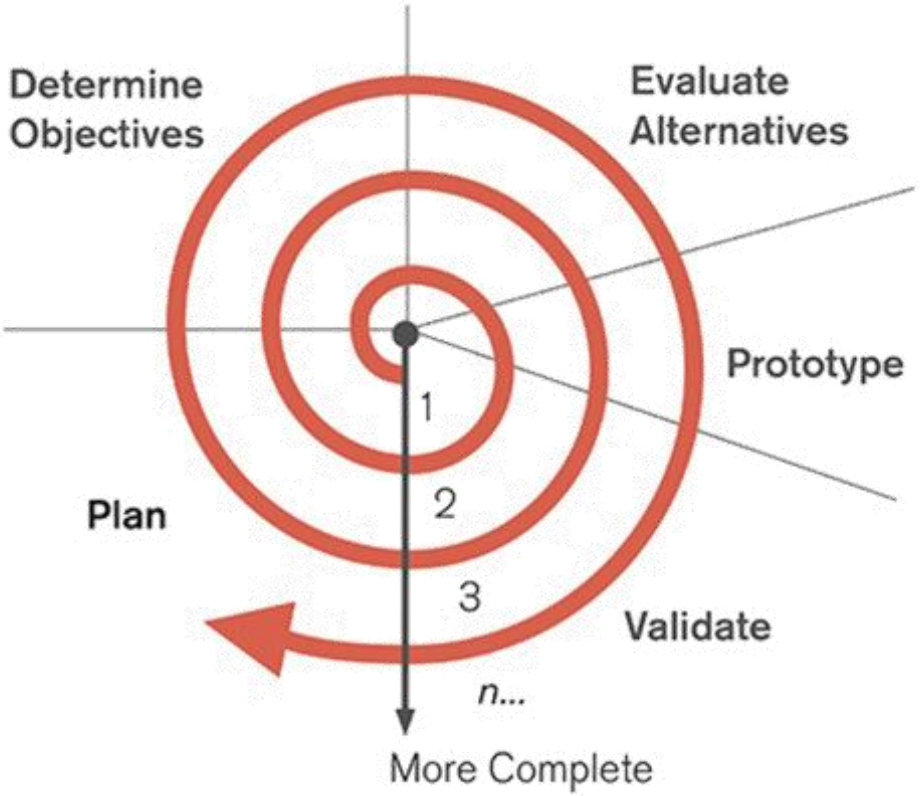
Dessiner la Joconde de manière incrémentale

Dessiner la Joconde de manière itérative

urce des images : Jeff Patton)

Impact du cycle de vie du développement logiciel sur le test

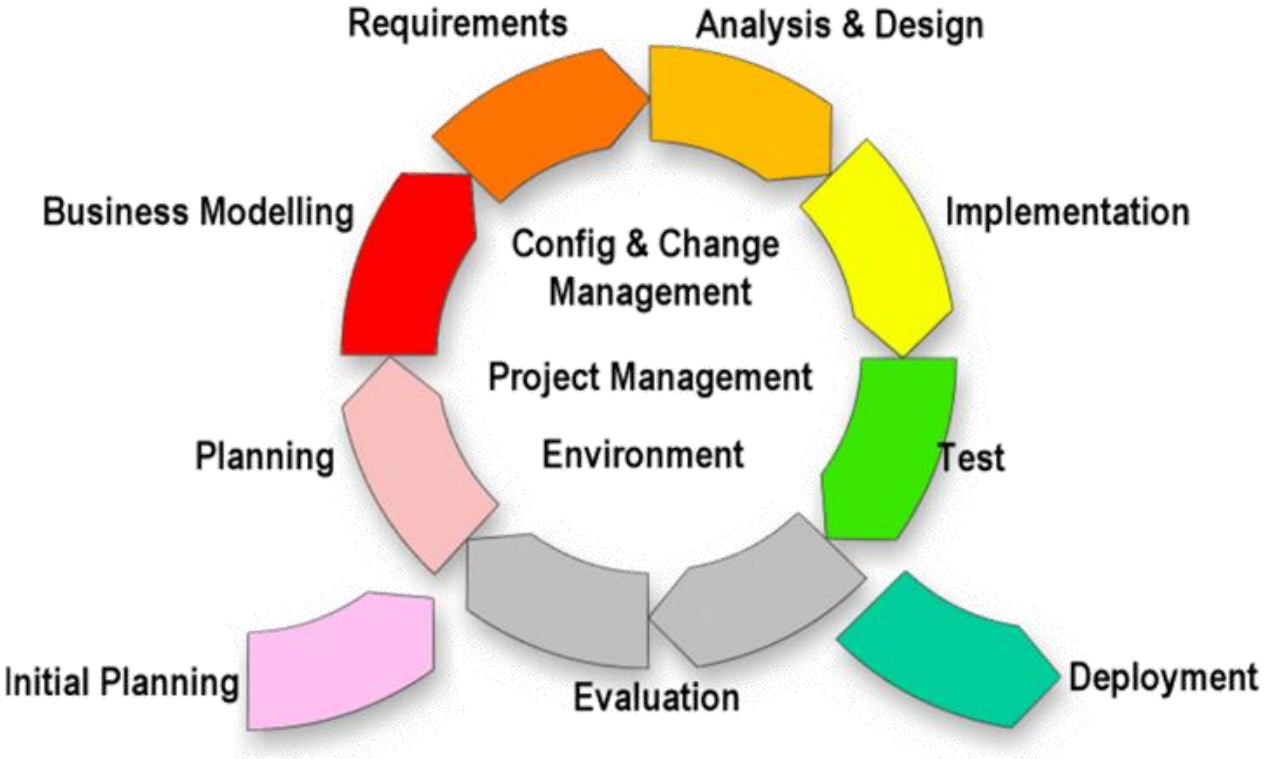
- Cycle itératif - Spiral



- Fondé sur une succession de prototypes pour converger vers un système satisfaisant les besoins

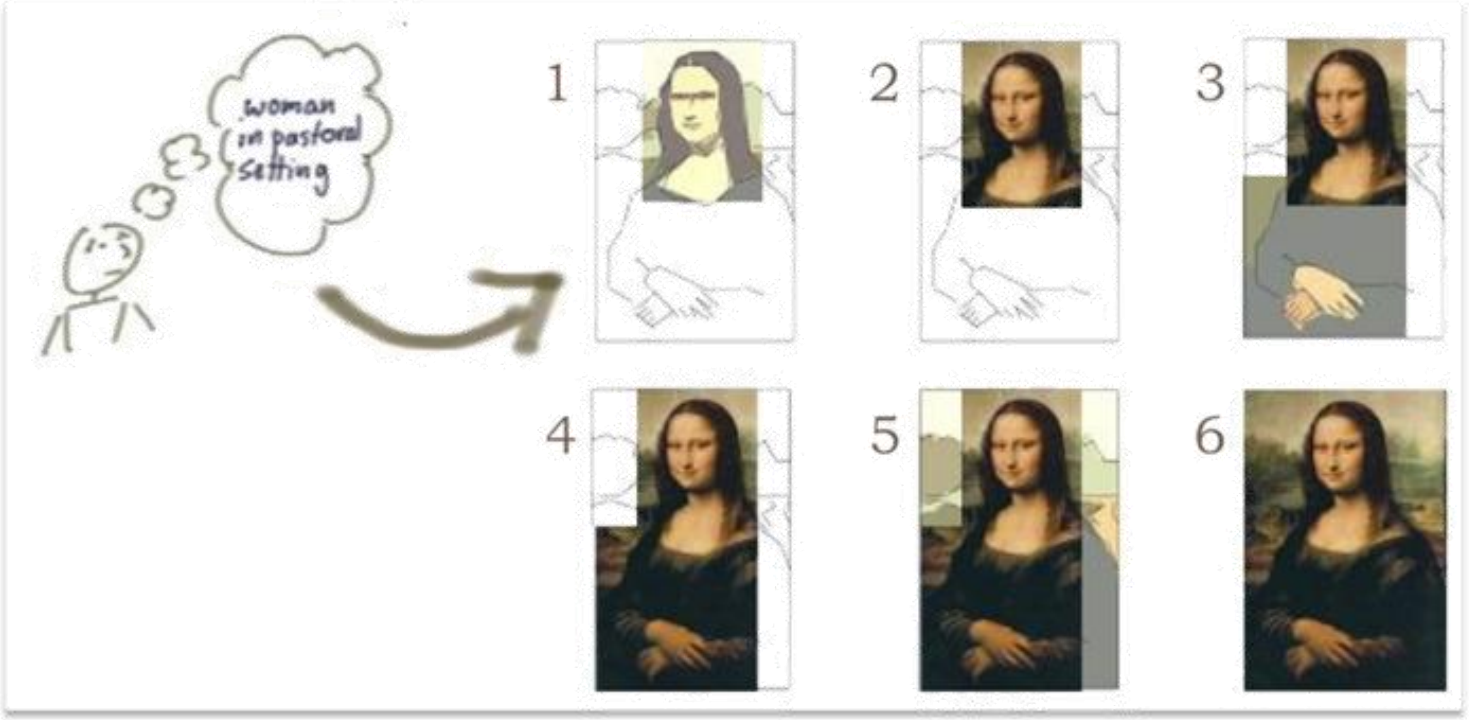
Impact du cycle de vie du développement logiciel sur le test

- Cycle incrémental - Rational Unified Process (RUP)



- Les itérations sont assez longues (typiquement 2 à 3 mois dans RUP)

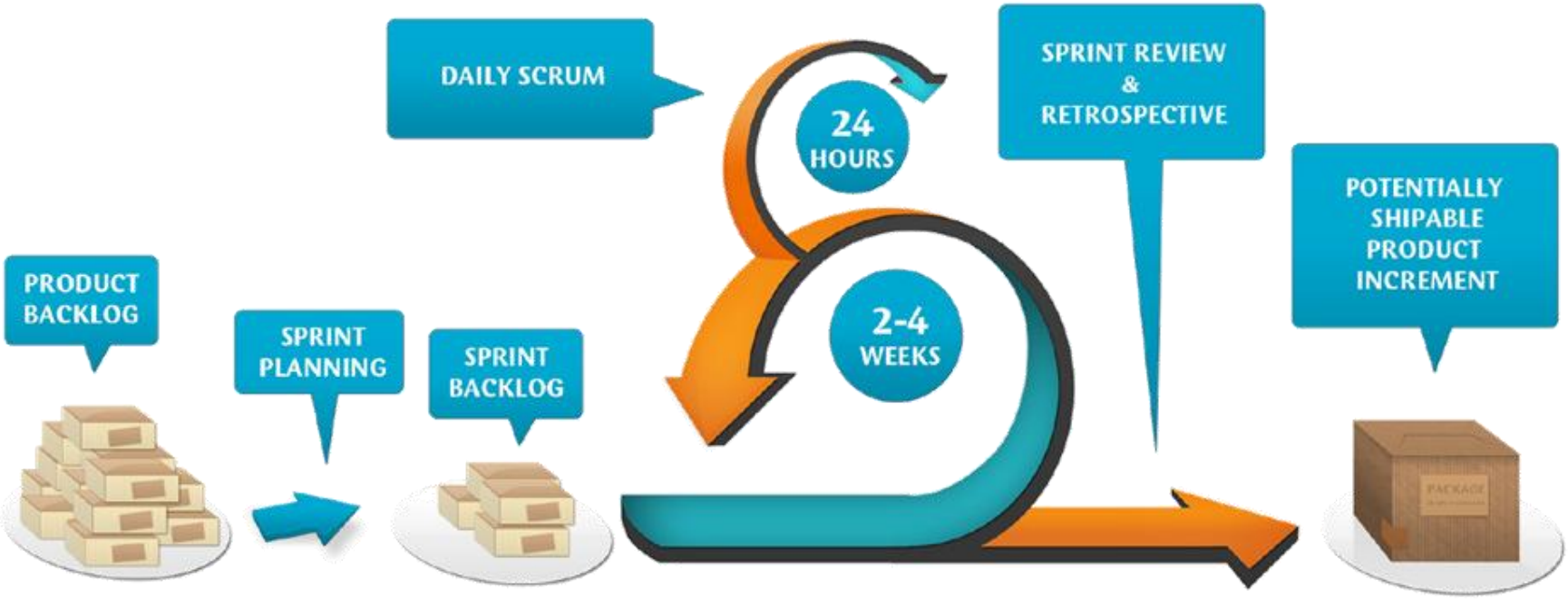
Impact du cycle de vie du développement logiciel sur le test



Dessiner la Joconde de manière itérative et incrémentale

Impact du cycle de vie du développement logiciel sur le test

- Cycle itératif et incrémental – SCRUM



- Les différents niveaux de test sont réalisés à l'intérieur des itérations.

Impact du cycle de vie du développement logiciel sur le test

- Cycle itératif et incrémental – Kanban

- Implémenté avec ou sans itérations de longueur fixe, qui peut fournir soit une seule amélioration ou fonctionnalité une fois achevée, soit regrouper les fonctionnalités pour mors dans une version.

Impact du cycle de vie du développement logiciel sur le test

- Le choix du cycle de vie du développement logiciel a une incidence sur:

  - Le périmètre et le calendrier des activités de test (par exemple, les niveaux de test et les types de test).
  - Le niveau de détail de la documentation des tests.
  - Le choix des techniques de test et de l'approche de test.
  - Le degré d'automatisation des tests.
  - Le rôle et les responsabilités d'un testeur.

- Il faut sélectionner le modèle correspondant au contexte, puis l'adapter si nécessaire.

