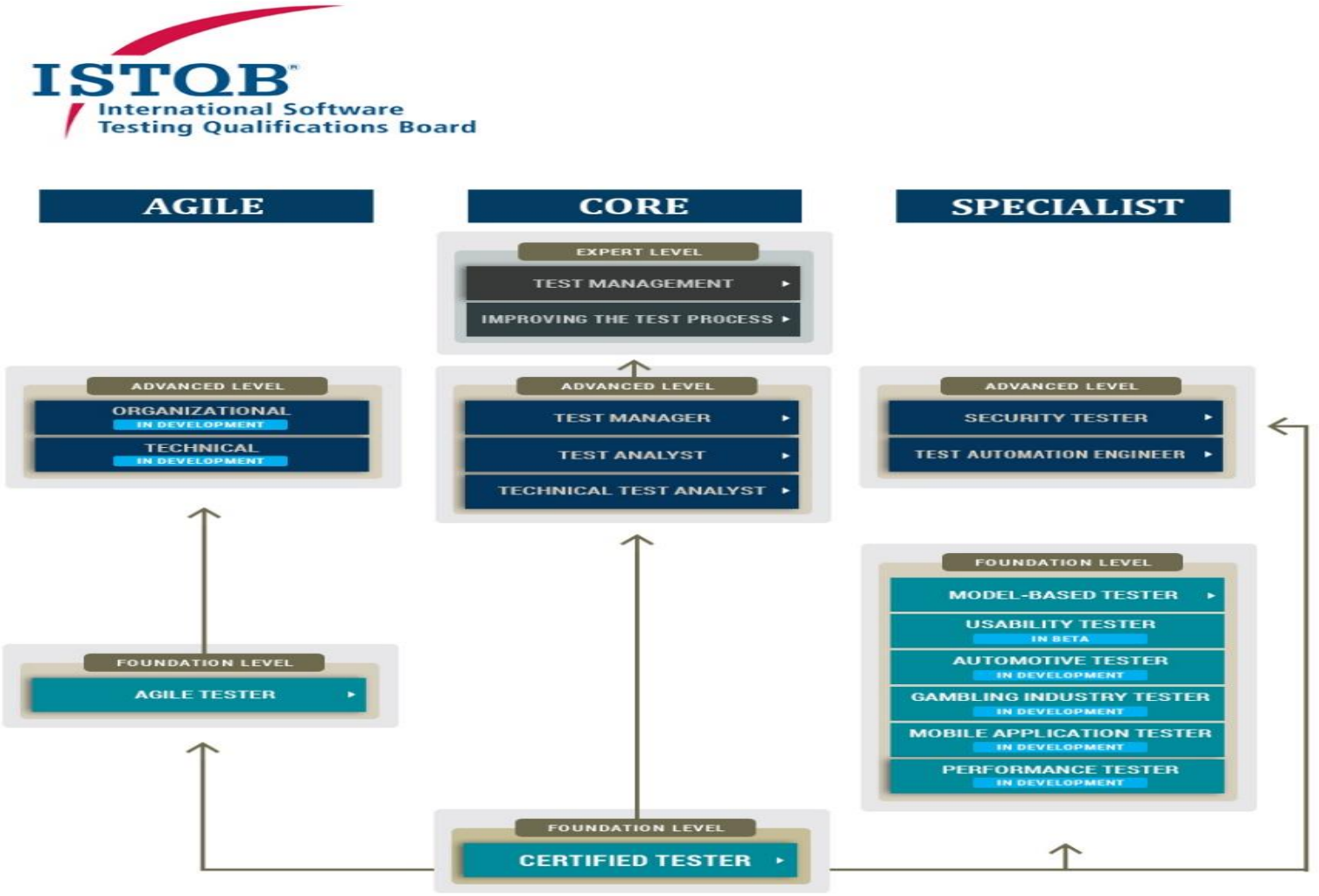
ISTQB Training

xx/xx/xxxx Boughdiri Aymen

ISTQB

- Fondé officiellement in Edenburg en Novembre 2002
- Organisme à but non lucratif qui fournit les bonnes pratiques de tests logiciels
- Développé par plus de 100 experts dans plus que 40 pays
- ISTQB est la certification la plus reconnue dans le monde de test logiciel
- Il y a plus que 650000 testeurs certifiés à travers le monde

ISTQB


ISTQB
International Software Testing Qualifications Board
AGILE
CORE
SPECIALIST
EXPERT LEVEL
TEST MANAGEMENT
IMPROVING THE TEST PROCESS
ADVANCED LEVEL
ORGANIZATIONAL
IN DEVELOPMENT
TECHNICAL
IN DEVELOPMENT
ADVANCED LEVEL
TEST MANAGER
TEST ANALYST
TECHNICAL TEST ANALYST
ADVANCED LEVEL
SECURITY TESTER
TEST AUTOMATION ENGINEER
FOUNDATION LEVEL
AGILE TESTER
FOUNDATION LEVEL
MODEL-BASED TESTER
USABILITY TESTER
IN BETA
AUTOMOTIVE TESTER
IN DEVELOPMENT
GAMBLING INDUSTRY TESTER
IN DEVELOPMENT
MOBILE APPLICATION TESTER
IN DEVELOPMENT
PERFORMANCE TESTER
IN DEVELOPMENT
FOUNDATION LEVEL
CERTIFIED TESTER

ISTQB

- Le test consiste à répondre à 40 questions à choix multiples
- Chaque bonne réponse donne un point
- Pour réussir le participant doit avoir au moins 65% de bonne réponse c'est-à-dire 26 bonnes réponses parmi les 40 questions
- La durée de l'examen est de 60 minutes (1 minute et 30 secondes par question)

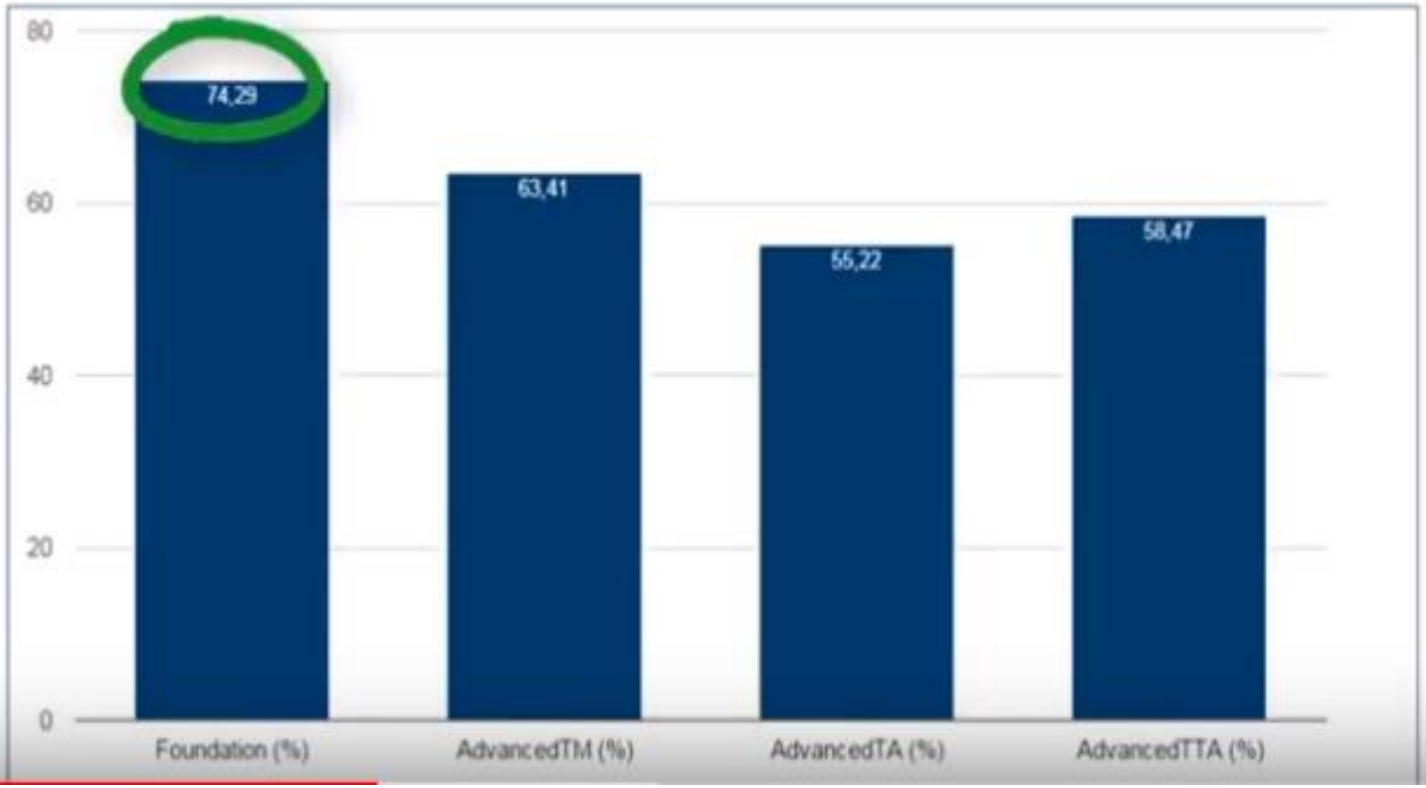
ISTQB

- 4 niveaux d'apprentissage:
  - K1: Se rappeler/ K2: Comprendre / K3: Appliquer / K4: Analyser
- Répartition par niveau cognitif:
  - 50% de chaque examen seront des questions de niveau K1;
  - 30% de chaque examen seront des questions de niveau K2; et
  - 20% de chaque examen seront des questions de niveau K3 / K4

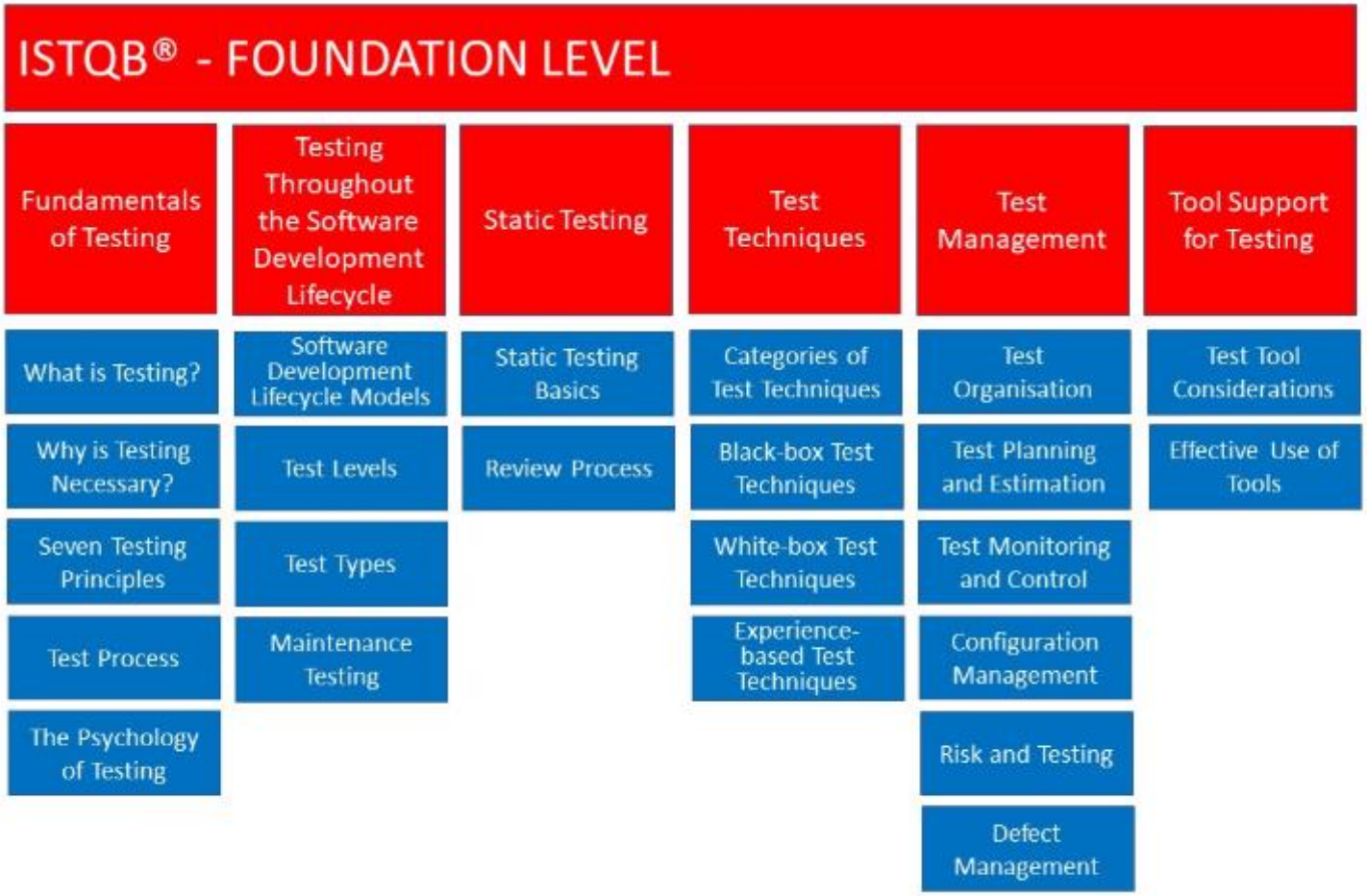
	Time (min)	*Time*	*Number of Questions*	*Range*	*K1 (+/- 1)*	*K2 (+/- 1)*	*K3/K4 (+/- 1)*
Chapter 1	155	18%	7.17	7 to 8	4	3	
Chapter 2	115	13%	5.32	5 to 6	4	2	
Chapter 3	60	7%	2.77	2 to 3	2	1	
Chapter 4	285	33%	13.18	12 to 15	4	2	6
Chapter 5	170	20%	7.86	7 to 8	3	3	2
Chapter 6	80	9%	3.70	3 to 4	3	1	
Total	865	100%	40	40	20	12	8

ISTQB

Le taux de succès



Agenda


ISTQB® - FOUNDATION LEVEL
Fundamentals of Testing
What is Testing?
Why is Testing Necessary?
Seven Testing Principles
Test Process
The Psychology of Testing
Testing Throughout the Software Development Lifecycle
Software Development Lifecycle Models
Test Levels
Test Types
Maintenance Testing
Static Testing
Static Testing Basics
Review Process
Test Techniques
Categories of Test Techniques
Black-box Test Techniques
White-box Test Techniques
Experience-based Test Techniques
Test Management
Test Organisation
Test Planning and Estimation
Test Monitoring and Control
Configuration Management
Risk and Testing
Defect Management
Tool Support for Testing
Test Tool Considerations
Effective Use of Tools

Agenda

- Chapitre 1 : Fondamentaux de tests (Jour 1)
- Chapitre 2 : Tester Pendant le Cycle de Vie Logiciel (Jour 1)
- Chapitre 3 : Techniques Statiques (Jour 2)
- Chapitre 4 : Techniques de Conception de tests (Jour 2)
- Chapitre 5 : Gestion des tests (Jour 3)
- Chapitre 6: Outil de support de tests (Jour 3)
- Test (Jour 3)

Chapitre 1 : Fondamentaux de Tests

Chapitre 1 : Fondamentaux de Tests

- Pourquoi les tests sont-ils nécessaires?
- Que sont les tests?
- Principes généraux de test fondamentaux
- Processus fondamental de test
- La psychologie des tests
- Code d’éthique

1.1 Pourquoi les Tests sont-ils Nécessaires

- Les systèmes logiciels deviennent une part de notre existence, des applications commerciales aux produits de grande consommation

- Des logiciels ne fonctionnant pas correctement peuvent générer de nombreux problèmes: pertes financières, temps, réputations, allant même aux blessures ou mort

- Il est important de détecter et de corriger ces défaillances avant qu'elles soient livrés aux clients

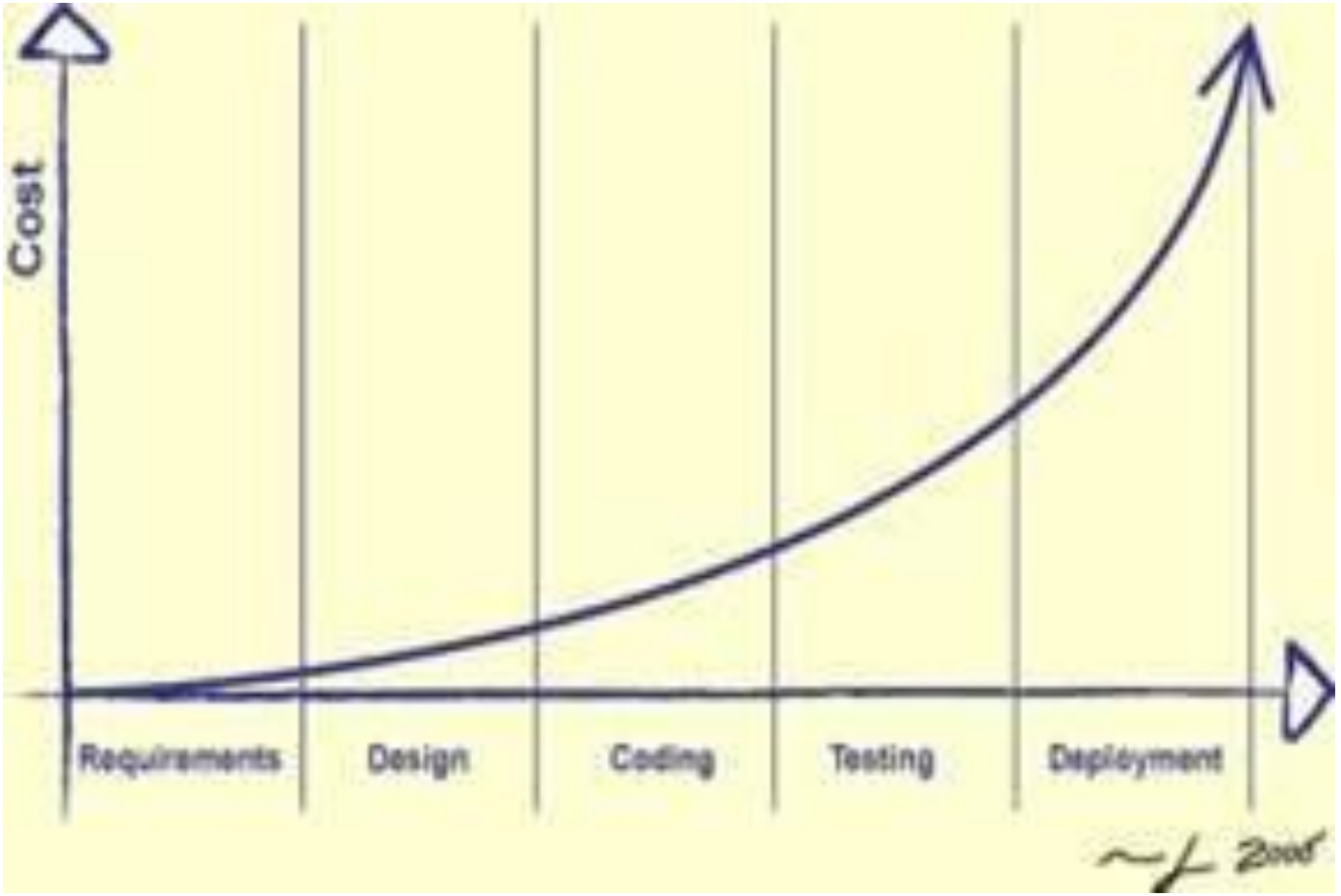
1.1 Pourquoi les Tests sont-ils Nécessaires

