Tests logiciels

Emmanuel Grolleau

Observatoire de Paris – LESIA – Service d'Informatique Scientifique

Master 2 « Outils et Systèmes de l'Astronomie et de l'Espace »

Sommaire

Introduction et rappels

- Pourquoi tester ?
- Notions de qualité logicielle
- Rappel sur les cycles de développement logiciel
- Agilité

Les tests

- Introduction
- Classification des tests
- Stratégie de test, plan de test
- Anomalie
- Matrice de couverture
- Intégration continue et Framework de test

Sommaire

Introduction et rappels

- **Pourquoi tester ?**
- Notions de qualité logicielle
- Rappel sur les cycles de développement logiciel
- Agilité

Les tests

- Introduction
- Classification des tests
- Stratégie de test, plan de test
- Anomalie
- Matrice de couverture
- Intégration continue et Framework de test

Activité de test : constat

- Souvent négligée et peu formalisée
- Considérée (à tort) comme moins « noble » que le développement
- Coût souvent > 50% du coût total d'un logiciel
- Très peu enseignée

Pourquoi tester ?

- **Le bug d'Ariane 5 (1996)**
  - 370 millions de dollars
- **Le bug de la sécurité sociale (2009)**
  - entre 900 millions et 2,5 milliards d'euros
- **Mars Climate Orbiter (1999)**
  - 300 millions de dollars.
- Une étude menée en 2013 par l'université de Cambridge évaluait le coût des bugs informatiques à **312 milliards de dollars par an**.

Sommaire

Introduction et rappels

- Pourquoi tester ?
- **Notions de qualité logicielle**
- Rappel sur les cycles de développement logiciel
- Agilité

Les tests

- Introduction
- Classification des tests
- Stratégie de test, plan de test
- Anomalie
- Matrice de couverture
- Intégration continue et Framework de test

Notions de qualité d'un logiciel
Norme ISO 9126

- Comment définir la qualité d'un logiciel ?
  - En se basant sur des indicateurs

- La **norme ISO 9126** (Qualité logicielle) définit 6 groupes de facteurs de qualité des logiciels : **FURPSE**

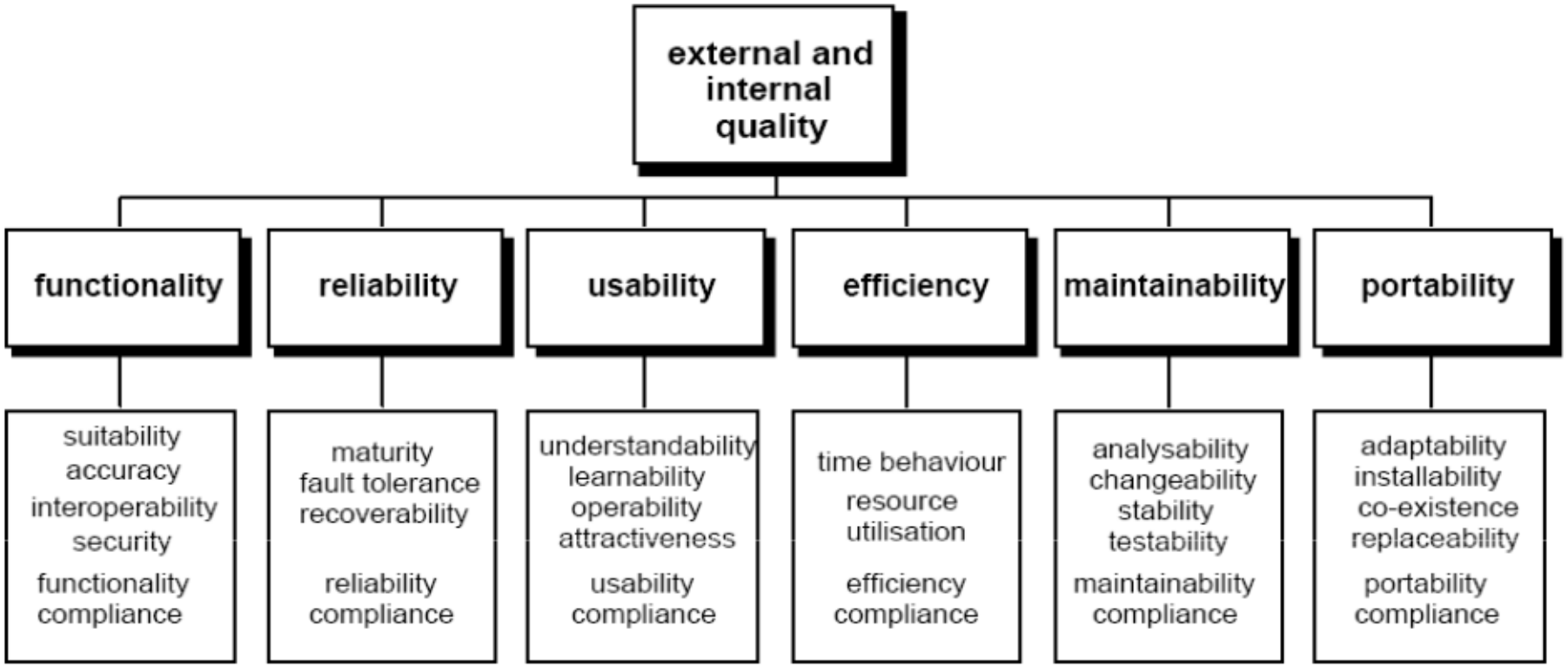
  - F (Functionality) : la capacité fonctionnelle (répondre aux exigences fonctionnelles)
  - U (Usability) : la facilité d'utilisation
  - R (Reliability) : la fiabilité
  - P (Performance, efficiency) : la performance
  - S (System Maintainability) : la maintenabilité
  - E (Portability (Evolutivity)) : la portabilité

Notions de qualité d'un logiciel Norme ISO 9126

- **Facteurs Qualités** : Ils décrivent la vue externe du logiciel, c'est la vue de l'utilisateur.

- **Critères Qualités** : Ils décrivent la vue interne du logiciel, c'est la vue du développeur.

Notions de qualité d'un logiciel
Norme ISO 9126



Les sous-caractéristiques ISO 9126 (1/2)

- F : Capacité fonctionnelle
  - Suitability / adéquation des fonctions
  - Accuracy / précision
  - Interopérability / interopérabilité
  - Security / sécurité
- U : Facilité d'utilisation
  - Understandability / facilité de compréhension
  - Learnability / formation-aides en lignes
  - Operability / facilités de mise en œuvre

Les sous-caractéristiques ISO 9126 (1/2)

- R : Fiabilité
  - Maturity / période de rodage suffisante
  - Fault tolerance / résistance aux pannes
  - Recoverability / reprise ± automatique en cas de panne
- P : Performance
  - Time behaviour / temps de réponse - durée des traitements
  - Resource behaviour / consommation de ressources (mémoire, E/S, …)

Les sous-caractéristiques ISO 9126 (2/2)

- S : Maintenabilité
  - Analyzability / facilités de diagnostiques
  - Changeability / aptitude aux changements et aux modifications
  - Stability / non-regréssion et compatibilité ascendante des interfaces
  - Testability / facilité de mise en œuvre des tests

- E : Portabilité
  - Adaptability / facilités d'adaptation à de nouvelles interfaces
  - Installability / facilité d'installation
  - Conformance / conformité des interfaces (exemple : API)
  - Replaceability / facilités de remplacement des modules (intégrabilité)

Les exigences

- Exigences fonctionnelles
  - règles métiers
- Exigences non fonctionnelles
  - facilité d'utilisation
  - fiabilité
  - performance
  - maintenabilité
  - sécurité informatique
- Contraintes
  - système d'exploitation
  - langage de programmation

How the customer explained it

How the team designed it

What the customer really needed

Sommaire

Introduction et rappels

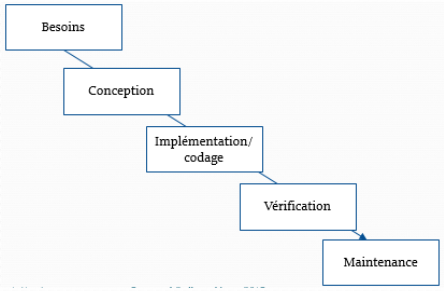
- Pourquoi tester ?
- Notions de qualité logicielle
- **Rappel sur les cycles de développement logiciel**
- Agilité

Les tests

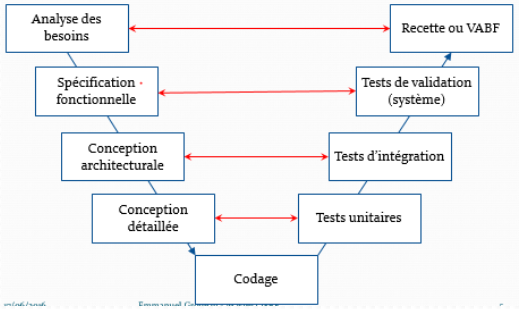
- Introduction
- Classification des tests
- Stratégie de test, plan de test
- Anomalie
- Matrice de couverture
- Intégration continue et Framework de test

Cycles de développement logiciel

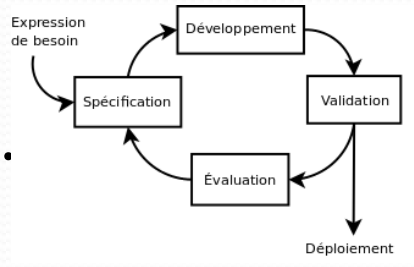
- **Cycle en cascade** …



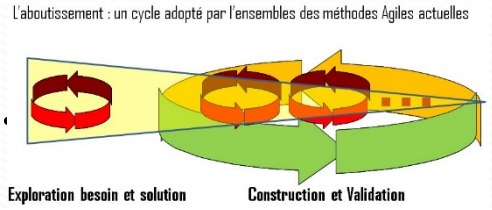
- Cycle en V ……



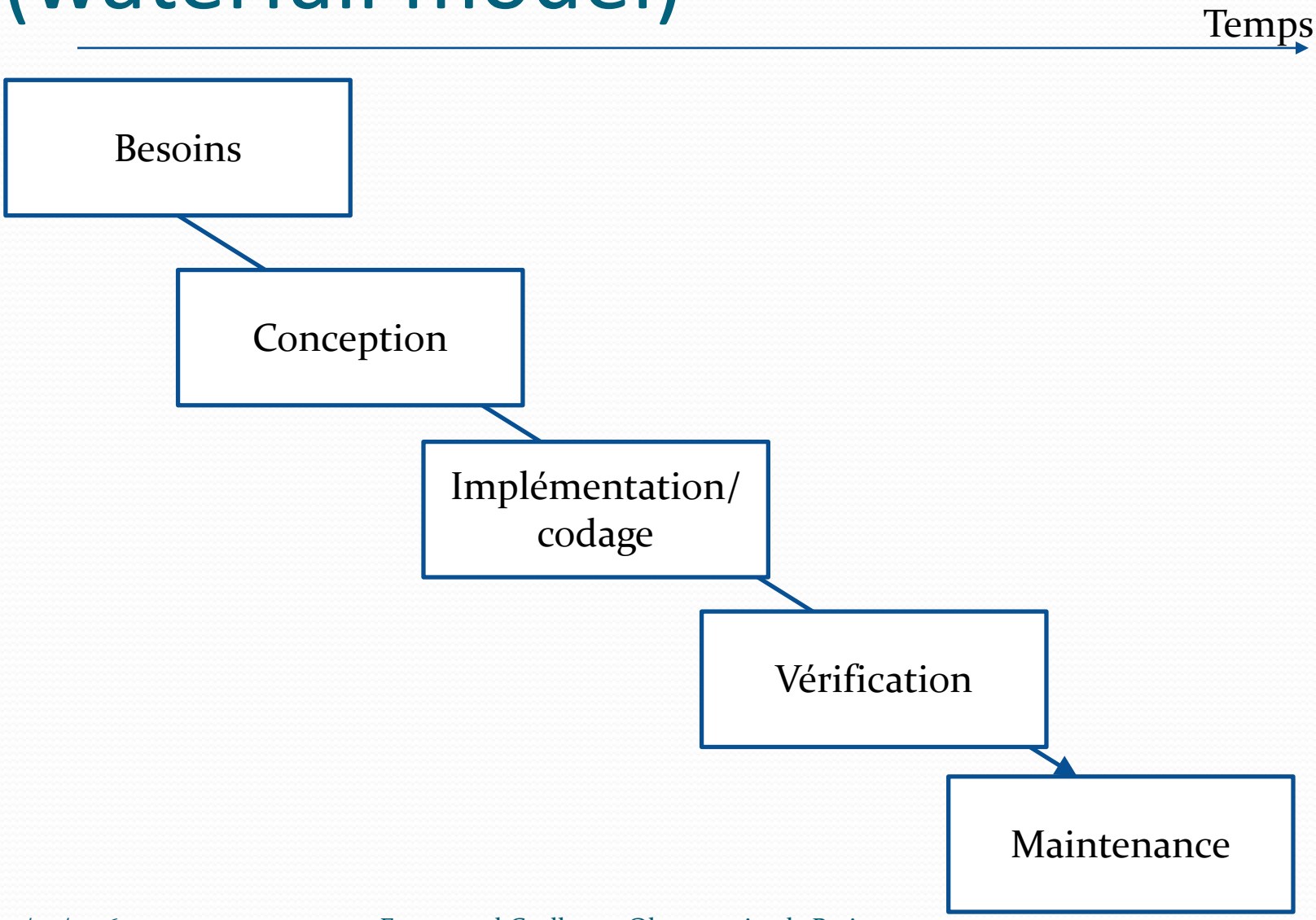
- Cycle itératif …….



