Test de Logiciels

Issu de:
Bilel Abed
Mme Asma Sallemi
Batata Sofiane
Mostefai Mohammed Amine

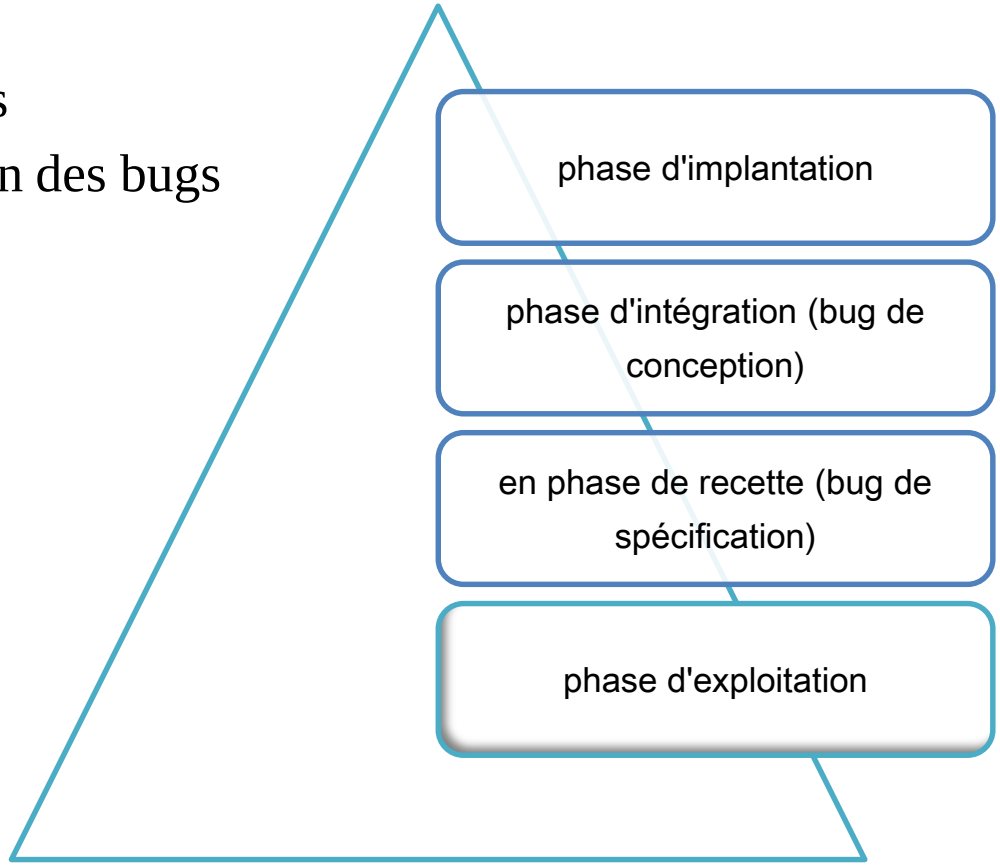
PLAN

1. Introduction
2. Types de tests
3. Tests unitaires

Pourquoi vérifier et valider?

Pour éviter les bugs

coût de la correction des bugs



Pourquoi vérifier et valider?

Pour assurer la qualité

Capacité fonctionnelle	• réponse aux besoins des utilisateurs
Facilité d'utilisation	• prise en main et robustesse
Fiabilité	• tolérance aux pannes
Performance	• temps de réponse, débit, fluidité...
Maintenabilité	• facilité à corriger ou transformer le logiciel
Portabilité	• aptitude à fonctionner dans un environnement différent de celui prévu

