Test Logiciel : Application aux systèmes embarqués

Animé par Paul-Ernest Martin.

Plan

- Idées essentielles
- Tester à chaque niveau de cycle
- Tester efficacement
- Les spécifications et les jeux de tests
- Les détails d'implémentation dans les tests
- Processus et test d'intégration
- Gérer les tests
- Outil pour les tests
- Génération automatique de jeux de test
- Tester des systèmes interactifs
- Tests, nouvelle mesure de complexité

Les idées essentielles

- logiciel sensible : 90% budget = test. DO 178
- Voiture -> ISO 26262
- Exigent analyse statique et preuve par couverture de tests

Avant propos :

- logiciels informatiques = essentiels
- != de conséquences pour les défaillances.
- Société numérique == matière première est logiciel

Logiciel doit faire ce qu'il doit faire et c'est tout.

Suffit pas de tester le produit final. Erreur de bout de chaîne.

Test en amont lors de l'assemblage.

Test cherche erreur

- fonctionnelle : logiciel fait ce qui doit faire.
- non fonctionnelle : temps acceptables, montée en charge

Approche statistique car non exhaustif

Les idées essentielles

Exemple pratique :

1962 -> Marinier 1, Nasa : 4 minutes de vol avant explosion.

Une virgule à la place d'un point !

Navire de guerre coulé par Exocet Français, les missiles français pas répertoriés comme ennemis !

F16 sur le dos au passage de l'équateur. (signe de la latitude !)

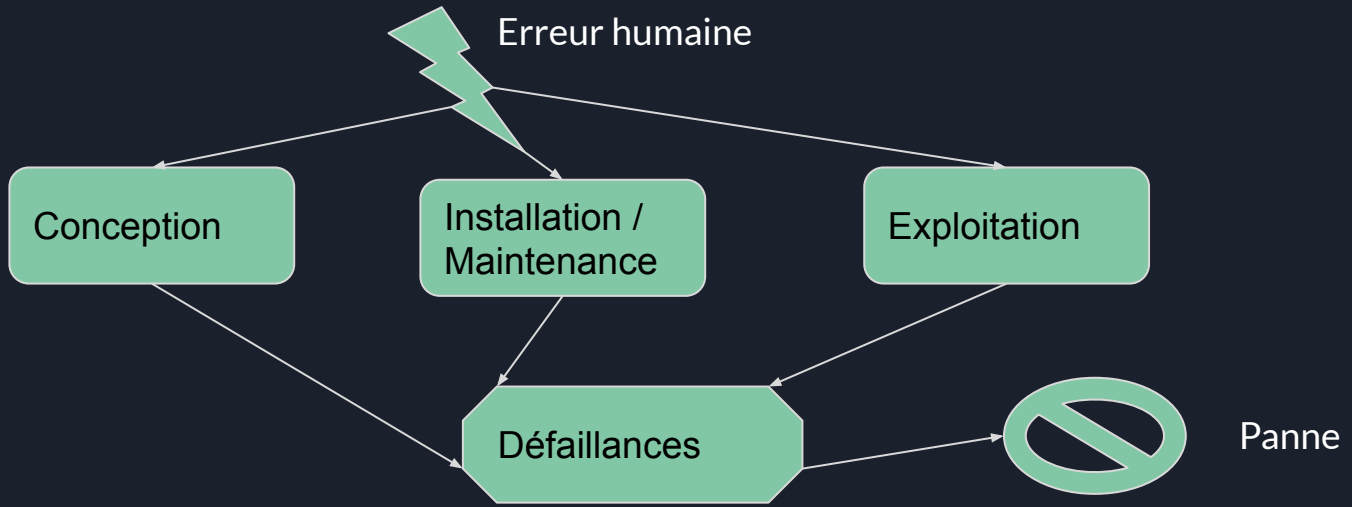
Sonde Mars Climate Orbiter perdu après 9 mois : confusion entre pied et mètre.

Les idées essentielles : Chaîne de l'erreur

Tests => démontrent défaillances logiciels.

Défaillance = calculs erronés, fenêtre mal placée, message non affiché ou service non rendu (donnée non stockée)

Panne = arrêt total ou partiel du logiciel. Nécessitant généralement un redémarrage complet du logiciel.



Les idées essentielles : Rôle des tests

Erreurs inévitables :

- construire simplement
- construire complexe pour résoudre des problèmes ambitieux

Possède des défauts mais ne doivent pas conduire à une défaillance.

Avant l'entrée en service.

Une fois déployé. Phase de maintenance

Ratio défauts corrigés / défauts engendrés > 1 alors le logiciel ne convergera pas vers une version stable.

Tester beaucoup n'est pas un gage de qualité.

Les idées essentielles : Bilan 1

1. Pourquoi est-il important d'allouer un budget important pour les tests de logiciels ?

2. Quelles sont les normes de sécurité spécifiques pour les logiciels utilisés dans les avions et les hélicoptères ?

3. Quelles sont les normes de sécurité spécifiques pour les logiciels utilisés dans les voitures ?

4. Qu'est-ce que l'analyse statique dans le contexte des tests de logiciels ?

5. Qu'est-ce que la preuve par couverture de tests ?

6. Pourquoi ne suffit-il pas de tester le produit final d'un logiciel ?

Les idées essentielles : Bilan 2

1. Quelles sont les deux types d'erreurs dans les logiciels ?

2. Qu'est-ce qu'une erreur fonctionnelle dans le contexte des logiciels ?

3. Qu'est-ce qu'une erreur non fonctionnelle dans le contexte des logiciels ?

4. Pourquoi l'approche statistique est-elle utilisée pour tester les logiciels ?

5. Quels sont les avantages et les limites de la construction simple de logiciels ?

Les idées essentielles : Bilan 3

1. Quels sont les avantages et les limites de la construction complexe de logiciels ?

2. Pourquoi est-il important de corriger plus de défauts que d'en engendrer pour que le logiciel converge vers une version stable ?

3. Pourquoi tester beaucoup de logiciels n'est pas un gage de qualité ?

4. Quelle est la différence entre une panne et une défaillance dans un logiciel ?

5. Comment définir les chaînes de l'erreur ?

Les idées essentielles : Les 7 principes

Principe	Explication
Tests montrent la présence de défauts	Ne peut pas prouver l'absence d'erreur !
Tests exhaustifs impossible	Addition 2 entiers sur 32bits : $2^{32} * 2^{32}$ -> 132 ans
Tester tôt	Le coût de la correction des erreurs augmente avec le temps. x10 entre conception et déploiement.
Regroupement des défauts	80 % des défauts sont dans 20% du logiciel

Les idées essentielles : Les 7 principes

Principe	Explication
Le paradoxe des pesticides	A force d’appliquer le même produit, il devient inefficace.
Les tests dépendent du contexte	L'utilisation du logiciel influe sur les types de tests.
Illusion de l’absence des défauts	

Les idées essentielles : Les 7 principes : Petit bilan

Question 1 : Pourquoi est-il impossible de prouver l'absence d'erreurs dans les logiciels ?

Question 2: Pourquoi les tests exhaustifs sont-ils impossibles pour certains types de logiciels ?

Question 3: Pourquoi est-il important de tester tôt dans le processus de développement de logiciels ?

Question 4: Qu'est-ce que le regroupement des défauts et comment cela peut-il aider les développeurs de logiciels ?

Question 5: Qu’est ce que le paradoxe des pesticides ?

Question 6: Pourquoi les tests de logiciels dépendent-ils du contexte ?

Question 7: En quoi l'illusion de l'absence de défauts peut-elle être dangereuse dans le développement de logiciels ?

Question 8: Quelles sont les limites des tests de logiciels ?

Les idées essentielles : Processus et psychologie

- Planifier les tests : efforts, logistiques
- Spécifier les tests : qu'elle défaut mettre en avant.
- Concevoir les tests : définir scénarios test et environnements.
- Établir les conditions de tests : paramètre en entré des tests
- Définir condition d'arrêt des tests
- Contrôler les résultats

Conseils :

- pessimistes
- oeil critique
- faire attention aux détails

Les idées essentielles : Tester à chaque cycle de vie

- Différents modèles de développement = méthode de production de logiciel.