- Cycle semi-itératif et méthodes agiles …

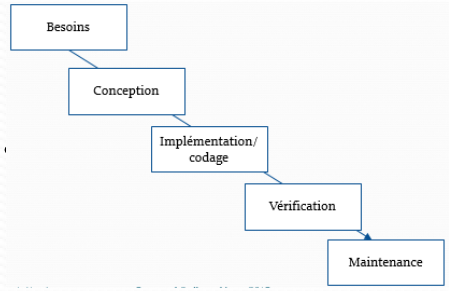


Modèle en cascade (waterfall model)

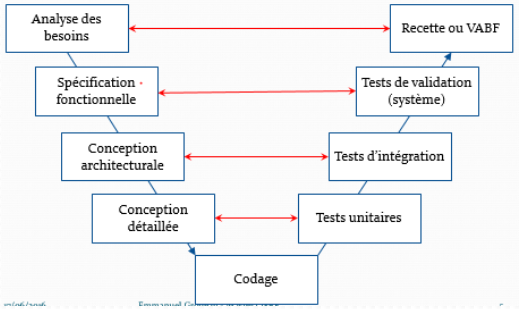


Cycles de développement logiciel

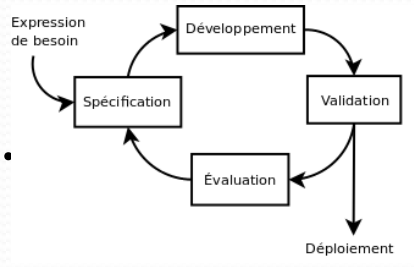
- Cycle en cascade ……



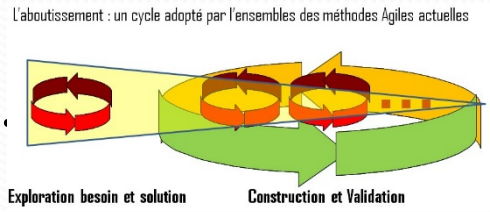
- **Cycle en V** ……



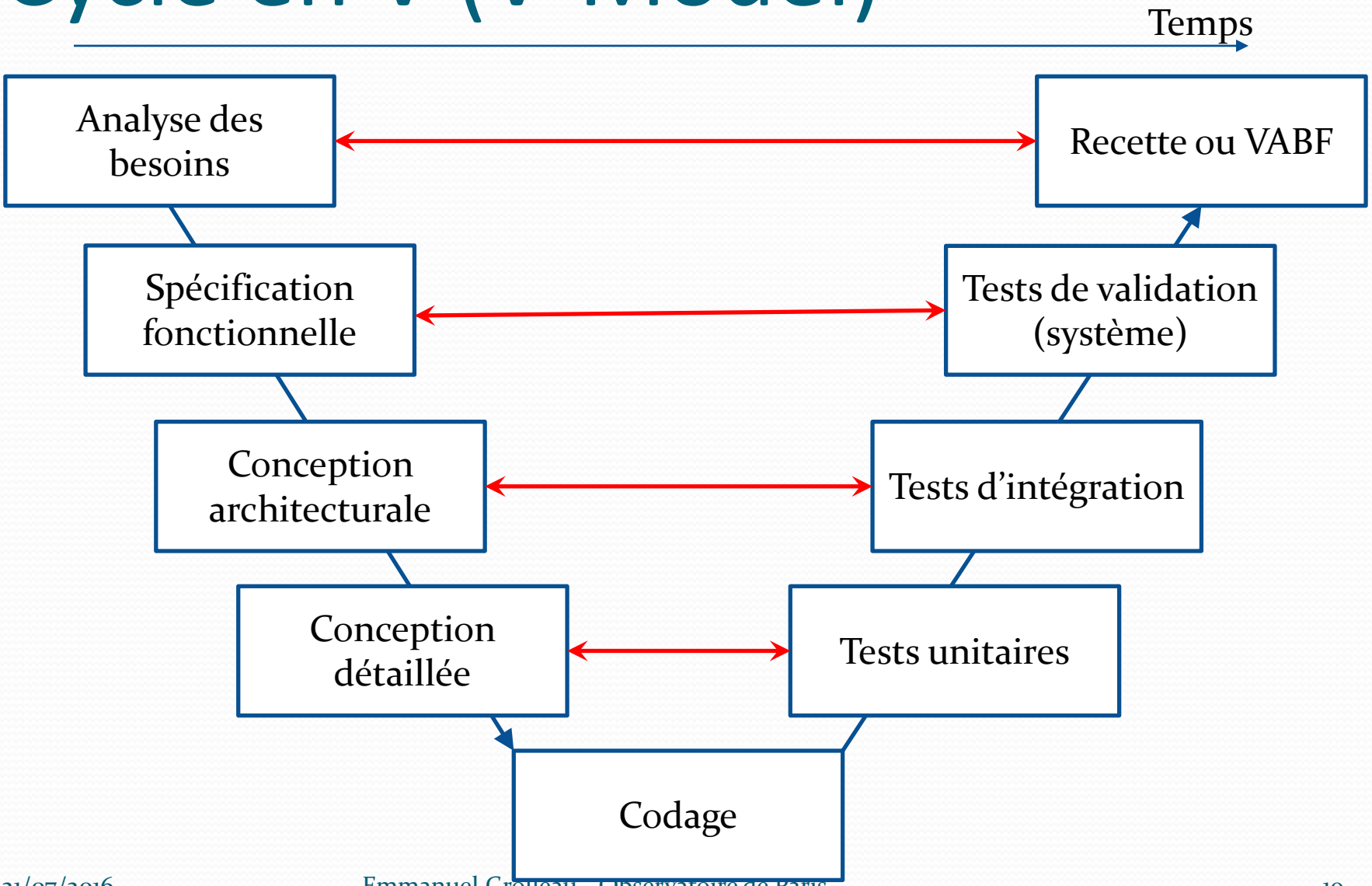
- Cycle itératif …….



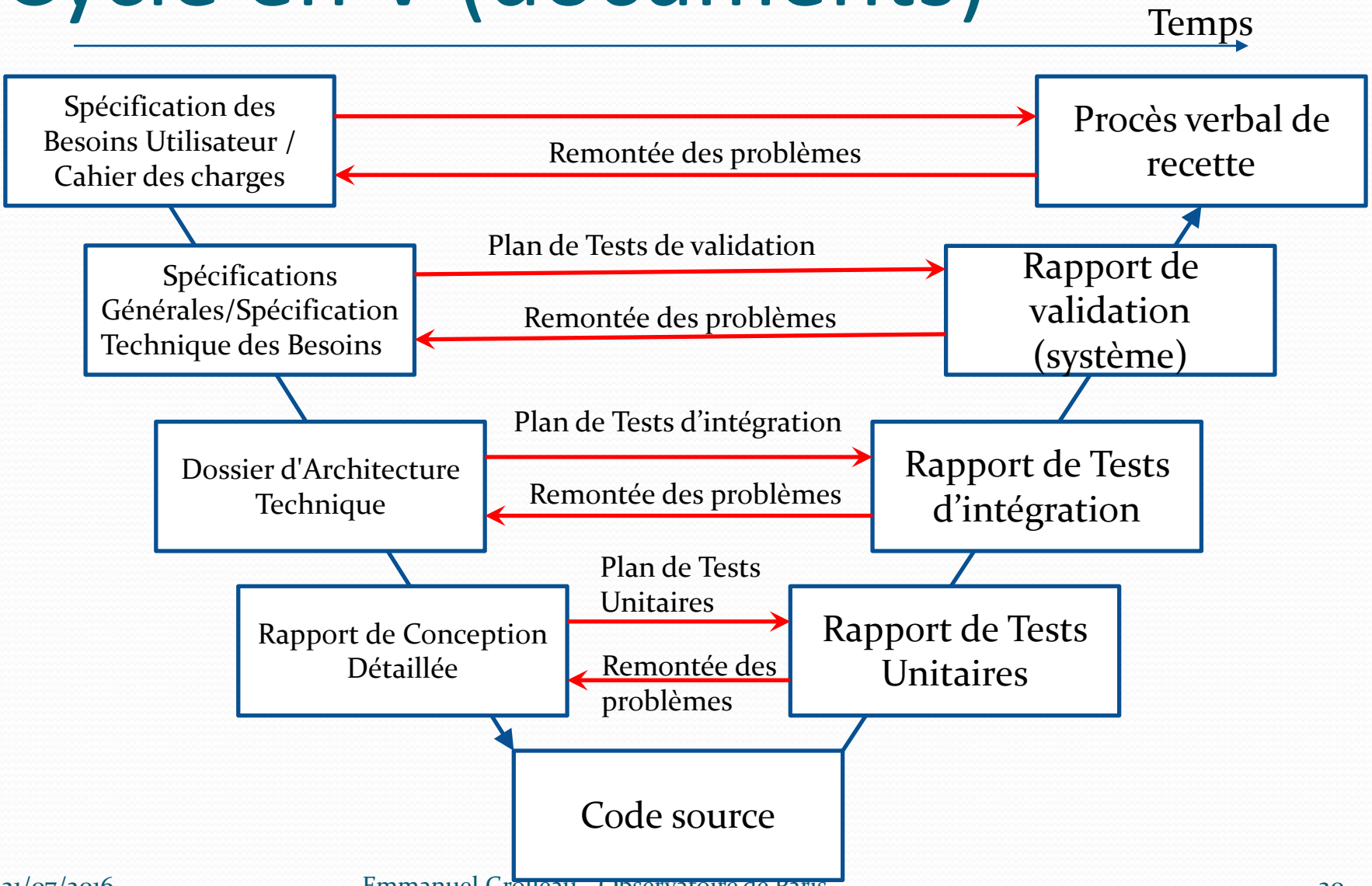
- Cycle semi-itératif et méthodes agiles …



Cycle en V (V-Model)



Cycle en V (documents)

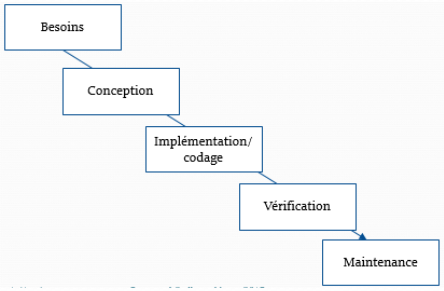


Problème du cycle en V

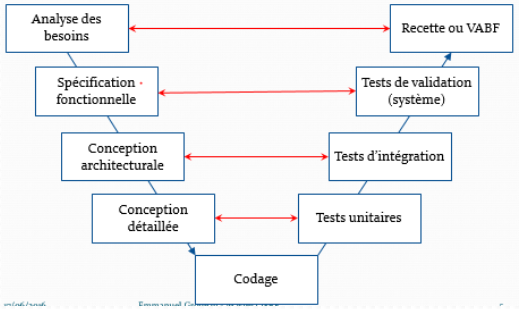
En pratique, il est difficile voire impossible de totalement détacher la phase de conception d'un projet de sa phase de réalisation.