- 1996: Ariane 5
  - Utilisation du même software de Ariane 4 (32 bits) sur Ariane5 (64 bits)

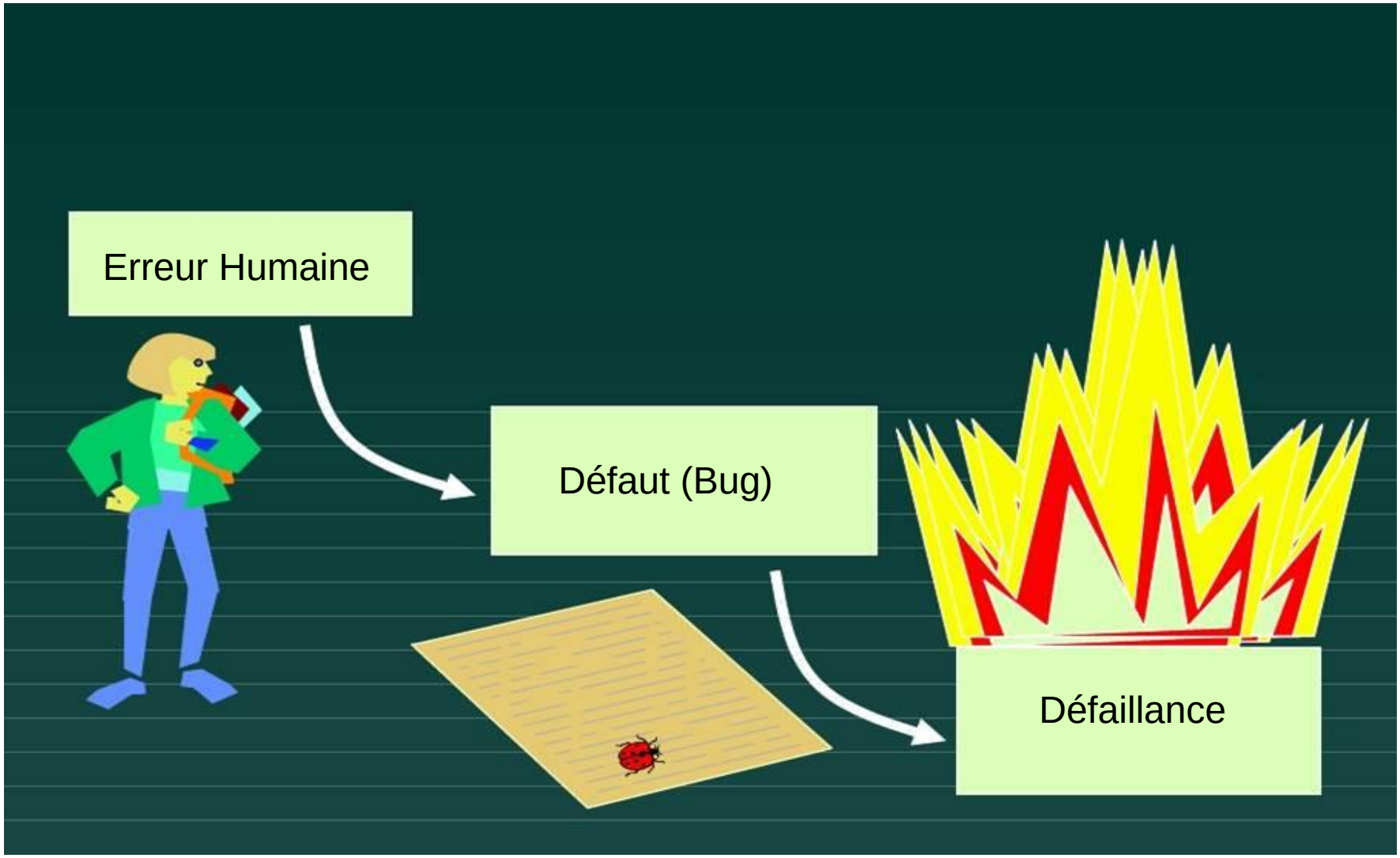
    ➔ 7 billion $
- 1999: Mars climate orbiter
  - Integration d'un module utilisant différents unité (Newton/Libra)

    ➔ 900 million $
- 1999: French Bank
  - Migration de windows NT à windows 2000 : comportment incorrect du software

1.1 Pourquoi les Tests sont-ils Nécessaires



1.1 Pourquoi les Tests sont-ils Nécessaires



1.1 Pourquoi les tests sont ils nécessaires

- Les défaillances peuvent être causées par:
  - Erreur humaine causée par une échéance serrée, complexité du code, infrastructure ou multiple interaction entre les systèmes
  - Condition d'environnement (radiation, magnétisme, pollution …)

1.1 Pourquoi les tests sont ils nécessaires

- Avec l'aide des tests, il est possible de mesurer la qualité des logiciels en termes de défauts trouvés

- En comprenant les causes premières des défauts trouvés dans d'autres projets, les processus peuvent être améliorés, ce qui ensuite peut prévenir l'apparition de ces défauts et, en conséquence, améliorer la qualité des systèmes futurs. C'est un aspect de l'assurance qualité

- Les tests devraient être intégrés comme une activité de l'assurance qualité (p.ex. au côté des standards de développement, de la formation et de l'analyse des défauts)

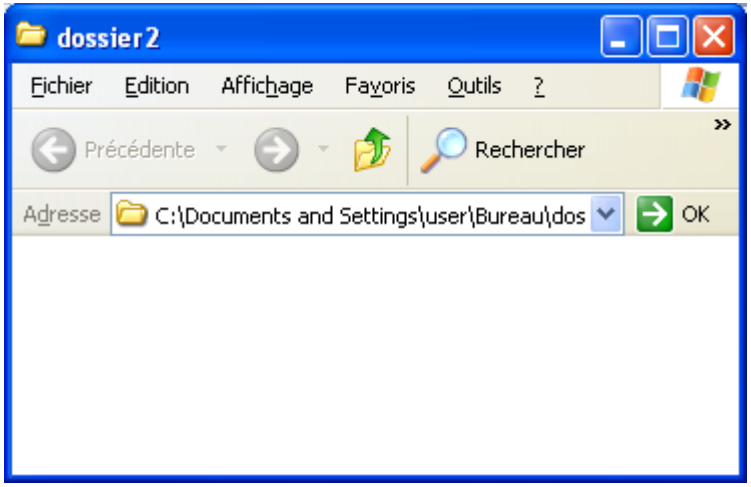
- C'est l'aspect d'aide à la décision => Si produit peut être en production ou non en prenant en compte **le niveau de risque**

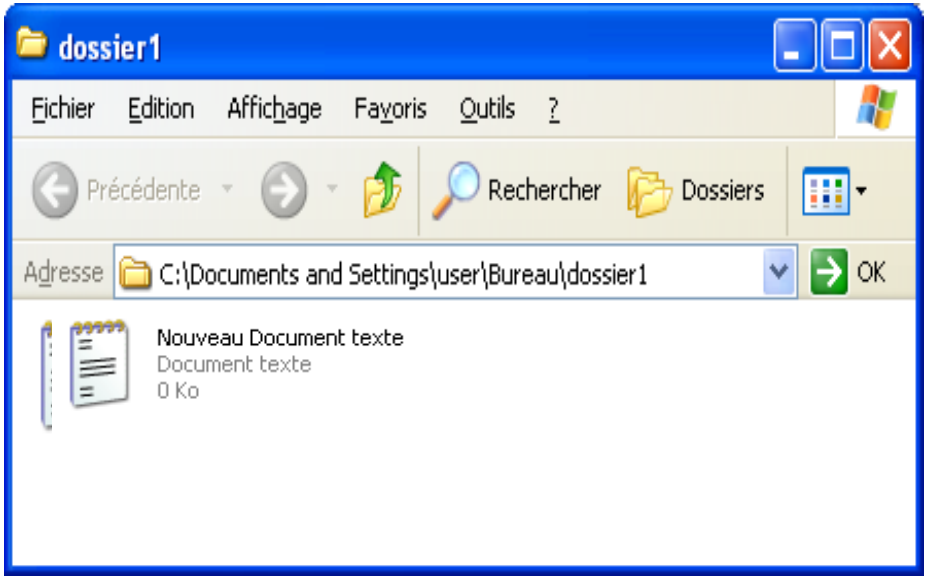
1.2 Que sont les Tests

- Des activités de test existent dans toutes les étapes de cycle de vie logiciel . Il peuvent être dynamiques ou statiques, allant du planning, de la préparation et évaluation du logiciel pour déterminer que le produit répond aux besoins

- Les objectifs de test peuvent varier :
  - **Trouver des défauts**
  - Acquérir de la confiance sur le niveau de qualité
  - Fournir de l'information utile aux prises de décision
  - Prévenir des défauts

1.3 Les 7 Principes Généraux des Tests





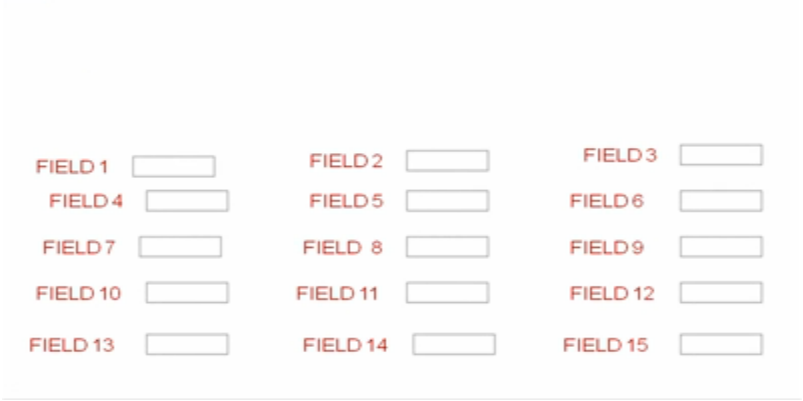
1.3 Les 7 Principes Généraux des Tests

- Essayer de transférer le fichier quand il est ouvert
- On a pas le droit d'écriture dans le dossier B
- Le dossier B est en partage et sa capacité limite est atteinte
- Le dossier B a un fichier qui de même nom

1.3 Les 7 Principes Généraux des Tests

- Supposons qu'on a 15 cases à tester, chaques cases peut prender 5 valeurs possibles. Le nombre de combinaison de test est 5^15= 30517578125

- Si on test toutes les combinaisons possible➔ Temps + couts explosent



- **<u>Les tests exhaustifs sont impossibles</u>**

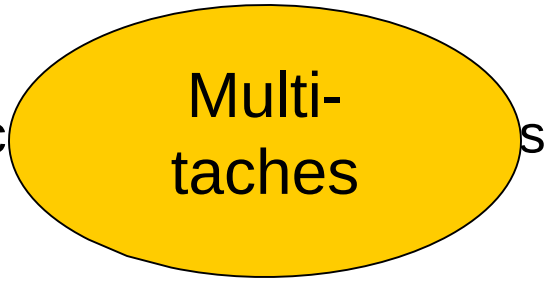
- On aura donc besoin d'un échantillon de test optimale en se basant sur le risque de l'application

1.3 Les 7 Principes Généraux des Tests

- Quelle opération est susceptible la plus de causer une defaillance dans votre application:

  a) Ouvrir Microsoft world

  b) Ouvrir internet explorer

  c) Ouvrir 10 applic 

s

- **Regroupement des défauts** : Un petit nombre de modules contiennent généralement la majorité des défauts détectés

1.3 Les 7 Principes Généraux des Tests

- Si les mêmes tests sont répétés de nombreuses fois, il arrivera que le même ensemble de cas de tests ne trouvera plus de nouveaux défauts

- C’est le principe du **Paradoxe du pesticide**

- Pour palier ce problème, les cas de test devraient régulièrement être revus, ajout de nouveaux cas de test afin de détecter de nouveaux bug

1.3 Les 7 Principes Généraux des Tests

» On ne peut jamais certifier qu'un logiciel ne contient aucun BUG

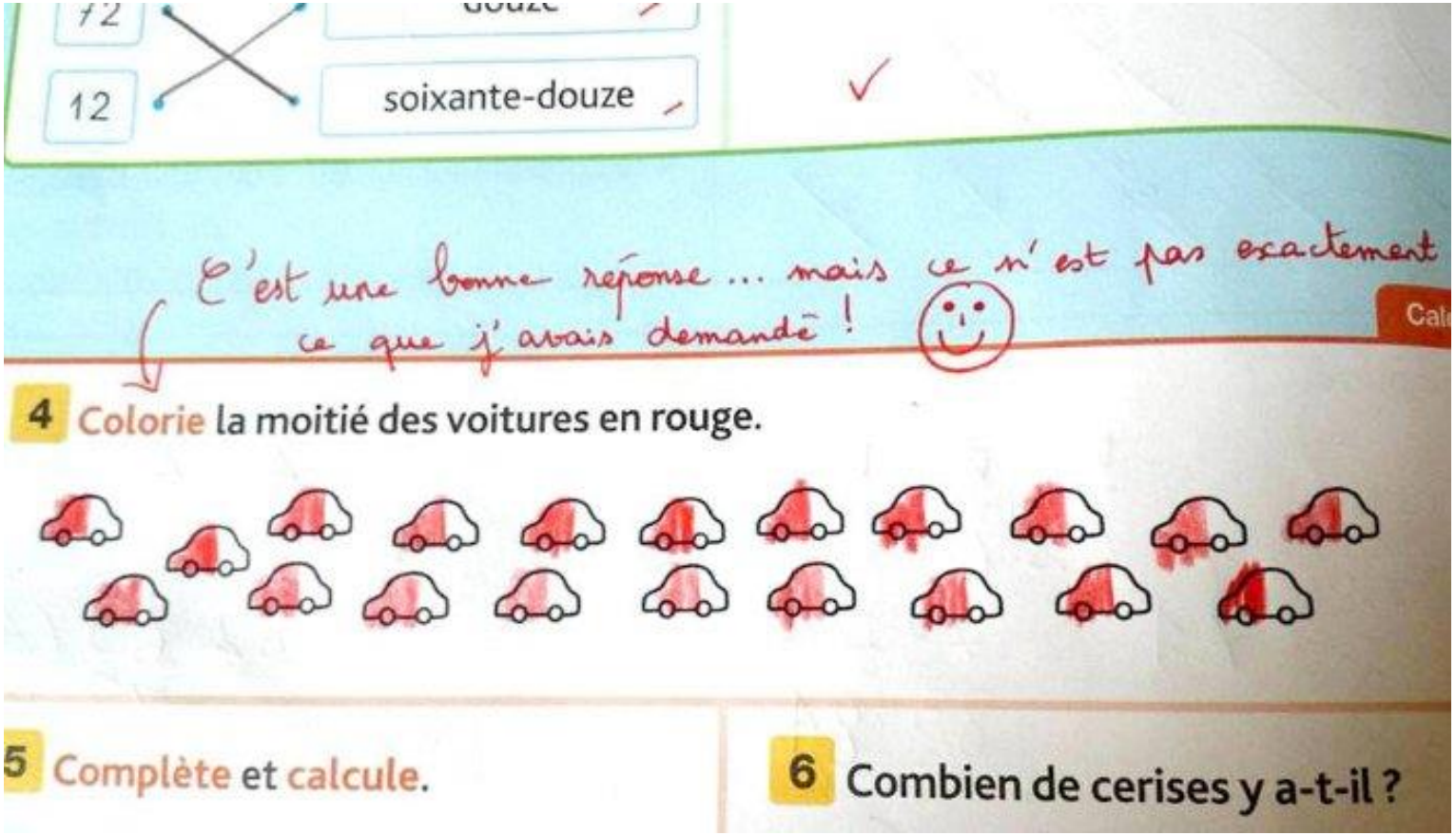
Bill Gates - Microsoft Window 98 crash on live TV.mp4

» http://www.youtube.com/watch?v=f-1TOeHY7as

» **Les tests montrent la présence de défauts**: les tests réduisent la probabilité de trouver des défaillances dans le logiciel mais même si aucun BUG n'est trouvé, ce n'est pas autant une preuve pour ne pas en contenir

1.3 Les 7 Principes Généraux des Tests

» Dessine 20 voitures et colorie la moitié des voitures rouges



1.3 Les 7 Principes Généraux des Tests

» Si le logiciel est certifié qu'à 99% ne contient pas de BUG, mais ce logiciel ne satisfait pas le besoin client

» **L'illusion de l'absence d'erreurs** : Trouver et corriger des défauts n'aide pas si le système conçu est inutilisable et ne comble pas les besoins et les attentes des utilisateurs

1.3 Les 7 Principes Généraux des Tests

- Pour régler ce problème on a le principe: **Tester tôt**

- Les tests devraient commencer le plutôt possible dans le cycle de développement

- **Les tests dépendent du contexte** : exemple d' un site de commerce électronique

1.3 Les 7 Principes Généraux des Tests

- Principe 1 – Les tests montrent la présence de défauts
- Principe 2 – Les tests exhaustifs sont impossibles
- Principe 3 – Tester tôt
- Principe 4 – Regroupement des défauts : Un petit nombre de modules contiennent généralement la majorité des défauts détectés.
- Principe 5 – Paradoxe du pesticide : Si les mêmes tests sont répétés de nombreuses fois, il arrivera que le même ensemble de cas de tests ne trouvera plus de nouveaux défauts
- Principe 6 – Les tests dépendent du contexte : exemple d' un site de commerce électronique.
- Principe 7 – L'illusion de l'absence d'erreurs : Trouver et corriger des défauts n'aide pas si le système conçu est inutilisable et ne comble pas les besoins et les attentes des utilisateurs.

1.4 Processus de Test Fondamental

1.4 Processus de Test Fondamental

» **Planification des Tests et Contrôle :**

– La planification des tests consiste à :

- définir les objectifs et les risques du projet
- Détermine l'approche de test
- Détermine les ressource nécessaires (humaines; matériels (PC), environnent de test …)
- Détermine les critères de sorties

– Le contrôle des tests est une activité continue de comparaison de l'avancement actuel par rapport au plan, et d'information sur l'état, y compris les déviations par rapport au plan

1.4 Processus de Test Fondamental

» **Analyse et conception des tests :**

– L'analyse et la conception des tests représentent les activités où les objectifs de test généraux sont transformés en des conditions de test et des conceptions de test tangibles.

– L'analyse et la conception des tests se composent des tâches majeures suivantes:

- Réviser les bases du test (telles que les exigences, le niveau d'intégrité logiciel (cad niveau de risque), les rapports d'analyse de risque, l'architecture, la conception et les interfaces).

- Evaluer la testabilité des exigences et du système.