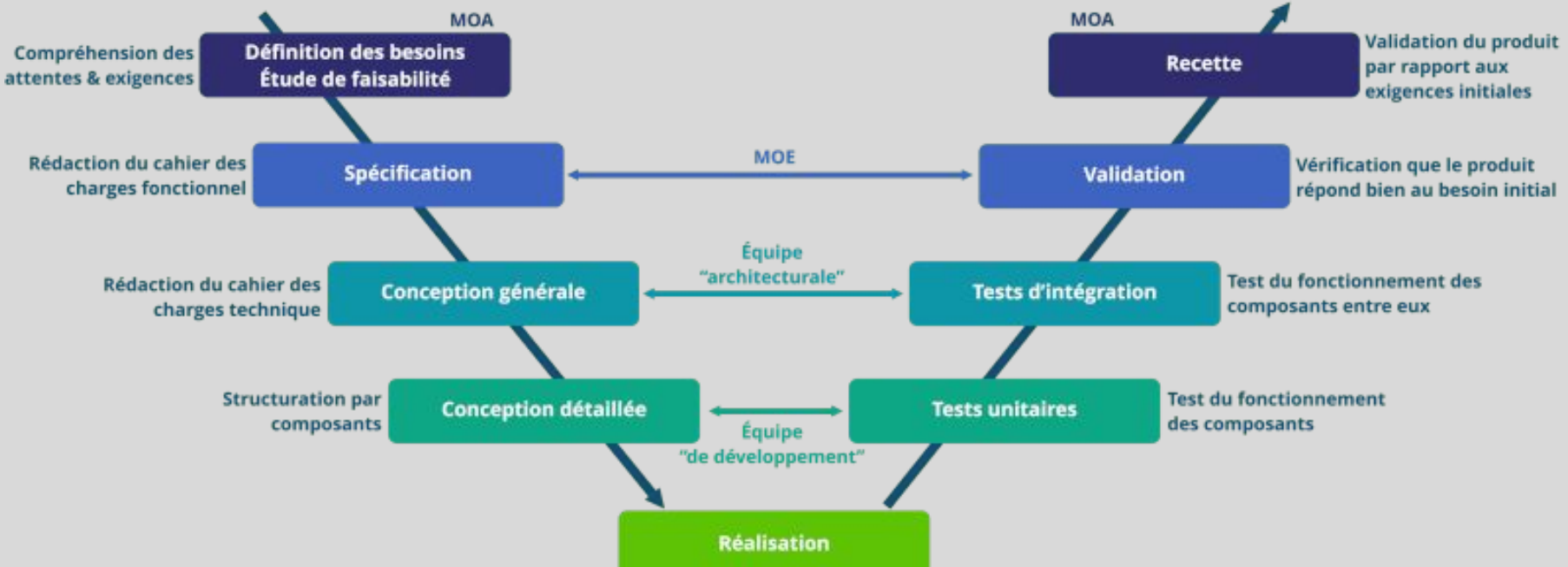
Cycle en V :

Conception.
Codage.

Conception : expression abstraite du logiciel à une réalisation concrète.
L'application du point de vue client aux mécanismes pour satisfaire les attentes et ainsi les besoins.

Codage : Implémentation, progressivement testé et mis en service

Les idées essentielles : Tester à chaque cycle de vie



Les idées essentielles : Tester à chaque cycle de vie

Préparation des tests pendant les premières étapes :

- Expression du besoin
- Analyse du besoin
- Spécifications fonctionnelles
- Conception globale :
  - forte cohésion dans un module
  - casser les longues chaînes en résultat intermédiaire
  - prévoir une politique de gestions des erreurs
  - définit précisément les interfaces + indépendances des modules
  - planifier les tests d'intégration avec les environnements

Les idées essentielles : Tester à chaque cycle de vie

Les Tests et les modèles itératifs.
“Agile” = réduire le cycle de vie du logiciel en accélérant son développement.

Version minimale puis intégration des fonctionnalités. Basées sur l’écoute du client.

Généralement fait appelle à l'intégration continue (cf Jenkins);

Les idées essentielles : Tester à chaque cycle de vie : Mini Bilan

1: Qu'est-ce que le cycle en V en développement de logiciel ?

2: Qu'est-ce que la conception en développement de logiciel ?

3: Qu'est-ce que le codage en développement de logiciel ?

4: Quels sont les conseils pour réussir les tests de logiciels ?

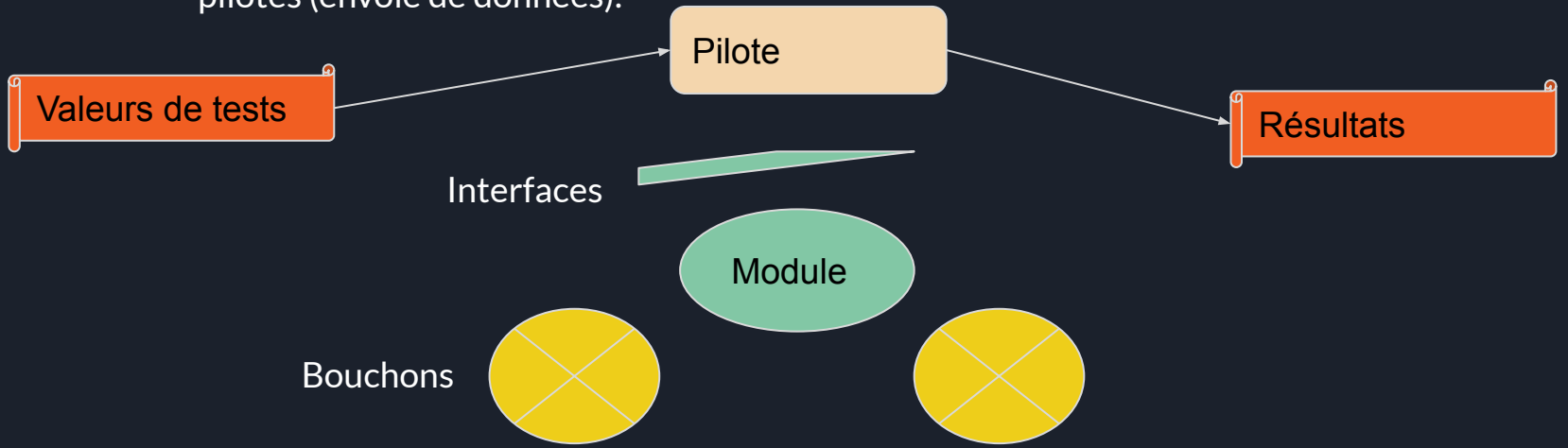
5: Quels sont les éléments clés pour la conception globale des tests de logiciels ?

Les idées essentielles : Tester à chaque cycle de vie

Les tests de composants ou les tests unitaires.

Composant = unité logicielle clairement identifiable, relativement autonome et avec des interfaces clairement définies.

Nécessite isolation du reste de l'application : bouchons (recevoir des données) et pilotes (envoie de données).



Les idées essentielles : Tester à chaque cycle de vie

Les tests de composants ou les tests unitaires.

Default recherchés :

- variables utilisées et non initialisées.
- conditions inversées
- sortie de boucle prématurée
- absence de code défensif concernant des conditions d'entrée dans le composant non prévues
- utilisation pointeur nul

Erreurs seront découverte par des défaillances :

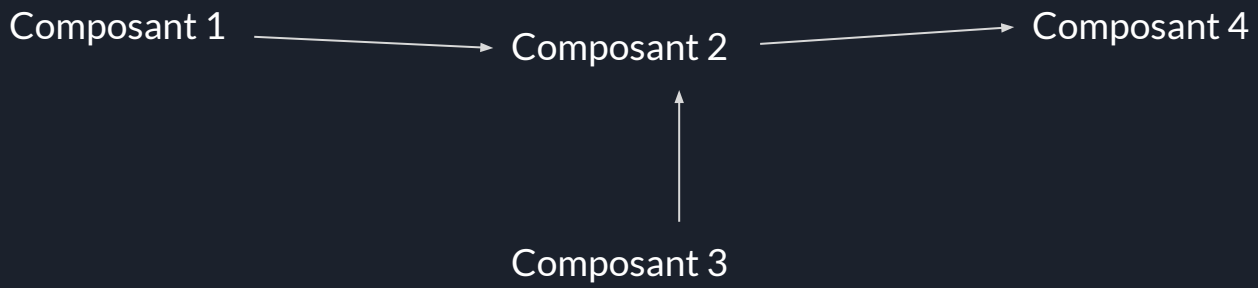
- messages d'erreurs sortis à tort
- sortie du composant sur des valeurs non prévues
- violation mémoire
- dépassement de capacité

Les idées essentielles : Tester à chaque cycle de vie : test unitaire : Bilan

- 1: Quelle est la différence entre un bouchon et un pilote dans les tests de composants ?
- 2: Qu'est-ce qu'un composant en développement de logiciel ?
- 3: Pourquoi les tests de composants nécessitent-ils l'isolation du reste de l'application ?
- 4: Quels sont les défauts recherchés lors des tests de composants ?
- 5: Comment les erreurs sont-elles découvertes lors des tests de composants ?
- 6: Quels types de défaillances peuvent être découverts lors des tests de composants ?
- 7: Qu'est-ce qu'un test unitaire en développement de logiciel ?

Les idées essentielles : Tester à chaque cycle de vie

Les tests d'intégration se concentrent sur les interfaces entre les composants, les différentes parties du système. Concentration sur les échanges au niveau des interfaces.



Les idées essentielles : Tester à chaque cycle de vie : Test intégration : Bilan

1. Qu'est-ce que les tests d'intégration ?

2. Sur quoi les tests d'intégration se concentrent-ils ?

3. Pourquoi est-il important de tester les échanges au niveau des interfaces ?

4. Quels sont les types d'erreurs qui peuvent être détectés lors des tests d'intégration ?

5. Comment les tests d'intégration diffèrent-ils des tests unitaires ?

Les idées essentielles : Tester à chaque cycle de vie

Les tests système ou de validation, tester le logiciel dans son intégralité. Vision fonctionnelle ET non-fonctionnelle.
Par nature test en boîtes noires.

Les documents relatifs à ces tests comprennent:

- exigences fonctionnelles,
- les spécifications non-fonctionnelles qui décrivent la mission (performance, tolérances aux pannes, sécurité..) et la façon (maintenabilité, évolutivité).
- manuel utilisateur
- les cas d'utilisations (aide pour la mise au point de scénarios de tests).
- documents historiques (versions, problèmes passés…).

