**SEG3503 revision**

**Logiciel**: Des programmes, procédures, ainsi que possiblement de la documentation et des données liées à l'opération d'un système informatique

**Qualite de logiciel**: degré auquel le produit logiciel répond aux besoins exprimés et implicites lorsqu'il est utilisé dans des conditions spécifiées

* **Vue transcendantale**:
  + Difficile a definir en termes abscraits
  + Propriete intangibles quis ravissent les utilisateurs
  + Un ideal qui ne pourra jamais etre complement atteint
* **Vue d’utilisateur**:
  + Est l’adequation au but
  + Depend de la facon dont le produit repond au besoin de ses utilisateurs pour les taches
  + Mesuré en fonction de la fiabilité, de la convivialité
* **Vue de fabrication**
  + Conforme aux normes de processus
  + Dépend de la manière a été construit
  + Mesuré en utilisant le nombre de defaut, le cout de retouche
* **Vue du produit**
  + Lie au caracteristique du produit
  + Mesuré et controlé par rapport au propriete interne
  + Propriete interne du produit : taille, code, commentaire, complexite
* **Vue base sur la valeur**
  + Depend de combien les clients sont prêt a payer
  + Cout et la qualite du produit
* **-**un logiciel est de bonne qualite si ca conforme aux exigence avec un bas ratio de defaut (# de defaut/unite de taille) et haute fiabilité (nombre de pannes par n heures d’opération)
* -haute fiablibite messure comme Temps Moyen entre Pannes probabilite d’operation sans panne dans un temps specifie

**Facteurs de qualite du logiciel**

* caracteristiques de la qualite globale du logiciel :
  + lies aux objectif et usage du produit
  + exactitude, fiabilite, efficacite, testatbilite, maintenabilite, reutilisabilite

**Pannes :** incapicite d’un système ou d’un composant executer ses fonctions requise ds les limites de exigences specifie

**Faute(defaut ou bug) :**  une etape, un processus ou une definition donnes incorrects ds un artefact logiciel

**Erreur :** une action humaine qui produit un resultat incorret

* une erreur lead a une faute
* une faute lead a une panne

**Assurance de qualite :** ensemble d’activite visant a assurer que peu ou pas de defaut subsistent ds le système logiciel lorsqu’il est livre a ses clienst ou mis sur le marché.

Pas uniquement les tests mais aussi

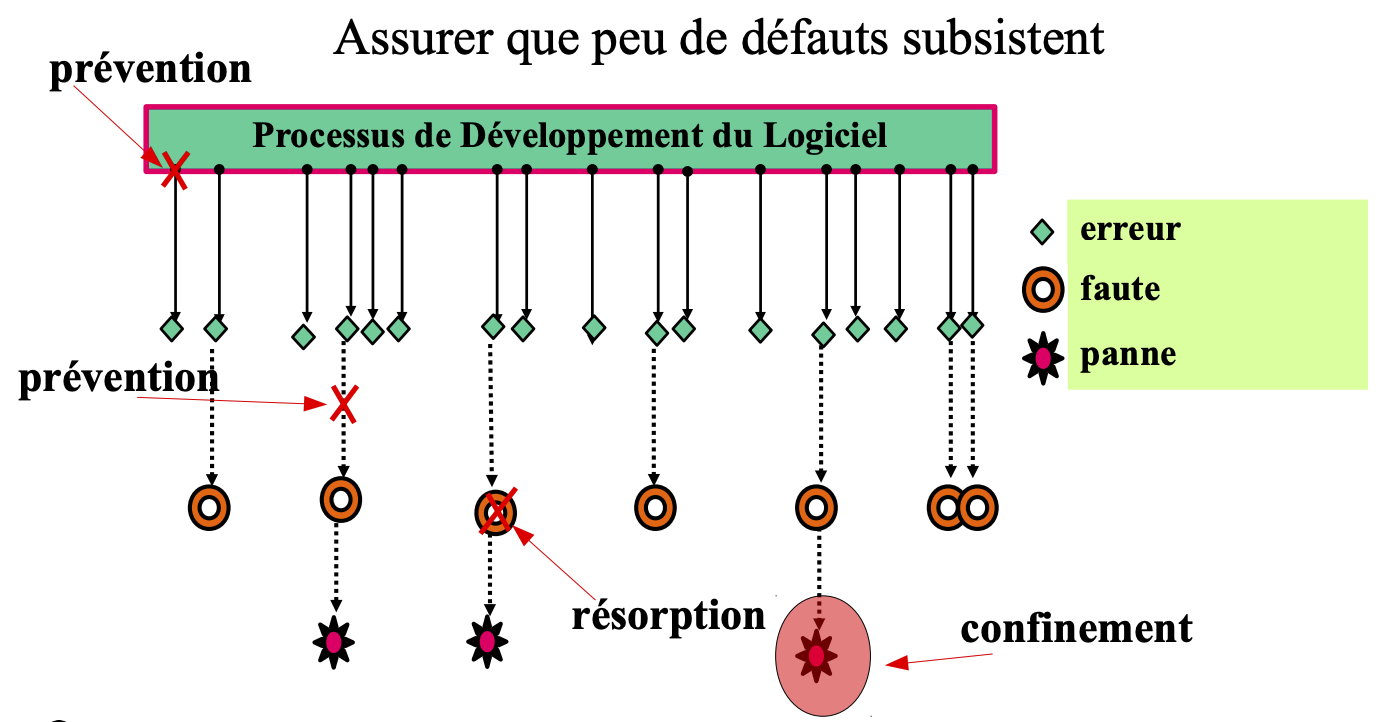
* revue du code
* formations
* metriques
* ingeneries de processus

**Objectif**:

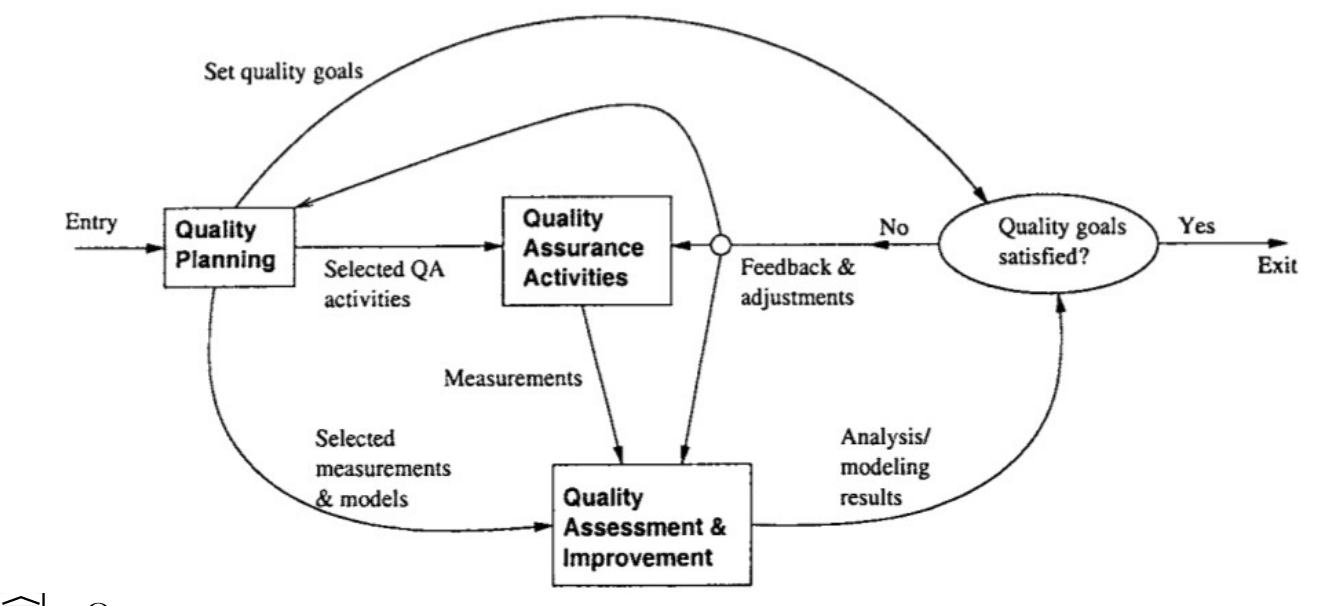
* Assurer un niveau de confiance acceptable-le logiciel est conforme aux exigences fonctionnelle technique
* Assurer un niveau de confiance acceptable – le logiciel est conforme aux exigence de gestion concernant l’echeancier et le bugdet.
* Initiation et activites de gestion pour l’amelioration et la plus grande efficience des activites de developpement, maintenance et d’assurance de qualite logicielle.
* On ne peut pas tout tester( test exhaustif)

**Principe du AQL**:

* Savoir ce que vs faites
  + Comprendre **ce qui** est entrain d’etre construit**, comment** il est construit et **ce qu’il fait**
* Savoir ce que vs devriez faire
  + Avoir des **exigences** et **specification** explicites
* Savoir mesurer la difference
  + avoir des mesures explicites comparant ce qui est entrain d'être fait de ce qui devrait-être fait
* L’assurance qualite est une approche de cycle de vie concernée par chaques aspect du processus de developpement logiciels
* Il faut comprendre
  + Ensemble complet d’objectif de qualite
  + Attribut mesurable de qualité pour evaluer le progres vers les objectifs
  + Objectif quantitatives de certification pour toutes les parties du processus de developement
* Ne pas oublier de prend en compte :
  + Exigences de produit des clients
  + Exigences de qualite et qualite corporatif
* En general l’assurance a comme objectif
  + S’assurer que peu de defaut subissent dans le système logiciel
* Classes d’activites d’AQ
  + Prevention de defauts (empecher les erreuts d’etres injecter dans le logiciel)
    - Eliminer les sources d’erreurs
    - Prevention ou blocage de defauts
    - Technique :
      * education et formation – elimine les idee fausse
      * méthodes formelles – eliminer les ambiguites
  + Resorption de defaut (detection et suppression les defaut avant la livraison)
    - Inspection, simulation de prototype, analyse statique, tes
  + Confinement de defauts(minimise les effets des pannes)
    - Contrôle des pannes locales de sorte qu’elles ne se propagent pas
    - Limite les dommages causés par les defauillances

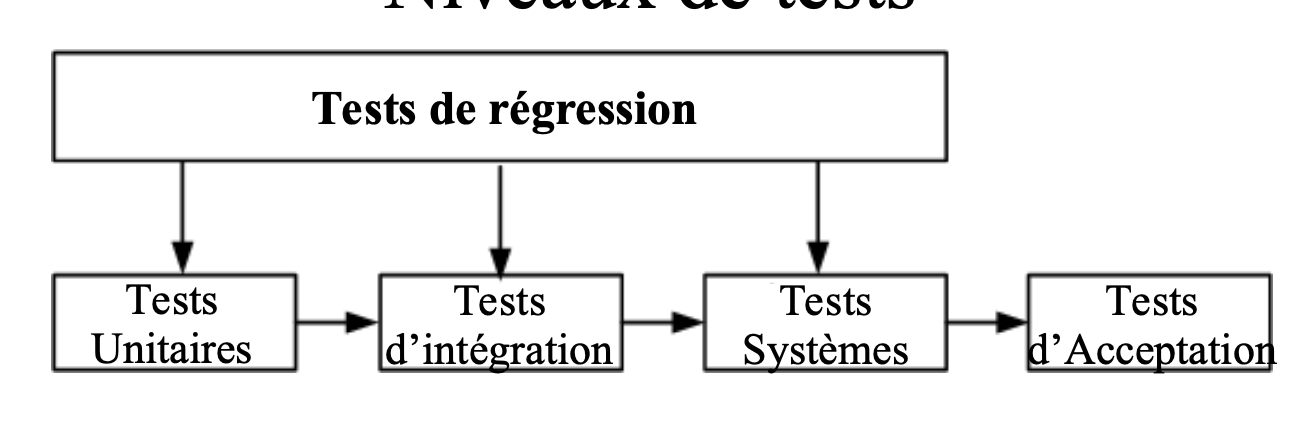


* **Verification vs validation**
  + Verification : construisons-nous le produit bien
    - Le logiciel doit etre conforme a ses specification
    - Processus d’evaluation d’un système ou d’un composant pour determiner si les produit d’une phase de developpement donnée satisfont aux condition imposé au debut
  + Validation : Construisons nous le bon produit
    - Le logiciel devrait faire ce dont l’utilisateur a vraiment besoin
    - Processus consistant a fournir la preuve que le logiciel et ses produits associés satisfont aux exigences système alloues au logiciel.
* **Ingenerie de la qualité**
  + Englobe SQA avec des activite connexes(planification, suivi, ameloriation)
  + Approche d’ingenierie pour gerer les attentes en matiere de qualité
    - Etablisement d’objectif de qualite
    - Mesure de la perfommance vers ces objectif
    - Analyse et amelioration des processus

