Qualité et Test Logiciels

IMEN AKROUT

PLAN du cours

1. Introduction
2. Terminologie
3. Les fondamentaux du software testing
4. Les niveaux du software testing
5. Les types des tests logiciels
6. Test Management
7. La qualité logicielle
8. Les outils de test et de qualité

PLAN du cours

1. **Introduction**
2. **Terminologie**
3. Les fondamentaux du software testing
4. Les niveaux du software testing
5. Les types des tests logiciels
6. Test Management
7. La qualité logicielle
8. Les outils de test et de qualité

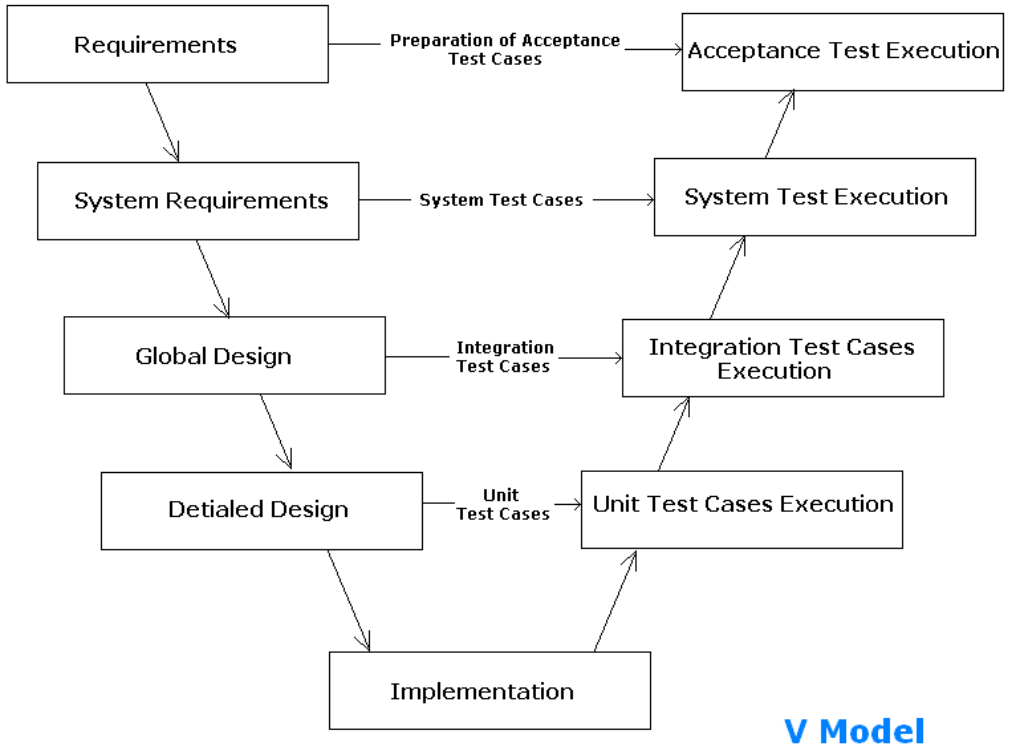
Introduction

- Test et Qualité? Qu'attendez-vous?
- Importance?

Terminologie

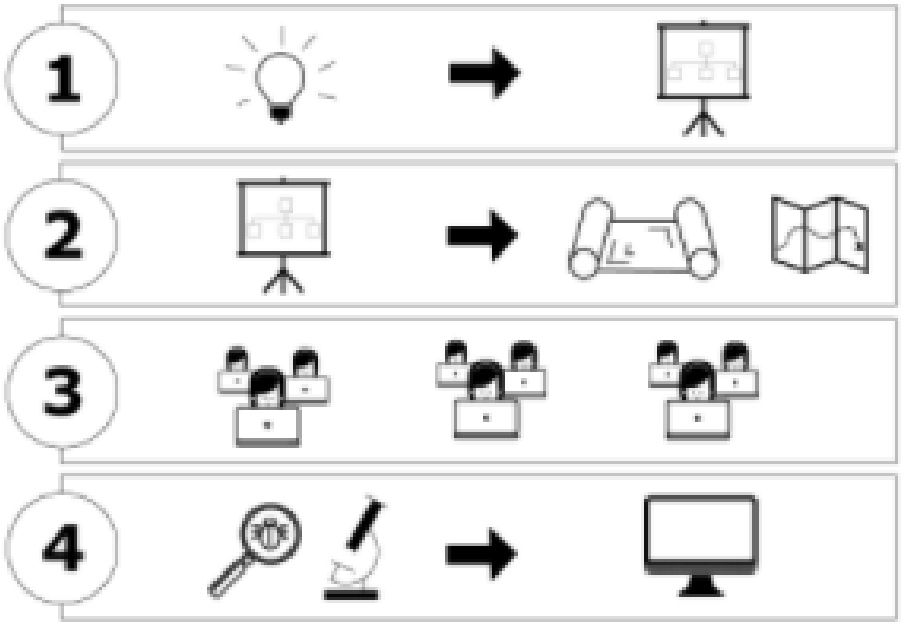
- Software Engineering, c'est quoi?
- SDLC? Etapes?
- Méthodologies? Process? Exemples?