Le test en tant que moteur du développement de logiciels

TDD : Test Driven Development

ATDD : Acceptance Test Driven Development

BDD : Behavioral Driven Development

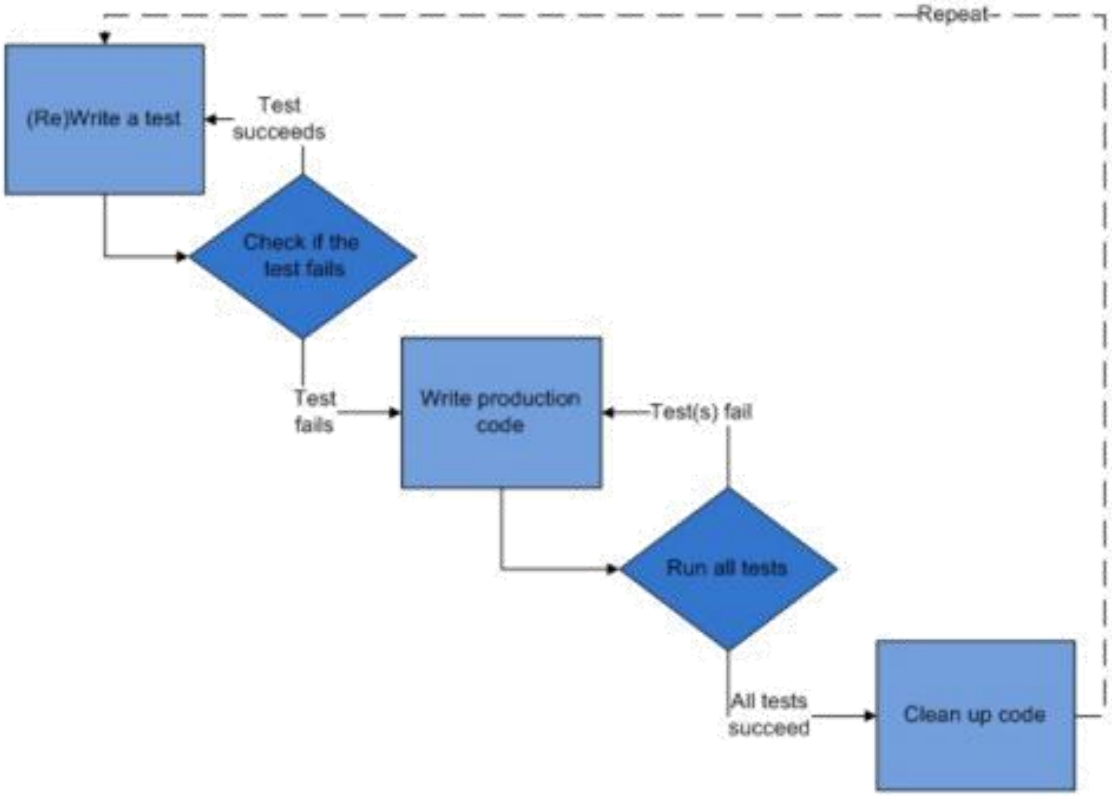
Ce sont des approches de développement similaires qui implémentent le principe des tests précoces:

- Les tests sont définis avant le code

Le test en tant que moteur du développement de logiciels

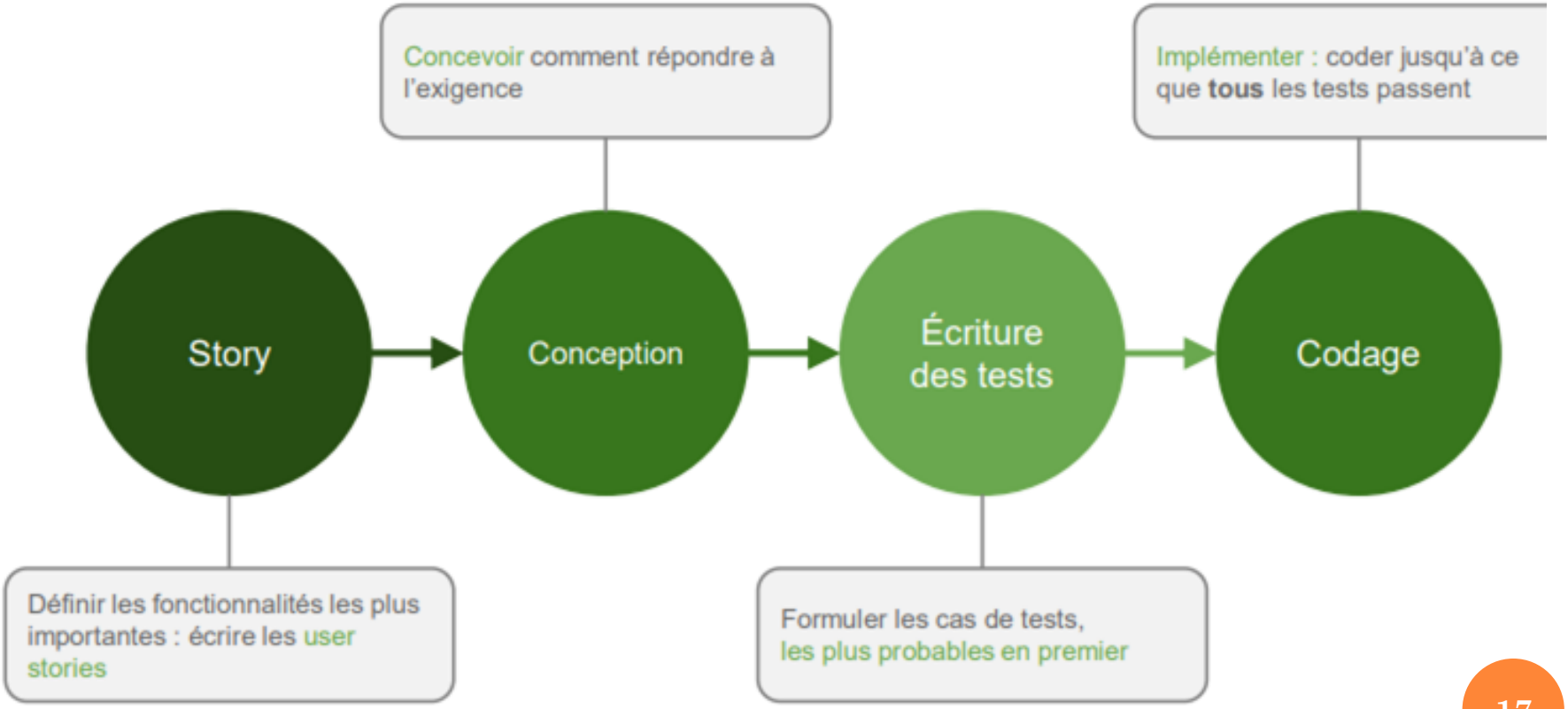
Développement piloté par les tests (TDD) :

- Ecrire les tests unitaires d'une fonction, méthode ou procédure avant d'écrire son code
- Piloter le développement par les tests
- Permet de réfléchir au rôle du code testé avant de l'écrire



Le test en tant que moteur du développement de logiciels

Développement piloté par les tests d'acceptation (ATDD) :



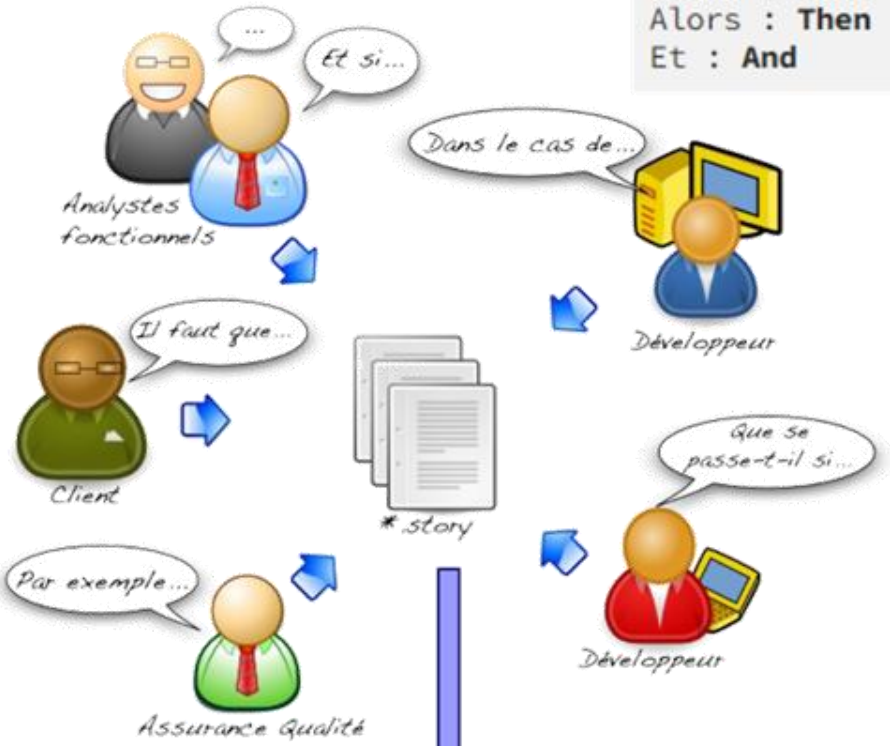
Le test en tant que moteur du développement de logiciels

Développement piloté par le comportement (BDD) :

Aligner les membres d'un projet :

- Créer une terminologie métier commune
- Définir le périmètre d'une fonctionnalité
- Obtenir une documentation vivante du produit

```
Étant donné : Given
Quand : When
Alors : Then
Et : And
```

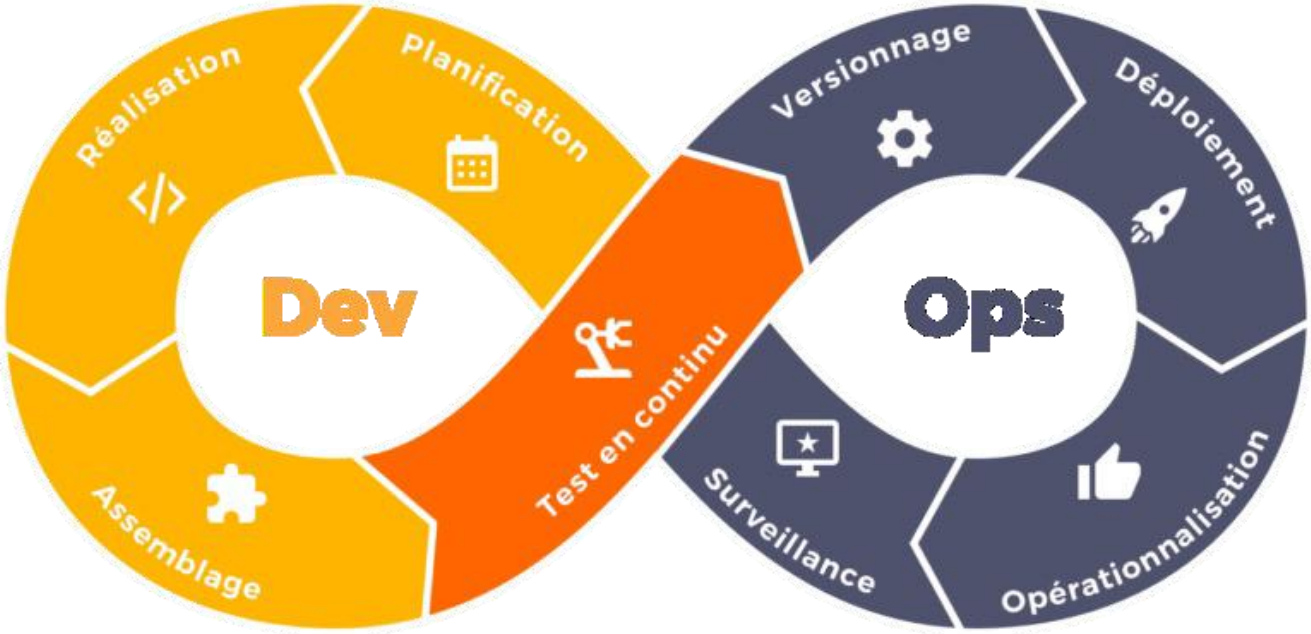



```
Étant donné que je visite "/login"
  Quand j'entre "Bruno" dans le champ "nom d'utilisateur"
    Et j'entre "testeur" dans le champ "mot de passe"
    Et j'appuie sur le bouton "connexion"
  Alors, je devrais voir la page "Accueil"
```

DevOps et tests

- Le DevOps est approche organisationnelle qui vise à concilier deux corps de métier : le développeur logiciel (dev) d'une part, l'administrateur en charge des opérations informatiques (ops) d'autre part.

- Un développement constant
- Des tests en continu
- Une intégration continue
- Une mise en oeuvre continue
- Un monitoring permanent



DevOps et tests

Quelques avantages du devops :

- Feedback rapide sur la qualité du code
- Encourager les développeurs à soumettre un code de haute qualité accompagné de tests
- Favoriser le développement et l'intégration continue
- Encourager l'automatisation des tests

» Moins de tests manuels répétitifs

» Tests de régression exécutés plus souvent

Quelques inconvénients du devops :

- Coût de mise en place élevé
- Maintenance de la chaine de développement et d'intégration continue

Des tests manuels seront toujours nécessaires.

Approche Shift Left

- Le principe du test précoce est parfois appelé shift left parce qu'il s'agit d'une approche dans laquelle le test est effectué plus tôt dans le cycle de vie du développement logiciel.

- Il ne signifie pas que les tests doivent être négligés dans les phases ultérieures du cycle de vie du développement logiciel.

Approche Shift Left

Quelques bonnes pratiques du « shift left » dans le test :

- **Examiner la spécification du point de vue des tests. Ces** activités de revue des spécifications permettent souvent de trouver des défauts potentiels, tels que des ambiguïtés, des incomplétudes et des incohérences.

- **Rédiger des cas de test avant l'écriture du code et faire exécuter** le code dans un harnais de test pendant l'implémentation du code.

- **Utiliser l'Intégration Continue et le Développement Continu,** permettant un feedback rapide et des tests de composants automatisés pour accompagner le code source lorsqu'il est déposé dans le référentiel de code.

- **Clôturer les tests statiques du code source avant les tests dynamiques, ou dans le cadre d'un processus automatisé**

- **Effectuer des tests non fonctionnels en commençant par le niveau de test de composants, dans la mesure du possible. Il** s'agit d'une forme de shift left, car ces types de tests non fonctionnels ont tendance à être réalisés plus tard dans le cycle de vie du développement logiciel, lorsqu'un système complet et un environnement de test représentatif sont disponibles.

Rétrospectives et amélioration de processus

Réunion organisée à la fin d'une étape du projet (cycle V) ou d'une itération/release (cycle Agile)

Bilan pour une amélioration continue

- Analyse des points positifs et négatifs
- Qu'avons-nous appris ?
- Y a-t-il des irritants ?
- Identification d'axes d'amélioration

Sujets abordés

- Efficacité des processus
- Les personnes et les relations
- L'organisation
- Les outils

Prise en compte des aspects spécifiques au test

- Efficacité et efficience des tests
- Qualité des cas de test
- Problèmes de testabilité

Rétrospectives et amélioration de processus

Approche Shift Left

Quelques bonnes pratiques du « shift left » dans le test :

- **Examiner la spécification du point de vue des tests. Ces** activités de revue des spécifications permettent souvent de trouver des défauts potentiels, tels que des ambiguïtés, des incomplétudes et des incohérences.

