

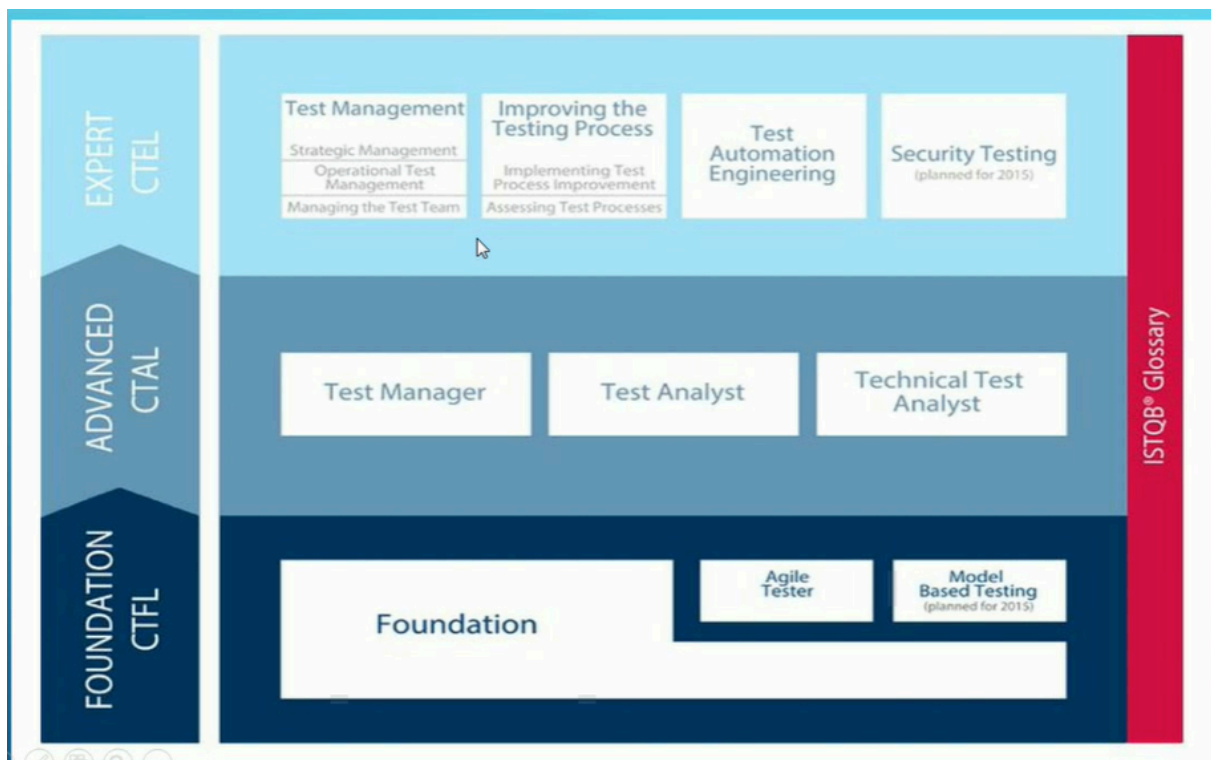
ISTQB CERT PREP:

INTRODUCTION:

ISTQB: International Testing Qualification Board

ISTQB Levels:

Foudation level,Advanced level,Expert Level



ISTQB Foudation Level:

ISTQB® – FOUNDATION LEVEL					
Fundamentals of Testing	Testing Throughout the Software Life Cycle	Static Techniques	Testing Design Techniques	Test Management	Tool Support for Testing
Why is Testing Necessary?	Software Development Models	Static Techniques and the Test Process	The Test Development Process	Test Organization	Types of Test Tools
What is Testing?	Test Levels	Review Process	Categories of Test Design Techniques	Test Planning and Estimation	Effective use of Tools: Potential Benefits and Risks
Seven Testing Principles	Test Types	Static Analysis by Tools	Specification-based Techniques (black-box)	Test Progress Monitoring and Control	Introducing a Tool into an Organization
Fundamental Test Process	Maintenance Testing		Structure-based Techniques (white-box)	Configuration Management	
The Psychology of Testing			Experience-based Techniques	Risk and Testing	
Code of Ethics			Choosing Test Techniques	Incident Management	

FUNDAMENTALS OF TESTING:

1-Why is Testing necessary:

✅ Les logiciels sont partout !

Ils sont essentiels aussi bien dans notre vie quotidienne 🏠 que dans le monde professionnel 💼.

⚠️ D'où viennent les bugs ?

Les erreurs peuvent être causées par l'humain 🧑 ou par des **facteurs externes** 🌐.

🔍 Tester, ce n'est pas juste trouver des défauts !

Un bon test s'assure aussi que le logiciel respecte les **lois, règles et standards** 📖 et garantit une **bonne qualité** 🏆.

💰 Moins de bugs = moins de coûts !

Détecter les erreurs tôt permet de **gagner du temps** ⌚ et d'**éviter les dépassements de budget** 💸.

🔥 **Certains logiciels sont critiques !**

Que ce soit pour la **santé** 🏥 ou l'**intelligence artificielle** 🤖, un bug peut avoir des conséquences graves.

🎯 **Un produit de qualité satisfait les clients !**

Moins de défauts = **meilleure expérience utilisateur** 👍 et respect des attentes ✅.

👉 **Conclusion : Tester, c'est assurer la fiabilité et la performance des logiciels !** 🚀

2- 🧐 Qu'est-ce que le testing ?

🔍 **Tester, ce n'est pas juste exécuter des tests !**

Le testing englobe **toutes les activités** avant et après l'exécution des tests 🔧.

🐛 **Son but ? Trouver des défauts** dans un logiciel et s'assurer qu'il fonctionne correctement ✅.

💡 **Pourquoi tester ?**

👉 Pour avoir **confiance en la qualité du produit** 🎯.

📌 **Le testing intervient à plusieurs étapes :**

- Développement 💻
- Maintenance 🔄
- Acceptation par les utilisateurs 👥

🚫 **Attention ! Debugging ≠ Testing**

Corriger un bug, c'est du **debugging**, mais le testing sert à **détecter les problèmes avant** 🕵️.

🎯 **En bref : Tester, c'est garantir un logiciel fiable, performant et prêt à être utilisé !** 🚀

3- 🏆 Les 7 Principes du Testing

1 Le testing révèle des défauts 🐛

Le fait de **ne pas trouver de bug** ne signifie pas qu'il n'y en a pas 🚩 !

2 Tester tout est impossible ❌

Il est **impossible de couvrir tous les scénarios** d'un logiciel. Il faut adopter **des méthodes intelligentes** 🎯.

3 Tester tôt, c'est économiser 💰 ⌚

Détecter les bugs dès le début **fait gagner du temps, réduit les coûts** et rend le client heureux 😊.

4 Les bugs aiment se regrouper 🔍

Les défauts ont tendance à **apparaître en clusters**, souvent dans certaines parties du code.

5 Le paradoxe du pesticide 🪴

Refaire toujours les **mêmes tests** finit par les rendre **inefficaces** ! Il faut **les mettre à jour régulièrement** 🔄.

6 Le testing dépend du contexte 🏗️

Chaque logiciel est unique et nécessite une **stratégie de test adaptée** 🧩.

7 L'absence d'erreur ne garantit pas un bon produit 🚦

Un logiciel peut être **sans bug**, mais **s'il ne répond pas aux besoins des utilisateurs, il reste inutile** !