Les idées essentielles : Tester à chaque cycle de vie

Les tests d'acceptation, tests alphas et tests bêta, pour s'assurer du fonctionnement du système en conformité avec les besoins et les spécifications :

- test d'acceptation par les utilisateurs (IHM et intuitivité)
- test d'acceptation opérationnelle qui principalement les administrations du système (sauvegarde, reprise après sinistre, gestion utilisateur, gestion sécurité..
- test d'acceptation contractuelle qui concerne le client
- les tests d'acceptation réglementaire par rapport aux règlements et législation en cours

Test alpha si l'équipe développement != equipe de validation mais équipe interne
Test beta = équipe de test chez le client

Les idées essentielles : Tester à chaque cycle de vie : Test Validation / Système : Bilan

1. Qu'est-ce que les tests système ou de validation et quel est leur objectif ?

2. Quels sont les types de documents relatifs aux tests système ou de validation ?

3. Quels sont les différents types de tests d’acceptation et à quoi ils servent ?

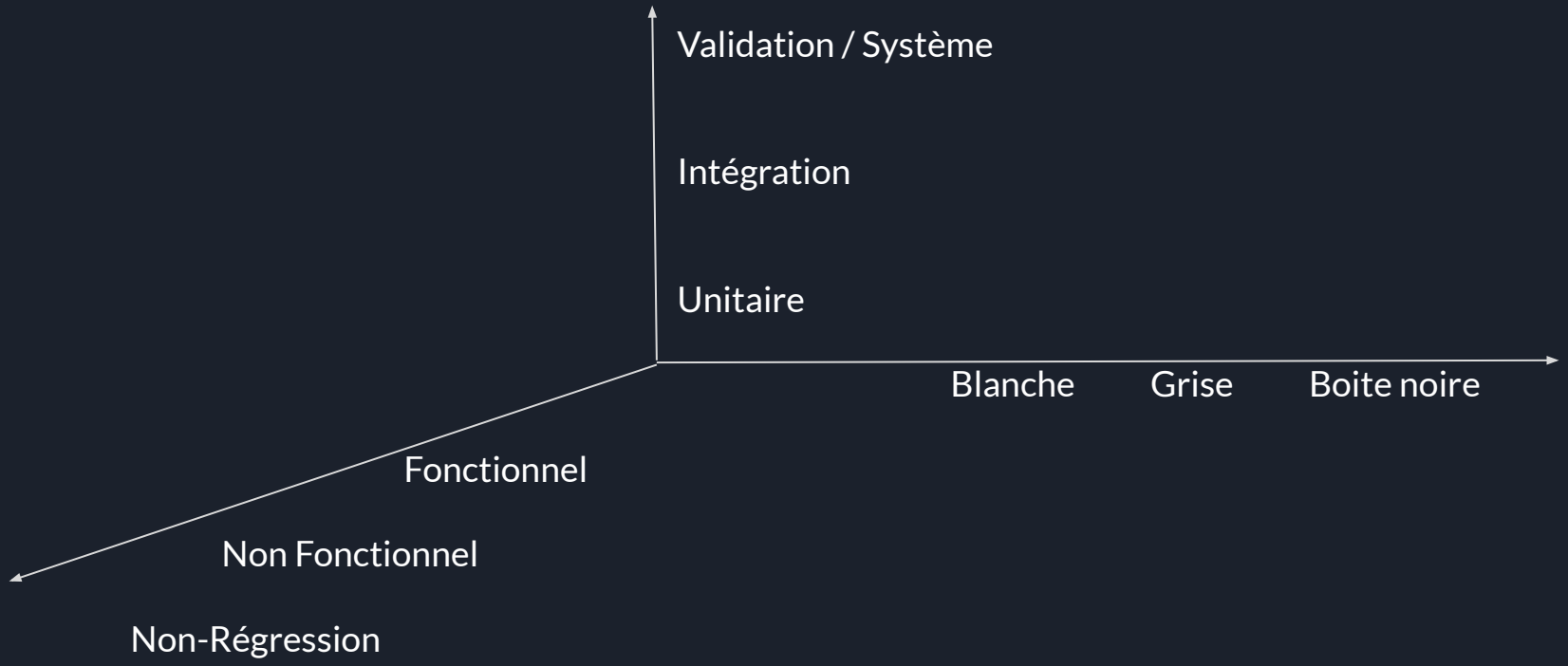
4. Quelle est la différence entre un test alpha et un test bêta ?

5. Pourquoi est-il important de réaliser des tests d'acceptation réglementaire ?

6. Pourquoi les tests système ou de validation sont-ils considérés comme des tests en boîte noire ?

Les idées essentielles : Tester à chaque cycle de vie

Résumé des différents type de test



Les idées essentielles : Tester à chaque cycle de vie

Fonctionnel : sur ce qu'il fait
Non-Fonctionnel : pas sur ce qu'il fait mais la façon

F	U	R	P	S	E
Functionnality	Usuability	Reliability	Performances	Serviceability	Evolutivité
Précision et fiabilité des résultats	Compréhension Exploitation Ergonomie	Tolérance aux pannes Remise en état de marche	Temps de réponse Utilisation des ressources	Analyse de défaillance Stabilité Test	Adaptation Remplacement Modifications Cohabitation

Plus complexes car demande analyse poussée.
Test de non régression.

Tester efficacement

Test Statique vs Test Dynamique (Automatisation possibles);

Tests Dynamiques :

- Consistent à exécuter le code pour vérifier son fonctionnement
- Permettent de découvrir des erreurs telles que les erreurs de mémoire, les erreurs de gestion des exceptions et les erreurs de traitement des données.
- Sont effectués en utilisant des entrées et des sorties pour vérifier le bon fonctionnement du code.

Tests Statiques :

- Consistent à examiner le code sans l'exécuter.
- Peuvent inclure des revues de code, des inspections et des analyses de code statique.
- Permettent de découvrir des erreurs telles que les erreurs de syntaxe, les erreurs de typage et les erreurs de mémoire.

Tester efficacement

- Aperçu des stratégies de Tests Statiques Automatisables :

Proche du compilateur :

- niveau d'imbrication des boucles
- nombre d'instructions par programme
- nombres de paramètres
- utilisation variable non initialisée.
- réalisation arithmétiques overflow
- présence de code mort

Spécial polymorphisme :

CBO = nombre de classes auxquelles une classe est couplée	NOC = nombre de sous-classes immédiates d'une classe	WMC = nombre de méthodes définies en classe
Deux classes sont couplées lorsque des méthodes déclarées dans une classe utilisent des méthodes ou des variables d'instance définies par l'autre classe	DIT = chemin d'héritage maximal de la classe à la classe racine	

Tester efficacement

- Tests statiques non automatisable :

Revue de code : vérification formelle du code source.

Effectué par une personne différente de celle qui a écrit le code pour améliorer la qualité du code et de le rendre plus robuste et fiable.

Revue technique : vérification formelle du design du logiciel.

> Le but est de s'assurer que le design est cohérent, qu'il répond aux exigences du projet et qu'il peut être implémenté avec succès. Effectué par une équipe de développeurs ou par un expert en conception de logiciels.

Inspection : vérification formelle dans laquelle un groupe de personnes examinent un document ou un produit pour détecter les erreurs et les problèmes potentiels. (modérateur, secrétaire, l'inspecteur…)

buddy check (formelle) vs walkthrough (reunion formelle) vs inspection

Tester efficacement

- Tests Dynamiques :

Tests Aléatoires :

Consistent à utiliser des entrées aléatoires pour tester le logiciel.

- Permettent de découvrir des erreurs qui pourraient ne pas être découvertes avec des entrées prédéterminées.
- Sont souvent utilisés en conjonction avec d'autres méthodes de test pour couvrir un large éventail de scénarios de test.

Tests Statistiques :

Consistent à utiliser des données statistiques pour valider le logiciel.

- Peuvent inclure des tests de fiabilité, des tests de performance et des tests de robustesse.
- Sont souvent utilisés pour évaluer la qualité du logiciel en fonction de critères quantitatifs tels que la fiabilité, les performances et la robustesse.

Tester efficacement

- Tests Dynamiques :

Technique de Boîte Noire :

Consiste à tester le logiciel sans connaître son implémentation interne.

- Se concentre sur les entrées et les sorties du logiciel plutôt que sur son implémentation interne.
- Permet de vérifier le bon fonctionnement du logiciel sans être affecté par les modifications apportées à l'implémentation interne.

Technique de Boîte Blanche :

Consiste à tester le logiciel en connaissant son implémentation interne.

- Se concentre sur l'implémentation interne du logiciel plutôt que sur ses entrées et sorties.
- Permet de découvrir des erreurs de programmation et de concevoir des tests plus ciblés en connaissant le fonctionnement interne du logiciel.

Tester efficacement : Bilan

1. Quelle est la différence entre les tests dynamiques et les tests statiques ?
2. Quelles erreurs peuvent être découvertes grâce aux tests dynamiques ?
3. Quelles erreurs peuvent être découvertes grâce aux tests statiques ?
4. Quels sont les quatre types de tests possibles ?
5. Quelles sont les stratégies de tests statiques automatisables ?
6. Qu'est-ce que la revue de code et qui l'effectue ?
7. Qu'est-ce que la revue technique et qui l'effectue ?
8. Qu'est-ce que l'inspection et qui la pratique ?