1.4 Processus de Test Fondamental

- Identifier et prioriser les conditions de test sur la base de l'analyse des articles de test, la spécification, le comportement et la structure du logiciel.
- Concevoir et prioriser les tests de haut niveau
- Identifier les données de test nécessaires pour les conditions de test et les cas de test
- Concevoir l'initialisation de l'environnement de test et identifier les infrastructures et outils requis
- Créer une traçabilité bidirectionnelle entre les bases de test et les cas de test

1.4 Processus de Test Fondamental

» **Implémentation et exécution des tests :**

– L'implémentation et l'exécution des tests se composent des tâches majeures suivantes:

- Finaliser, développer et prioriser les cas de test (y compris l'identification des données de test).
- Développer et prioriser les procédures de test, créer les données de test et éventuellement préparer les harnais de test et écrire les scripts de tests automatiques.
- Créer des suites de tests à partir des procédures de test pour une exécution rentable des tests.
- Vérifier que les environnements de tests ont été mis en place correctement.

1.4 Processus de Test Fondamental

- Vérifier et mettre à jour la traçabilité bidirectionnelle entre les bases de test et les cas de test.
- Exécuter les procédures de test soit manuellement soit en utilisant des outils d'exécution de tests, en suivant la séquence planifiée.
- Consigner les résultats de l'exécution des tests et enregistrer les identités et versions des logiciels en test, outils de test et testware
- Comparer les résultats actuels et les résultats attendus.

1.4 Processus de Test Fondamental

- Signaler les divergences comme des incidents et les analyser de façon à établir leur cause (p.ex. défaut dans le code, dans les données de test, dans la documentation de test, ou méprise dans la manière d'exécuter le test)

- Répéter les activités de test en réponse aux actions prises pour chaque divergence. Par exemple, réexécution d'un test qui était préalablement défaillant de façon à valider une correction (test de confirmation), exécution d'un test corrigé et/ou exécution de tests de façon à s'assurer que des défauts n'ont pas été introduits dans des secteurs non modifiés du logiciel ou que le défaut corrigé n'a pas découvert d'autres défauts (test de régression)

1.4 Processus de Test Fondamental

» **Evaluer les critères de sortie et informer :**

– Evaluer les critères de sortie est l'activité où l'exécution des tests est évaluée en fonction des objectifs définis. Ceci devrait être fait pour chacun des niveaux de test

– Evaluer les critères de sortie contient les tâches majeures suivantes:

- Vérifier les registres de tests en fonction des critères de sortie spécifiés dans la planification des tests
- Evaluer si des tests supplémentaires sont requis ou si les critères de sortie doivent être changés
- Ecrire un rapport de synthèse des tests pour les parties prenantes

1.4 Processus de Test Fondamental

» **Activités de clôture des tests :**

– Les activités de clôture des tests incluent les tâches majeures suivantes:

- Vérifier quels livrables prévus ont été livrés
- Clôturer les rapports d'incidents ou créer des demandes d'évolution pour ceux restant ouverts
- Documenter l'acceptation du système
- Finaliser et archiver les testwares, environnements de test et infrastructures de test pour une réutilisation future.
- Fournir les testwares à l'organisation en charge de la maintenance

1.4 Processus de Test Fondamental

- Analyser les leçons apprises pour identifier les changements nécessaires pour les versions et projets futurs
- Utiliser l'information collectée pour améliorer la maturité des tests

1.5 La Psychologie des Tests

- Un certain degré d'indépendance (évitant le parti-pris de l'auteur) est souvent plus efficace pour détecter des défauts et des défaillances.

- Plusieurs niveaux d'indépendance peuvent être définis, comme les niveaux suivants présentés du plus faible au plus élevé :

  - Tests conçus par la (les) personne(s) qui a (ont) écrit le logiciel à tester (niveau faible d'indépendance).

  - Tests conçus par une (des) autre(s) personne(s) (p.ex. de l'équipe de développement).

  - Tests conçus par une (des) personne(s) d'un groupe différent au sein de la même organisation (p.ex. équipe de test indépendante) ou par des spécialistes de test (p.ex. spécialistes en tests de performance ou utilisabilité)

1.5 La Psychologie des Tests

- Tests conçus par une (des) personne(s) d'une organisation ou société différente (p.ex. sous-traitance ou certification par un organisme externe)

» Il existe plusieurs manières d'améliorer la communication et les relations entre les testeurs et leurs interlocuteurs :

- Commencer par une collaboration plutôt que par des conflits – rappeler à chacun l'objectif commun de systèmes de meilleure qualité

- Communiquer les découvertes sur le produit de façon neutre et factuelle sans critiquer la personne responsable, par exemple, écrire des rapports d'incidents (ou des résultats de revues) objectifs et factuels.

1.5 La Psychologie des Tests

- Essayer de comprendre ce que ressent une autre personne et pourquoi elle réagit comme elle le fait.
- Confirmer que l'autre personne a compris ce que l'on a dit et vice versa.

1.6 Code d'éthique

- En référence au code d'éthique D'ACM et de l'IEEE pour les ingénieurs, l'ISTQB définit le code d'éthique suivant :
- PUBLIC – les testeurs de logiciels certifiés doivent agir en fonction de l'intérêt public
- CLIENT ET EMPLOYEUR – les testeurs de logiciels certifiés doivent agir pour l'intérêt de leur client et de leur employeur tout en respectant l'intérêt public
- PRODUIT – les testeurs de logiciels certifiés doivent assurer que les fournitures qu'ils produisent (concernant les produits et les systèmes qu'ils testent) répondent le plus possible aux standards professionnels
- JUGEMENT – les testeurs de logiciels certifiés doivent conserver leur intégrité et leur indépendance dans leur jugement professionnel

1.6 Code d’éthique

- GESTION – les chefs de projet de test de logiciels certifiés et les responsables doivent respecter et promouvoir une approche morale dans la gestion de projets de test de logiciels
- PROFESSION – les testeurs de logiciels certifiés doivent mettre en avant l’intégrité et la réputation du métier en cohérence avec l’intérêt public
- COLLEGUES – les testeurs de logiciels certifiés doivent être loyaux, aider leurs collègues, et promouvoir le partenariat avec les développeurs de logiciels
- PERSONNELLEMENT – les testeurs de logiciels certifiés doivent participer en permanence à de la formation pour leur métier et doivent promouvoir une approche morale concernant sa pratique.

Chapitre 2 : Tester Pendant le Cycle de Vie Logiciel

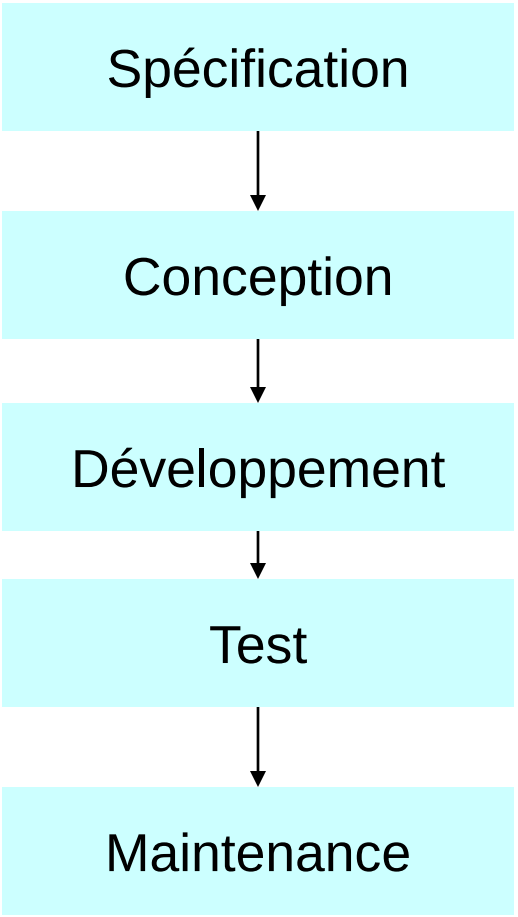
Chapitre 2 : Tester Pendant le Cycle de Vie Logiciel

- Modéles de développement logiciel
- Niveaux de tests
- Types de test

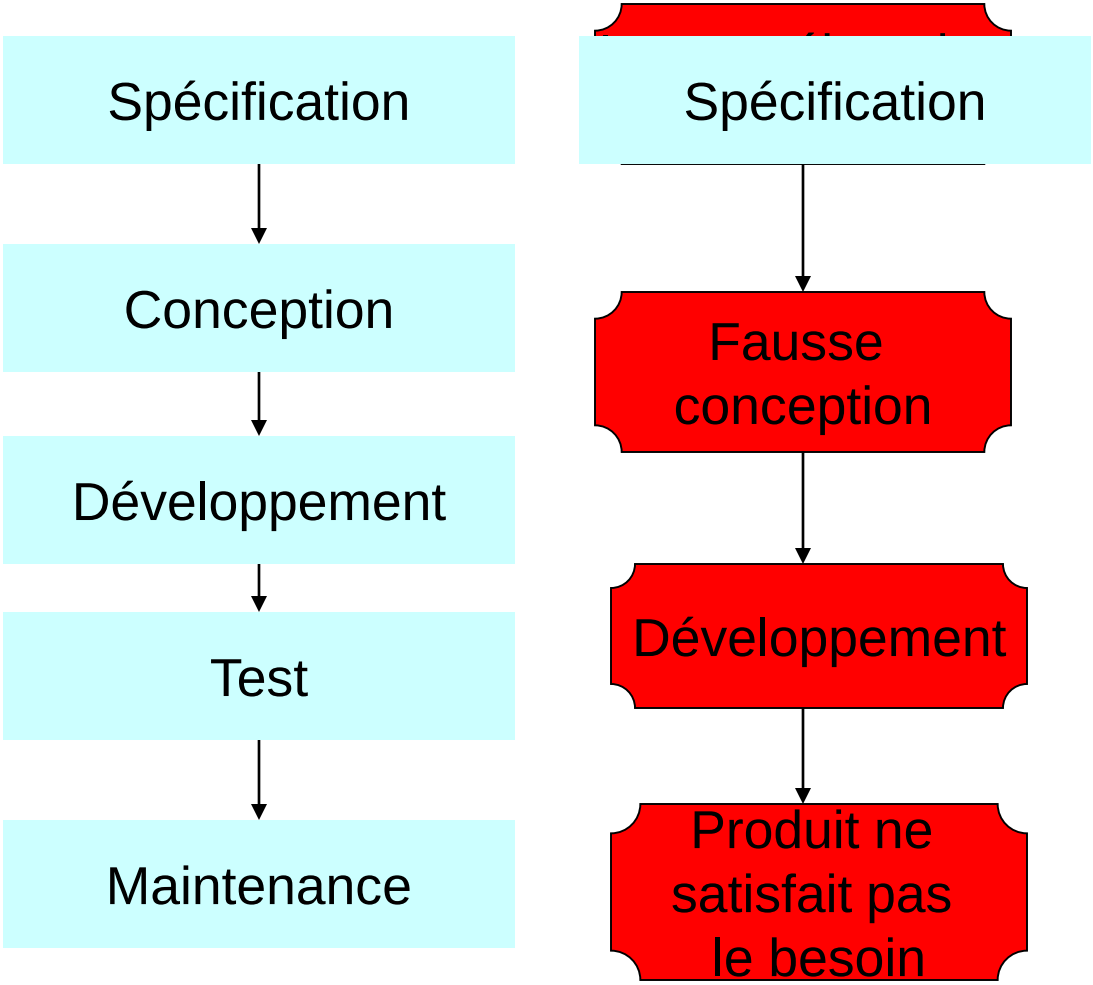
Chapitre 2 : Tester Pendant le Cycle de Vie Logiciel

2.1 Modèle en cascade

Cycle de vie logiciel en cascade

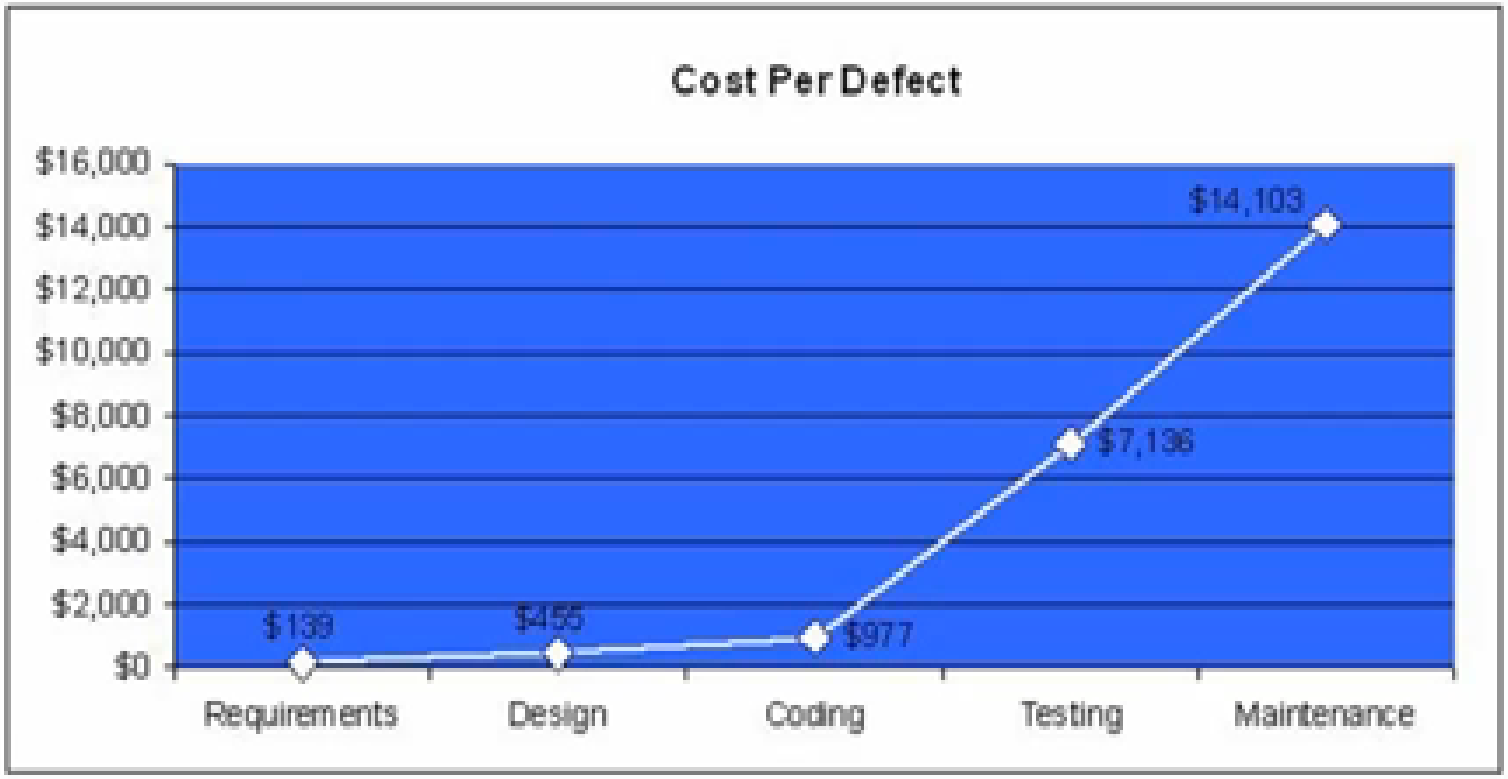


2.1 Modèle en cascade

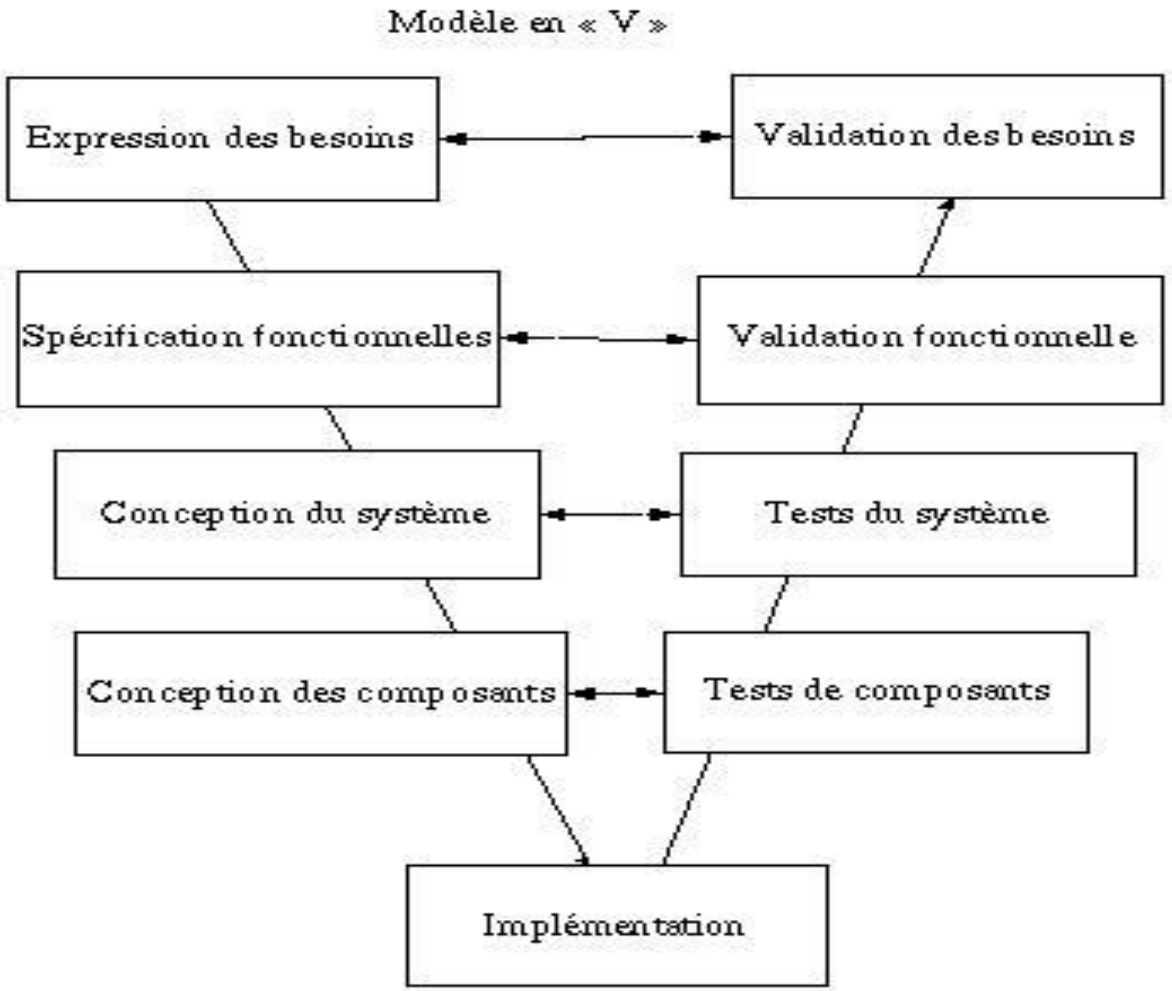


2.1 Modèle en cascade

50% des défaillance sont introduites lors de la phase de collecte des besoins client et de l'écriture des spécifications



2.1 Modèle en V



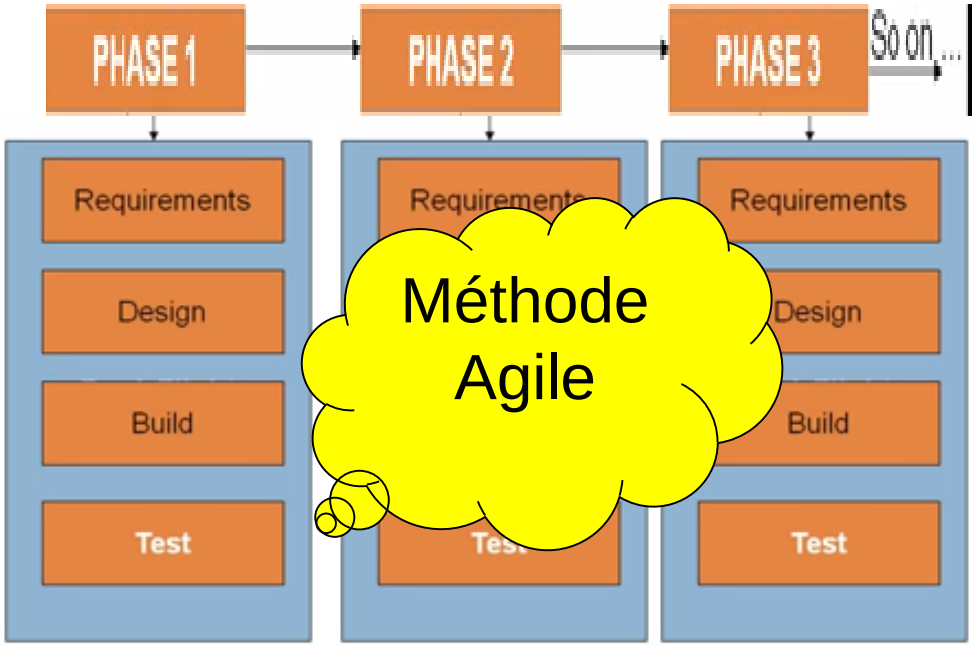
2.1 Modèles de Développement Logiciel

» **Modèle en V :**

– Un modèle en V standard utilise quatre niveaux de tests, correspondant aux quatre niveaux de développement (tests de composants (unitaires); tests d'intégration; tests système; tests d'acceptation.