Cycles de développement logiciel

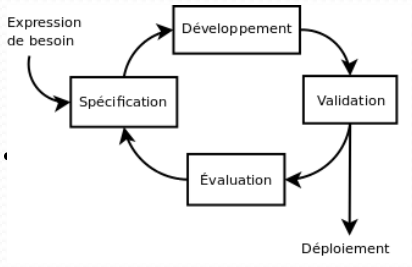
- Cycle en cascade …



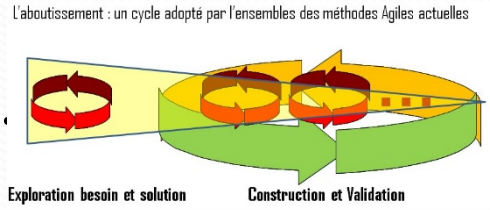
- Cycle en V ……



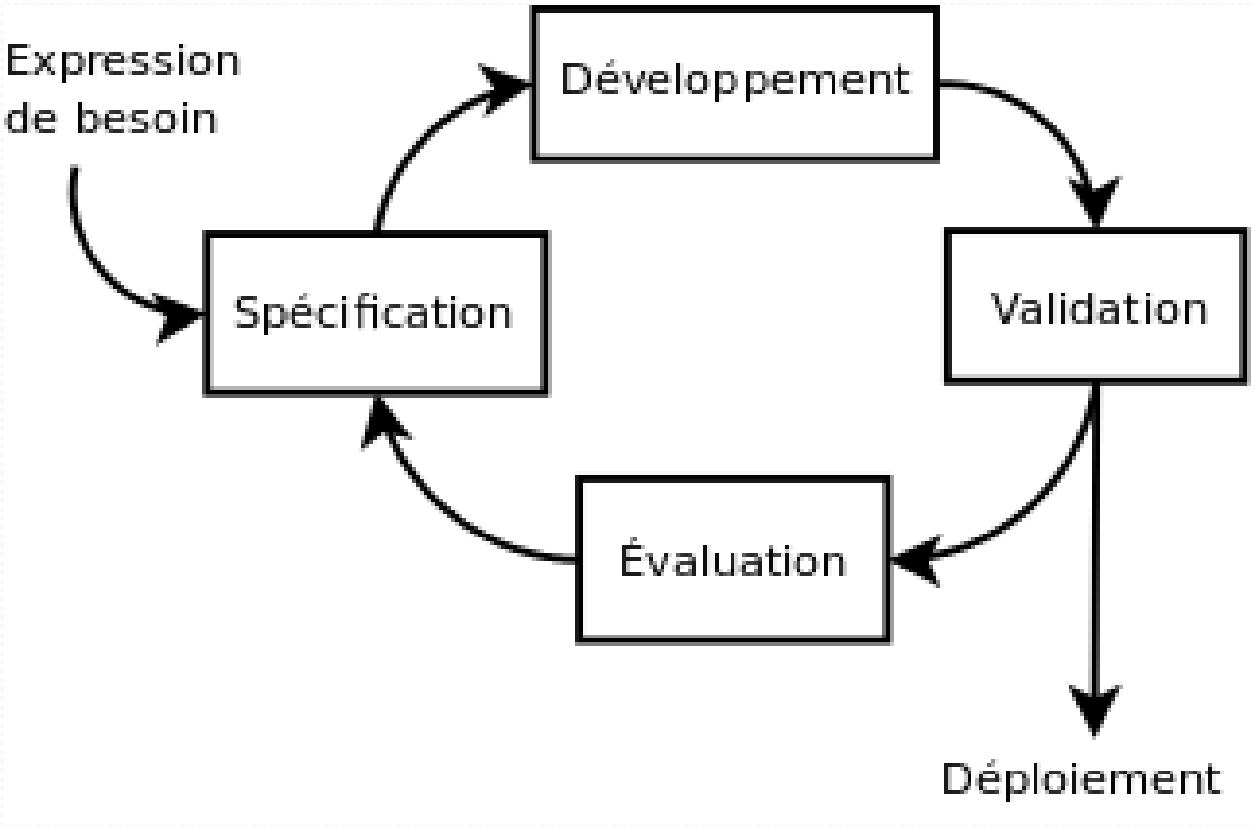
- **Cycle itératif** ……



- Cycle semi-itératif et méthodes agiles …

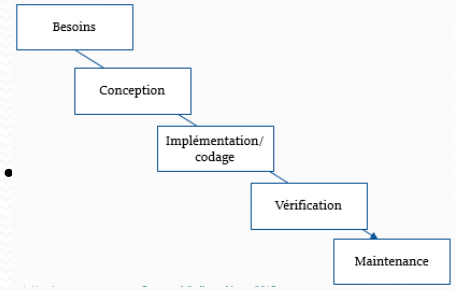


Cycle itératif

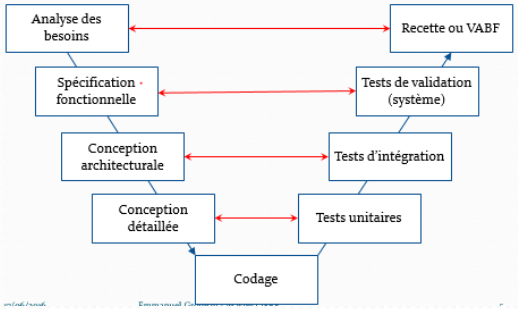


Cycles de développement logiciel

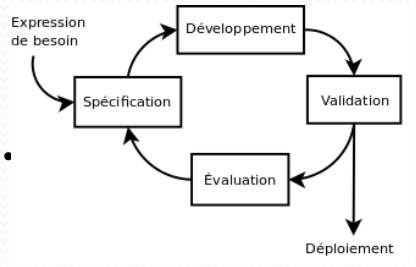
- Cycle en cascade ……



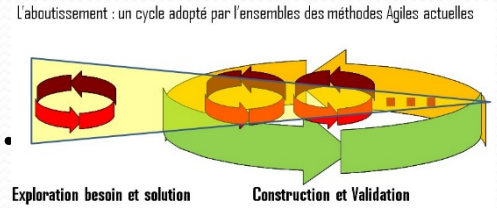
- Cycle en V ……



- Cycle itératif …….

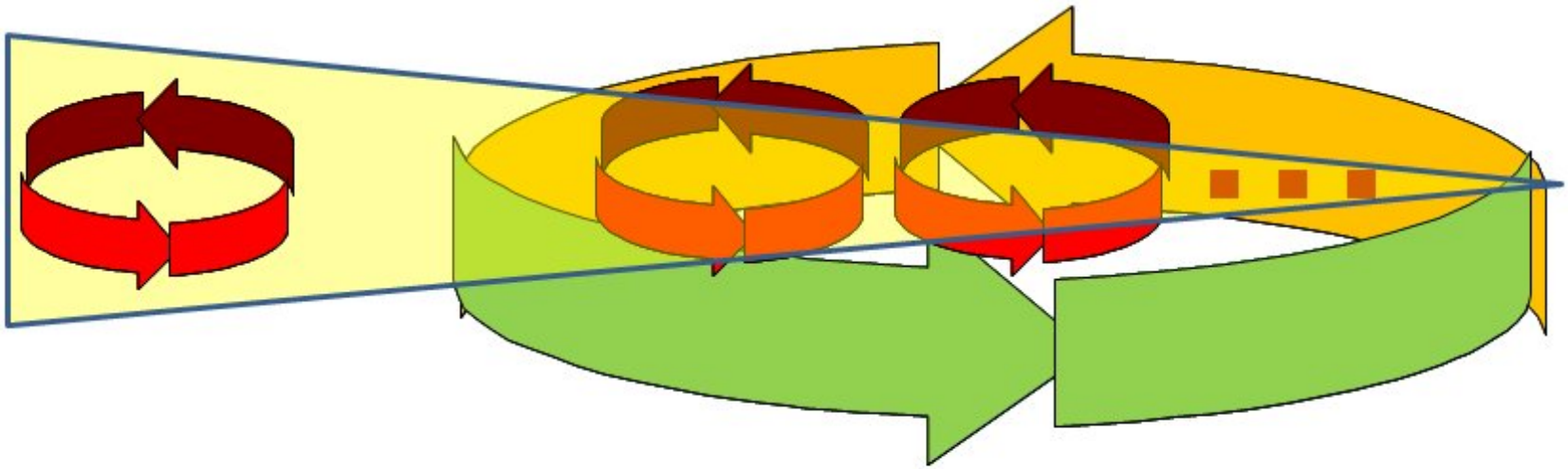


- **Cycle semi-itératif et méthodes agiles** .



Cycle semi-itératif et modèle agile

L'aboutissement : un cycle adopté par l'ensembles des méthodes Agiles actuelles



Sommaire

Introduction et rappels

- Pourquoi tester ?
- Notions de qualité logicielle
- Rappel sur les cycles de développement logiciel
- **Agilité**

Les tests

- Introduction
- Classification des tests
- Stratégie de test, plan de test
- Anomalie
- Matrice de couverture
- Intégration continue et Framework de test

Le manifeste « Agile »

- **4 valeurs** et **12 principes fondateurs**.

Nous découvrons de meilleures approches pour faire du développement logiciel, en en faisant nous-même et en aidant les autres à en faire. Grâce à ce travail nous en sommes arrivés à préférer et ***favoriser****:*

- **Les individus et leurs interactions** plus que les processus et les outils.
- **Du logiciel qui fonctionne** plus qu'une documentation exhaustive.
- **La collaboration avec les clients** plus que la négociation contractuelle.
- **L'adaptation au changement** plus que le suivi d'un plan.

Cela signifie que bien qu'il y ait de la valeur dans les items situés à droite, notre préférence se porte sur les items qui se trouvent sur la gauche.

Le manifeste Agile (12 principes)

- Notre plus haute priorité est de **satisfaire le client** en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.
- Accueillez **positivement les changements** de besoins, même tard dans le projet. Les processus agiles exploitent le changement pour donner un avantage compétitif au client.
- **Livrez fréquemment un logiciel opérationnel** avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts.
- **Les utilisateurs et les développeurs doivent travailler ensemble** quotidiennement tout au long du projet.

Le manifeste Agile (12 principes)

- **Réalisez les projets avec des personnes motivées**. Fournissez-leur l'environnement et le soutien dont elles ont besoin et faites-leur confiance pour atteindre les objectifs fixés.
- La méthode la plus simple et la plus efficace pour transmettre de l'information à l'équipe de développement et à l'intérieur de celle-ci est **le dialogue en face à face**.
- Un **logiciel opérationnel** est la principale mesure d'avancement.
- Les processus agiles encouragent un rythme de développement soutenable. Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant.

Le manifeste Agile (12 principes)

- Une attention continue conduit à **l'excellence technique** et à une bonne conception renforce l'agilité.
- La **simplicité** – c'est-à-dire l'art de minimiser la quantité de travail inutile – est essentielle.
- Les meilleures architectures, spécifications et conceptions émergent d'équipes **auto-organisées.**
- À intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis règle et modifie son comportement en conséquence.

Sommaire

Introduction et rappels

- Pourquoi tester ?
- Notions de qualité logicielle
- Rappel sur les cycles de développement logiciel
- Agilité

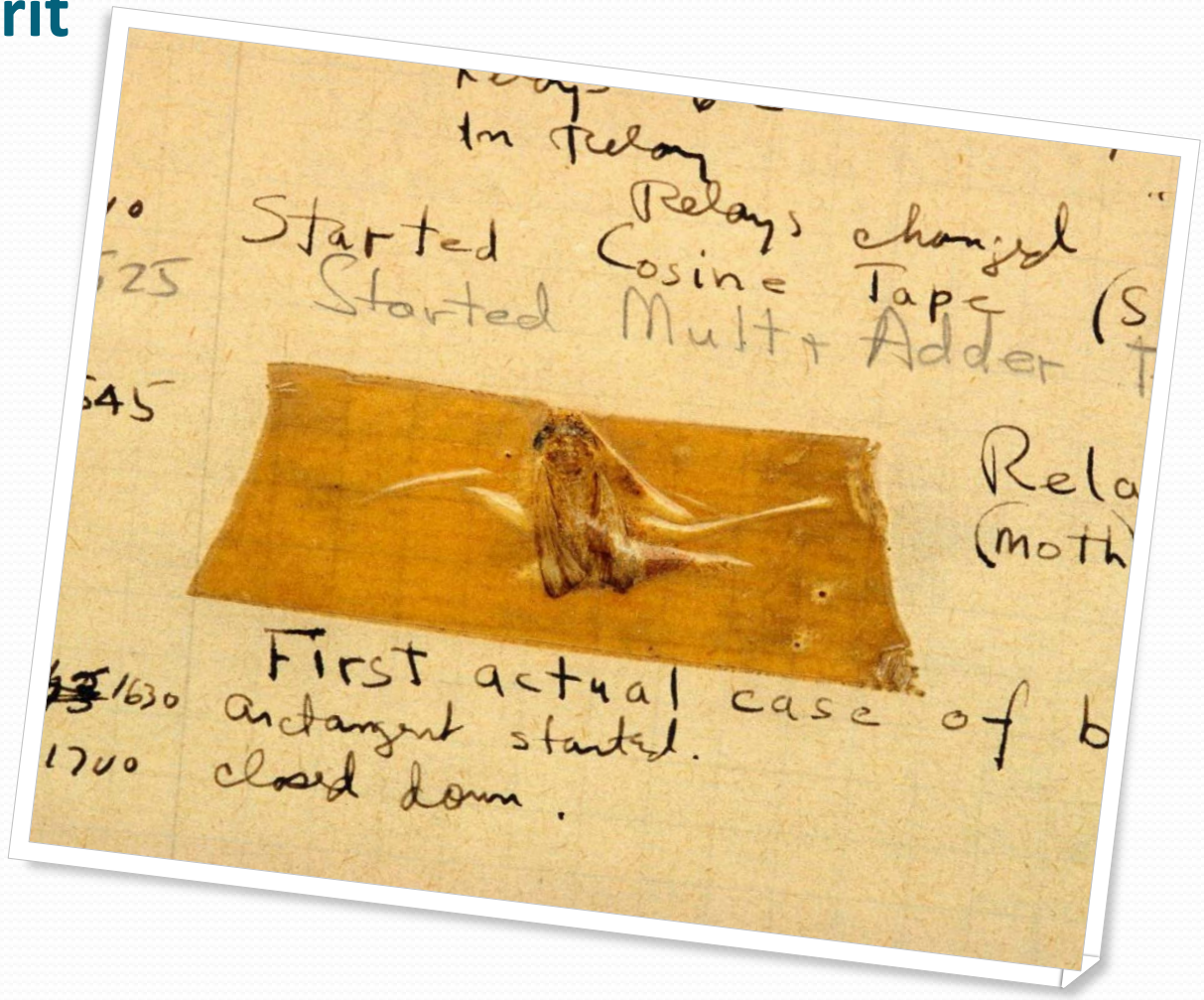
Les tests

- Introduction
- Classification des tests
- Stratégie de test, plan de test
- Anomalie
- Matrice de couverture
- Intégration continue et Framework de test

Les tests

Premier bug décrit 1947

Grace Hopper's operational logbook for the Harvard Mark II computer



Grace Murray Hopper, brillante mathématicienne et pionnière de la programmation

Chaîne de l'erreur

Défaut : défaut introduit dans le logiciel lors d'une des phases du cycle de développement

Anomalie (ou défaillance) : comportement observé différent du comportement attendu ou spécifié. C'est la manifestation concrète du défaut. Elle peut ne jamais avoir lieu.

