1.5 Software Testing (Les tests logiciels - اختبار البرمجيات)

1.5.1 Définitions et objectifs

Le logiciel présente la particularité de ne pas être sujet aux erreurs de fabrication puisque cette fabrication se limite en général à une simple copie. Par contre il souffre très souvent d'erreurs de conception : le ou les programmes développés et installés ne font pas ce qui était attendu, ne correspondent pas aux besoins. Plus ces besoins sont complexes et de nature variée, plus le développement et la conception sont propices aux erreurs.

La définition donnée par Swebok est la suivante :
Le test de logiciel consiste à vérifier dynamiquement, par le biais d'un ensemble fini de jeux d'essai, le comportement réel d'un programme par rapport au comportement spécifié et attendu.

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

Le test est donc l'exécution ou l'évaluation d'un programme et de ses données (avec des jeux d'essai), par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou pour identifier les différences entre les résultats attendus et les résultats obtenus.

L'objectif du test est, d'exécuter un programme avec l'intention de trouver des erreurs dans le but d'apporter une valeur ajoutée en matière de qualité et de fiabilité. Le coût de ce processus (qui représente 50% du coût total de développement) doit être compensé par l'amélioration de la valeur du produit logiciel testé (en termes de fiabilité, donc de qualité).

1.5.2 Méthodes de Validation & Vérification du logiciel

- ✓ **V & V :**

– Vérification : Est-ce que le logiciel fonctionne correctement ?
– Validation : Est-ce que le logiciel réalise les fonctions attendues ?

‹

- ✓ **Méthodes de V & V**

– Test statique : Review de code, de spécifications, de documents de design
– Test dynamique : Exécuter le code pour s'assurer d'un fonctionnement correct
– Vérification symbolique : Run-time checking, Execution symbolique, …
– Vérification formelle : Preuve ou model-checking d'un modèle formel, raffinement et génération de code

Actuellement, le test dynamique est la méthode la plus diffusée et représente jusqu'à 60 % de l'effort complet de développement d'un produit logiciel

Le processus de test confronte deux problèmes essentiels :
1-Identifier des ensembles de données similaires aux données réelles (appelées cas de test), qui seront manipulées par le système.

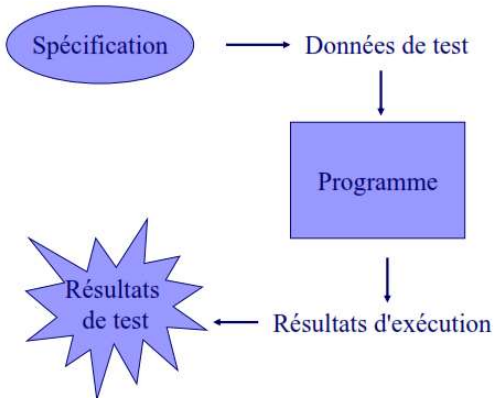
2- Comment aborder le test ?

1.5.3 Les Cas de test et Approches de test

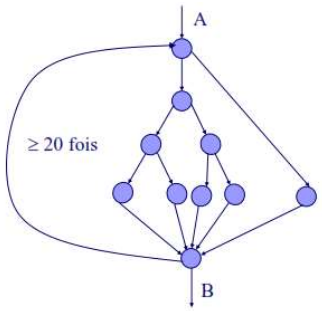
Il est généralement impossible de tester toutes les données d'entrée possibles. Il faut, donc, sélectionner des données d'entrée : C'est les cas de test.

Les cas de test doivent faire fonctionner l'ensemble du système, et permettre de vérifier que ce dernier se comporte de la façon prévue en présence de données incorrectes.

Il existe deux approches de construction de cas de test :
- **L'approche fonctionnelle** (test de boite fermée ou boite noire): Les cas de test sont construits à partir des spécifications (on s'intéresse aux fonctionnalités du logiciel).
On cherche donc à tester la conformité du logiciel par rapport à la spécification



- **L'approche structurelle** (test de boite ouverte ou boite blanche): Les cas de test sont construits à partir de la structure interne du programme. On doit suivre les chemins des données à travers les structures de contrôle du programme et parfois des données.