Objectifs des tests

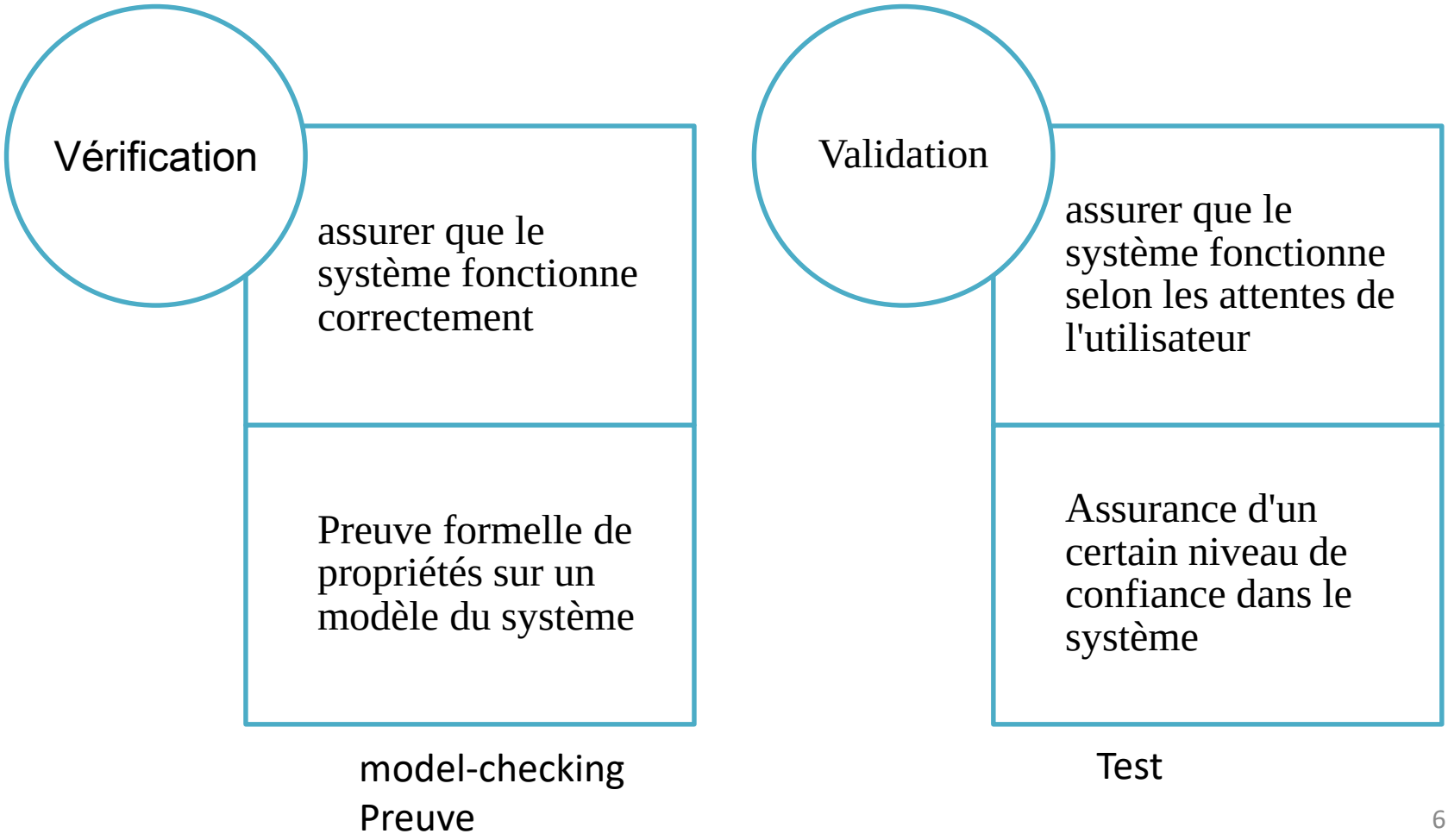
Vérification

Validation

Recensement des défaillances

Méthodes de vérification et validation

« Are we building the right product ? »



Méthodes de vérification et validation

- La vérification consiste à vérifier une spécification par rapport aux ***attentes techniques***. Par exemple, après la validation d'une facture, retrouver le champ « valide » à « true » dans la base de données
- La validation consiste à vérifier une spécification par rapport aux ***attentes métier***. Par exemple, après la validation d'une facture, la quantité du produit vendu doit décroître
- Une défaillance est une variation entre les résultats ***actuels*** et les résultats ***attendus***. L'origine de la défaillance peut être dans l'expression de besoins, la conception, l'analyse ou l'implémentation.

Comparaison des méthodes de V&V

Test :

✓ Nécessaire : exécution du système réel, découverte d'erreurs à tous les niveaux (spécification, conception, implantation)

✗ Pas suffisant : exhaustivité impossible

Preuve :

✓ Exhaustif

✗ Mise en oeuvre difficile, limitation de taille

Model-checking :

✓ Exhaustif, partiellement automatique

✗ Mise en oeuvre moyennement difficile (modèles formels, logique)

Comparaison des méthodes de V&V

Méthodes complémentaires :

✓ *Test* non exhaustif mais facile à mettre en oeuvre
✓ *Preuve* exhaustive mais très technique
✓*Model-checking* exhaustif et moins technique que la preuve

Mais :

✓Preuve et model-checking *limités par la taille du système* et vérifient des propriétés sur un *modèle du système* (distance entre le modèle et le système réel ?)
✓Test repose sur *l'exécution du système réel*, quelles que soient sa taille et sa complexité

Definitions

✓Tester un logiciel consiste à l'exécuter en ayant la totale *maîtrise des données qui lui sont fournies en entrée (jeux de test) tout en vérifiant que son comportement est celui attendu*

✓Le test est l'exécution ou l'évaluation d'un système ou d'un composant, par des moyens automatiques ou manuels, pour vérifier *qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus.* (IEEE)

✓Tester c'est exécuter le programme dans l'intention d'y *trouver des anomalies ou des défauts.* (G. Myers, The Art of Software Testing)

✓Le test est une technique de contrôle consistant à s'assurer, au moyen de son exécution, que le comportement d'un programme *est conforme à des données*

Objectifs des test

- Les tests répondent aux questions suivantes :

Est-ce que ça marche comme prévu ?

Est-ce que c'est conforme aux spécifications ?

Est-ce que ça correspond aux attentes du client ?