Terminologie

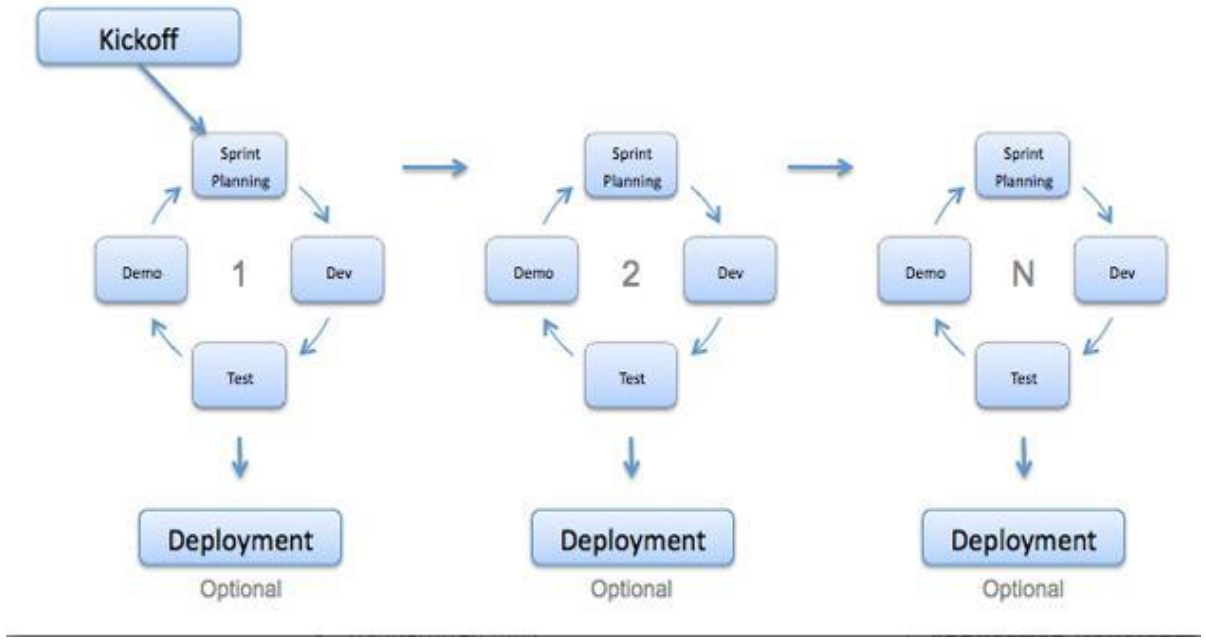


V Model

Terminologie



Terminologie



Terminologie

Méthodologie Séquentielle	Méthodologie Agile
▪ Simple ▪ Facile à utiliser ▪ l'intégralité du projet est définie dès le début ▪ Budget est connu	▪ Très flexible avec les changements ▪ Le projet va lui-même être découpé en étapes ▪ Test et validation pour chaque livraison ▪ Les corrections et les change request sont supportés
▪ Très rigide ▪ Pas de prototype ▪ N’est pas efficace avec des gros projets	▪ C’est déconseillé de changer de l’équipe ▪ Spec non clair ▪ Budget non clair

PLAN du cours

1. Introduction
2. Terminologie
3. **Les fondamentaux du software testing**
4. Les niveaux du software testing
5. Les types des tests logiciels
6. Test Management
7. La qualité logicielle
8. Les outils de test et de qualité

Pourquoi on a besoin de tester

- Les systèmes logiciels font partie de la vie:
  - Applications commerciales (par exemple bancaires),
  - …
  - Produits de consommation (par exemple, les voitures).
- La plupart des gens ont eu une expérience avec un logiciel qui n'a pas fonctionné comme prévu.
- Un logiciel qui ne fonctionne pas correctement peut entraîner de nombreux problèmes:
  - Perte d'argent,
  - Perte de temps ,
  - Réputation professionnelle,
  - Causer des blessures ou la mort.

Pourquoi on a besoin de tester

- Vol 501 d'Ariane 5: explosion du fusée seulement 36,7 secondes après décollage),
- an 2000 (la suite de 99 est 100 => après 1999 on a eu 1900),
- Therac-25: Radiothérapie: impliqué dans au moins six accidents durant lesquels des patients reçurent des doses massives de radiation,
- etc.

Causes des défauts logiciels

- Un être humain peut faire une erreur:
  - Temps,
  - Code complexe,
  - Infrastructure complexe,
  - Changement de la technologie.

- Des conditions environnementales influencer l'exécution du logiciel en modifiant les conditions matérielles:
  - le rayonnement,
  - le magnétisme,
  - la pollution.

Error: Bug, Defect, Failure, Fault, Mistake?

- Si un défaut est exécuté, le système peut ne pas faire ce qu'il devrait faire (ou faire quelque chose qu'il ne devrait pas faire), provoquant un échec.
- Donc, quand le résultat actuel dévie du résultat attendu, on parle d'erreur (Defect)
- De plus, n'importe quelle déviation par rapport au requirements document est un "Defect".
- On trouve de différents noms pour l'erreur d'un software: bug, issue, incidents, etc.

