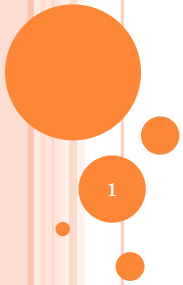
TESTS



1

Le test est une activité cruciale dans le monde du logiciel, comme l'attestent les données chiffrées suivantes :

- le poids de l'activité du test dans l'industrie du logiciel aux USA s'élève à plusieurs dizaines de milliards de dollars par an
- en moyenne, le test représente 30% du coût de développement d'un logiciel standard
- pour un logiciel critique (avionique, nucléaire, médical, transport), cette part moyenne monte à 50 %

Cette citation de Donald Knuth : « Beware of bugs in the above code ; I have only proved it correct, not tried it ».

Enfin, le test est la technique la moins coûteuse et la plus efficace pour capturer une grande partie des défauts d'implantation (bugs) et on ne peut donc pas s'en priver.

2

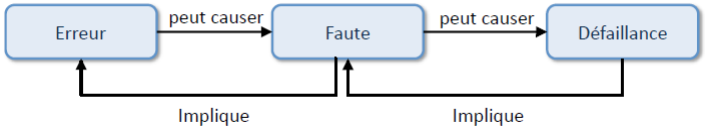
L'importance des tests

- L'erreur est humaine, presque tous les programmes contiennent des erreurs
- Les erreurs sont difficiles à identifier:
  - grande complexité des logiciels
  - invisibilité du système développé
- L'activité de test est essentielle au développement de logiciels de qualité

3

Définitions

- **Erreur!(*error*)**: commise par un développeur
  - Erreur de programmation
  - Erreur de logique
- **Faute!**ou **défault!(*fault,defect*)**: état interne invalide d'un logiciel après l'activation d'une erreur
- **Défaillance!(*failure*)**: manifestation externe d'une faute
  - Observable (ex: le programme dévie de sa spécifications)



4

- **Test**: exécution d'un programme dans le but de découvrir des erreurs [Myers 1979], non pas « … dans le but de démontrer l'absence d'erreurs »
  - Démontrer l'absence est très difficile ou impossible même pour de petits programmes
  - « Si on tente de démontrer l'absence d'erreurs, on n'en découvre que très peu. Si on tente de démontrer la présence d'erreurs, on en découvre la grande majorité. » [Myers, 1979]
  - Similairement, « … dans le but de démontrer que le programme fait ce qui est demandé » n'est pas suffisant : un programme peut contenir une erreur s'il fait autre chose en plus de ce qui est demandé.

5

L'importance des tests

- On peut généralement faire l'hypothèse qu'un programme contient des erreurs
- Les tests peuvent être utilisés durant toutes les étapes du développement
  - **Avant** de débuter l'implémentation
  - **Durant!**le développement
  - **Après!**que le logiciel soit complété
- Les tests représentent une partie importante du développement
  - Jusqu'à 33% du budget
  - Jusqu'à 27% du temps de développement

6

L'art du test

- Bien tester est difficile
  - Tester est un processus destructif
  - Tester une activité très intellectuelle et requiert beaucoup de créativité
- Faire échouer!le programme est une réussite lorsqu'on développe des tests
  - Éventuellement, les tests permettent d'atteindre un niveau de qualité acceptable

7

Principes de base

- Ne jamais planifier un effort de test sous l'hypothèse tacite qu'aucune erreur ne sera trouvée.
- Un élément nécessaire d'un test est une définition de la sortie ou du résultat
- Un programmeur doit éviter de tester son propre programme
- Les tests doivent être écrits pour les conditions d'entrée qui sont invalides et inattendues, ainsi que pour celles qui sont valides et attendues.
- L'examen d'un programme pour déterminer s'il ne fait pas ce qu'il est censé faire n'est que la moitié de la bataille, l'autre moitié, consiste à déterminer si le programme fait ce qu'il n'est pas censé faire.

8

L'activité de test

- 2 catégories d'activités :
  - Tests humains : basés sur la lecture du code par un être humain
  - Tests informatisés : basés sur l'exécution d'un programme par une machine

9

Inspections revues

- Techniques de test par des humains
  - Une équipe lit ou inspecte le code visuellement
- L'objectif d'une inspection ou revue est l'identification d'erreurs de programmation ou de logique
  - Pas de solutions identifiées en général
- Avantages :
  - Des développeurs autres que l'auteur du code sont impliqués
  - Peut être appliqué avant que le code ne soit fonctionnel
  - Permet de réduire le coût de débogage puisque l'emplacement des erreurs est précisément identifié
- Efficacité :
  - Permet de trouver de 30% à 70% des erreurs
  - Est plus efficace que des tests automatisés à détecter certains types d'erreurs