– Les livrables logiciels produits pendant le développement sont souvent les bases des tests d'un ou plusieurs niveaux de tests. Des références pour des livrables génériques sont disponibles dans le modèle CMMI (Capability Maturity Model Integration) ou dans l"IEEE/IEC 12207 (Processus de cycle de vie logiciel, Software life cycle processes)

Modèle itératif



2.1 Modèles de Développement Logiciel

» **Modèle de développement itératif :**

– Le mode de développement itératif est une succession d'activités exécutées comme une série de petits développements: exigences, conception, construction et tests d'un système exple : RAD, RUP, méthodologies agiles.

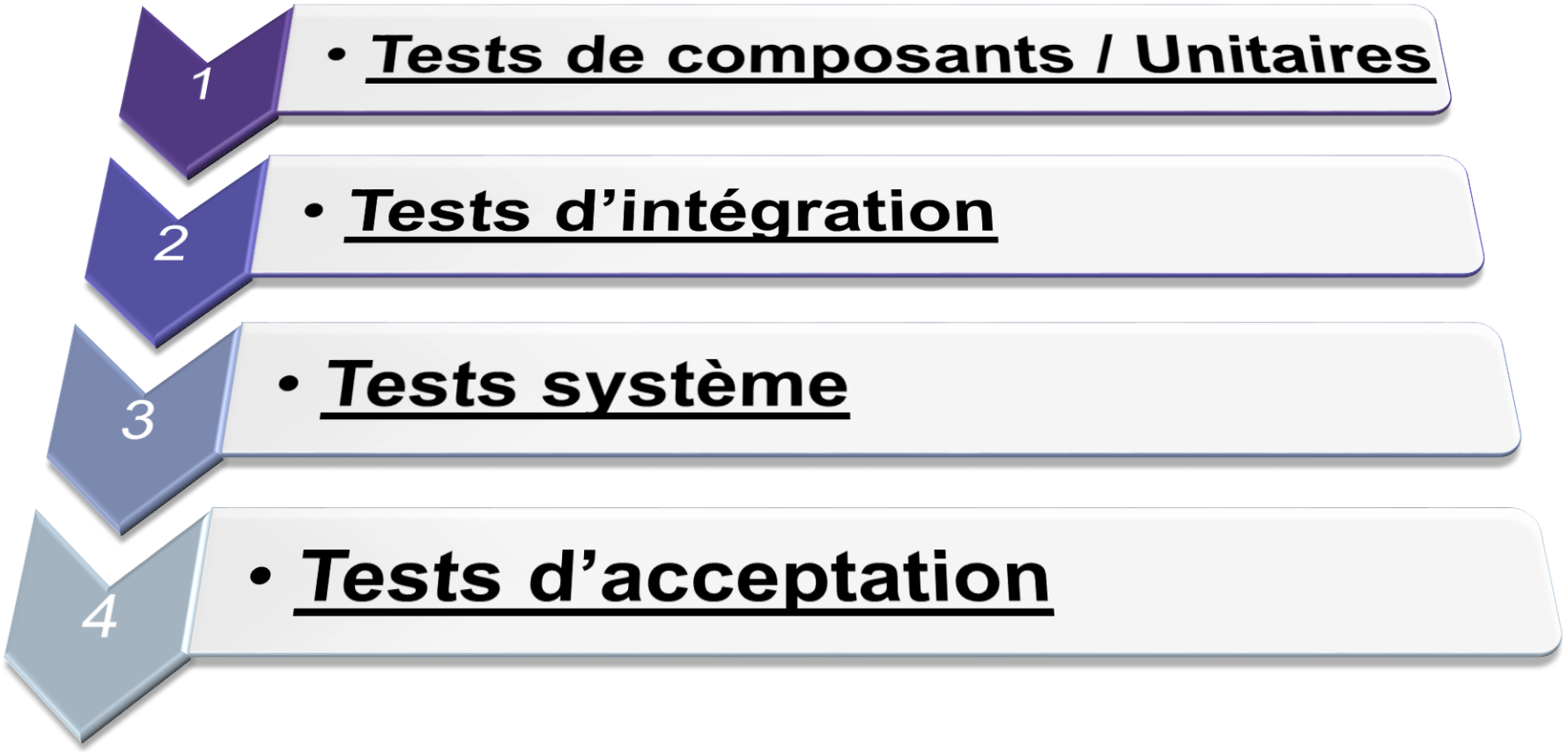
– Le système logiciel résultant d'une itération est appelé incrément.

2.1 Modèles de Développement Logiciel

» **Tester au sein d'un modèle de cycle de vie :**

- A chaque activité de développement, correspond une activité de test et des objectifs de test.
- L'analyse et la conception des tests pour un niveau de test devraient commencer pendant l'activité correspondante de développement.
- Les testeurs doivent être impliqués dans la revue des documents aussi tôt que des brouillons sont disponibles dans le cycle de développement.
- Les niveaux de tests peuvent être combinés ou réorganisés selon la nature du projet ou de l'architecture du système.

2.2 Niveaux de Tests



2.2 Niveaux de Tests

» **Tests de composants / Unitaires :**

– Bases de tests: Exigences des composants, conception détaillée, Code

– Objets habituels de test: Composants, programmes, conversions de données / utilitaires ou programmes de migration, modules de bases de données

– Les tests de composants peuvent inclure des tests de fonctionnalités et des tests de caractéristiques non-fonctionnelles, telles que le comportement des ressources (p.ex. fuites mémoire) ou des tests de robustesse, ainsi que des tests structurels (p.ex. couverture des branches).

2.2 Niveaux de Tests

- Les cas de test sont dérivés des livrables tels que les spécifications des composants (spécifications détaillées), la conception du logiciel ou le modèle de données.
- Ils sont faits dans un environnement de **développement.**
- Préparer et automatiser les cas de tests unitaires avant le développement s'appelle **tester d'abord**.

» **<u>Tests d'intégration :</u>**

- Test d'intégration des composants teste les interactions entre les composants logiciels (fait après test de composant)
- Test d'intégration système teste l'intéraction entre les différents systèmes (fait après test système)

2.2 Niveaux de Tests

- Les tests d'intégration testent les interfaces entre les composants, les interactions entre différentes parties d'un système comme par exemple le système d'exploitation, le système de fichiers, le matériel ou les interfaces entre les systèmes

- Afin d'isoler facilement les fautes, et détecter les défauts au plus tôt, l'intégration devrait normalement être incrémentale plutôt qu'être effectuée en une fois ("big bang")

2.2 Niveaux de Tests

» **Tests système :**

– Ce sont des tests qui traitent le comportement du système / produit complet dans un environnement de test qui devrait correspondre à la cible finale ou à un environnement de production de façon à minimiser les risques de défaillances dues à l'environnement..

– Ces tests sont opérés par une **équipe de test indépendante**, ils peuvent être aussi bien fonctionnels que non-fonctionnels.

– Ils peuvent aussi être des tests boîte blanche (basées sur les structures) ou boîte noire (basées sur les spécifications avec une table de décision par exemple).

2.2 Niveaux de Tests

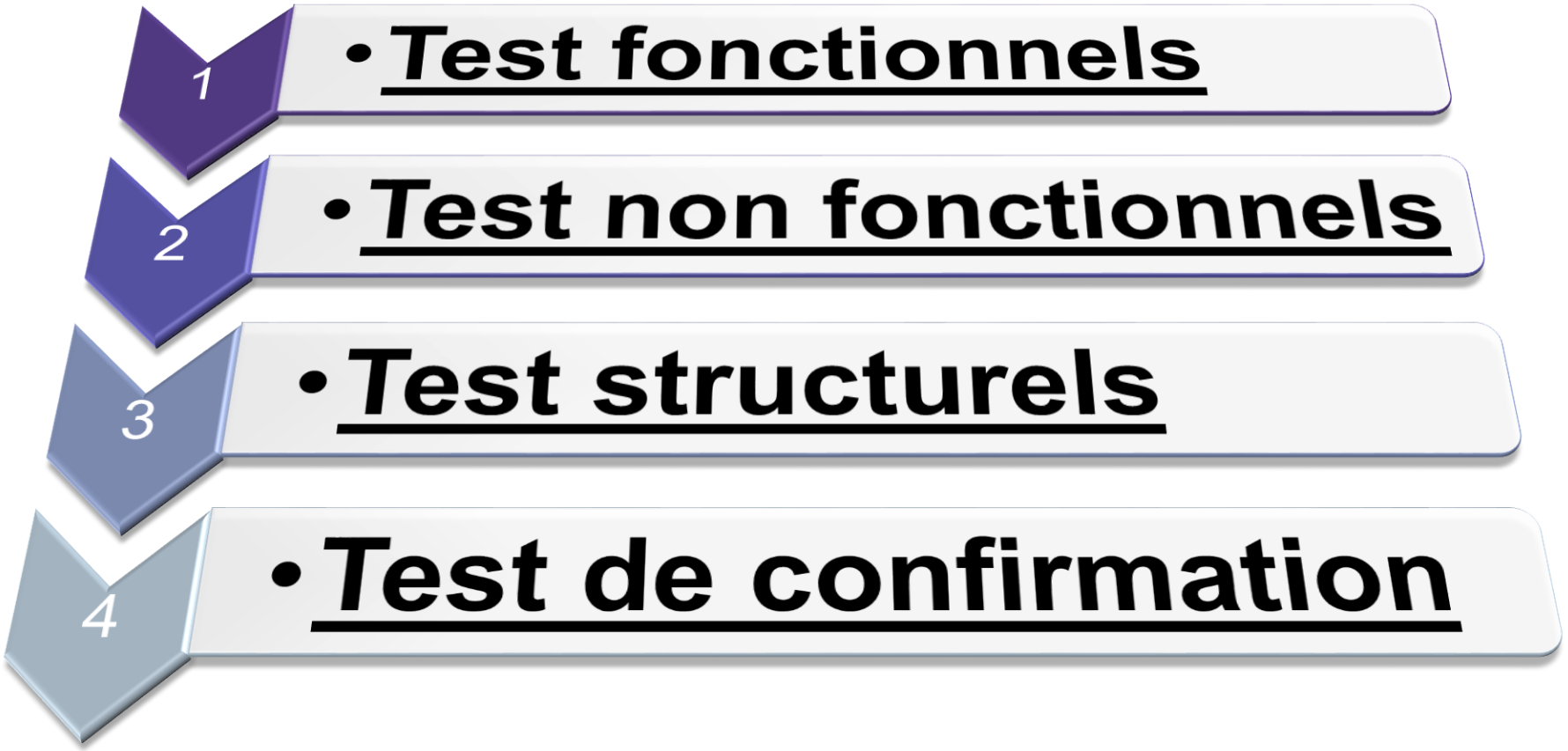
» **Tests d'acceptation :**

- C'est la responsabilité du client finale ou the user finaux du systèmes
- L'objectif est d'établir un niveau de confiance par rapport au logiciel
- La recherche d'anomalies n'est pas l'objectif principal des tests d'acceptation.
- Les tests d'acceptation peuvent évaluer si le système est prêt à être déployé et utilisé, bien que ce ne soit pas nécessairement le dernier niveau ; par exemple, une intégration système à grande échelle peut arriver après les tests d'acceptation du système.

2.2 Niveaux de Tests

- Les formes habituelles des tests d'acceptation incluent :
  - *Tests d'acceptation utilisateur* qui vérifient l'aptitude et l'utilisabilité du système par des utilisateurs.
  - *Tests (d'acceptation) opérationnelle* qui englobe les tests des backups et restaurations; la reprise après sinistre; la gestion des utilisateurs; les tâches de maintenance; les chargements de données et tâches de migration; la vérification périodique des vulnérabilités de sécurité.
  - *Tests d'acceptation contractuelle et réglementaire*
  - **Les Alpha tests** sont exécutés sur le site de l'organisation effectuant le développement mais pas par les équipes de développement. **Les Béta tests** ou tests sur le terrain sont exécutés par des personnes sur leurs sites propres.

2.3 Types de Tests



2.3 Types de Tests

» **Les Tests Fonctionnels :**

– Les tests fonctionnels concernent le comportement extérieur du logiciel (tests boîte noire) et peuvent être exécutés à tous les niveaux de tests.

– Un type de test fonctionnel, le test de **sécurité**, examine les fonctions (p.ex. pare-feu) liées à la détection de menaces, comme des virus, provenant de tiers malveillants. Un autre type de test fonctionnel, le test d'**interopérabilité**, évalue la capacité du logiciel à interagir avec un ou plusieurs composants ou systèmes spécifiés.

» **Tests Non-Fonctionnels :**

– Les tests non-fonctionnels incluent, mais pas uniquement, les tests de **performances**, tests de **charge** (grand nombre d'utilisateurs), tests de **stress**, tests **d'utilisabilité**, tests de **maintenabilité**, tests de **fiabilité** et les tests de **portabilité**.

2.3 Types de Tests

- Ces tests évaluent “comment” le système fonctionne
- Les tests non fonctionnels concernent l’aspect extérieur du logiciel et la plupart du temps utilisent les techniques de conception de tests boîte noire.
- Ces tests peuvent être référencés dans un modèle qualité tel que celui défini par l’ISO9126 Ingénierie Logicielle – Qualité des Produits Logiciels.
- Les tests non-fonctionnels peuvent être effectués à **tous les niveaux de tests**. Le terme de tests non-fonctionnels décrit les tests requis pour **mesurer** les caractéristiques des systèmes et logiciels.
- Les tests non-fonctionnels peuvent être effectués à **tous les niveaux de tests**. Le terme de tests non-fonctionnels décrit les tests requis pour **mesurer** les caractéristiques des systèmes et logiciels.

2.3 Types de Tests

» **Tests de la Structure / Architecture Logicielle (Tests Structurels) :**

– Les tests structurels (boîte blanche) peuvent être effectués à tous les niveaux de tests. Les techniques structurelles sont utilisées de façon optimale après les techniques basées sur les spécifications, pour aider à mesurer l'ampleur des tests via l'évaluation de la couverture d'un type de structure.

– A tous les niveaux de tests, mais spécialement dans les tests de composants et les tests d'intégration de composants, des outils peuvent être utilisés pour mesurer la couverture du code

– Si la couverture n'est pas de 100%, alors de nouveaux tests peuvent être conçus pour tester les éléments manquants et ainsi augmenter la couverture (voir chapitre 4 pour la couverture).

2.3 Types de Tests

» **Tests de Confirmation et de Régression :**

– Quand un défaut est détecté et corrigé, le logiciel devrait être re-testé pour confirmer que le défaut original a été correctement ôté. Ceci est appelé **test de confirmation**

– Ré-exécuter des tests sur un programme déjà testé s'appelle « **test de régression** ». Cela se fait après que des modifications du programme aient eu lieu.

– Les tests de régression peuvent être exécutés à tous les niveaux de tests, et **s'appliquent aux tests fonctionnels, non-fonctionnels et structurels** ; ce sont les bons candidats à l'automatisation.

2.3 Types de Tests

» **Tests de Maintenance :**

– Les tests de maintenance sont effectués sur un système opérationnel existant et sont déclenchés par des modifications, migrations ou suppression de logiciels ou de systèmes.

– Les modifications incluent les changements dûs aux évolutions planifiées (p.ex. livraisons de nouvelles versions), aux modifications correctives et d'urgence, ainsi qu'aux changements d'environnements tels que les montées en version planifiées des systèmes d'exploitation, des bases de données ou des COTS. Elles incluent également les patchs de correction des vulnérabilités de sécurité potentielles ou découvertes d'un système d'exploitation.