👉 **En résumé : Un bon testing va bien au-delà de la chasse aux bugs, c'est une stratégie essentielle pour garantir un produit fiable et performant ! 🚀**

4-🔍 Le Processus Fondamental du Testing:

1 📝 Planification et Contrôle des Tests

- Définir les **objectifs et le périmètre** du test 🎯
- Rédiger les **spécifications et stratégies de test** 📄
- Suivre et ajuster le **progrès du projet** 📊

2 🔍 Analyse et Conception des Tests

- Examiner la **base des tests** 🤖

- Prioriser les **scénarios de test** ⌚
- Définir les **tests de haut niveau** et identifier les **données nécessaires** 📁

3 ⚙️ Implémentation et Exécution des Tests

- Rédiger les **cas de test** et les **scripts automatisés** 💻
- Exécuter les tests, comparer les **résultats attendus/réels** ✅
- **Enregistrer les bugs** et effectuer des **tests de régression** 🔁

4 📢 Évaluation des Critères de Sortie et Reporting

- Vérifier si les **critères de sortie sont atteints** 🏁
- Résumer les résultats pour les **managers et clients** 📁
- Décider si d'autres tests sont nécessaires 🔁

5 ➡️ Clôture des Tests

- Vérifier si toutes les **livraisons ont été effectuées** 📦
- **Clôturer les défauts** identifiés et conserver les **données de test** pour l'avenir 🗝️
- Analyser le processus pour **s'améliorer sur les prochaines versions** 🚀

👉 **Un bon testing suit un processus bien défini pour garantir un logiciel fiable, performant et sans surprise ! 🎯**

5- 🧠 La Psychologie du Testing

1 🧑 Développeurs vs. Testeurs : Deux Mentalités Différentes

- **Développer** est un processus **constructif** 🏗️
- **Tester** est un processus **destructif** 🔨 (chercher les failles !)
- Leur **vision évolue avec le temps** 🔁

2 ⚖️ Niveau d'Indépendance du Testing

- Un **développeur teste son propre code** 🔧
- Un **autre développeur de la même équipe teste** 🧑
- Un **testeur de la même organisation** analyse les bugs 🔍
- Un **testeur externe** apporte une nouvelle perspective 🌐

3 🗣️ Une Communication Constructive lors du Reporting des Bugs

- Restez **pro** et **ne blâmez pas** les développeurs 🤝
- **Mettez-vous à leur place** avant de critiquer 🤔
- Assurez-vous que le **bug est bien compris** pour être corrigé ✅

4 🎯 Un Objectif Commun : Un Produit de Qualité

- Développeurs et testeurs veulent **le même but** : un **logiciel fiable** 🚀
- **Chaque bug trouvé pendant le test fait gagner du temps** en évitant des problèmes plus tard ⌚

👉 **Tester, c'est bien plus que chercher des erreurs : c'est une collaboration intelligente pour garantir un produit au top !** 🔥

6- ⚖️ Le Code d'Éthique du Testing

1 🌐 Public

- Assurer que les logiciels sont **sûrs et fiables** pour tous 🏆
- Prendre en compte **l'impact sur la société** et la sécurité 🛡️

2 👤 Client et Employeur

- Travailler avec **intégrité et transparence** 🤝
- Respecter la **confidentialité des données** 🗂️

3 🛠️ Produit

- Viser un **logiciel de qualité** sans compromis 🚀
- Tester avec rigueur pour **minimiser les défauts** 🎯

4 ⚖️ Jugement

- Prendre des décisions basées sur **les faits et l'éthique** ✅
- Éviter les **conflits d'intérêts** ⚠️

5 🇫🇷 Management

- Promouvoir un **environnement de travail équitable** 😊
- Fournir aux équipes les **ressources et le soutien** nécessaires 🎯

6 🎓 Profession

- Continuer à **apprendre et évoluer** 📖
- Suivre les **meilleures pratiques de l'industrie** 🏆

7 🤝 Collègues

- Respecter et collaborer avec les **autres professionnels** 💡
- Partager les connaissances pour **élever le niveau collectif** 🔄

👉 **Un bon testeur n'est pas seulement un chasseur de bugs, mais aussi un garant de l'éthique et de la qualité ! 100**

Fundamental Testing throughout the software life cycles of testing:

1- 💧 Le Modèle Waterfall (Cycle en Cascade):

📌 **Un développement qui coule comme une cascade !** Ce modèle suit un processus linéaire où chaque phase doit être terminée avant de passer à la suivante. Il est idéal pour les **projets courts avec des exigences bien définies** 📖✅

◆ Les étapes du modèle Waterfall :

- 1 📌 **Collecte des besoins** – Définition claire des exigences du projet.
- 2 🖥️ **Conception du système** – Planification de l'architecture et du design.
- 3 💻 **Implémentation** – Développement du logiciel.
- 4 🔍 **Tests** – Vérification et correction des bugs.
- 5 🚀 **Déploiement** – Mise en production du logiciel.
- 6 🛠️ **Maintenance** – Corrections et mises à jour après la livraison.

✅ **Avantages du modèle Waterfall**

- ✓ **Facile à comprendre et à suivre** 📖
- ✓ **Des jalons clairs à chaque phase** ⌚

✓ **Idéal pour les petits projets** sans complexité 🎯

✓ **Bonne documentation** pour chaque étape 📝

✗ **Inconvénients du modèle Waterfall**

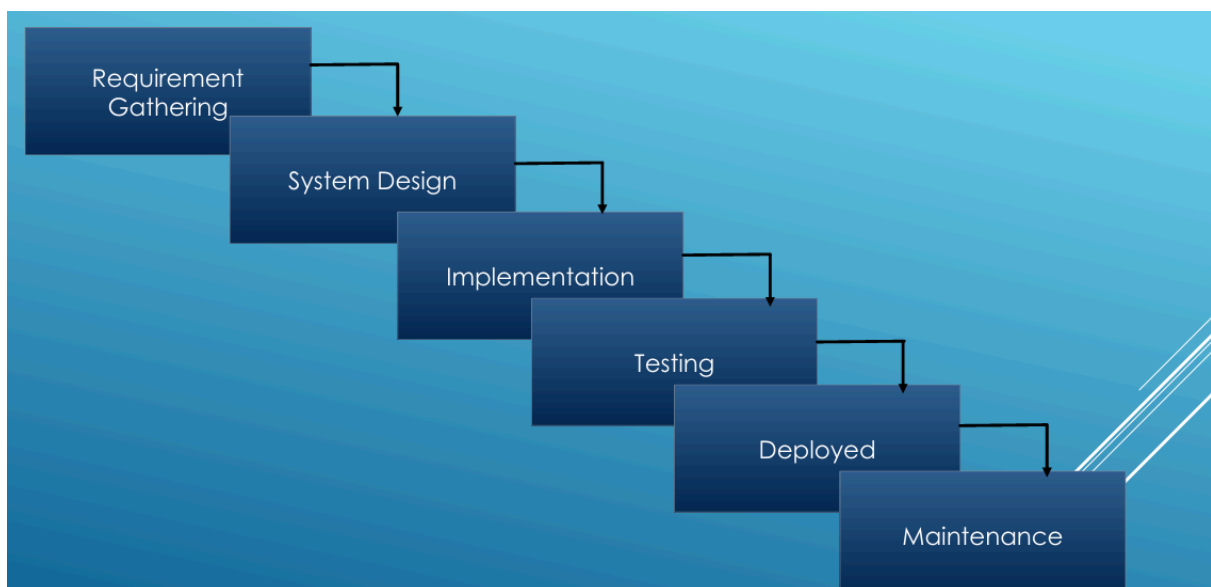
✗ **Peu flexible** – Pas adapté aux projets avec des exigences changeantes 🔄

✗ **Difficile de mesurer l'avancement** dans les étapes intermédiaires 📉

✗ **Dépend fortement des exigences initiales** – Une erreur au début peut compromettre tout le projet ⚠️

✗ **Les clients veulent souvent des modifications après avoir vu le logiciel** – ce qui oblige à tout recommencer 😞

👉 **Un modèle structuré mais rigide, parfait pour des projets simples mais risqué pour des besoins évolutifs ! 🔍**



2- **Vérification vs Validation : Quelle différence ?**

Vérification = "Est-ce qu'on construit le système correctement ?" 🏗️

Validation = "Est-ce qu'on construit le bon système ?" 🎯

Vérification (Avant exécution du logiciel)

✓ Vérifie si les **livrables respectent les exigences** 📄

✓ Utilise des **inspections, revues et walkthroughs** 🧐

✓ Se fait **sans exécuter le logiciel** 🚫💻

🔍🔧 **Validation** (Après exécution du logiciel)

✓ Vérifie si le **logiciel répond aux attentes du client** 👤

✓ Utilise des **tests fonctionnels, régression, système, UAT...** 🔄

✓ Se fait **en exécutant le logiciel** ▶️💻

👉 La **vérification** s'assure que l'on suit les bonnes étapes, tandis que la **validation** s'assure que le produit final satisfait le client ! 🎯✅

3-🔹 Le Modèle V (V-Model)

📌 **Aussi appelé Modèle de Vérification et Validation**, le **V-Model** associe chaque phase de développement à une phase de test correspondante, garantissant une approche structurée pour trouver les bugs dès leur entrée dans le produit. Il est une **extension du modèle Waterfall**, mais avec un processus de test parallèle qui dépasse les limitations du Waterfall 🚀.