Bogue (bug) : Aujourd'hui le terme « bogue » est utilisé aussi bien pour désigner un défaut qu'une anomalie.

Erreur Humaine → Défaut Dans le logiciel → Anomalie

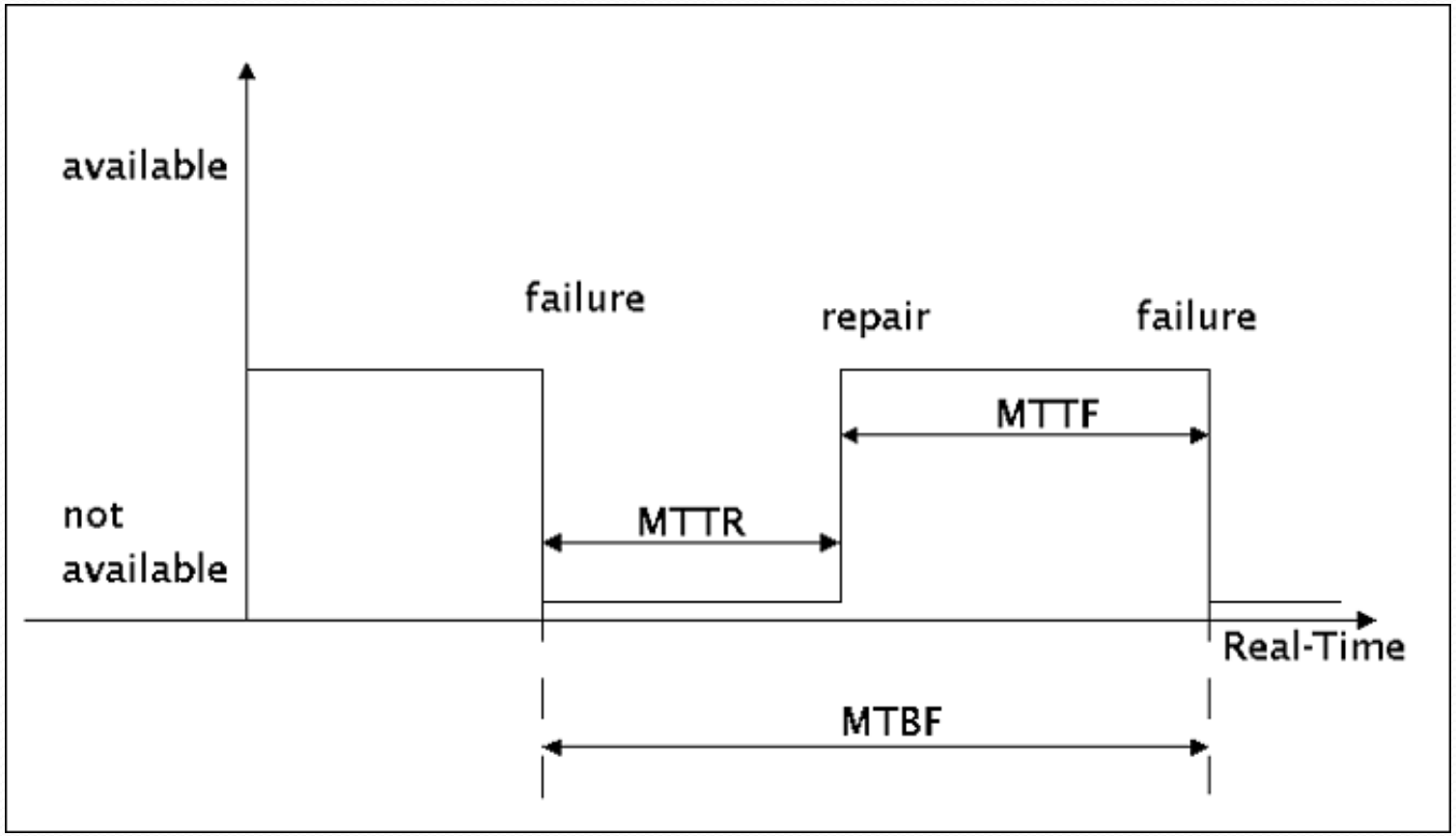
Statistiques

On estime qu'un développeur de bon niveau introduit en moyenne :

50

défauts par 1000 lignes de code.

Mesures de disponibilité



$$Disponibilité = \frac{MTBF}{MTBF + MTTR}$$

Les tests

- Dans toutes les cycles de développement vus précédemment les phases de test (ou de validation) sont primordiales.

- Dans les méthodes agiles, le but est de fournir le plus rapidement possible du **logiciel qui fonctionne,** c'est-à-dire que le logiciel est testé **presqu'en continu!**

Définition

- Qu'est ce qu'un **test logiciel** ?

Processus d'analyse d'un programme avec l'intention de détecter des anomalies dans le but de le valider.

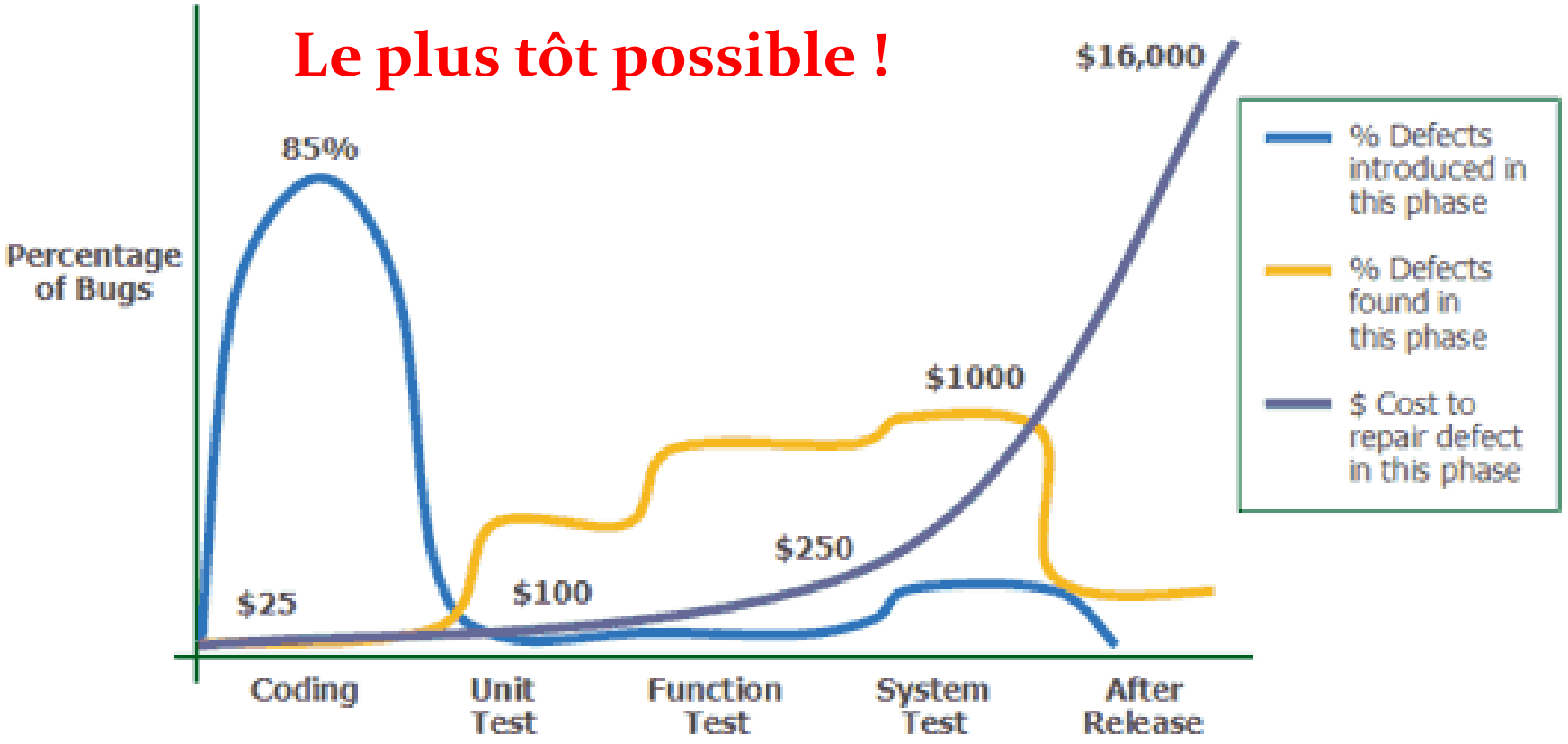
- Anomalies par rapport à quoi ?
- **Par rapport aux spécifications**

- D'où la nécessité d'avoir une spécification qui comporte des informations complètes et correctes pour :
  - le programmeur = programmer sans erreurs par rapport à la spécification
  - le testeur = trouver le plus d'erreurs par rapport à la spécification

Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence.

Edsger W. Dijkstra , The Humble Programmer (1972), Communications of the ACM 15 (10), octobre 1972

Quand tester ?



Source: Applied Software Measurement, Capers Jones, 1996

Règle 40 - 20 - 40

- 40% de l'effort est requis pour spécifier et concevoir,
- 20% de l'effort va à la programmation,
- **40% de l'effort va aux tests de vérification et de validation.**

Se focaliser uniquement sur la programmation, c'est se préparer à une catastrophe qui ne manquera certainement pas d'arriver.

En fin de programmation on est généralement à ≈50% du temps de la fin effective du projet (moins vrai en programmation agile).

Idées fausses sur les tests

- Il faut éliminer tous les défauts
  - Vrai et Faux, il faut éliminer tous les défauts par rapport aux exigences fonctionnelles et non fonctionnelles.
- Je programme sans erreur, ce n'est pas la peine de tester …
- L'amélioration de la fiabilité est proportionnelle au taux de couverture du code par les tests
  - Faux, la garantie de fiabilité nécessite des tests de fiabilité spécifiques (interruption, reprise sur erreur).
- Croire que c'est simple et facile
- Croire que cela n'exige ni expérience, ni savoir-faire, ni méthodes

Testabilité d'un logiciel

- **Testabilité**: Facilité avec laquelle les tests peuvent être développés à partir des documents de conception

- **Facteurs de bonne testabilité**:
  - Précision, traçabilité des documents
  - Architecture simple et modulaire
  - Politique de traitements des erreurs clairement définie

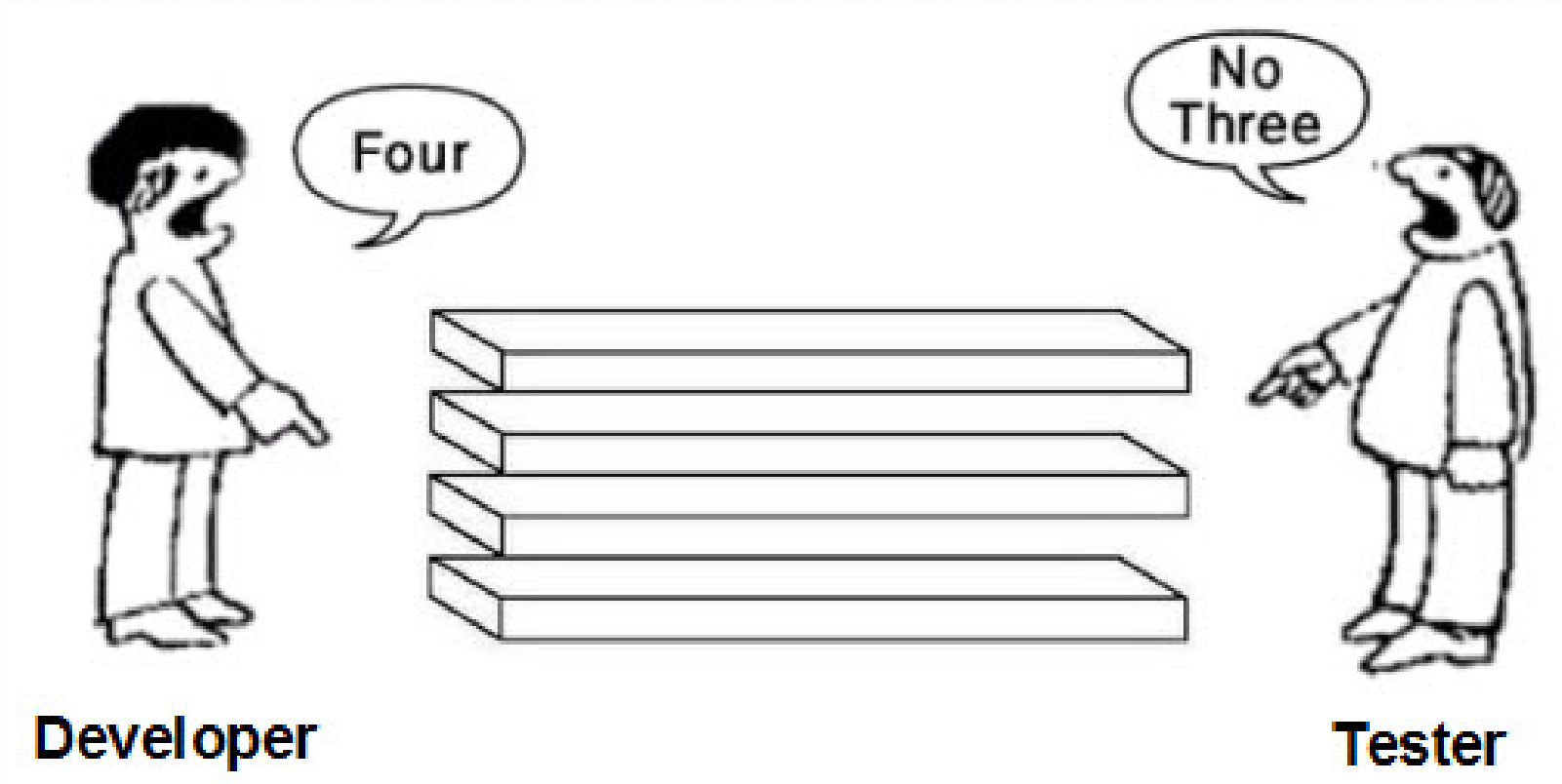
- **Facteurs de mauvaise testabilité:**
  - Fortes contraintes d'efficacité (espace mémoire, temps)
  - Architecture mal définie

