Tests logiciels

Kent Beck « Une fonctionnalité sans test automatique n'existe tout simplement pas»

Penser que les tests [et le refactoring] ralentissent le développement
c'est comme penser que prendre des voyageurs ralentit le bus
David Evans

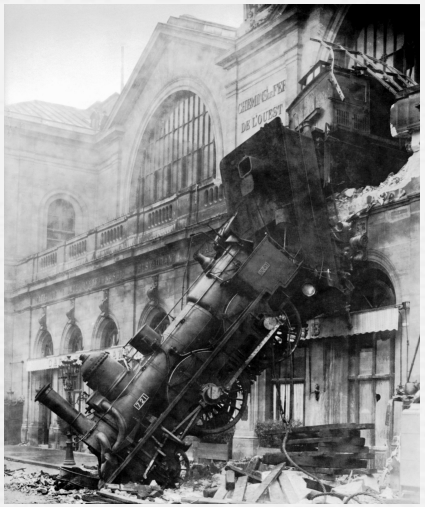
Merci à tous ceux qui ont rendu leurs cours et exposés
disponibles sur le web & dans les livres,
voir Biblio.

M. Blay-Fornarino
blay@unice.fr,
IUT Département Informatique

Bibliographie

- Programmation par les tests, ESIREM, Céline ROUDET
- Comment écrire du code testable, Conférence Agile France 2010, Florence CHABANOIS
- Reflexion on Software Quality and Maintenance, Alexandre Bergel, Chili
- An Introduction to Test-Driven Development (TDD), Craig Murphy
- Tests et Validation du logiciel, http://home.nordnet.fr/~ericleleu
- Test à partir de modèles : pistes pour le test unitaire de composant, le test d'intégration et le test système, Yves Letraon
- Les tests en orienté objet, J. Paul Gibson http://www-inf.int-evry.fr/cours/CSC4002/Documents
- Mocks and Stubs, Martin Fowler
- Introduction au test du logiciel, Premiers pas avec JUnit, Mirabelle Nebut
- Écrire du code testable Par Aurélien Bompard

Des bugs et des tests



- Objectifs
  - **Savoir où sont les bugs,** pour
    - Les corriger ou
    - les documenter et donner des contournements :
      - de version en version on voit les corrections apparaître (même dans le hard il y a des versions);
      - Eviter la mauvaise image de la découverte du bug par le client
  - **Eviter le update qui régresse**, c'est pire qu'un bug présent, ...
  - Oser améliorer son code sans avoir peur d'introduire de «nouveaux» bugs
  - Retour sur les exigences(bugs show product usage)
- En conclusion : 80% du code sert à tester les cas d'erreur (principe de Pareto)

Tests...

Tests...

- unitaire
- integration
- GUI
- non regression
- coverage
- load
- stress
- performance
- scalability
- volume
- usability/utilisateurs
- security
- recovery
- L10N/l18N (langue?, symbole)
- accessibility
- Installation/configuration
- Documentation
- Platform
- samples/tutorial
- code inspections

Qui teste ?

Qui teste ?

- L'utilisateur
- Les collègues en charge du test (s'il y en a)
- Le **développeur** : il a le devoir de fournir un code le plus clair et le mieux testé possible....
- La **machine**

Qu'est-ce qu'on teste et comment ?

Qu'est-ce qu'on teste et comment ?

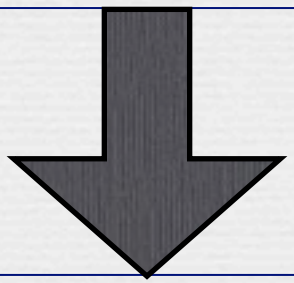
Quoi ?

- Fonctionnalité
- Sécurité / intégrité
- Utilisabilité
- Cohérence
- Maintenabilité
- Efficacité
- Robustesse
- Sûreté de fonctionnement

Qu'est-ce qu'on teste et comment ?

Quoi ?

- Fonctionnalité
- Sécurité / intégrité
- Utilisabilité
- Cohérence
- Maintenabilité
- Efficacité
- Robustesse
- Sûreté de fonctionnement

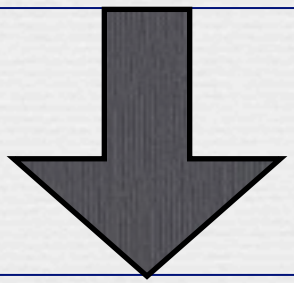


Une spécification exprime ce qu'on attend du système, elle prend différentes formes en fonction de ce qui est ciblé : cahier des charges, use cases, données numériques, ...

Qu'est-ce qu'on teste et comment ?

Quoi ?

- Fonctionnalité
- Sécurité / intégrité
- Utilisabilité
- Cohérence
- Maintenabilité
- Efficacité
- Robustesse
- Sûreté de fonctionnement



Comment ?

- **Test statique :**
  - relecture / revue de code
  - analyse automatique (vérification de propriétés, règles de codage ...)
- **Test dynamique :**
  - exécution du programme, et observation du comportement en fonction des valeurs en entrée.

Une spécification exprime ce qu'on attend du système, elle prend différentes formes en fonction de ce qui est ciblé : cahier des charges, use cases, données numériques, ...

S'organiser pour tester

- Module contenant les tests, indépendant du code « réel »
- Faire des tests indépendants les uns des autres
  - Pas de tests imbriqués
  - Un test qui échoue ne fait pas échouer les tests suivants
- Automatiser les tests
  - Ant ou Makefile : compiler et lancer automatiquement tous les tests
  - Utiliser un Environnement de tests : framework JUnit
  - Tests Unitaires/Tests de non régression
- Réutiliser les tests et leurs résultats comme documentation