- **Rédiger des cas de test avant l'écriture du code et faire exécuter** le code dans un harnais de test pendant l'implémentation du code.

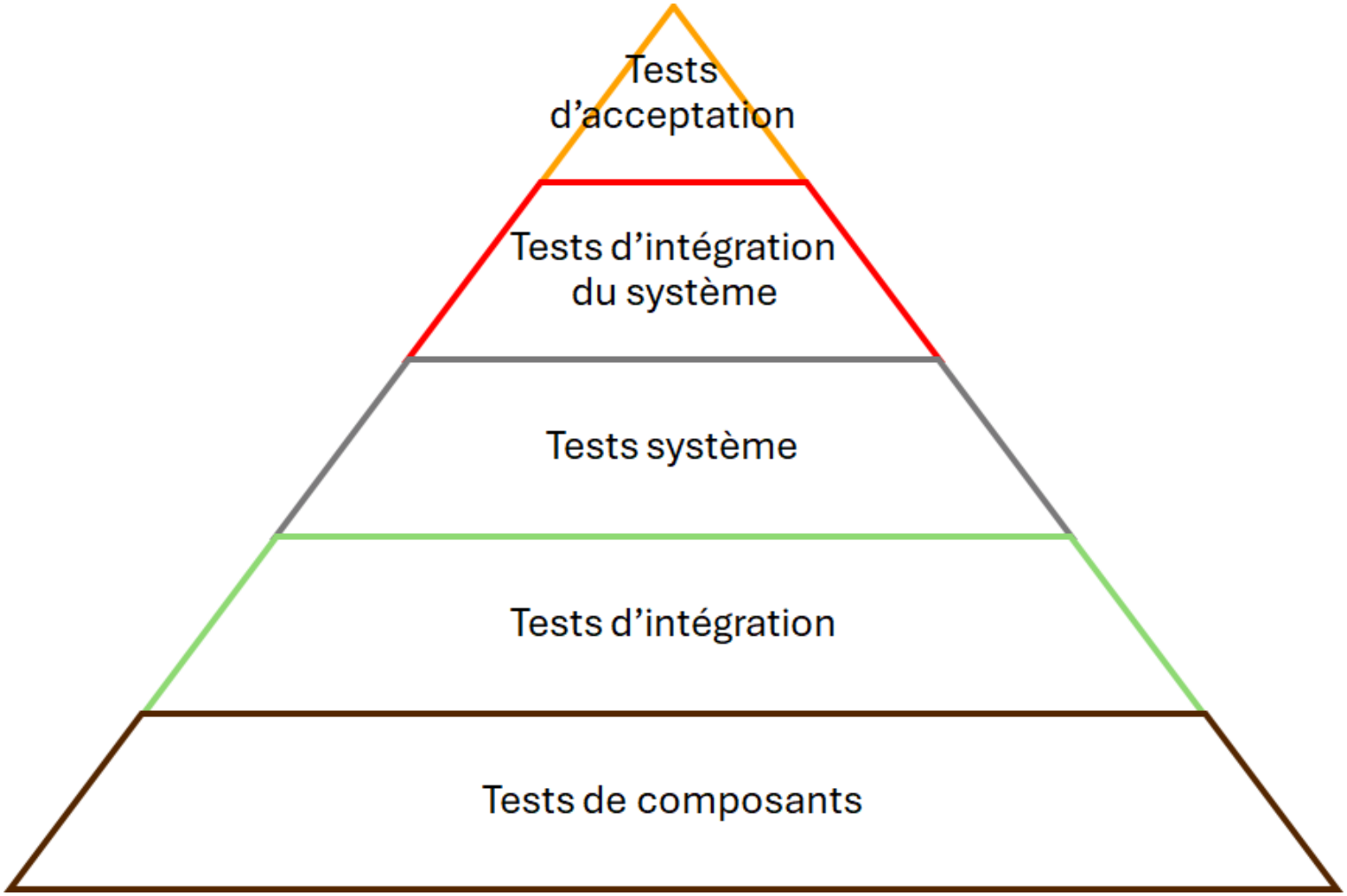
- **Utiliser l'Intégration Continue et le Développement Continu,** permettant un feedback rapide et des tests de composants automatisés pour accompagner le code source lorsqu'il est déposé dans le référentiel de code.

- **Clôturer les tests statiques du code source avant les tests dynamiques, ou dans le cadre d'un processus automatisé**

- **Effectuer des tests non fonctionnels en commençant par le niveau de test de composants, dans la mesure du possible. Il** s'agit d'une forme de shift left, car ces types de tests non fonctionnels ont tendance à être réalisés plus tard dans le cycle de vie du développement logiciel, lorsqu'un système complet et un environnement de test représentatif sont disponibles.

Niveau de test

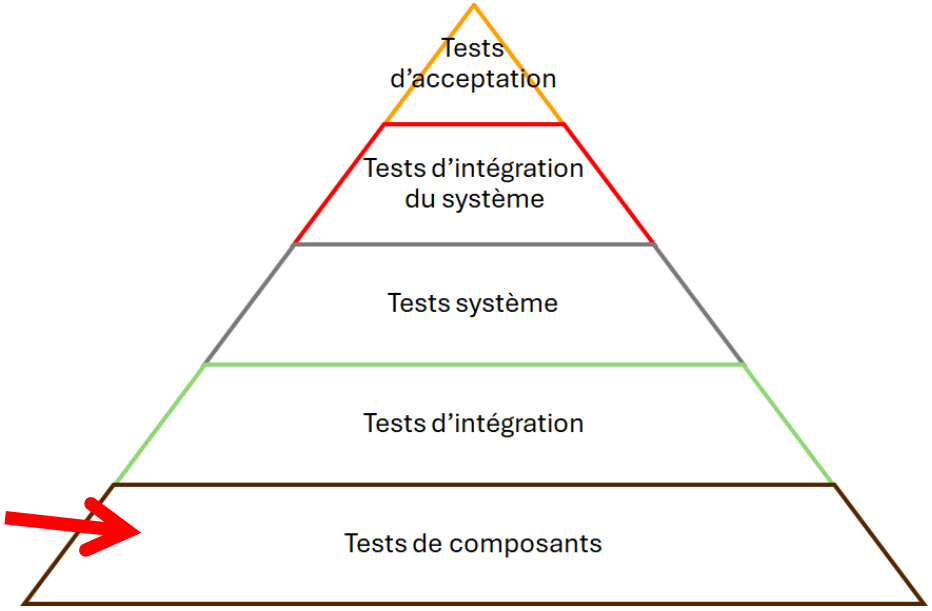
- le test ne se résume pas juste aux tests fonctionnels.
- Ces niveaux de tests peuvent être représentés sous la forme de la pyramide ci-dessous :



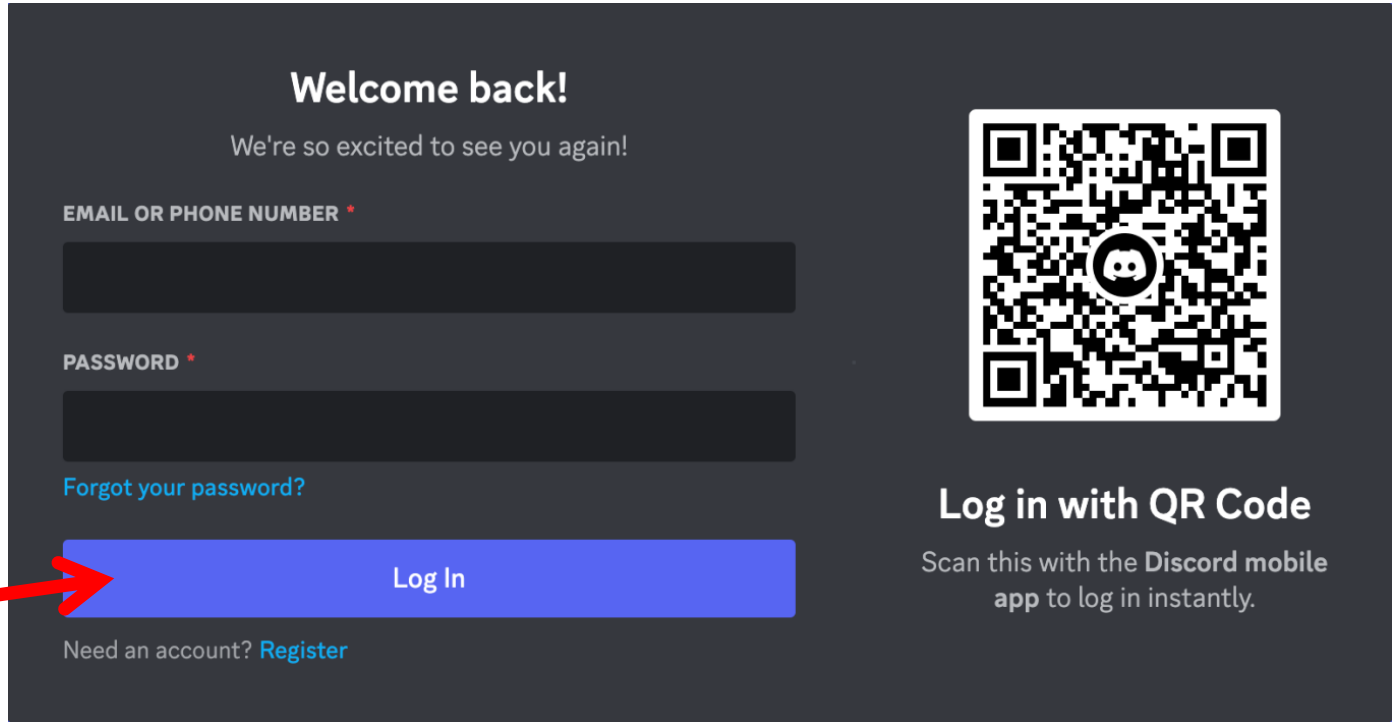
Niveau de test – Test de Composants

- Les tests de composants ont pour but de tester les différents composants du logiciel séparément afin de s'assurer que chaque élément fonctionne comme spécifié. Ces tests sont très souvent l'équivalent des tests unitaires. Dans ce cas ils sont généralement écrits et exécutés par le développeur qui a écrit le code du composant.

- Lorsque ces tests sont bien les tests unitaires il est important de les automatiser lorsque c'est possible.



Niveau de test – Test de Composants



Pour une authentification, le bouton « se connecter » peut être vu comme un composant.

Niveau de test – Test de Composants
Test bouchonnés

- Imaginons que l'on travaille sur MiamApp et qu'il faut tester une nouvelle fonctionnalité, comme le calcul du meilleur itinéraire pour les Buguinos. Cette fonctionnalité est déjà codée, mais elle dépend d'une autre partie du logiciel qui n'est pas encore terminée, comme la base de données des itinéraires ou le serveur de gestion des commandes.

- **Problème :**
  Comment tester la fonctionnalité si le reste du système n'est pas prêt ? C'est là qu'interviennent les **tests bouchonnés**, ou **stubs**. Un **bouchon** est une version simplifiée ou temporaire d'un composant logiciel. Il remplace une partie manquante ou non encore développée du système pour que l'on puisse tester ce qui est déjà prêt.

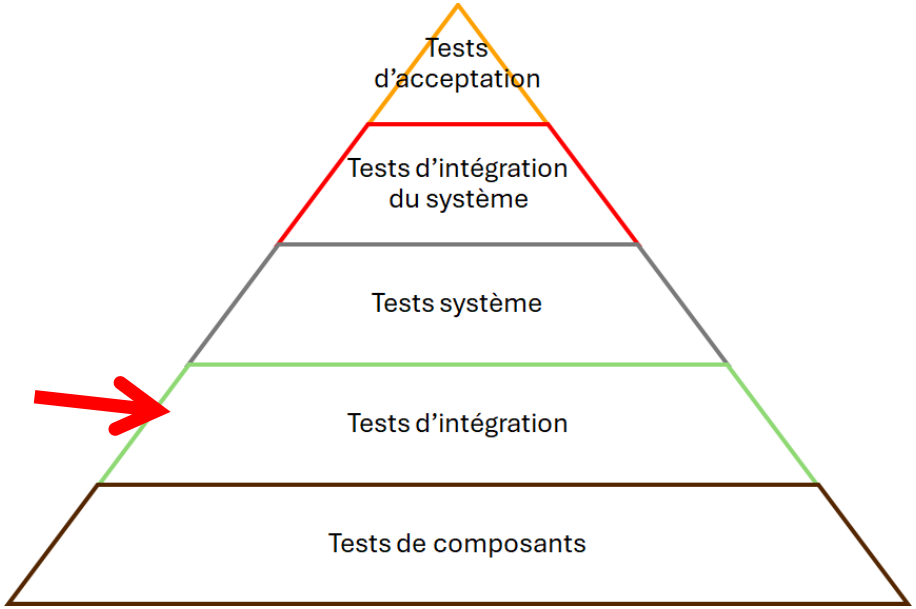
Le bouchon fait semblant d'être la base de données.

- Quand ta fonctionnalité demande un itinéraire au bouchon, il lui répond : "Ok, prends ce chemin tout droit !".
- Tu peux ainsi tester si ta fonctionnalité traite correctement l'information donnée.

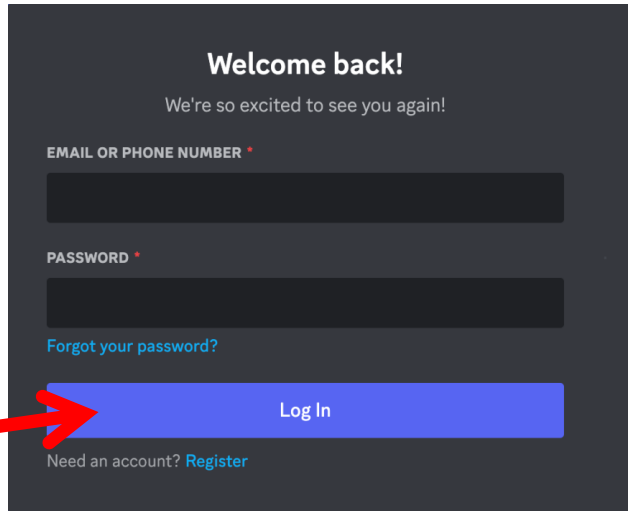
Niveau de test – Test d'intégration

- Les tests d'intégrations sont des tests effectués entre les composants afin de s'assurer du fonctionnement des interactions et de l'interface entre les différents composants. Ces tests sont également gérés, en général, par des développeurs.