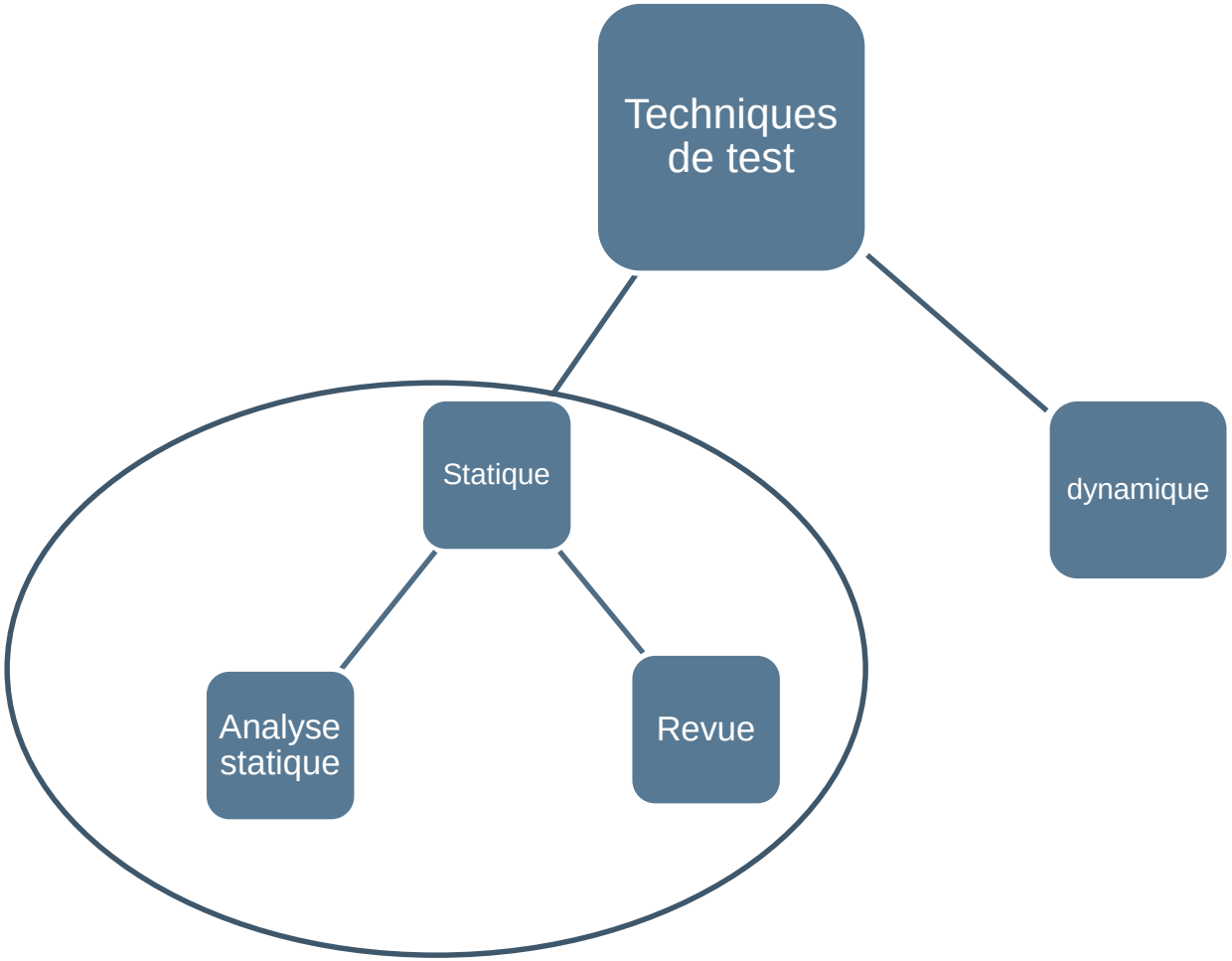
– Selon le changement, les tests de maintenance peuvent être effectués à chacun ou à tous les niveaux de tests et pour certains ou tous les types de tests (selon l'analyse d'impact)

Chapitre 3 : Techniques Statiques

Chapitre 3 : Techniques Statiques

- Techniques statiques et processus de test
- Process de revues
- Analyse statique outillée

3.1 Techniques statiques et processus de test



3.1 Techniques statiques et processus de test

- Les techniques de tests statiques reposent sur **l'examen manuel** (revues) ou **l'analyse du code** (analyse statique) ou de **la documentation** du projet sans exécution du code.

- Tout produit logiciel peut être revu, y compris les exigences, les spécifications, les spécifications de conception, le code, les plans de test, les spécifications de test, les cas de test, les scripts de test, les guides utilisateur ou pages web.

- Les revues permettent une détection et une correction anticipées des défauts.

- Une revue pourrait être effectuée entièrement manuellement ou à l'aide des outils de support.

- Les revues, les analyses statiques et les tests dynamiques:

  - Ont le même objectif : identifier des défauts.

  - Sont complémentaires : les techniques statiques trouvent **les causes** des défauts, les tests dynamiques trouvent **les défaillances** elles mêmes.

3.2 Processus de revue

- **Phases d'une revue formelle**
- Planification :
  - Définir les critères de revues, critères d'entrée et de sortie pour des types de revues plus formels et sélectionner la partie des documents à revoir.
  - Choisir le personnel et allouer les rôles.
- Lancement:
  - Distribuer les documents et expliquer les objectifs, le processus aux participants.
- Préparation individuelle.
  - Préparer la réunion de revue en revoyant les documents, écriture des défauts potentiels, questions et commentaires.

3.2 Processus de revue

- Examen/évaluation/enregistrement des résultats (réunion de revue):
  - Discuter ou enregistrer, avec des résultats ou minutes documentés (pour des types de revues plus formels).
  - Noter les défauts, faire des recommandations concernant le traitement des défauts, prendre des décisions à propos des défauts.
- Re travail
  - Correction des défauts détectés (réalisé généralement par l'auteur).
  - Enregistrer le statut modifié des défauts (dans les revues formelles).

3.2 Processus de revue

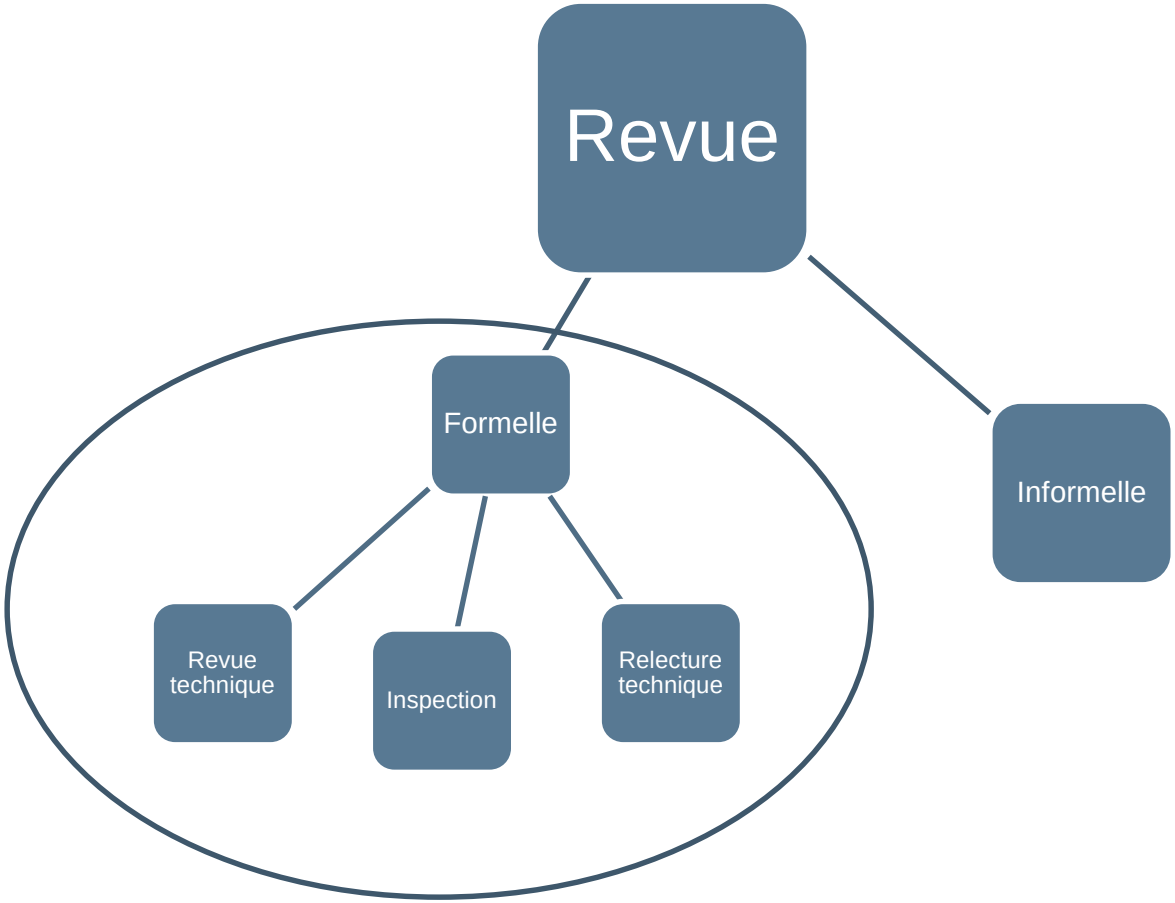
- Suivi:
  - Vérifier que les défauts ont bien été traités.
  - Récolter les métriques.
  - Contrôle sur la base des critères de sorties (pour des types de revues plus formels).

3.2 Processus de revue

» Rôles et responsabilités

- Manager : décide l'exécution des revues, alloue le temps dans la planification du projet et détermine si les objectifs de revue ont été atteints.
- Modérateur: la personne qui dirige la revue des documents, incluant la planification et l'exécution de la revue, et le suivi post-réunion.
- Auteur : l'auteur ou la personne à qui incombe la responsabilité principale des documents à revoir.
- Réviseurs : les individus avec une culture technique ou métier spécifique qui identifient et décrivent les constatations dans le produit en cours de revue.
- Scribe (ou greffier): documente tous les aspects, problèmes et points ouverts identifiés pendant la réunion.

3.2 Types de revues



3.2 Processus de revue

» Types de revues

– Revue informelle

- Pas de processus formel.
- Peut inclure la programmation par paires ou une revue de conception et de code par un responsable technique.
- Les résultats peuvent être documentés.
- Peut varier en utilité selon les réviseurs.
- Objectif principal : manière bon marché d'obtenir des résultats.

3.2 Processus de revue

- Relecture technique
  - Réunion dirigée par l'auteur.
  - Peut prendre la forme de scénarios, répétitions à blanc, participation de groupes de pairs.
  - Sessions sans limite de durée
  - Optionnellement un scribe (qui n'est pas l'auteur).
  - Objectifs principaux: apprendre, gagner en compréhension, trouver des défauts.

3.2 Processus de revue

- Revue technique
  - Documentée, processus de détection de défauts défini incluant des pairs et des experts techniques avec optionnellement la participation de l'encadrement.
  - Idéalement dirigée par un modérateur formé (pas l'auteur).
  - Réunion de préparation par les réviseurs.
  - Peut optionnellement utiliser des check-lists.
  - Préparation d'un rapport de revue.
  - Objectifs principaux: discuter, décider, évaluer des alternatives, trouver des défauts, résoudre des problèmes techniques et vérifier la conformité aux spécifications, plans, réglementations et standards.

3.2 Processus de revue

- Inspection
  - Dirigée par un modérateur formé (pas l'auteur).
  - Rôles définis (Lecteur facultatif), critères d'entrée et de sortie spécifiés pour l'acceptation du produit logiciel.
  - Inclut des métriques.
  - Processus formel basé sur des règles et des check-lists.
  - Réunion de préparation.
  - Rapport d'inspection incluant la liste de constatations.
  - Processus formel de suivi.
  - Objectif principal: trouver des défauts.

3.2 Processus de revue

» Facteurs de succès des revues

- Revues avec des objectifs prédéfinis et clairs.
- Les personnes impliquées sont adéquates pour les objectifs de la revue.
- Les testeurs sont des réviseurs de valeur qui contribuent à la revue et ainsi prennent connaissance du produit afin de pouvoir préparer les tests plus tôt.
- Les résultats de la revue ne sont pas utilisés pour évaluer les participants.
- Les techniques de revue adaptées aux objectifs, livrable logiciel, et réviseurs.
- Des check-lists ou des rôles sont utilisés lorsque cela est approprié.
- Des formations sont données sur les techniques de revue, en particulier celles concernant les techniques plus formelles telles que les inspections.

3.3 Analyse statique outillée

- L’accent est mis sur l’apprentissage et l’amélioration du processus.
- L'objectif de l'analyse statique est de trouver des défauts dans le code source et les modèles logiciels.
- Les outils d'analyse statique analysent le code du programme et sont typiquement utilisés par des développeurs avant et pendant les tests de composants et les tests d'intégration.
- Les outils d'analyse statique peuvent produire un nombre important de messages d'avertissement qui doivent être gérés convenablement afin de permettre une utilisation optimale de l'outil.

3.3 Analyse statique outillée

» La valeur des tests statiques est :

- La détection très tôt de défauts avant l'exécution des tests.
- Une information très tôt sur certains aspects suspects du code ou de la conception, par le calcul de métriques, par exemple une mesure de complexité élevée.
- L'identification de défauts difficilement détectables par des tests dynamiques.
- La détection de dépendances et d'inconsistances dans les modèles logiciels tels que des liens dans les modèles logiciels.
- L'amélioration de la maintenabilité du code et de la conception.
- La prévention des défauts, si les leçons sont prises en compte lors du développement.

3.3 Analyse statique outillée

» Les défauts typiques découverts par des outils d'analyse statique incluent :

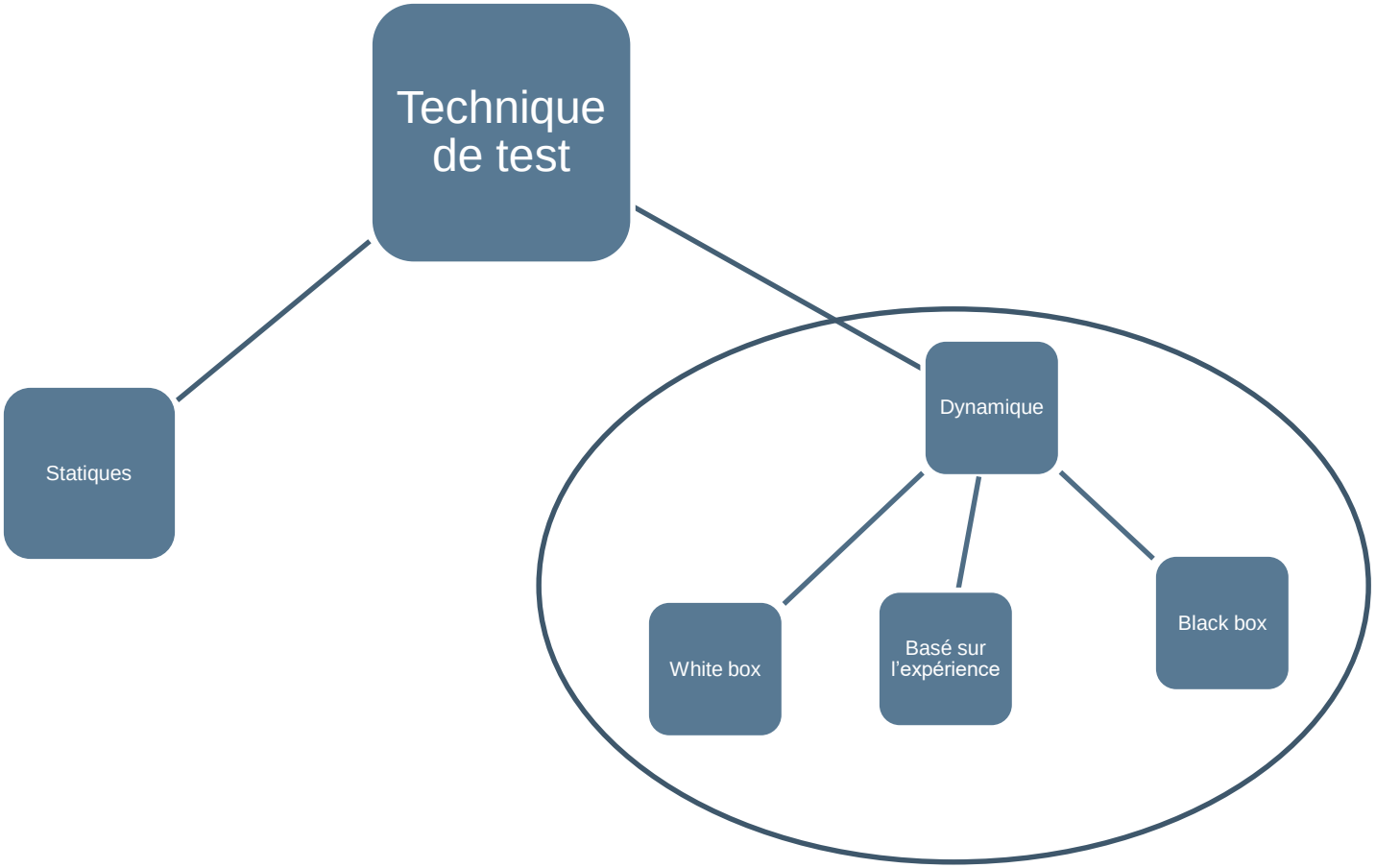
- Référencement d'une variable avec une valeur indéfinie.
- Interface inconsistante entre modules et composants.
- Variables qui ne sont jamais utilisées ou déclarées de façon incorrecte.
- Code non accessible (code mort).
- Logique absente et erronée (potentiellement des boucles infinies).
- Constructions trop compliquées.
- Violation des standards de programmation.
- Vulnérabilités de sécurité.
- Violation de syntaxe dans le code et les modèles logiciels.

Chapitre 4 : Techniques de Conception de Tests
Techniques de Conception de Tests

Chapitre 4 : Techniques de Conception de Tests

- Technique Boite noire
- Technique boite blanche
- Technique basée sur l’expéraince

4.1 Techniques de Conception de Tests



4.2 Catégories de techniques de conception de tests

- Statique (pas d'éxecution)
  - Examination des documents,source code …

- Fonctionnelle (black box)
  - Basée sur le comportement

- Structurelle (white box)
  - Basée sur la structure

4.2 Catégories de techniques de conception de tests

- Les techniques de conception **boite noire** (techniques basées sur les spécifications) sont une façon de dériver et de sélectionner les **conditions de test**, les **cas de test** où les **données de test**.

- Les techniques de conception **boite noire** incluent les tests fonctionnels et non fonctionnels.

- Les techniques basées sur les spécifications n'utilise aucune information concernant la structure interne d 'un composant ou système à tester.

- Les techniques de conception **boite blanche** (techniques basées sur les structures) sont basées sur une analyse de la structure d'un composant ou du système.

- Ces deux techniques peuvent être combinées avec des techniques basées sur **l'expérience** pour déterminer ce qui dit être testé.

4.2 Catégories de techniques de conception de tests

- Caractéristiques des techniques de conception basées sur les spécifications:
  - Les modèles, soit formels soit informels sont utilisés pour la spécification des problèmes à résoudre, des logiciels ou des composants.
  - Depuis ces modèles les cas de test sont dérivés de façon systématique.
- Caractéristiques des techniques de conception basées sur la structure:
  - L'information sur la manière dont le logiciel est construit est utilisée pour dériver les cas de test.
  - Le niveau de couverture du logiciel peut être mesuré à partir des cas de test existants.
  - Des cas de test complémentaires peuvent être dérivés systématiquement pour augmenter la couverture.

4.2 Catégories de techniques de conception de tests

- Caractéristiques des techniques basées sur l'expérience:
  - La connaissance et l'expérience des personnes sont utilisées pour dériver les cas de test.
  - La connaissance des testeurs, des développeurs, utilisateurs et autres parties prenantes concernant le logiciel, son utilisation et son environnement sont autant de sources d'information.
  - La connaissance des défauts possibles et de leur distribution est une autre source d'information.

4.3 Techniques basées sur les spécifications ou boîte noire

» **Partitions d'équivalence:**

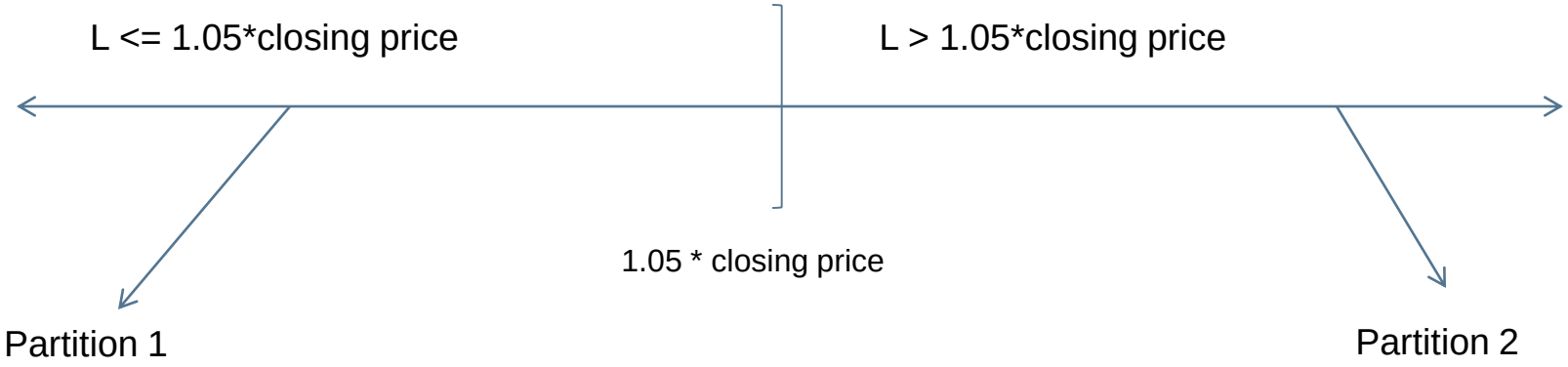
- Les entrées d'un logiciel sont divisées en groupes qui doivent monter un comportement similaire et un traitement identique.
- Les partitions peuvent être identifiées pour les sorties, les valeurs internes, les valeurs liées au temps et pour les paramètres d'interface.
- Des tests peuvent être conçus pour couvrir toutes les partitions.
- Les partitions sont applicables à tous les niveaux de tests.