Est-ce que le client apprécie ?

Est-ce que c'est compatible avec les autres systèmes ?

Est-ce que ça s'adapte à un nombre important d'utilisateurs ?

Quels sont les points à améliorer ?

Est-ce que c'est prêt pour le déploiement ?

Bénéfices des tests

- Les réponses à ces questions vont servir à :

Economiser le temps et l'argent en identifiant rapidement les défaillances

Rendre le développement plus efficace

Augmenter la satisfaction du client

Correspondre le résultats aux attentes

Identifier les modifications à inclure dans les prochaines versions

Identifier les composants et les modules réutilisables

Identifier les lacunes des développeurs

Bug?

Anomalie (fonctionnement) : différence entre comportement attendu et comportement observé

Défaut (interne) : élément ou absence d'élément dans le logiciel entraînant une anomalie

Erreur (programmation, conception) : comportement du programmeur ou du concepteur conduisant à un défaut

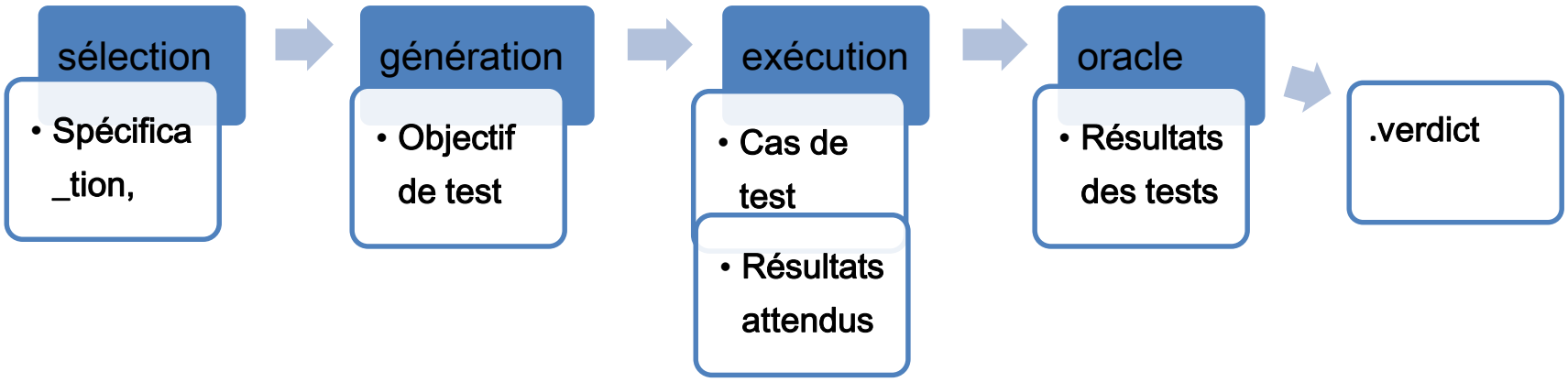
Processus de test

1.Choisir les comportements à tester (objectifs de test)

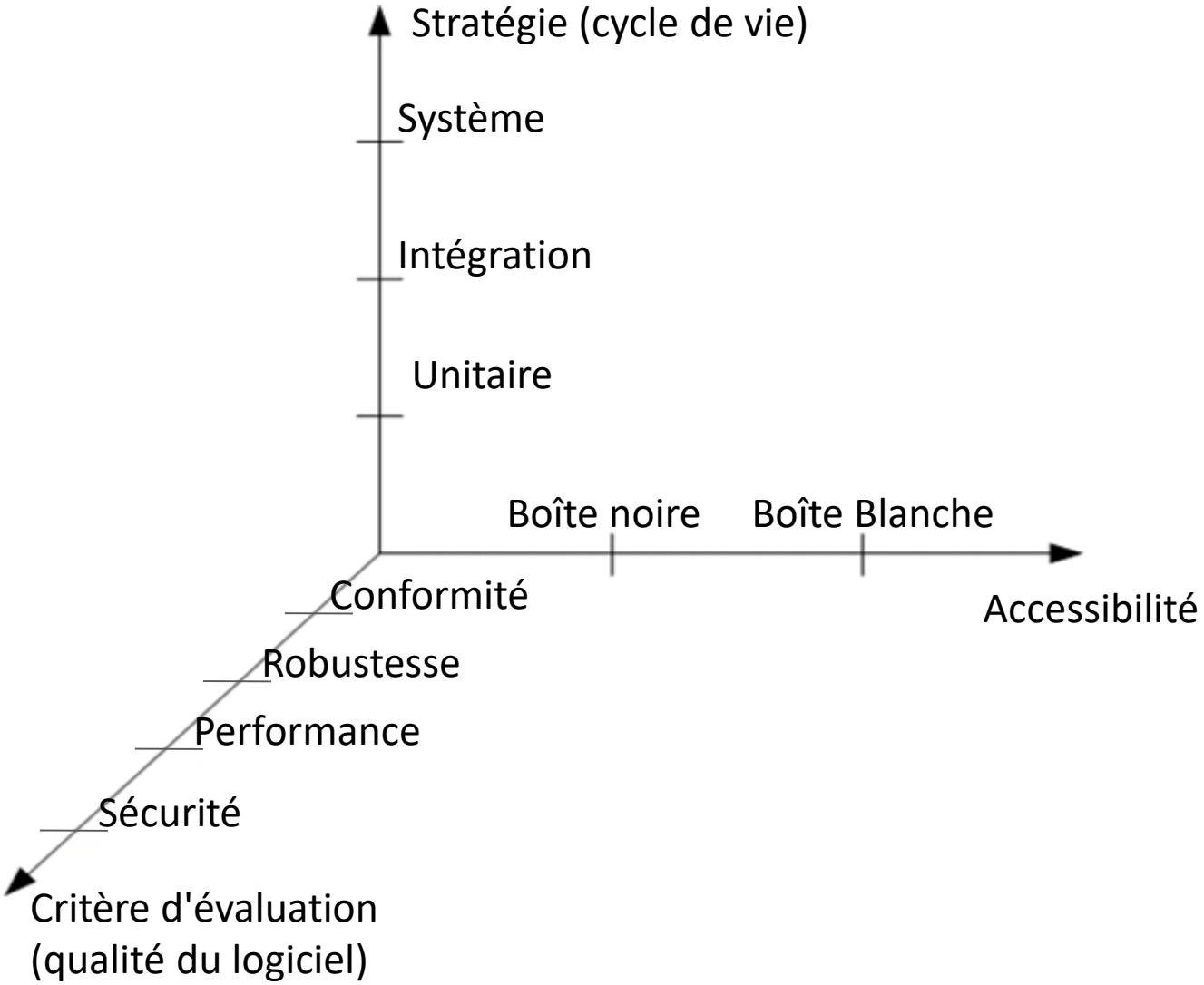
2. Choisir des données de test permettant de déclencher ces comportements & décrire le résultat attendu pour ces données