Tester efficacement : Bilan 2

1. Quelle est la différence entre un buddy check et un walkthrough ?
2. Qu'est-ce que les tests aléatoires et pourquoi sont-ils utiles ?
3. Quels sont les types de tests statistiques ?
4. Quelle est la fonction des tests statistiques ?
5. Qu'est-ce que la technique de boîte noire et sur quoi se concentre-t-elle ?
6. Qu'est-ce que la technique de boîte blanche et sur quoi se concentre-t-elle ?

Tester efficacement : Bilan 3

1. Pourquoi est-il utile de connaître l'implémentation interne d'un logiciel lors de la réalisation de tests boîtes blanches?
2. Comment la technique de boîte noire peut-elle aider à vérifier le bon fonctionnement d'un logiciel ?
3. Comment la technique de boîte blanche peut-elle aider à découvrir des erreurs de programmation ?
4. Comment les tests dynamiques peuvent-ils être effectués ?
5. Comment les tests statiques peuvent-ils être effectués ?

Les spécifications et les jeux de tests

Tests All Singles : Consistent à tester chaque entrée individuellement.

- Permet de couvrir toutes les entrées possibles et de découvrir les erreurs qui pourraient se produire avec une entrée donnée.

Tests All Pairs : Consistent à tester toutes les combinaisons possibles de deux entrées.

Systèmes	**Navigateur**	**Type de fichiers**	**Connexions**
W10 / 08 / 11	Chrome	jpeg	sécurité
Linux	Firefox	mp3	non sécurisé
MacOs	Explorer	pdf	
		csv	

Construire les 2 tableaux pour all pairs and all singles

Les spécifications et les jeux de tests

Tests All Singles, Tests All Pairs;

Les **classes d'équivalences :** consistent à regrouper des entrées de test en groupes logiques basés sur leur comportement similaire attendu.

- Permettent de réduire le nombre de cas de test nécessaires en ne testant qu'une représentation représentative de chaque classe d'équivalence.
- Les classes d'équivalences peuvent être basées sur des critères tels que les limites de valeur, les types de données, les conditions de bordure et les relations entre les entrées.
- Il est important de déterminer les classes d'équivalences de manière exhaustive pour s'assurer que tous les cas importants sont testés.

Validité des entrées : int	**Classes d'équivalences**	**Données de test**
Entrèes valides : âge	[0: 126]	60
Entrées invalides	[minInt : -1]	-4
Entrées invalides	]126 : maxInt]	140

Les spécifications et les jeux de tests

Construction des classes d'équivalences :

Condition Entrée définit par un intervalle de valeur : département francais 1 - 95

Défintion de la Condition d'entrée	**Classses d'équivalences necessaires**	**Exemples**
Intervalle de valeurs	Valide + 2 invalides (chaque bout de l'intervalle)	département francais : 1 - 95
N valeurs	Valide : N valeurs + 2 invalides (mois de N et plus de N)	Tableau
Un ensemble de valeurs	classe valides au cas par cas. classes invalides si possible	Enumeration de valeurs
Définit par une contrainte	valides (condition vérifiée) + invalies condition non vérifiée	pas commencer par une lettre
Classe complexes	possibilité de découper en sous catégories ?	

Les spécifications et les jeux de tests

Exemple : Lendemain. valide pour les années entre 1582 et 3000

Entrée	**Classes d'équivalences valides**	**Classes d'équivalences invalides**
Jour		
Mois		
Année		

Les spécifications et les jeux de tests

Exemple : Lendemain. valide pour les années entre 1582 et 3000

Entrée	**Classes d'équivalences valides**	**Classes d'équivalences invalides**
Jour	[1, 31]	< 1 > 31
Mois	[1, 12]	< 1 > 12
Année	[1582, 3000]	< 1582 >3000

Les spécifications et les jeux de tests

Exemple : Lendemain. valide pour les années entre 1582 et 3000. Affinons les classes d'équivalences valides pour les mois et année.

Entrée	**Classes d'équivalences valides**
Mois	
Année	

Les spécifications et les jeux de tests

Exemple : Lendemain. valide pour les années entre 1582 et 3000. Affinons les classes d'équivalences valides pour les mois

Entrée	**Classes d'équivalences valides**
Mois	{1, 3, 5, 7, 8 10, 12} {4, 6, 9, 11} 2
Année	Année bissextileentre [1582, 3000] Année non bissextile entre [1582, 3000] Année 2000

Les spécifications et les jeux de tests

Exemple : Lendemain. valide pour les années entre 1582 et 3000. Affinons les classes d'équivalences valides pour les mois et les jours

Entrée	**Classes d'équivalences valides**
Jour	
Mois	{1, 3, 5, 7, 8 10, 12} {4, 6, 9, 11} 2

Les spécifications et les jeux de tests

Exemple : Lendemain. valide pour les années entre 1582 et 3000. Affinons les classes d'équivalences valides pour les mois et les jours

Entrée	**Classes d'équivalences valides**
Jour	[1, 28] 29 30 31
Mois	{1, 3, 5, 7, 8 10} {4, 6, 9, 11} 12 2

Les spécifications et les jeux de tests

Exemple : Lendemain. Tableau complet.

Entrée	**Classes d'équivalences valides**
Jour	[1, 28] 29 30 31
Mois	{1, 3, 5, 7, 8 10} {4, 6, 9, 11} 12 2
Année	Année bissextileentre [1582, 3000] Année non bissextile entre [1582, 3000] Année 2000

Exercice Tableaux all pairs + Jeux de valeurs pour tester les classes invalides

Les spécifications et les jeux de tests

Exemple : Lendemain. Autre type de test (plus en boite noire)
Les tests dit au limites.
Prendre les valeurs limites des classes d'équivalences (mais prendre les valeurs valides pour compararer les résultats).

Entrée	**Classes d'équivalences valides**	**Valeurs limites**
Jour	[1, 28]	
Mois	{1, 3, 5, 7, 8 10} {4, 6, 9, 11} 12 2	
Année	Année bissextileentre [1582, 3000] Année non bissextile entre [1582, 3000] Année 2000	

Les spécifications et les jeux de tests

Exemple : Lendemain. Autre type de test (plus en boite noire)
Les tests dits aux limites.
Prendre les valeurs limites des classes d'équivalences (mais prendre les valeurs valides pour comparer les résultats).

Entrée	**Classes d'équivalences valides**	**Valeurs limites**
Jour	[1, 28]	1 et 28
Mois	{1, 3, 5, 7, 8 10} {4, 6, 9, 11} 12 2	1, 10, 4, 11, 12, 2
Année	Année bissextileentre [1582, 3000] Année non bissextile entre [1582, 3000] Année 2000	1600, 2000 1582, 3000

Les spécifications et les jeux de tests

Tester grâce à une table de décisions.
Logiciel peut être décrit par ses actions.

Condition / Variable	**Combinaison de Valeurs**		
C1	V1	V1	V1
C2	V2	V2	V2
C3	V3	V3	V3
Action			
A1			
A2			

Les spécifications et les jeux de tests

Tester grâce à une table de décisions. Exemple Problèmes des Triangles.

Programme avec en entrée trois valeurs entière inférieure à 20 qui affiche “isocèle”, “equilatérale”, “scalène” ou “quelconque”.

Trouver Actions et Conditions du programme pour préparer un tableau de décision.
A1 = ..

C1 = …
C2 =....

Les spécifications et les jeux de tests

Tester grâce à une table de décisions. Exemple Problèmes des Triangles.

Condition / Variable		
C1 :	C2	C3
Action		
A1		

Les spécifications et les jeux de tests

Tester grâce à une table de décisions. Exemple Problèmes des Triangles.

Condition / Variable		
C1 : a < 20	C4 : a < a + c	C7 : a = b
C2 : b < 20	C5 : b < a + c	C8 : a = c
C3 : c < 20	C6 : c < a + b	C9 : b = c
Action		
A1 : Affiche Equilatéral	A3: Affiche scalène	A5 : Erreur entrée
A2 : Affiche Isolèce	A4: affiche pas un triangle	

Les spécifications et les jeux de tests

Tester grâce à une table de décisions. Exercice avec Lendemain

Trouver Actions et Conditions du programme pour préprarer un tableau de décision.
A1 = ..

C1 = …
C2 =....

Les spécifications et les jeux de tests

Tester grâce à une table de décisions. Exemple avec Lendemain. Puis construire la table de décision

Condition / Variable		
C1 : M1 31 jours	C4 : M4 : décembre	….
C2 : M2 moins de 30 jours	C5 : J1 [1, 27]	
C3 : M3 fèvrier	C6 : J2 28	
Action		
A1 : chgt année	A3: chgt jour sans autre modification	
A2 : chgt mois	A4: data incorrecte	

Les spécifications et les jeux de tests

Tester grâce à un **diagramme états.** Représenter les différents états d'un système et les transitions entre ces états. Un distributeur de billets :

- États :
  - **Initial** : L'appareil est prêt à recevoir une demande de retrait.
  - **Demande de retrait** : L'utilisateur a entré une demande de retrait.
  - **Vérification des fonds :** Le système vérifie si le solde de l'utilisateur est suffisant pour le retrait demandé.
  - **Retrait accepté :** Le retrait est autorisé et les fonds sont déduits du solde de l'utilisateur.
  - **Retrait refusé :** Le retrait est refusé en raison d'un solde insuffisant ou d'une erreur système.
- Transitions :
  - De Initial à Demande de retrait : L'utilisateur entre une demande de retrait.
  - De Demande de retrait à Vérification des fonds : Le système vérifie les fonds de l'utilisateur.
  - De Vérification des fonds à Retrait accepté : Le retrait est autorisé.
  - De Vérification des fonds à Retrait refusé : Le retrait est refusé.
  - De Retrait accepté à Initial : Le système est de retour à l'état initial et prêt à accepter une nouvelle demande.
  - De Retrait refusé à Initial : Le système est de retour à l'état initial et prêt à accepter une nouvelle demande.