Error: Bug, Defect, Failure, Fault, Mistake?

- **Failure**: **Défaillance**: c'est la manifestation de l'erreur. Le logiciel retourne un résultat différent du celui qui est attendu.
- **Fault**: **Faute**: la cause de l'erreur. Une erreur humaine faite durant une activité software (design, implementation, test) et qui cause une défaillance.
- Toutes les erreurs ne reproduisent pas une défaillance. Certaines erreurs peuvent rester inactives dans le code et on peut ne pas s'en apercevoir. (erreurs dans un "dead code")

Rôle des tests

- Trouver des défauts
- Fournir des informations pour la prise de décision
- Prévenir les défauts
- Assurer la compréhension des exigences (Requirements)
- Assurer et mesurer la qualité
  - réduire le niveau global de risque dans un système.
  - la qualité du système logiciel augmente lorsque ces défauts sont corrigés.

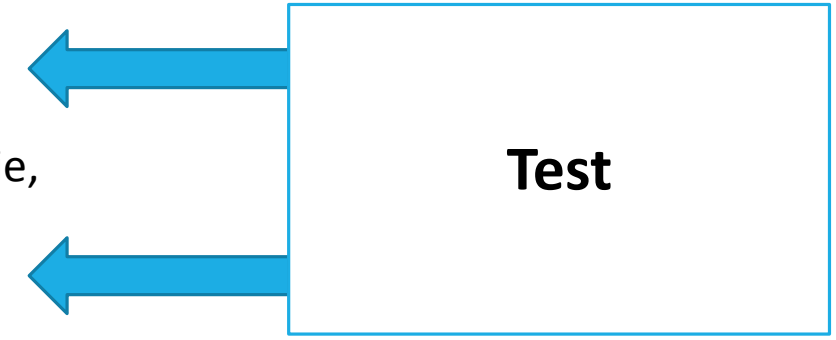
Combien de tests nous avons besoin?

- Niveau de risque
  - Risque technique
  - Risque de sécurité
- Contrainte du temps
- Contrainte du budget

C’est quoi un test?

- ‘Tester’ est: réaliser l’exécution d’un programme !?

- **Plusieurs activités:**
  - Planification et contrôle,
  - Choisir les conditions de test, concevoir et exécuter des cas de test,
  - Vérifier les résultats,
  - Evaluer les critères de sortie,
  - Valider.



Vérification

- Une opération qui permet de s'assurer que le produit est conçu pour fournir toutes les fonctionnalités requises au client
- La vérification est réalisée tout au long du "development process".
- Elle inclue les reviews, les meetings, les walkthroughs, les inspections de code, etc... pour évaluer documents, plannig, code, requirements et specifications.
- Elle permet de répondre à la question: est ce que je suis en train de construire le logiciel correctement?

Validation

- La validation détermine si le système est conforme aux requirements, couvre les fonctionnalités pour lesquelles il est destiné et satisfait le besoin du client.
- La validation est réalisée à la fin du “development process” et a lieu après la finalisation des différentes vérifications
- Elle permet de répondre à la question: est ce que je suis en train de construire le bon logiciel? (Am I building the right product?)

PLAN du cours

1. Introduction
2. Terminologie
3. Les fondamentaux du software testing
4. **Les niveaux du software testing**
5. Les types des tests logiciels
6. Test Management
7. La qualité logicielle
8. Les outils de test et de qualité

Les niveaux du software testing

1. Unit Testing
2. Component Testing
3. Integration Testing
4. Component Integration Testing
5. System Testing
6. System Integration Testing
7. Acceptance Testing
8. Alpha Testing
9. Beta Testing

Unit Testing

- Un “unit” est la plus petite partie testable dans une application (fonction, classe, procédure, interface…)
- Un test unitaire est une méthode qui permet de tester des “units” individuels et de déterminer leur conformité aux besoins
- Les tests unitaires sont créés et exécutés par les développeurs

Unit Testing

- Le but du “unit testing” est de
  - Séparer chaque partie de l’application
  - Tester que les parties individuelles fonctionnent correctement.
- Le développeur définie l’input et l’output d’un “code unit” et doit prendre en charge le résultat retourné par le “unit test” (successful, failed)

Unit Testing

- Le unit testing sont exécutés durant le white box testing
- Le white box testing est connu aussi sous les noms
  - Structure-based testing
  - Structural testing
  - Glass box testing.
- Durant cette étape, le testeur doit connaître comment le logiciel est implémenté et comment il fonctionne.