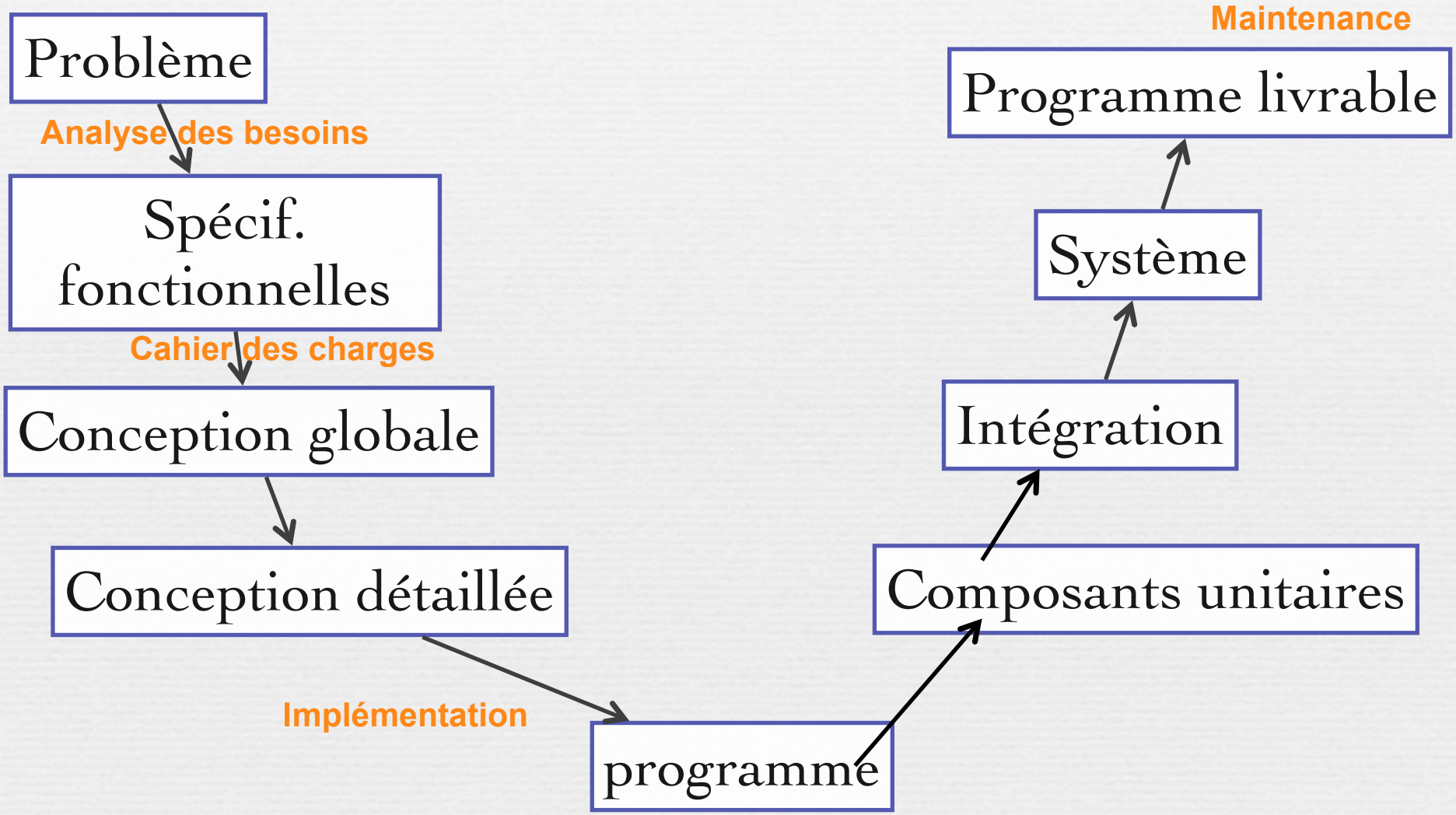
Construire des suites de tests

Types de Tests

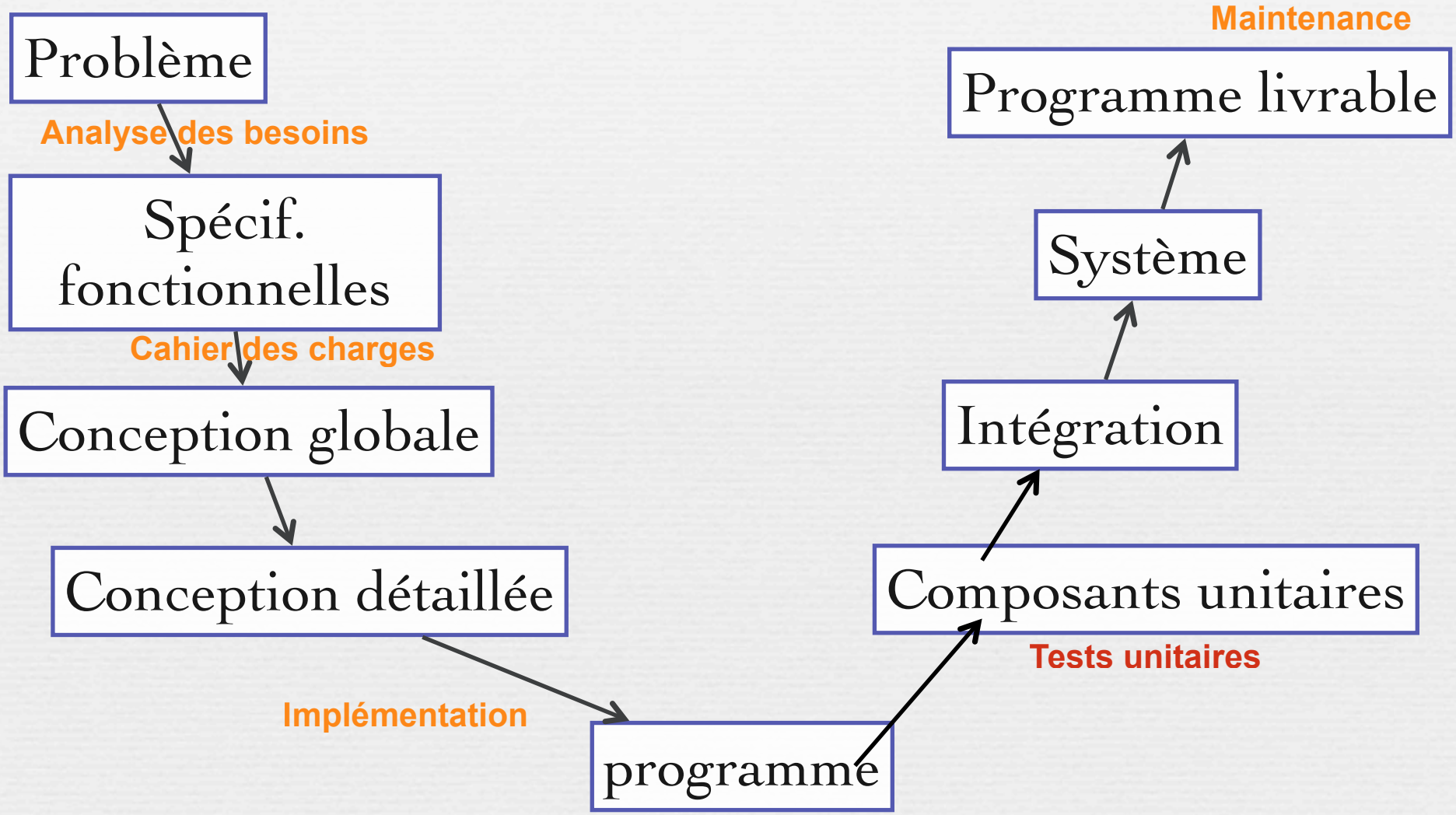
Types de tests

- Tests en boîte noire: par ex. Tests système/fonctionnels
  - Utilise la description des fonctionnalités du programme
  - Provenant des spécifications (scenarii, uses cases)
- Tests en boîte blanche: par ex Tests structurels
  - Utilise la structure interne du programme
- Tests de (non) régression (après modifications)
  - Correction et évolution ne créent pas d'anomalies nouvelles
- Tests de robustesse
  - Cas de tests correspondant à des entrées non valides
- Tests de performance (application intégrée dans son environnement)
  - load testing : résistance à la montée en charge
  - stress testing : résistance aux demandes de ressources anormales