Les spécifications et les jeux de tests

Tester grâce à un **diagramme états.** Comment constructuire des séquences de tests :

La construction de séquences de tests consiste à identifier les scénarios de test nécessaires pour couvrir tous les états et toutes les transitions d'un système modélisé. Cela implique généralement d'appliquer de règles pour s'assurer que les séquences de test soient complètes et exhaustives.

Les règles couramment utilisées pour la construction de séquences de tests incluent :

- Atteindre au moins une fois chaque état : Cela garantit que chaque état a été testé et que le systèm peut fonctionner correctement dans tous les états.

- Couvrir toutes les transitions :. Cela garantit que le système peut fonctionner correctement dans toutes les situations et que toutes les erreurs potentielles peuvent être détectées.

- Éviter les tests inutiles : Cela peut aider à réduire le temps et les coûts de test.

Les spécifications et les jeux de tests

- Addition des détails d’implémentation dans les tests :

Les tests boîte blanche sont un type de tests de logiciel qui se concentrent sur la structure interne du code. Ils visent à tester chaque ligne de code, chaque branche de décision et chaque condition dans le code, afin de garantir qu'ils fonctionnent correctement.

Les tests boîte blanche sont généralement effectués en utilisant un graphe de contrôle, qui est une représentation graphique de la structure du code. Le graphe de contrôle montre les relations entre les différentes parties du code et les différents chemins d'exécution qui peuvent être empruntés.

Lors de la planification des tests boîte blanche, il est important de déterminer les critères de couverture pour les tests. Les critères de couverture déterminent les parties du code qui doivent être testées. Les critères les plus courants incluent :

- Toutes les instructions : ex division par zéro
- Tous les chemins : ex division par zéro
- Toutes les conditions modifiées : ex Vitesse, Pente

Les spécifications et les jeux de tests

Exemple

- Toutes les conditions modifiées :

  Freiner = (Vitesse > 100 or Pente > 30) and ((Mode == 0) or Mode ==1)

Faire la Table MC/MD

Vitesse >100	Pente > 30	Mode = 1	Mode = 2	Feiner

Les spécifications et les jeux de tests

Exemple

Freiner = (Vitesse > 100 or Pente > 30) and ((Mode == 0) or Mode ==1)

Faire les complémentaires

Vitesse >100	Pente > 30	Mode = 1	Mode = 2	Feiner	Vitesse >100	Pente > 30	Mode = 1	Mode = 2

Les spécifications et les jeux de tests

Exemple

Freiner = (Vitesse > 100 or Pente > 30) and ((Mode == 0) or Mode ==1)

Faire les complémentaires

Vitesse >100	Pente > 30	Mode = 1	Mode = 2	Feiner	Vitesse >100	Pente > 30	Mode = 1	Mode = 2

Processus et test d’intégration

Processus d'intégration :

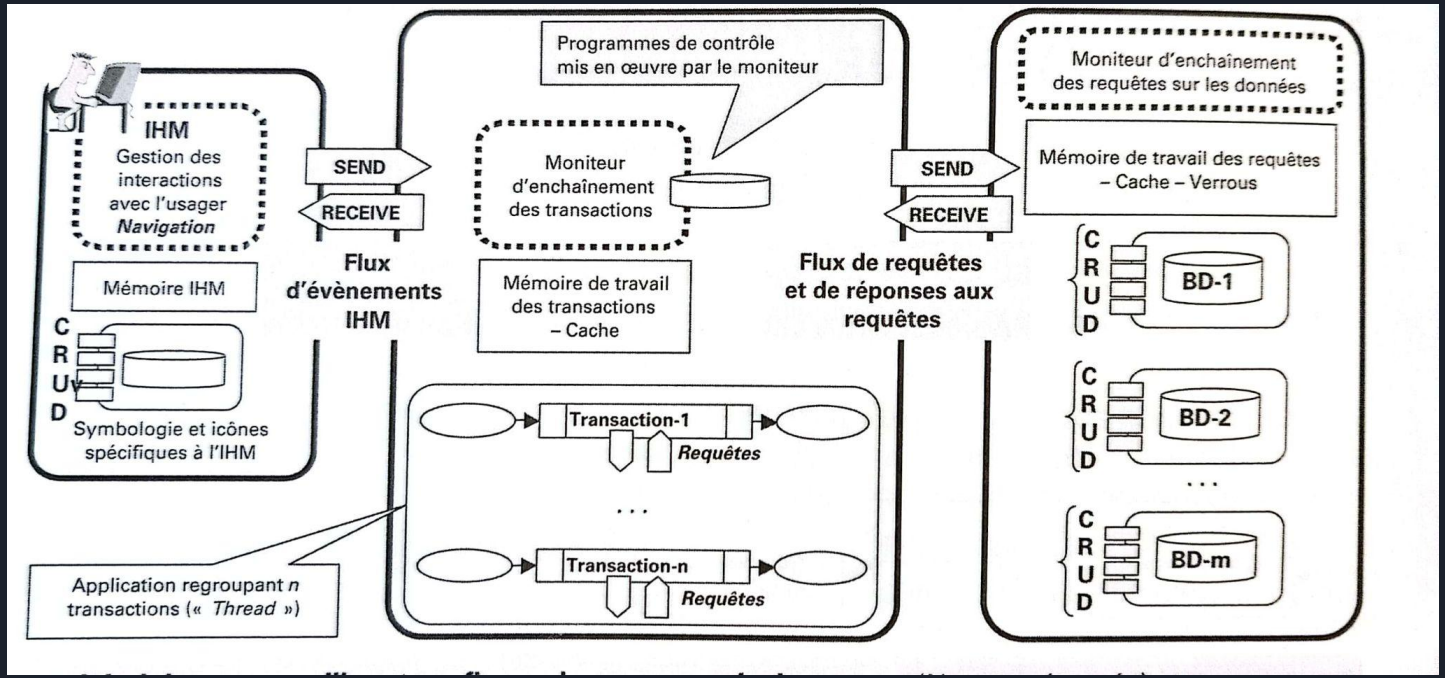
- Le processus d'intégration consiste à combiner des composants logiciels individuels en un système cohérent et fonctionnel
- Il s'agit d'une étape clé dans le développement de logiciels, qui permet de vérifier que les composants individuels fonctionnent ensemble comme prévu.
- Le processus d'intégration peut être effectué à différents niveaux d'abstraction, y compris les tests de module, les tests de composant, les tests de sous-système et les tests de système.

Tests d'intégration :

- Les tests d'intégration sont utilisés pour s'assurer que les composants individuels fonctionnent ensemble comme prévu.
- Les tests d'intégration peuvent être effectués à différents niveaux d'abstraction, y compris les tests de module, les tests de composant, les tests de sous-système et les tests de système.
- Les tests doivent vérifier que les composants individuels sont correctement connectés et qu'ils communiquent correctement entre eux.
- Les tests doivent également vérifier que les interfaces entre les composants sont correctement implémentées et que les données sont correctement transmises.
- Les tests d'intégration peuvent être effectués manuellement ou automatiquement à l'aide d'outils de test.