Unit Testing

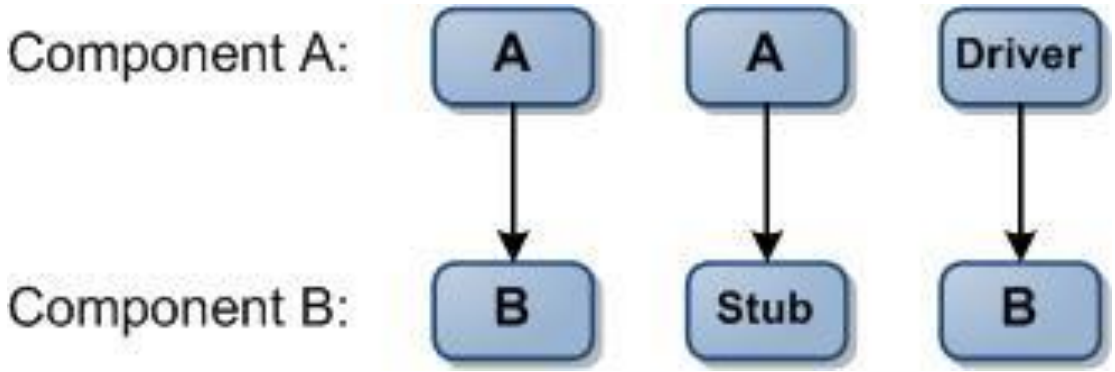
- Avantages des unit tests:
  - Defects découverts tôt dans le projet
  - Faciliter la maintenance et la modification du code. Le fait de définir des “unit” testables rend le code moins interdépendant, d’où la réduction de l’impact des changements.
  - Simplifier le debuggage du système
- C’est la base de la méthodologie TDD (Test Driven Development)
  - Préparer et automatiser les cas de test avant de coder,

Component Testing

- C’est une méthode où chaque composant d’une application est testé séparément.
- Connu aussi sous le nom de “module testing”, “ program testing”.

Component Testing

- Puisque le test d'un composant se déroule généralement en isolation par rapport aux autres composants du système, des "**Stubs**" et "**Drivers**" sont utilisés pour simuler les interfaces manquantes.



Integration Testing

- Il consiste à tester les interfaces entre les différents composants d'un système
- Il survient après l'intégration d'au moins 2 composants (module) du système à développer "Component Integration testing "
- Ou, il survient après l'intégration du système avec l'environnement physique ou avec d'autre système "System Integration testing "
- Il est réalisé par les testeurs

Integration Testing

- Les testeurs doivent se concentrer sur l'intégration elle-même:
  - Si on doit tester le Module A avec le Module B, on doit tester la communication entre ces deux modules et non pas les fonctionnalités de chacun.
- On définie 2 techniques (Stratégies):
  - le "big bang integration testing" et,
  - le "incremental integration testing"

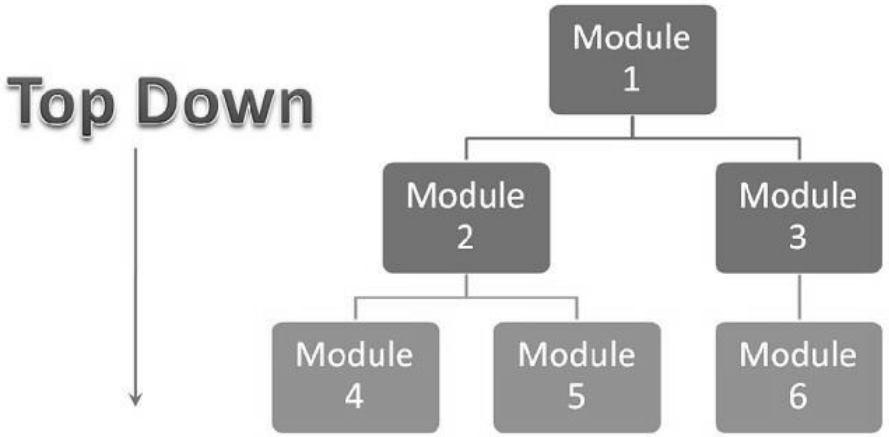
Integration Testing: Big Bang

- Il consiste à intégrer les composants du système simultanément. Du coup, le système est testé comme un ensemble
- **Avantage**: Le développement de tous les composants est terminé avant de commencer les tests
- **Inconvénient**: Difficulté d'isoler l'erreur et de trouver sa vrai cause

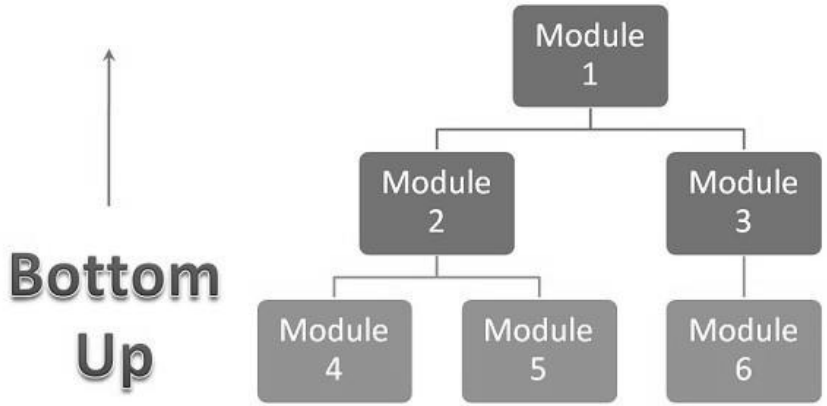
Integration Testing: Incremental

- Il consiste à intégrer les composants un par un et une session de test est réalisée à chaque étape d'intégration
- **Avantage**: Détection rapide et facile de la cause des erreurs
- **Inconvénient**: Consomme énormément de temps et d'effort surtout pour développer les stubs et les drivers