- Stratégie d'intégration : ascendante, descendante ou big-bang.

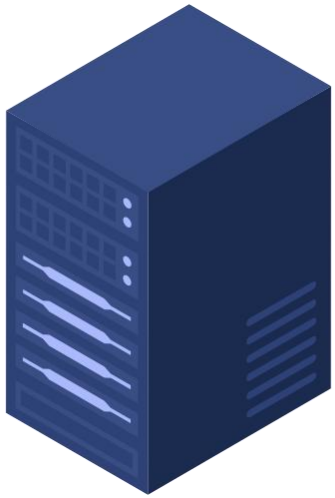


Niveau de test – Test de Composants



Clic sur le bouton « Log In »

message

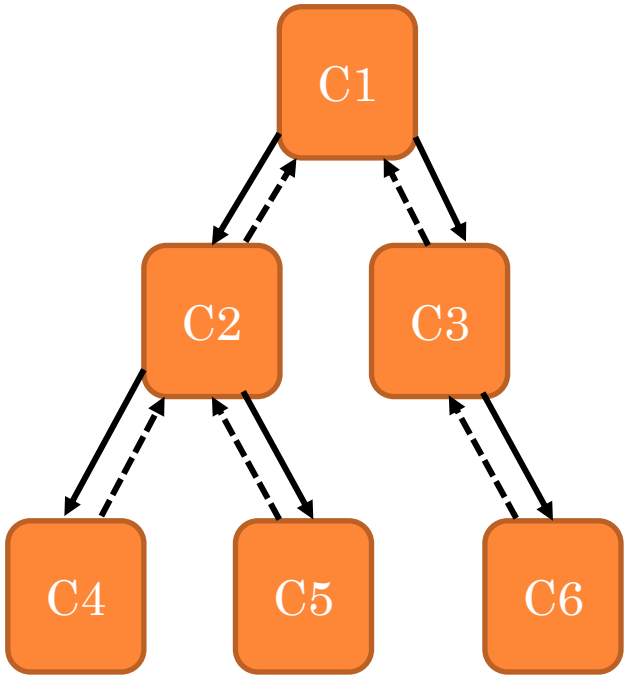


serveur d'authentification

Toujours depuis l'authentification ici on vérifie que le message envoyé après l'appui sur le bouton « se connecter » est bien reçu par le serveur d'authentification.

Niveau de test – Test d'intégration - big-bang

Approche « Big-Bang »



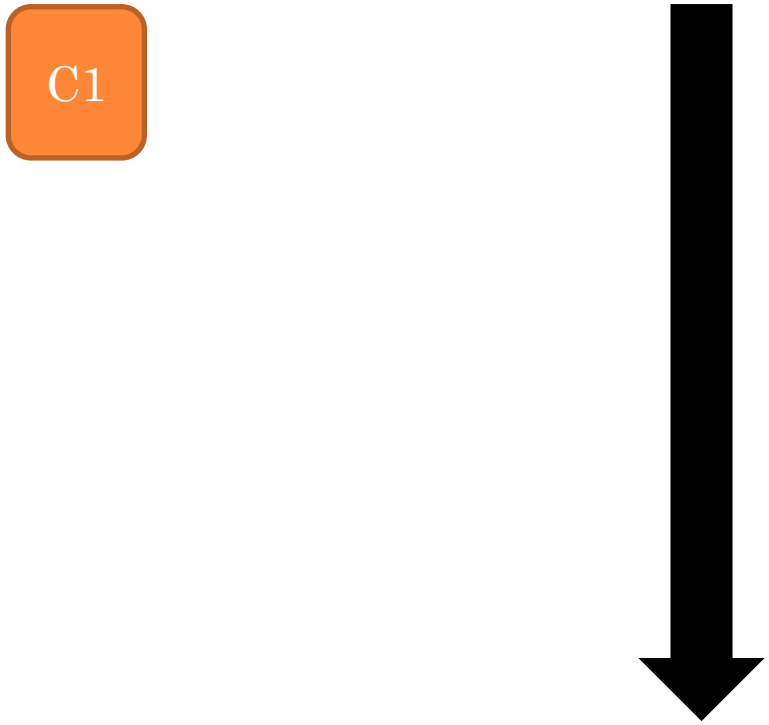
Tous les composants sont intégrés en même temps

A EVITER !

- Pratique peu sûr, les anomalies sont détectées très tard

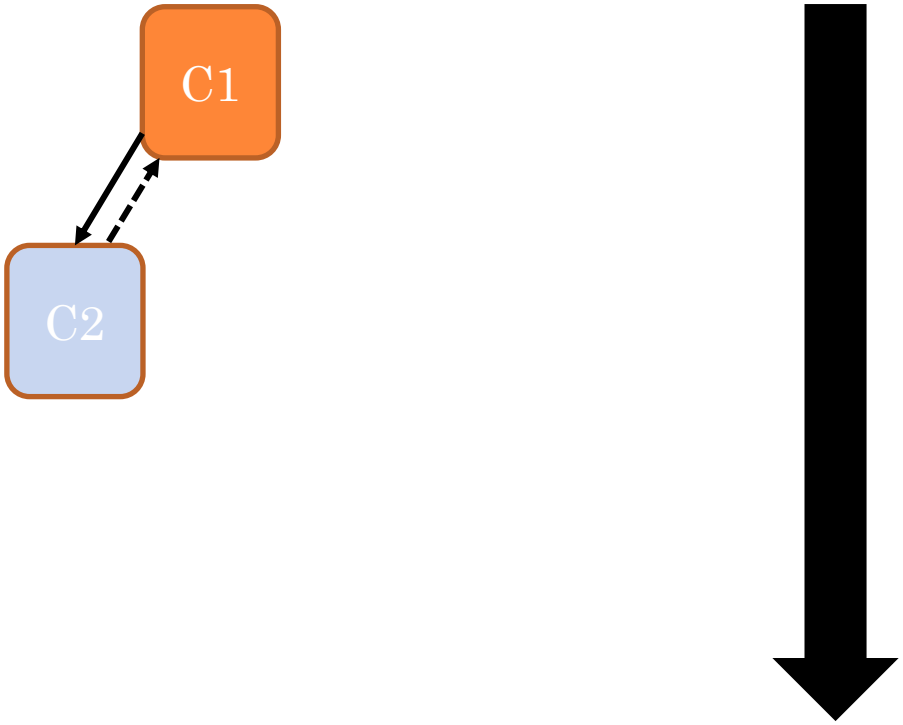
Niveau de test – Test d'intégration - ascendante

Approche « de haut en bas », avec des bouchons



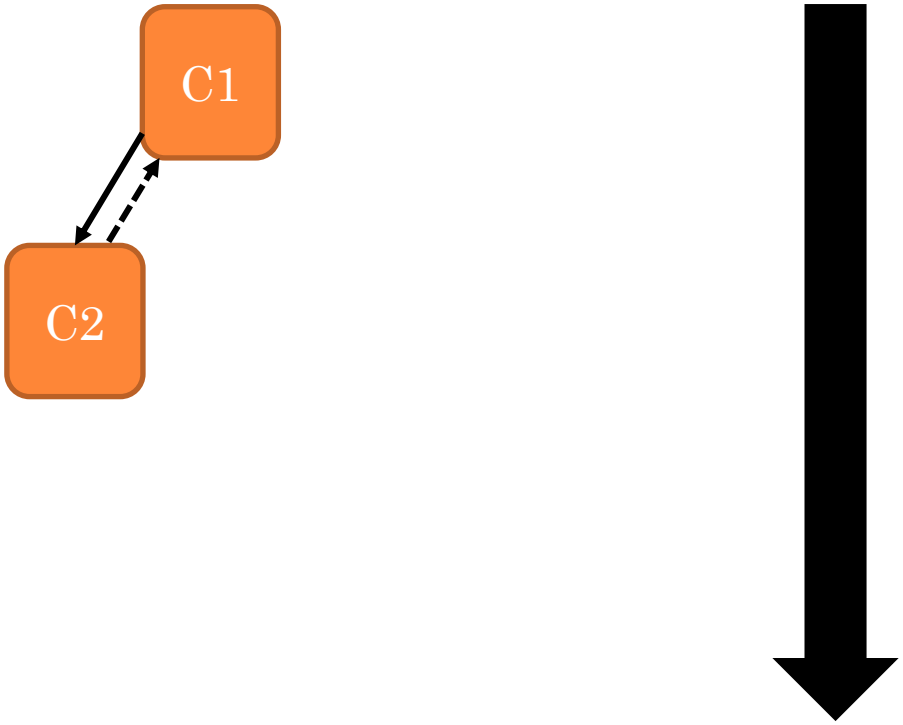
Niveau de test – Test d'intégration - ascendante

Approche « de haut en bas », avec des bouchons



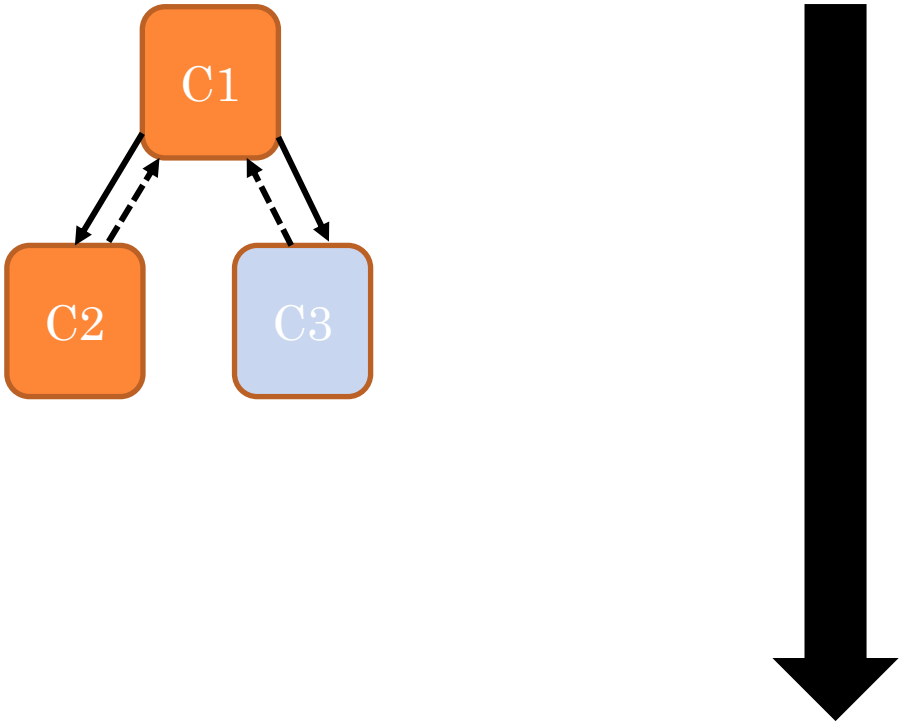
Niveau de test – Test d'intégration - ascendante

Approche « de haut en bas », avec des bouchons



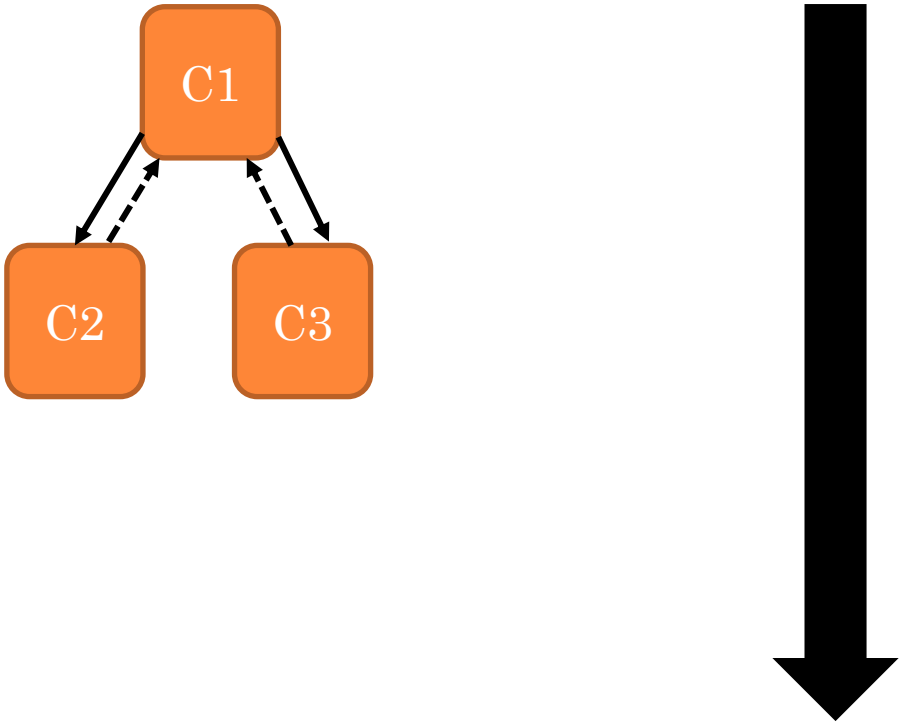
Niveau de test – Test d'intégration - ascendante