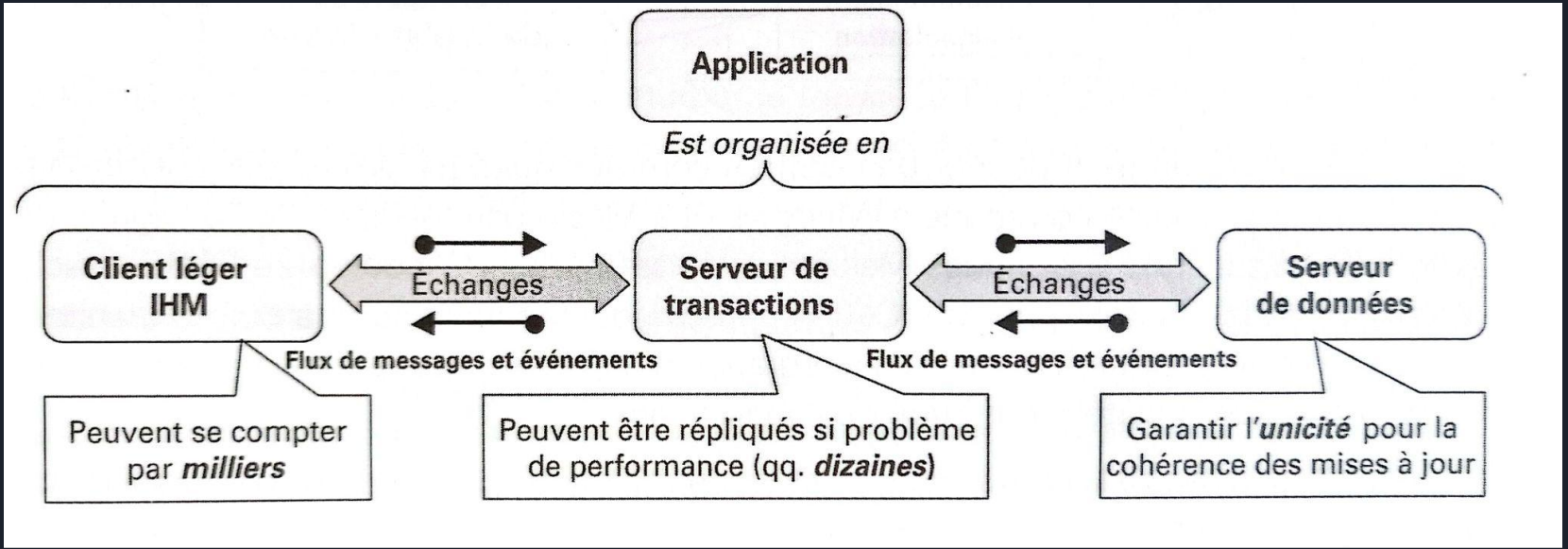
Processus et test d'intégration

Exemple intégration client / serveur



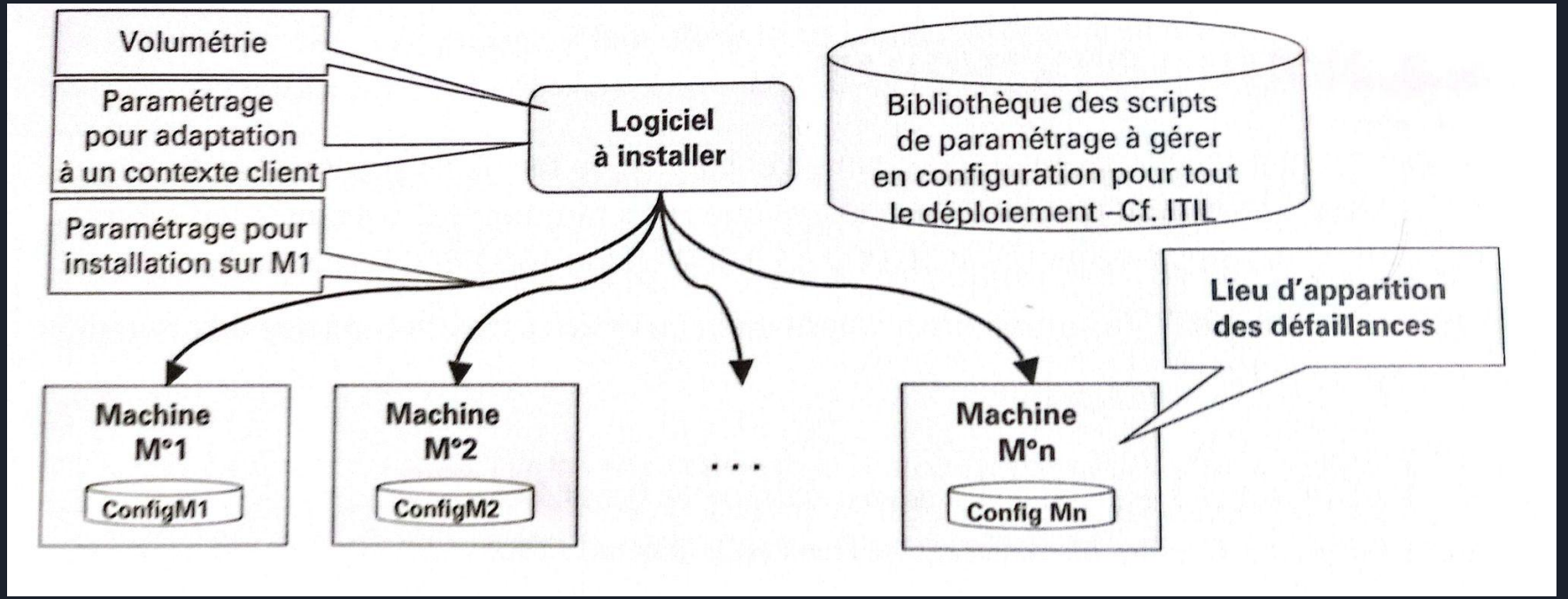
Processus et test d'intégration

Exemple intégration client / serveur : Schéma simplifié.



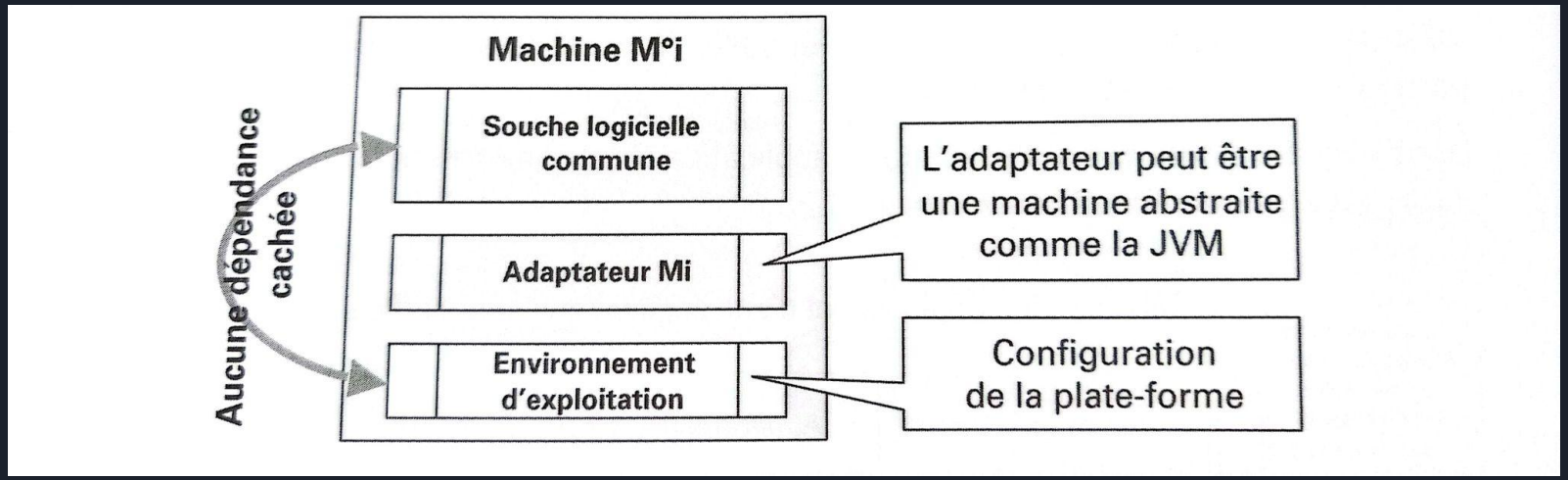
Processus et test d'intégration

Configuration du déploiement :