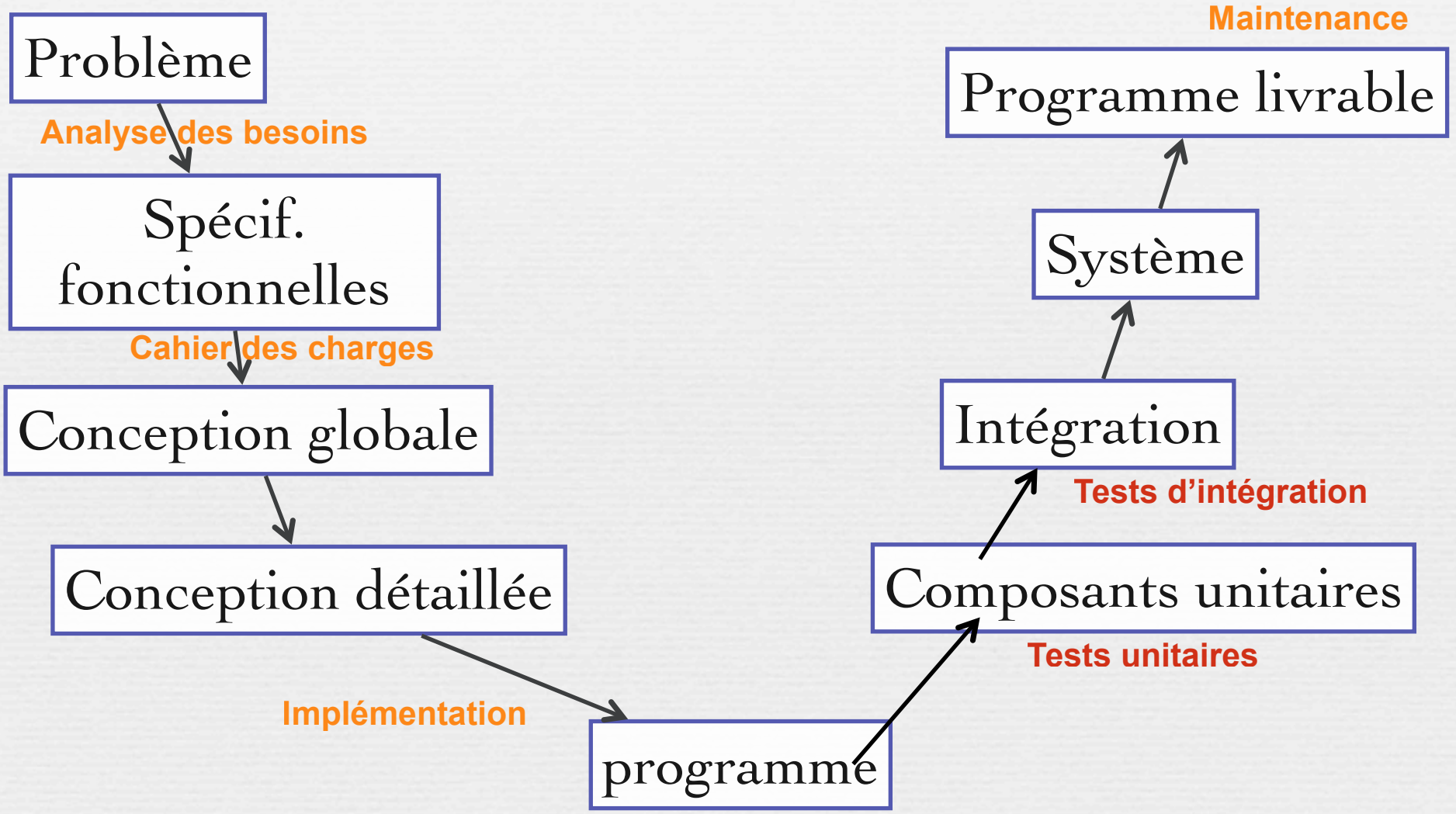
Hiérarchisation des tests dans le cycle en V



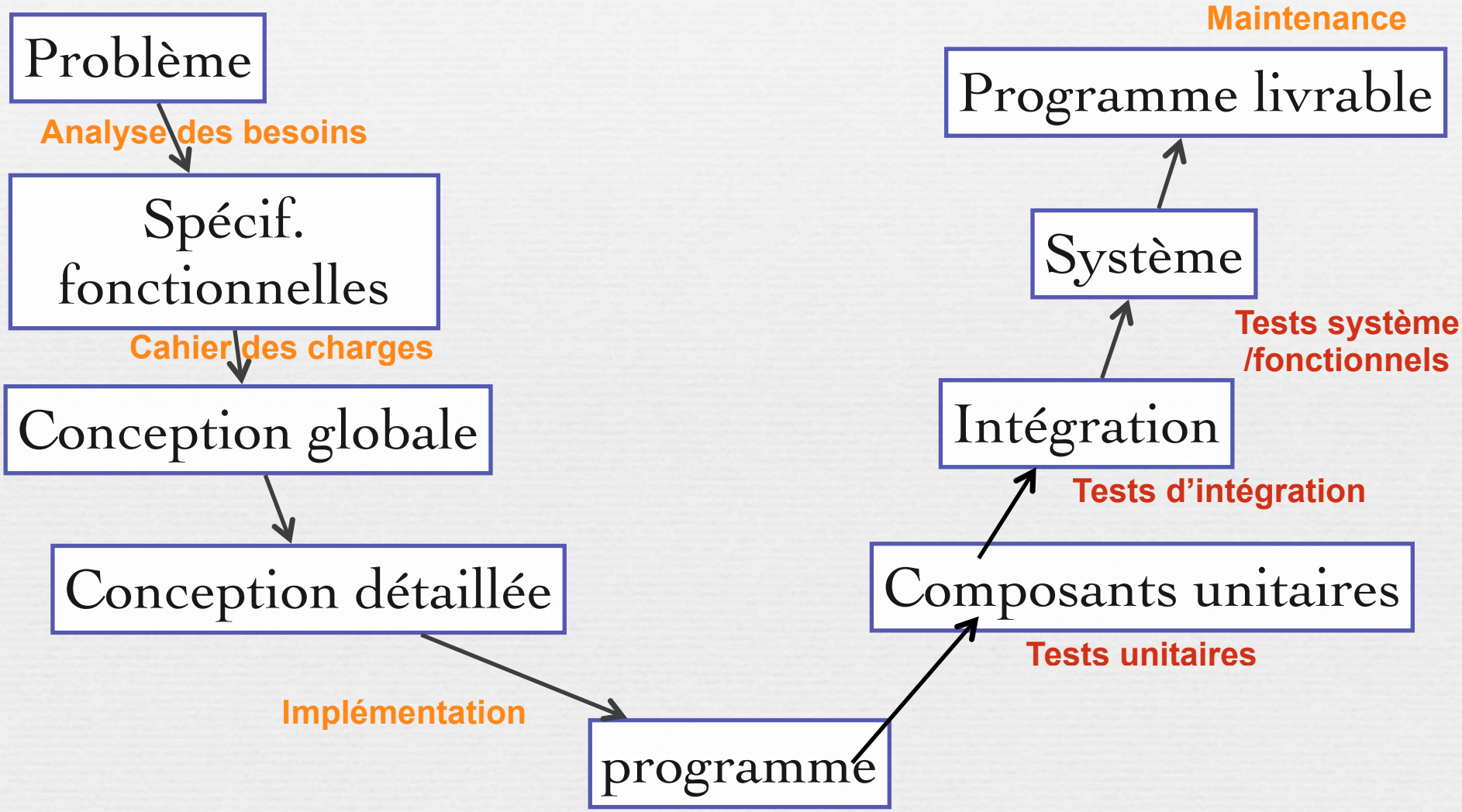
Hiérarchisation des tests dans le cycle en V