10

Inspections

- Une inspection de code est un ensemble de procédures et de techniques de détection d'erreurs pour la lecture de code en groupe.
- Une inspection est généralement effectuée en groupe de 4 personnes, et dure entre entre 1 et 2 heures
  - Un modérateur
  - Un programmeur (auteur du code inspecté)
  - Deux autres développeurs

11

Inspections - Déroulement

- Les participants reçoivent le matériel à l'avance (code, spécification) et se familiarisent avec celui-ci
- Durant l'inspection :
  - L'auteur du code fait la narration de la logique du code, ligne par ligne.
  - Les autres participants posent des questions, et l'équipe tente d'y répondre pour découvrir les erreurs.
  - Le programme est analysé à l'aide d'une liste d'erreurs fréquentes
- Contient des erreurs qui sont indépendantes du langage utilisé ainsi que des erreurs spécifiques au langage
- À la fin de la session, le programmeur reçoit une liste d'erreurs identifiés. Si plusieurs erreurs sont trouvées, une autre inspection peut être demandée.
  - Cette liste peut servir à améliorer le processus pour les inspections futures

12

Exemple – List d'erreurs fréquentes

- Erreurs de référence de données
  - Variable non initialisée
  - Index invalide pour les accès aux tableaux
- Ne respecte pas les limites, valeur autre qu'un entier
  - Mémoire utilisée n'a pas été préalablement allouée
- Erreurs de déclaration de données
  - Variables non déclarées
  - Les attributs d'une classe n'ont pas les bonnes propriétés (type, visibilité)
  - Variable ont des noms trop similaires (ex: volt, volts)
- Erreurs de calcul
- Calculs impliquant des types mixtes ou incohérents
  - Débordement des valeurs (overflow)
  - Division possible par zéro
  - Problème de précédence des opérateurs

13

- Erreurs d'interface
  - Nombre de paramètres incorrect
  - Incohérence des unités entre arguments et paramètres déclarés
- Erreurs d'entrée / sortie
  - Fichiers non ouverts avant l'écriture
  - Fichiers non fermés après l'écriture
  - Exceptions dues aux opérations d'entrée / sortie ne sont pas traitées correctement
  - Fautes d'orthographe / de grammaire dans le texte généré ou affiché
- Autres erreurs
  - Variables non utilisées
  - Validité des arguments non testés
  - Fonction absente du programme
  - Présence d'alertes lors de la compilation

14

Inspection - Avantages

- Permet d'identifier rapidement les erreurs dans les parties du système qui sont les plus à risque
- Permet de recevoir des commentaires sur le style de programmation ou le choix d'algorithmes
- Permet d'apprendre des erreurs des autres programmeurs
- Permet de concentrer l'effort de test automatisé sur les régions problématiques du programme

15

Revues structurées

Similaire aux inspections

- Rôles :
  - Modérateur
  - Secrétaire
  - Programmeur (auteur du code en revue)
  - Testeur
- Déroulement :
  - Les participants se préparent avant la revue, comme pour une inspection
  - Le testeur apporte une courte liste de tests qui sont « exécutés » mentalement
- Ces tests permettent de lancer la discussion
  - Le programmeur reçoit une liste d'erreurs (compilée par le secrétaire), comme pour l'inspection

16

Un exemple de « Check-list »

Initialisation des variables ?
Constantes nommées ou littérales ?
Validité des prédicats dans les conditionnelles (=, <, ≤, ...)
Prédicats pour la terminaison des boucles (<, ≤, ...)
Bornes inférieures et supérieures des tableaux (0, 1, taille, taille-1)
Délimiteurs des chaînes de caractères (ex: \0 en fin de chaîne de en C)
Correction des allocations dynamiques (allocation/désallocation)
Passage de paramètres dans les sous-programmes (selon les langages)
Gestion des liens dans les listes chaînées
Parenthésage des instructions dans les boucles/conditionnelles
Parenthésage des expressions booléennes
Traitement des exceptions (récupération, propagation)
...

17

Quelques principes de base

Présentation de quelques principes de base pour réaliser des tests.

Principe 1 : Un programmeur ne doit pas tester ses propres programmes. Tout simplement pour ne pas être juge et partie.

Principe 2 : Ne pas effectuer des tests avec l'hypothèse de base qu'aucune erreur ne va être trouvée.

Principe 3 : La définition des sortie ou résultats attendus doit être effectuée avant l'exécution d'un test.
Ceci est une erreur fréquente du test. Ce n'est pas lorsque l'on effectue le test d'une procédure qu'il faut se poser la question de la validité des résultats, car il y a des chances non négligeable de prendre un résultat erroné mais semblant cohérent pour un résultat correct.