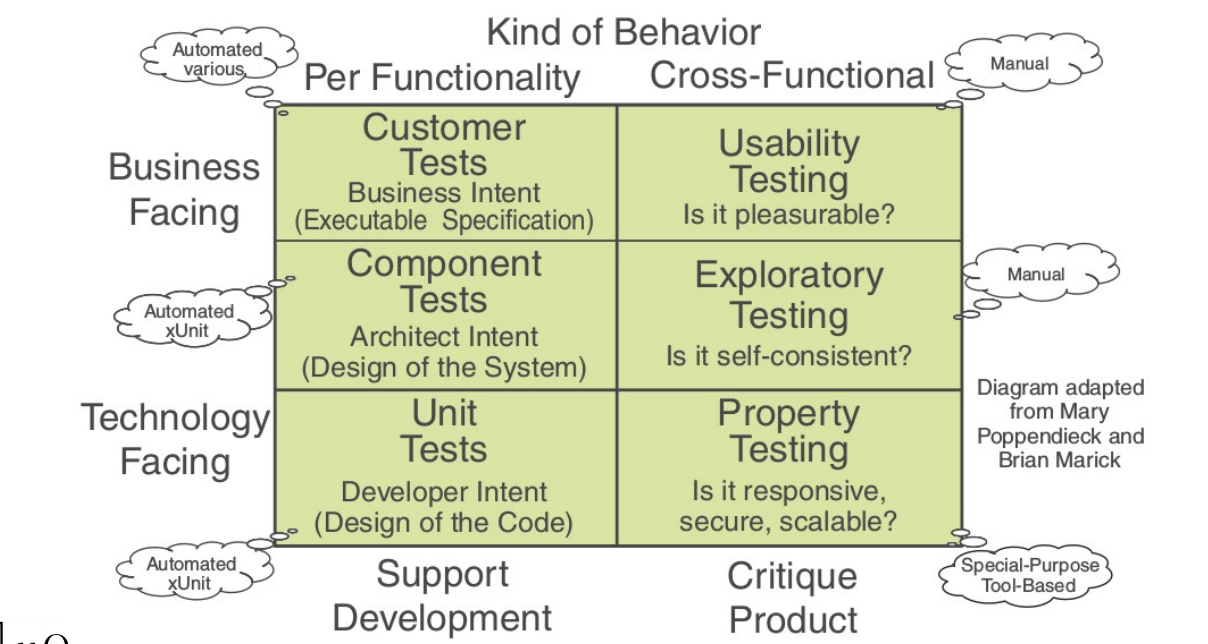
****

**INTRO AU TEST**

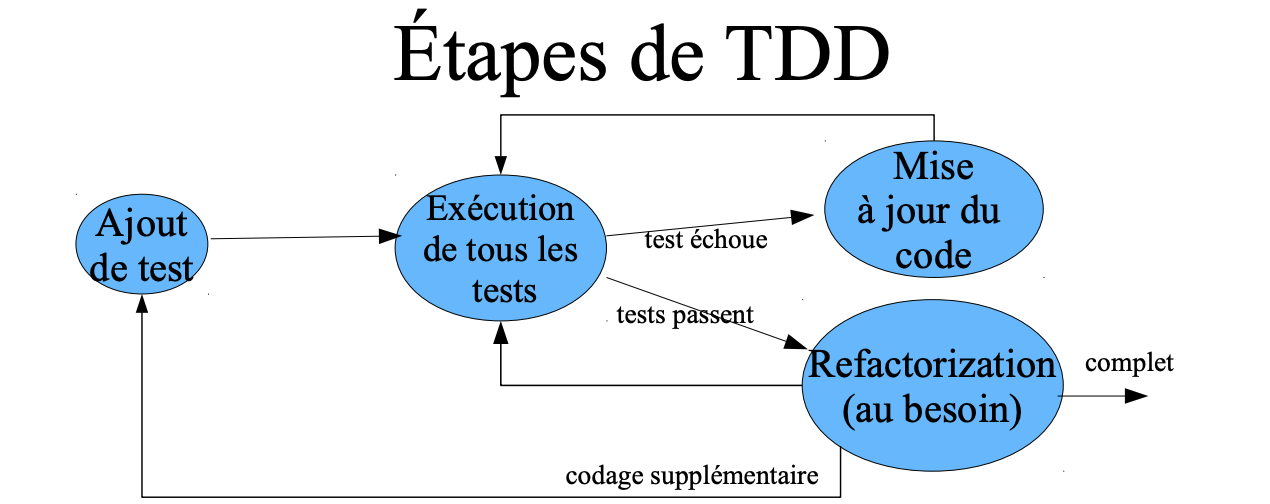
* Un test est un examen d’une, de plusieurs d’unite ou d’ensemble integré de composants logiciels par execution
* **Objectif :**
  + Detecter des defauts avant qu’ils ne causent une pannes
  + Amenere le logiciel tester à un niveau acceptable de qualité
  + Effectuer les test requis de facon efficiente et effective dans les limite de temps et budget definis
  + Compiler un ensemble de situation d’erreurs devant servier a la prevention des erreurs
* **Axiomes des test**
  + Il est impossible de tout tester car il y a
    - Un grand espace de donnees (entree) et sorties
    - Grand espace d’etats
    - Grand nombre de chemin d’execution
    - Subjectivité des specification
  + Les test ne peuvent prouver l’absence de defaut
  + **Paradoxe du pesticide**: Un systemen tend a developper de la resistance aux technique de test utilisés
* **Ce qu’un test peut faire**
  + Identifier des defauts qui seraient impossible ou trop couteux a trouver en utilisant d’autres techniques
  + Montrer que le système est conforme a ses exigence pour une suite de test donnes
  + Les test sont facilités plus efficaces par de bonnes pratique en matiere de genie logiciel
* **Processus de test**
  + Planification, Design, Implementation, excution, verification de resultats, gestion de test

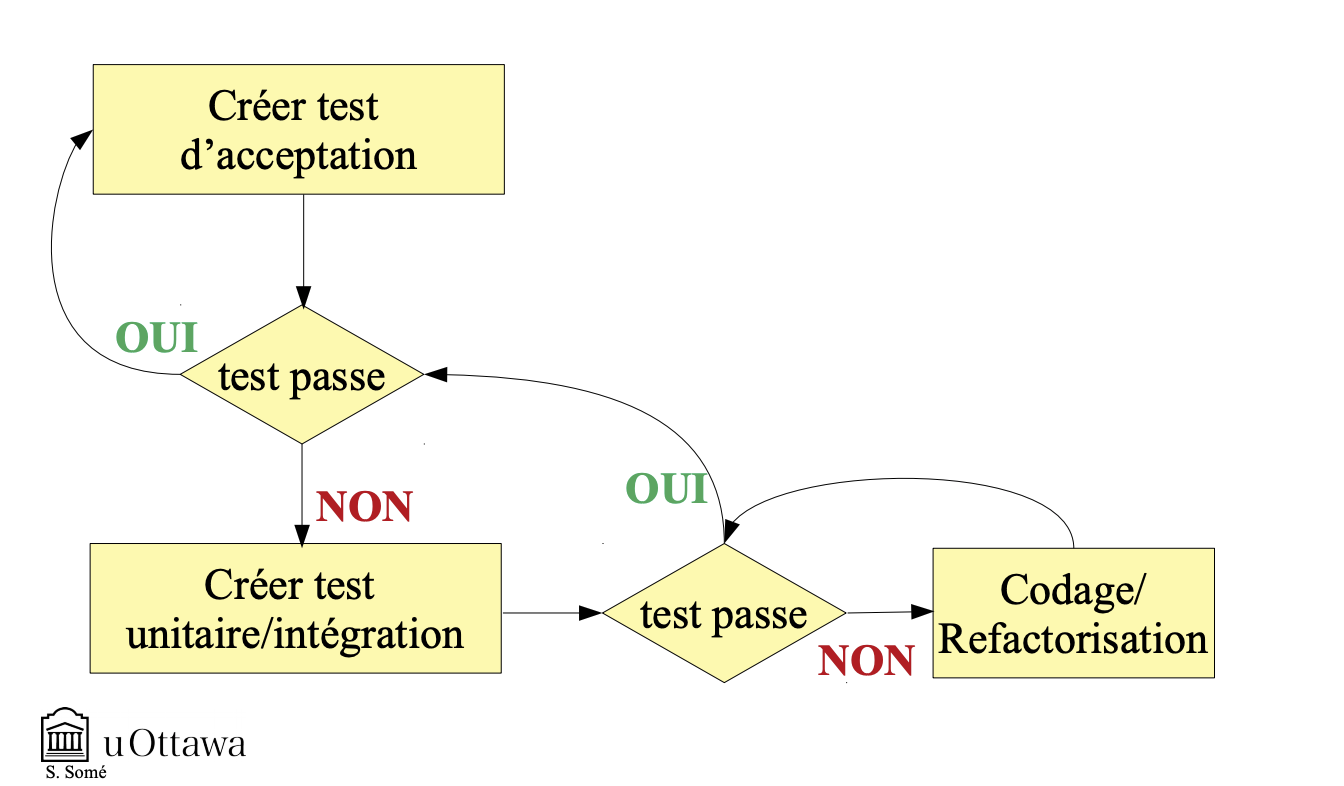
****

* Apres des mises-a jour : selectionner et prioriser les executions pour s’assurer que les changements n’introduisent pas de defauts
* **Test Agiles**

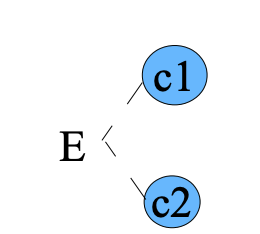
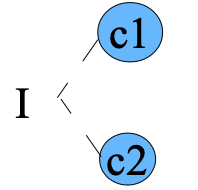
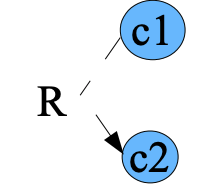
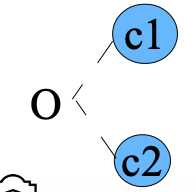
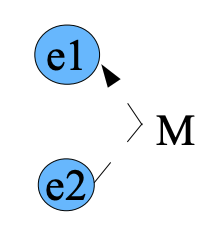
****

* + Test de clients :
    - Verifient le comportement de l’application du point de vue du cliens
    - Developer en fonction des besoins en specifianr des critere d’acceptation en collaboration du clients, utilisateur, testeurs,developpeur
    - Ecrit dans un langage comprehensible par les clients
  + Test unitaires
    - Verifient le comportement d’unités de programme(classe, methode, fonction) resultatnt d’une decision de conception
    - Resument le comportement de l’unites sous la forme de tests ecrit par le developppeurs
  + Test de composants(integrations)
    - Verifient les composants constitués de groupe d’unité qui fournissent collectivement un certain services
    - Ecrit par les developperur(derivers de test de clients)
  + Test de propriete(non-fonctionnel, cross-fonctionnels)
    - Verifient des exigences non-fonctionnel(ex :temps de reponse, capacite, stress, securite)
    - Test automatiser est essentiellle pour la plupart de ces tests et doivent debuter le plus tot possible
  + Test d’utilisabilité
    - Verifient l’adequation au but
    - Les utilisateur reels peuvent-ils utiliser l’applicarion logicielle pour atteindre leur objectifs
  + Test exploratoires
    - Pour determiner si le produit est auto-coherent
    - Les utilisent utilisent le produit, observent le comportement, formule des hypothese, concoivent des test selon des hypothese et les executent
* **Developpement pilote par les test (TDD)**
  + Approche de developpement logiciel basées sur des test en premier
  + Technique de bse de methodes agiles
  + Plusieurs test, executés souvent

****

* **Developpement piloté par les test d’acceptation (ATDD)**
  + Le developpemt commecnce par l’ecriture d’un test d’acceptation
    - Comme le comportement attendu du système du pt de vue des utilisateurs
    - Sans detail techniques
    - Ecrit en collab par experts du domaine, des clients, utilisateurs et developpeur
    - Ecrit en langage simple
* ****