✅ **Avantages du modèle V :**

✓ **Suivi proactif des défauts** – Les défauts sont détectés tôt, ce qui permet d'économiser du temps et des ressources 💡

✓ **Gestion facile** – Chaque phase a des objectifs et buts bien définis 🎯

✓ **Produit de qualité** – Le modèle suit un processus strict pour mesurer la productivité, l'efficacité et la qualité du logiciel 📊

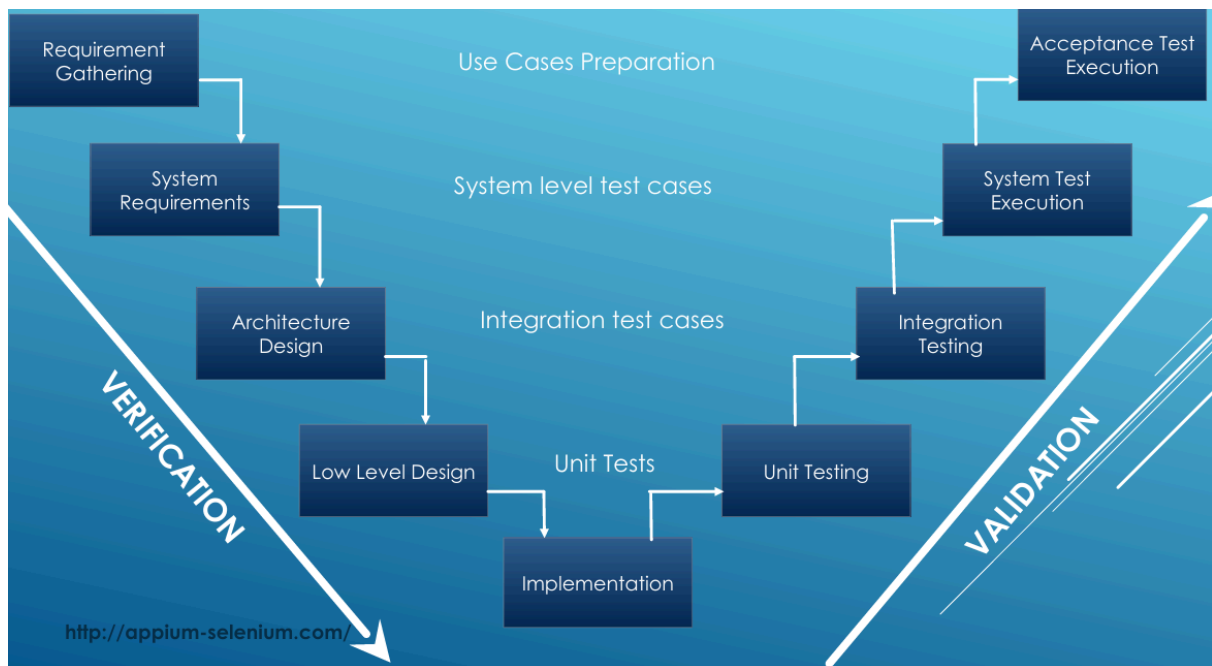
❌ **Inconvénients du modèle V :**

❌ **Pas de prototypes précoces** – Le logiciel est développé pendant la phase d'implémentation, ce qui empêche des tests avant cette phase ⌚



❌ **Modifications difficiles** – Si des changements surviennent en cours de projet, les documents de test et d'exigences doivent être mis à jour 📄🔄

❌ **Ressources nécessaires** – Ce modèle demande beaucoup de ressources humaines et matérielles 🧰

👉 Le **modèle V assure une couverture complète des tests tout en garantissant la qualité du produit, mais il peut être rigide et exigeant en ressources.** 📈






4- **Le Modèle Itératif (ou Modèle Incrémental)**

 Le **Modèle Itératif** divise le projet en petits modules qui peuvent être livrés au fur et à mesure. Une version fonctionnelle du logiciel est produite dès le premier module, et chaque version suivante ajoute de nouvelles fonctionnalités jusqu'à ce que le système complet soit atteint. Ce modèle est particulièrement efficace lorsqu'il s'agit de travailler avec de nouvelles technologies .

Les étapes du modèle itératif :

- 1 **Exigences → Conception → Tests → Livraison & Maintenance**
- 2 Répétition de ces étapes avec de nouveaux modules à chaque itération jusqu'à la version finale.

Avantages du modèle itératif :

- ✓ **Livraison rapide** – Le logiciel fonctionnel est disponible dès les premières phases du projet 
- ✓ **Changements moins coûteux** – Plus facile de modifier les exigences par rapport à d'autres modèles 
- ✓ **Développement et tests simplifiés** – Avec de petites itérations, il est plus facile de développer et tester 

✓ **Feedback rapide** – Le client peut donner son avis rapidement à chaque étape ➡

✓ **Itérations parallèles possibles** – Plusieurs itérations peuvent être menées simultanément ↻

✗ **Inconvénients du modèle itératif :**

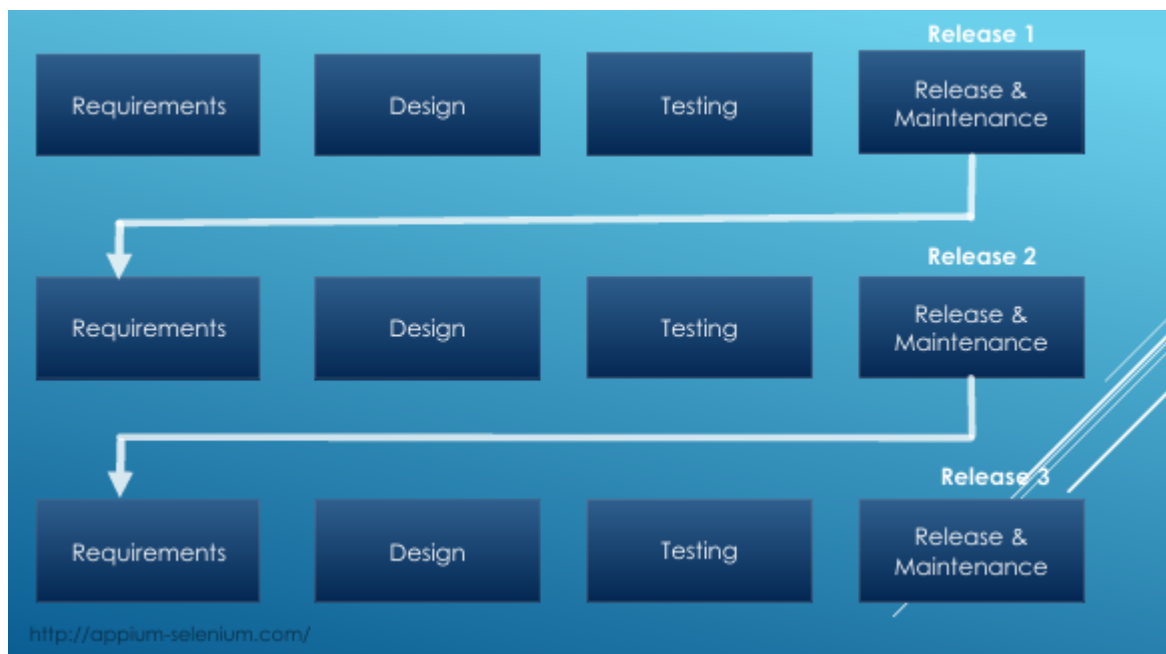
✗ **Ressources plus nombreuses** – Ce modèle nécessite davantage de ressources humaines et matérielles 🔧

✗ **Gestion de projet complexe** – Un gestionnaire qualifié est nécessaire pour éviter que le coût du projet n'augmente 📁

✗ **Architecture de projet définie trop tôt** – Si le projet commence avec une architecture complète, des problèmes peuvent surgir plus tard 🔍

✗ **Coût plus élevé** – Le modèle itératif peut être plus coûteux que le modèle Waterfall 💰

👉 **Le modèle itératif est idéal pour les projets où la flexibilité et l'amélioration continue sont importantes, mais il nécessite une gestion stricte et des ressources adaptées.** 📈



5- ♦ Modèle de Développement d'Applications Rapides (RAD)

📌 Le **modèle RAD** est une variation du modèle incrémental, conçu pour livrer rapidement un logiciel aux utilisateurs. Ce modèle accorde moins de temps à la planification et plus de temps à l'**intégration et au développement** du produit 🏗️.