3. Exécuter les cas de test sur le système + collecter les résultats

4. Comparer les résultats obtenus aux résultats attendus pour établir un verdict

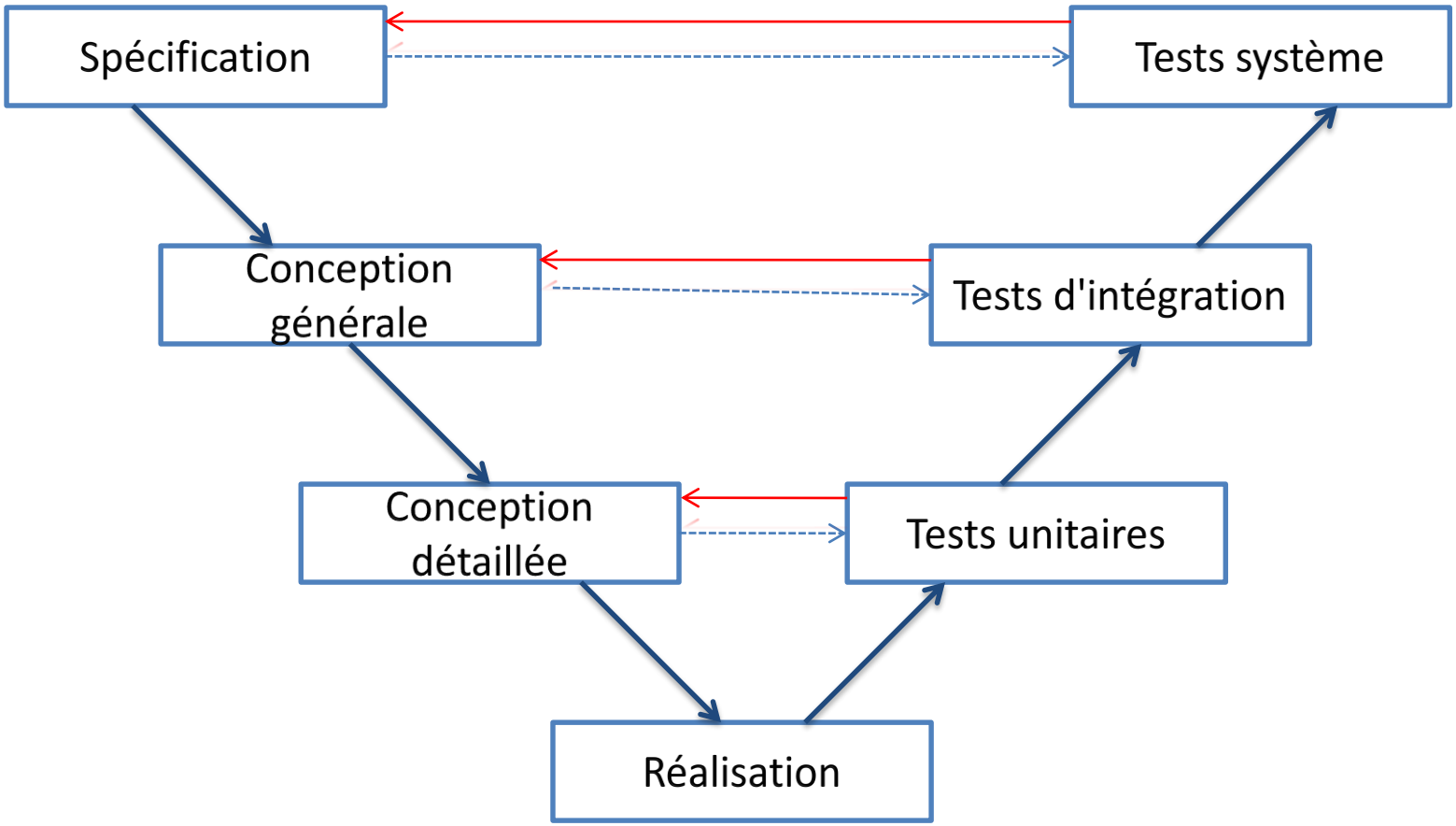


Types de tests



Cycle de vie du logiciel

Cycle en V



Etapes et hiérarchisation des tests

Test fonctionnel - boîte noire

✓Indépendant de l'implémentation

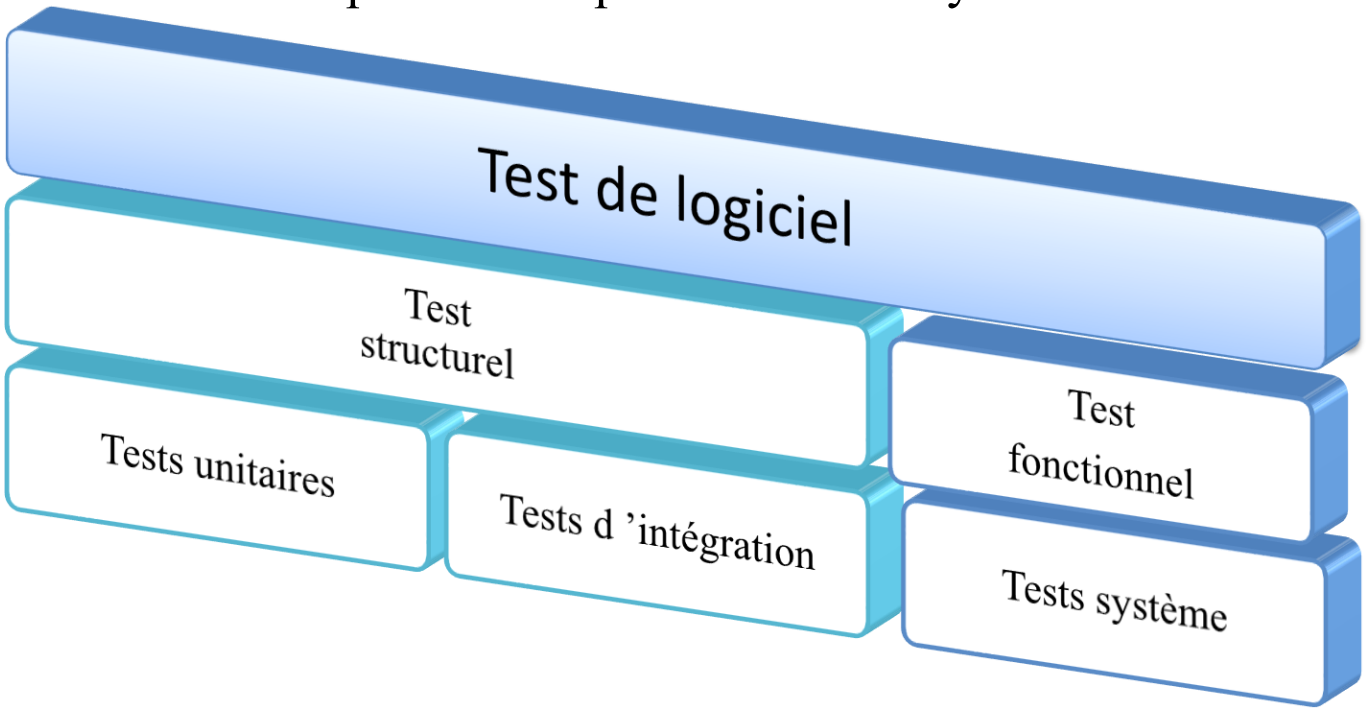
✓Planifier à partir de la spécification fonctionnelle

✓Réutilisables

Test structurel - boîte blanche

✓Dépend de l implémentation

✓Les données de test sont produites à partir d'une analyse du code source



Processus de tests

- Les tests ne sont pas le travail d'un seul homme mais celui d'une ***équipe***
- La taille de cette équipe dépend de la ***taille*** et de la ***complexité*** du projet
- Les développeurs doivent avoir un rôle dans les tests mais d'une façon ***réduite***
- Le testeur doit être minutieux, critique (pas au sens jugement), curieux doté d'une bonne communication

Equipe de testeurs

- Les testeurs doivent poser des questions que les développeurs peuvent trouver embarrassantes

Comment pouvez-vous dire que ça marche ?