Integration Testing: Incremental



Stub



Driver

Integration Testing: Incremental - Top Down

- Il consiste à tester le logiciel de haut en bas, en suivant le flux de contrôle du système ou sa structure architecturale. les composants ou autres systèmes sont remplacés par des stubs.
- **Avantage**: le produit testé est cohérent vu que les tests suivent le fonctionnement réel du système
- **Inconvénient**: quelques fonctionnalités basiques sont testées tard dans le cycle

Integration Testing: Incremental – Bottom Up

- Il consiste à tester le logiciel de bas en haut, en suivant l'inverse du flux de contrôle du système. les composants ou autres systèmes sont remplacés par des drivers.
- Avantage: Les conditions de test sont facilement créées
- Inconvénient: les drivers sont à implémenter à tous les niveaux du système

System Testing

- Il consiste à tester le comportement de tout le système comme défini dans les requirements specifications
- Assimilé souvent au black box testing bien que ce dernier peut intervenir à n'importe quelle phase de testing
- Il est réalisé par les testeurs

System Integration Testing

- Il consiste à tester les interactions entre les différents systèmes formant la solution complète et peut être fait après le system testing.
- L'objectif est de valider que toutes les dépendences du logiciel sont fonctionnellement correctes et que l'intégrité des données est maintenue à travers tous les modules

Acceptance Testing

- Après les validation et la correction des différents bugs, le système sera livré au client pour l'acceptance testing.
- Le but de ce niveau de test est d'établir une confiance envers le système
- L'acceptance testing peut intervenir à l'installation du système, à l'intégration de modules ou suite à une amélioration fonctionnelle...

Acceptance Testing - Types

- **User Acceptance test:** orienté vers la partie fonctionnelle du système
- **Operational Acceptance test:** (Production acceptance test) le plus commun et il consiste à tester le système en se basant sur les requirements. Il peut inclure aussi les tests des systèmes de backup, disaster recovery, securité...

Acceptance Testing - Types

- **Contract Acceptance testing:** basé sur un contrat préalablement établi entre le client et le fournisseur
- **Compliance acceptance testing:** (regulation acceptance testing) réalisé pour valider la conformité du système avec des normes, des lois ou des exigences gouvernementales.

Alpha Testing

- C’est des tests réalisés lorsque le produit est sur le point d’être finalisé
- Il est effectué par l’équipe de test (parfois épaulée par quelques utilisateurs finaux) et il se déroule généralement sous la supervision de l’équipe de développement.
- A l’issue de ces tests, des changements mineurs peuvent être apportés au produit,

Alpha Testing

- Il se déroule dans des conditions similaires à celles attendues lorsque le produit est en production
- C’est le test final avant la mise en production du produit pour le grand public
- Peut être assimilé à un “internal acceptance testing”

Beta Testing

- Il est exécuté chez le client et dans des conditions similaires à celles de la production
- Les testeurs sont généralement des utilisateurs finaux du système
- On distingue 2 types de Beta tests:
  - fermés: on choisit les testeurs parmi un groupe d'utilisateurs et on leur envoie des invitations)
  - ouverts: tout utilisateur du système est testeur et peut reporter des bugs ou des recommandations

PLAN du cours

1. Introduction
2. Terminologie
3. Les fondamentaux du software testing
4. Les niveaux du software testing
5. **Les types des tests logiciels**
6. Test Management
7. La qualité logicielle
8. Les outils de test et de qualité

Les types des tests logiciels

- On définit principalement 4 types de tests logiciels:
  - Functional testing
  - Non-Functional testing
  - Structural testing
  - Change-related testing

Functional Testing

- Il consiste à tester le volet fonctionnel du système
- il peut être réalisé à n'importe quel niveau du test
- Le test est exécuté conformément aux spécifications fonctionnelles du système

Non-Functional testing

- Il consiste à tester le volet non fonctionnel du système
- Comme le “Functional Testing”, il peut être réalisé à n’importe quel niveau du test
- On définit des domaines de tests non-fonctionnels:

Non-Functional testing

- Reliability testing
- Usability testing
- Maintainability testing
- Portability testing
- Baseline testing
- Compliance testing
- Documentation testing
- Endurance testing
- Load testing
- Performance testing
- Compatibility testing
- Security testing
- Scalability testing
- Volume testing
- Stress testing
- Recovery testing
- Internationalization testing
- Localization testing

Structural Testing

- Il consiste à tester la structure du système ou ses composants
- Il désigne généralement le “white box” testing où les testeurs regardent le comportement du système (ou ses composants) au cours de son exécution
- Il peut être exécuté à n’importe quel niveau de test

Change-related Testing

- A la découverte d'un Defect dans le système, des modifications sont apportées pour les corriger
- Suite à ces changements, des tests sont à refaire vous valider la correction et pour vérifier qu'aucun nouveau bug n'est survenu suite à ces modifications
- Les tests à refaire sont généralement définis suite à une "analyse d'impact"
- Ces tests sont généralement connus sous le nom de "tests de non régression" (non-regression tests)

PLAN du cours

1. Introduction
2. Terminologie
3. Les fondamentaux du software testing
4. Les niveaux du software testing
5. Les types des tests logiciels
6. **Test Management**
7. La qualité logicielle
8. Les outils de test et de qualité

Test Management

- Après l'élaboration des spécifications système, l'équipe de Test et validation commence son travail de planning et de définition de stratégie de test.
- Selon la complexité du projet, il faut spécifier:
  - Niveaux des tests,
  - Types des tests.