Flot de contrôle d'un petit programme

Les techniques fonctionnelles et structurelles sont utilisées de façon complémentaire. Les approches structurelles détectent plus facilement les erreurs commises. Les approches fonctionnelles détectent plus facilement les erreurs d'omission et de spécification.

En test fonctionnel, l'ensemble des données d'entrée est en général infini ou très grande taille.
Exemple : un logiciel avec 5 entrées analogiques sur 8 bits admet 2^{40} valeurs différentes en entrée.

En test structurel, le parcours du graphe de flot de contrôle conduit à une forte explosion combinatoire.
Exemple : le nombre de chemin logique dans le graphe de la figure précédente est supérieur à $10^{14} \approx 5^{20}+5^{19}+\ldots+5^{1}$
(il y a une boucle 20 fois et 5 chemins différents)

=> le test est une méthode de vérification partielle de logiciels
=> la qualité du test dépend de la pertinence du choix des données de test

1.5.4 Les étapes de test

Le deuxième problème posé (comment aborder le test) est dû au fait qu'il n'est pas réaliste de vouloir tester un logiciel comme une seule unité. De la même façon que pour le processus de conception, les opérations de test se feront par étapes, chaque étape constituant la suite logique de l'étape précédente.

Dans le cycle de vie d'un projet de développement, il y a généralement 3 grandes étapes de tests :

1. **Les tests unitaires, ou tests de programmation**

Les tests unitaires sont un processus ayant pour but de tester chaque composant du logiciel pris isolément. Dans un système bien conçu, chaque fonction doit avoir une spécification clairement établie. Les tests de programmation ont, dans un premier temps, pour but de vérifier la mécanique, l'ergonomie et la présentation des programmes. Dans un deuxième temps on s'assure que toutes les règles sont implantées. Dans un troisième temps, on s'assure du bon fonctionnement des interfaces et traitements.

Les tests unitaires présentent plusieurs avantages, parmi lesquels on peut citer :
• Concentration de l'effort de test sur chaque composant.
• En cas de comportement anormal du composant, il est plus facile de découvrir l'erreur, de la localiser, et enfin de la corriger.
• Les tests unitaires introduisent la notion de parallélisme durant la compagne de test, dans le sens où plusieurs composants du logiciel peuvent être testés indépendamment des autres et surtout simultanément, ce qui permet de gagner du temps.

2. Les tests d'intégration

A ce stade, les différents composants du logiciel qui ont été testés individuellement, devront être progressivement assemblés, pour construire un programme exécutable qui va être testé. Ces tests permettent de découvrir des erreurs de conception ou de codage, et de vérifier la bonne utilisation des interfaces entre modules. Ils permettent également de vérifier que le système, dans son ensemble, réalise bien les fonctions spécifiées lors de la définition des besoins.

3. Les tests de validation

Les tests de validation interviennent à la fin du processus d'intégration. Ils permettent de tester le système avec des données réelles. Ils mettent en évidence des erreurs dans un logiciel ne satisfaisant pas les niveaux de fonctionnalité ou de performance attendus par l'utilisateur.

Dans les systèmes complexes, et en plus de ces trois types de tests de bases, on peut élaborer les tests suivants :

Tests de non-régression :

Il faut vérifier que les corrections ou évolution dans le code n'ont pas créées de nouvelles anomalies.

Tests nominal ou test de bon fonctionnement :

On parlera de scénario nominal pour un scénario testant un cas d'utilisation « normal ». Les cas de test correspondent à des données d'entrée valide.
=> Test-to-pass

Tests de robustesse :

Il s'agit de tests au cours desquels on va simuler une charge importante d'utilisateurs sur une durée relativement longue, pour voir si le système testé est capable de supporter une activité intense sur une longue période sans dégradations des performances et des ressources applicatives ou système.