**Black Box Testing Approach**:

* Test base sur une bue externe de l’entite testee
* On developpe ce qui est requis
* Desavantage :possibilite de manquer du code
* **Divination d’erreurs**
  + Approche ad-hoc basée sur l’experience et l’intuition
  + Test crees a partir d’un modele d’erreur pour l’Unite sous test (UUT)
  + Faire une liste d’erreurs possibles ou situations conduisant a des erreurs
  + Developper des cas tests pour couvrir le modele d’erreurs
  + **Avantage**: efficace, rapide
  + **Limitation**: la qualite de test depend de l’experience du testeur, mesures relatives a la suffisance du test ne peuvent pas etre colletctees, non applicacble par les testeurs in-experimentes
* **Partitionnement en classe d’equivalences :**
  + Peut etre applique aux entree, sortie, calcul
  + Partionnement basé sur la comprehension du fonctionnement de l’unité à tester
  + Les erreurs ont tendance a survenir vers les extremes
  + **AVB** ameliore le partitionnement en classes d’equivalences en selectionnant les element juste à et autour des bornes de chacun des CEs, derivant des cas de tests en considerant des CEs des resultats egalement.
  + **Approche pour cree un test**
    - **Une a une :**un cas de test par CE
    - **Minimisation :** autant de CEs couvert que possible par cas de test
    - **Approcge de selection de Myer** 
      * CEs valide –Minimisation
      * CEs invalide – une a une
    - **Approche combinatoire**
    - **Approche force-brute**
      * Donne une bonne couverture mais non pratiqye lorsque le nombre d’entrés et classes associes est grande
      * Possibilité de redondances
  + Des niv de combinaison plus faibles peuvent etre utilisés pour obtnir moins de cas de test
  + Couverture t-way
    - Ensemble de test couvre toute les combinaison de t elements
    - 1<= t <= k (le nombre total de parametrer
    - **Covering array –** objet mathematique qui couvre toute les combinaison t-wat de valeurs parametres au moins une fois
  + **Avantages :** necessite peu de cas de test, plus probable de decter des defauts
  + **Limitation :** specification ne definie pas tjrs les resultats attendus pour les cas de test invalides, langes elimine le besoin de la consideration de certain donnes invalide, approche de myer est faible lorsque les variable de donne ne sont pas independants
* **Partitionnement de categories :**
  + Methodologie systematique qui utilise une specification fonctionnelle informelle pour produire une specification de test formelle
  + Etape :
    - Identifier les parametres et conditions environnementales
    - Identifier des categories pour chaque parametre et conditions
    - Partitionner les categories en choix
    - Ajouter des contraintes aux choix
    - Créer de trame de tests comme selections de choix
    - Créer de cas de test à partir des trames de test
* **Table de Decisions**
  + Ideale pour une situation ou :
    - une combinaison d’action est selectionnee sous un ensemble de conditions variables
    - Les conditions dependent des variable de donnee
    - La reponse produite ne depend pas de l’ordre d’assignation ou evaluation des donnees
    - La reponse produite ne depend pas d’entre/sortie precedentes
  + Etape :
    - Identifier les variables et conditions de decisions
    - Identifier actions resultantes selectionnées ou controlées
    - Identifier quelle action doit etre produite en réponse a une combinaison d’actions particulieres
  + Condition **Don’t care**: peut-etre V ou F sans changer l’action
  + Condition **Can’t Happen**: reflete une hypothese selon laquelle des entres sont mutuellement exclusif, ne peuvent pas etre produite par l’environnement
  + Condition **Don’t know :** reflete un modele incomplet
* **Graphes Causes-effects**
  + Approche systematique d’aide a la selection d’ensemble de cas de test a haute efficacite
  + ^ (And) , v (or), ~ (not)
  + ** Exclusif :** ne peuvent pas etre simultanement vrais
  + ** Inclusif :** une c1 et c2 doit etre vrai ( les 2 peuvent pas etre F)
  + ** Requiere :** C2 doit etre vrai pour que C1 le soit
  + ** Une et seulement une :** une et seulement une de c1 ou c2 peut-etre vrai a un moment donné
  + ** Masque :** e1 forcé d’etre faux si e2

**TEST BLANCHE**

* on teste ce qui est déjà la
* **Desavantage**: fonctionnalité présente dans la spec mais non implemente4 peut etre manquée
* Peut etre lourde
* Approche oriente flot de contrôle
  + Basée sur l’analyse du flot de contrôle a traves programme
  + Exemple de critere de couverture de test :
    - Couverture d’instructions
      * Chaque instructions est visiter au moins une fois
    - Couverture de branches
      * Chaque branche est visiter au moin une fois
    - Couverture de chemins
      * Tout le chemins possible sont couvert
    - Couverture de conditions
      * La condition est vrai et faux une fois
    - Couverture de conditions/branches
      * Branche + conditions
    - Couverture de boucle
      * Loop simple :
        + Loop run 0 fois
        + Loop run 1 fois
        + Loop run a nombre typique de fois
        + Loop run a un nombre maximale de fois
      * Boucle imbriquês
        + Commence avec la boucle la plus interne

Mettre les boucles externe a leur valeur minimun

Mettrre toutes les autre boucle a une valeur type

Effectuer les situation de test de boucles simple

* + - * + Remonter a la boucle au-dessus et repeter
        + Si la boucle plus externe est faite, effecturer les situations de tests de boucles simples pour toutes les boucles simultanement
* Sensibilisation de chemins
  + Pour determiner les donnes visant a forcer un chemin selectionné
  + Strategie en arriere (backward) – de sortie vers entree
  + Strategie en avant (Forward) – de l’entre vers la sortie
  + Probleme avec chemins impossible – peuvent amener a devoir re-ecrire le programme
* Test de mutation
  + Un mutant – version d’un programme obtenue en repliquant le programme d’origine avec un petit changement
  + On peut change la valeur d’une constante, indice ou parametre
  + On peut aussi modifie une decision (eg : < a >=) ou une instruction ( effacer ou changer une declaration)
* **Graphe de flot de contrôle**
  + **Un graphe est represente par des noeuds**
  + Nœud de decision (plusieur sortant)
  + Nœud de jonction (plusieurs entrants)

**TEST BOITE GRIS**

* Strategie de test basée sur connaissance partielle(abstraite) de l’implementattion
* Test Basé sur les modele(ex : archicterure, etats, UML, etc)
* Couveture basé sur un modele

**Test basé sur machine à etat**

* **Systeme à etats**: le comportement depend des evenement passe donc peut avoir des reactions differents a la meme entre entrees
* **Machine à etat :**  abstraction du comportement d’un système a etats ex : Simple, stateChart
  + Modelise la partie du contrôle d’un système
* Dans un modele  il y a un Etat intiale et finale et des transitions
* Les transition consiste : Evenement(ce qui declenche la transition), la garde(la condion optionnelle) et l’Action (operation optionnele)
* **Probleme**: Abscence de model a etats(necessite reverse-engineering), modele a etat incomplet
  + Comment verifier qu’une transition se termine dans un etat specifique :
    - Facile si l’implementation fourni des rapporteur d’etat
    - Sinon des approches indirecte doivent etre utilise comme : UIO sequence, distinguishing sequence ou characterizing sequence ou repetition test

**Fautes de contrôle etat**

* **Dead State(**pas de sortue d’un etat non final)
* **Dead Loop (**Sous ensemnle d’etats qui ne peuvent pas etre sortis une fois entrés)
* **Magic State(** etat ou on peut pas entrer-pas de transition d’entree ou transition vers d’autres etat-etat initiale supplementaire)

**Couverture basee etats :**

* Tout les etats sont passe a travers au moins une fois

**Strategie de Test N+**

* **Revele :**
  + Toute les fautes de contrôle etats
  + Tous les sneak paths
  + Plusieur etat corrompus

**Verification indirecte de l’etat**

* **UIO :** pour un etat donné (sequence d’entree qui profuit une sequence de reponse differente de tout etat)
* Peut fourni une reponse a suis-je dans un etat donne
* Aucune garantie qu’une sequence UIO existe

**Characterixing Sets(CS)**

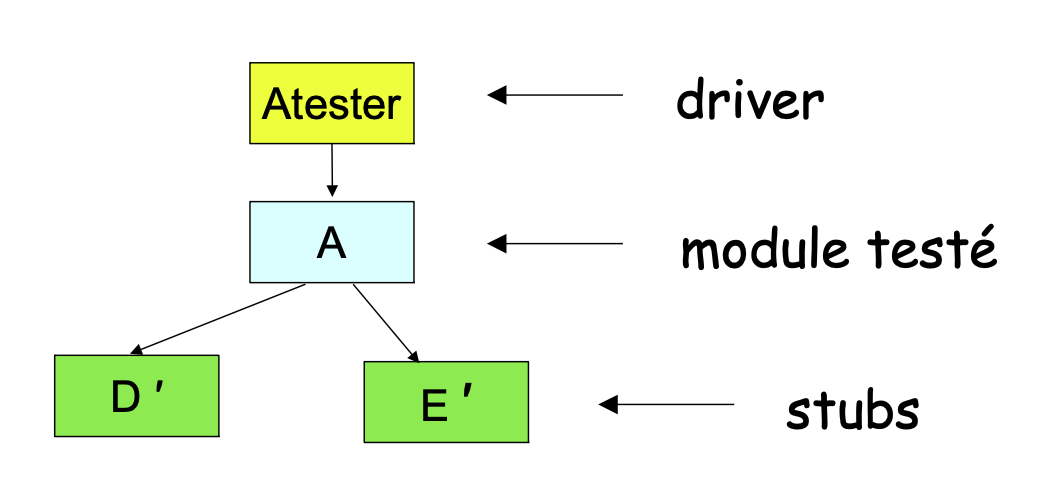
* Un CS pour un etat s est un ensemble de sequence d’entre telle que lorsque chaque sequence est appluque a la machine dans l etat s, l’ensemble des sequences de sortie est unique

**Repetition de Test**

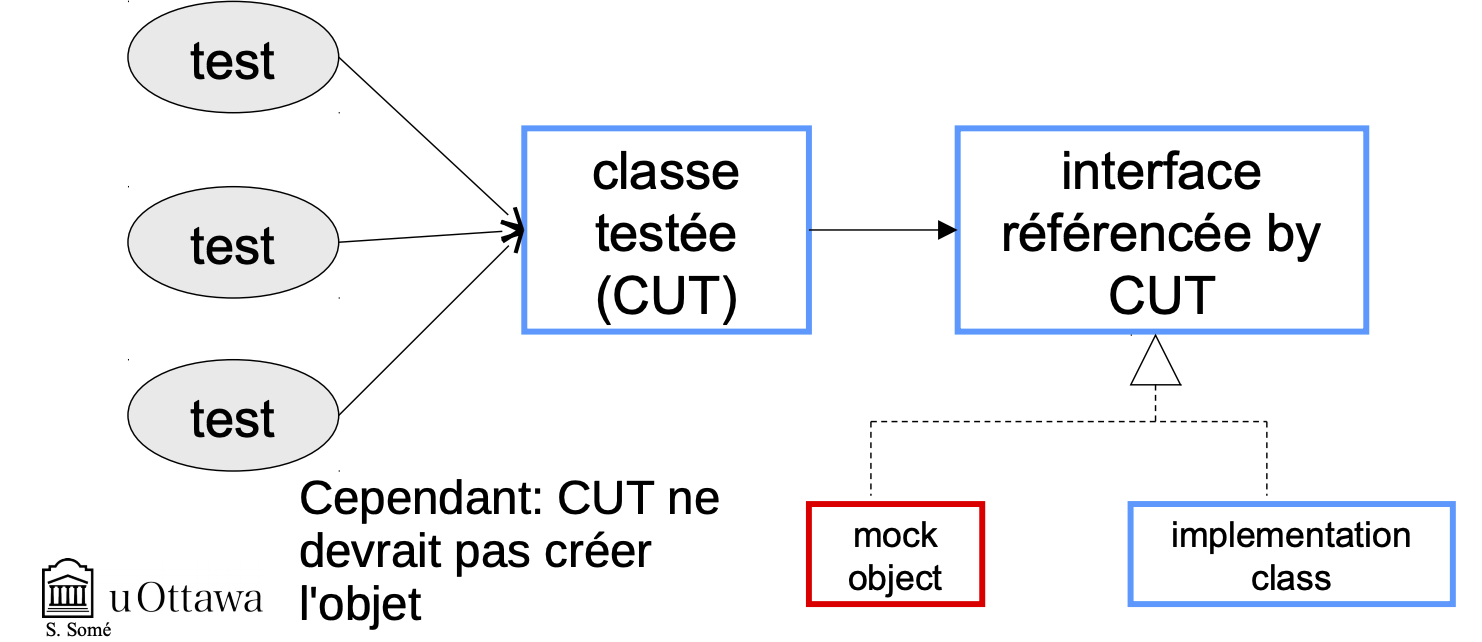
* Heuristique pour reveler les etats corrompus lorsqu’il est impossible d’interroger/examiner les etats
  + Executer le cas de test en sauvergardant toutes les action en sortie
  + Repeter le test et comparer la nouvelle sortie avec celle sauvegarder
* Presenxce d’etats corrokpus lorsque la trace ne reste pas coherente