Test Management

- Selon les niveaux et les types des tests, il faut fixer les rôles de:
  - L'équipe de dev
  - L'équipe de test
- Les membres et les rôles dans une équipe de test:
  - Test Leader (test manager or test coordinator)
  - Testeurs

Test Management

- Test leader:
  - La définition des objectifs du test, les stratégies de test et les plans de test (test plans).
  - L’estimation et la définition des ressources.
  - Les tests manuels ou les tests automatiques.
  - La conception et la planification des tests cases et la test campaign.
  - La surveillance de la mise en œuvre et l'exécution des tests cases.
  - L’analyse des rapports et des métriques
  - La création et le suivi des Bugs

Test Management

- Testeur:
  - Analyser, examiner et évaluer les exigences et les spécifications
  - Examiner le test plan (Intervenir parfois à la création et la conception des tests cases)
  - Exécuter et contrôler les tests.
  - Evaluer les rapports de test
  - Documenter et Déclarer les Bugs

Test Management: Process

- Steps:
  - Planning and Control
  - Analysis and Design
  - Implementation and Execution
  - Evaluating exit criteria and Reporting
  - Test Closure activities

Test Management: Process

- Planning de test:
  - Déterminer la portée et les risques et identifier les objectifs du test.
  - Définir la stratégie de test
  - Son résultat est un document décrivant l'objectif, l'approche, les ressources et le planning des activités de test:
    - IEEE 829 STANDARD TEST PLAN TEMPLATE

Test Management: Process

- Test design:
  - Après la définition des plan et stratégie du test, on passe à la définitions et la spécification des tests à exécuter:
    - To identify test conditions.
    - To design the tests.
    - To evaluate testability of the requirements and system.
    - To design the test environment set-up and identify and required infrastructure and tools.

Test Management: Test Case

- **Implémentation «** Test Case »:
  - Une description des actions à effectuer pour valider une fonctionnalité du système
- Un test case est défini par:
  - ses conditions initiales
  - les étapes d'exécution
  - le résultat attendu
- Test suite: un ensemble de test cases où la post condition d'un test case est la pré condition du prochain test case à exécuter

Test Management: Exécution

- Pour valider une version du produit, une compagne de test (test compain) est planifiée
- Pour chaque compagne de test, on définie la liste des tests cases (et test suites) à exécuter pour valider la version du produit
- Un test case peut appartenir à plusieurs compagnes de test
- Les test cases d'une compagne de test doivent couvrir toutes les fonctionnalités prévues pour la version du produit. On parle de "couverture de code" (code coverage)