◆ Les étapes du modèle RAD :

1 Exigences → Conception Utilisateur → Phase Constructive → Phase de Transition (Cutover)

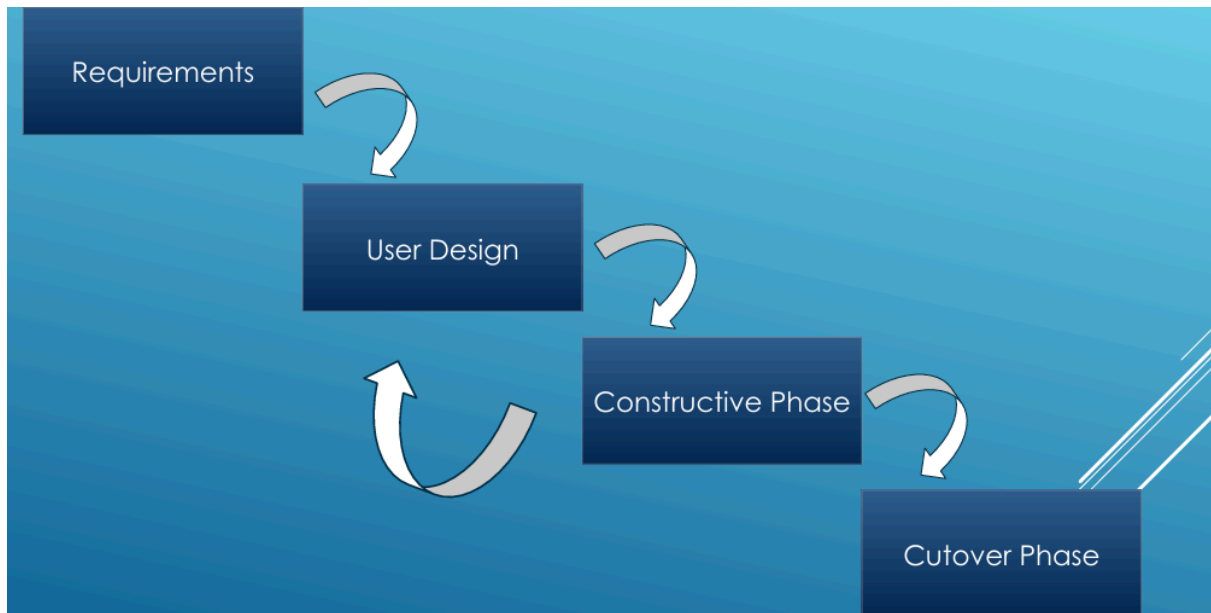
✅ Avantages du modèle RAD :

- ✓ **Projets livrés à temps et dans le budget** – Le modèle RAD permet de terminer les projets plus rapidement 💡
- ✓ **Meilleure qualité** – Une collaboration plus étroite entre les développeurs et les utilisateurs améliore la qualité du produit final 🏆
- ✓ **Réduction du temps de développement** – Le temps de développement est considérablement réduit grâce à des méthodes efficaces ⌚
- ✓ **Identification précoce des problèmes** – Les problèmes sont repérés tôt dans le processus de développement 🔍
- ✓ **Développement rapide** – RAD permet une conception et un développement accélérés 🚀
- ✓ **Respect du budget** – Les projets sont souvent complétés dans les délais et le budget alloué 💰



❌ Inconvénients du modèle RAD :

- ❌ **Effort accru des ressources** – Les ressources doivent être impliquées à la fois dans la collecte des exigences et le développement du logiciel 🔄
- ❌ **Ressources expérimentées nécessaires** – Les outils et les exigences changeants nécessitent une expertise pour éviter des erreurs de conception 🧰
- ❌ **Conception insuffisante** – Les développeurs se concentrent souvent trop sur la fonctionnalité et moins sur la conception globale, ce qui peut entraîner des défauts de structure 🏠

👉 Le modèle RAD est particulièrement utile lorsque la rapidité de livraison est cruciale, mais il nécessite des ressources qualifiées et une gestion soignée. ⚡



6- **Développement Agile (Agile Development)**

 **Agile** permet de livrer des résultats plus rapidement, en **réduisant le temps de mise sur le marché**. Il implique également le client pour s'assurer que le produit livré correspond exactement à ses besoins, et aide les équipes à **mitiger les risques dès les premières étapes du cycle de vie du produit** .

Méthodologie Agile Scrum :

1 Méthodologie Scrum – Scrum est une approche de développement logiciel **itérative et incrémentale** utilisée pour gérer le développement des produits.

2 Backlog Grooming – Le client définit les besoins, que nous transformons en **User Stories** (Backlog du produit).

3 Rôles Scrum :

- **Product Owner** – Responsable de la vision du produit et de la gestion du backlog.
- **Équipe Scrum** – Développeurs et testeurs qui travaillent sur les User Stories.
- **Scrum Master** – Facilite l'équipe et assure que les processus Scrum sont suivis.

4 Sprint Planning – Une fois le backlog préparé, l'équipe planifie le travail à réaliser lors du **Sprint**.

5 Sprints – Chaque itération dure entre 2 et 3 semaines, incluant la **collecte des exigences**, la **conception**, le **développement** et les **tests**.

6 Réunion quotidienne – Chaque membre de l'équipe répond à trois questions :

- Qu'ai-je fait hier pour aider à atteindre l'objectif du sprint ?
- Que vais-je faire aujourd'hui pour aider à atteindre l'objectif du sprint ?
- Y a-t-il des obstacles empêchant l'atteinte de l'objectif du sprint ?

7 Réunion de rétrospective – À la fin de chaque sprint, l'équipe réfléchit à ce qui a bien fonctionné et ce qui pourrait être amélioré pour le sprint suivant. Après les retours du client, de nouvelles exigences sont rassemblées pour le prochain sprint.

✓ **Avantages du Modèle Agile :**

- ✓ **Amélioration de la qualité** des livrables grâce aux itérations fréquentes.
- ✓ **Suivi quotidien** de l'avancement du projet, ce qui permet de garder le cap.
- ✓ **Adaptabilité** aux changements tout au long du développement.
- ✓ **Livraison fréquente de versions fonctionnelles** du produit qui peuvent être mises en production rapidement.
- ✓ **Satisfaction du client** car il a un contrôle constant sur l'évolution du produit.

✗ **Inconvénients du Modèle Agile :**

✗ **Documentation limitée** – Moins de documentation formelle par rapport à d'autres modèles.


✗ **Ressources expérimentées nécessaires** – Le modèle exige des équipes hautement qualifiées et proactives.

✗ **Plus de testeurs nécessaires** pour assurer une couverture adéquate pendant les sprints.


✗ **Livraisons fréquentes** peuvent entraîner des cycles de tests et de mises à jour continus, ce qui peut être lourd à gérer.

👉 **En résumé, Agile est idéal pour des projets dynamiques où la flexibilité, la collaboration continue et la livraison rapide sont essentielles.** 🚀


7- **Component Testing**

- **Test unitaire** ou **Module testing**.
 - Chaque module est testé seul (ex : tester une page d'un site web).
 - Fait après les tests unitaires des développeurs. 
-

Stubs & Drivers

- **Stubs** : Composants fictifs pour remplacer ceux manquants.
 - **Drivers** : Simulent des appels à un module. 
-

Pourquoi c'est important ?

- **Détecter les erreurs tôt** 
- **Gagner du temps** pour l'intégration. 

8- **Integration Testing**

L'**integration testing** se concentre sur les interactions entre différents composants du système. Voici une explication plus détaillée :

Objectifs

1. **Interfaces entre composants** : Vérifier que chaque composant peut interagir correctement avec les autres.
 2. **Interactions avec d'autres parties du système** : Tester la communication entre les composants et les systèmes externes.
 3. **Systèmes de fichiers et matériel** : Assurer que tout fonctionne ensemble, du logiciel au matériel.
-

Quand se fait le test ?