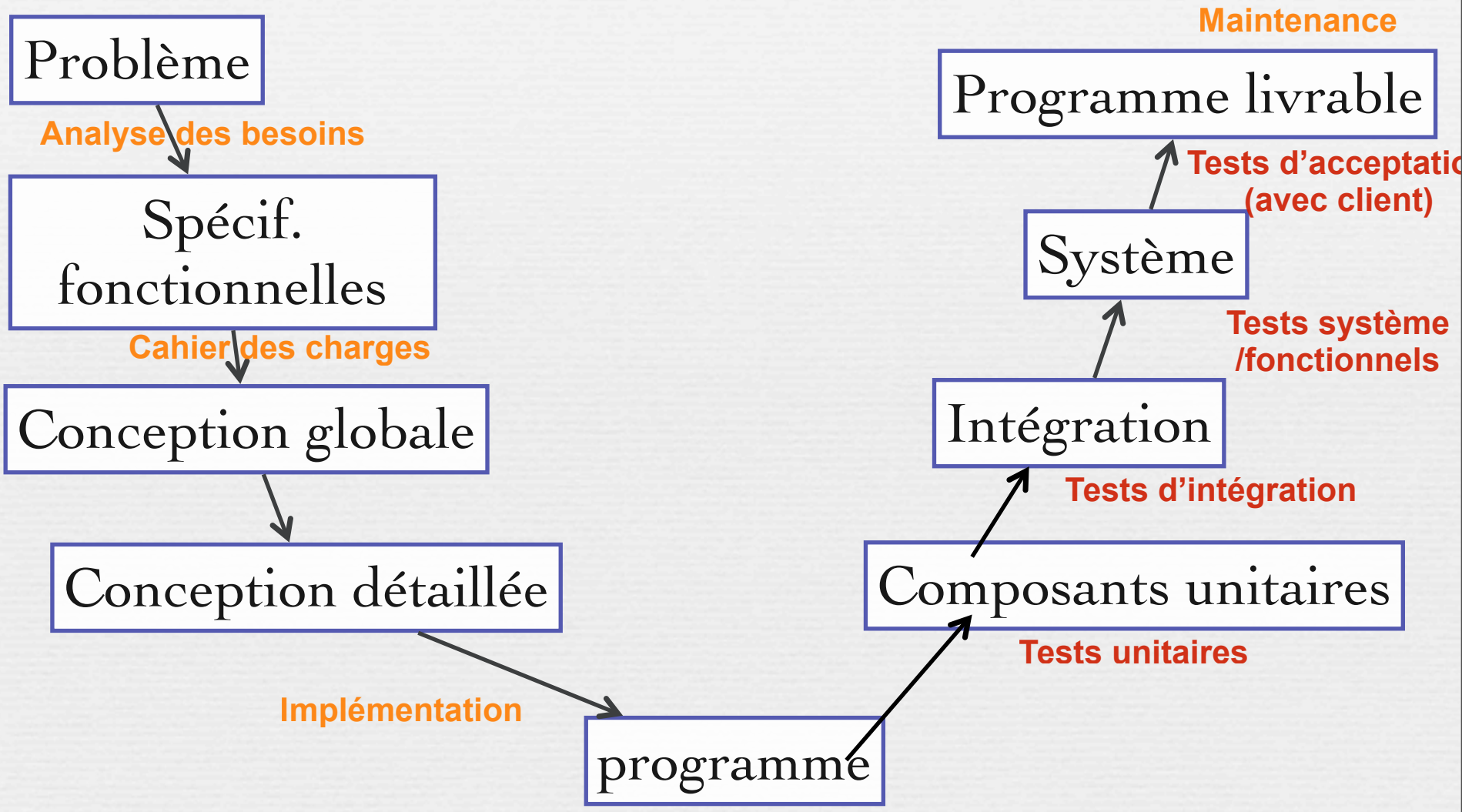
Hiérarchisation des tests dans le cycle en V



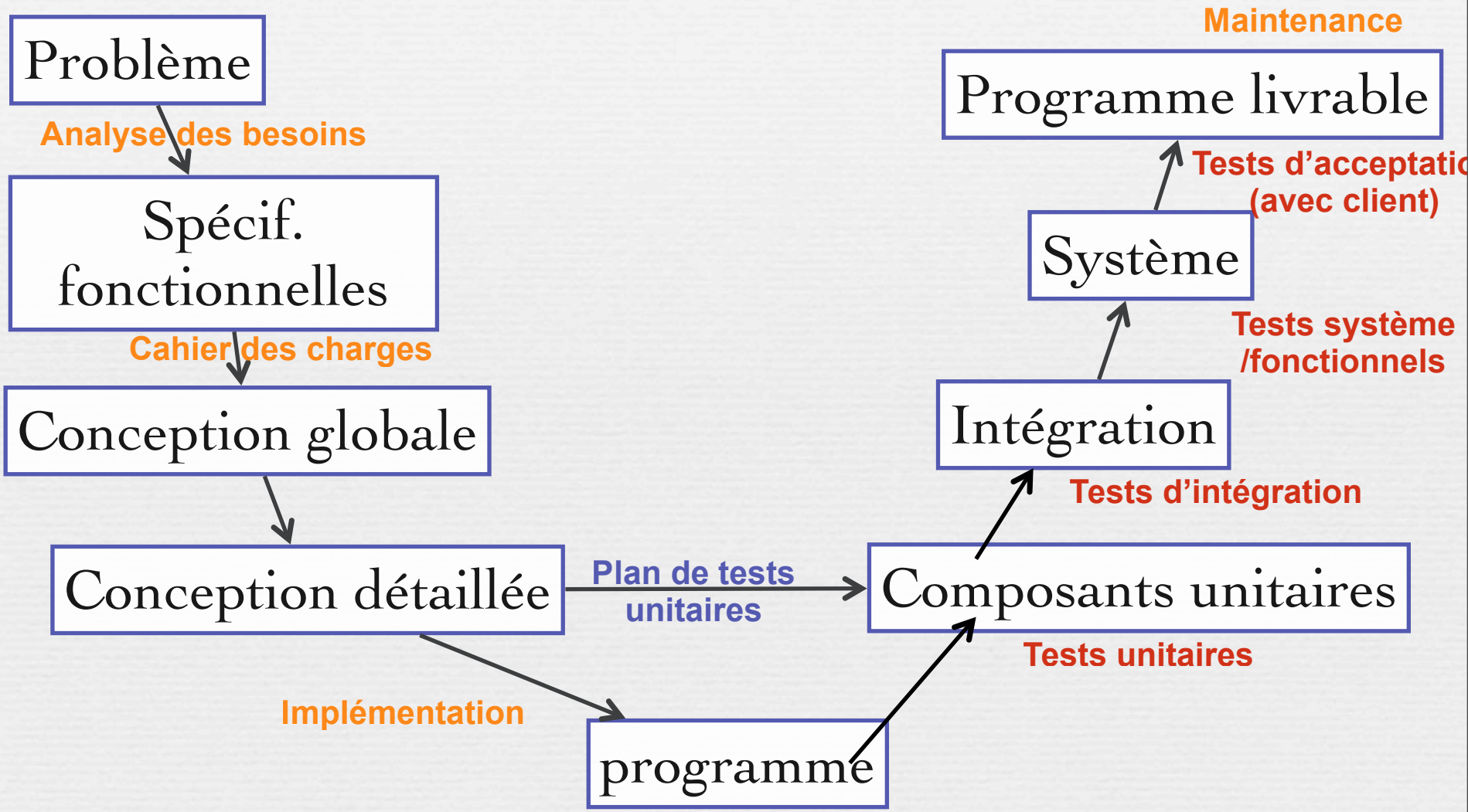
Hiérarchisation des tests dans le cycle en V