4.3 Techniques basées sur les spécifications ou boîte noire

» Closing price filter 5%

– If Limit <= 1.05 * closing price , l'ordre est acquité

– If Limit > 1.05 * closing proce , L'ordre est rejeté



Excercice

Q. 358: A program validates a numeric field as follows:

Values less than 10 are rejected, values between 10 and 21 are accepted, values greater than or equal to 22 are rejected. Which of the following input values cover all of the equivalence partitions?

A. 10,11,21
B. 3,20,21
C. 3,10,22
D. 10,21,22

4.3 Techniques basées sur les spécifications ou boîte noire

- **Analyse des valeurs limites**

- Les valeurs **min** et **max** d'une partition sont ses **valeurs limites**.

- Des tests peuvent être conçus pour couvrir les valeurs limites valides et invalides.

- Une valeur de chaque limite est sélectionnée pour un test.

- L'analyse de ces valeurs peut être appliquée à tous les niveaux de test.

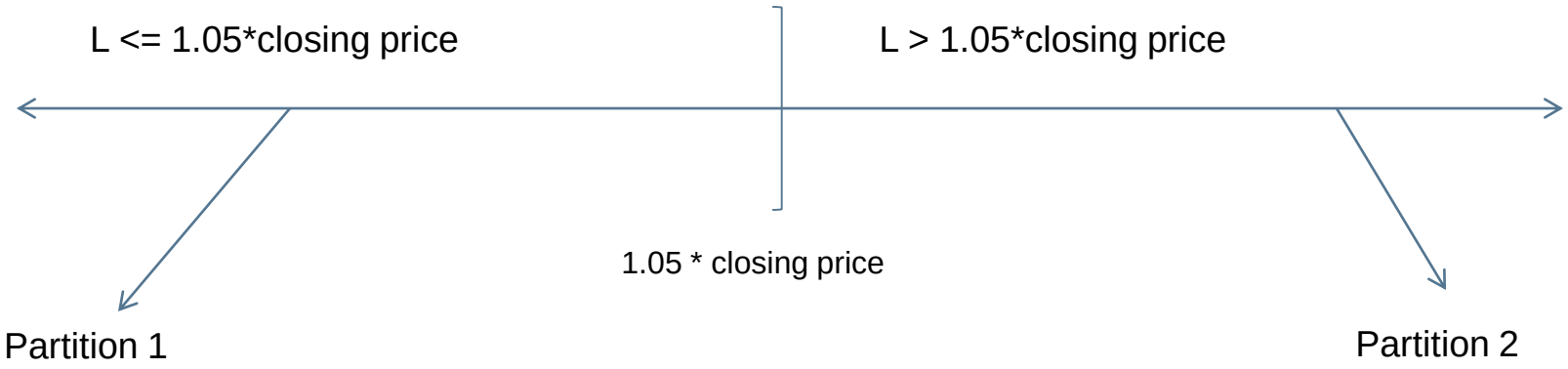
- C'est une technique complémentaire à celle des partitions d'équivalence.

4.3 Techniques basées sur les spécifications ou boîte noire

» Closing price filter 5%

– If Limit <= 1.05 * closing price , l'ordre est acquité

– If Limit > 1.05 * closing proce , L'ordre est rejeté



BV1= 1.05*closing price

BV2= 1.05*closing price -tick

4.3 Techniques basées sur les spécifications ou boîte noire

» **Exercice**

Invalid partition	Valid (for 3% interest)		Valid (for 5%)		Valid (for 7%)
–$0.01	$0.00	$100.00	$100.01	$999.99	$1000.00

Test conditions	Valid partitions	Invalid partitions	Valid boundaries	Invalid boundaries
Balance in aU$_o$Unt	$0.00 $100.00	< $0.00	$0.00	-$0.01
	$100.01-$999.99	>$Max	$100.00	$Max+0.01
	$1000.00- $Max	non-integer (if balance is an input field)	$100.01	
			$999.99	
			$1000.00	
			$Max	

Exercice

Q. 345: A program validates a numeric field as follows:

Values less than 10 are rejected, values between 10 and 21 are accepted, values greater than or equal to 22 are rejected. Which of the following covers the MOST boundary values?

A. 9,10,11,22
B. 9,10,21,22
C. 10,11,21,22
D. 10,11,20,21

4.3 Techniques basées sur les spécifications ou boîte noire

» Tests par tables de décision

- Les conditions d'entrée et les actions sont souvent décrites de façon à ce qu'elles peuvent être soit **vraies** soit **fausses** (Booléen).

- Les tables de décision contiennent les **conditions de déclenchement**, souvent des combinaisons de vrais et faux pour toutes les conditions d'entrée, et sur les actions résultantes pour chaque combinaison de conditions.

- Chaque colonne de la table correspond à une règle de gestion qui définit une combinaison unique de conditions qui résultent en l'exécution de l'action associée à cette règle.

4.3 Techniques basées sur les spécifications ou boîte noire

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Modality	Marker	Market	Any Price	Any Price
Validity	FOK	E&E	FOK	E&E
Actions				
Results	Y	Y	N	N

Excercice

	Rule 1	Rule 2	Rule 3	Rule 4
Condition				
Age	<21yrs	21-29 yrs	30-50yrs	>50yrs
Insurance class	A	A or B	B,C or D	C or D
Actions				
Premieum	100	90	70	70
Excess	2500	2500	500	1000

A. 23 year old in insurance class A Premium is 100 and excess is,500.
B. 51 year old in insurance class C Premium is 70 and excess is 100.
C. 31 year old in insurance class B Premium is 70 and excess is 500.
D. 43 year old in insurance class C Premium is 70 and excess is 1000.

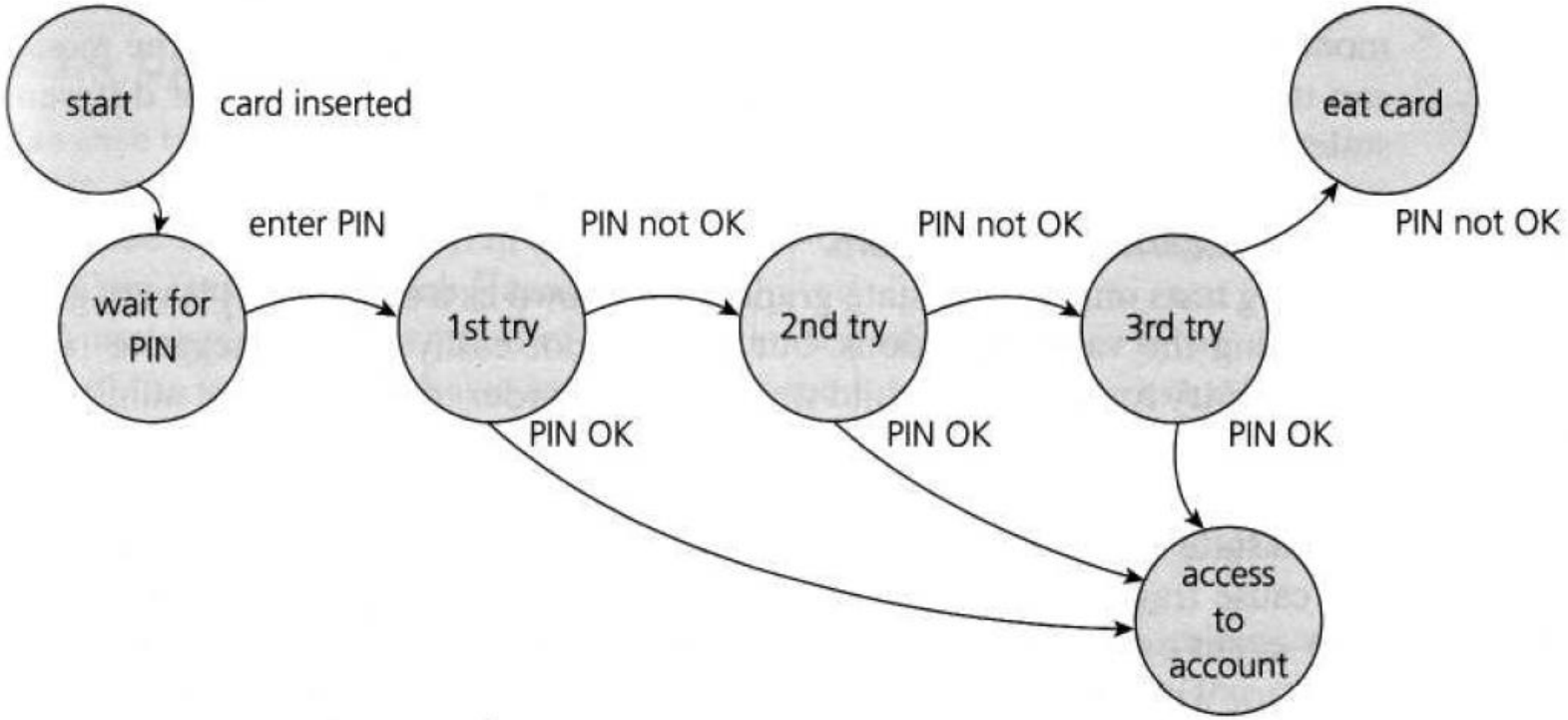
4.3 Techniques basées sur les spécifications ou boîte noire

» Test de transition d'état

- Un système peut montrer plusieurs réponses différentes en fonction des conditions **actuelles** ou **passées** (son état).
- Cet aspect du système peut être montré par un diagramme **d'états** et de **transitions**.
- Cela permet au testeur de visualiser le logiciel en termes d'états, de transitions entre les états, de données d'entrées et des actions qui peuvent résulter de ces transitions.
- Cette technique est utilisée pour couvrir une séquence typique d'états, et pour exécuter toutes les transitions.

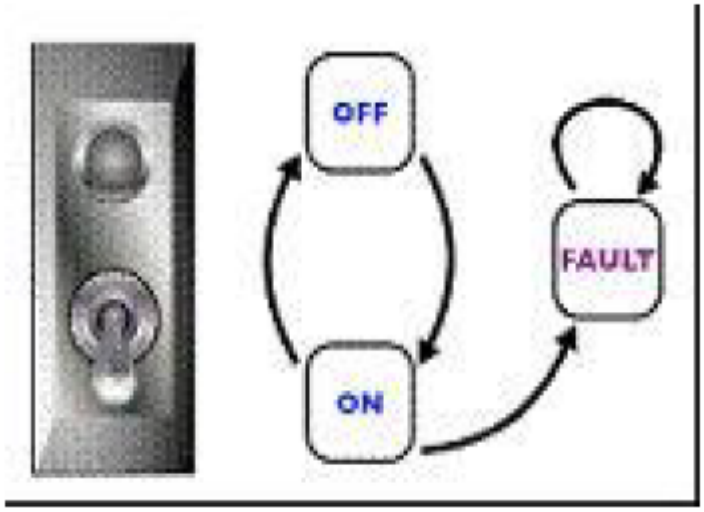
4.3 Techniques basées sur les spécifications ou boîte noire

» **Exemple:**



Exercice

Q. 286: Consider the following state transition diagram of a switch. Which of the following represents an invalid state transition?



A. OFF to ON
B. ON to OFF
C. FAULT to ON

4.3 Techniques basées sur les spécifications ou boîte noire

» Test de cas d'utilisation

- Un cas d'utilisation décrit l'interaction entre **acteurs** qui produit un résultat ayant une valeur pour l'utilisateur du système ou le client.

- Chaque cas d'utilisation a des **pré-conditions**, qui doivent être atteintes pour que ce **cas d'utilisation** soit exécuté avec succès.

- Chaque cas d'utilisation se termine par des **post-conditions**, qui sont les résultats observables et l'état final du système après la fin de l'exécution du cas d'utilisation.

4.3 Techniques basées sur les spécifications ou boîte noire

» **Exemple:**

	Step	Description
Main Success Scenario **A: Actor** **S: System**	1	A: Inserts card
	2	S: Validates card and asks for PIN
	3	A: Enters PIN
	4	S: Validates PIN
	5	S: Allows access to account
Extensions	2a	Card not valid S: Display message and reject card
	4a	PIN not valid S: Display message and ask for re-try (twice)
	4b	PIN invalid 3 times S: Eat card and exit

4.4 Techniques basées sur la structure ou boîte blanche

» Test des instructions et couverture

- La couverture des **instructions** est l'évaluation du pourcentage d'instructions exécutables qui ont été exercées par une suite de cas de test.

- La couverture des instructions est déterminée par le nombre d'instructions exécutables couvertes par des cas de test (conçus ou exécutés) divisé par le nombre de toutes les instructions exécutables dans le code testé.

$$= \frac{\text{nombre d'instructions exécutables}}{\text{nombre de toutes les instructions exécutables}}$$

4.4 Techniques basées sur la structure ou boîte blanche

1	read(a)
2	IF a > 6 THEN
3	b = a
4	ENDIF
5	print b

Test case	Input	Expected output
1	7	7

As all 3 statements are ‘covered’ by this test case, we have achieved 100% statement coverage

4.4 Techniques basées sur la structure ou boîte blanche

- **Test des décisions et couverture:**

- La couverture des **décisions**, liées aux tests de **branches**, est l'évaluation des pourcentages de résultats de décisions (p.ex. les options Vrai et Faux d'une instruction IF) qui ont été traitées par une suite de cas de test.

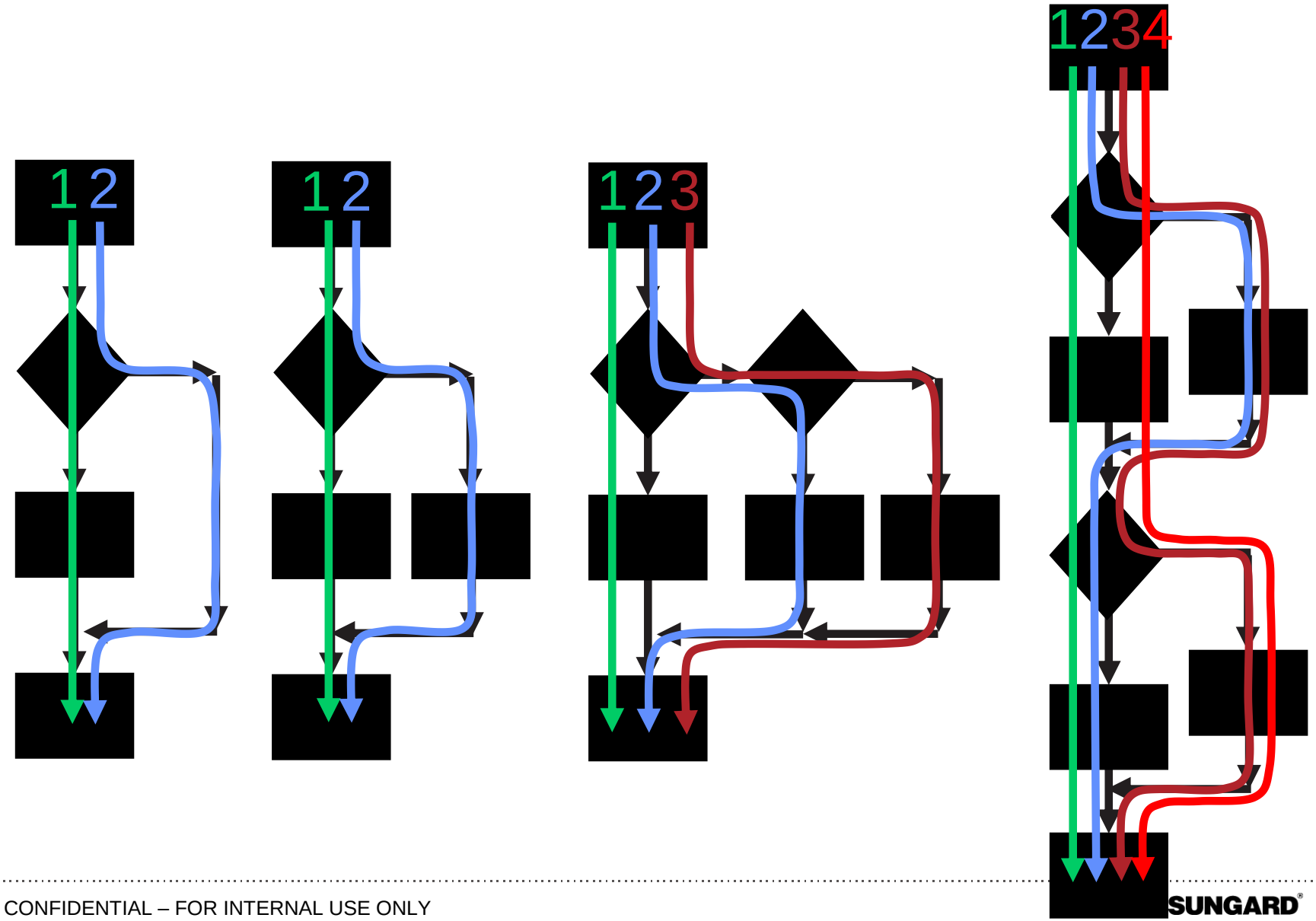
- La couverture des décisions est supérieure à la couverture des instructions:

  - Une couverture de 100% des décisions garantit une couverture à 100% des instructions.