Que veut dire pour vous « ça marche » ?

Pourquoi ça marchait et que ça ne marche plus ?

Qu'est-ce qui a causé le mauvais fonctionnement ?

Fonctions et rôle des testeurs

- Le ***coordinateur de tests*** crée les plans de tests et les spécifications de tests sur la base des spécifications techniques et fonctionnelles
- Les ***testeurs*** exécutent les tests et documentent les résultats

Principales catégories de tests

Tests en boîte noire

Tests en boîte blanche

Boîtes blanches / boites noires

- Les test en boîte noire s'exécutent en ***ignorant*** les mécanismes internes du produit
- Les tests en boîte blanche sont des tests qui prennent les mécanismes internes ***en considération***
- Dans les tests en boîte noire, le testeur ***n'accède pas*** au code source

Types de tests

Tests unitaires

Tests d'intégration

Tests du système et de fonctionnement

Tests d'acceptation

Tests de régression

Béta-Tests

Les tests unitaires

- Les tests unitaires sont les tests de ***blocs individuels*** (par exemple des blocs de code)
- Les tests unitaires sont généralement écrits par les ***développeurs*** eux-mêmes pour la validation de leurs classes et leurs méthodes
- Les tests unitaires sont exécutés généralement par les machines

Les tests d'intégration

- Exécuté en boîte noire ou boîte blanche
- Les tests d'intégration vérifient que les composants ***s'intègrent bien*** avec d'autres composants ou systèmes
- Les tests d'intégration vérifient aussi que le produit est ***compatible*** avec l'environnement logiciel et matériel du client

Les tests fonctionnels

- S'exécute en boîte noire
- Vérifie que le produit ***assure les fonctionnalités*** spécifiées dans les spécifications fonctionnelles

Tests systèmes: spécifications non fonctionnelles

- S'exécute en boîte noire
- S'oriente vers les ***spécifications non fonctionnelles***
- Composé de plusieurs tests :
- ***Tests de stress*** : tester le produit au-delà des attentes non fonctionnelles du client. Par exemple, un système de sauvegarde qui doit tout sauvegarder deux fois par jour, le tester en mode trois fois par jour.
- ***Tests de performance*** : évaluation par rapport à des exigences de performances données. Par exemple, un moteur de recherche doit renvoyer des résultats en moins de 30 millisecondes.

Tests système: chargement et utilisabilité

- ***Tests de chargement*** : vérifie que l'application fonctionne dans des conditions normales (contraire aux tests de stress)
- ***Tests d'utilisabilité*** : vérifier les exigences d'apprentissage demandées à un utilisateur normal pour pouvoir utiliser le produit

Tests d'acceptation

- S'exécute en boîte noire
- Les tests d'acceptation sont des tests formalisés par le ***client*** qui sont exigés par le client pour qu'il accepte le produit

Tests de (non) régression

- Les tests de régression sont un ***sous-ensemble*** des tests originaux
- Les tests de régression vérifient qu'une modification n'a pas eu ***d'effets de bord*** sur le produit
- Les tests de régression sont utiles parce que la ré-exécution de tous les tests est *très coûteuse*

Les bêta tests

- Quand un produit est ***quasiment terminé***, il est distribué gratuitement à certains de ses utilisateurs
- Ces utilisateurs sont appelés ***testeurs béta***
- Ces utilisateurs vont utiliser le produit et reporter tout éventuel disfonctionnement de celui-ci

Comparaison des tests

Test	Portée	Catégorie	Exécutant
Unitaires	Petites portions du code source	Boîte blanche	Développeur Machine
Intégration	Classes / Composants	Blanche / noire	Développeur
Fonctionnel	Produit	Boîte noire	Testeur
Système	Produit / Environnement simulé	Boîte noire	Testeur
Acceptation	Produit / Environnement réel	Boîte noire	Client
Beta	Produit / Environnement réel	Boîte noire	Client
Régression	N'importe lequel	Blanche / noire	N'importe

Le plan de tests

- La planification des tests doit se faire dans les ***premières phases*** du projet
- Le plan de tests est un document ***obligatoire*** déterminant le ***déroulement*** des tests durant le projet
- Le tableau suivant indique le contenu d'un plan de test selon le standard « American National Standards Institute and Institute for Electrical and Electronic Engineers Standard 829/1983 for Software Test Documentation"

Le plan de tests

Élément	Description	Objectif
Responsabilités	Acteurs et affectation	Décrit qui fait quoi dans les tests. Assure le suivi et les affectations
Tests	Portée de tests, plannings, durées et priorités	Définit le processus et détaille les actions à entreprendre
Communication	Plan de communication	Tout le monde doit savoir ce qu'il doit savoir avant les tests et ce qu'il doit faire savoir après les tests
Analyse de risques	Éléments critiques à tester	Identification des domaines qui sont critiques dans le projet
Traçage des défaillance	Documentation des défaillances	Documenter les défaillances et les détails les concernant