- **Après** les tests de composants (unit testing).
 - **Vérifie la communication** entre deux composants, mais **pas** leur fonctionnalité individuelle.
-

Test de performance

- Parfois intégré dans les tests d'intégration pour vérifier que les **performances** du système sont maintenues lorsque les composants interagissent ensemble.
-

Types d'approches

1. Big Bang

- **Tout est intégré en même temps**, puis testé.
- **Inconvénient** : Difficile de repérer la cause des erreurs car tout est intégré d'un coup.

2. Incremental

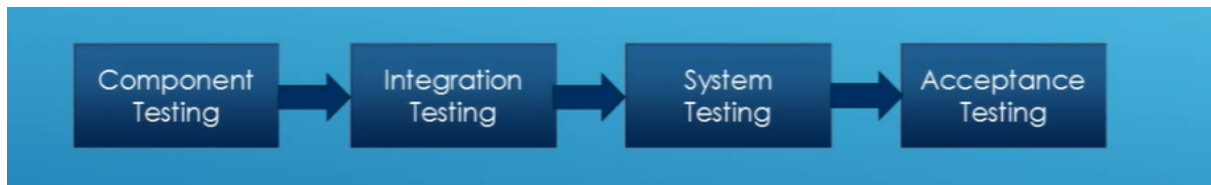
- **Composant par composant** est intégré et testé.
- **Inconvénient** : Prend plus de temps car des outils comme des "stubs" et des "drivers" sont nécessaires.

3. Sandwich Testing


- **Commence au milieu** et se déplace simultanément vers le haut et vers le bas du système.
 - Mélange les deux approches (top-down et bottom-up).
-

Approches incrémentales

- **Top-Down** :
 - **Composants** intégrés du **haut vers le bas**.
 - Utilisation de **stubs** pour simuler les composants non intégrés.
- **Bottom-Up** :
 - **Composants** intégrés du **bas vers le haut**.
 - Utilisation de **drivers** pour tester les composants intégrés au fur et à mesure.




9- **System Testing**

- **Après** l'intégration et avant l'acceptation.
- Vérifie **tout le système** (comportement global).
- C'est le **dernier test** avant la livraison. 

But

- Trouver **le plus de défauts possible**.
- Tester **fonctionnalités** et **performances** du système.

Environnement contrôlé

- Test en **conditions réelles** (comme en production).
- Vérifie la conformité aux **spécifications**. 

10- **Acceptance Testing**

L'**acceptance testing** est crucial pour vérifier si le système répond aux besoins du client avant sa mise en production. Voici les points clés :

Quand se passe-t-il ?

- **Après** le **system testing**.
- **Test effectué dans un environnement similaire à la production**, pour s'assurer que tout fonctionne comme prévu.

But principal

- **Valider** que le système est prêt à être utilisé par le client. Cela donne **confiance** dans la qualité et l'efficacité du produit.

Caractéristiques du Test

- **Black Box Testing** : Les testeurs ne regardent pas le code, mais vérifient si le système fonctionne comme prévu.
 - **Test réalisé par des testeurs indépendants**, les utilisateurs finaux, et d'autres parties prenantes.
 - **Validation** : Le système doit être jugé comme **apte à l'utilisation**.
-

Questions clés durant l'acceptance testing

1. Le système peut-il être mis en production ?
 2. Quels sont les risques d'affaires qui demeurent ?
 3. Le développement a-t-il respecté ses engagements ?
-

Types de Tests d'Acceptation

1. **Compliance Acceptance Testing** : Vérifie si le système respecte les contrats, les lois, et les réglementations applicables.
 2. **Operational/Production Acceptance Testing** : Vérifie si le système répond aux exigences de fonctionnement pour être mis en production.
-

Tests à différents niveaux

- **Produit COTS (Commercial Off-The-Shelf)** : Le test d'acceptation se fait lors de son installation ou de son intégration.
- **Test d'acceptation de la facilité d'utilisation** : Cela peut se faire pendant les tests de composants.
- **Test d'acceptation pour de nouvelles améliorations fonctionnelles** : Peut se faire avant le system testing.

11- Functional Testing

Le **functional testing** permet de vérifier ce que fait un système par rapport aux besoins du client, en se concentrant sur **les fonctionnalités** spécifiques du produit.

But principal 🎯

- Vérifier **la fonctionnalité** du système (c'est-à-dire ce que le système est censé faire).
-

Caractéristiques clés 📝

- **Black Box Testing** : L'accent est mis sur **l'entrée et la sortie**, sans regarder le code sous-jacent.
 - **Conformité aux exigences** : Le test se concentre sur la satisfaction des **exigences du client**.
 - **Assurance qualité** : Ce processus fait partie du processus global d'assurance qualité.
 - **Basé sur la documentation** : Les tests sont réalisés selon **les spécifications du produit** et la compréhension des fonctionnalités par le testeur.
-

Méthodes de test :

1. **Basé sur les exigences** : Utilisation des spécifications des utilisateurs comme référence, avec **priorisation des tests**.
 2. **Basé sur les processus métier** : Utilisation de **cas d'utilisation** développés à partir des flux métier.
-

Étapes du testing fonctionnel 🔄

1. **Vérification de la fonctionnalité** : Vérifier comment le produit doit fonctionner.
 2. **Création des données de test** : Générer les données nécessaires pour les tests.
 3. **Détermination des résultats attendus** : Définir ce que l'on attend comme résultat.
 4. **Exécution des tests** : Lancer les tests pour voir si le système répond comme prévu.
 5. **Comparaison des résultats** : Comparer les résultats réels aux résultats attendus pour vérifier le bon fonctionnement.
-

Conclusion : Le testing fonctionnel est crucial pour s'assurer que le produit répond aux exigences du client et fonctionne correctement dans un contexte réel. 👍

12-🔧 Non-Functional Testing

Le **non-functional testing** vérifie **comment bien** le produit fonctionne en se concentrant sur des **caractéristiques qualitatives** plutôt que sur les fonctionnalités.

But principal 🎯

- Tester **comment** le produit fonctionne, par exemple, sa **performance**, son **efficacité** et sa **fiabilité**.

Types de Non-Functional Testing 🔧

1. **Performance** : Teste la **vitesse** et la **réactivité** du système, comparée à d'autres systèmes.
2. **Charge** : Teste comment le logiciel se comporte pendant les heures **normales** et **de pointe**.
3. **Stress** : Teste le système au-delà de ses **limites** pour voir comment il réagit en cas de surcharge.
4. **Utilisabilité** : Vérifie si l'utilisateur **se sent à l'aise** avec l'interface du logiciel.
5. **Maintenabilité** : Teste la facilité avec laquelle le produit peut être **maintenu** et **modifié**.
6. **Fiabilité** : Vérifie si le logiciel reste **fiable** sous différentes conditions et donne les bons résultats.

Conclusion

Le **non-functional testing** permet de s'assurer que le produit **répond aux attentes qualitatives** des utilisateurs et peut **fonctionner efficacement** dans un environnement réel. 👍

13- Testing lié aux changements:

Les tests liés aux changements sont essentiels pour **confirmer** que les modifications effectuées dans le système ne créent pas de nouveaux problèmes et pour **valider** que les défauts précédemment identifiés ont bien été corrigés.

Confirmation Testing ou Retesting

- **But : Re-exécuter les tests** pour confirmer que le défaut a été **réellement corrigé**.
 - **Méthode** : L'exécution du test doit se faire de la **même manière** que lors de la découverte du défaut.
-

Regression Testing

- **But** : Vérifier que la correction d'un défaut n'a pas introduit de **nouveaux défauts** dans le système.
 - **Test suite** : De nombreuses organisations créent un **pack de tests de régression** pour automatiser ce processus.
 - **Automatisation** : La régression doit être **automatisée** pour rendre le processus plus rapide et fiable.
 - **Quand le faire** : Les tests de régression sont effectués chaque fois que le logiciel change, soit à la suite de corrections, de nouvelles fonctionnalités ou de changements dans l'environnement.
-

Conclusion

Les tests liés aux changements, comme le **retesting** et la **régression**, permettent de garantir que les modifications apportées ne perturbent pas le bon fonctionnement du système et que les défauts sont effectivement corrigés sans en introduire de nouveaux. 🙌

14- Maintenance Testing:

Le **Maintenance Testing** est un type de test réalisé pendant le cycle de vie du logiciel pour **gérer les changements**, les améliorations et les migrations du système.