  - L'inverse n'est pas vrai.

$$= \frac{\text{number of decisions outcomes exercised}}{\text{total number of decision outcomes}}$$

Couverture des chemins d'exécution



Example 1

Wait for card to be inserted

IF card is a valid card THEN

display “Enter PIN number”

IF PIN is valid THEN

select transaction

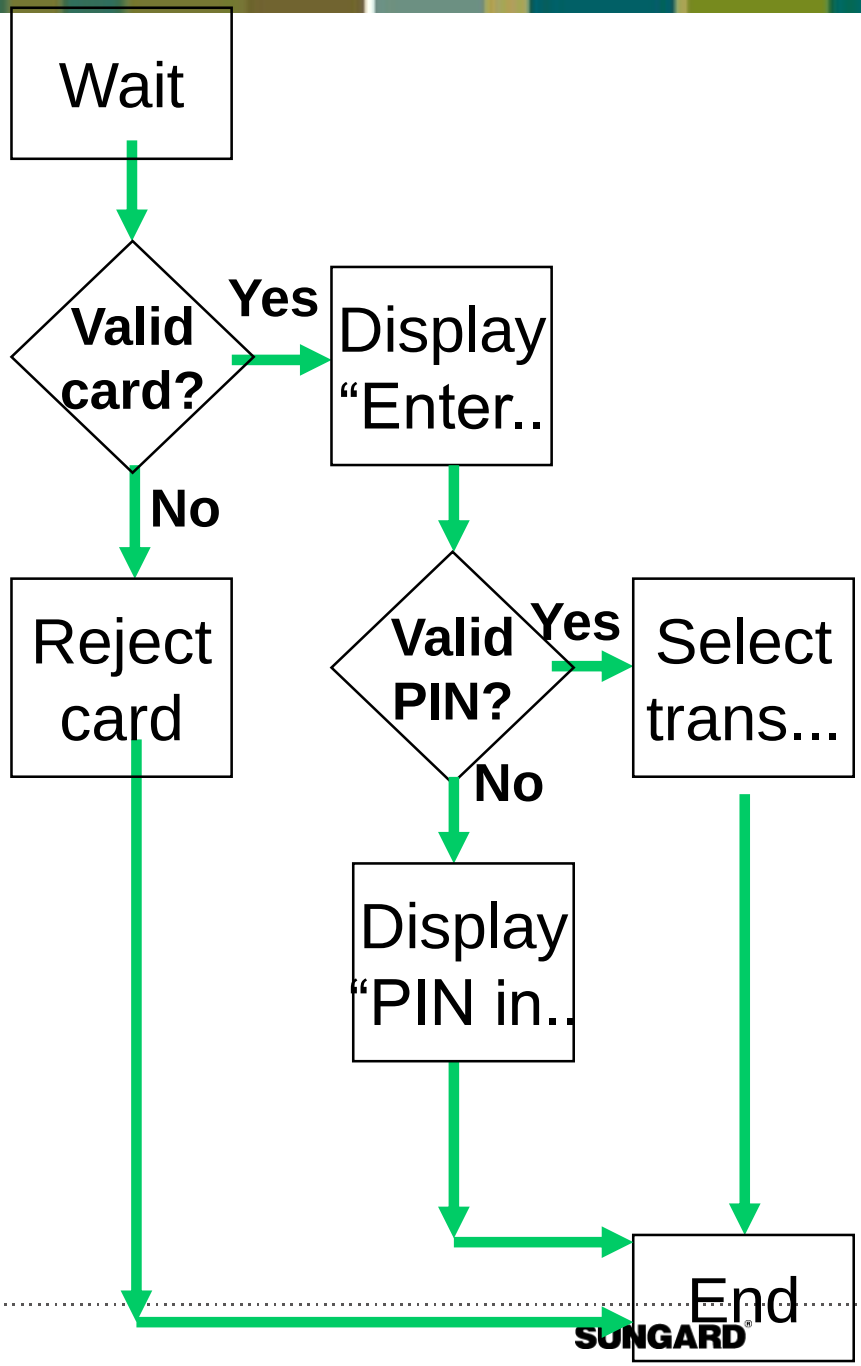
ELSE (otherwise)

display “PIN invalid”

ELSE (otherwise)

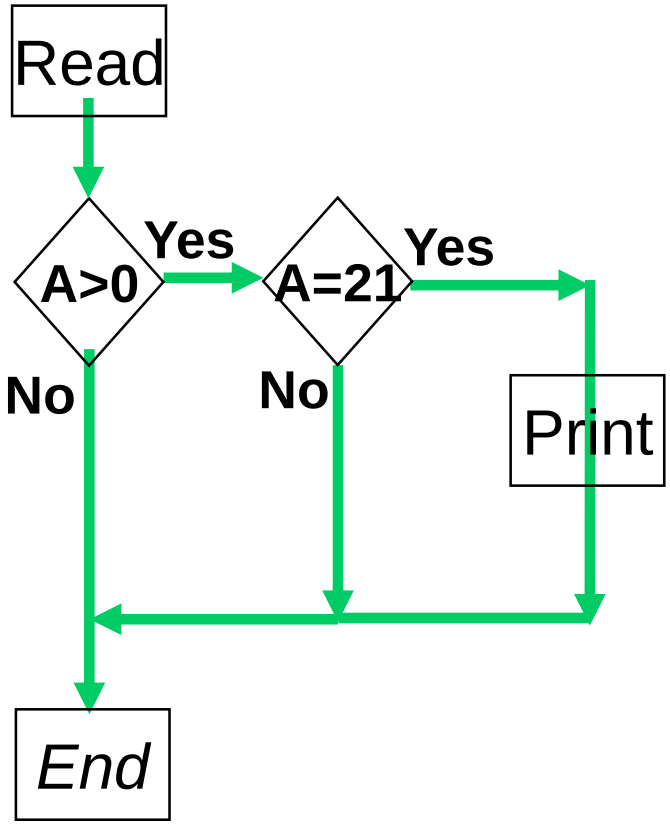
reject card

End



Example 2

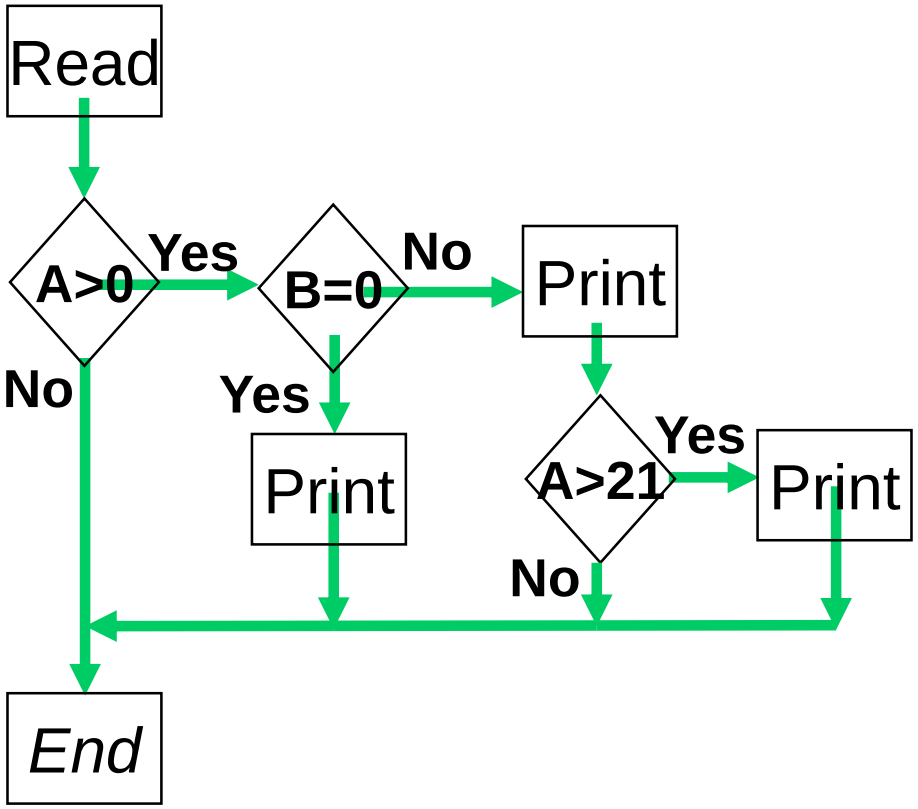
```
Read A
IF A > 0 THEN
    IF A  = 21 THEN
        Print “Key”
    ENDIF
ENDIF
```



- Cyclomatic complexity: 3
- Minimum tests to achieve:
  - Statement coverage: 1
  - Branch coverage: 3

Example 3

```
Read A
Read B
IF A > 0 THEN
    IF B  = 0 THEN
        Print “No values”
    ELSE
        Print B
        IF A > 21 THEN
            Print A
        ENDIF
    ENDIF
ENDIF
```

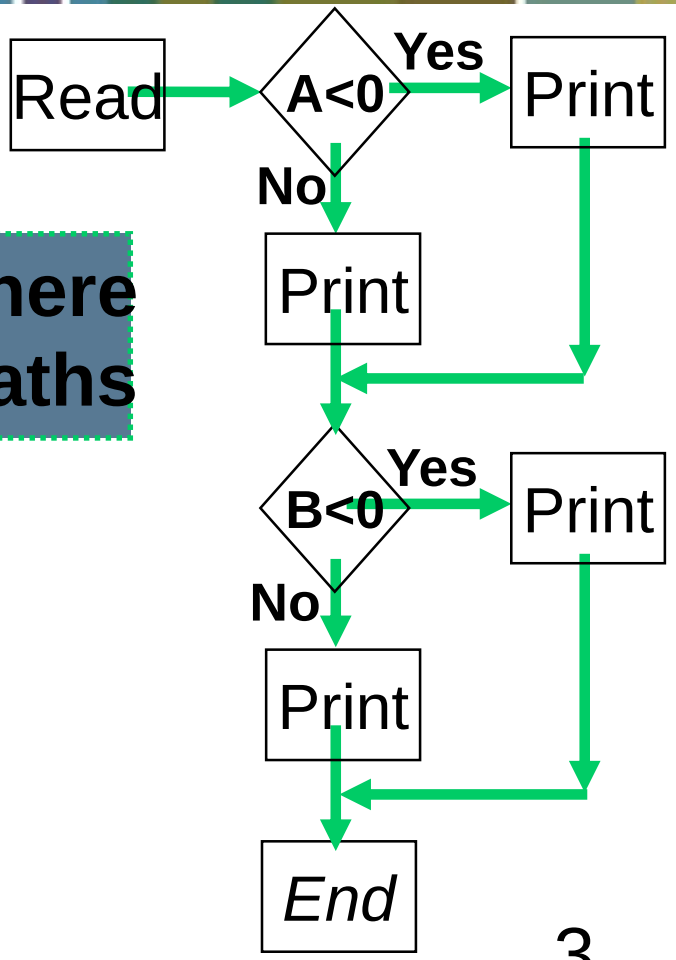


- Cyclomatic complexity: 4
- Minimum tests to achieve:
  - Statement coverage: 2
  - Branch coverage: 4

Example 4

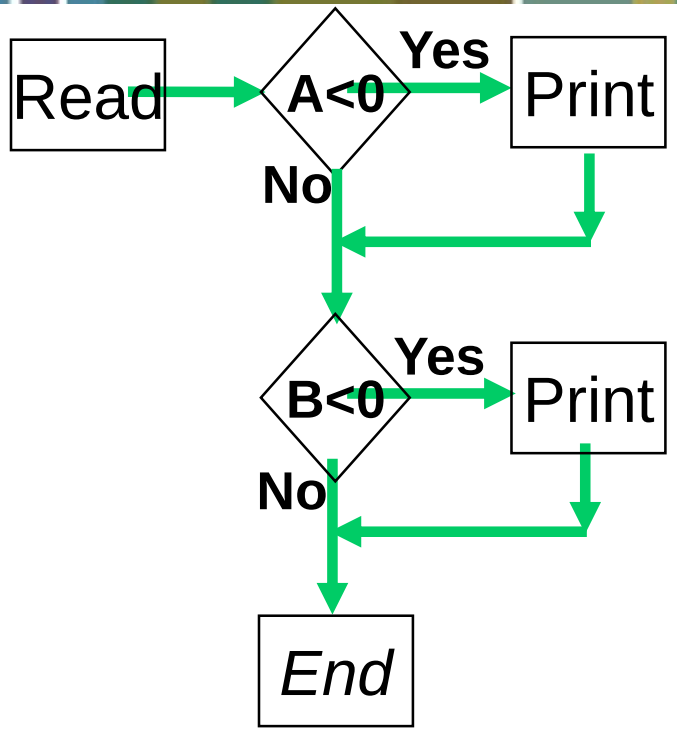
```
Read A
Read B
IF A < 0 THEN
    Print  “A negative”
ELSE
    Print  “A positive”
ENDIF
IF B < 0 THEN
    Print  “B negative”
ELSE
    Print  “B positive”
ENDIF
```

Note: there are 4 paths



- Cyclomatic complexity: 3
- Minimum tests to achieve:
  - Statement coverage: 2
  - Branch coverage: 2

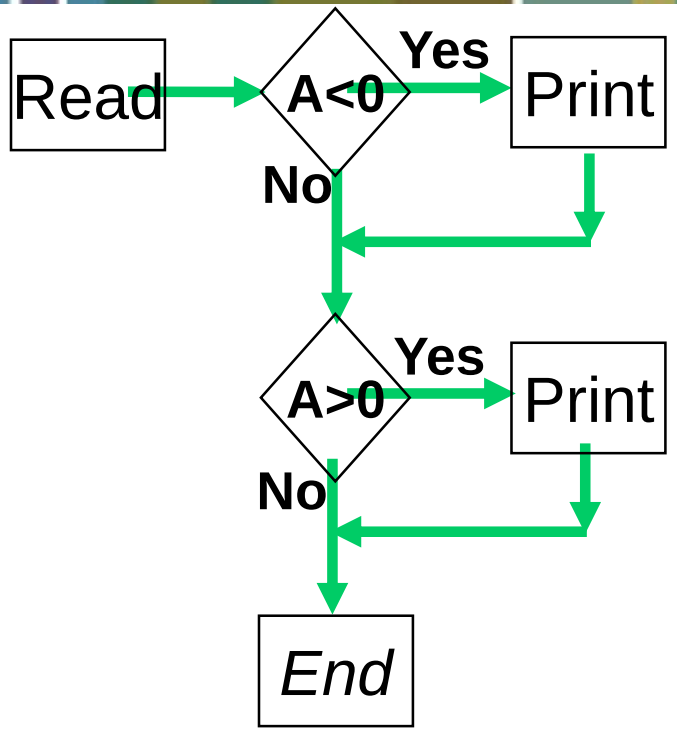
Example 5




```
Read A
Read B
IF A < 0 THEN
    Print  “A negative”
ENDIF
IF B < 0 THEN
    Print  “B negative”
ENDIF
```

- Cyclomatic complexity: 3
- Minimum tests to achieve:
  - Statement coverage: 1
  - Branch coverage: 2

Example 6




```
Read A
IF A < 0 THEN
    Print  “A negative”
ENDIF
IF A > 0 THEN
    Print  “A positive”
ENDIF
```

- Cyclomatic complexity: 3
- Minimum tests to achieve:
  - Statement coverage: 2
  - Branch coverage: 2

Exemple 7

- **Analyser la procédure suivante très simplifiée:**
- **Ask: "What type of ticket do you require, single or return?"**
- **IF the customer wants ‘return’**
- **Ask: "What rate, Standard or Cheap-day?"**
- **IF the customer replies ‘Cheap-day’**
- **Say: "That will be £11:20"**
- **ELSE**
- **Say: "That will be £19:50"**
- **ENDIF**
- **ELSE**
- **Say: "That will be £9:75"**
- **ENDIF**

Exemple 7

- **Maintenant, déterminer le nombre minimum de tests qui sont nécessaires pour faire en sorte que toutes les questions ont été posées, toutes les combinaisons ont eu lieu et toutes les réponses données.**
- **a) 3**
- **b) 4**
- **c) 5**
- **d) 6**

4.4 Techniques basées sur la structure ou boîte blanche

- **Couverture de conditions et couverture de conditions multiples**
  - Le concept de couverture peut aussi être appliqué aux autres niveaux de tests
  - Par exemple, au niveau intégration, le pourcentage des modules, composants ou classes qui ont été exercées par une suite de cas de test peut être exprimé comme une couverture de modules, composants ou classes,

4.5 Techniques basées sur l'expérience

- Les tests basé sur l'expérience sont conçus à partir **des compétences des testeurs**, de leur **intuition** et de leur **expérience** avec des applications et technologies similaires.

- Les tests intuitifs peuvent être utiles pour identifier des tests spéciaux, difficilement atteints par des approches formelles.

- **l'estimation d'erreur** ou « attaque par faute »:

  - les testeurs anticipent les défauts basés sur l'expérience.

  - Enumérer une liste des défauts possibles et de concevoir des tests

- **Les tests exploratoires**:

  - Comprennent la conception et l'exécution des tests, l'écriture des résultats de tests et l'apprentissage

  - Très utile lorsque les spécifications sont rares ou non adéquates, et que le test est soumis à de sévères contraintes de temps

4.6 Sélectionner les techniques de test

- Le choix des techniques de tests à utiliser dépend:
  - De système à tester.
  - Les objectifs de test.
  - Les exigences client.
  - Le risque.
- Lors de la création de cas de test, les testeurs utilisent généralement une combinaison de techniques de tests.

Chapitre 5 : Gestion des tests

5.1.Organisation des tests

» Organisation du test et indépendance

– L’efficacité de la découverte d'anomalies par le test et les revues peut être améliorée par l'emploi de testeurs indépendants. Les options pour l'indépendance sont les suivantes :

- Pas de testeurs indépendants, développeurs testant leur propre code.
- Testeurs indépendants incorporés à l'équipe de développement.
- Équipe ou groupe de test indépendant au sein de l'organisation, qui réfère au gestionnaire du projet ou aux responsables décisionnaires.
- Testeurs indépendants de l'organisation, de la communauté des utilisateurs et de l’informatique.

5.1.Organisation des tests

- Les avantages de l'indépendance :
  - Des testeurs indépendants voient des défauts différents et d’une autre nature et sont impartiaux.
  - Un testeur indépendant peut vérifier les hypothèses faites pendant la spécification et l'implémentation du système.
- Les inconvénients :
  - Déconnexion vis-à-vis de l'équipe de développement (en cas de totale indépendance).
  - Les développeurs perdent le sens de la responsabilité pour la qualité.
  - Des testeurs indépendants peuvent constituer un goulet d'étranglement comme dernier point de vérification et être accusés des retards.

5.1.Organisation des tests

» Tâches du responsable des tests et des testeurs

– Les tâches habituelles du responsable de test peuvent comprendre les suivantes :

- Coordonner la stratégie et le plan du test avec le chef de projet et d'autres acteurs
- Établir ou réviser une stratégie de test pour le projet ainsi qu'une politique de test pour l'organisation
- Apporter le point de vue du test aux autres activités du projet, comme la planification de l'intégration
- Planifier les tests en considérant le contexte et en comprenant les objectifs et les risques
- Démarrer la spécification, la préparation, l'implémentation et l'exécution des tests ainsi que surveiller et contrôler l'exécution.
- Adapter le planning en fonction des résultats et de l'avancement du test et entreprendre les actions nécessaires pour résoudre les problèmes .

5.1.Organisation des tests

- Introduire des mesures appropriées pour mesurer l'avancement du test et évaluer la qualité du test et du produit
- Sélectionner les outils pour aider le test et organiser la formation des testeurs à l'usage des outils.

– Les tâches habituelles des testeurs

- Passer en revue les plans du test et y contribuer.
- Analyser, passer en revue et évaluer, quant à leur testabilité, les exigences utilisateurs, les spécifications et les modèles.
- Créer des spécifications de test.
- Mettre en place l'environnement de test (souvent en coordination avec l'administration système et la gestion de réseau).
- Préparer et obtenir les données de test.

5.1.Organisation des tests

- Implémenter des tests à tous les niveaux, exécuter et consigner les tests, évaluer les résultats et documenter les écarts vis-à-vis des résultats attendus.
- Employer les outils d'administration ou de gestion des tests et les outils de surveillance des tests en fonction du besoin.
- Automatiser les tests (éventuellement avec l’aide d'un développeur ou d'un expert en automatisation de test).
- Mesurer les performances des composants et systèmes (si pertinent).
- Passer en revue les tests développés par d'autres.