Priorisation des défaillances

- Le plan de tests détermine la ***priorisation*** des défaillances
- La priorisation facilite la communication entre développeurs et testeurs ainsi que l'affectation et la planification des tâches

Quantification des défaillances

- Par exemple, une priorisation de 1 à 6 est souvent adoptée

Priorité	Description
1 – Fatal	Impossible de continuer les tests à cause de la sévérité des défaillances
2- Critique	Les tests peuvent continuer mais l'application ne peut passer en mode production
3- Majeur	L'application peut passer en mode production mais des exigences fonctionnelles importantes ne sont pas satisfaites
4- Medium	L'application peut passer en mode production mais des exigences fonctionnelles sans très grand impact ne sont pas satisfaites
5- Mineur	L'application peut passer en mode production, la défaillance doit être corrigée mais elle est sans impact sur les exigences métier
6- Cosmétique	Défaillances mineures relatives à l'interface (couleurs, police, …) mais n'ayant pas une relation avec les exigences du client

Tests en boite noire

- Les tests en boîte noire détermine si un produit fait ce qu'on attend de lui ***indépendamment*** de sa structure interne
- Plusieurs tests sont effectués en boîte noire tels que les tests fonctionnels et les tests d'acceptation
- Les tests en boîte noire déterminent les fonctions manquantes ou mal implémentées, les erreurs d'interface, des anomalies de performance ou de comportement, les bugs et les crashs
- Quels que soit la consistance ou le volume des tests, ***aucune garantie*** n'est donnée sur l'absence d'erreurs

Organisation tests en boite noire

- Il vaut mieux que les testeurs soient des personnes différentes des programmeurs
- Les testeurs s'intéressent à une fonction en terme d'entrée et de sorties

Cas de tests

- Un cas de test est un ensemble ***d'entrées de tests***, de ***conditions d'exécution*** et de ***résultats attendus*** pour un objectif particulier tel que la conformité du programme avec une spécification données

Exemple de tests en boite noire

ID
Description
Priorité
Préconditions
Scénario
Résultats attendus
Résultats actuels
Remarques

Anatomie d'un cas de test

- L'ID est un ***numéro unique*** qui permet la traçabilité des cas de test, les lier à des spécifications ou à d'autres cas de test
- Les préconditions déterminent les ***conditions nécessaires*** à un cas de test. Elles indiquent aussi les ***cas de test qui doivent être exécutés précédemment***
- La description est un texte ***décrivant*** le tests et ses attentes
- Le scénario détermine les ***étapes détaillées*** à suivre par le testeur
- Les résultats attendus sont ***attendus*** de l'exécution du scénario
- Les résultats actuels sont les ***vrais résultats*** obtenus, s'ils sont différents des résultats attendus, une défaillance est signalée
- Les remarques sont signalées par le testeur pour ajouter des information concernant une exécution donnée

Suite de l'anatomie

- Le scénario est composé de plusieurs ***actions numérotées***
- Chaque action ***peut*** avoir un ***résultat attendu***
- L'exécution de test affecte un ***résultat actuel*** à chaque étape attendant un résultat
- Si chaque résultat actuel est ***conforme*** au résultat attendu alors le test ***réussit*** sinon le test ***échoue***

Exemple de cas de tests

ID	Description	Préconditions	Scénario	Résultats attendus	Résultats actuels	Remarques
1	Ouverture d'un document	Existence d'un document	1- Cliquez sur le bouton « Ouvrir » 2- Sélectionnez le fichier désiré 3- Appuyez sur OK	2- Une boîte d'ouverture de fichiers s'affiche 3- Le Document est ouvert et prêt à être manipulé		
2	Impression d'un document	TC 1	1- Cliquez sur le bouton imprimer 2- Sélection la config puis cliquer sur « Imprimer »	1- La boîte de configuration d'imprimante s'affiche 2- Le document sort sous format papier		

Campagne de tests

- Une campagne de tests sélectionne un certain nombre de cas de tests sémantiquement relatifs et les exécute
- Chaque campagne de tests a un objectif tracé : validation du produit, performance, … etc.

Définition d'un bon plan de tests

- Un bon plan de tests ***minimise*** les cas de tests et ***maximise*** la probabilité de détection des défaillances

Les tests unitaires

- Les tests unitaires sont des tests en boîte blanche
- Les tests unitaires sont composé d'un ensemble de classes appelées « ***classes de test*** »
- Ces classes valident que des portions de code répondent à un certain besoin
- Les tests unitaires sont importants car ils permettent de détecter ***le maximum de défaillance*** avant les tests en boîte noire et qu'ils peuvent s'exécuter d'une manière automatique

Objectifs des tests unitaires

- Les tests unitaires ont deux objectifs : « ***couverture de code*** » et « ***couverture de données*** »
- La couverture du code stipule de tester chaque ligne de code écrite (appel de fonction, boucles, décisions, décisions multiples,…)
- La couverture des données oriente les tests vers les données (données valides, données invalides, accès aux éléments en dehors de la capacité d'un tableau, peu de données, trop de données,…etc)