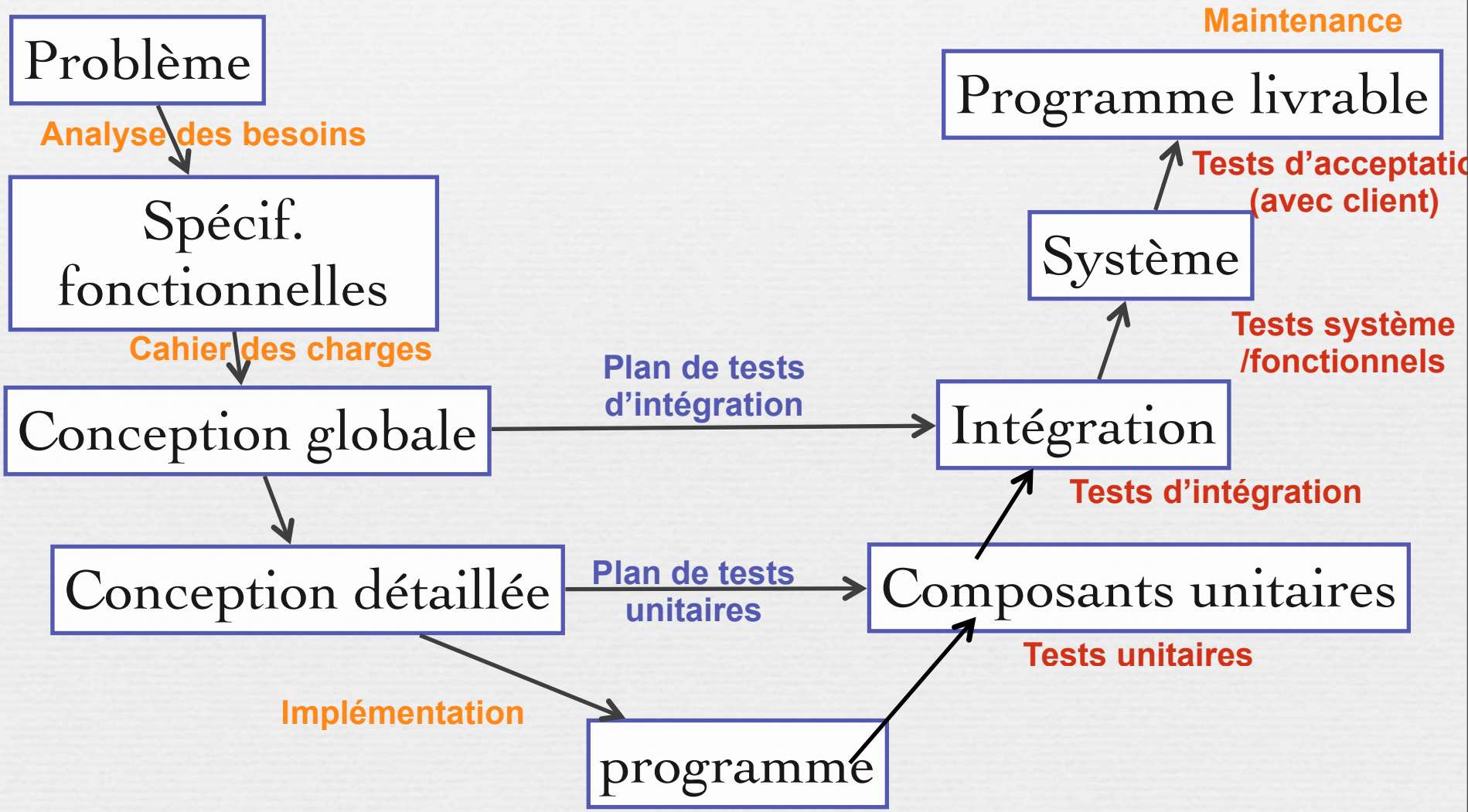
Hiérarchisation des tests dans le cycle en V



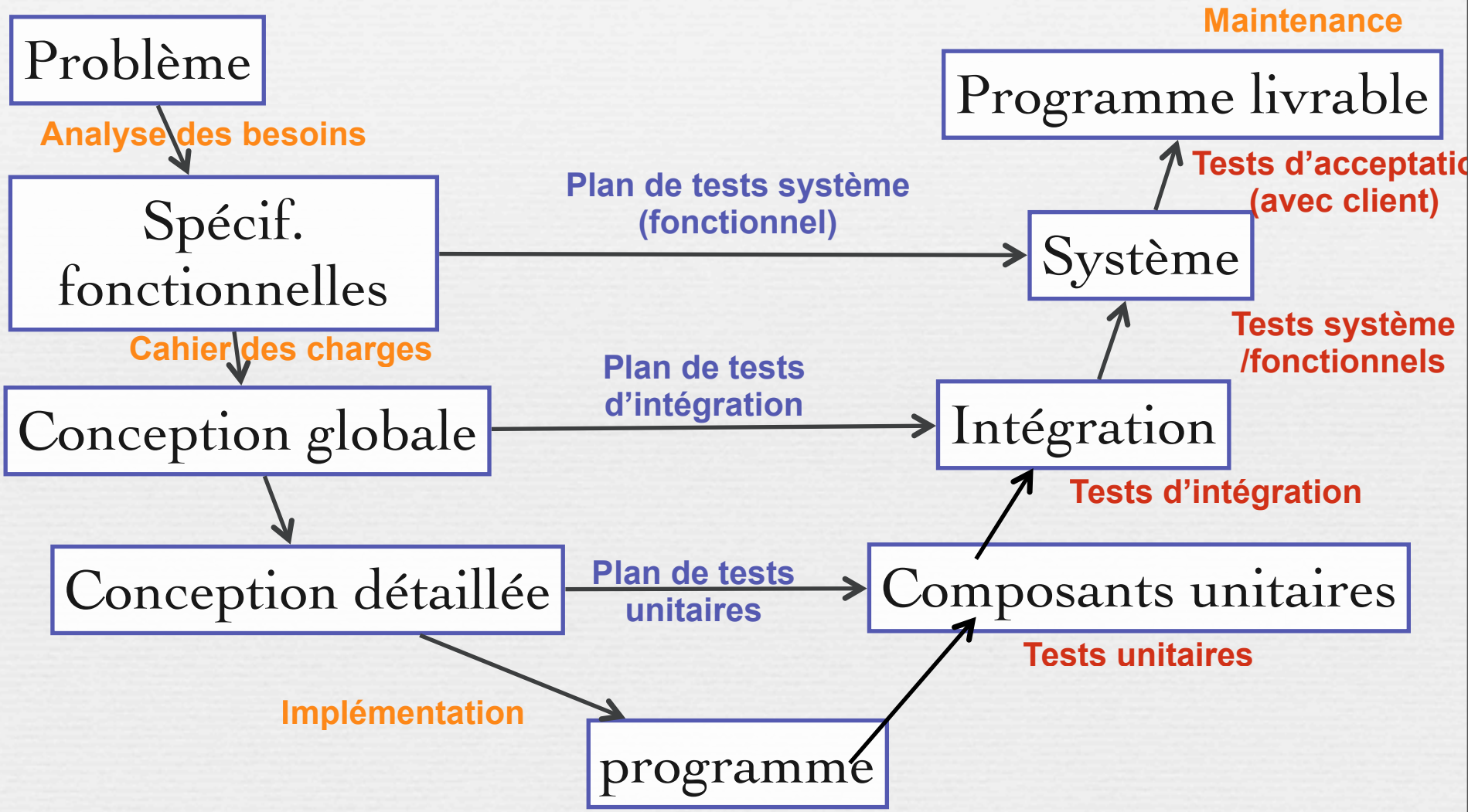
Hiérarchisation des tests dans le cycle en V



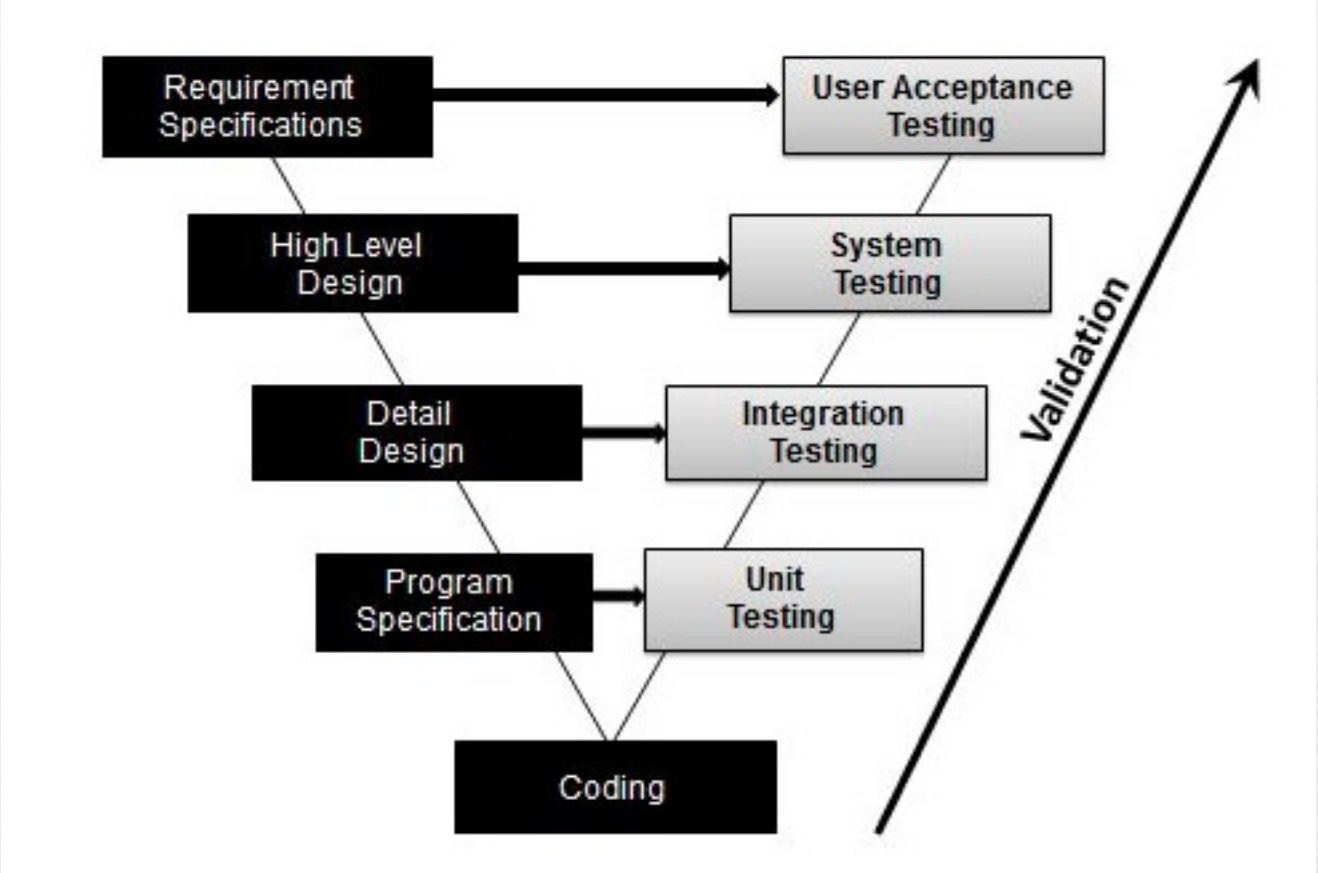
Hiérarchisation des tests dans le cycle en V