**TEST INTEGRATION**

* **C est quoi un test d’integration :**
  + Un module logiciel est un element distinct d’un système ou les test unitaite testent les modules indivuellement
  + Les modules doivent etre assemblés pour construire le syteme **ce qui n est pas une tache simple en raison de defaut d’interface potentielles**
  + **Objectif :**
    - Tester les interfaces entre les composants
    - Les interactions avec differentes parties d’un système ex :
      * SE, Système de fichier, interface entre les systemeset materiel
  + **Test d’integration**
    - Construire une version fonctionnelle du système en assemblant les modules**,** tout en veillant à ce que les modules supplementaires fonctionnent comme prevu sans pertuber les fonctionnalite des autres modules déjà assemblé
  + **Defaut d’integration :** 
    - souvent probleme liês a des erreurs d’interface
      * mauvaise hypothese sur les fonctionnalité fournies
      * mauvaise supposition concernant l’interface(ex : mauvais parametre, responsabilité des verification de precondition)
      * mauvais traitement d’erreur
      * hypothese erronées concernant le timing des evenement
  + **Strategie d’integration :**
    - Determiner comment les module de bas-niveau sont assemblés pour former des modules plus haut-niveau – ordre d’integration des composants
    - Impact sur :
      * Forme d’ecriture des test unitaires
      * Type d’outils à uiliser
      * L’ordre de codage/test de unités
      * Cout de generation de cas de test
      * Cout localisation/correction des erreurs detectées
  + On a besoin de savoir la **structure de dependance des composantes**
    - Description de comment les composants sont inter-reliés
* Lors du test d’un module en isolation
  + les modules appelés doivent etre remplacés
  + les modules testé doit etre invoqué

****

* **Driver :**  le module utiliser pour invoquer les modules sous test en envoyant des valeurs au modules en cours de test est recuperant les valeurs retournées
  + Les cas de test JUNIT sont essentiellement des drivers de test
* **Stubs :**  module specifiquement utilisé pour les test qui remplace un module **appelé**
* **Pourquoi :** pour acceder a une interface qui n est pas visible autrement , car le module réel n est pas prêt ou fiableou le module réel necessite un gros effort de mise en œuvre
* **Creation d’un stub**
  + le stub doit avoir la meme signature que le module réel (donc meme non, liste de parametre, type de retour, modificateur)
  + **Fonction du stub :**
    - Verifier que les parametre entrant sont tels que prevu et ca retourne les valeurs appropries
    - NB : le stub ne devrait essayer de reproduire la fonctionnalités du module réel et les valeur de retour preregler doivent etre utilisées
* **MOCK oject :**  utiliser commen un object de substitution ou un stud est necessaire
  + Plus facile a initier et controler que un stub traditionnel
  + Isole le code de details pouvant etre fournis plus tard
  + Peut –etre incementallement raffinés en remplacent par le code réel

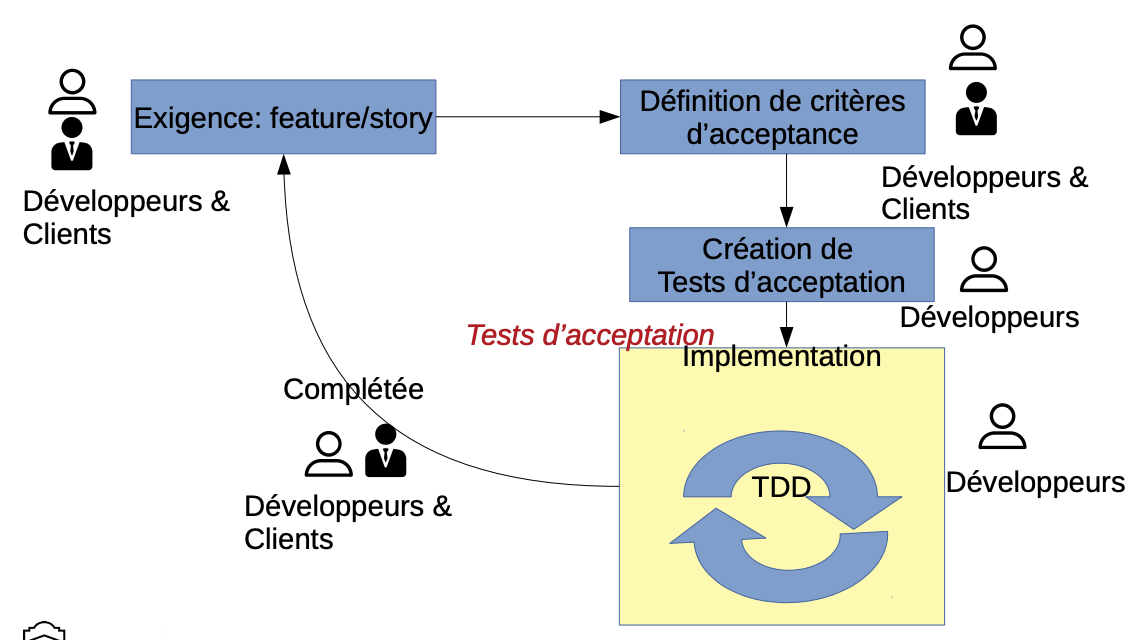
****

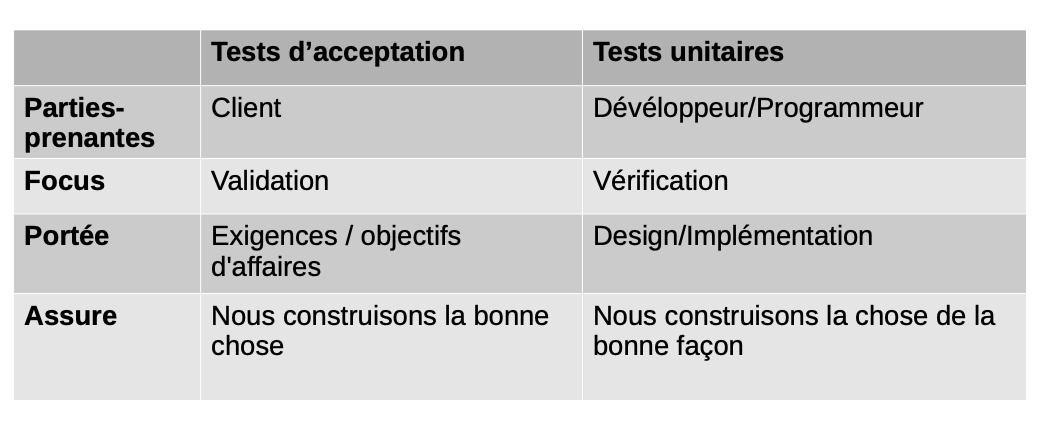
* **Dependency Injection principle (DIP)**
  + Les object ne doivent pas créer pas directement les object dont ils dependent
    - SI l’object A est censé fonctionner avec objet B, A ne doit pa créer B
    - A et doivent plutôt etre créer séparement, puis avoir leur references passées
* **Strategie de test d’integration**
  + critere de comparaison : Localisation de faite, effort (pour stubs et drivers) , degre de test des modules, possibilité de developpement parallele
  + **Integration Big Bang :**  on test tout les composantes ensemble
    - **Avantage :**  convient aux système petit/stables
    - **Des-avantage :** non incrementiel, ne permet pas le developpement en parallele, difficle de localiser les fautes et facile de rater des fautes d’inteface
  + **Integration Top-down :** strategie incrementielle, ou il faut tester les modules de haut-niveau, puis le module appelés jusqu’aux de plus bas niveau
    - **Avantage :**  facilité de localisation de fautes, peu ou pas de drivers, possibilité d’obtenier un prototype, different ordre de test/implementaions possible et les probleme majeurs de design sont trouvés en premier
    - **Des-avantage :** demande plus de stubs, Module potentiellememt reutilisables
  + **Integration Bottom-UP :** strategie incrememtielle ou il faut tester les module de bas-niveau puis module les appelant jusqu’au module de plus niveau
    - **Avantage :** localisation de fautes plus effective que Big bang, pas besoin de stubs, test en profondeur de module reutilisables et les test peuvent etre en paralllele avec l’implementation
    - **Des-avantage :** Besoin de drivers, module de haut-niveau testés en dernier et le moinset pas de concept de squelette prototype
  + **Integration Sandwich :**  combinaison de top-down et bottom-up
    - Distingue les 3 couches :
      * Logique (au dessus)- testée selon top-down
      * Intermediare
      * Operationnele (dessous) –testée selon bottom-up
  + **Integration basées sur le risque**
    - Integrer selon criticalité – module critique ou complexes ainsi que modules qu’ils invoquent intégrés en premier
  + **Integration basée Fonctions/Threads**
    - Integration de modules selon les threads/fonctions auquel ils appartiennent
      * Integration incrementielle de threads
      * Verifie les problemes de multi-threading
  + **Integration intra-classe**
    - Test adequat des operation d’une classe ou methode
      * Approches boite noire , blanche
      * Chque attribut mis a jour et interroge
      * Chaque exception lancée au moins une fois
      * Chque interruption forcée à intervenir au moins une fois
      * Test basés etats
    - Integration de methode
      * Big-banfg indique pour situattion ou les methode sont fortement couplées
      * Methode complexes peuvent-etre tester avec stubs/mocks
  + **Integration de clusters :**  cela demande un graphe de dependance de classes
    - **Approche Big Bang :**  recomander uniquement lorsque le cluster est stable avec peu d’element ou quand le cluster est petit ou quand les composants sont très fortement couplés
    - **Apprioche Bottom-up :** Technique la plus utilisées
      * Integration de classes – debutant aux feuille et allant vers le haut du graphe de dependance
    - **Approche Top-down :** possible uniquement lorsqu’il aune classe de tête(ex cluster avec classe facade)
    - **Approche basée scenarios :**
      * Scenario : decrit des interactions de classe et representé par des diagrammes d’interactions
      * Approche :
        + Lier les collaborations à l’arbre de dependance
        + Choisir la sequence d’application des collaborations(ex : Plus simple en premie,r avec lemoins de stubs en premier , etcc)
        + Developper et executer des tests selon la collaboration
  + **Integration de sous-syteme/système**
    - **Integration Big-Bang :** Non recommandé en general
    - **Integration Bottom-up :** okay pour système de petite ou taille medium
    - **Integration Top-down**
    - **Integration basée sur cas-d’utilisation :** basée sur cas d’utilisation de haut-niveau (externes)
  + **Integration de client/serveurs**
    - Tester chaque client avec stubs pour serveurs et middle-tier
    - Tester serveur avec stubs pour chaque type de client et middle-tier.
    - Tester chaque client avec middle-tier et proxy serveur
    - Tester serveur avec middle-tier et proxy client
    - Tester clients avec middle-tier et serveur actuel
  + **Integration continue**
    - Chaque developpeur integre son code au moins une fois par jour, parfois plusieurs fois par jour
    - le code des developpeurs est unitairement testé puis integré avec le code existant et l’ensemble complet de test d’integration est executer
    - **Avantages :** 
      * Detecyion precoce des defaits d’integration
      * evite le merge hell-lorsque les developpeurs mettent trop de temps à fusionner leur code, entrainant trop de conflits
      * il y a tjr une version deployable du système
      * meilleur suivi du projet
* **Probleme liés a l’heritage**
  + Difficulté de comprehension du code
  + Variable d’instance similaires aux variable globales
  + Suite de test d’une methodes quasiment jamais appropriées pour methode la redefinissant(overriding)
  + Reutilisation accidentelle
  + Heritage de methode non supposée
  + Heritahe multiple( cas de test multiples necessaire, possibilité de liaison incorrecte, heritage repetées
  + Classes abstraites (besoiin d’une instantiation)
  + Classes generiques