Approche « de haut en bas », avec des bouchons



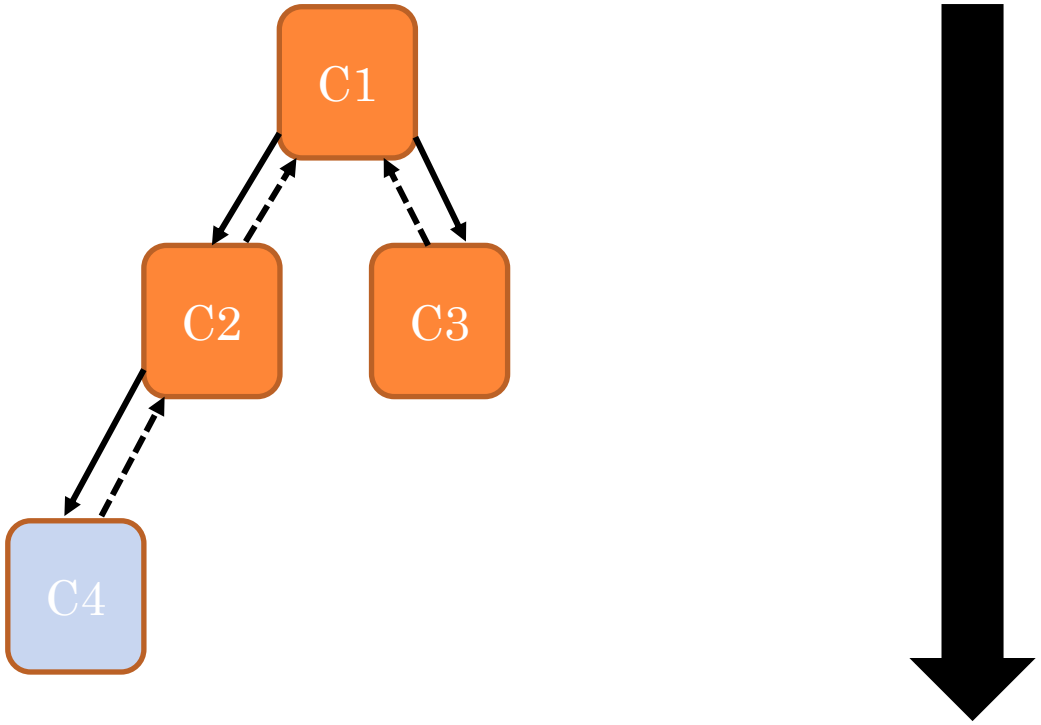
Niveau de test – Test d'intégration - ascendante

Approche « de haut en bas », avec des bouchons



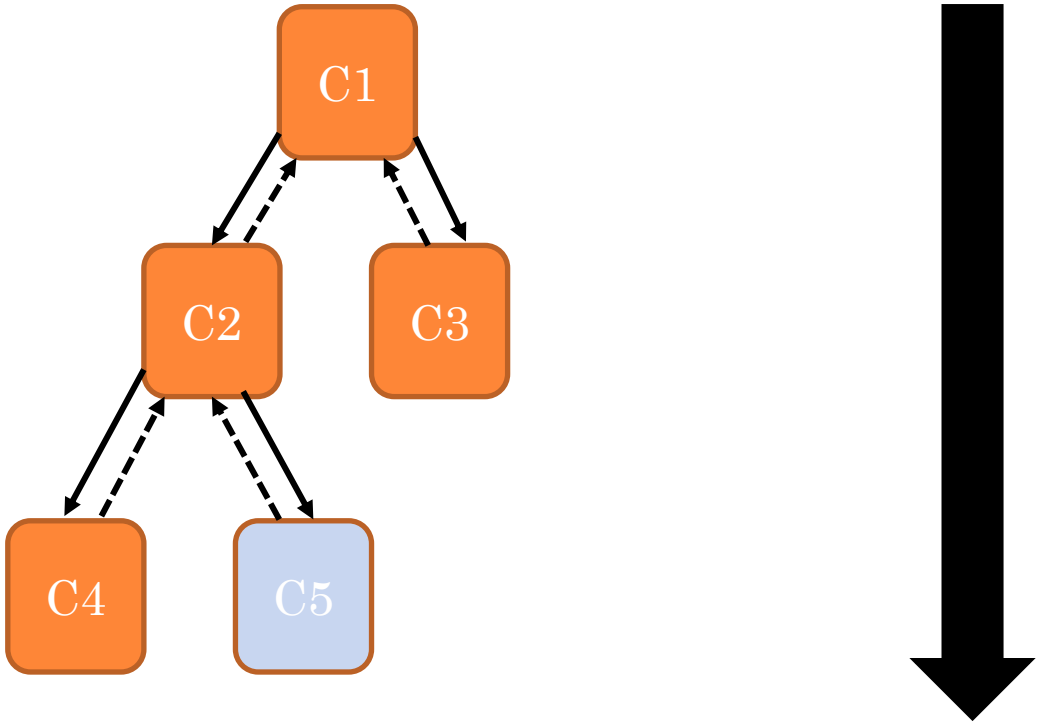
Niveau de test – Test d'intégration - ascendante

Approche « de haut en bas », avec des bouchons



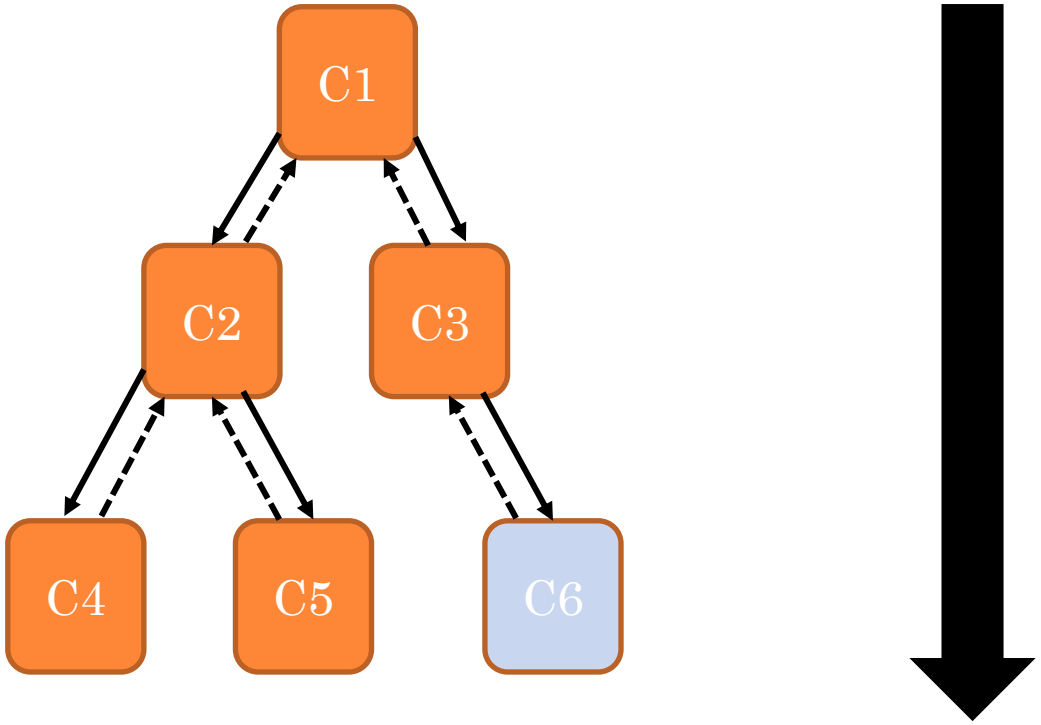
Niveau de test – Test d'intégration - ascendante

Approche « de haut en bas », avec des bouchons



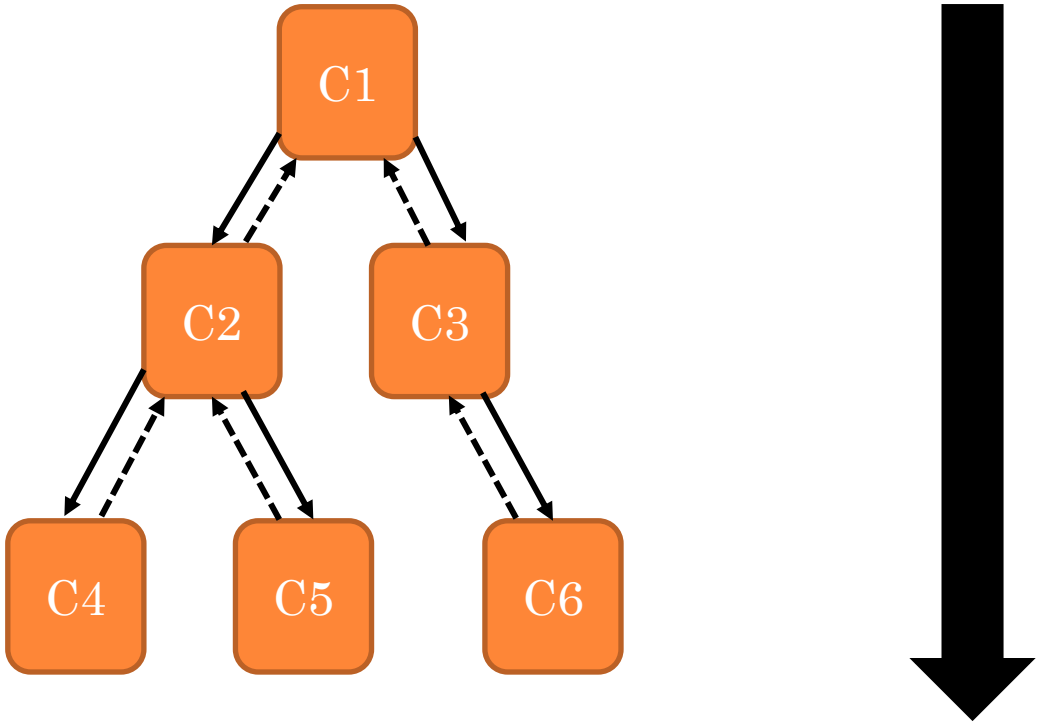
Niveau de test – Test d'intégration - ascendante

Approche « de haut en bas », avec des bouchons



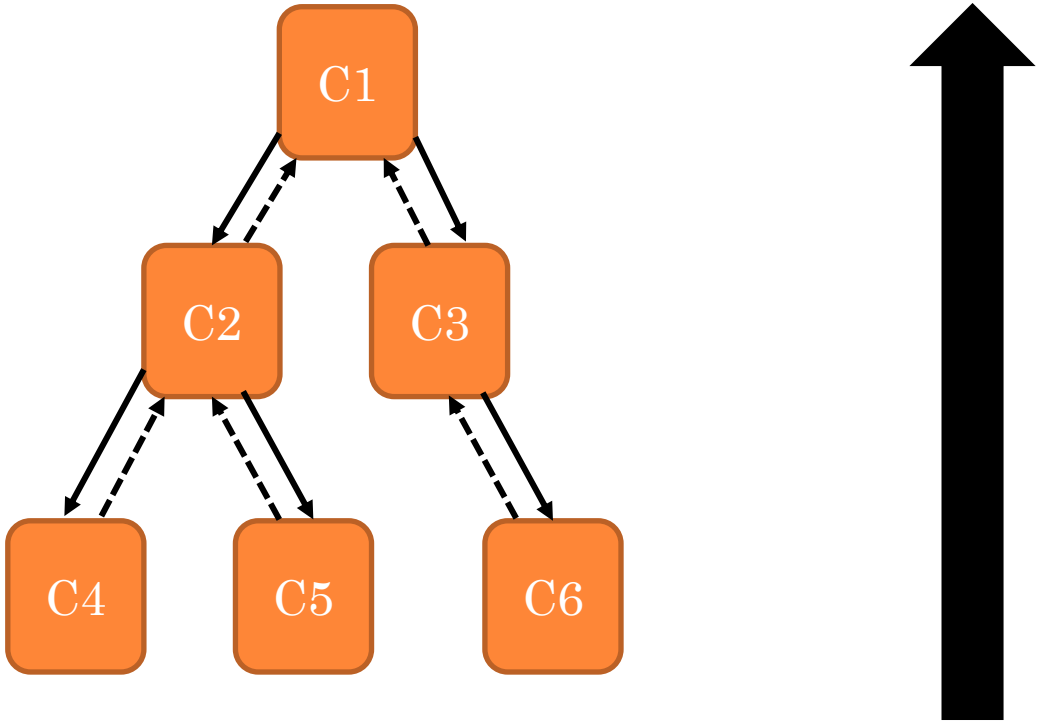
Niveau de test – Test d'intégration - ascendante

Approche « de haut en bas », avec des bouchons



Niveau de test – Test d'intégration - descendante

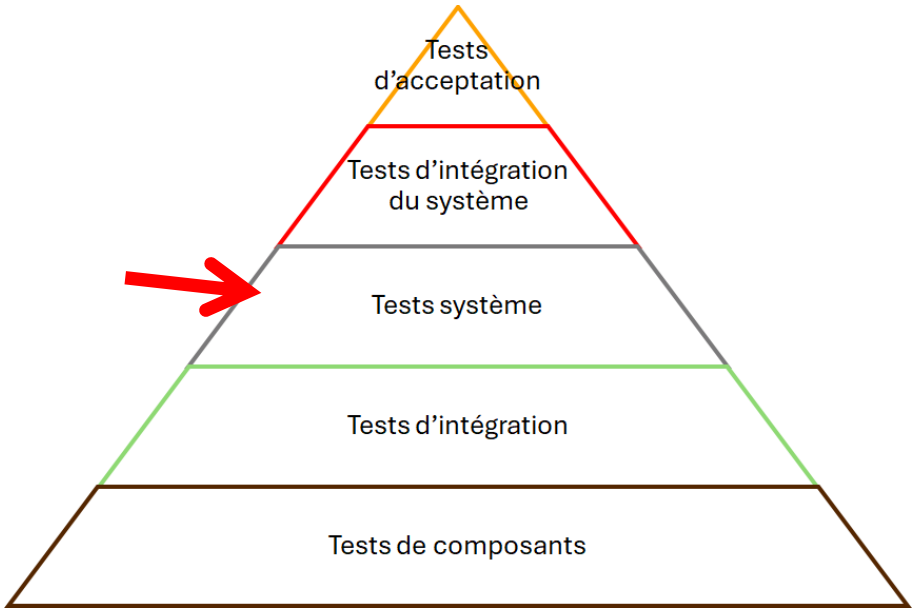
Approche « de bas en haut », avec des pilotes



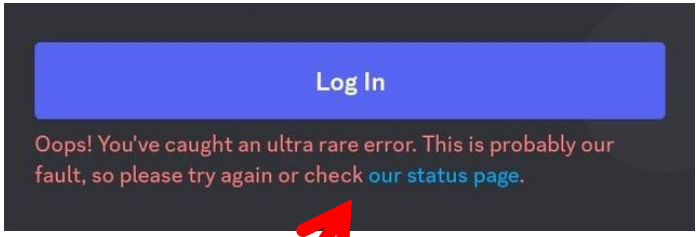
Niveau de test – Test Système

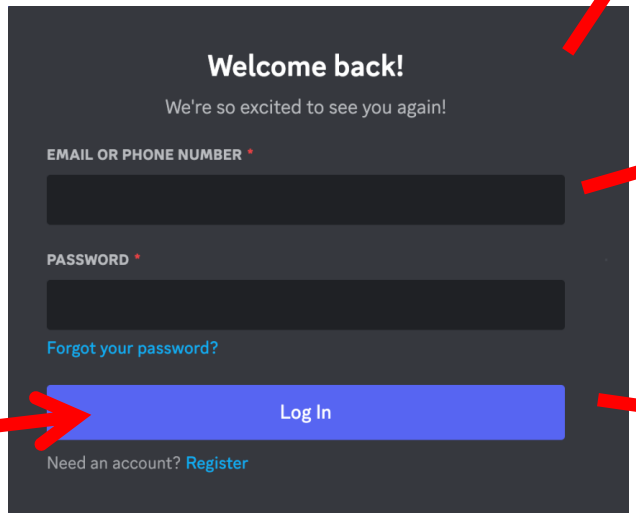
- C'est les tests au sens le plus instinctif et généralement les plus effectués par les ingénieurs de tests. Leurs but est de vérifier que le système (le logiciel ou l'application dans son ensemble) répond aux exigences définies dans les spécifications. On les appelle souvent tests fonctionnels même si c'est un abus de langage car il existe des tests « non fonctionnels» qui peuvent être spécifiés

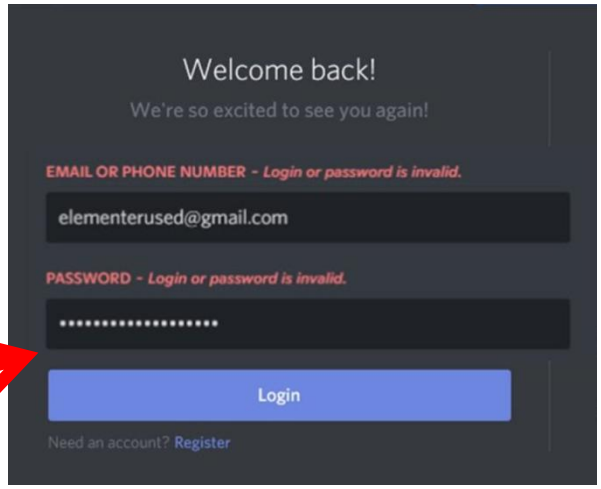
- Ces tests peuvent être manuels ou automatisés, en général un mixte de tests automatisés et de tests manuels est ce qui a le meilleur retour sur investissement.