Hiérarchisation des tests dans le cycle en V



Tests de validation



http://www.tutorialspoint.com/software_testing_dictionary/validation_testing.htm

Diagrammes UML et Tests

- Niveau Application (spécification)
  - Diagramme des cas d'utilisation — Tests Système/Fonctionnel
  - Diagramme de classes : test des associations, des agrégations
    - multiplicité,
    - création, destruction — Tests d'intégration
  - Diagramme de séquence : test de séquences
    - construction d'un graphe de flot
- Niveau Classes (conception détaillée)
  - Classes détaillées — Tests d'intégration
  - Diagrammes de machine à états — Tests unitaires

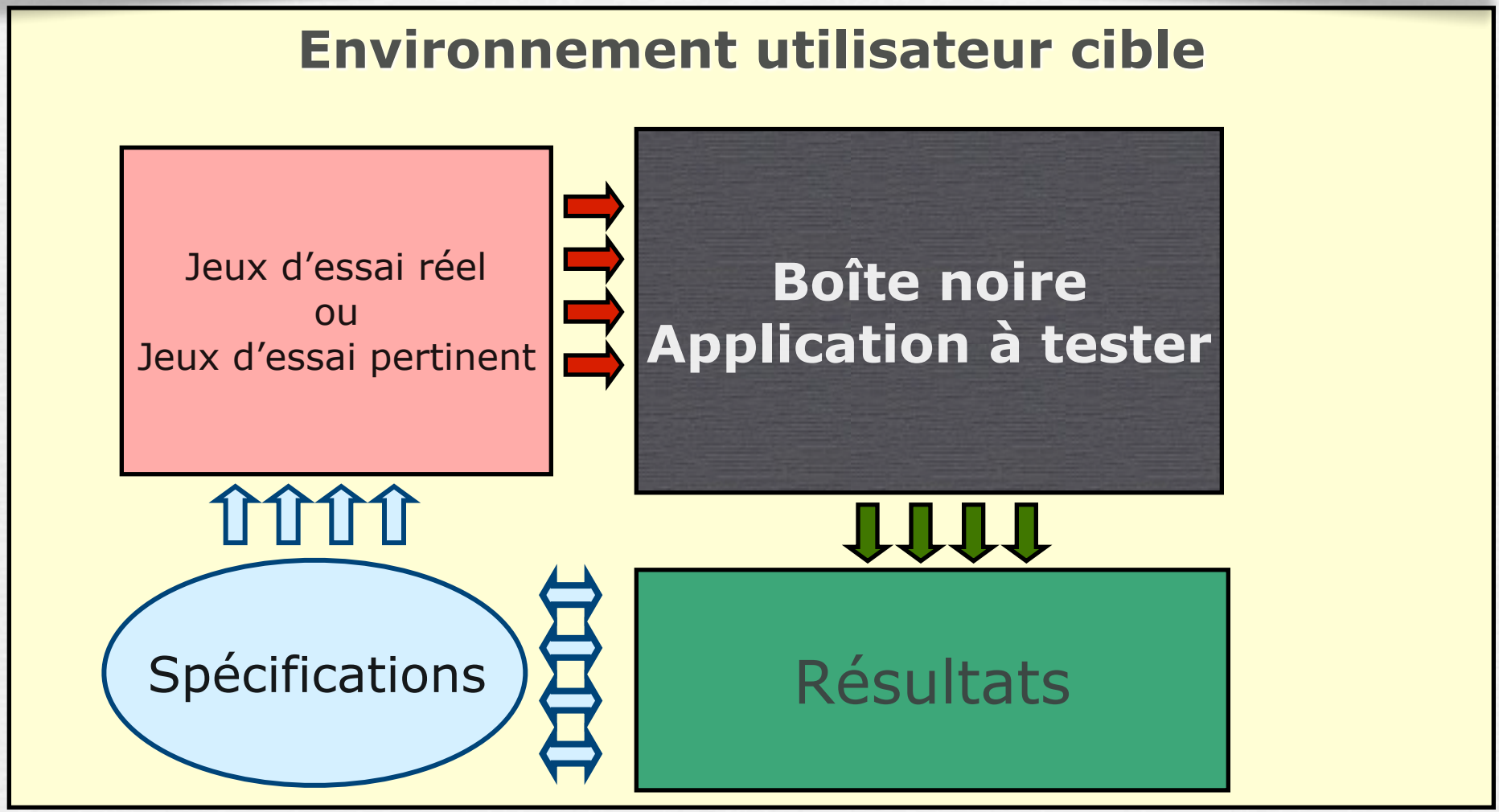
De la préparation des tests à leur exécution

- A) Préparation :
  - 1) Tests de validation
  - 2) Tests d'intégration
  - 3) Tests unitaires
- B) Codage et exécution
  - 4) Tests unitaires
  - 5) Tests d'intégration
  - 6) Tests de validation