**TEST SYSTEME**

**Test fonctionnel**

* **Test systemes/ acceptance**
  + Effecturer a partir de la persperctive d’un logiciels assemblé
  + Donc on verifie la satisfaction des exigences fonctionnalité, securité. Performance (Load, stress), configuration
  + **Test d’acceptance :**  sont des test systemes effectué par les clients ou sous leur responsablité en verifiant si le système fonctionne selon les attentesdes clients
  + **Types commun de test d’acceptance**
    - **Test alpa :** test usagers effectuer sur un syteme pouvant avoir des fonctionnalité incompète dans le cadre de l’environnement de developpement
    - **Test beta :** test usagers effectuées dans le cadre d’environnement de l’usager final
  + **Test de fonctionnalité :**  ce sont des test qui ont comme objectif de s’assurer que le système supporte les exigences fonctionnele requise **et sont derivé des exigence**  sous forme traditionnelle,cas d’utilisations, users stories
  + **Derivation des test a partir des exigences :**  chaque exigence doit etre testable( donc precis, non ambigu, non divisible) alors la description doit clairement identifier ( entrées, conditions, Actions associes et resultats)
    - **Pour chaque exigence :** fournir des cas de test montrant le fonctionnement de l’exigence(test positif) , cas de test essayant de falsifier le fonctionnement(test negatif), un cas pour tester les borne et les contraintes autant que possible
  + **Derivation de cas de test a partir de cas d’utilisation :**  pour tous les cas d’utilisation -> developper un graphe de scenarios, determiner tous les scenarios possible , analyser et ordonner les scenarios, generer des cas de test a partir des scenarios pour atteindre l’objectif de couverture et executer les cas de test
    - **Probleme :** necessite de remplir des gaps. Donc ca se peut que les specifiations soit incompletete et doit etre validé avec l’equipe d’exigence
  + **Derivation de cas de test a partir des user stories :**  les users stories manquent generalement de details pour la genearation directe de test. Il se concentrent sur les objectif des utilisateur. On peut verifier les user stories avec les **critere d’acceptation**
    - **Critere d’acceptation :**  sont des conditions generale d’acceptation sans donner de detail, ils clarifie les contours d’une User Story et confirment lorsque’une story est terminée et fonctionne comme prévu et developpé en collaboration entre clients, utilisateur et developpeurs
    - **Test d’acceptation(clients) :**  derivés des criteres d’acceptation en fournissant des conditions spéficiques d’acceptation et doivent etre formulés dans un langage comprehensible
  + **Acceptance Test Driven Development (ATDD) :** 
    - une bonne pratique du developpement agile de logiciels est de **commencer avec des test d’acceptation**
      * capturer les exigences des parties prenantes
      * affiner en collaboration les exigences en identifiant les criteres d’acceptation
      * créer des test d’acceptation piur les critère d’acceptation
      * implementer en vue de passer les test d’acception
        + impliquer plusieurs cycle de TDD

****

****

**FORCED-ERRR Testing**

* **Objectif :**  rechercher les defauts pouvant resulter d’une gestion incorrecte des situation d’erreurs **en forcant** le systematiquement le système dans toutes les situation d’erreurs
  + **On verife :**  la conception du traitement d’erreurs, consistance des methode de communication, detection/traitement de condition d’erreurs communes, recouvrement du programme de chaque condition d’erreurs et correction d’etats instable provoqués par les erreurs
* **Verification de chacun des messages d’erreur pour s’assurer**
  + La correspondance entre message et type d’erreur
  + la description d’erreur est claire et concise
  + le message ne contient pas de faute d’ortographe et de grammairs
  + L’usager a des options raisonnables pour contourner et recouver des erreurs
* **Comment obtenir une liste de conditions d'erreurs**
  + Une liste de messages d'erreurs de la part des développeurs
  + Interview des développeurs
  + Inspection de chaînes de caractères dans fichier de ressource
  + Information de la spécification
  + Usage d'utilitaire pour l'extraction de chaîne du code binaire ou source
  + Analyse de tous les événements possibles avec un oeil sur les situations d'erreurs
  + Usage de l'expérience
  + Usage de matrice de test standard d'entrées valides/invalides (Partition en Classes d’équivalences)
* **pour chaque condition d’erreur au moins un test qui**
  + force la condition
  + verifie la logique de detection d’erreur
  + verifie la logique de traitement d’erreur en se demandant
    - si l’app offre une possibilité de remission adequate et permet a l’utilisateur de recuperer des erreurs avec elegance
    - si l’app manipule la condition d’erreur avec elegance
    - si le syteme recuper avec elegance
    - lorsque le système est remis en marche , est-il possible que certain service ne redemare pas avec succes ?
  + Verifie la communication d’erreur
    - Si le message d’erreur apparait et correspond bien a l’erreur
    - ND : la communication peut etre par un autre medium tel que audio ou visuel
  + Rechercher d’autres probleme

**Test de securité**

* **Vulnerabilité :** defaut logiciel ou materiel ou mauvaise configuration qu’un individu malveillant peut exploiter pour acces non autorisé ou manipulation
* **Objectif :** identifier les vulnerabilité du système du système puis les supprimer pour proteger le système ( comme l’integrite et la confidentialité des données ou le reseau informatique)
  + Veridier que les erreurs sont gerer correctement
  + Verifier que les cross-site browser scripting ne sont pas possibles
  + Veifier que l’entrée fournie par l’utilisateur ne passe pas non filtrée en tant qu’instruction SQL/NoSQL sur la base de données prenant en charge l’application
  + Verifier que le depassement d’une taille de mémoire tampon ne permet pas l’installation de code malveillant
  + Verifier que les données privées sont proteger pendant leur transit ou leur stockage
  + Verifier les contrôle d’acces
* **Strategie**
  + Assure que les cas d’utilisation « attaque » - utilisation malveillante des ressources système – font partie de la compagne de test
  + Assurer que la securite fait partie de la strategie de developpement
  + Modeliser les possibilités de menaces et utiliser les resultats de l’analyse
  + Focaliser les test sur les elements identifier comme etant les plus a risque
  + Lorsque des entrées illégales sont possible, essayez de créer des entrées qui provoquent deliberement une erreur du programme pour voir si l’application peut entrer dans un etat imprevu
* **Vulnerabilité commune**
  + Injection
    - **Principe :**  tromper un syteme a executer des code ou du code en soumettant des données non fiable
    - **Ex :**
      * **Injection de commande :**  possible lorsqu une app envoue des appels a un système shell peut provoque une vulnerabiltés de commande script lorsque l’app transmet des données securisées fournis par l’utilisateur au shell et peut entrainer une execution arbritraire de commande
        + **Prevention :** utillser les fonctions de la bibliotheque du langage plutôt que les appels direct aux commandes du SE, utiliser les fonctions *escapeshel ou espaceshellcmd* en PHP**, valider les entree** (supprimer les caractere speciaux) et executer avec moins de privileges possibles
      * **Injection SQL :** a la capaciter de lire, modifier, executer des operation administratives dans la bd et recuperer et envoyer du contenu
        + **Causee par :** la capaciye d’inclure des commets dans une commande SQL, combiner plusieurs commandes SQL et la capacite d’utiliser SQL pour interroger le *metadata*  dans un ensemble de tables systemes standard
        + **Prevention :** sanitization des entree, Escapte/Quotesafe des entrées , Emploe de parameters borné, Limitation de l’acces aux table, Emploi de stor-proc pour l’acces a la BD, Configuration du rapport des erreurs
        + **Test :** determier les champs de saisie potentiel qui pouuraient etre utilisés dans une elaboration d’une requete SQL en testant un a la fois en entrant des caractere speciaux et ensuite verifier les message d’erreurs pour detecter l’execution de l’entée brute saisie par le SGBD
  + Authentification defectueuse
    - **Possible lorsque**: des identifiant faible sont autorise (psswrd), des processus de recuperation des identifiant et de psswrd oubliée sont utilisé, cryptage et stockage incorrect des psswrd et les identifant de sessoin est exposer dans un URL, reutiliser, non invalidés correctement
    - **Exemple d’attaque :** credential stuffing, vol de session
    - **Prevention :**
      * authentification multi-facteurs
      * modifier les informations d'identification par défaut (ex: pour les serveurs, le SGBD, ...)
      * vérification automatisée des mots de passe faibles (par rapport aux listes de mots de passe les plus mauvais)
      * utiliser la même réponse pour toutes les erreurs d'authentification, la récupération des informations d'identification
      * limiter/retarder les tentatives de connexion infructueuses
      * enregistrer les tentatives de connexion infructueuses
      * utiliser un gestionnaire de session intégré et sécurisé côté serveur, qui génère des identifiants de session aléatoires
      * invalidation appropriée des identifiants de session après la déconnexion / expiration
    - **Test :** 
      * Tester le transport des informations d'identification
      * Tester les combinaisons userid-mot de passe par défaut ou devinables
      * Testez les scénarios susceptibles de contourner complètement l'authentification:
      * Traversée de répertoire: test qu'un utilisateur ne peut pas voyager en dehors de la structure de dossiers d'une application
      * Tester l'accès à la session après la déconnexion:
  + Exposition de données sensibles
    - **Prevention :**
      * Classifier les donnes traitées, stockees ou transminser par rapport a leur sensibilitee
      * Ne pas stocker des données sensibles inutlement
      * Crypter toutes les donnes sensibles stockees avec des algoritme, protocole et cles standard mis a jour et performants
      * Crypter toutes les donnees en transit avec des protocoles securisees(ex TLS)
      * Desactiver la mise en cache pour les reponses contenant des données sensibles
  + XML External Entities
    - Possible lorsque l’app accepte du XML qui est analysé par un processeur XML(request SOAP, REST) ou lorsque le processur XML supporte les references d’entité dans les definitions de type de document (DTD)
    - **Prevention :**
      * Utiliser des formats simples que XML(ex ; JSON)
      * Corriger ou mettre a niveau tous les processeurs et librairies XML
      * Desactiver le traitement des entités externe XML et des DTD dans tous les analyseurs syntaxiques XML de l’application
      * Limiter la quantité de mémoire utilisée par les processurs XML
      * Valide/sanitizer les entrées
  + Contrôle d’acces defectueux
  + Mauvause configuration de securité
  + Cross-Site Scripting(XSS)
    - Attaque où du code script est injecté ds un site web de confiance pour etre executer par les fureteurs de clients
    - **Impact :** permet a un attaquant de voler de l’info d’identification/Cookies, Deni-de-service, Modifiaction de pages Web, Excution de commande sur la macchine cliente
    - **Type** 
      * **Reflected**(non-persistant)**:** via lien d’un site web ou par courrel
      * **Stored** (persistant)**:** via forum, bulletin board, formulaire de commentaires
      * **Base sur le DOM** (Ajax) : injection de script comme fragement
    - **Prevention :**  validation des données (pas seulement du cote client) en utilisant des framework appropiés et Encodage des sorties
    - **Test :**pour chaque pts d’entrées de données usager (ex : parametre requetes, cookies value, entete http) -> entrer des caracter de scripts (ex : <, >) scripts et verifier les sorties pour s’assure du filtrage approprié, la non execution de scripts,--
  + Cross-Site Request Forgery (CSRF)
    - Tromper un usager a executer des actions indesirables sur un web app ds laquelle il est actuellement authentifié
    - **Attaque basée** sur le Traitement par navigateur Wb d’info relative aux sessions(ex : cookie, authentification info http), Connaissance par l’attaquant des URL d’application web valideset sur la Gestion de session d’applications reposant uniquement sur des info connues par le navigateur
    - **Prevention :** 
      * Referer headers :indique l’URL du document d’origine de la requete, donct les attaques CSRF peut etre evite en ignorant les demande avec un referent inattendu, souvent desactivite pour confidentialité
      * Re**-**authentification : exiger la re-authentification de l’utilisateur pour chaque actions sensibles. Cela peut entrainer une mauvaise utilisabilités
      * Synchronizer Token pattern :jetin secure aleatoire(ex : CSRF token) qui est ajouté en tant que champ masqué pour les formulaires ou ds l’URL pour l’operation GET. Alors ‘app rejecte l’action sile jeton echoe la validation
  + Depassement de buffer (Buffer overflow)
    - Consequence de transferts de données vers des chaines/tableaux en memoire sans controle de borne approprie
    - Vient comme stack, heap, Integer overflow
    - **Prevention :** assurer l’usage de function **safe**  pour les chaine de caractere (ex:fget, strncpy, ..)
    - **Test :** 
      * Envoyer de grosses quantités de données
      * verifier les borne des buffers -> données exactement de la taille du buffer -> données de la taille du buffer -1 **->**  données la taillle du buffer +1
      * ecriture de caractere speciaux et espace
* **Surface d’attaque :** points d’interface ou un user malveillant peut accede a une application **comme** API, pipes, shared memory, sockets, port reseau, fichier, etc..
* **Tests de gestion de session**
  + identifiées les sessions
    - Possibilités: cookie, session ID, champs caché (hidden field)
    - Les identificateurs sont-ils...
      * décodables? (ex: cryptés de manière non sécurisée à partir de quelque chose comme un nom d'utilisateur)
      * reutilisables?
      * prédictables?
  + Des cookies valides peuvent-ils être générés par un utilisateur?
  + Les variables de session sont-elles exposées?
* **OWASP Web Application Security Testing Guide :** Open Web Application Security Project
  + Recommende la combinaison d'approches automatisées et manuelles: utilisez des outils mais connaissez leurs forces et leurs limites ●
  + TQuatre éléments: – Inspection manuelles et revues – Modélisation de menaces (Threat modelling) – Revue du Code – Tests de Pénétration
* **Modélisation de Menaces :** pour évaluer un logiciel par rapport à la sécurité il faut identifier les zones du logiciel susceptible d’etre exploitées pour les attaques de securité
  + **Etape de modelisation :**
    - Rassembler l'équipe de modélisation de menaces (développeurs, testeurs, experts de sécurité)
    - Identifier ce qui pourrait-intéresser les « pirates »
    - Créer une vue d'ensemble de l'architecture (parties majeurs du système ainsi que leur mode de communication, frontières de confiance entre parties)
    - Décomposer l'application (identifier comment/où les données circulent à travers le système, quelles sont les mécanismes de protection de données) basés sur les diagrammes de flots de données et d'états Modélisation de Menaces
    - Identifier les Menaces en considérant chaque composant comme cible,
      * est-ce possible de consulter/attaquer/modifier les composants?
      * est-ce possible de prévenir l'accès d'usagers autorisés aux composants?
      * est-ce possible d'avoir accès et de prendre le contrôle du système?
    - Documentez les Menaces (description, cible, forme d'attaque, contremesure) 7
    - Classer les menaces selon: • Potentiel de dommage • Reproductibilité • Exploitabilité • Usagers affectés • Potentiel de découverte
* **Tests de Pénétration Web :** tentative de violation d'un système en exploitant des vulnérabilités. On essaie de tester une application en cours d'exécution (à distance) pour trouver des failles de sécurité, sans connaître le fonctionnement interne de l'application elle-même en combinat une varieté de test de vulnerabilité
  + **Etape :**
    - Collecte d'informations
    - Tests de configuration et de gestion du déploiement
    - Tests de gestion d'identité
    - Test d'authentification
    - Test d'autorisation
    - Test de gestion de session
    - Test de validation d'entrée
    - Gestion des erreurs
    - Cryptographie
    - Test de logique d’affaire
    - Test côté client
* **Collecte d'informations:** 
  + Découverte et reconnaissance des fuites d'informations
  + Spidering (crawling) - pour mapper les chemins de l'application, découvrir les pages cachées
  + Fingerprinting – pour découvrir des informations sur – web server – framework d’application – application web
  + Examen des commentaires HTML et des métadonnées
  + Cartographie de l'architecture de l'application