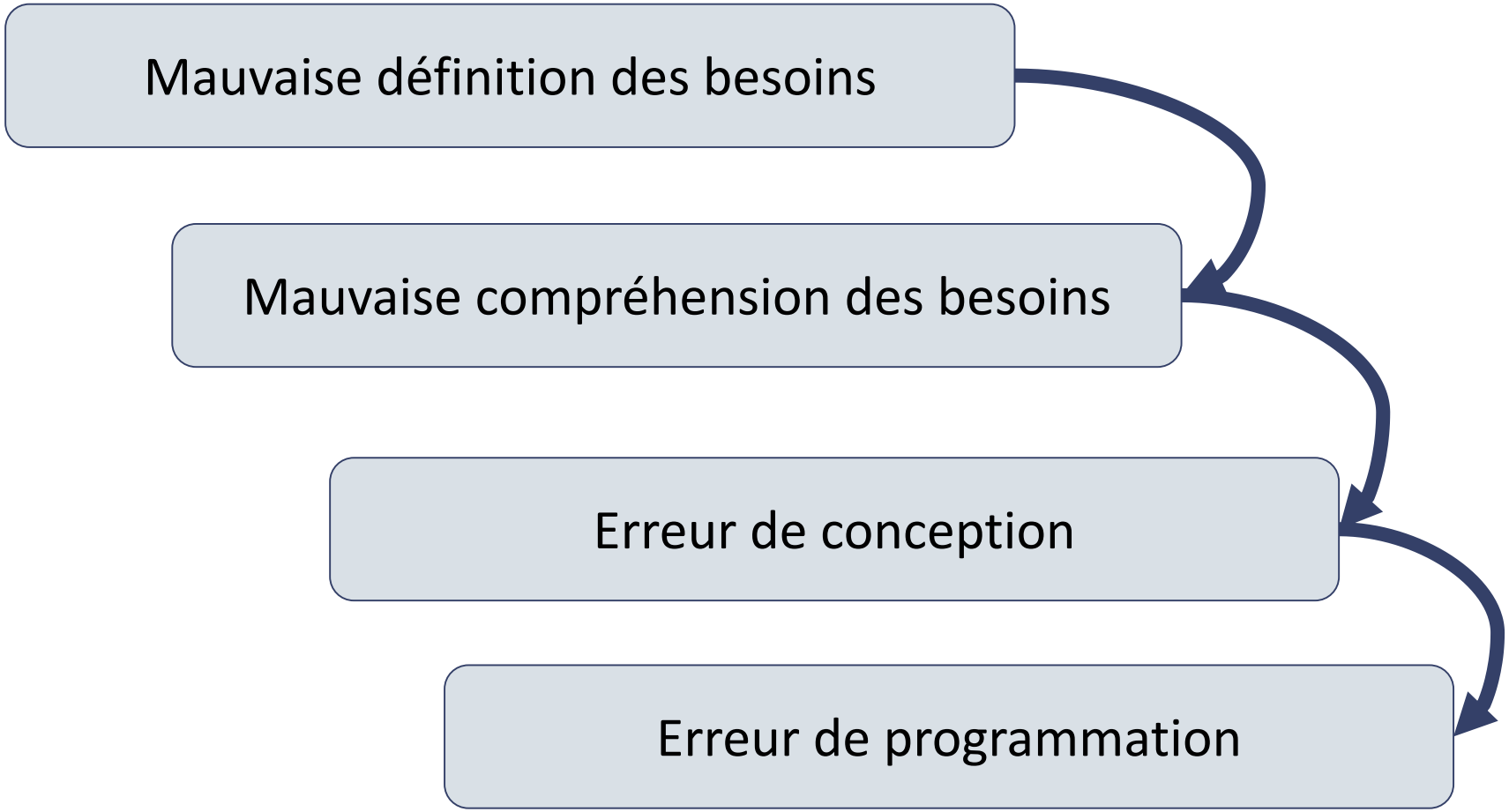
Test Management: Exécution

- Chaque test case est suivi d'un état (status): soit réussi (Successful), soit échoué (Failed)
- Un test case échoué doit faire l'objet d'un "defect" à rentrer dans le système de gestions des defects du projet
- Si le nombre de bugs altère le taux d'acceptance pour une livraison (généralement 97%), l'équipe de développement reprend le travail et une 2ième compagne est planifiée

PLAN du cours

1. Introduction
2. Terminologie
3. Les fondamentaux du software testing
4. Les niveaux du software testing
5. Les types des tests logiciels
6. Test Management
7. **La qualité logicielle**
8. Les outils de test et de qualité

La qualité logicielle: Introduction



La qualité logicielle: Introduction

Phase	Pourcentage des erreurs	Effort de correction
Spec	56%	82%
Conception	27%	13%
Prog	7%	1%
Autres	10%	4%

La qualité logicielle: Définition

- C’est l’appréciation globale d’un logiciel en se basant sur plusieurs indicateurs
- Selon l’IEEE, la qualité logicielle est:
  - Le degré avec lequel un système, un composant ou un processus satisfait à ses exigences spécifiées
  - Le degré avec lequel un système, un composant ou un processus satisfait aux besoins ou attentes de ses clients/usagers.

0% of failure

La qualité logicielle: Définition (Garvin, 1984)

TRANSCENDENT	La qualité c'est une propriété simple, non analysable que nous apprenons à reconnaître seulement par l'expérience.
PRODUCT - BASED	La qualité est une variable précise et mesurable trouvée dans les composants et les attributs d'un produit.
USER - BASED	Les consommateurs ont des besoins différents, et ceux qui répondent le mieux à leurs préférences ont la meilleure qualité
MANUFACTURING - BASED	La conformité aux exigences - Prévenir les défauts (Do things tight the first time).
VALUE - BASED	Définir la qualité en termes de coûts et de prix.