Vérification et Validation (V&V)

- Vérification
  - *Le système fait-il correctement son travail ?*
- Validation
  - *Le système réalisé correspond-il aux besoins du client?*

6 grands principes

- P1: **Indépendance**
  - Un programmeur ne doit pas tester ses propres programmes.
- P2: **Paranoïa**
  - Ne pas faire de tests avec l'hypothèse qu'il n'y a pas d'erreur.
- P3: **Prédiction**
  - La définition des sorties/résultats attendus doit être effectuée avant l'exécution des tests.
  - La définition des sorties/résultats attendus est un produit de la spécification (plan de test)


Four
No
Three
Developer
Tester

6 grands principes

- P4: **Vérification**
  - Il faut inspecter minutieusement les résultats de chaque test.
  - C'est la séparation de l'exécution et de l'analyse.
- P5: **Robustesse**
  - Les jeux de tests doivent être écrits pour des entrées invalides ou incohérentes aussi bien que pour des entrées valides.
- P6: **Complétude**
  - Vérifier un logiciel pour vérifier 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

Sommaire

Introduction et rappels

- Pourquoi tester ?
- Notions de qualité logicielle
- Rappel sur les cycles de développement logiciel
- Agilité

Les tests

- Introduction
- **Classification des tests**
- Stratégie de test, plan de test
- Anomalie
- Matrice de couverture
- Intégration continue et Framework de test

Classification des tests

- Y-a-t-il plusieurs types de test ?

Classification des tests

Approche : Statique / Dynamique

- Tests **statiques**
  - Test «par l'humain», sans machine.
    - Une personne lit le code (inspection ou revue de code)
  - Réunions, qui ?

- Tests **dynamiques**
  - Test « par la machine », exécution de l'application ou d'une partie
  - un test réussi si les résultats obtenus sont les résultats attendus, sinon il échoue
  - Notion de **Fiabilité** du critère de test.

Classification des tests

Approche : Fonctionnel / Structurel

- **Tests fonctionnels**
  - Vérifier que le logiciel respecte sa spécification.
- **Tests structurels**
  - Détecter les fautes d'implémentation.
  - Vérifier qu'il n'existe pas de cas de plantage (overflow, non initialisation, …)

Classification des tests

Approche : Fonctionnel / Structurel

- **Tests fonctionnels** (**tests en boîte noire**)
  - reposent sur une spécification du programme
  - le code source du programme n'est pas utilisé
  - aucune connaissance de l'implantation n'est nécessaire
  - Possibilité d'écrire les tests avant le code

- **Tests structurels** (**tests en boîte blanche**)
  - reposent sur des analyses du code source
  - étude de la structure du programme (flot de contrôle ou de données) pour en déterminer les tests. On fixe les valeurs des données d'entrée pour parcourir les diverses branches du programme.
  - Impossibilité d'écrire les tests avant le code

Classification des tests

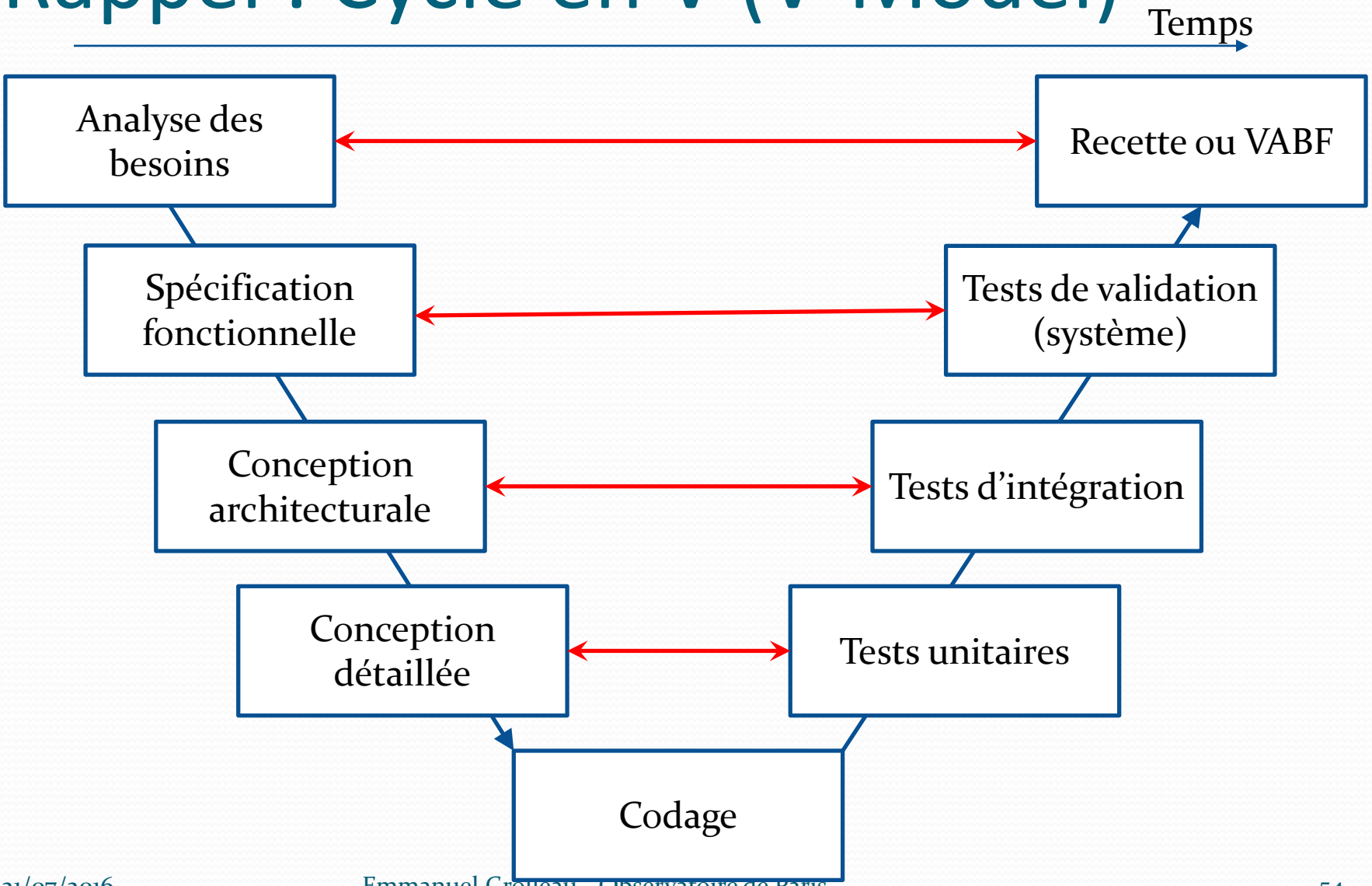
Approche : Manuel / Automatique

- Tests **manuels**
  - l'opérateur saisit les données d'entrée du test manuellement (via interface) et lance le test
  - l'opérateur observe les résultats et les compare avec les résultats attendus
  - fastidieux, possibilité d'erreur humaine
  - ingérable pour les grosses applications

- Tests **automatisés**
  - Outils de tests qui permettent automatiquement :
    - le lancement des tests
    - l'enregistrement des résultats
    - parfois de la génération des données d'entrée et des résultats attendus

Classification des tests

Rappel : Cycle en V (V-Model)



Classification des tests

Approche : niveau du cycle de développement

Le Comité français du Test Logiciel (CFTL) identifie quatre **niveaux** de test :

- **Unitaire** (ou composant)
  - test des (petites) parties du code, séparément.
- **Intégration**
  - Test d'un ensemble de parties du code qui coopèrent. Test des modules entre eux.
- **Système** (test fonctionnel, ou test de validation)
  - Test du système entier, en inspectant sa fonctionnalité.
- **Test d'acceptation** ou UAT (*user acceptance testing, ou recette utilisateur, ou VABF)*

Micro projet

Spécification fonctionnelle

- Un client souhaiterait posséder une application modulaire permettant le calcul d'une moyenne et/ou d'une médiane et/ou d'un écart type à partir d'une liste de valeurs contenues dans un fichier.

- Il souhaiterait que les résultats obtenus (moyenne, écart type, médiane) soient écrits dans un fichier résultat.

Micro projet

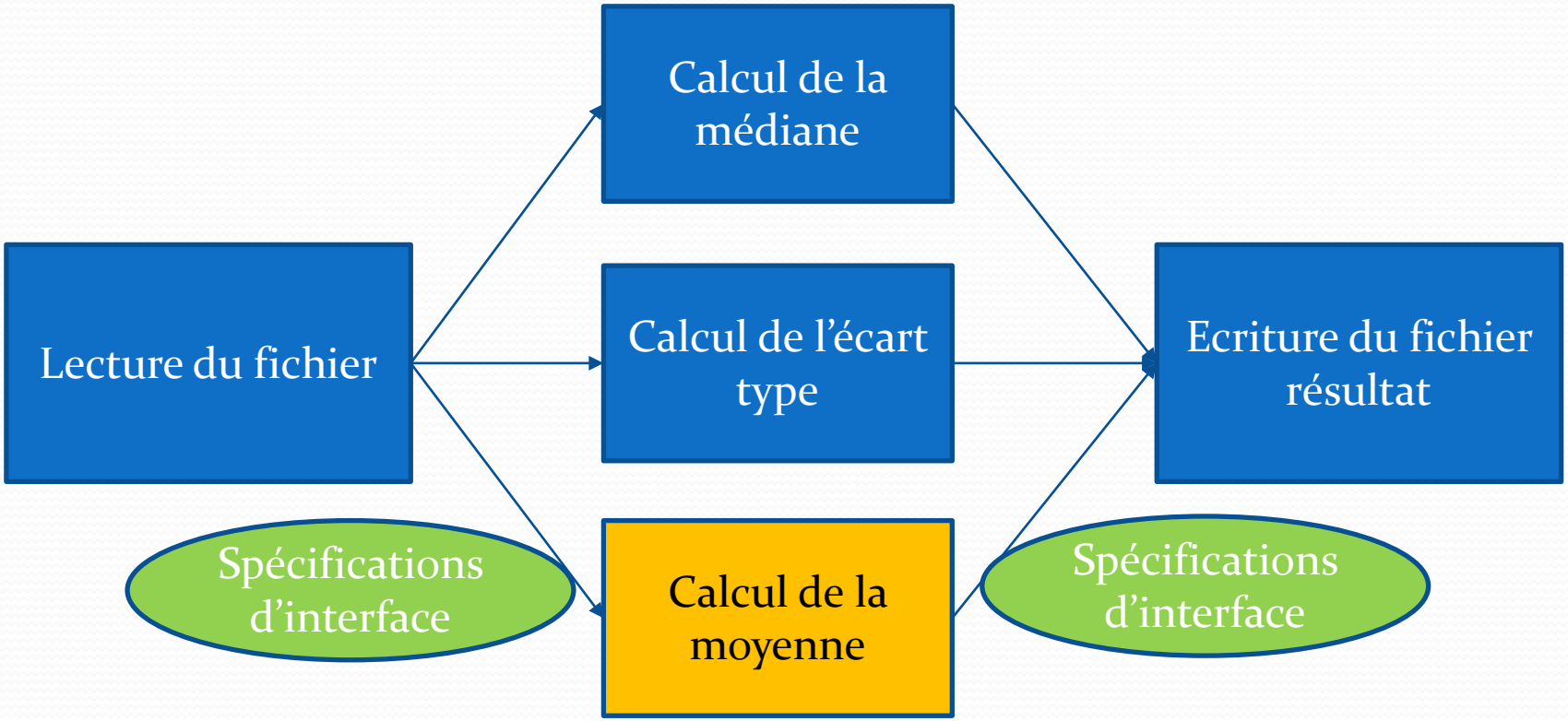
- Quels **tests systèmes** (ou validation) ?
  - A partir de plusieurs jeux de test en entrée, exécution de l'application pour :
  - Calcul d'une moyenne
  - Calcul d'un écart type
  - Calcul d'une médiane
  - Calcul d'une moyenne et d'un écart type
  - ….