**Test de configuration :** on testent la configurabilité du syteme, la compatibilité avec des configuratin de logiciel/materiel et le degré avec lequel une app ou composant peut-etrre configuré en variantes multiples

* **Compabilité :** 
  + on effectue des test de compatibilté avec des configuratiobs logiciel/materies. On regarder :
    - compatibilité avec les autres ressources système de l'environnement d'opération . ex: logiciels, bases de données, standards, etc.
    - compatibilité code source- ou objet avec différentes versions de l'environnement d'opération
  + Compatibilité en arrière/en avant
    - Compatibilité en arrière -> Compatibilité avec les versions précédentes du logiciel
    - Compatibilité en avant -> Compatibilité avec les versions futures du logiciel
* **Tests de Configuration :**  test qui consiste en l’ececution d’ensemble de test sous differentes configiration. Ce sont des test exhibant les principales fonctionnalités du syteme tout en verifiant les exigences non-fonctionnelles du système – ex : perfomance , utilisabilité
  + **Difficulté :**
    - nombre énorme d’options de configuration –
    - nombre énorme de composants matériel/logiciels
    - infrastructure nécessaire pour mettre en œuvre et exécuter des tests
  + **Difficulté : Solution**
    - Nécessité de sélectionner les configurations à tester
      * décider du type de composants devant être testé
      * sélectionner les marques, modèles, drivers à tester
      * décider quels caractéristiques, modes, options du matériels sont possibles
      * appliquer des méthodes d’élimination de composantes . ex: basé sur la popularité, l'âge
    - Methode combinatoire (t-way)
      * peut-être utilisée pour reduire le nombre de combinaison

**Test de perfomance :**

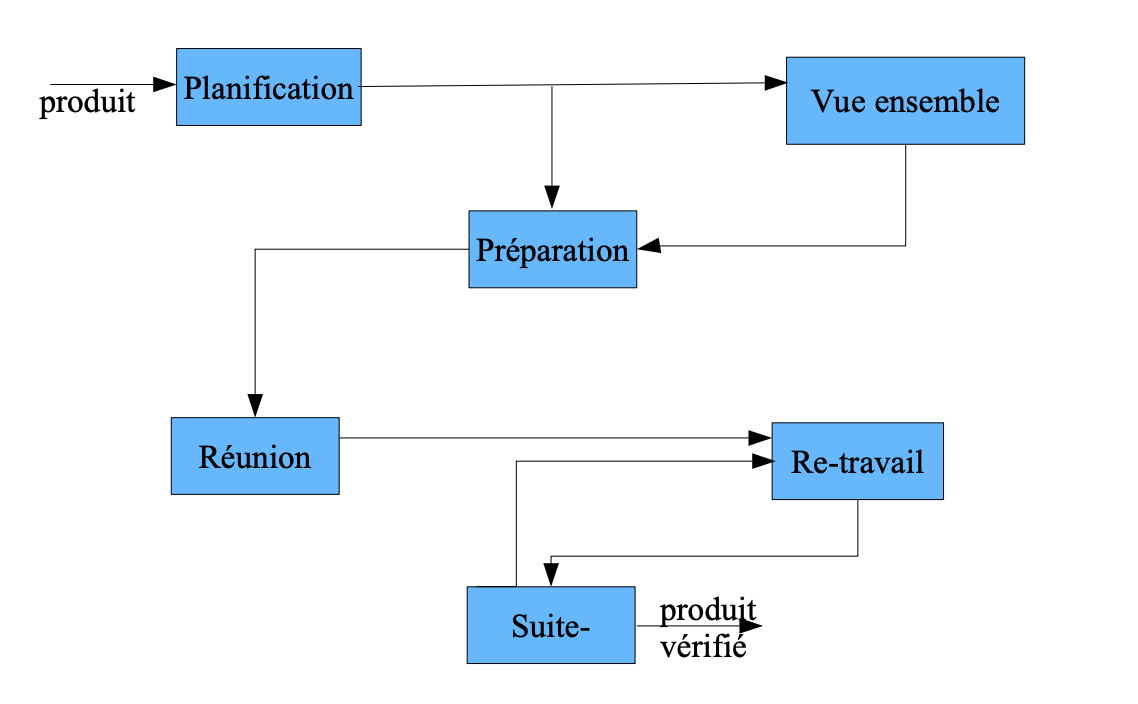
* Destiner à determiner/valider la reactivité, la vitesses , scalabiltté, caracteristique de stabilité d’un systemes
* C**ella est conduit pour**: evaluer l’etat de preparation a la mise en production, evaluer p/r aux exigences de performance, comparer les performances de plusieurs systemes ou configurations système, trouver la source des problemes de perfomance, supporter le reglage du système,et trouver le niveaux de debit acceptable
* **Objectif est de repondre a des questions telles :**
  + Combiend’utilisateur le systme peut-il gerer tout en maintenant un temps de reponse raisonnable
  + Le système peut-il servir X utilisateur sans erreur ?
  + Quel est le temps de reponse pour les differents composants de l’application etant données une charge X
  + Quels composants logiciel/materiel affectent le plus la perfomance
* **Les test de perfomance aide a determiner :** 
  + La capacité maximale pouvant etre gerée tout en conservant une perfomance acceptable
  + Si le reglage des perfomances materielles et/ou logicielles est necessaires
* **Les differents sous-categories** 
  + Test de charge (Load test)\
    - Verifie le support d’une charge particuliere tout en maintenant des temps de reponse acceptable -> on verifie differents modele de charge (ex : conditions de charge normales, conditions de charge de pointe, ..)
  + Test de stress
    - On veut se concentrer sur le comportement du sytemes aprs ou au-dela des condition de surcharge -> on pousse le syteme a l echec , emphase pres de la charge specifiée , limite de volume, verifie les defaillance gracieuses et la degradation non abrupte des perfomances
  + Test de capacité
  + Test d’endurance
    - On veut determiner / valider les caracteristique de performance lorsque un systeme est soumis a des modeles de charge de travail sur une longue periode de temps
  + Perfomance Smoke Test
    - Faite pour determiner le nombre de user et/ou de transactions qu’un systeme donné prendra en charge tout en respectant les objectif de performance
  + Test de pointe (Spike testing) :
    - On determine/valide les caracteristique de performance lorsque les volumes de charge augmentent de facon repetée au-dela des conditions de surcharge pendant de courtes periodes de temps
* **Base de reference de perfomance (baseline) :**
  + On concoit des test pour capturer des données de mesure de perfomances en vue d’evaluer l’efficacité des modifications ulterieures au système ou a l’app
    - on etablit la norme de comparaison pour suivre les optimisation ou regresssion dans le future
    - Toutes les caracteristiques et configurations, a l exception de celles qui sont specifiquement modifiées pour la comparaison doivent rester les memes afin de pouvoir etre comparées avec une base de reference
    - Une reference peut etre crée pour un système, un composant ou une application
* **Processus de test de performance**
  + **Identifier l’environnement de test**: materiel, logicilel et configuration reseau
  + **Identifier les critères d'acceptation**: temps de réponse, débit, ressources objectifs d'utilisation et contraintes
  + **Planifier et concevoir des tests:** créer des scénarios de test réalistes
  + **Préparer l'environnement de test:** setup de l’environnement, outils et ressources nécessaire pour effectuer les scénarios de tests prévus
  + **Enregistrer le plan de test**: enregistrement des scénarios de test selon les outils sélectionnés
  + **Exécuter les tests**: exécution des plans de test - en commençant par une charge légère pour vérifier l'exactitude des scripts de test et des résultats de sortie
  + **Analyser les résultats**, signaler et retester: identifier les zones de goulot d'étranglement qui nécessitent une attention (par exemple, système, base de données ou application)
* **Optimisation de perfomance** 
  + Pour resoudre les probleme de perfomance depend du domaine de préoccupation ex :
  + changements dans les ressources système (ex: mémoire, processeur, bande passante, ..)
  + optimisation de la base-de-donnée
  + extension de la capacité de l'application / du serveur de base de données
  + re-architecture ou re-conception de l'application