Qu'est-ce que le Maintenance Testing ?

- **But** : Il s'agit des tests effectués pendant les **cycles de modification** ou de **mise à jour** du logiciel.
 - **Différence** : Ce n'est pas la même chose que le **maintainability testing** (test de maintenabilité), qui se concentre sur la capacité du système à être facilement maintenu.
-

Composants clés du Maintenance Testing :

1. **Tester les changements** : Toutes les modifications apportées au logiciel doivent être **testées minutieusement**.
 2. **Ne pas affecter la fonctionnalité existante** : Les changements doivent être testés pour s'assurer qu'ils ne **perturbent pas** les fonctionnalités déjà existantes, ce qui nécessite des **tests de régression**.
-

Testware et Test Basis

- **Testware** : Ensemble des **documents** produits lors des tests, y compris les **cas de tests**, les **plans de tests**, et les **rapports de tests**.
 - **Test Basis** : La **source d'information** nécessaire pour créer des cas de tests et analyser les résultats des tests.
-

Types de Modifications

1. **Modifications Planifiées** :
 - **Perfective** : Adaptation aux souhaits des utilisateurs, comme l'ajout de nouvelles fonctionnalités ou l'amélioration des performances.
 - **Adaptive** : Adaptation du logiciel aux changements **environnementaux**.
-

En résumé :

Le **Maintenance Testing** garantit que les modifications et améliorations apportées au logiciel n'introduisent pas de nouveaux défauts et que les

fonctionnalités existantes sont préservées. Il implique également l'analyse des impacts des changements sur le système global. 🖥️🔍

STATIC TESTING:

Le

Static Testing est une méthode de vérification qui permet de trouver des erreurs sans exécuter le logiciel, en se concentrant sur l'analyse des documents et du code. Il contribue à une meilleure compréhension du projet et à des économies de coûts. ✅

1-🔍 Roles and Responsibilities in Reviews

Les

revues sont un processus structuré impliquant différents rôles : le **modérateur** dirige, l'**auteur** rédige et améliore, le **scribe** prend des notes, les **relecteurs** trouvent des défauts, et le **manager** supervise et décide de la mise en œuvre des revues. Ce processus favorise l'amélioration continue de la qualité des produits. 🛠️

2-Types de Revues dans les Tests Logiciels 📝

Les **revues** sont essentielles pour examiner et améliorer la qualité des documents et du code. Voici les principaux types de revues utilisés dans les projets de développement.

2-1. Walkthrough (Revue Informelle) 🚶

- **Dirigé par l'auteur** : L'auteur **explique** le document étape par étape à tous les participants.
- **Objectif** : Assurer une **compréhension commune** du document par tous.

- **Documents concernés** : Utilisé pour des documents **de haut niveau**, comme les documents de **spécifications**.
 - **Relecteurs divers** : Participants de **différents niveaux** dans l'organisation pour enrichir la revue.
 - **Transfert de connaissances** : Un moyen idéal pour **partager des connaissances** avec d'autres utilisateurs.
-

2-2. Technical Review (Revue Technique)

- **Revue formelle mais moins stricte qu'une inspection** : Revue plus **informelle** comparée à l'inspection.
 - **Dirigé par un modérateur** : Le **modérateur** coordonne la revue et anime les discussions.
 - **Focus technique** : Se concentre sur le **contenu technique** du document ou du code.
 - **Participants** : **Architectes, concepteurs**, et autres **experts techniques**.
 - **Peer Review** : Effectuée **entre pairs**, sans la gestion hiérarchique.
-


2-3. Inspection (Inspection Formelle)

- **Revue la plus formelle** : La **plus structurée** et formelle des revues.
 - **Dirigée par un modérateur formé** : Le processus est dirigé par un **modérateur** qualifié.
 - **Enregistrement précis** : Un **scribe** ou un **rapporteur** consigne toutes les discussions et les **défauts** relevés.
 - **Suivi rigoureux** : Toutes les **étapes** de la revue sont suivies à la lettre pour garantir la qualité maximale.
-

3-TEST DESIGN TECHNIQUES:







3-1-Identification des Conditions de Test et Conception des Cas de Test

1. Formalité de la Documentation des Tests





- La **formalisation** de la documentation des tests dépend du **projet**, de la **société**, etc. 

3-2. Analyse des Tests : Identification des Conditions de Test




- **Condition de test** : Un élément que nous pouvons tester, par exemple, pour un **document de spécification**, son contenu serait une **condition de test**.
- **Base de tests** : Les tests sont écrits à partir d'une **base de tests**. Cette base peut être un **email**, un **document**, une **vidéo**, etc.   
- **Tests exhaustifs** : Tester **tout** n'est pas possible, il faut donc procéder à une **priorisation**. 
- **Techniques de test** : Des **processus intelligents** aident à guider la sélection des tests à réaliser. 
- **Traçabilité** : Lier les conditions de test à leurs sources dans la base de tests, ce qui est utile lors de **changements** de spécifications, de **tests échoués** et pour **vérifier** que toutes les exigences sont couvertes. 

3-3. Conception des Tests : Spécification des Cas de Test

- La **documentation des cas de test** dépend de l'expérience du testeur. 
- Le **résultat attendu** doit être connu **avant** d'écrire le test.
- **Écrire d'abord les parties critiques** (les tests "effrayants"). 
- L'information sur le comportement correct du système est appelée **Oracle**. 
- Les cas de test doivent être **significatifs**, et les tests **négatifs** (tests de défaillance) sont également importants. 

3-4. Mise en œuvre des Tests : Spécification des Procédures ou Scripts de Test

- **Regroupement des cas de test** : Les cas de test doivent être organisés en **groupes**.
- Les **procédures de test** ou les **scripts** sont les **étapes** à suivre pour exécuter les tests. 

En résumé, l'identification des **conditions de test** et la conception des **cas de test** sont essentielles pour garantir que les tests couvrent toutes les exigences et sont bien structurés pour détecter les anomalies. 🛠️✓

4-Techniques de Test Basées sur la Spécification (Black-Box)



4-1. Introduction aux Tests Basés sur la Spécification 🔍

- Chaque conception présente différents types de défauts. 💡
- Les tests peuvent être divisés en **tests statiques** et **tests dynamiques**.
- Les tests basés sur la spécification, aussi appelés **tests black-box**, se concentrent sur **ce que le logiciel fait**, pas sur **comment il le fait**.
- Le testeur considère le système sous test comme une **boîte noire** et sait seulement que, **en envoyant un certain input**, il obtiendra la sortie attendue. 🎯
- Cela inclut les tests **fonctionnels** et **non fonctionnels** (performance, maintenabilité, etc.). 🛠️

Les quatre types de tests basés sur la spécification sont :

1. **Partitionnement d'équivalence**
2. **Analyse des valeurs limites**
3. **Tables de décision**
4. **Test de transition d'état**

5- Dérivation des Cas de Test à partir des Cas d'Utilisation 📝

- Les cas de test dérivés des cas d'utilisation permettent de vérifier que **toutes les fonctionnalités** du système sont testées en fonction des actions et des interactions des **acteurs** avec le système. 🔍

5-chois de technique de test:

- Choisir une technique de test implique de considérer plusieurs facteurs : système, expertise du testeur, et exigences spécifiques.
- Il est souvent préférable de **combiner plusieurs techniques** pour une couverture complète des défauts possibles. 🌟

TEST MANAGEMENT:

1-Organization des Test:

- Le **test indépendant** est souvent plus efficace mais peut poser des défis en termes de communication et d'intégration.
- Un **test leader** joue un rôle clé dans la gestion et la planification des tests.
- Les **testeurs** doivent être rigoureux et impliqués dans toutes les étapes du processus.
- Une équipe de test efficace doit combiner **connaissance métier, compétences techniques et expertise en test.** 🚀

2- 📌 Plan de Test, Estimation & Stratégie de Test 🚀

1 Le But et le Contenu du Plan de Test 📋

✓ Pourquoi un plan de test est-il important ?

- Il définit **comment les tests seront réalisés.**
- Écrire un plan de test aide à **structurer la pensée** et mieux comprendre le projet.
- Il sert de **modèle** pour identifier les défis et peut être personnalisé selon les besoins du projet.
- Il facilite la **communication** avec les autres équipes.