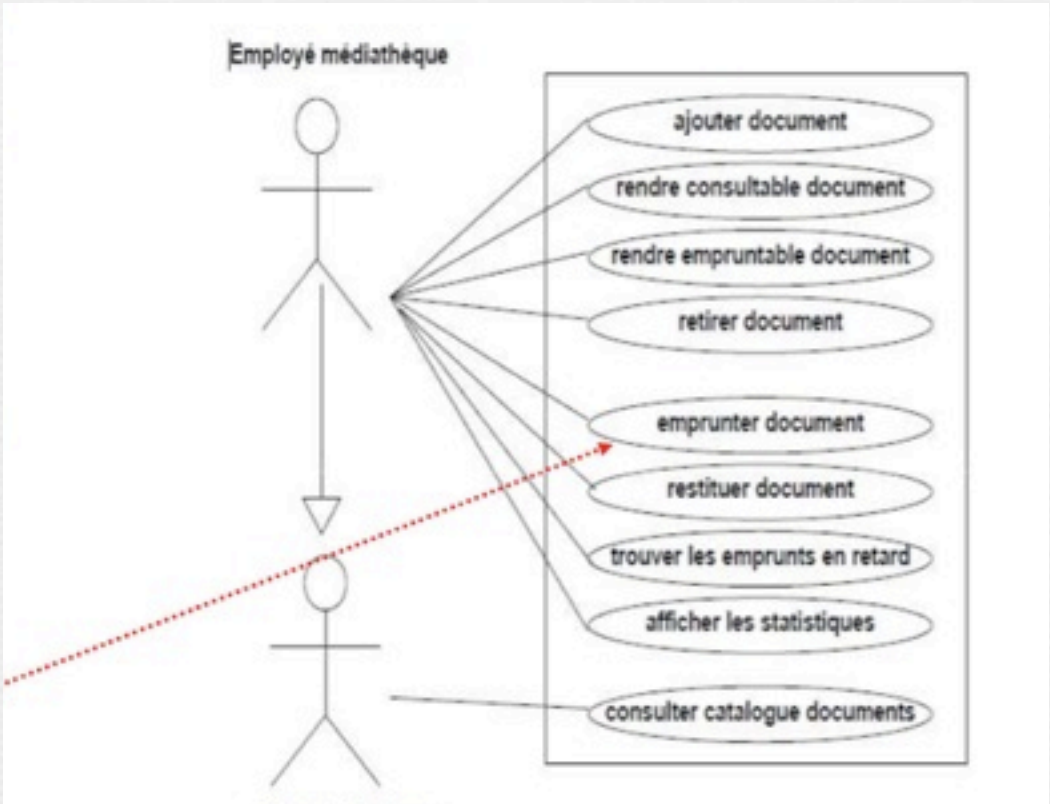
Tests Fonctionnels

Test Système (fonctionnel)

Vérifier que les fonctions correspondant aux attentes sont *bien atteintes*



Tests fonctionnels & UML



Tests fonctionnels & UML

Données d'entrée

client: peut être inscrit ou non;

emprunts: déja effectués par le client

- existe-t-il un emprunt en retard ?
- le nombre d'emprunts déja effectués correspond-il au nombre maximum de ce client ?

document:

- existe?
- empruntable ou consultable?,
- déjà emprunté ou disponible?

Tests fonctionnels & UML

Données de sortie
`Emprunt` accepté ou refusé.

Remarque: la définition des jeux de tests de validation pour le cas d'utilisation `emprunter document` permet de soulever au moins les questions suivantes (à poser au client):

- un abonné qui n'est pas à jour de sa cotisation peut-il tout de même emprunter un document?
- doit-il être considéré comme un client au tarif normal tant qu'il n'a pas renouvelé son abonnement?
- ou doit-il se réabonner avant de pouvoir emprunter un document?

D'une manière générale, la préparation des jeux de tests permet de lever les ambiguïtés et réparer des oublis.

Use cases et Tests

- Pour chaque d'utilisation, sélectionner les scénarios et définir les tests correspondants.

Info | Flow of Events | Details | Requirements | Diagrams | Test Plan | References | Description

Normal Scenario ... Main ...

Steps	Procedures	Expected Results
1. System issue delivery request to courier company	System issue delivery request (WS-D001) to courier company through web services.	Delivery request WS-D001 generated.
2. Courier company collect items from warehouse, pack it and ship it out	Once courier company receive WS-D001, then collect items from warehouse and pack it up. Then mark WS-D001 as ready and past to delivery staff.	Delivery staff receive WS-D001 which is marked as ready.
3. Courier company mark delivery complete	Delivery staff mark delivery complete in handheld device.	Order mark delivery completed.
4. System confirm order closed	System receive delivery complete signal, and mark order as closed automatically.	Order marked as closed.

http://www.visual-paradigm.com/product/vpuml/tutorials/testingprocedure.jsp

Tests fonctionnels & US

En tant que passager PRIMO, **je veux** annuler ma réservation,...

Seules les taxes « Autres » me sont remboursées moins les frais administratifs

En tant que passager BUSINESS FLEX,, **je veux** annuler ma réservation, ...

L'ensemble du billet m'est remboursé à 100% moins les frais administratifs

Tests fonctionnels...

http://seleniumhq.org/docs/

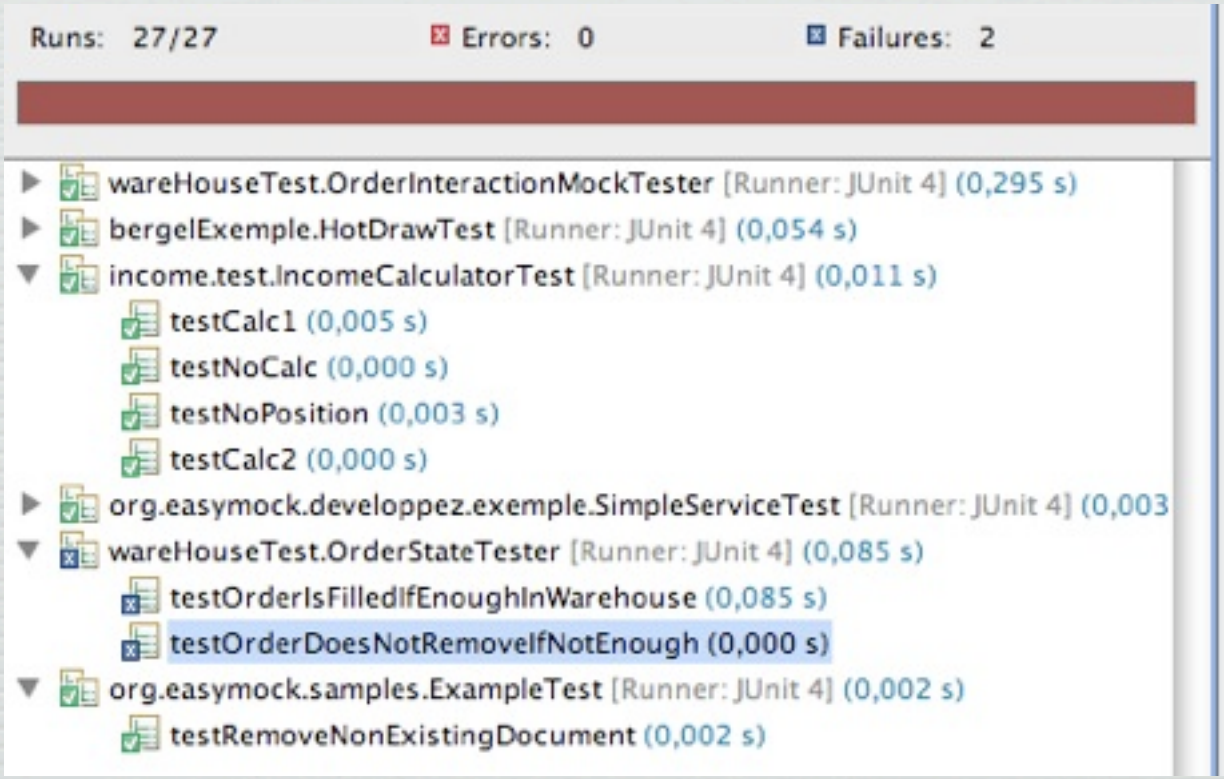
http://watir.com

Une démonstration...

http://www.opensourcetesting.org

Tests Unitaires

exemple JUnit



Tests Unitaires (déjà étudiés en AP)

- Tester une unité logicielle isolée du reste du système
  - Plus petit composant compilable
  - Pour un langage procédural : unité de test = procédure
  - Pour un langage objet : unité de test = classe
  - Test de **l'interface**
  - Les unités sont-elles suffisamment spécifiées ?
  - Le code est-il lisible, maintenable ...?

- Importance de la notion de contrats (pré/post)

Tests d'intégration (voir cours dédié)

✓Différents modules d'une application peuvent fonctionner unitairement, leur intégration, entre eux ou avec des services tiers, peut engendrer des dysfonctionnements.