18

Principe 4 : Inspecter minutieusement les résultats (trace) de chaque test.

Pour une raison similaire au principe 2, il faut séparer l'exécution (action) et l'analyse des résultats.

Principe 5 : Les jeux de tests doivent être écrits pour des entrées invalides ou incohérentes aussi bien que pour des entrées valides.

Simplement afin de découvrir les anomalies.

Principe 6 : Vérifier un logiciel pour détecter qu'il ne réalise pas ce qu'il est supposé faire n'est que la moitié du travail. Il faut aussi vérifier ce que fait le programme lorsqu'il n'est pas supposé le faire.

Afin d'évaluer la robustesse du logiciel.

19

Tests unitaires

Pour démontrer que chaque module effectue toute la fonction prévue et seulement cette fonction. On peut distinguer dans ces tests unitaires :

- les tests de logique (recherche d'erreur, vérification de l'enchaînement correct des branches parcourues) ;
- les tests de calcul (vérification des résultats des calculs, des performances, de l'exactitude des algorithmes).

Typiquement, les tests de calcul comprennent les tests de données dans les limites des spécifications (état normal), aux limites spécifiées et en dehors de ces limites (état anormal). Les tests de comportements anormaux (hors limites, erreurs) sont généralement appelés tests de robustesse.

20

Tests d'intégration du logiciel

Pour démontrer le bon fonctionnement d'unités fonctionnelles constituées d'un assemblage de modules.

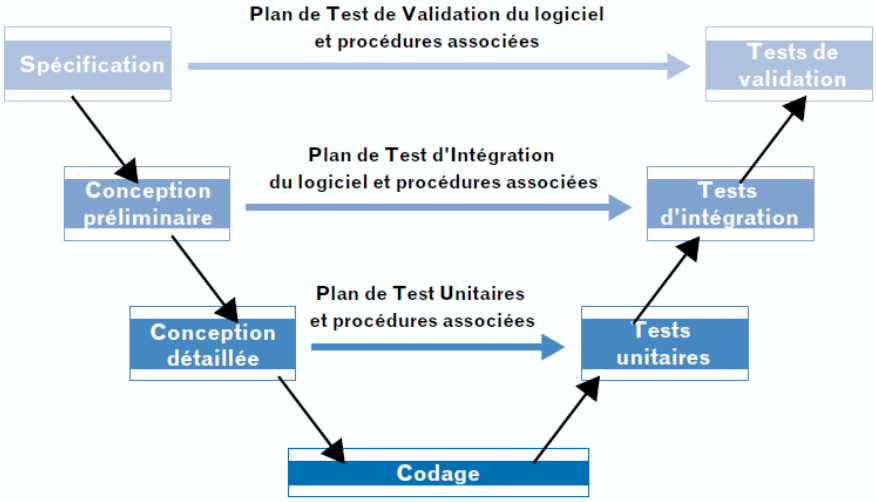
Ils portent principalement sur la vérification des enchaînements entre modules, la circulation des données, les aspects dynamiques, les séquences d'événements prévus et les reprises en cas d'interruption.

21

Tests de validation

Pour s'assurer que le logiciel implanté dans le matériel répond aux spécifications fonctionnelles, en vérifiant plus particulièrement les fonctions générales, les interfaces matériel /logiciel, le fonctionnement temps réel, les performances, l'utilisation et l'allocation des ressources.

22



Les processus de développement et de test agiles et plus précisément le développement orienté par les tests lient très fortement les activités de développement et de test. Les exigences sont spécifiées sous forme de tests, et le code de test est souvent écrit avant le code applicatif.

Planification

Livrable « théorique »	Contenu
Planification des tests	Objectifs, stratégie, techniques et méthodes, organisation et responsabilités, moyens nécessaires, identification des tests et démonstration de couverture.
Procédures de tests	Description détaillée de chaque test en termes de mode opératoire, de données d'entrée, de résultats attendus et de critères d'arrêt.