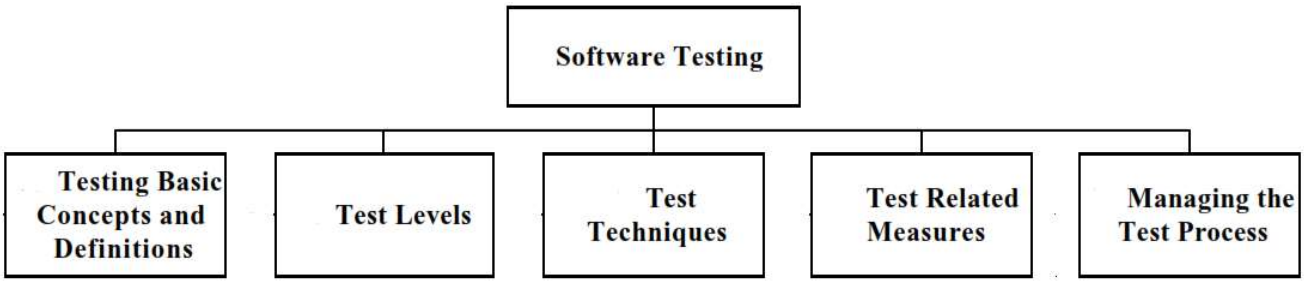
Les cas de test correspondent à des données d'entrée invalide
=> Test-to-fail

Règle : Les tests nominaux sont passés avant les tests de robustesse.

Test de performance

– Load testing = Test de charge (test avec montée en charge) : où l'on mesure le comportement d'un système en fonction de la charge d'utilisateurs simultanés.
– Stress testing = simulation de crise (soumis à des demandes de ressources anormales) : il s'agit d'un test au cours duquel on va simuler l'activité maximale attendue en heure de pointe de l'application, pour voir comment le système réagit au maximum de l'activité attendue des utilisateurs.

Dans SWEBOK, la partie tests logiciels est structurée comme suit :



La section Software Testing est donc composée de 5 sous sections :

- ✓ Testing Basic Concepts and Definitions (Bases de tests, Concepts et Définitions - قواعد الاختبار المفاهيم والتعريفات)
- ✓ Tests levels (Niveaux de tests - مستويات الاختبار)
- ✓ Test techniques (Techniques de tests - تقنيات الاختبار)
- ✓ Test related measures (Mesures relatives aux tests - تدابير ذات الصلة للاختبار)
- ✓ Managing the test process (Gestion du processus de test - إدارة عملية الاختبار)

1.6 Software maintenance (La maintenance de logiciels - صيانة البرمجيات)

1.6.1 Définitions et objectifs

Le développement de logiciels, lorsqu'il est mené à terme avec succès, se termine par la livraison d'un logiciel qui devrait satisfaire les exigences initiales du client. Par la suite, après sa mise en service, le logiciel devra évoluer pour répondre aux nouveaux besoins d'un environnement en constante évolution. De plus, c'est pendant l'opération du logiciel que l'organisation découvrira des anomalies, ainsi que de nouveaux besoins d'affaires qui feront surface, et qu'il faudra, après un certain nombre d'années, en modifier la plate-forme technologique.

La maintenance du logiciel est un des grands thèmes de connaissance reconnus comme faisant partie de la discipline de l'ingénierie du logiciel, tel que cela a été décrit dans le guide SWEBOK. Ce documents confirme que :

« La maintenance du logiciel est un domaine spécifique du génie logiciel, et conséquemment, il est donc nécessaire de se pencher sur des processus et les méthodologies qui tiennent compte des caractéristiques spécifiques de la maintenance du logiciel »

Selon SWEBOK, la maintenance est :
« La totalité des activités qui sont requises afin de procurer un support, au meilleur coût possible, d'un logiciel. Certaines activités débutent avant la livraison du logiciel, donc pendant sa conception initiale, mais la majorité des activités ont lieu après sa livraison finale »

Selon IEEE, la maintenance est :
« La modification d'un logiciel, après sa livraison, afin de corriger des défaillances, d'améliorer sa performance ou d'autres attributs ou de l'adapter suite à des changements d'environnements ».