Micro projet

Module existant (à intégrer)

Module à créer

Conception architecturale

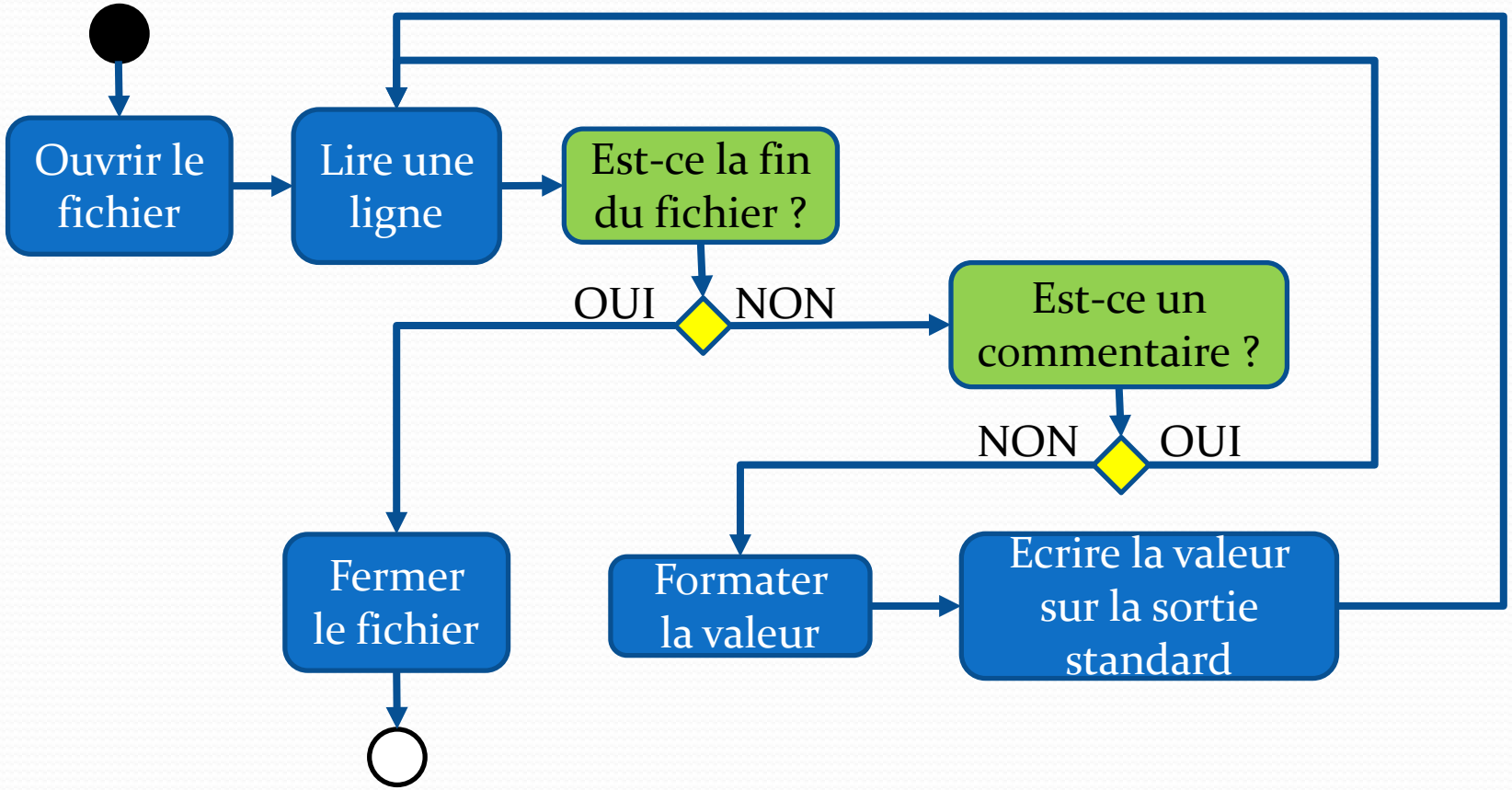


Micro projet

- Quels **tests d'intégration** ?
  - Enchainement du module de lecture et du module de calcul d'une moyenne
  - Enchainement du module de lecture et du module de calcul d'une moyenne et du module de calcul d'une médiane
  - Ecriture d'une suite de chiffres via la commande « cat fichier» sur l'entrée standard et exécution du module de médiane et d'écriture des résultats
  - ….

Micro projet

<u>Conception détaillée,</u> Module Lecture du fichier



Micro projet

- Quels **tests unitaires** ?
  - Ecriture d'une suite de chiffres via la commande « cat fichier» et exécution du module « calcul d'un écart type », lecture des résultats sur la sortie standard
  - Exécution du module « lecture de fichier », lecture des résultats sur la sortie standard
  - …

Micro projet

<u>Codage,</u> Module Lecture du fichier

Idem pour les autres modules

OUVRIR le fichier
TANT QUE non fin de fichier
    LIRE ligne
    SI ligne est un commentaire, CONTINUE
    FORMATER la valeur
    ECRIRE la valeur sur la sortie standard
FERMER le fichier

Classification des tests

Rappel : Qualité d'un logiciel

- La **norme ISO 9126** (Qualité logicielle) définit 6 groupes d'indicateurs de qualité des logiciels : **FURPSE**
  - F (Functionality) : la capacité fonctionnelle (répondre aux exigences)
  - U (Usability) : la facilité d'utilisation
  - R (Reliability) : la fiabilité
  - P (Performance) : la performance
  - S (System maintenability) : la maintenabilité
  - E (Evolutivity) : la portabilité

Classification des tests

Approche : facteur qualité testé

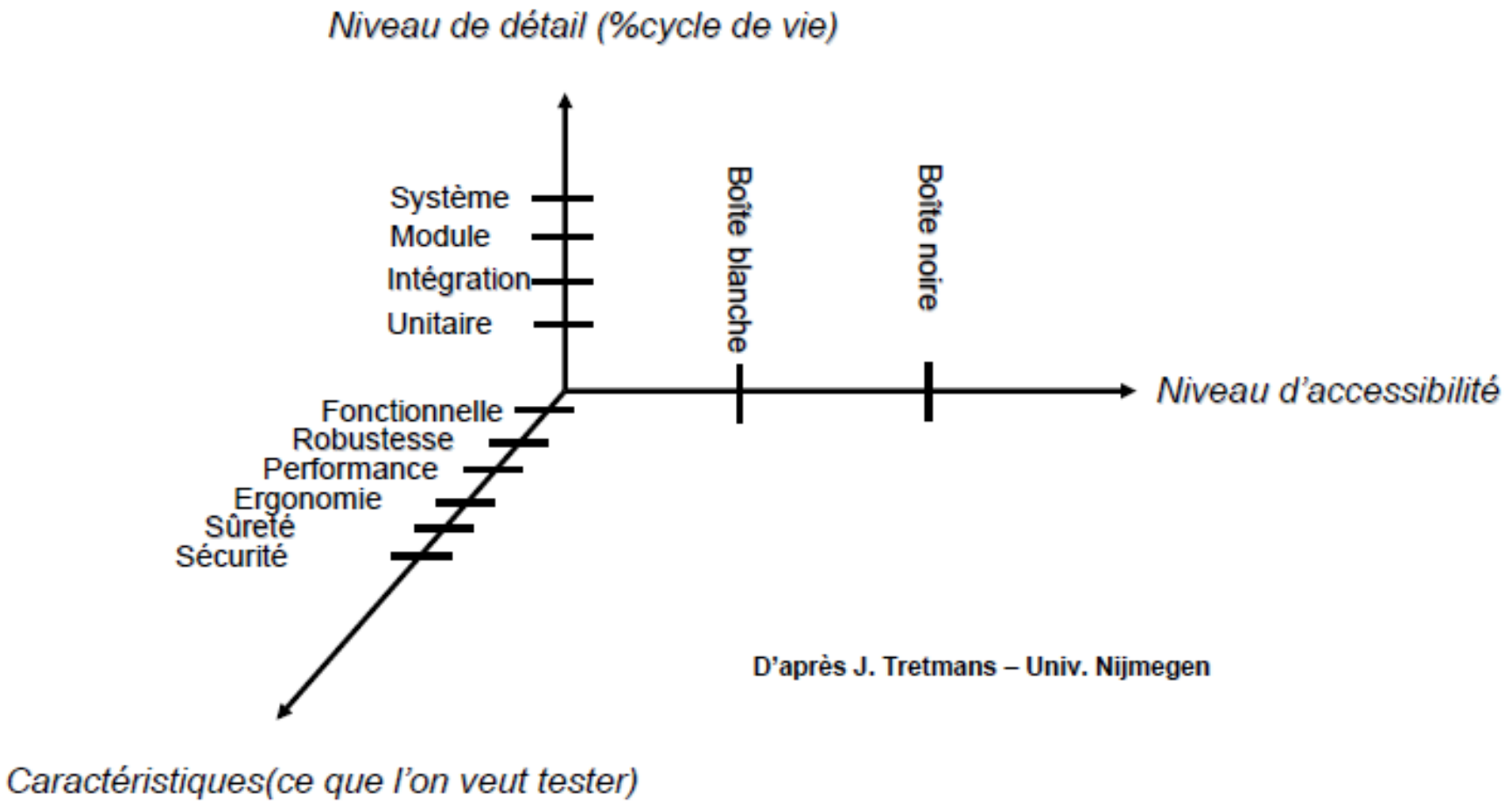
- F : capacité fonctionnelle
  - **test fonctionnel** : vérifier que les fonctionnalités demandées sont bien supportées.
  - **test de vulnérabilité** : vérification de sécurité du logiciel.
- U: facilité d'utilisation
  - **test utilisateur :** faire tester le logiciel par un utilisateur en décrivant un cas d'utilisation
- R : fiabilité
  - **test de robustesse** : valider la stabilité et la fiabilité du logiciel

Classification des tests

Approche : facteur qualité testé

- P : performance
  - **test de performance** : valider que les performances annoncées dans la spécification sont bien atteintes. (test de charge, ..)
- S : maintenabilité
  - **test de non régression**
- E : portabilité
  - **test sur différents O.S.**

Classification des tests



Tests de robustesse

- Valider la stabilité et la **fiabilité** du logiciel
- **Fiabilité** : capacité d'un logiciel à rendre des résultats corrects quelles que soient les conditions d'exploitation (résistance aux pannes).
- Créer des jeux de test avec des valeurs d'entrée :
  - aux limites de la plage de définition des valeurs de la spécification;
  - hors des limites de la plage de définition;
- Interrompre le logiciel pendant son fonctionnement

Micro projet

- Test de robustesse
  - Créer des jeux de test (fichier d'entrée) avec des valeurs négatives
  - Créer des jeux de test avec des valeurs très grandes, positives ou négatives
  - Créer des jeux de test vide !
  - Créer des jeux de test avec N milliard de lignes

Réaliser ce type de test en « test unitaire », « test d'intégration », « test de validation »

Tests de performance

Dans un **cas nominal**

- Test des temps de réponse - durée des traitements – par rapport à la spécification

- Test de la consommation de ressources (CPU, mémoire, E/S, …) par rapport à la spécification

Mêmes tests dans un cas de **montée en charge**

- Test de charge : montée en charge (par exemple hausse du nombre d'utilisateurs) du logiciel

Micro projet

- Test de performance
  - Créer des jeux de test avec N * millions de lignes, N variant de 1 à 10. Etablir la courbe d'évolution du temps de réponse en fonction de N.