**Test de fiabilité :**

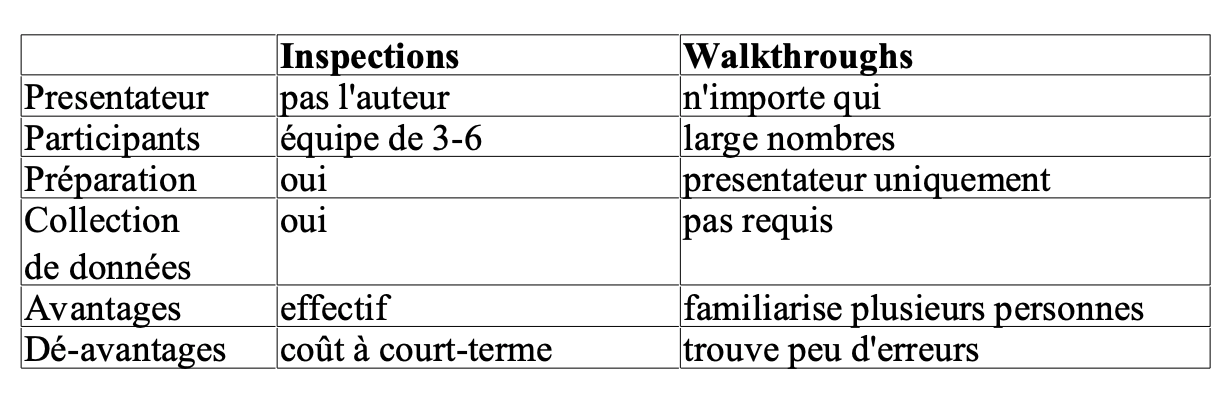
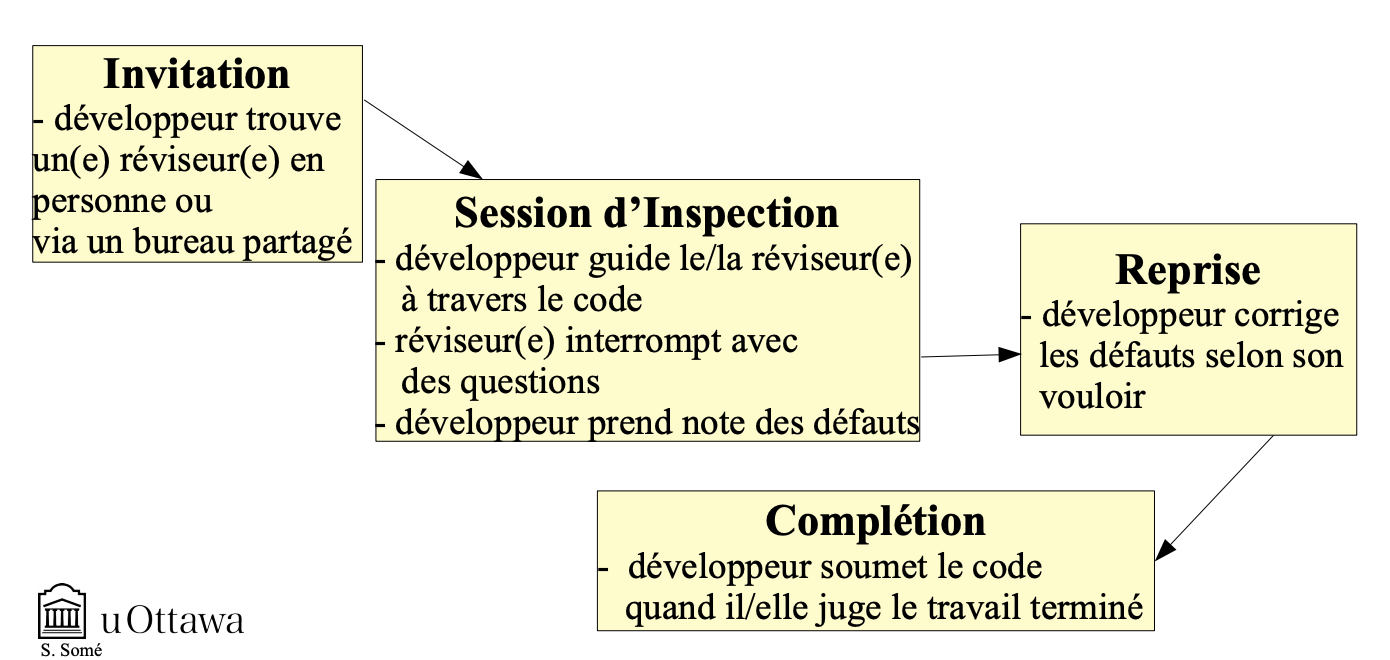
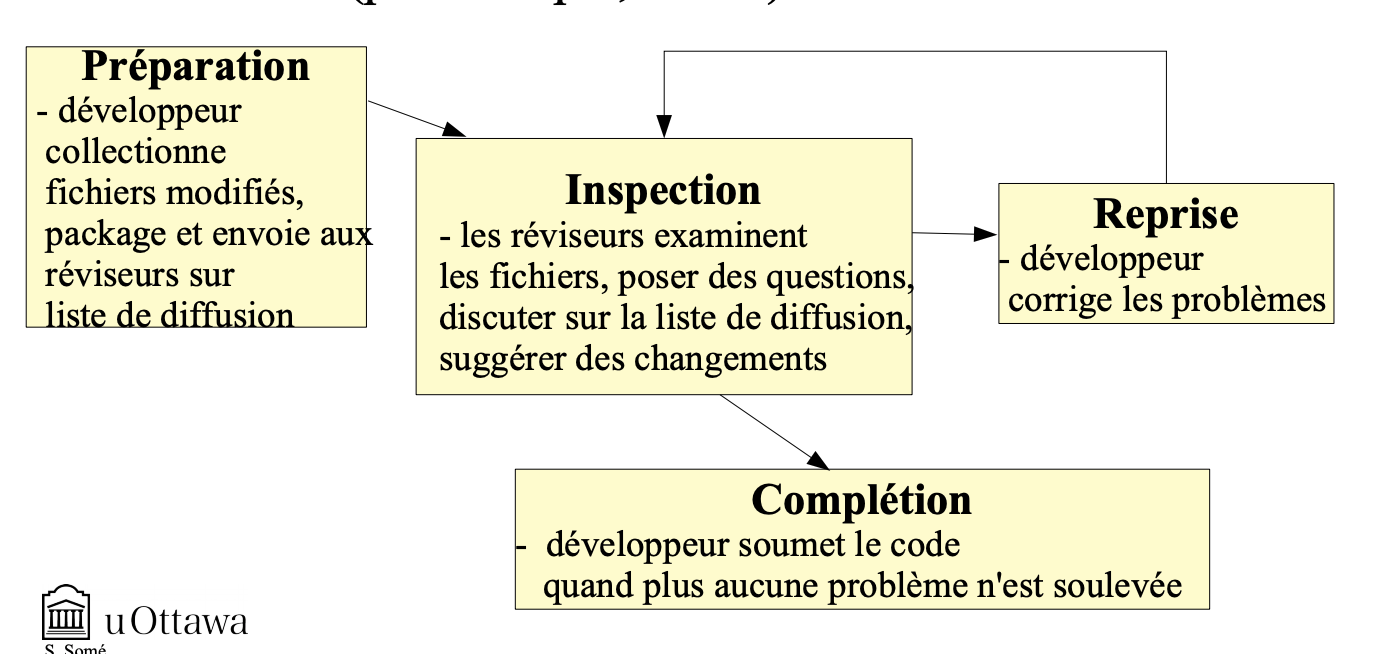
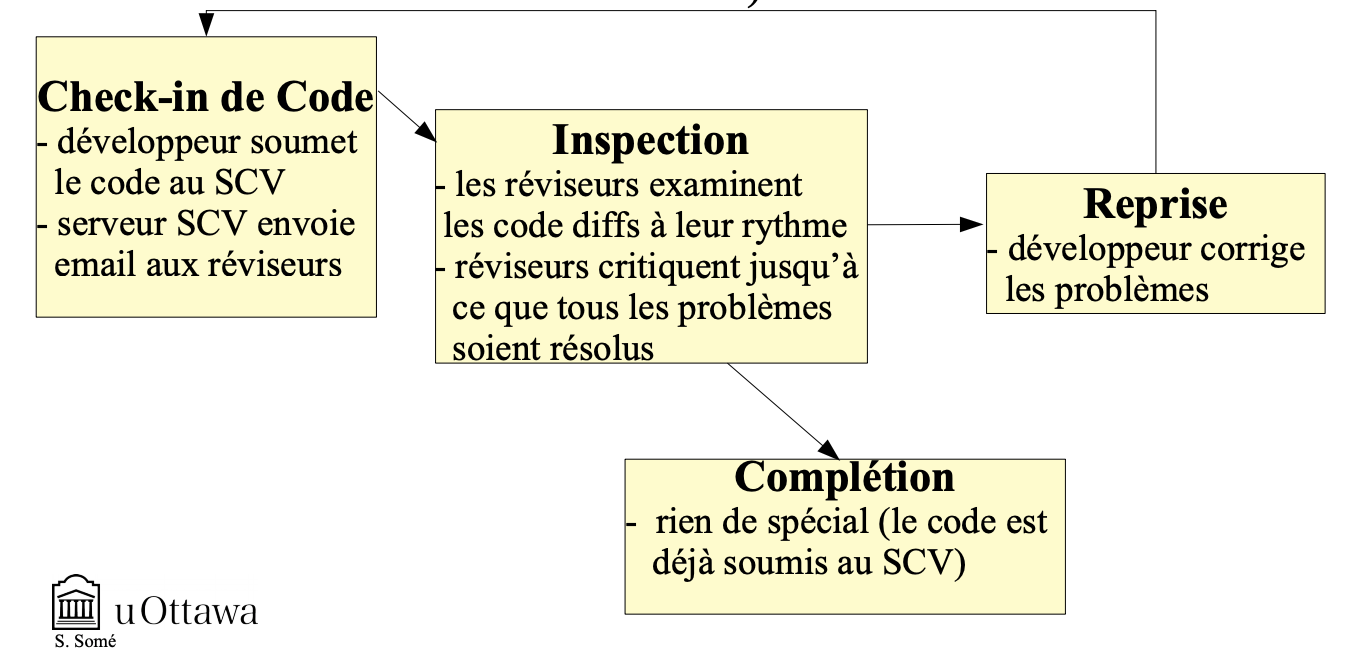
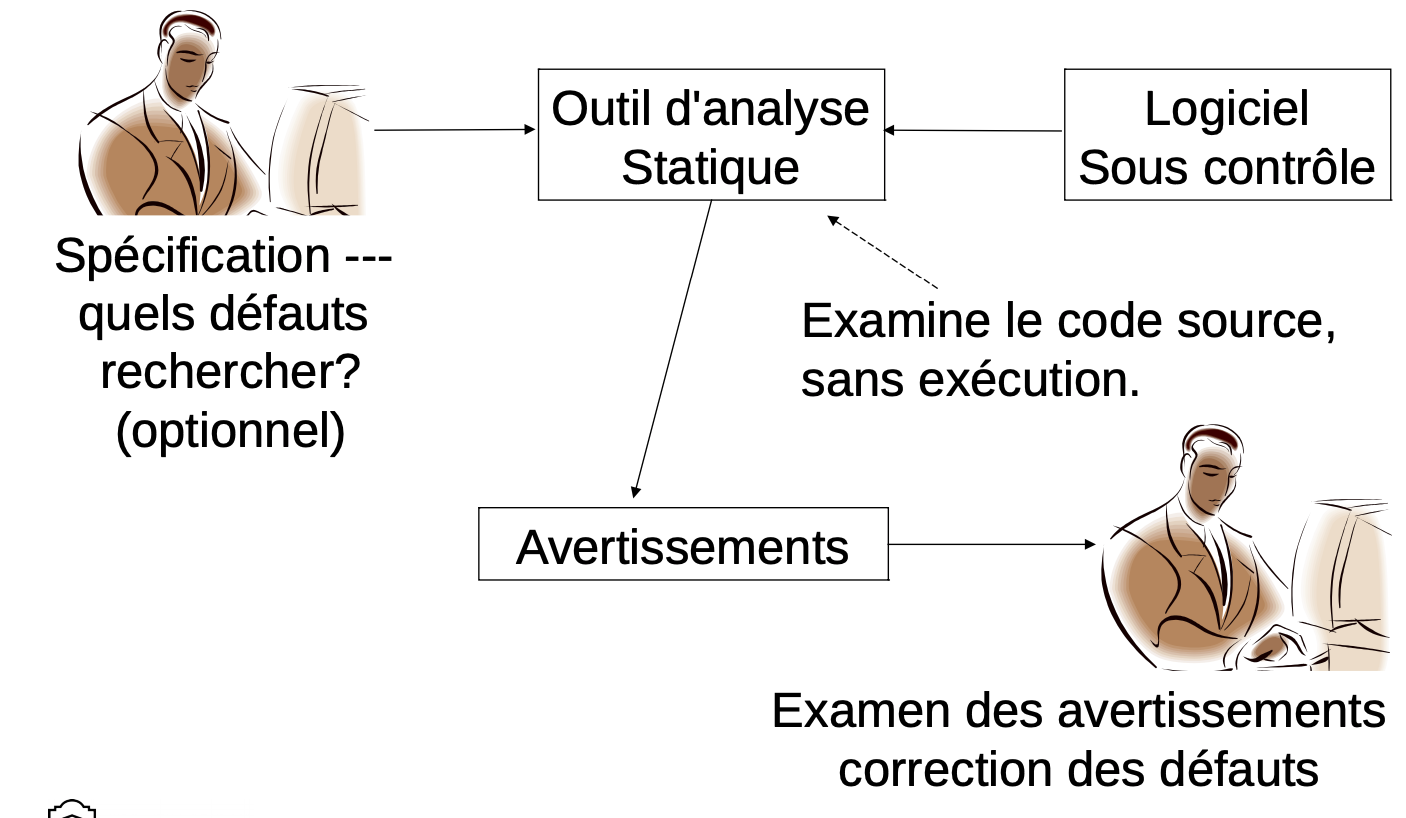
* c’est une application de technique statique a des donnees collectees durant le developpement et operation(profil operationnel) pour specifier, predire, estimer et evalure la fiabilité d’un syteme
* Les exigence de fiabilté sont exprimer en terme de probabilite de non pannne dans un intervalle de temps specifie et en temps moyen entre pannes escompte (MTTF)
* Les test de fiabilte / statistique sont base sur un **modele d’usage** 
  + Developpement d’un modelel usage operationnel de l’utilisation de l’utilisation du logiciel
  + Genreation aleatoire des cas de test du modele d’usage
  + Interpretation des resultats de test selon les modeles mathematiquees et statistique pour obtenir des mesures de qualité de logiciel et pour examiner la suffisance
* **Modele d’usage :** c est une representation des usages possible du sytemes
  + **Peut etre specifie selon different contexte :**  contexte d’usage nomarl, conditions de stress, condition dangereuses et conditions de maintenance
  + **Peut etre representer sous forme graphe de transition oú** les nœud sont des etats d’usage et les arcs sont des transition entre etats d’usages
* **Ex – System d’alarme**
  + Pour portes, fenêtres, etc. -> détecteur envoyant un signal (trip signal) en cas de mouvement
  + Activé en pressant bouton Set. -> Lumière dans bouton Set illuminée lorsque système est on
  + Alarme émis si signal trip pendant que système est actif
    - Code de 3 digits doit être entré pour stopper l'alarme
    - En cas d'erreur lors de l'entrée du code, usager doit appuyer sur bouton Clear avant de recommencer.
  + Chaque unité a un code de dés activation pré-codé
* **Les probabilité d’usage sont obtenues de**  donnee du terrain, estimation a partir d’interview des clients et instrumentation de version precedentes du systemes
  + Pour que l’approche soit effective, les probabilités doivent refleter l’usage futur
* **Analyse du modele d’usage :** c est basé sur des calculs standard de chaine de Markov
  + Possible d’obtenir des estime pour :
    - **Temps de residence de chaque état** – le profil d'usage en tant que pourcentage du temps passé dans chaque état.
    - **Probabilité d'occurrence des états** – probabilité d'occurrence de chaque état lors d'un usage aléatoire du système.
    - **Fréquence d'occurrence des états** – nombre escompté d'occurence de chaque état lors un usage aléatoire du système.
    - **Première occurrence de chaque état** – le nombre d'usage escomptés du systeme j'usqu'à la première occurence de chaque état.
    - **Longueur de séquence escomptée** – le nombre escompté de transitions d'états lors d'un usage aléatoire du système; taille moyenne des cas de test
* **Generation aleatoire de cas de test** 
  + En traversant le modele usage selon les probabilite de transition
  + A chaque cas de test , on debute au nœud initiale et termine a un nœud final et cela consite en succesion de stimulus
  + Cas de test sont un parcours aleatoire du modele d’usage – generateur aleatoire est utiliser a chaque etat pour determiner le stimuli suivant
* **Comment mesurer que les test suffit (quand arreter les test)**
  + **Chaîne d'usage** – modèle d'usage utilisé pour générer les cas de tests. Cela permet de déterminer le temps de residence des états
  + **Chaîne de tests** – utilisés pendant les tests pour suivre les états traversés
    - ajout de compteur initialisé à 0 à chaque arc
    - incrémentation du compteur d'un arc lorsqu'un cas de test exécute la transition
  + **Discriminant-** différence entre chaîne d'usage et chaîne de tests (degré selon lequel l'expérience de test est représentatif de l'usage escompté)
  + **⇒** les tests peuvent stopper lorsque le discriminant connait un plateau
* **Mesure de fiabilité**
* **Etats de pannes** – ajoutés à la chaîne de test lorsque des pannes surviennent lors des tests
* **Fiabilité du logiciel** – probabilité d'effectuer un parcours aléatoire à travers la chaîne de test de l'invocation à la terminaison sans rencontrer d'état de panne.
* **Mean Time to Failure (MTTF)** – nombre moyen de cas de tests jusqu'à ce qu'une panne survienne