1.6.2 Types de maintenance

Il existe trois types de maintenance d'un logiciel :

Maintenance perfective (évolutive)

Elle intervient suite à des changements dans les spécifications d'un point de vue fonctionnel ou de performance. Elle concerne donc les modifications demandées par l'utilisateur ou les programmeurs du système. Cela demande d'installer dans le programme de nouvelles fonctionnalités ou d'améliorer les performances.

Maintenance adaptative

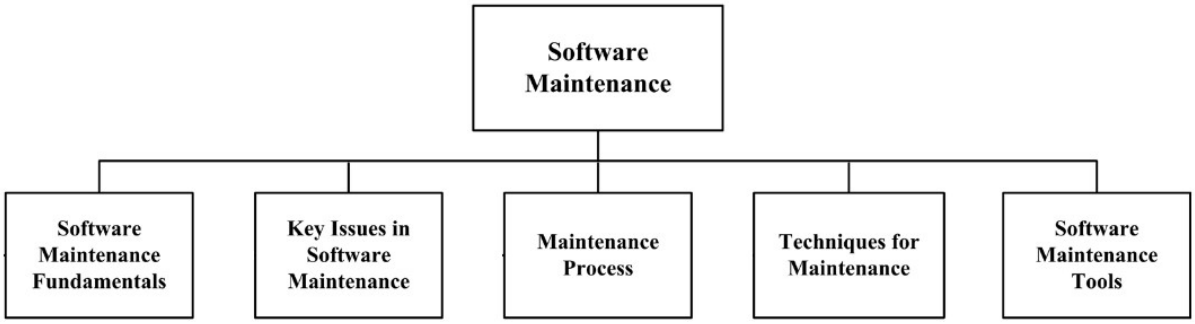
Elle est due aux changements de l'environnement du programme (environnement logiciel ou matériel). Elle concerne donc l'adaptation du programme à ce nouvel environnement.

Maintenance corrective

C'est la maintenance qui permet de corriger les erreurs jusqu'à présent ignoré dans le système (correction des défauts résiduels).

1.6.3 La maintenance de logiciels dans SWEBOK

Le Guide SWEBOK présente une taxonomie des connaissances nécessaires à l'ingénieur logiciel travaillant en maintenance du logiciel.



Cette taxonomie de la maintenance comprend cinq axes :

– Software Maintenance Fundamentals (Concepts de base de la maintenance de logiciels – المفاهيم الأساسية لصيانة البرمجيات)

- Définitions et terminologie
- Majorité des couts de la maintenance
- La nature de la maintenance
- L'évolution du logiciel
- Le besoin de maintenance
- Les catégories de maintenance

– Key issues in Software Maintenance (Préoccupations clés de la maintenance de logiciels – القضايا الرئيسية لصيانة البرمجيات)

- Techniques
- Gestion management
- Couts et estimation
- Mesures de la maintenance

– Maintenance Process (Processus de la maintenance – نمط الصيانة)

- Modèles de processus
- Activités de la maintenance

– Techniques for Maintenance (Techniques de maintenance – تقنيات الصيانة)

- Compréhension du logiciel
- Réingénierie (réorganisation d'un système existant)
- Rétro-Ingénierie

– Software Maintenance Tools (Outils de maintenance de logiciels – أدوات صيانة البرمجيات)

1.7 Software Configuration Management SCM (Gestion de la configuration logicielle - إدارة تكوين البرمجيات)

1.7.1 Définitions et objectifs

La gestion de configuration d'un logiciel introduit des principes de contrôle qualité dans le processus de création et de développement de logiciel.