- Il sert de **référence** pour les discussions passées (outils, ressources, méthodologie).
 - Il est **mis à jour** au fur et à mesure de l'évolution du projet.
 - **Différents plans** peuvent être créés en fonction des exigences spécifiques.
-

2 Que Faire Lors de la Planification ? 🤔

📌 Questions essentielles à se poser :

- Quels sont les éléments inclus et exclus du périmètre de test ?
 - Quels sont les risques produit ?
 - Quels niveaux de test seront utilisés ? (ex: test unitaire, d'intégration, système...)
 - Quels sont les besoins pour l'environnement de test ?
 - Quel niveau de documentation est nécessaire ?
-

📌 Résumé : Pourquoi un bon plan de test est essentiel ?

- ✓ Clarifie les objectifs et les processus de test.
- ✓ Améliore la communication et la coordination entre équipes.
- ✓ Anticipe les défis et les risques.
- ✓ S'adapte aux évolutions du projet.

Un bon plan de test = **des tests plus efficaces et une meilleure qualité du produit !** 🚀

3- 📌 Suivi, Surveillance et Contrôle des Tests 🔍

1 Suivi des Activités de Test 📊

✓ Pourquoi surveiller les tests ?

- Fournit une **vision claire** de l'avancement des tests aux équipes et aux managers.
- Permet de **collecter des données** pour améliorer les futurs tests.

- Utilise des **documents, feuilles Excel et outils spécialisés** pour suivre les progrès.
 - Génère des **rapports** montrant :
 - ◆ Nombre de tests exécutés et leur statut.
 - ◆ Défauts ouverts et leur date d'apparition.
 - ◆ Temps réel passé à tester chaque fonctionnalité.
 - Aide à **déterminer quand arrêter les tests** (critères de sortie atteints).
-

Reporting de l'État des Tests

Différence entre surveillance et reporting

- **Surveillance** = collecte des données.
- **Reporting** = partage des résultats.

Pourquoi est-ce important ?




- Aide à **visualiser les résultats** grâce à des **graphiques et tableaux**.
 - Permet de discuter des **objectifs de test** et de vérifier s'ils sont atteints.
-

Contrôle des Tests

Pourquoi est-il nécessaire ?

Les projets ne se déroulent **jamais exactement comme prévu** ! 🤪

Exemples de situations nécessitant un ajustement :

-  Le logiciel est livré **en retard**, impactant le calendrier des tests.
-  Les scripts de test de performance étaient prévus **hors heures de bureau**, mais doivent être déplacés au **week-end** car l'application est maintenant utilisée en continu.
-  L'**environnement de test n'est pas disponible** à temps.

Le rôle du contrôle des tests ?

- S'adapter aux **changements imprévus**.
 - **Revoir et ajuster** la planification.
 - Assurer que les tests restent **efficaces et pertinents**.
-

En résumé : Pourquoi un bon suivi des tests est crucial ?

- ✓ **Anticiper** les problèmes et ajuster les plans.
- ✓ **Communiquer efficacement** avec les parties prenantes.
- ✓ **S'assurer que les tests couvrent bien les besoins** avant la mise en production.

 Un suivi rigoureux = **moins de surprises, plus de qualité !**

4- Gestion de la Configuration (Configuration Management)




Pourquoi la Gestion de la Configuration est-elle Cruciale ?

Dans un projet logiciel, on ne gère pas seulement le **code source** ! 😲

Il faut aussi gérer :

- ✓ **Les scripts de test** 
- ✓ **Les logiciels tiers** utilisés 
- ✓ **Le matériel** 
- ✓ **Les données** 
- ✓ **Toute la documentation** de développement et de test 

 **Sans gestion de configuration, voici les risques :**

-  Tester une **mauvaise version du logiciel**.
-  Signaler des **bugs impossibles à reproduire**.
-  Travailler sur des **versions obsolètes** du code.

Pourquoi les petites équipes ignorent-elles souvent cette gestion ?

😅 Elles pensent que c'est **inutile...** jusqu'à ce qu'un **gros problème** survienne !

 **Mais en réalité, c'est une arme secrète !**





- Permet de **savoir exactement** sur quelle version un bug a été trouvé.
- Assure que **les tests sont effectués sur la bonne version** du logiciel.

- Évite les **incohérences entre équipes** (développeurs, testeurs, etc.).
-

La Gestion de la Configuration en Action

- ◆ **Version Control** (Git, SVN...) pour suivre les changements du code.
 - ◆ **Tagging et branches** pour tester différentes versions sans confusion.
 - ◆ **Base de données de suivi des bugs** (JIRA, Bugzilla...) liée aux versions.
 - ◆ **Automatisation** pour tester toujours la dernière version stable.
-

En résumé : Un atout indispensable !

-  **Tester la bonne version** = Moins d'erreurs ❌
-  **Un suivi clair des bugs** = Développement plus efficace ✅
-  **Une meilleure collaboration** = Moins de stress ! 🤝
-  **Adoptez la gestion de configuration avant qu'un bug ne vous force à le faire !** 😓

5- **Risques & Tests : Ce qu'il faut savoir !**

Comprendre les Risques dans le Test Logiciel

⚠️ **Un risque, c'est la possibilité d'un problème !**

Il peut entraîner :

- ❌ **Perte d'argent** 💰
- ❌ **Perte de temps** ⌚
- ❌ **Perte d'efforts** 💪
- ❌ **Baisse de qualité** 📉

🔍 **Exemple concret :**

Un simple rhume peut être **inoffensif pour un jeune**, mais **dangereux pour une personne âgée**.





➡ **Tout dépend du contexte et des conséquences !**

Types de Risques en Test Logiciel

1 Risques liés au Produit (Product Risks)

 Concernent le **logiciel lui-même** et son fonctionnement.

Exemples :

- ◆ Un logiciel qui **plante fréquemment** 
- ◆ Un logiciel qui **fait perdre de l'argent aux clients** 
- ◆ Des failles de **sécurité** 
- ◆ Des problèmes de **performance** 





 **Solution :**

- 👉 Identifier les risques **dès le début** (réunions, analyse des exigences...).
 - 👉 S'appuyer sur **les retours d'expériences des projets précédents**.
 - 👉 Appliquer une approche **basée sur les risques** pendant tout le cycle de test.
-

2 Risques liés au Projet (Project Risks)

 Concernent **l'organisation des tests** et la gestion du projet.

Exemples :

- ◆ Livraison tardive des éléments à tester 
- ◆ Manque de ressources humaines ou techniques 
- ◆ Bugs qui prennent trop de temps à être corrigés 
- ◆ Catastrophes naturelles 

 **Solution :**

- 👉 **Planifier à l'avance** (exemple : prévoir du personnel de secours en cas de départ).
 - 👉 **Avoir un plan B** pour faire face aux imprévus.
 - 👉 **Optimiser la communication** entre les équipes (éviter les blocages inutiles).
-

Gérer et Atténuer les Risques

 Quelques stratégies efficaces :

- ✅ **Anticiper** : Analyser les risques dès le début du projet.

- ✅ **Planifier** : Avoir un **plan de secours** en cas de problème.
 - ✅ **Former** : S'assurer que l'équipe a les compétences nécessaires.
 - ✅ **Surveiller** : Suivre l'évolution du projet pour détecter les risques à temps.
-

En résumé...


 **Les risques sont inévitables, mais une bonne préparation permet de les maîtriser !**

 **Mieux vaut prévenir que guérir ! 😊**




6- Gestion des Incidents en Test Logiciel

Qu'est-ce qu'un incident en test logiciel ?

 **Un incident, c'est quand les résultats réels ne correspondent pas aux résultats attendus !**



 On les appelle aussi **bugs, défauts, problèmes ou issues**.