5.2 Estimation et planification des tests

» **Planification des tests:**

– La planification est influencée par la politique de test de l'organisation, la portée du test, les objectifs, les risques, les contraintes, la criticité, la testabilité et la disponibilité des ressources.

– La planification du test est une activité continue et est effectuée tout au long des processus et activités du cycle de développement.

– Le retour issu des activités de test est employé pour constater l'évolution des risques et modifier alors la planification.

5.2 Estimation et planification des tests

» Activités de planification des tests

– Les activités de la planification des tests pour un système ou pour une partie de celui-ci peuvent être:

- Définir le périmètre du test, les risques et identifier les objectifs du test
- Définir l'approche générale du test (stratégie de test), y compris la définition des niveaux de test ainsi que celle des critères d'entrée et de sortie.
- Intégrer et coordonner des activités de test dans les activités du cycle de développement : acquisition, fourniture, développement, exploitation et maintenance.
- Prendre des décisions quant à ce qui doit être testé, quels rôles vont exercer quelles activités, quand et comment ces activités doivent être exercées, comment évaluer les résultats des tests et quand arrêter les tests (critères de sortie).

5.2 Estimation et planification des tests

- Planifier les activités d'analyse et de conception des tests
- Planifier les activités de conception, d'exécution et dévaluation des tests
- Assigner les ressources aux différentes tâches définies.
- Définir le volume, le niveau de détail, la structure et les modèles pour la documentation du test.
- Sélectionner des mesures pour suivre et contrôler la préparation et l'exécution des tests, l'élimination des défauts et la résolution des problèmes relatifs aux risques.
- Déterminer le niveau de détail pour les procédures de test de façon à fournir suffisamment de détails pour permettre une préparation et une exécution reproductibles des tests.

5.2 Estimation et planification des tests

» Critères d'entrée

- Les critères d'entrée définissent quand démarrent les tests (par exemple quand débute un niveau de test, quand un jeu de test est prêt à être exécuté).
- Typiquement, les critères d'entrée peuvent couvrir:
  - Disponibilité et préparation de l'environnement de test.
  - Préparation des outils de tests dans l'environnement de test.
  - Disponibilité de code testable.
  - Disponibilité des jeux de données.

5.2 Estimation et planification des tests

» Critères de sortie

- L’objectif des critères de sortie est de définir quand arrêter le test, par exemple, à la fin d‟un niveau de test ou lorsqu'une série de tests a atteint un objectif donné.

- Typiquement, les critères de sortie peuvent comprendre les suivants :

  - Des mesures d'exhaustivité, comme la couverture de code, de fonctionnalités ou de risques.

  - L'estimation de la densité des anomalies ou des mesures de fiabilité.

  - Le coût

  - Les risques résiduels, comme les anomalies non corrigées ou le manque de couverture du test dans certaines parties.

  - Un calendrier, par exemple, basé sur la date de mise sur le marché.

5.2 Estimation et planification des tests

» Estimation des tests

Deux approches pour l'estimation de l'effort de test sont couvertes par le présent syllabus :

– Estimation de l'effort de test basée sur des mesures issues de projets antérieurs ou similaires ou basée sur des valeurs typiques.

– L’approche experte : estimation des tâches par le détenteur de ces tâches ou par des experts.

5.2 Estimation et planification des tests

- L’effort de test peut dépendre d'un certain nombre de facteurs, dont les suivants :

- Les caractéristiques du produit : la qualité des exigences et autres informations utilisées pour les modèles de test (c'est-à-dire la base de test), la taille du produit, la complexité du domaine, les exigences de fiabilité et de sécurité ainsi que celles de la documentation.

- Les caractéristiques du processus de développement : la stabilité de l'organisation, les outils employés, le processus de test, le savoir-faire des personnes impliquées et les contraintes de temps.

- Les résultats du tests : le nombre de défauts et le volume des reprises exigées.

5.2 Estimation et planification des tests

» Stratégie de test, Approche de test

- L’approche de test est la mise en œuvre d’une stratégie de test pour un projet spécifique. Elle est définie et affinée dans les plans et scénarii de test. Elle comprend typiquement les décisions basées sur le projet (de test), ses buts ainsi qu'une analyse des risques.

- Elle constitue le point de départ pour la planification du processus de test, pour sélectionner les types et techniques de test qui seront appliqués ainsi que pour définir les critères d'entrée et de sortie.

- L'approche sélectionnée dépend du contexte et peut prendre en considération les risques, la sécurité et les dangers, les ressources disponibles et leur niveau d'expertise, la technologie et la nature du système ,les objectifs, les normes et règlements en vigueur.

5.3 Suivi et contrôle du déroulement des tests

Suivi de l'avancement des tests

- L'objectif du suivi du test est de fournir un retour et une visibilité sur les activités de test.
- Des mesures peuvent aussi être utilisées pour évaluer l'avancement par rapport au calendrier et au budget planifiés. Les mesures de test habituelles sont:
  - Le pourcentage du travail consacré à la préparation des cas de test (ou pourcentage des cas de test planifiés et préparés).
  - Pourcentage du travail consacré à la préparation de l'environnement de test.
  - L'exécution de cas de test (par exemple, nombre de cas de test exécutés ou non et nombre de cas de test réussis ou échoués).
  - Les informations sur les défauts (par exemple, densité des défauts, défauts trouvés et corrigés, taux des défaillances et résultats du re-test).

5.3 Suivi *et contrôle du déroulement des tests*

Reporting des tests

»Le reporting des tests consiste à résumer les informations relatives à l'entreprise du test ; il comprend les activités suivantes :

- Ce qui s'est passé pendant une phase de test, comme les dates où les critères de sortie ont été atteints.
- Les informations et mesures analysées pour étayer les recommandations et décisions pour de futures actions, comme une évaluation des défauts restants, les avantages économiques de tests prolongés, les risques non couverts et le niveau de confiance dans le logiciel testé.

»Des mesures devraient être recueillies pendant et à la fin d'un niveau de test dans le but dévaluer:

- L'adéquation des objectifs du test avec ce niveau de test.
- L'adéquation des approches du test empruntées.
- L'efficacité du test par rapport à ses objectifs.

5.3 Suivi et contrôle du déroulement des tests

» Contrôle des tests

» Le contrôle du test décrit les actions d'orientation et de correction entreprises comme résultat des informations et mesures recueillies et consignées.

» Ces actions peuvent couvrir toute activité de test et influencer toute autre activité ou tâche du cycle de vie logiciel.

» Exemples d'actions de contrôle des tests :

- Prendre des décisions sur la base des informations recueillies lors du suivi des tests.
- Une nouvelle affectation de priorités aux tests en cas de mise en évidence d'un risque identifié (par exemple, retard de livraison du logiciel).
- Une modification du calendrier de test en raison de la disponibilité d'un environnement de test.
- Définition d'un critère d'entrée exigeant que des corrections soient testées par le développeur avant de les accepter dans une version.

5.4 Gestion de configuration

»L'objectif de la gestion de configuration est d'établir et de maintenir l'intégrité des produits (composants, données et documentation) du logiciel ou du système durant le cycle de vie du projet et du produit.

»La gestion de configuration aide le testeur à identifier de manière unique (et à reproduire) l'élément testé, les documents de test, les tests et le harnais de test.

»Pour le test, la gestion de configuration peut permettre d'assurer que :

– Tous les éléments faisant partie du testware sont identifiés, sous contrôle de versions, que les changements sont identifiés et re-traçables, reliés les uns aux autres et aux éléments de développement (objets de test), de sorte que la traçabilité peut être maintenue pendant tout le processus du test.

– Tous les documents identifiés et les éléments du logiciel sont référencés de manière non ambiguë dans la documentation de test.

5.5 Tests et risques

» Risques liés au projet

Les risques liés au projet sont les risques menaçant la capacité de ce dernier à atteindre ses objectifs, tels que :

- Facteurs organisationnels :manque de compétence et de formation du personnel ,problèmes de personnel, problèmes politiques,…
- Problèmes techniques :problèmes pour définir des exigences correctes ,La mesure selon laquelle les exigences seront satisfaites en fonction de contraintes existantes ,Environnement de test indisponible, qualité de la conception, du code et des tests,…
- Problèmes d'acquisition :défaillance d'une tierce partie ,problèmes contractuels,…

5.5 Tests et risques

Risques liés au produit

- Les risques relatifs au produit sont un type particulier de risque pour le succès d'un projet.
- Le test, comme activité de contrôle des risques fournit un retour quant aux risques restants par la mesure de l'efficacité de l'élimination des défauts critiques et des plans de contingence.
- Les risques sont employés pour décider quand commencer à tester et où tester davantage ; le test est utilisé pour réduire le risque qu'un événement indésirable ne survienne ou pour réduire l'impact de ce dernier .
- De plus, le test peut aider à identifier de nouveaux risques, à déterminer quels risques doivent être minimisés et à réduire l'incertitude relative aux risques.

5.6 Gestion des incidents

- Les différences entre les résultats attendus et les résultats effectifs doivent être consignées en tant qu'incidents.

- Un incident doit être analysé et peut par la suite devenir un défaut. Des actions adéquates doivent être définies afin de traiter les incidents et les défauts. Les incidents et les défauts doivent être suivis depuis leur découverte et classification jusqu'à leur correction et la confirmation de leur résolution.

- Pour pouvoir gérer tous les incidents jusqu'à la fin d'un projet, une organisation devrait établir un processus de gestion des incidents et des règles pour leur classification.

- Les incidents peuvent survenir pendant le développement, les revues, le test ou l'utilisation d'un produit logiciel. Ils peuvent survenir en raison de problèmes dans le code, dans le système opérationnel ou dans tout type de documentation .

5.6 Gestion des incidents

» Les rapports d'incidents peuvent avoir les objectifs suivants :

- Fournir aux développeurs et aux autres parties un retour sur le problème concerné pour en permettre l'identification, la localisation et la correction nécessaires.
- Fournir aux responsables du test le moyen de suivre la qualité d'un système sous test et l'avancement du test.
- Fournir des idées pour l'amélioration du processus de test.

» La structure d'un rapport d'incident est couverte par le guide « Standard for Software Test Documentation » (**IEEE Std 829-1998).**

Chapitre 6 : Outils de Support aux Tests

6.1 Les types d'outils de tests

- **Outils de support aux tests**

- Un outil utilisé pour un ou plusieurs activités:

  - Outils qui sont directement utilisés dans le test (Outils d'exécution de test, de génération de données et comparaison de résultats).

  - Outils qui aident en contrôlant le processus de test (gérer les tests, les résultats de test, des données, des conditions, des incidents…etc).

  - Outils qui sont utilisés dans la reconnaissance ou exploration (outils qui contrôle l'activité d'un fichier pour une application).

  - N'importe quel outil qui facilite le test (Un tableur).

6.1 Les types d'outils de tests

» Un outil peut avoir un ou plusieurs buts selon le contexte:

– Améliorer l'efficacité des activités de test (automatisation des taches répétitives, supporter des activités de test manuelles).

– Automatiser les activités qui exigent des ressources importantes (Tests statiques)

– Automatiser les activités qui ne peuvent pas être exécutées manuellement.

– Augmenter la fiabilité du test (Comparaison de beaucoup de données).

» Framework de test :

= Harnais de tests : Bibliothèques réutilisables et extensibles de test

– Un type de conception d'automatisation des tests (pilotés par les données, mots-clés).

6.1 Les types d'outils de tests

» Outils d'aide à la gestion du test et des tests

– Outils de gestion des tests (TestLink)

Ils permettent d'exécuter des tests, dépister les défauts et gérer les exigences ainsi que la traçabilité des objets de tests vers les spécifications des exigences.

– Outils de gestion des exigences (TestLink)

Ces outils enregistrent les énoncés et les attributs des exigences ainsi que la tracabilité des exigences vers les tests individuels.

– Outils de gestion d'incidents (outils de suivi de défauts BUGTRACKER Mantis)

Ces outils enregistrent et gèrent les rapports d'incidents et aident à gérer leur cycle de vie.

– Outils de gestion de configuration de test (Perforce)

Le stockage et la gestion de version du testware et du logiciel associé.

6.1 Les types d'outils de tests

» Outils d'aide aux tests statiques

– Outils de revue

Ces outils aident aux revues des processus, des check-lists, des directives de revue et sont utilisés pour enregistrer et communiquer les commentaires des revues, les rapports sur les défauts et les essais.

– Outils d'analyse statique (D Klocwork & SONAR)

Ces outils fournissent une aide pour introduire des normes de codage, l'analyse des structures et des dépendances. Ils peuvent également aider à la planification ou l'analyse de risque en fournissant des métriques pour le code.

– Outil de modélisation (D)

Ces outils sont employés pour valider les modèles de logiciel en énumérant des incohérences et en trouvant des défauts. Ces outils peuvent souvent aider en produisant quelques cas de test basés sur le modèle.

6.1 Les types d’outils de tests

» Outils d’aide à la spécification des tests

– Outils de conception de tests:

Ces outils permettent à des tests d’être exécutés automatiquement en fournissant habituellement un registre de test pour chaque exécution de test.

– Outils de préparations de données de tests:

Les outils de préparation des données agissent sur des bases de données, fichiers ou transferts de données afin d’élaborer des données de tests utilisables lors de l’exécution des tests, ceci en assurant la sécurité des données grâce à leur anonymisation.

6.1 Les types d’outils de tests

- **Outils d’aide à l'exécution et à l’enregistrement des tests**

- **Outils d’exécution des tests**

Ces outils permettent à des tests d'être exécutés automatiquement, ou semi-automatique ment, utilisant les entrées enregistrées et les résultats prévus, par l'utilisation d'un langage de script et fournissent habituellement un registre de test pour chaque exécution de test. Ils peuvent également être employés pour enregistrer des tests, et disposent habituellement d'un langage de script ou une configuration via une interface graphique utilisateur pour le paramétrage des données et toute autre personnalisation dans les tests.

- **Harnais de tests/Outils framework de tests unitaires (D**)

Un harnais ou framework de tests unitaires facilite le test des composants ou des parties d'un système en simulant l'environnement dans lequel cet objet de tests s'exécutera, par la fourniture d'objets factices comme des bouchons ou des pilotes.

6.1 Les types d’outils de tests

» **Comparateurs de tests** (WinMerge, ExamDiff)

Les comparateurs de tests déterminent des différences entre les fichiers, bases de données ou résultats de tests.

Outils de mesure de couverture (D) (D gcov, Cobertura)

Ces outils mesurent le pourcentage de certains types de structures de code qui ont été exercés (par exemple, des déclarations, des branches ou des décisions, et appels de module ou fonction) par un ensemble de tests.

» **Outils de test de sécurité** (OWASP Open Web Application Security Project)

Ces outils sont utilisés pour évaluer les caractéristiques de sécurité du logiciel. Ceci inclut l'évaluation de la capacité du logiciel à protéger la confidentialité, l'intégrité, l'authentification, l'autorisation, la disponibilité, et la non-répudiation des données.

6.1 Les types d’outils de tests

- **Outils de support de performance et de surveillance**

- **Outils d'analyse dynamique (D)** (D Purify pour la fuite de mémoire)

Les outils d'analyse dynamique détectent des défauts qui ne se manifestent que lors de l'exécution du logiciel.Ils sont typiquement utilisés dans des tests de composants et d'intégration de composants, et dans les tests de middleware.

- **Outils de test de performance/test de charge/test de stress** (Jmeter & LoadRunner)

Les outils de test de performance surveillent et rapportent sur la façon dont se comporte un système selon une grande variété de conditions d'usage simulées en termes de nombre d'utilisateurs simulanés, de leur modèle de montée en charge, de fréquence et de pourcentage relatif de transactions.

- **Outils de surveillance**

Les outils de surveillance analysent continuellement, vérifient et rendent compte de l'utilisation de ressources systèmes spécifiques, et donnent des alertes sur de possibles problèmes de service.

6.1 Les types d’outils de tests

- **Outils de support pour des besoins de tests spécifiques**

- **Evaluation de la qualité des données**

Les données sont au centre de certains projets tels que des projets de conversion/migration des données et des applications comme le stockage de données. Leurs caractéristiques peuvent varier en termes de criticité et volume. Dans de tels contextes, des outils dévaluation de la qualité des données doivent être utilisés pour revoir et vérifier les règles de conversion et de migration des données, pour s'assurer que les données traitées sont correctes, complètes et se conforment à une norme prédéfinie spécifique au contexte.

- Objectif: revoir et vérifier les règles de conversion et de migration des données

6.2 Utilisation efficace des outils

- **Bénéfices potentiels et risques liés aux outils de test (pour tous les outils)**
  - Réduction du travail répétitif.
  - Répétabilité et cohérence accrues
  - Evaluation objective
  - Facilité d’accès aux informations concernant les tests ou leurs exécution

6.2 Utilisation efficace des outils

» Risques liés à l'utilisation d'outils :

- Attentes irréalistes placées dans l'outil.
- Sous-estimation du temps, du coût et de l'effort pour son introduction initiale.
- Sous-estimation du temps et de l'effort pour obtenir des bénéfices continues.
- Sous-estimation de l'effort requis pour maintenir les acquis générés par l'outil.
- Confiance excessive dans l'outil.
- Négligence du contrôle des éléments de test dans l'outil.
- Risque de faillite de l'éditeur de l'outil, de le retirer ou de le vendre.

6.3 Introduire un outil dans une organisation

» Les principes

- Evaluation de la maturité de l'organisation, de ses forces et de ses faiblesses et identification des possibilités d'amélioration du processus de test.
- Evaluation au regard d'exigences claires et de critères objectifs.
- Une preuve de concept (test d'évaluation du bon fonctionnement de l'outil avec l'existant en terme d'infrastructure et avec l'outil sujet de test).
- Evaluation du vendeur (aspects de formation, de support et de commerce).
- Identification des exigences internes pour le soutien et la tutelle dans l'utilisation de l'outil
- Evaluation de besoin de formation.
- Evaluation du rapport coût/bénéfice basés sur un cas métier concret

6.3 Introduire un outil dans une organisation

- **Commencer par un projet pilote:**
  - Apprendre l'outil plus en profondeur.
  - Adapter l'outil aux processus et pratiques existants.
  - Décider d'une manière standard d'utiliser, de gérer, de stocker et de maintenir l'outil et le testware (exemple : nommage des fichiers).
  - Evaluer si les bénéfices escomptés seront atteints pour un coût raisonnable.

6.3 Introduire un outil dans une organisation

» Les facteurs de succès:

- Etendre l'outil au reste de l'organisation de façon *incrémentale*
- Adapter et améliorer les processus de façon à les adapter à l'utilisation de l'outil.
- Fournir de la formation et une assistance aux nouveaux utilisateurs.
- Etablir des guides d'utilisation.
- Implémenter une manière de tirer des enseignements de l'utilisation de l'outil.
- Surveiller l'utilisation de l'outil et les bénéfices recueillis.
- Fournir le support pour l'équipe de test pour un outil donné.
- Recueillir l'expérience acquise de toutes les équipes.

Test

- To start **ISTQB Mock Test - 1** , Click Here
- To start **ISTQB Mock Test - 2** , Click Here
- To start **ISTQB Mock Test - 3** , Click Here