La gestion de configuration aide les équipes de développement de logiciels à répondre aux exigences de plus en plus fortes de fiabilité qui leur sont demandées. Gérer la configuration d'un logiciel consiste à gérer les différentes versions de ses composants, les documents associés, les anomalies, les demandes de modification et les environnements (espaces de développement, de validation, de qualification, de production). Cette activité apporte au génie logiciel les moyens de réaliser les produits avec une qualité et une maîtrise du processus de développement élevées.

Selon SWEBOK, la configuration est une :
Discipline qui permet d'identifier et documenter les caractéristiques fonctionnelles et physiques d'un article de configuration et de vérifier la conformité avec les exigences spécifiées.

Selon ISO, la configuration est une :
Discipline de management de projet qui permet de définir, d'identifier, de gérer et de contrôler les articles de configuration tout au long du cycle de développement d'un logiciel.

Un article de configuration est un ensemble de matériaux, de services ou un sous-ensemble défini de ceux-ci, retenu pour la gestion de configuration et traité comme une seule entité dans le processus de gestion de configuration.

Une suite de points de contrôle fait preuve de fiabilité :
1 - La phase de développement
Dès le départ, un planning donne les échéances de livraison, à modifier en cas de retard. Une liste de contrôle précise les actions de chaque équipe.

2 - La validation et la qualification
Les logiciels développés sont transférés à la base de données de l'équipe de validation, qui les vérifie. Si les tests sont satisfaisants, les éléments validés transitent vers l'équipe de qualification, toujours de base à base. Elle vérifie à son tour si l'étape précédente a été bien réalisée. Toute anomalie constatée par l'une des équipes est remontée au chef de produits.

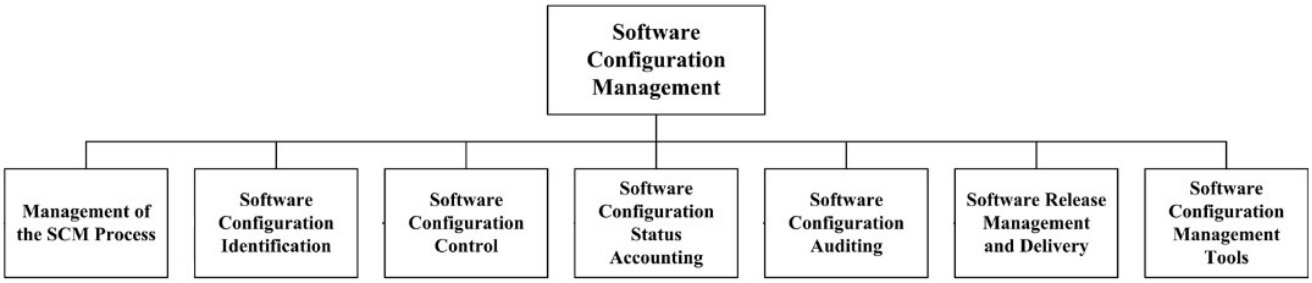
3 - La livraison
Un responsable du produit final livre celui-ci s'il n'y a pas de problème révélé par la qualification. Il récupère les sources, les transforme en exécutables, et ajoute ce qui les accompagne, comme les scripts de base de données. Puis il fabrique un CD d'installation et fournit les documents.

4 - La gestion des anomalies
Les anomalies donnent lieu à des demandes de corrections par des clients ou lors de la validation ou de la qualification. Le gestionnaire des anomalies enregistre les incidents avant de les soumettre au chef de produits. C'est lui qui décide de la suite. En cas de correction, l'anomalie refait le circuit développement, etc.

5 - La gestion des modifications
Pour traiter les demandes de modification, il faut un tableau de bord qui récapitule l'état des modifications demandées et des bugs. Il faut aussi disposer d'une traçabilité. Pour cette raison, des outils capables de traiter automatiquement les demandes de modification sont recommandés.