Réaliser ce type de test en « test unitaire », « test d'intégration », « test de validation »

Tests de non régression

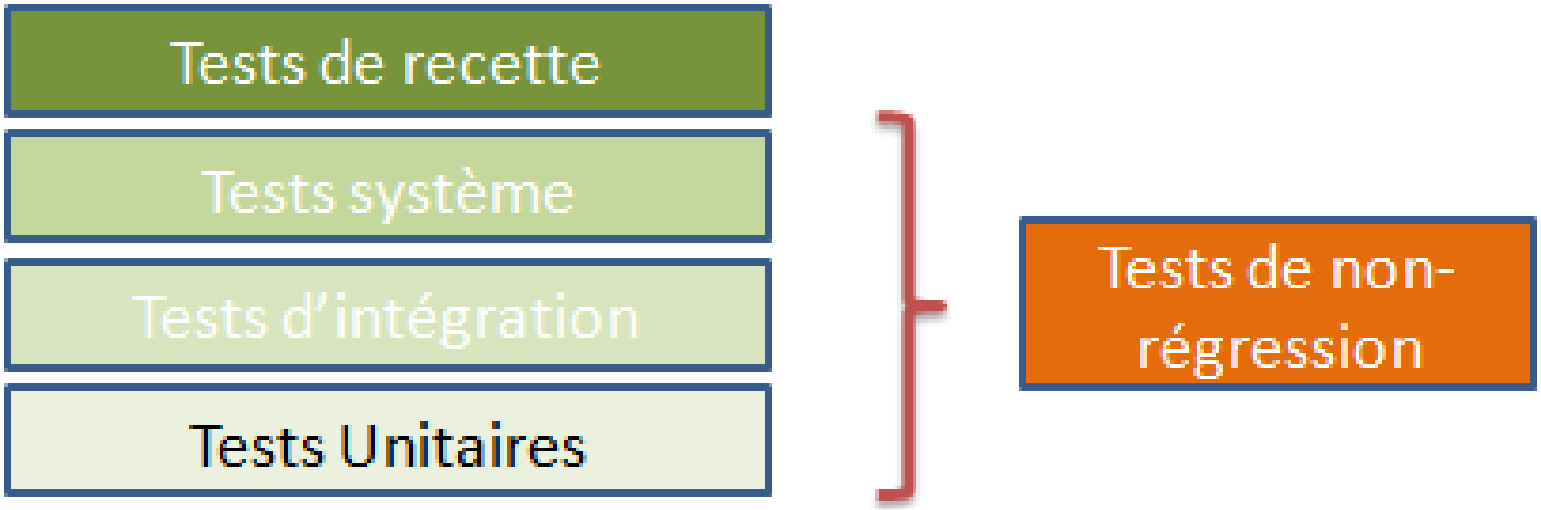
- Des études ont montré que la probabilité qu'un programme corrigé fonctionne comme avant la correction est de :

  50 %

- Tests de non régression : tests d'un programme **préalablement testé**, après une modification, pour s'assurer que des défauts n'ont pas été introduits ou découverts dans des parties non modifiées du logiciel.

- Intérêt de la mise en œuvre d'un framework de test automatique (intégration continue).

Tests de non régression



Sommaire

Introduction et rappels

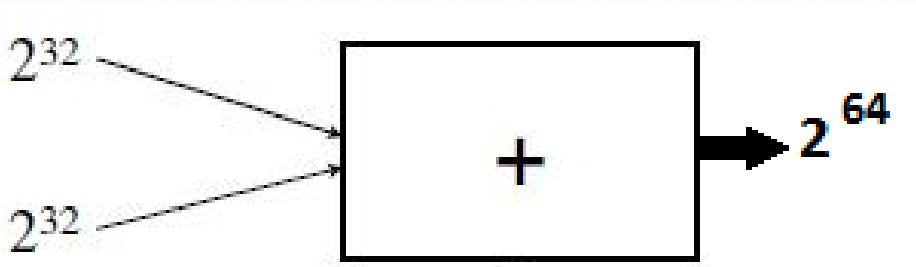
- Pourquoi tester ?
- Notions de qualité logicielle
- Rappel sur les cycles de développement logiciel
- Agilité

Les tests

- Introduction
- Classification des tests
- **Stratégie de test, plan de test**
- Anomalie
- Matrice de couverture
- Intégration continue et Framework de test

Stratégie de test

- Un programme a un nombre infini (ou extrêmement grand !) d'exécutions possibles
  - Explosion combinatoire



- Les tests n'examinent qu'un nombre fini (ou très petit) d'exécutions

- Il faut **définir une stratégie de test** la plus pertinence possible

Stratégie de test

- La stratégie de test va définir :
  - Les ressources mises en œuvre
  - Le **processus** de mise en œuvre des tests

- La stratégie de test est contrainte par :
  - La méthode, le **cycle de développement** utilisé
  - Les **langages** de programmation utilisés
  - Le **type d'application** développée
  - …
- L'aboutissement tangible de la stratégie de test sera le « **Plan de test** »

Stratégie de test
Processus de test

Processus de test

Processus consistant en ***toutes les activités*** *du cycle de vie,* ***statiques et dynamiques****, concernant la* ***planification*** *et* ***l'évaluation*** *de produits logiciels, pour déterminer s'ils* ***satisfont aux exigences****, pour démontrer qu'ils sont* ***aptes aux objectifs*** *et pour en* ***détecter des anomalies****.*

Définition du Comité français du Test Logiciel (CFTL)

Stratégie de test
Processus de test

Stratégie de test
Processus de test

- Il s'agit d'un véritable processus qui va pouvoir s'appuyer sur des ressources :
  - Des moyens humains : des opérateurs de test, un responsable des tests, un responsable qualité, …
  - Des outils : outil de gestion de version des tests, archivage des jeux de données, archivage des résultats, framework d'éxécution, …

Stratégie de test
Définitions

Test : ensemble d'un ou plusieurs **cas de tests**

Cas de test : un ensemble de **valeurs d'entrée** (données de tests), de **préconditions** d'exécution, de **résultats attendus** et de **post conditions** d'exécution, développées pour un **objectif** ou une condition de tests particulière.

Exemple : exécuter un chemin particulier d'un programme ou vérifier le respect d'une exigence spécifique.

Stratégie de test
Plan de test

C'est un document décrivant l'étendue, l'approche, les ***ressources*** *et le* ***planning*** *des activités de test prévues. Il identifie entre autres les* ***éléments*** *et caractéristiques à tester,* ***qui*** *fera chaque tâche, le degré d'indépendance des testeurs,* ***l'environnement*** *de test, les* ***techniques*** *de* ***conception*** *des tests et les techniques de* ***mesure*** *des tests à utiliser, et tout risque.*

[IEEE 829].

Plan de test

Le plan de test doit répondre aux questions :

Pourquoi tester ?

- Les plans de test logiciel doivent répertorier les objectifs à atteindre

Quoi tester ?

- Périmètre d'intervention de l'activité de test, lister les éléments du logiciel qui seront testés
- Définir les éléments qui seront exclus de la stratégie

Source : ALL4TEST.

Plan de test

Comment tester ?

- Approche globale du test, c.-à-d.. spécifier les niveaux de test, les types de test et les méthodes, en fonction des objectifs.
- Définir les critères utilisés afin d'établir si chaque élément du logiciel a réussi ou échoué.
- Définir les ressources matérielles :
  - configuration matérielle
  - configuration logicielle
  - outils de production et de support des tests
  - prérequis fonctionnels et techniques

Source : ALL4TEST.

Plan de test

Qui teste ?

- responsabilités de chaque équipe/personne
  - Test Manager, opérateur, …

Quand ?

- calendrier des tests

Définition des livrables

- Plan de Test
- Cas de test (données d'entrée, résultats attendus, …)
- Scripts de Test
- Un rapports de test contenant des fiches de test, journal de test, fiche d'anomalie

Source : ALL4TEST.

Plan de test (exemple)

Extrait du **Plan d'Assurance Qualité** pour le développement des applications du Segment Sol de la mission spatiale CoRoT :

QUOI

*Chaque **application est l'objet de tests**. Les tests des applications du segment sol doivent répondre, **de la manière la plus exhaustive possible, aux exigences** émises dans le document DA 11 (Spécification ….).*

POURQUOI

QUOI **QUI**

*Les tests **unitaires** sont **à la charge des développeurs**. Pour chaque produit dont ils sont responsables, les développeurs doivent écrire au moins un test de cas **nominal** par « fonction » (telle que définie dans le document DA 11) et plusieurs tests de **cas aux limites** (paramètres en entrée proche des valeurs limites) et plusieurs tests de **cas dégradé** (paramètre d'entrée hors des limites de fiabilité de l'algorithme).*

COMMENT

Plan de test (exemple)

QUOI

*Les **tests d'intégration, de validation et de non régression** sont élaborés en accord avec le **chef de projet logiciel et le responsable qualité des laboratoires**.*

QUI

*Les tests sont **numérotés** séquentiellement pour chaque produit. Les fichiers de tests sont placés dans le répertoire « test » de chaque produit. Chaque test **est placé dans un répertoire** T<numéro du test>.*

*Le répertoire « in » contient **les données d'entrée** du test fournis par le responsable de produit. Le répertoire « out » contient **les résultats attendus** de l'exécution du process (i.e. les résultats de référence en vue de la recette).*

LIVRABLES

***Chaque exécution** d'un test doit faire l'objet d'une **fiche de test**. Le numéro d'identification du test est composé comme suit :*

<CODE_PRODUIT>.T<Numéro_chronologique>

Plan de test (exemple)

La fiche de test *indique le « type de test » dont les modalités sont les suivantes :*

- *U pour les tests unitaires …;*
- *I pour les tests d'intégration …;*
- *N pour les tests de non régression …;*
- *V pour les tests de validations ;*
- *P pour les tests de performance.*

La fiche de test est placée directement dans le répertoire T<numéro du test>.

Les fiches de test doivent clairement indiquer comment les données d'entrées doivent être accédées et où seront placées les données de sortie afin de pouvoir les comparer aux fichiers présents dans le répertoire out.

FICHE DE TEST			
N° Test : CODE_PRODUIT.TX	Opérateur :	**Date de test :**	
Titre :		**Type de test : U/I/NR/V**	
Détail des tests	Résultats attendus	Résultats obtenus	**Conforme**
Objectif : **Mode opératoire :**			**Oui** **Non**
Step 1			
Step 2			
Step 3			
=== FIN DU TEST ===			

Données d'entrées :

Environnement :

Observations :

Journal de test

- Il s'agit d'un document qui regroupe pour une campagne de tests donnée, l'ensemble des résultats, généralement sous la forme :
  - Référence du test
  - Date d'exécution
  - Résultat « OK ou Fail »
  - Commentaire en cas d'échec.

Sommaire

Introduction et rappels

- Pourquoi tester ?
- Notions de qualité logicielle
- Rappel sur les cycles de développement logiciel
- Agilité

Les tests

- Introduction
- Classification des tests
- Stratégie de test, plan de test
- **Anomalie**
- Matrice de couverture
- Intégration continue et Framework de test

Fiche d'anomalie

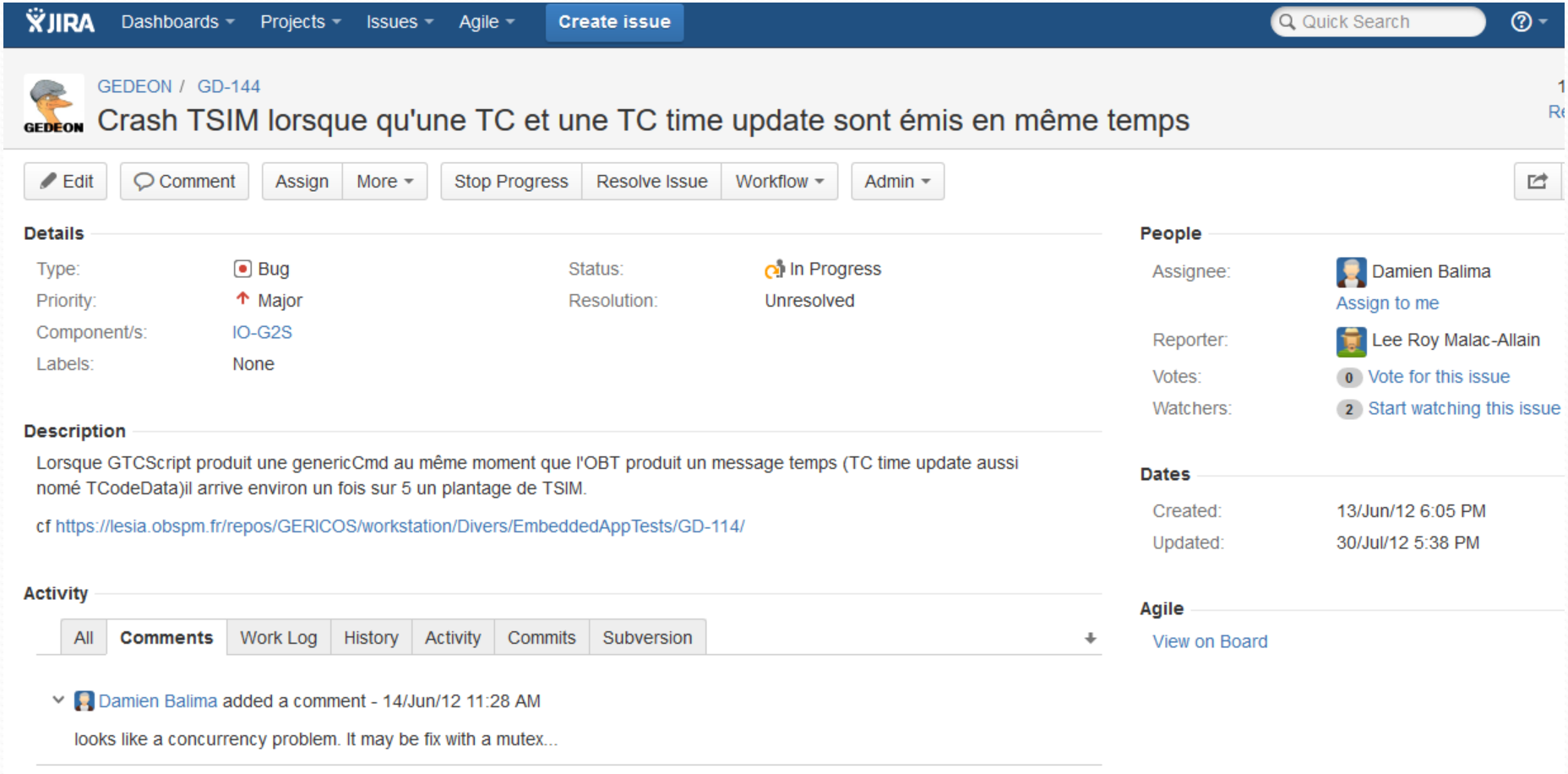
- Lorsqu'un test échoue (résultat attendu <> résultat obtenu) pour tout ou partie du test on rédige **une fiche d'anomalie (FA).**
- La FA reprend et complète la partie « résultat obtenu » de la fiche de test mais peut ne porter que sur un point particulier de l'exécution du test.
- De nombreuses applications internet permettent de saisir et de suivre le cycle de vie des FA (Jira, Redmine, Bugzilla, …)