 **Pourquoi c'est important ?**






- ✅ Le statut des incidents permet d'évaluer l'avancement du projet 
 - ✅ Une bonne gestion des incidents améliore la qualité du produit 
 - ✅ Des outils spécialisés facilitent le suivi et la résolution des incidents 
-

Ce qu'un rapport d'incident doit contenir

✅ **Infos générales :**

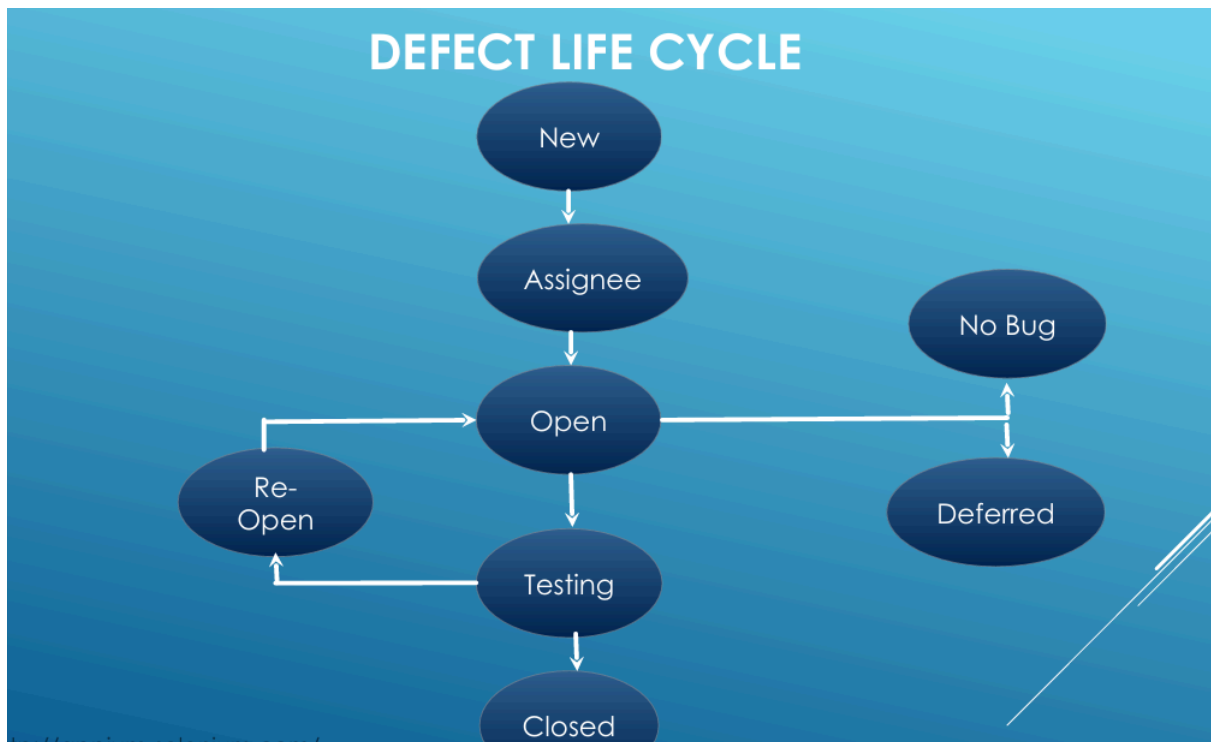
- ◆ Date de l'incident  17
- ◆ Organisation et personne assignée 

✅ **Détails techniques :**

- ◆ Gravité et priorité 
- ◆ Références (ex : quel cas de test a révélé le bug ?) 
- ◆ Résultats attendus vs réels  
- ◆ Environnement de test 

✅ **Données pour la résolution :**

- ◆ Logs, captures d'écran, fichiers dump 🖨️
- ◆ Cycle de vie du logiciel où l'incident est apparu 🔄
- ✅ **Suivi de l'incident :**
- ◆ Statut (ouvert, en cours, corrigé...) 🟡
- ◆ Historique des actions et solutions appliquées 📖



7-🔍 Les Différents Types d'Outils de Test Logiciel 🛠️

📌 1. Outils de gestion des tests 🏗️

📋 **Permettent de :**

- ✅ Planifier et exécuter les tests 🎯
- ✅ Suivre les activités de test 📅
- ✅ Assurer la traçabilité des tests 🔗
- ✅ Générer des rapports d'avancement 📊

📌 2. Outils de gestion des exigences 📖

Pourquoi les utiliser ?

- ✓ Assurer une meilleure qualité des exigences 🏆
 - ✓ Stocker et organiser les exigences 🔍
 - ✓ Vérifier la cohérence et détecter les ambiguïtés 🚨
 - ✓ Prioriser les exigences pour optimiser les tests 📌
-

3. Outils de gestion des incidents 🚨

Aussi appelés :

- ✓ Outils de suivi des défauts
- ✓ Outils de gestion des bugs

Leur rôle :

- ✓ Stocker et suivre les bugs 🐛
 - ✓ Ajouter des pièces jointes 📎
 - ✓ Attribuer des tâches et définir les priorités 📌
 - ✓ Suivre l'état des incidents (ouvert, en cours, résolu...) 🚦
-

4. Outils de gestion de configuration ⚙️

Utilisés pour :

- ✓ Gérer les versions et les builds du logiciel 📌
 - ✓ Assurer un test contrôlé et structuré 🔄
 - ✓ Faciliter la gestion des releases 🚀
-

5. Outils de test statique 📖

Permettent de :

- ✓ Stocker et trier les commentaires de revue 🔍
 - ✓ Communiquer les remarques aux équipes 💬
 - ✓ Suivre l'état des revues 🚦
-

6. Outils d'analyse statique 🖥️

 Principalement utilisés par les développeurs

- ✓ Vérification du respect des normes de codage 🛑
 - ✓ Calcul de la complexité cyclomatique 📊
 - ✓ Amélioration de la compréhension du code 🔍
-

📌 7. Outils de modélisation 🎨

📌 Utilisés pour :

- ✓ Concevoir l'architecture du logiciel 🏗️
 - ✓ Vérifier la cohérence du modèle 📌
 - ✓ Identifier et prioriser les zones à tester 🔍
-

📌 8. Outils de conception de tests 🖥️

📌 Permettent :

- ✓ L'extraction automatique des résultats attendus
 - ✓ La comparaison avec les résultats réels 📊
-

📌 9. Outils de préparation des données de test 📊

- ✓ Extraire des enregistrements de bases de données 🏠
 - ✓ Générer de nouvelles données respectant des critères spécifiques
-

📌 10. Outils d'exécution des tests 🔄

◆ Aussi appelés outils de **capture/rejeu**

📌 Utilisés pour :


- ✓ Automatiser les tests de régression 🔄
 - ✓ Capturer et rejouer les interactions utilisateur 🎥
 - ✓ Générer des rapports de test 📊
-

📌 11. Outils de test unitaire 🧩

👤 Principalement utilisés par les développeurs


- ✓ Création de **stubs** et **drivers** pour isoler les composants 🔄
- ✓ Enregistrement des résultats (succès/échec) ✓✗
- ✓ Support au débogage 🔧

12. Outils de comparaison de tests

- ✓ Automatiser la comparaison entre les résultats attendus et réels
 - ✓ Comparer des fichiers volumineux 
-



13. Outils de mesure de couverture

 Utilisés par les développeurs

- ✓ Vérifier le code testé ✓
 - ✓ Calculer le pourcentage de couverture 
-



14. Outils de sécurité

 Utilisés pour :

- ✓ Détecter les failles de sécurité 
 - ✓ Identifier les ports ouverts et les vulnérabilités réseau 
-




15. Outils d'analyse dynamique

 Utilisés par les développeurs pour :




- ✓ Détecter les fuites mémoire 
 - ✓ Identifier les erreurs de pointeur ✗
 - ✓ Repérer les liens morts dans le code 
-

16. Outils de test de performance, de charge et de stress

 Trois types de tests :







- ✓ **Test de performance** : Évaluer les performances dans un contexte spécifique 
- ✓ **Test de charge** : Augmenter progressivement la charge jusqu'à la limite 
- ✓ **Test de stress** : Tester le comportement au-delà des limites prévues 

 Ces outils permettent aussi :

-  Générer des charges pour le système
 -  Mesurer les temps de réponse
 -  Produire des graphiques d'évolution
-

17. Outils de surveillance

 Utilisés pour :

-  Surveiller en continu l'état des systèmes 
 -  Identifier rapidement les problèmes 
 -  Alerter en cas d'anomalie 
-

En résumé...

 Les outils de test aident à automatiser, structurer et améliorer le processus de test !

 Chaque outil a son rôle pour garantir un logiciel fiable et performant ! 