Processus et test d'intégration

Installation Particulière avec un containeur

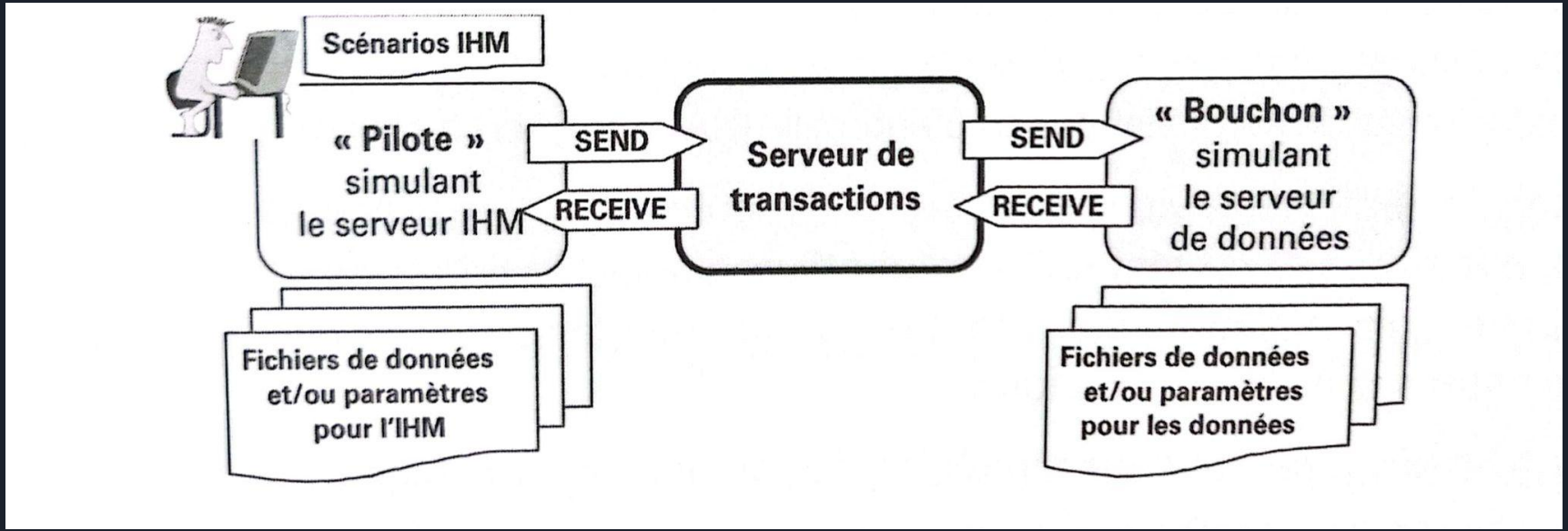


Processus et test d’intégration

Environnement d’intégration pour le client léger IHM

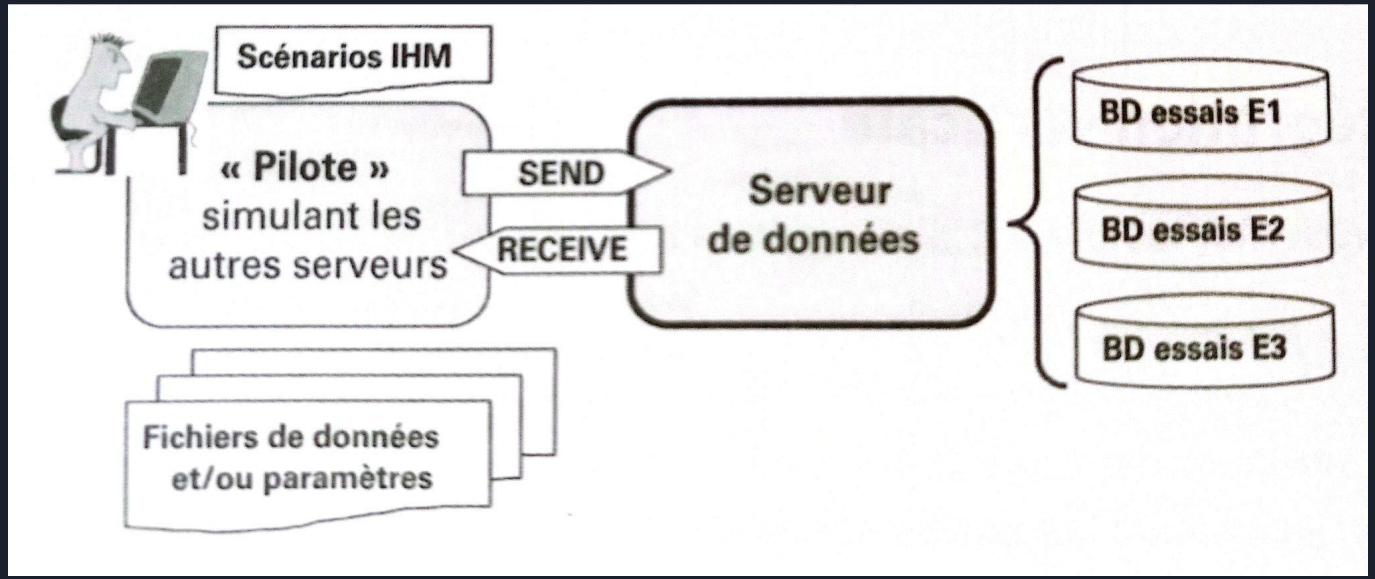
Processus et test d'intégration

Environnement d'intégration pour le serveur de transactions



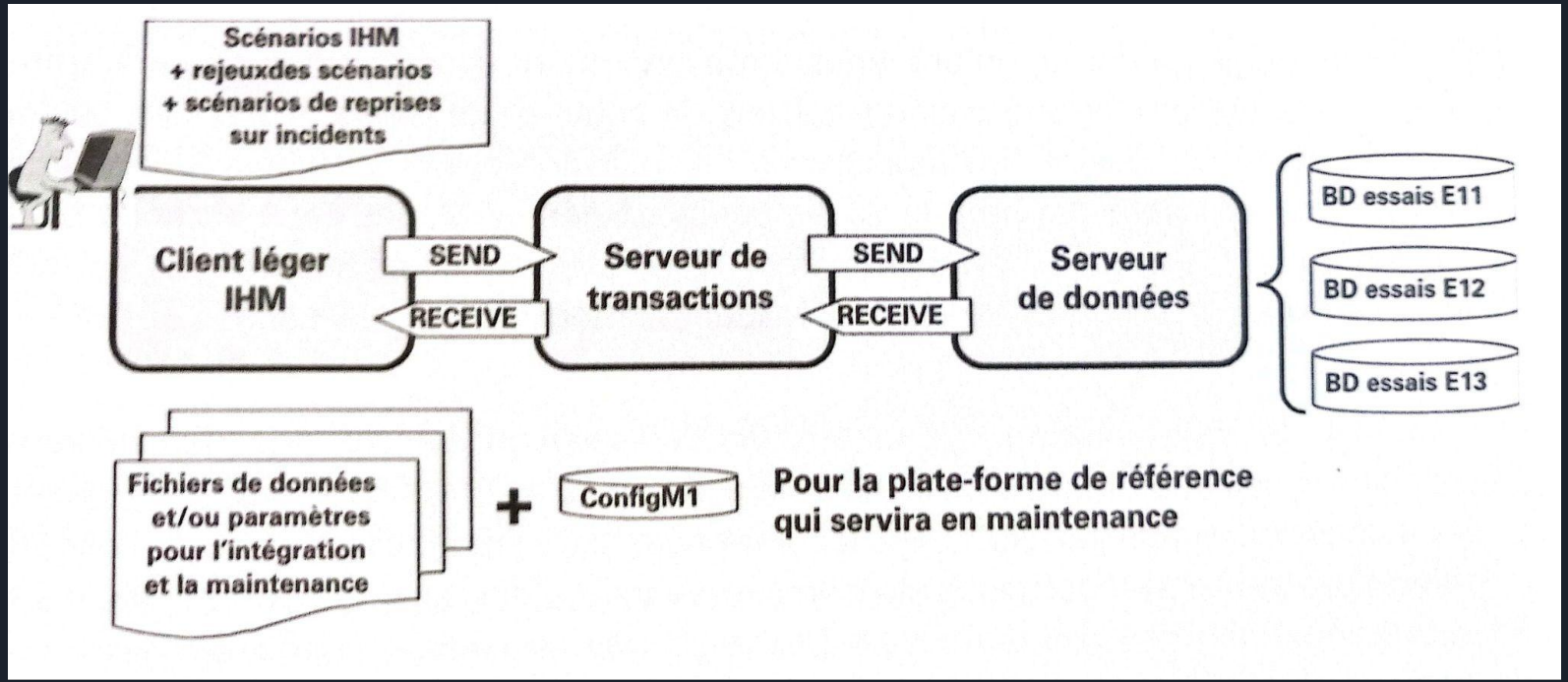
Processus et test d’intégration

Environnement d’intégration pour le serveur de données



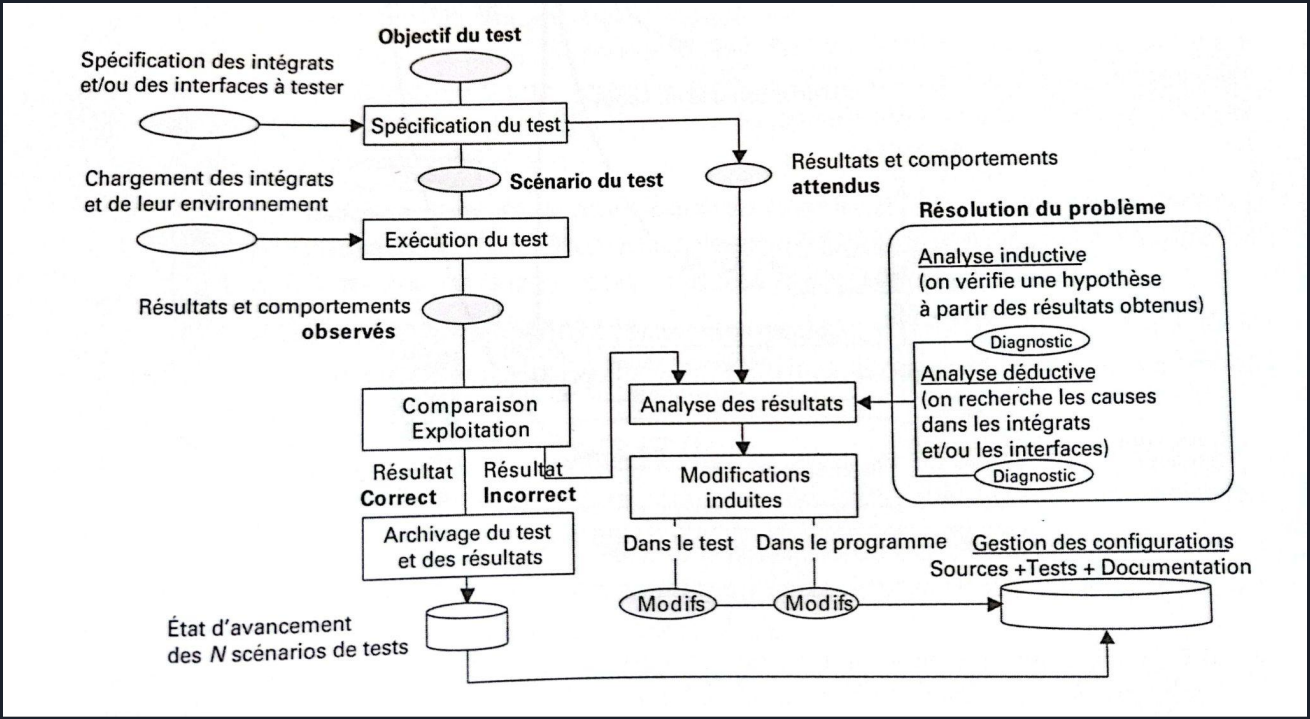
Processus et test d'intégration