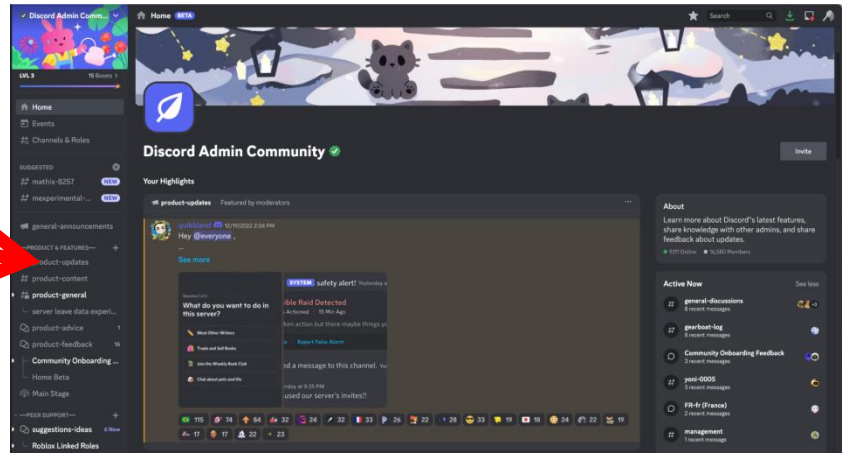


Niveau de test – Test Système





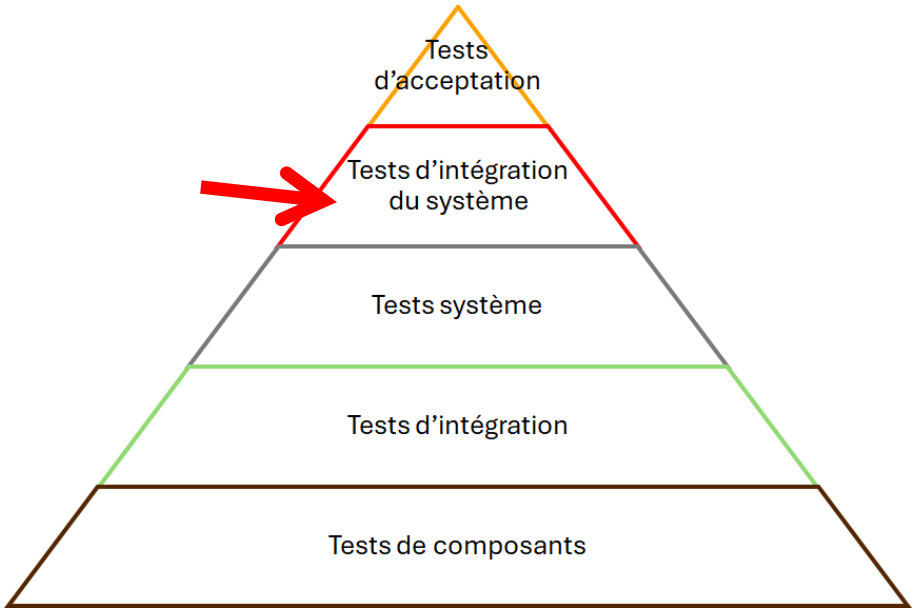




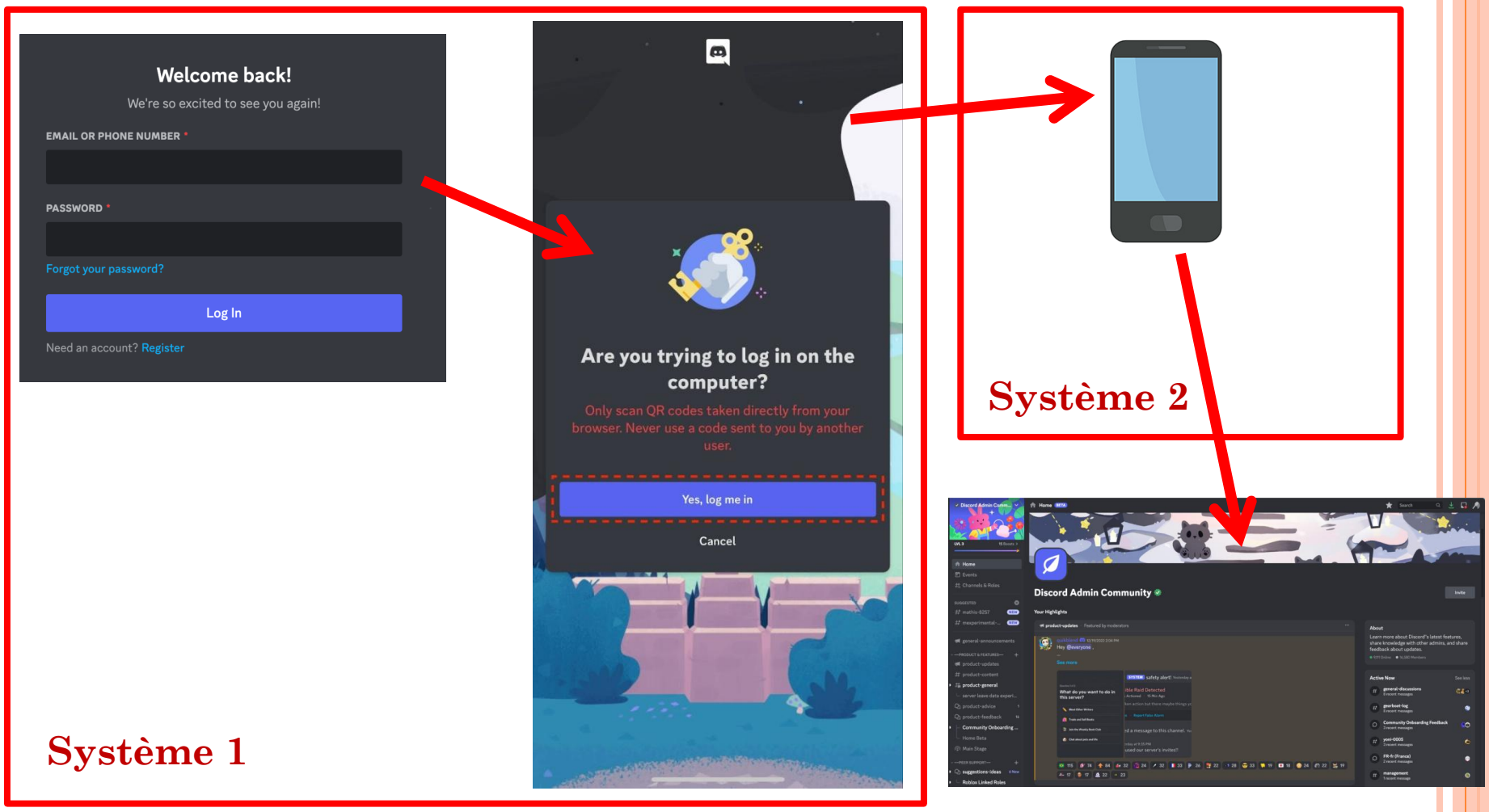
Ici on vérifie que l'authentification fonctionne bien, que les bonnes erreurs sont remontées…

Niveau de test – Test d'intégration Système

- Dans le contexte actuel il est très rare de travailler sur un service numérique totalement isolé. Un service numérique interagit avec d'autres services et il ne peut fonctionner qu'avec ces autres systèmes. L'objectif de ces tests est de vérifier que notre système est capable de fonctionner dans cet environnement non cloisonné.



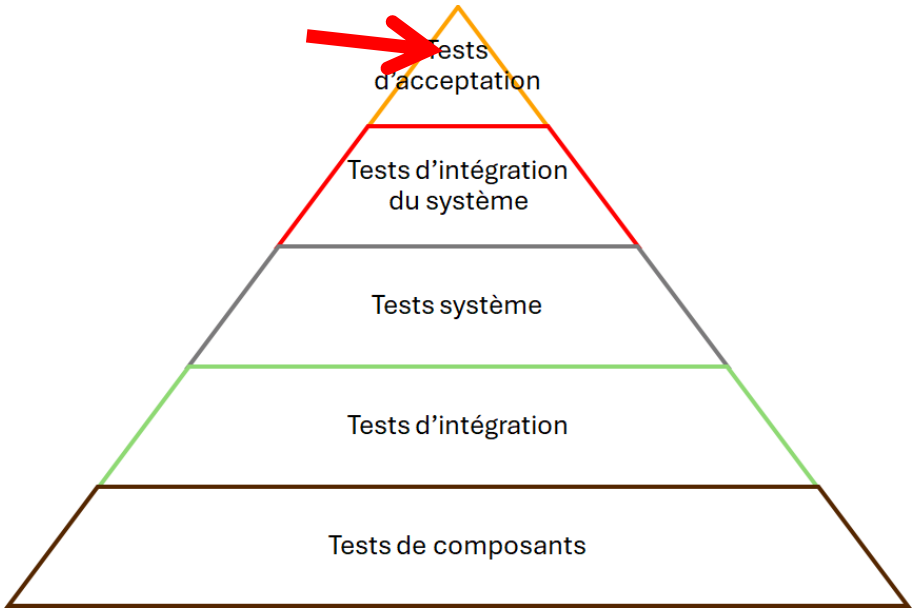
Niveau de test – Test d'intégration Système



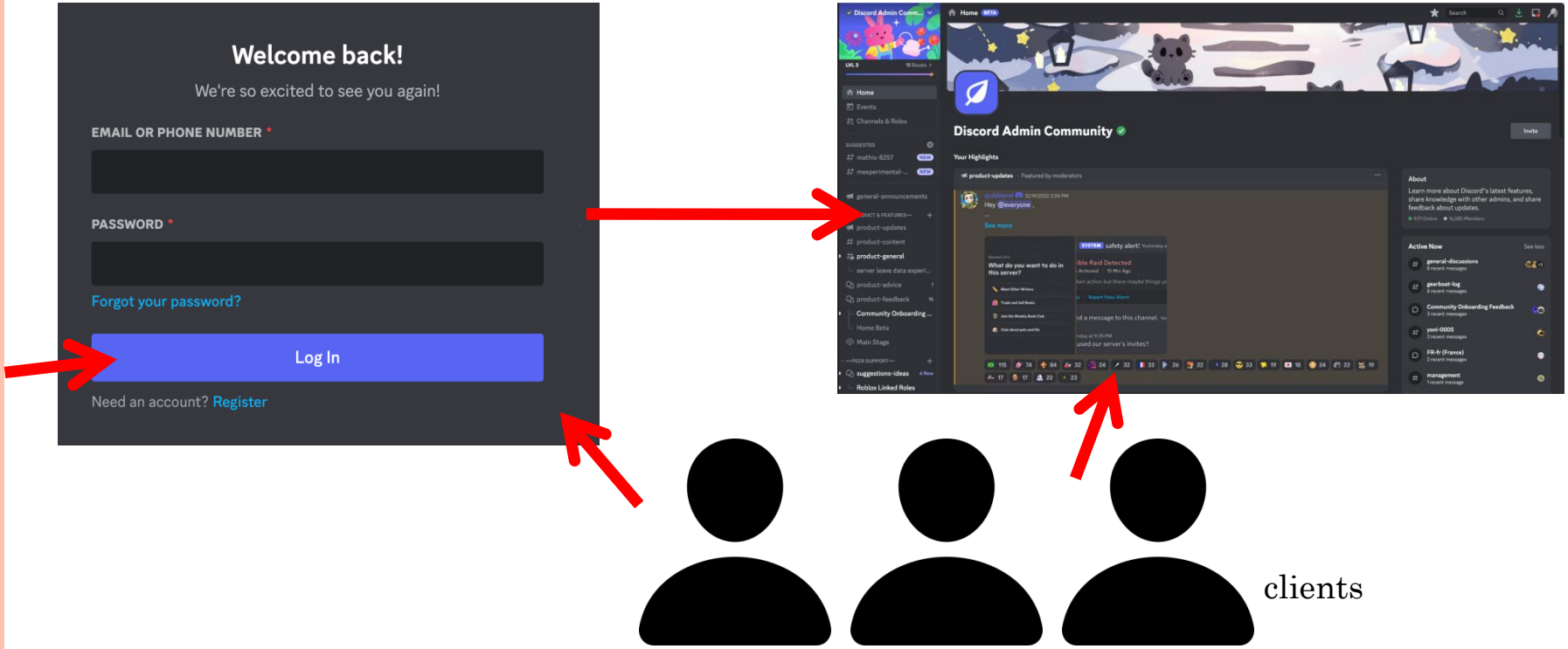
Dans l'exemple de l'authentification on peut imaginer des tests vérifiant le lien entre la demande d'authentification et la validation par la réception d'un code SMS.