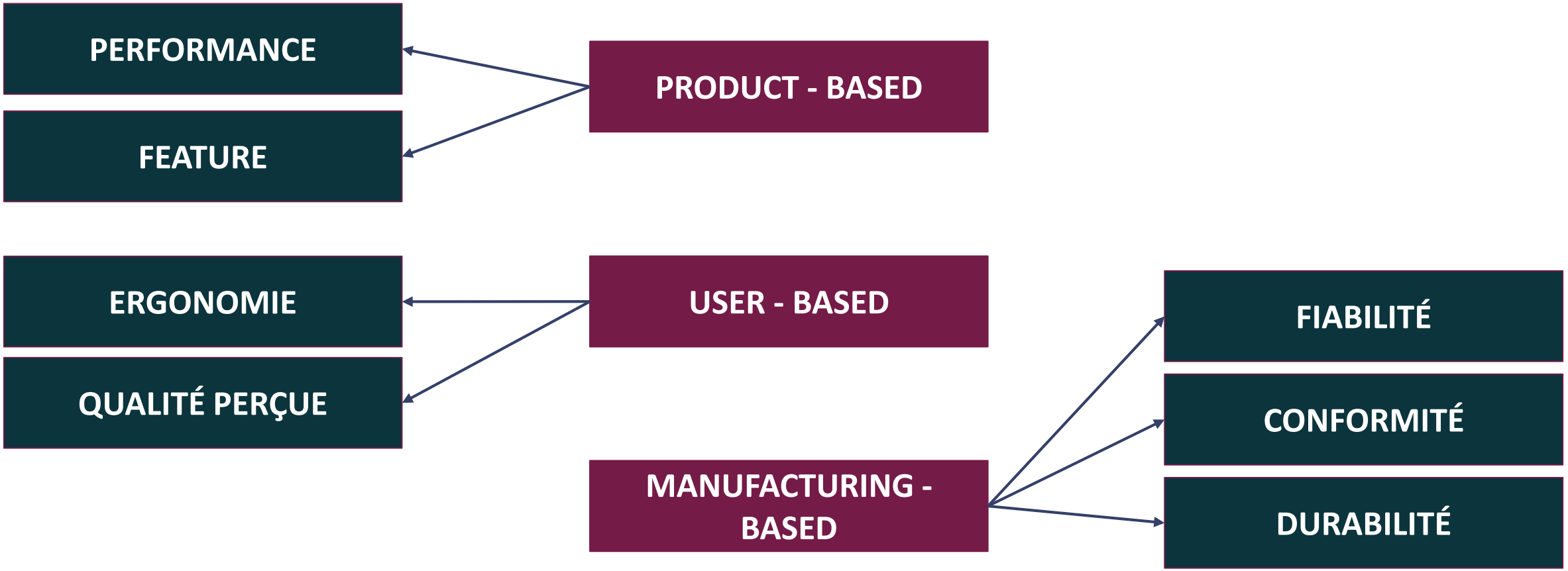
La qualité logicielle: Facteurs

PERFORMANCE	Principales caractéristiques de fonctionnement du produit - l'efficacité avec laquelle, le produit doit attendre son but prévu (temps, mémoire, …).
FEATURE	Les caractéristiques supplémentaires d’un produit.
FIABILITÉ	La probabilité d'échec d'un produit dans une période de temps spécifiée - Tolérance aux pannes, précision, simplicité, etc
CONFORMITÉ	La mesure dans laquelle les caractéristiques de fonctionnement d'un produit correspondent à des normes préétablies (complétude).
DURABILITÉ	La mesure de la durée de vie du produit (Techniquement, Economiquement)

La qualité logicielle: Facteurs

MAINTENABILITÉ	La rapidité et la simplicité de réparation d'un produit.
ERGONOMIE	L'apparence, l'impression, l'apprentissage et la communicabilité avec un produit. Utilisation aisée d'un produit,
QUALITÉ PERÇUE	La manière dont les consommateurs expérimentent l'utilisation du produit.

La qualité logicielle: Facteurs



La qualité logicielle: Mesure

- Définir un ensemble d’indicateurs pour la qualité logicielle
- Définir un processus d’évaluation logiciel et la spécification d'exigences fonctionnelles ou non-fonctionnelles
- Suivre une approche?
- Suivre une méthodologie?

La qualité logicielle: Mesure - Approche

- La norme ISO 9126: définit 6 caractéristiques permettant de décrire la qualité du logiciel:
  - Capacité fonctionnelle
  - Fiabilité
  - Facilité d'utilisation
  - Rendement ou efficacité
  - Maintenabilité
  - Portabilité

La qualité logicielle: Mesure - Méthodologie

- ISO 25041: permet d'intégrer la norme ISO 9126 selon le processus suivant:
  - Fixer les exigences de qualité
  - Etablir un modèle de qualité
  - Fixer les métriques de la qualité
  - Conduire les évaluations

La qualité logicielle: Indicateurs

- Définir un ensemble de métriques, des échelles de mesure et une méthode pour mesurer la qualité d'un produit
- Trouver des moyens de mesurer et de surveiller la capacité et la productivité d'une équipe de développement.
- Les métriques ou les mesures nous donnent une meilleur vision sur l'adéquation et l'efficacité des Tests Cases

La qualité logicielle: Indicateurs

- **Test Case Adequacy:**
  - C’est un indicateur pour évaluer l’équipe de test.
  - Vérifier si l’estimation est adéquate par rapport projet, équipe, etc

$$\textbf{Test Case Adequacy} = \frac{\text{Nbre of actual test cases}}{\text{Nbre of test cases estimated}}$$

La qualité logicielle: Indicateurs

- **Cost of finding a defect in testing (CFDT)**
  - C’est un indicateur pour évaluer l’équipe de développement.
  - Vérifier si l’estimation est adéquate par rapport projet, équipe, etc

$$\textbf{CFDT} = \frac{\text{Total effort spent on testing}}{\text{defects found in testing}}$$

La qualité logicielle: Indicateurs

$$\textbf{Effort Variance} = \left(\frac{\text{(Actual Efforts - Estimated Efforts)}}{\text{Estimated Efforts}} \right) * 100$$