Environnement d'intégration de l'application



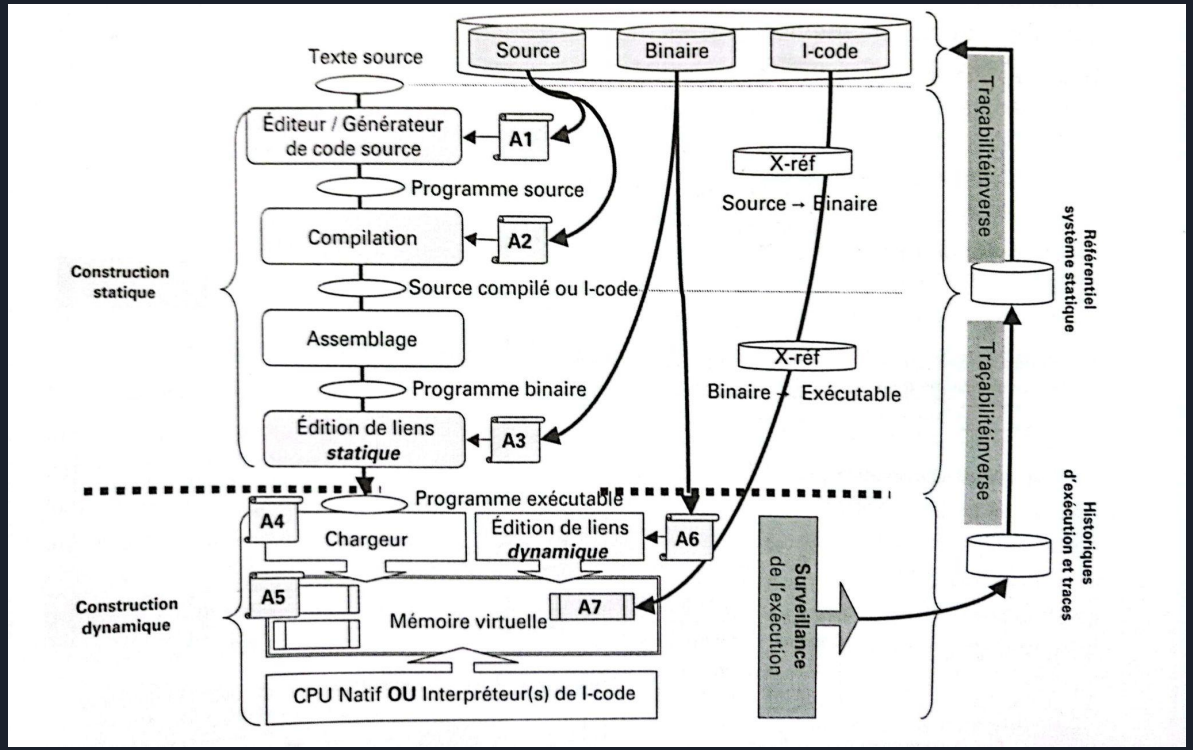
Processus et test d'intégration

Structure du processus de test



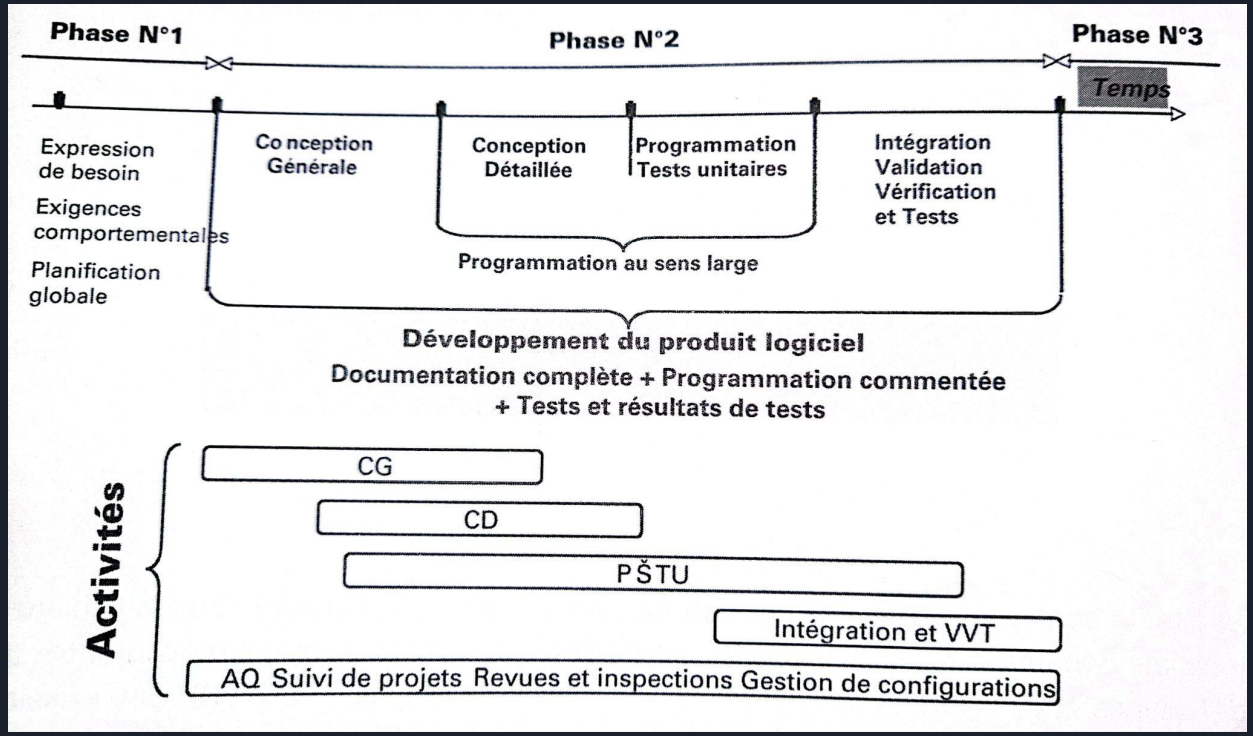
Processus et test d'intégration

Procédé de construction et d'intégration du système



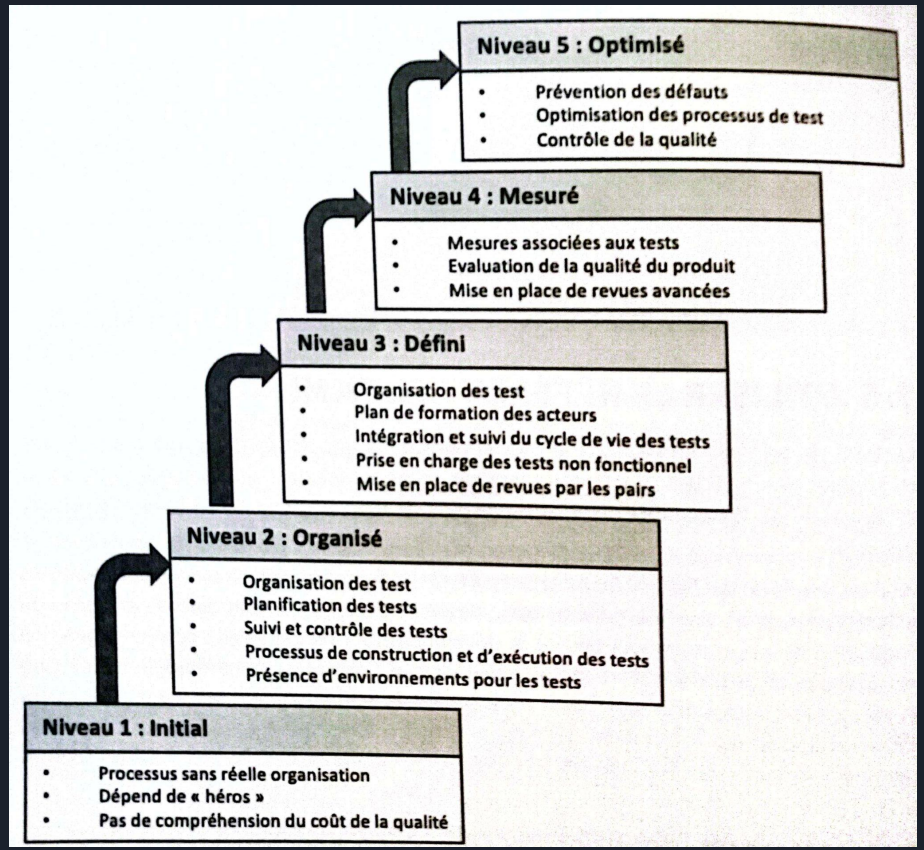
Gérer les tests

Gantt de la phase de développement et activités de tests



Gérer les tests

Le référentiel TMMP (Test Maturity Model Integration)



Outils pour les tests : Intégration continue