**INSPECTION ET ANALYSE**

**Revue du logiciel et analyse statique**

* **Revue du logiciel**  est un examen d’un produit par humain pour trouver des defauts , evaluer la qualité, se familiariser avec le produit, effecturer un audit
  + Cela est applicable a tout doc produit(exigences, design, code source, cas test, etc..) **->** les tests ne peuvent etre utiliser que sur une implementation executable
  + Prouvé comme etant une activite essentiel pour la qualite
  + **Les types :**
    - **Inpesction-** tres structuré et formel
      * Introduiy par Fagan chez IBM comme forme de revue tres formelle pour detecter les defaurs, collectionner des donnee et communiquer des doc
      * Fait en equipe 3 à 6
        + ***Moderateur****:* personne tres important qui assure que la procedure d’inspection est bien suivie. Il verifie que le produit est prêt pour l’inspection et que le produit satifait au critre d’entree d’inspection. C est lui qui rassemble l equipe et garde la reunion dans les bornes et verifie que les criter de sortie d inspection sont atteint
        + ***Enregisteur :*** Il se charge de documenter les defaut trouver lors de la reunion dans une liste de defaut d’inspection du logiciel
        + ***Critique :*** Il analyse et detecte des defaut dans le produit - tout le monde font ca
        + ***Lecteur :*** mene l equipe lors de la reunion en lisant a haute voix de petit unite logique, paraphrasanr le cas echeant
        + ***Producteur :*** auteur du produit et il est reponsable de la correction de tout les defaut trouvé
      * **Processus d’inspection**
        + 
        + **Suite :** Réunion entre le modérateur et l'auteur pour déterminer si des défauts trouvés au cours de la réunion d'inspection ont été corrigés et pour s'assurer qu'aucun défaut additionnel n'a été introduit et Assemblage et recapitilation de données d'inspection finale, clôture officielle de l'inspection.
      * **Rapport d’inspection**
        + Rapport de Notification d'inspection – pour déclencher le processus d'inspection
        + Liste de défauts d'inspection – identifie la localisation, description, sévérité des défauts
        + Sommaire des défauts d'inspection – défauts selon sévérité – ex : Majeur (provoquerait le système à ne pas satisfaire à une exigence) , Mineur (tous les autres) ●

type (données, satisfaction des exigences, interface, ...)

* + - * + Rapport de gestion d'inspection
      * **Directive d’inspection de Pressman**
        + Passez en revue le produit, pas le producteur.
        + Déterminez un ordre du jour et maintenez-le.
        + Limitez la discussion et la réfutation (90 à 120 minutes pour la réunion).
        + Listez les domaines problématiques, mais la résolution des problèmes devrait être remise à plus tard, après la réunion de revue.
        + Prenez des notes écrites (par exemple sur un tableau).
        + Limitez le nombre de participants et exigez sur leur préparation.
        + Développez une liste de contrôle (checklist) pour chaque produit susceptible d'être passé en revue.
        + Assignez les ressources planifiez les revues.
        + Formez les critiques.
        + Revissez le processus de revue.
      * **Avantage :** Plusieurs études montrent que les inspections sont efficaces pour détecter les défauts et garantir la qualite
      * **Incovenient :**  couteux (plusieur personnes soont implique , prend bcp de temps), processus lourde ( rigide, bureaucratique )
    - **Walkthrough –** moins formel, format proche de on a presentation
      * Moin formel que inspection , fait pour detcter des defauts, se familiariser avec le developpement
      * En equipe de 2 a 7,  le matériel doit être distribué en avance (Seul le présentateur doit se préparer ) et Chaque participant liste ses défauts potentiels et points à clarifier
      * Peut-être dirigé par n'importe qui et les changements seulement suggérés (Investigation subséquente non incluse dans le processus
      * **Inspections vs WalkThrought**
      * ****
    - **Buddy check –** verification informelle par personne tiece
      * **Revue over the shoulder** (type informel de revue de code)
        + ****
        + **Avantage :** facile à mettre en œuvre – peut travailler à distance (ex : avec le partage de bureau et les conférences téléphoniques)
        + **Desavantage :** aucune vérification que les défauts sont corrigés – le relecteur est guidé au rythme du développeur – on ne voit que le code sur lequel on a travaillé (facile de passer par-dessus d'autres fichiers modifiés / affectés) – processus volontaire (impossible à mandater) – aucune métrique n est collecter pour la mesur/amelioration du processus
      * **Revue par transmission d email** (technique pour open source projet)
        + ****
        + **Avantage :** fonctionne bien avec les projets distribués – facile d’impliquer des réviseurs avec différents niveaux d'expertise – les réviseurs peuvent travailler à leur rythme et à leur rythme
        + **Desavantage :** trouver et collecter tous les fichiers peut être fastidieux – les réviseurs doivent extraire les fichiers et créer des diffs – nombre de communications peut devenir accablant et difficile à gérer par les auteurs – difficile de savoir quand la revue est complète – aucune métrique n'est collectée pour la mesure/amélioration du processus
      * **Revue Email Pass-Around Apres-check-in**
        + effectué après check-in du code au Système de Contrôle de Version (décharge de la collecte, de packaging des fichiers et des diffs manuels)
        + ****
        + **Avantage :** – même que Email Pass-Around plus – packaging, création de diffs par le SCV
        + **Desavantage :** nombre de communications peut devenir accablant et difficile à gérer par les auteurs – difficile de savoir quand l'examen est fait – aucune métrique n'est collectée pour la mesure / l'amélioration du processus
      * **Pair programming (programmation en binome)**
        + Pratique de développement logiciel agile
        + Une paire de développeurs code ensemble sur un seul poste de travail – Pilote écrit le code – Navigateur passe en revue le code, souligne les problèmes, suggère des améliorations, suggère des tests, … – Changent de rôle souvent
        + **Avantage :** efficace pour la recherche de bugs et le transfert de connaissances – réviseur est très proche du code – les développeurs avec la bonne personnalité aiment cela
        + **Desavantage :** – réviseur risque de rater des problèmes car trop proche du code – consomme beaucoup d'effort – ne convient pas à chaque personnalité de développeur – ne fonctionne pas avec les développeurs distants – pas de métrique pour mesure/amélioration de processus
    - **Revue personnelle -**  verfication par autuer
  + **Manuelle esstientille –** toutefois certainnes activite peuvent etre supportées par des outils
  + **Check List**
    - Outil le plus important pour revues.
    - Check-lists génériques existent pour types variés de revues
      * exigences
      * conception fonctionnelle
      * code générique
      * code en langage spécifique (C, C++, Java, etc.)
      * document générique
    - Organisations développent des check-lists spécifiques
      * objectifs particulier, expérience passée, etc.
    - Doivent être maintenus - améliorés, développés, mis-à-jour
  + **Analyse statique :**
    - Dérivation automatisé d'informations sur un programme – sans l'exécuter – analyse automatisée de code (bytecode)
    - L’objectif est de découvrir automatiquement les défauts potentiels d’un programme
    - Limitations – potentiel de
      * **Faux négatifs** – défauts non rapportés
      * **Faux positifs** – fausses alarmes
    - 
  + **Types :**
    - Analyse de flots de contrôle.
      * recherche les boucles avec plusieurs points de sortie ou d’entrée, trouve le code inaccessible, etc.
    - Analyse de flots de données.
      * détecte les variables non initialisées, les variables écrites deux fois sans affectation intermédiaire, les variables déclarées mais jamais utilisées, etc.
    - Analyse d'interface.
      * vérifie la cohérence des déclarations de routines et de procédures et leur utilisation (ex: séquences d'appels d'API)
    - Correspondance syntaxique de patrons de code (Syntactic pattern matching)
      * vérifie la présence de patrons/idiomes de code «suspects» connus
    - Interprétation abstraite
      * mappe des valeurs sémantiques concrètes en valeurs sémantiques abstraites plus simples / calculables, analyse sur des représentations abstraites
  + **Defaut qu’on peut trouver :**
    - Buffer overflows/underflows
    - Buffer overrun/underrun
    - Integer overflows
    - Incohérences dans la hiérarchie des classes
    - Violations d'accès simultanés aux données
    - Division par zéro
    - Déréférence de pointeurs Null
    - Memory leaks
    - Dead code
    - Vulnérabilités de sécurité (ex: XSS, CSRF)
    - Conformité aux normes de codage
  + **Analyse statique de programmes** 
    - Exemple: vérifie que chaque opération d'un programme ne causera jamais d'erreur (division par zéro, dépassement de tampon, blocage, etc.)
    - **Analyse de flots de données simple**
      * Actions de flot de données typiques sur une variable de programme
        + définition (d) –
        + réference (r)
        + un-définition (u)
      * Pour chaque paire d'action successive
        + dd - suspect
        + du – probablement un défaut
        + dr – cas normal
        + ud - okay
        + uu - probablement un défaut
        + ur - un defaut
        + rd - okay
        + ru - okay
        + rr - okay
      * Première occurrence d'une action de variable
        + u – suspect
        + d - okay
        + r - suspect (variable globale ?)
      * Dernière occurrence d'une action sur une variable
        + u - okay
        + d - suspect (définie jamais utilisée)
        + r - okay (peut-être oubli de désallocation)