24

Contenu des produits issus de la définition des tests de validation - Content of the products stemming from the definition of the validation tests

- **Organisation, responsabilités**
  Personnes impliquées dans la validation. Organisation. Responsabilités.
- **Objectifs génériques des tests**
  Par exemple :
  - Couverture de toutes les fonctions ; de tous les modes de fonctionnement ; des exigences de sûreté en cas de défaut ; des exigences de performances ;
  - Conformité à la documentation d'utilisation, d'exploitation ;
  - Vérification d'endurance, etc.
- **Stratégie de tests**
  Démarche retenue, avec ordonnancement en étapes visant des objectifs spécifiques ou s'appuyant sur des environnements ou des techniques spécifiques, par exemples :
  - Test de tel groupe de fonctions ; test de tel mode de fonctionnement ;
  - Test d'enchaînement de telles fonctions, tests de bout en bout ; tests de paramétrage ;
  - Tests d'endurance ; tests avec simulation dans tel environnement, etc.
- **Méthodes et techniques de tests**
  Méthodes et techniques utilisées, par exemple : techniques manuelles ou automatiques, analytiques ou statistiques, tests de domaines, tests aux limites, etc.
- **Environnements et outils de tests**
  Identification des outils de tests utilisés (simulateurs, outils de mesures, comparateurs, etc.).
  Si l'environnement du logiciel n'est pas le même dans les différentes étapes, description de chacun de ces environnements.

25

Livrable « théorique »	Contenu
Rapport de tests	Conditions de réalisation des tests, éléments testés, tests exécutés, résultats obtenus, interprétation des résultats et bilan.

Détaille le contenu des livrables (rapports de tests) issus de la phase d'exécution des tests de validation.

Les informations sur les objectifs de tests, les environnements et outils de tests utilisés, etc., sont requis pour une bonne compréhension des rapports de validation.

On notera qu'il doit y avoir autant de rapports que de versions fournies.

26

Détails :

Contenu des produits issus de l'exécution des tests de validation - Content of the products stemming from execution of the validation tests

- **Objectifs des tests**
  Doivent être définis en tenant compte de la référence (identification des exigences, paragraphe d'un document, etc.)et forment une partie des objectifs généraux de la planification des tests.
- **Environnements et outils de tests utilisés**
  Peut être fourni par référence à la documentation de définition des tests.
  A compléter par les informations concernant l'étalonnage des appareils de mesure (données de calibration), le cas échéant, et par les écarts au plan de test.
- **Données et / ou signaux en entrée**
  A fournir avec leur séquencement et leur valeur.
  Si les tests sont automatisés, ces informations peuvent prendre la forme d'enregistrements numériques à condition de disposer des règles et moyens pour les interpréter.
- **Données et/ou signaux attendus en sortie**
  Idem ci-dessus.
- **Mode opératoire**
  Suite des actions à réaliser pour mener le test.

27

- **Critères de réussite**
  Implicitement, l'obtention des données et/ou signaux attendus est un des critères.
  Ce critère doit être complété lorsque nécessaire par d'autres critères, par exemples : temps de réponse, tolérance sur les valeurs en sortie, absence ou occurrence de tel événement avant tel laps de temps.
- **Essais réalisés**
  Peuvent être fournis par référence aux procédures de tests.
  A compléter par les écarts au plan de test, le cas échéant (tests non exécutés).
- **Chronologie des essais**
  Dates effectives des différents essais, ordonnancement des essais.
- **Faits marquants**
  Écart à l'ordonnancement prévus, abandons de certains tests, etc..
  Peut faire l'objet d'un document séparé dit « journal de test ».
- **Détail des résultats obtenus**
  Pour chaque test, signaux et/ou données en sortie constatés (valeurs et séquencement) .
  Ces résultats peuvent prendre la forme d'enregistrements numériques,
  à condition que les règles et moyens de les interpréter soient disponibles.
- **Liens avec la gestion de modification**
  Pour chaque test en échec, lien vers la demande de modification correspondante.
- **Bilan des non-conformités**
  Liste synthétique des non-conformités détectées, avec leur impact sur la sécurité .
- **Preuve de couverture des tests**
  Par exemple, une liste des tests élémentaires et une référence croisée entre les exigences à tester et les tests identifiés.

28

CONCLUSION

Le logiciel « zéro défaut » n'existe pas. La présence de fautes logicielles systématiques introduites à la conception d'un dispositif programmé doit donc être considérée avec beaucoup d'attention, en particulier lorsque les conséquences de ces fautes peuvent influer sur la sécurité d'un dispositif.

Les tests sont la composante principale de la vérification.

Pour détecter un maximum de fautes, ces tests doivent être conduits en suivant une démarche qui débute par la définition préalable des objectifs de tests et de la stratégie adoptée pour atteindre ces objectifs, pour finir par la démonstration que ces objectifs sont satisfaits.

29

La réflexion induite par cette démarche impose au concepteur d'appréhender le processus test dans sa globalité, sans se focaliser sur la seule phase d'exécution.

Cette réflexion constitue une condition préalable indispensable pour réussir les tests d'un logiciel (en termes d'efficacité de détection) et minimiser ainsi les risques d'apparition de défaillances dangereuses en exploitation.

30