Niveau de test – Test d'acceptation

- Les tests « finaux » effectués par le métier, les utilisateurs finaux ou leurs représentants (par exemple avec une bêta test). Leurs but est de confirmer que le produit final correspond bien aux besoins des utilisateurs finaux.
- Attention : ce n'est pas parce qu'une application répond aux spécifications qu'elle répond aux besoins des utilisateurs.
- Les tests d'acceptation sont des tests manuels.



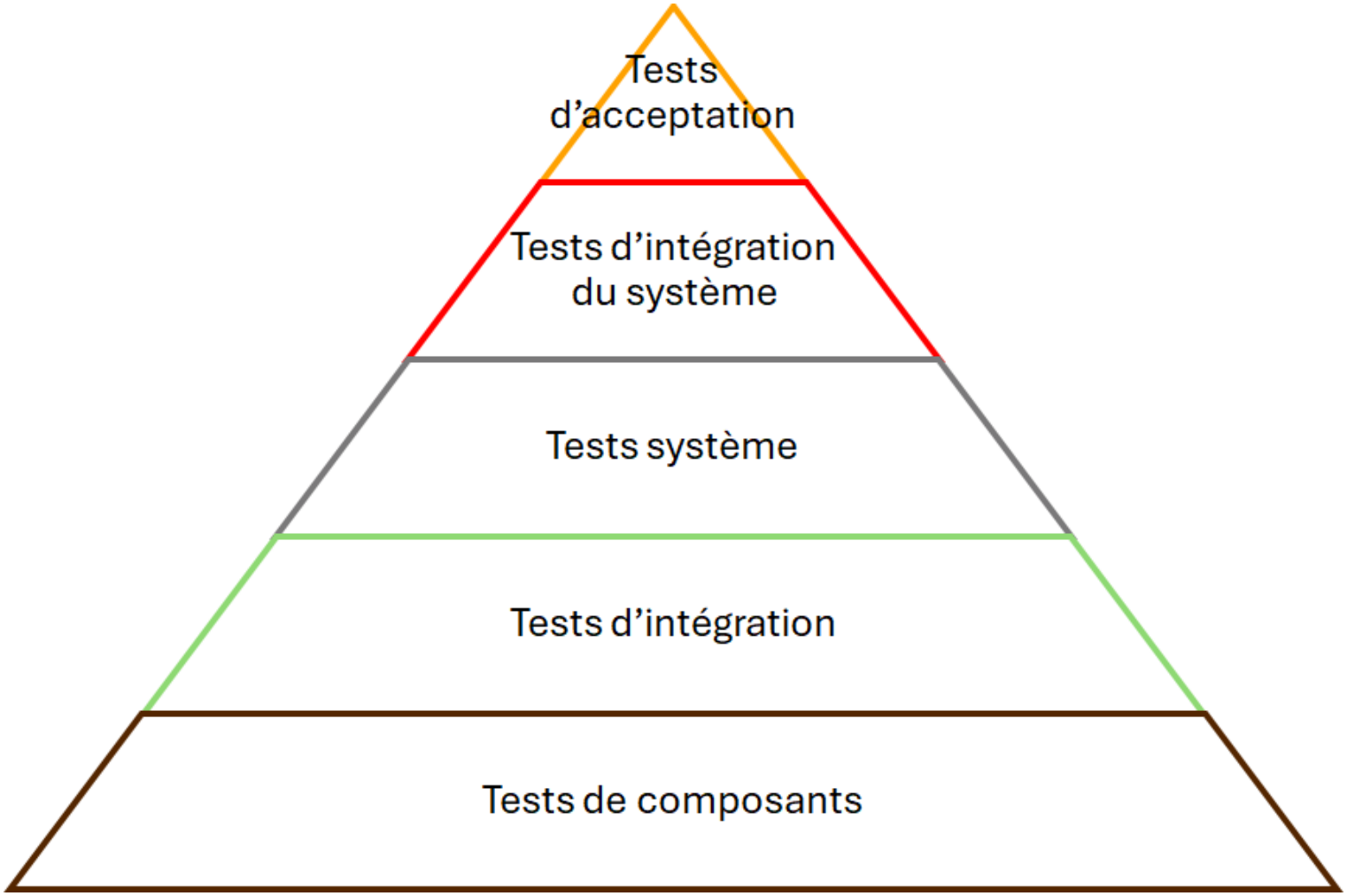
Niveau de test – Test d'acceptation



Avec ces tests on vérifie qu'en plus de répondre aux exigences l'authentification correspond bien à ce à quoi le métier ou les clients finaux s'attendent (un champ authentification trop petit peut être problématique par exemple).

Niveau de test

- Ces niveaux de tests sont généralement représentés par une pyramide car plus on est bas dans la pyramide plus le nombre de cas de tests est censé être important.



Niveau de test –exemple

- **Le contexte:**

- Le client, un commerçant spécialisé dans la vente de souvenirs aux touristes. Dans ce cadre je souhaite étoffer mon offre et proposer des puzzles dont voici les caractéristiques:
- Il veut un puzzle représentant une pyramide car c'est le monument qui a le plus de succès auprès des touristes
- Le puzzle doit être constitué de 100 pièces
- Le puzzle doit être en couleur
- L'image doit être nette (résolution supérieure ou égale à 2048*1536)
- La longueur du puzzle doit être de 70 cm
- Le puzzle doit être en carton renforcé par du plastique pour assurer une résistance des pièces

Niveau de test –exemple

- **Les tests de composants (souvent unitaires):**

- Les pièces du puzzle sont-elles:
  - De la bonne taille
  - Avec le bon dpi (bon nombre de pixels)
  - Avec le bon matériau

Niveau de test –exemple

- **Les tests d'intégrations:**

- Chaque pièce a-t-elle sa place
- Chaque pièce s'emboite t-elle correctement avec ses voisines
- L'image est-elle bien continue entre chaque pièce?

Niveau de test –exemple

- **Les tests systèmes (vérification des spécifications):**
- L'image représente t-elle bien une pyramide ?
- Le nombre de pièces est-il le bon ?
- La taille totale du puzzle est-elle la bonne ?
- Le puzzle est-il en couleur ?

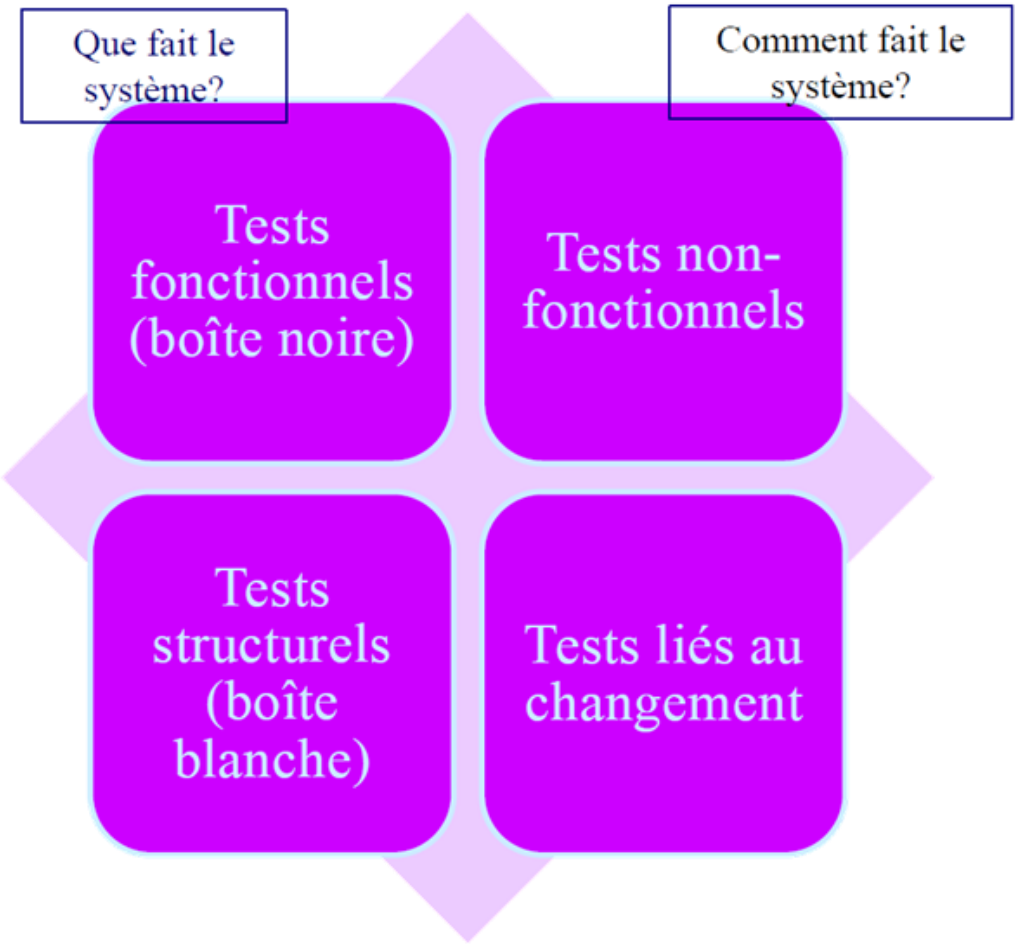
Niveau de test –exemple

- **Tests d'acceptation:**
- Le puzzle proposé ne satisfait aucunement le client. En effet, le client n'est pas localisé en Égypte mais au Mexique !
  Le puzzle n'est donc pas vendable et ne correspond pas au besoin du client.

Type de test

- Ces niveaux de tests sont généralement représentés par une pyramide car plus on est bas dans la pyramide plus le nombre de cas de tests est censé être important.

4 principaux types de test



Type de test – tests fonctionnels

Évaluer si la structure ou l'architecture du composant ou du système est correcte, complète et conforme aux spécifications

Niveaux concernés

- **Tous**

Bases de test

- Spécifications des exigences métier
- Épics et User Stories
- Cas d'utilisation ou les spécifications fonctionnelles
- Non documentées

Techniques de test

- Celles associées aux tests « Boîte noire »

Complétude (couverture)

- Couverture fonctionnelle: pourcentage du ou des types d'éléments couverts

Compétences

- Connaissance du problème métier

Type de test – tests fonctionnels - exemple

- **Exemple (MiamBot) :**
  Imaginez que vous testez la fonctionnalité "Créer une commande" dans MiamApp.

- Vous entrez les informations suivantes : un plat de "Burger Frites", une adresse de livraison, et une heure de commande.
- Vous validez.
- Ensuite, vous vérifiez que l'application :
  - Affiche le bon récapitulatif de commande.
  - Envoie les infos au Buguino pour démarrer la livraison.

- Le test fonctionnel consiste à vérifier que **toutes les étapes fonctionnent comme prévu**.

Type de test – tests non-fonctionnels

Évaluer des caractéristiques de qualité non-fonctionnelles, telles que la fiabilité, la performance, la sécurité, la compatibilité et la facilité d'utilisation

Niveaux concernés

- **Tous**

Bases de test

- Spécifications des exigences métier
- Épics et User Stories
- Cas d'utilisation ou les spécifications fonctionnelles et techniques
- Non documentées

Techniques de test

- Celles associées au type « Boîte noire ». Voire celles du type « boite blanche », par ex pour la performance.

Complétude (couverture)

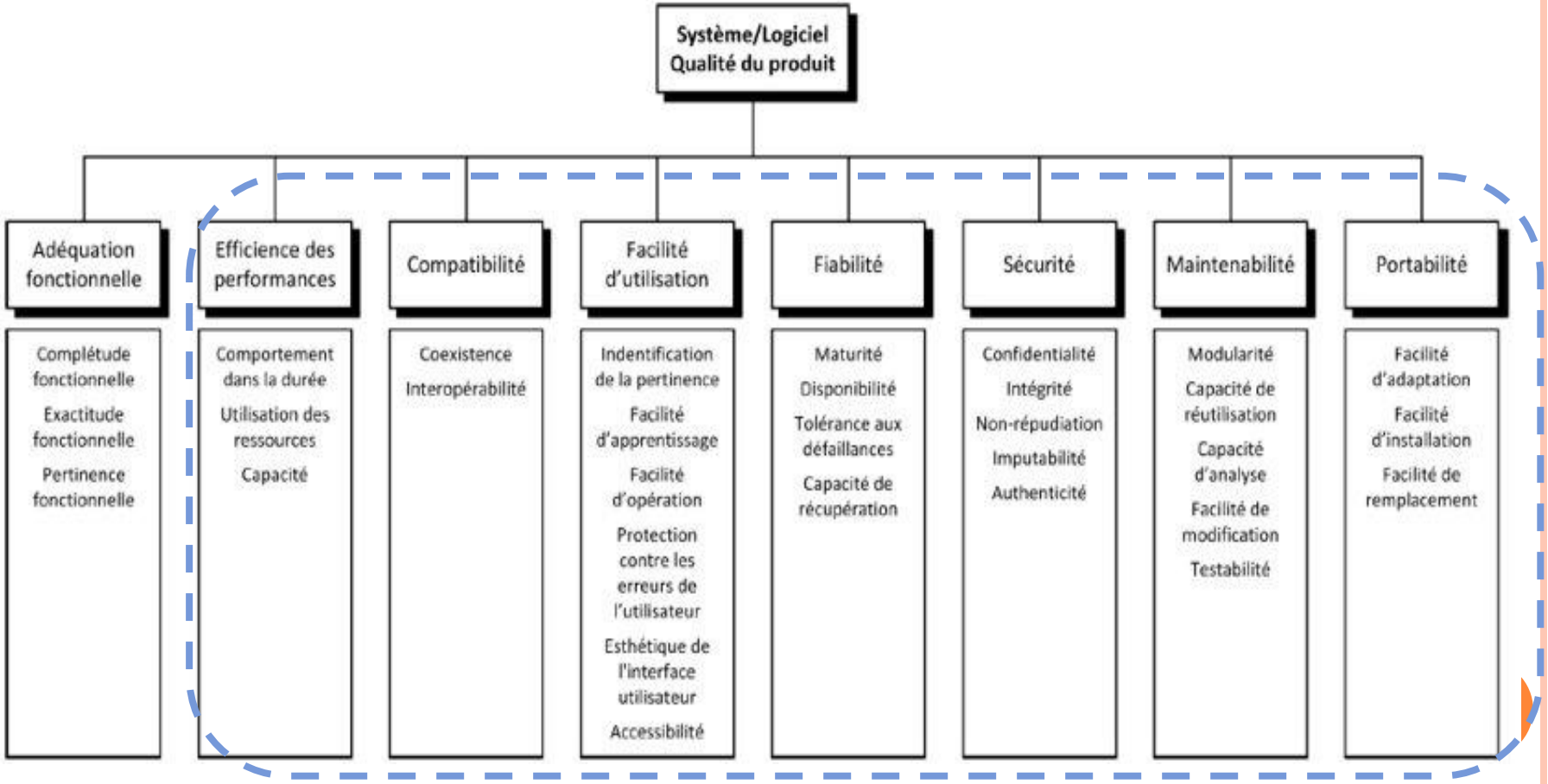
- Couverture non fonctionnelle: pourcentage de couverture par les tests d'un certain type d'élément

Compétences

- Connaissances spécifiques (technologie ou catégories d'utilisateurs par ex.)