✓Il est souvent impossible de réaliser les tests unitaires dans l'environnement cible avec la totalité des modules à disposition.

➡Les tests d'intégration ont pour objectif de créer une version complète et cohérente du logiciel (avec l'intégralité des modules testés unitairement) et de garantir sa bonne exécution dans l'environnement cible.

Autres tests...

Usability Testing

- Purpose: Find out is the system is really usable by its intended audience

- Why?
  - System is built by developers … but used by Business Users
  - Even minimal UI changes can confuse business
  - users with years of experience of “doing it this way”
  - System has to face real-life usage

Usability Testing

- How: Almost impossible to automate
- Tips:
  - Involve ergonomic specialists early in the project
  - Use reusable, standardized UI components
  - Take performance into account: a slow responding system won't be accepted easily
  - Have Business Users test early on UI mock

Usability Testing

- **Tools**: Eye tracking

Extraits d'un rapport de tests utilisateurs

Méthodologie de test

➔ Déroulement des tests :

6 entretiens individuels, d'un durée moyenne de **1h30** chacun, ont été réalisés du **18 au 26 juin 2014** à Nice et Sophia Antipolis. Lors de chaque passation, l'interface a été enregistrée via le **logiciel Morae** et le comportement des utilisateurs a été filmé grâce à une webcam. Les extraits vidéo les plus pertinents seront communiqués au laboratoire I3S.

Chaque test s'est déroulé en **trois étapes** :

1. Entretien pré-test (questions sur le profil du participant),
2. Test de l'interface (plusieurs tâches à réaliser),
3. Entretien post-test (questionnaire de satisfaction et évocation).

➔ Tâches réalisées :

Tâche A : « Vous avez besoin de diffuser des informations sur un écran lors d'un événement. Vous souhaitez diffuser un diaporama photo avec fondu ainsi que de brèves actualités. »

Tâche B : « Vous pouvez maintenant retourner en page d'accueil. »

Tâche C : « Vous souhaitez modifier votre écran d'affichage. Vous voulez ajouter un calendrier avec le diaporama photos (dans la même zone). »

Tâche D : « Vous souhaitez construire un écran dans le cadre de votre travail, selon vos envies et besoins. Vous êtes libre. Comment vous y prenez-vous ? »

Profil des participants

UTILISATEURS






Nathalie	Françoise	Sophie	Laurence	Olivier	Clément
47ans	35ans	36ans	41ans	37ans	30ans
Secrétaire	Chargé de communication	Chargé d'affaires	Ingénieur d'Etudes, Responsable administratif	Chargé de projets	Animateur socioéducatif
depuis plus de 10 ans	depuis plus d'1 an	depuis plus de 5 ans	depuis plus de 5 ans	depuis 4 ans	depuis plus de 5 ans

Tous pourraient être amenés à configurer un diffuseur d'informations dans le cadre de leur travail.

Profil des participants

INTERNET

Des testeurs connectés en permanence (4/6)

sinon plusieurs heures par jour

Niveau en

ANGLAIS

3 avancés
1 intermédiaire
2 débutants

Connaissances en

GRAPHISME

1 beaucoup
3 un peu
2 pas du tout

Connaissances en

PROGRAMMATION

Aucun
0

3/6 utilisent des comparateurs de prix, des outils de configuration en ligne

Classification des principaux problèmes

	Fréquence	Gravité	Resultats (sévérité)
Absence d'aperçu	4	3	Très haut
Incompréhension du mode de sélection	4	3	Très haut
Manque d'explications	4	3	Très haut
Vocabulaire confus	4	3	Très haut
Absence de plan	4	2	Haut
Undo complexe	4	2	Haut
Choix inutiles	3	2	Moyen
Icônes peu parlantes	3	2	Moyen
Contenu de la rubrique « help »	3	2	Moyen
Impossibilité de supprimer une configuration	2	2	Moyen
Affordance de la colonne de droite	4	1	Moyen
Langue anglaise	1	3	Bas

Test de charge

- Volume des données traitées
- Nombre d'utilisateurs simultanés
  - Exemple de Système RESARAIL qui permet d'obtenir les horaires, les prix, effectuer la réservation et émettre des billets SNCF

Nécessité d'une bonne estimation des besoins dès le début et prise en compte de leur évolution

Tests de charge

Les tests définis précédemment, valident le logiciel en terme de fonctionnement simple et unitaire. En effet, dans la plupart des cas, chaque test simule **une opération d'un unique utilisateur**. Cela permet de s'assurer de la **fiabilité** du logiciel mais ne garantie en rien sa **robustesse**.
Quel sera son comportement lorsque plusieurs utilisateurs accèderont aux fonctionnalités du logiciel simultanément ?

Les tests de charge simulent, via des injecteurs, le comportement d'un nombre définit d'utilisateurs. Ils se basent sur des scripts simulant des actions utilisateurs réelles ou réalistes et avec des jeux de données proches de celles de production. En effet, l'action de 100 utilisateurs demandant le détail d'un client sera différente si le nombre de clients est de 10 ou de 100 000. De même, si les 100 utilisateurs demandent le détail du même client, il est probable que celui-ci soit mis en cache quelque part par le logiciel et que le résultat soit extrêmement rapide.

Il est par conséquent extrêmement important de bien penser le test de charge pour le rendre le plus réaliste possible et donc plus efficace. Comme tous les tests, le plus tôt ceux-ci seront mis en œuvre, plus vite une éventuelle erreur dans la conception, l'implémentation ou le dimensionnement de l'infrastructure sera détectée et plus rapidement les correctifs pourront être mis en œuvre en minimisant leur impact.

Limites du test

- L'espace des entrées -La *complexité*
- Les séquences d'exécution -La *complexité*
- Sensibilité aux fautes –Transformation entre le *fini* et *l'infini*
- Tester un programme permet de montrer la présence de fautes mais en aucun cas leur absence.
- Les tests basés sur une *implémentation* ne peuvent pas révéler des omissions car le code manquant ne peut pas être testé
- Comment être sûr qu'un système de test est correct … il faut le tester?

Tests Statiques

A voir ultérieurement

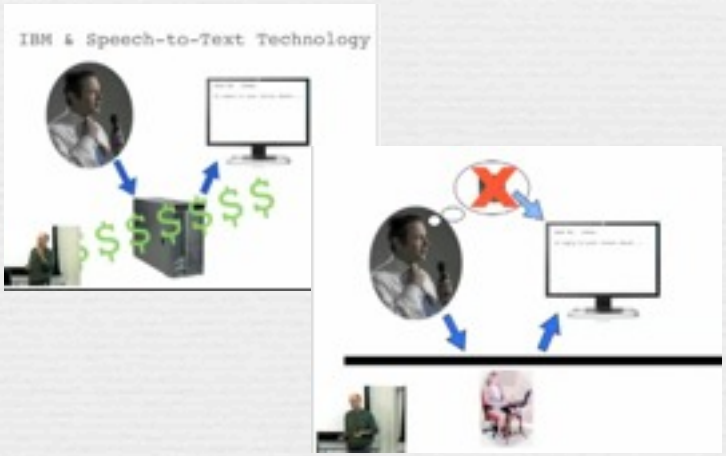
next

Encore plus en amont, le «preprototype» ;-)

Pretotyping	Prototyping
• **Investment:** hours, days	• **Investment:** days, weeks
• **Main Q:** Would we use it?	• **Main Q:** Can we build it?
• **Deliverable:** [Working] pretotype	• **Deliverable:** Working prototype

Palm «pre»

- ➡now beats later
- ➡doing beats talking
- ➡simple beats complex
- ➡commitment beats committees



http://www.pretotyping.org/

"Whenever you are tempted to type something into a print statement or a debugger expression, write it as a test instead." -- Martin Fowler

Les Tests sont un tuteur

http://emmanuelchenu.blogspot.com/