Caractéristiques des tests unitaires

- Les tests unitaires doivent être conforme à l'acronyme ***FIRST***:
- ***FAST*** : un test unitaire doit s'exécuter rapidement
- ***INDEPENDANT***: chaque test doit être indépendant des autres
- ***REPEATABLE***: chaque test peut être répété autant de fois que voulu
- ***SELF-VALIDATING***: Un test se valide lui-même par un succès ou par un échec
- ***TIMELY***: On écrit les tests quand on en a besoin

Exemple de test unitaire

```
public class Calculator
{
public int add(int x, int y)
{
        return x + y;
}

public int multi(int x, int y)
{
        return x*y;
}
}
```

Suite exemple: test classe Calculator

```
// tester la classe
public class CalculatorTest
{
// tester l'addition
public int addTest()
{
        // valeurs en entrée
        int a = 15;
        int b = 18;
        Calculator calc = new Calculator();
        int c = calc.Add(a,b);
        // résultat attendu et résultat actuel
        assertEquals(33, c);
}

}
```

Test de conformité

But : Assurer que le système présente les fonctionnalités attendues par l'utilisateur

Méthode : Sélection des tests à partir de la spécification, de façon à contrôler que toutes les fonctionnalités spécifiées sont implantées selon leurs spécifications

Exemple: Service de paiement en ligne
✓Scénarios avec transaction acceptée/refusée, couverture des différents cas et cas d'erreur prévus

Test de robustesse

But : Assurer que le système supporte les utilisations imprévues

Méthode : Sélection des tests en dehors des comportements spécifiés (entrées hors domaine, utilisation incorrecte de l'interface, environnement dégradé...)

Exemple: Service de paiement en ligne
✓Login dépassant la taille du buffer
✓ Coupure réseau pendant la transaction

Test de sécurité

But : Assurer que le système ne possède pas de vulnérabilités permettant une attaque de l'extérieur

Méthode : Simulation d'attaques pour découvrir les faiblesses du système qui permettraient de porter atteinte à son intégrité

Exemple: Service de paiement en ligne
✓Essayer d'utiliser les données d'un autre utilisateur
✓Faire passer la transaction pour terminée sans avoir payé

Test de performance

But : Assurer que le système garde des temps de réponse satisfaisants à différents niveaux de charge

Méthode : Simulation à différents niveaux de charge d'utilisateurs pour mesurer les temps de réponse du système, l'utilisation des ressources...

Exemple: Service de paiement en ligne

✓ Lancer plusieurs centaines puis milliers de transactions en même temps

Test de non régression
Un type de test transversal

But : Assurer que les corrections et les évolutions du code n'ont pas introduit de nouveaux défauts

Méthode : À chaque ajout ou modification de fonctionnalité, rejouer les tests pour cette fonctionnalité, puis pour celles qui en dépendent, puis les tests des niveaux supérieurs

✓Lourd mais indispensable
✓Automatisable en grande partie

Techniques de « test statique »

✓*Définition* : ne requiert pas l ’exécution du logiciel sous test sur des données réelles

✓*Efficacité* : plus de 50 % de l ’ensemble des fautes d ’un projet sont détectées lors des inspections si il y en a (en moyenne plus de 75%)

✓*Défaut* : mise en place lourde, nécessité de lien transversaux entre équipes, risques de tension…tâche plutôt fastidieuse

✓*Règles*

➢être méthodique

➢un critère : le programme peut-il être repris par quelqu’un qui ne l’a pas fait(R1)

➢un second critère : les algorithmes/l’architecture de contrôle apparaît-elle clairement ?(R2)

➔décortiquer chaque algorithme et noter toute redondance curieuse et toute discontinuité

Test d'intégration

✓*But* : vérifier les interactions entre composants (se base sur l'architecture de conception

✓*Difficultés principales de l'intégration*

- interfaces floues
- implantation non conforme au modèle architectural (dépendances entre composants non spécifiées)
- réutilisation de composants (risque d'utilisation hors domaine)

Test d'intégration

✓*Stratégies possibles (si architecture sans cycles)*

- big-bang : tout est testé ensemble. Peu recommandé !
- top-down : de haut en bas. Peu courant. Ecriture uniquement de drivers de tests.
- bottom-up : la plus classique. On intègre depuis les feuilles.

Conclusion

Facteurs de bonne testabilité :

✓Précision, complétude, traçabilité des documents

✓Architecture simple et modulaire

✓Politique de traitements des erreurs clairement définie

Facteurs de mauvaise testabilité :

✓Fortes contraintes d'efficacité (espace mémoire, temps)

✓Architecture mal définie

Difficultés du test :

✓Indécidabilité : une propriété indécidable est une propriété qu'on ne pourra jamais prouver dans le cas général

✓Explosion combinatoire : Le test n'examine qu'un nombre fini (ou très petit) d'exécutions

➔*Impossibilité d'une automatisation complète satisfaisante*