Type de test – tests non-fonctionnels

La norme ISO (ISO/CEI 25010) présente des catégories et attributs à tester



Type de test – tests non-fonctionnels Exemple

- **Exemple (MiamBot) :**
  Imaginez que vous voulez tester la **performance** de l'application.

- Que se passe-t-il si **1000 utilisateurs commandent en même temps** ?
- Est-ce que MiamApp répond toujours en moins de 3 secondes ?
- Est-ce que les Buguinos reçoivent les commandes rapidement ou est-ce qu'ils se bloquent dans les "bouchons numériques" ?

- → Les tests non fonctionnels permettent de voir **si l'application tient le choc** et reste utilisable dans des situations extrêmes.

Type de test – boite blanche

Évaluer si la structure ou l'architecture du composant ou du système est correcte, complète et conforme aux spécifications

Niveaux concernés

- **Tous**

Bases de test

- Documentation et information sur le code, l'architecture, les flux de travail et/ou les flux de données

Techniques de test

- Couverture des instructions, des branches

Complétude (couverture)

- Couverture structurelle: pourcentage de couverture par les tests d'un certain type d'éléments structurels (instruction, décision par exemple)

Compétences

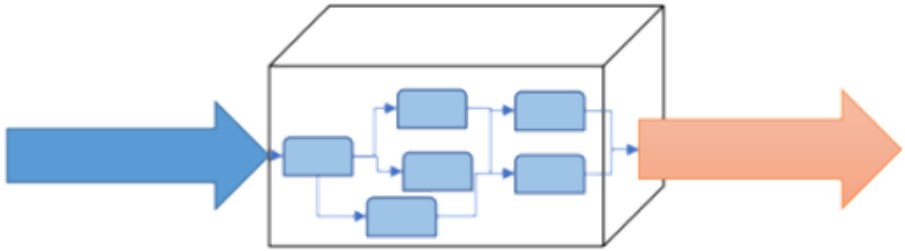
- Connaissances spécifiques technique: architecture, code

Type de test – boite blanche Exemple

Boite blanche

Visibilité :

- Entrée
- Sortie
- Structure (Code)



Référence :

- Les structures

Type de test – boite blanche Exemple

- **Exemple (MiamBot) :**
  Un développeur analyse le code qui calcule l'itinéraire des Buguinos.

- Ce code a une **boucle** qui vérifie chaque rue pour choisir le chemin le plus rapide.

- Le test structurel consiste à vérifier si toutes les possibilités de cette boucle (par exemple : rues bloquées, chemins alternatifs) sont bien couvertes et ne causent pas d'erreurs.

- → Ici, vous regardez **le "moteur" de l'application** (le code) pour vérifier qu'il fonctionne correctement.

Type de test – boite noire

Basé sur les spécifications et dérive les tests de la documentation externe à l'objet de test

Niveaux concernés

- **Tous**

Bases de test

- Spécifications : l'analyse de la documentation pour concevoir

Techniques de test

- Partitions d'équivalence, analyse des valeurs limites, tables de décisions, Tests de transition d'état

Complétude (couverture)

- Couverture des partitions, des valeurs limites, des conditions, des états, des transitions

Compétences

- Modélisation, analyse de spécifications

Type de test – boite noire Exemple

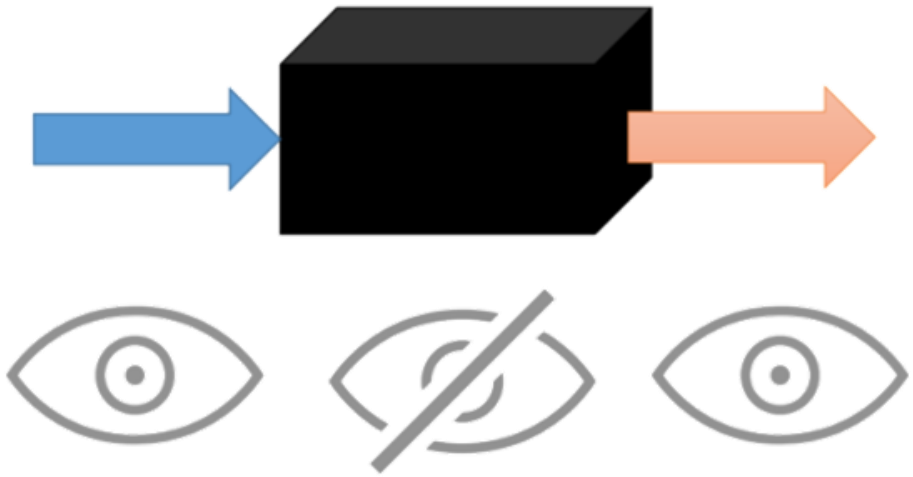
Boite noire

Visibilité :

- Entrée
- Sortie
- Structure (Code)

Référence :

- Les spécifications



Type de test – boite noire Exemple

- **Exemple (MiamBot) :**
  Vous testez la **fonctionnalité d'annulation d'une commande**.

- Vous passez une commande pour une pizza à "15 Rue des Croissants".
- Vous appuyez sur "Annuler la commande".

- Vous vérifiez :
  - Que l'application affiche "Commande annulée avec succès".
  - Que le Buguino ne démarre pas.
  - Que votre paiement est annulé.

- → Ici, vous ne vous occupez pas du code interne ou de l'algorithme. Vous vérifiez uniquement si le système fait **ce qui est attendu en fonction des entrées fournies.**

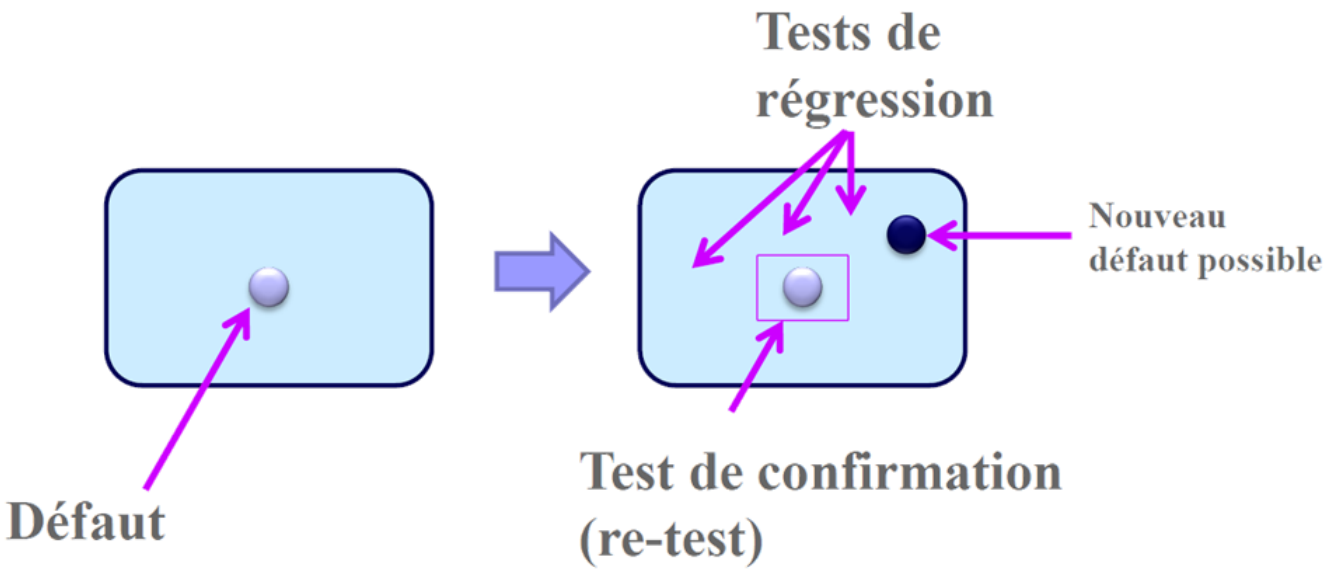
TYPE DE TEST – TEST DE NON RÉGRESSION

- Quand un défaut est détecté et corrigé, le logiciel doit être retesté pour confirmer que le défaut original a été correctement ôté : tests de confirmation (re-test)

- Après que des modifications du programme ont eu lieu, pour identifier tout nouveau défaut dû à ce(s) changement(s) : tests de (non) régression

  – Les tests de régression sont exécutés lors d'une modification du logiciel de son environnement

  – Une analyse de risque doit être effectuée pour déterminer le périmètre des tests de régression.



Type de test – test de non régression exemple

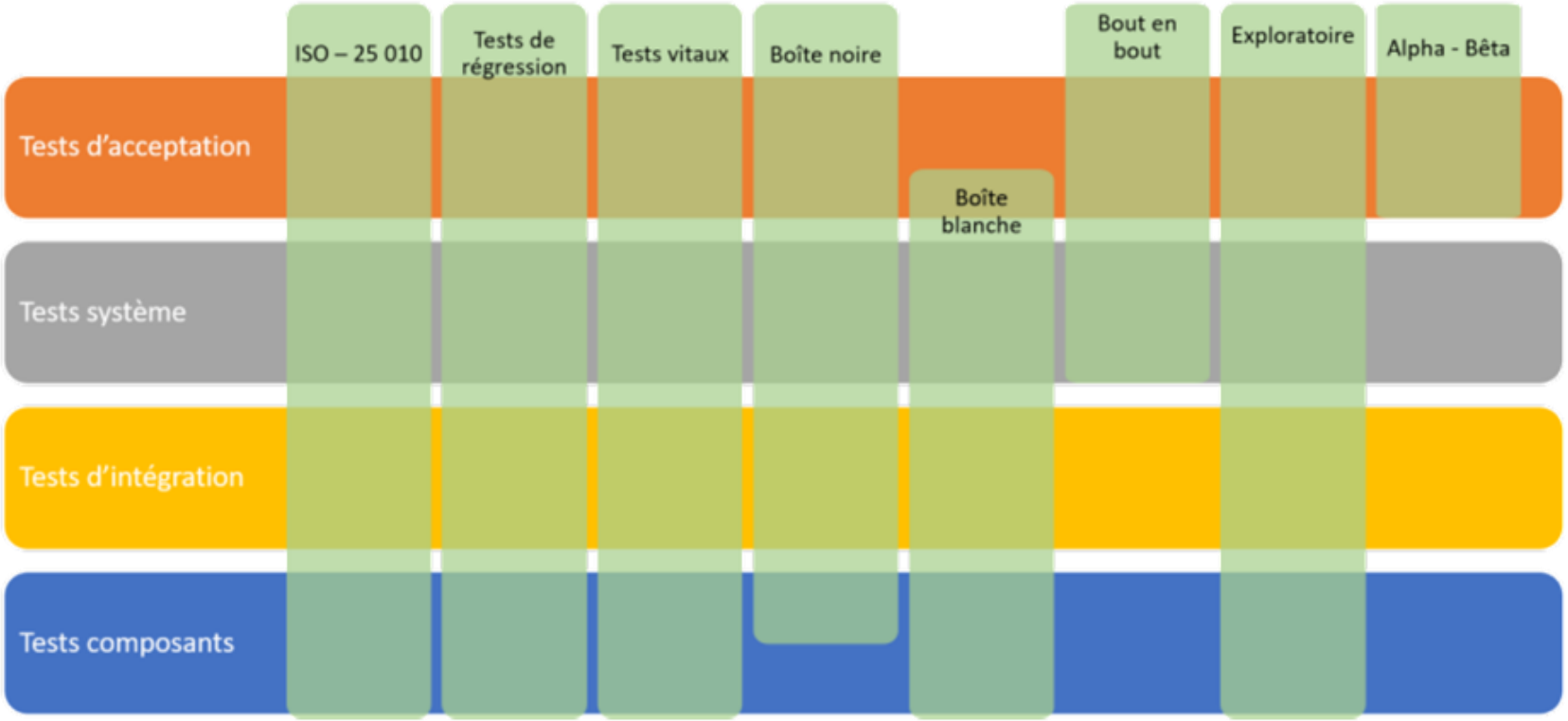
- **Exemple (MiamBot) :**
  Imaginez que vous venez de mettre à jour MiamApp pour ajouter une nouvelle fonctionnalité : la gestion des commandes "prioritaires".

- Vous devez vérifier que cette mise à jour n'a **pas cassé les fonctionnalités existantes** :
  - Est-ce que les commandes normales fonctionnent toujours ?
  - Est-ce que les Buguinos continuent de livrer correctement ?

- → Les tests de régression servent à s'assurer que tout ce qui **marchait avant marche toujours** après un changement.

Type de test

- Les quatre types de tests vus peuvent être appliqués à tous les niveaux de test



Niveau & Types de test: une vue non exhaustive d'ensemble

Type de test – Test de maintenance

Les changements rendent nécessaires les tests de maintenance :

Changements du système lui-même	• Corrections • Nouvelles fonctionnalités • Modification de paramétrage
Changements dans l'environnement	• Systèmes d'exploitation • Bases de données et migrations de données • Logiciels interfacés
Déclassement (application en fin de vie)	• Tests de la migration ou de l'archivage des données

Type de test – Test de maintenance

- Analyse d'impact :

- Principe: pour un changement envisagé, programmé ou effectué, on identifie les éléments liés directement ou indirectement, pour les mettre à jour en conséquence.

- Certains facteurs rendent difficile l'analyse d'impact:
  – Manque de documentation
  – Traçabilité manquante ou obsolète entre bases de test, conditions de test et test
  – Absence de connaissance du domaine