Fiche d'anomalie

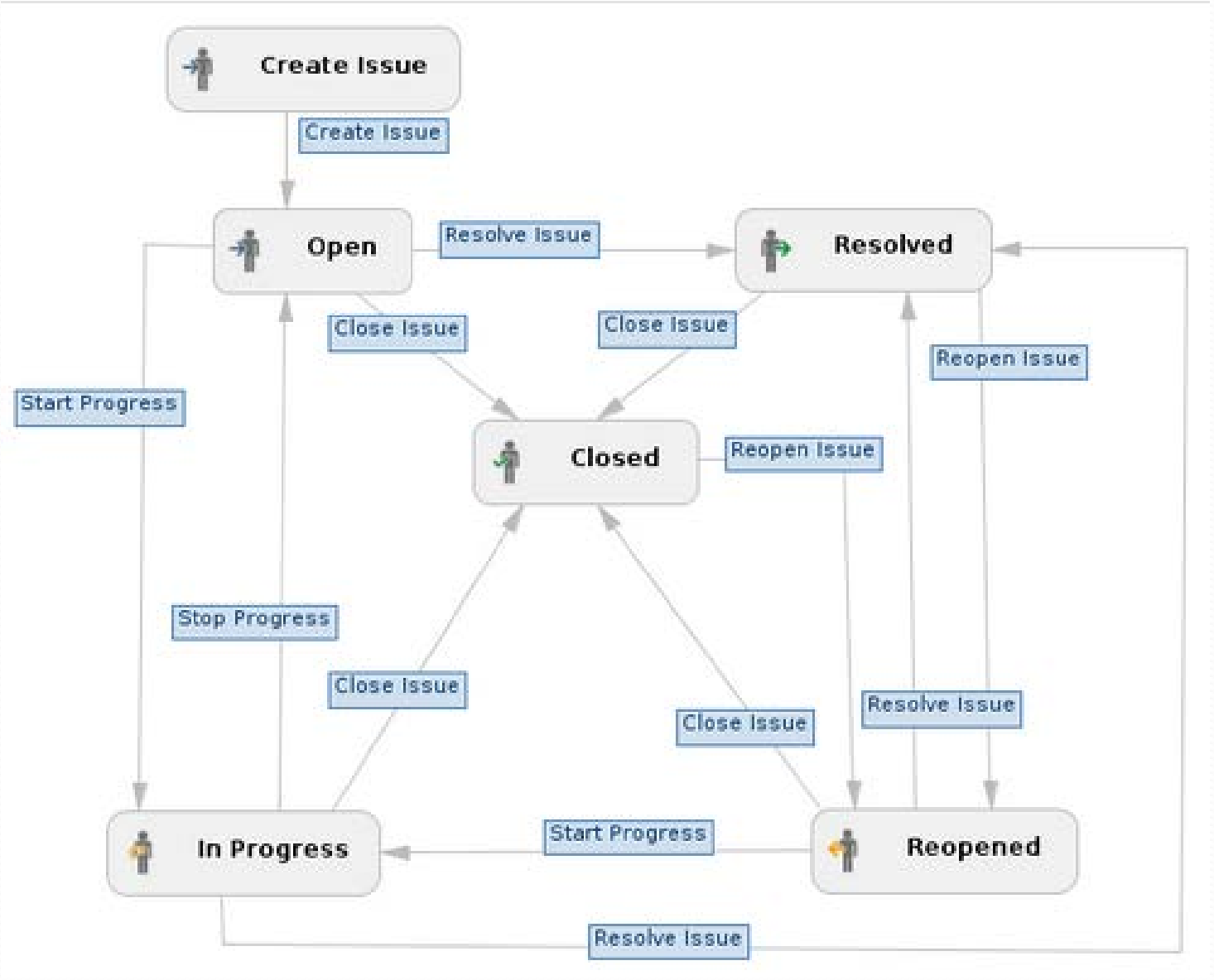
Une fiche d'anomalie comporte généralement les champs suivants :

- Référence du produit
- Version du produit
- Référence anomalie
- Rédacteur
- Détectée par
- Date de détection
- Type de fiche (Anomalie ou Demande d'évolution)
- Etat (en fonction du workflow des anomalies, ouvert, en cours de résolution, refusé, résolu, testé, validé, livré, fermé, …)
- Reproductible ou non (Heisenbug)
- Référence du test (le cas échéant)
- Description
- Environnement d'exécution (fichiers joints)
- Historique des échanges

FA sous le logiciel JIRA



Cycle de vie d'une anomalie sous JIRA



Déclinaison des anomalies

La découverte d'une anomalie à un niveau du cycle de développement va donner lieu à son analyse et à une éventuelle déclinaison de nouvelles FA aux niveaux inférieurs voire supérieurs.

Micro projet : anomalie au niveau validation (Top-Down)

Découverte d'un résultat obtenu différent du résultat attendu pour le **test de validation** de bout en bout « calcul d'une médiane »

- Ouverture d'une **fiche d'anomalie niveau validation**
- Analyse :
  - le module d'écriture des résultats ne reproduit pas fidèlement la sortie du module de calcul de la médiane => **nouvelle FA au niveau intégration**
  - Le module de calcul de la médiane donne également un résultat erroné => **nouvelle FA au niveau unitaire**

Micro projet : anomalie au niveau unitaire (Bottom-Up)

Découverte d'un résultat obtenu différent du résultat attendu pour le **test unitaire** « calcul d'une médiane » au niveau du module de calcul de la médiane

- Ouverture d'une **fiche d'anomalie niveau unitaire**
- Analyse :
  - Le module de calcul de la médiane donne un résultat erroné.
    - Parcours du cycle de développement (avec éventuellement ouverture de FA au niveau intégration voire validation)

Sommaire

Introduction et rappels

- Pourquoi tester ?
- Notions de qualité logicielle
- Rappel sur les cycles de développement logiciel
- Agilité

Les tests

- Introduction
- Classification des tests
- Stratégie de test, plan de test
- Anomalie
- **Matrice de couverture**
- Intégration continue et Framework de test

Matrice de couverture

- C'est un tableau reliant chaque test à une ou plusieurs exigences.
  - Chaque case dans le tableau signifie : si ce test échoue, cette spécification/exigence n'est pas respectée.
  - Chaque exigence doit être couverte **par au moins un test**.
  - Typiquement on aura une matrice de couverture par niveau de développement (tests unitaires, d'intégration, de validation).

	Exigence 1	Exigence 2	Exigence 3
Test 1		X	
Test 2	X		X
Test 3			X

Sommaire

Introduction et rappels

- Pourquoi tester ?
- Notions de qualité logicielle
- Rappel sur les cycles de développement logiciel
- Agilité

Les tests

- Introduction
- Classification des tests
- Stratégie de test, plan de test
- Anomalie
- Matrice de couverture
- **Intégration continue et Framework de test**

Intégration continue

*L'**intégration continue** est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier **à chaque modification de code source** que le résultat des modifications **ne produit pas de régression** dans l'application développée.*

Wikipedia

Intégration continue

- Nécessite obligatoirement une **automatisation** de la compilation et des tests.

- Prérequis :
  - Utilisation d'un **logiciel de gestion de version**
  - Obligation des développeurs de faire des commit réguliers
  - Utilisation d'un **framework de test** pour obtenir une standardisation des résultats interprétable par un logiciel
  - Utilisation d'une **application d'intégration continue** pour compiler, exécuter les tests, automatiser le déploiement, présenter les résultats.

Intégration continue

Avantages

- test immédiat des modifications
- notification rapide en cas de code incompatible ou manquant
- les problèmes d'intégration sont détectés et réparés de façon continue, évitant les problèmes de dernière minute
- une version est toujours disponible pour un test, une démonstration ou une distribution

Framework de test

- Framework d'automatisation des tests
  - Python : Pytest, unittest (Pyunit), Nose, …
  - C : Cunit, Check, …
  - C++ : CPPUnit, doctest, …
  - Java : Junit, TestNG, …

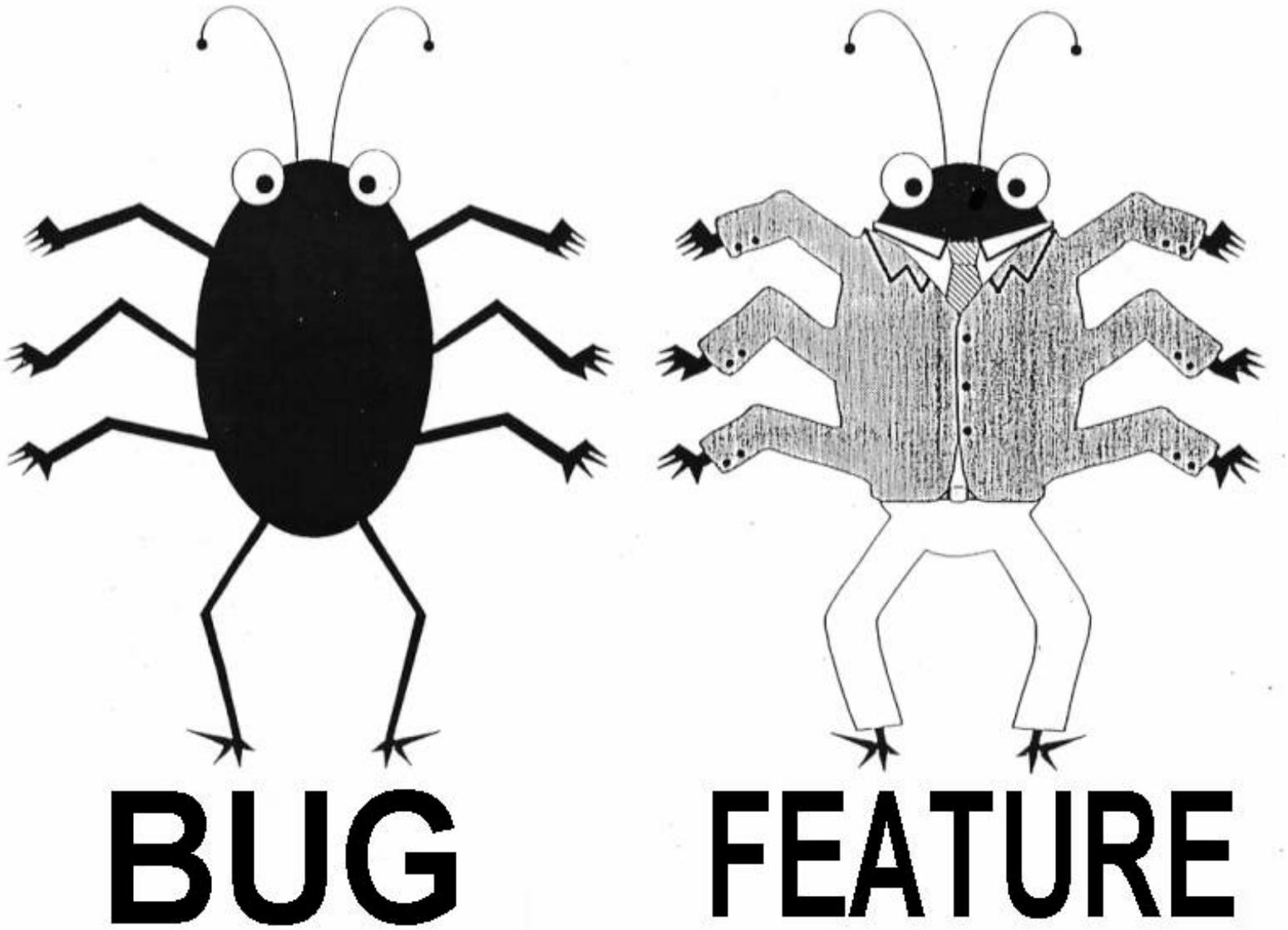
https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

- Applications de suivi des anomalies et demandes de modification
  - JIRA
  - Redmine
  - Bugzilla
  - Mantis
  - Module « Issues » de GitHub
  - …

Test Driven Development (TDD)

La logique des tests poussée à l'extrême :

Le ***Test-Driven Development*** (**TDD**) ou en français **développement piloté par les tests** est une technique de développement de logiciel qui préconise **d'écrire les tests unitaires avant d'écrire le code source** d'un logiciel.


BUG
FEATURE

3 BIGGEST
SOFTWARE LIES
#1.
"The program is fully tested & bug-free"
#2.
"We're working on documentation"
#3.
"Of course, we can modify it!"
DOH!
TestMatick