1.7.2 La gestion de configuration logicielle dans SWEBOK

Dans SWEBOK, la partie SCM est structurée comme suit :



La section Software Configuration Management est composée de 7 sous sections :

- Management of the SCM Process (Gestion du processus de configuration logicielle - إدارة عملية تكوين البرمجيات)
  - Contexte organisationnel pour la gestion de la configuration logicielle
  - Contraintes et orientation pour le processus de gestion de la configuration logicielle
  - Planification de la gestion de la configuration logicielle
  - Projet de gestion de la configuration logicielle
  - Surveillance de la gestion de la configuration logicielle

- Software Configuration Identification (Identification de la configuration logicielle - تحديد تكوين البرمجيات)
  - Identification des articles à contrôler
  - Librairies logicielles

- Software Configuration Control (Contrôle de la configuration logicielle - التحكم في تكوين البرمجيات)
  - Demande, évaluation et approbation des changements logiciels
  - Implémentation des changements logiciels
  - Déviations et renonciations

- Software Configuration Status Accounting (Etat de comptabilité de la configuration logicielle - حالة حسابات تكوين البرمجيات)
  - Information d'état de configuration logicielle
  - Rapports d'état de configuration logicielle

- Software Configuration Auditing (Vérification de la configuration logicielle - مراجعة تكوين البرمجيات)
  - Vérification de la configuration logicielle fonctionnelle
  - Vérification de la configuration logicielle physique
  - Vérifications de base du logiciel en cours

- Software Release Management and Delivery (Gestion des versions de logiciels et livraison - إدارة الإصدارات وتسليم البرامج)
  - Construction des logiciels
  - Gestion des versions de logiciels

- Software Configuration Management Tools (Outils de gestion de configuration logicielle - أدوات إدارة تكوين البرمجيات)

1.7.2 Les coûts et avantages de la gestion de configuration logicielle

Les coûts de la gestion de configuration logicielle peuvent représenter 5 à 10 % du coût global d'un projet. Pour certains secteurs nécessitant de très hauts niveaux de sécurité tels que l'aéronautique ou le spatial, ce coût peut atteindre 15 à 20 % par projet. L'adaptation des pratiques de maintenance à la gestion de configuration logicielle peut se révéler coûteuse ou irréalisable donc contre-productive, si elle n'est pas mise en place dans la foulée des développements.

En permettant d'établir et de valider des référentiels fiables, la gestion de configuration logicielle constitue un fondement stable de la construction et de l'évolution des systèmes.

Parmi les avantages, nous citons :
- Confiance accordée au système ;
- Moral des équipes informatique ;
- Satisfaction des acteurs;
- Crédibilité des informaticiens ;
- Détente des négociations budgétaires.

1.8 Outils et méthodes du génie logiciel

On va aborder les outils de génie logiciel au sens large
- Méthode de développement de programmes
- Outils pour mettre en œuvre ces méthodes

- **Méthodes:**
  - Design patterns
  - Refactoring
  - Test / Débogage
- **Outils:**
  - Eclipse
  - CVS/SVN
  - ANT
  - JUnit
  - BugZilla

A partir de tous ce qui a été présenté dans les sections précédentes, nous pouvons retenir que le logiciel est un programme exécutable et sa documentation, et que le développement logiciel est la discipline s'occupant de tous les aspects de la création de ce logiciel :
- Utilisation de méthodes et outils pour arriver à ce but
- Gestion de contraintes (financières, temporelles…)
- Prise en compte d'autres aspects que l'écriture de code: gestion de projet, développement d'outils, méthodes et théories permettant le développement de logiciels.

1.8. IDE Integrated Development Environment

Un IDE est un environnement de développement intégré qui combine :
- Un éditeur de texte
- Un compilateur/interpréteur/debugger
- Un outil de construction d'interface graphique

Pouvoir tout faire dans un seul environnement permet de :
- Simplifie les interactions entre les étapes de développement
- Limite l'apprentissage d'outils très spécifiques

Exemples d'IDE :
- Visual Studio (MS)
- Sun Studio (Sun)
- IntelliJ (jetBrains)
- Eclipse (OSGI/IBM)