Conception Logicielle
I - Cycle de développement & Tests

- - - - - - - - - -
1 – Qualités d'un logiciel
2 – Cycles de développement & place des tests
- - - - - - - - - -

ENSMA A3-S5 - période A

2021-2022

M. Richard
richardm@ensma.fr

Sommaire

I - Qualités d'un logiciel

1 – Qualités externes & internes

- Classées en deux catégories :
  1. *Qualités externes* : importantes car impactent l'utilisation du logiciel par les utilisateurs
  2. *Qualités internes* : très importantes car impactent fortement le cycle de vie du logiciel

- Mais les qualités internes influent aussi sur les qualités externes :
  - *Justesse* : juste par rapport aux spécifications
  - *Robustesse* : réagit correctement à des conditions anormales (i.e. non prévues par les spécifications)
  - *Évolutivité* : facilement ajustable aux changements de spécifications
  - *Vérifiabilité* : facilite la mise en place de procédures de test
  - *Extensibilité* : extension/ajout de fonctionnalités possible et facilitée
  - *Réutilisabilité* : facilement associable à d'autres éléments logiciels (Interface graphique, BDD, ...),
  - *Efficience* : nécessite peu de ressource (optimisation, complexité, ...)
    - parfois contradictoire avec réutilisabilité et extensibilité
  - *Portabilité* : Facilement transférable d'un environnement [1] à un autre
  - *Utilisabilité* : apprentissage rapide et utilisation aisée (nécessite très souvent de bien connaitre les futurs utilisateurs)
  - *Intègre* : sécurité des données, accès sécurisés, ...

[1] Un environnement est défini par l'ensemble matériel + Système d'exploitation

- Comment répondre au mieux à l'ensemble de ces critères ?
  $\Longrightarrow$ Faire un compromis . . .

Néanmoins, quatre qualités importantes doivent être privilégiées :

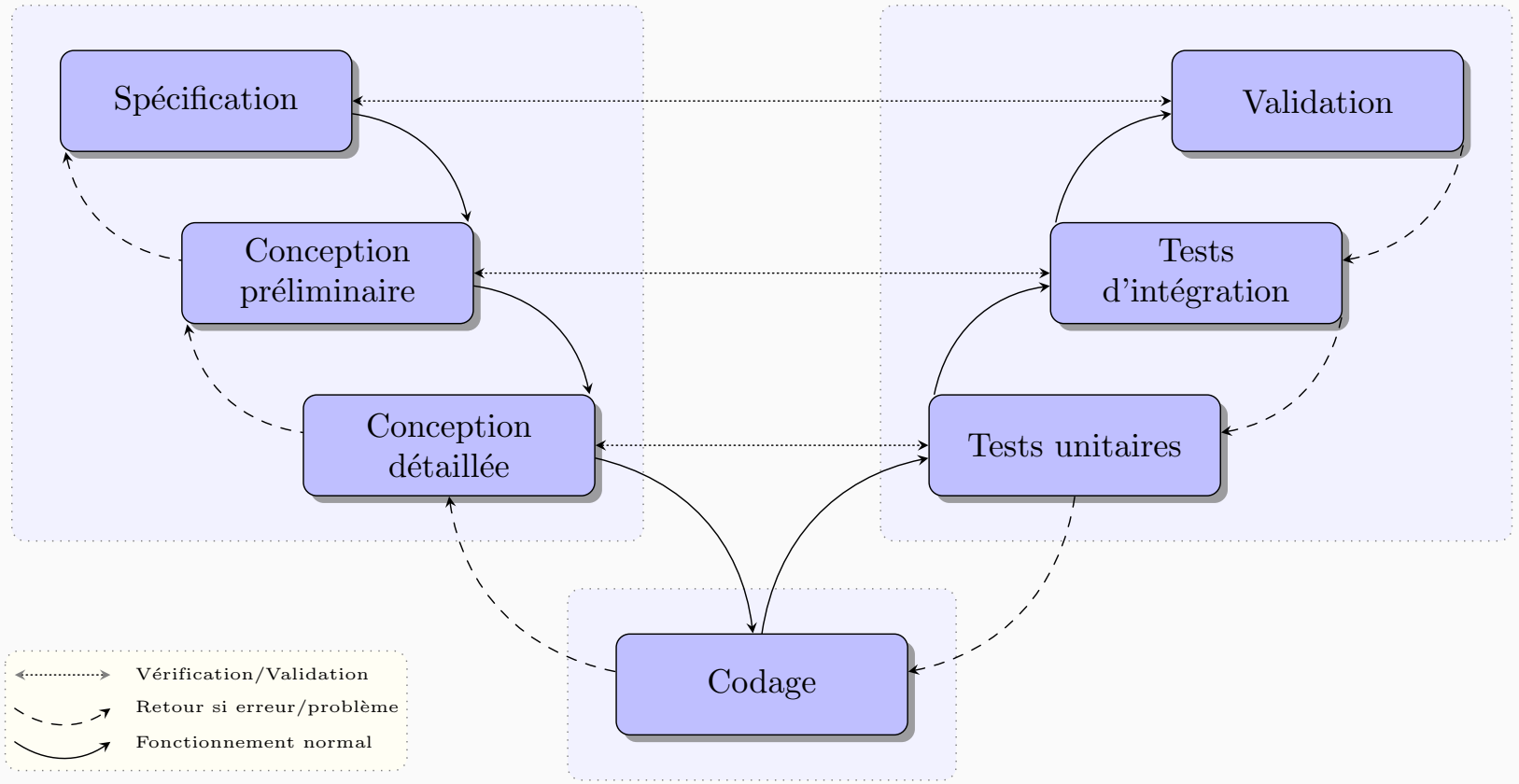
- *Justesse et robustesse :*
  Le respect de ces deux premiers points sera facilité par l'utilisation de :
  - Spécifications formelles (Méthode B, ...)
  - *Tests (unitaires, d'intégration)*
  - . . .

- *Extensibilité et réutilisabilité :*
  - utilisation de formalismes de modélisation (comme UML par exemple)
    - permettront de décomposer le logiciel en entités quasi autonomes afin d'obtenir le logiciel le plus modulaire possible.
  - mettre en œuvre le principe *d'architecture logicielle*
  - utilisation de *patrons de conception*
  - documentation interne
  - . . .

II - Cycles de développement & place des tests

1 – Cycle en V
2 – Cycle en spirale & Test Driven

↪ Rappel


Spécification
Conception préliminaire
Conception détaillée
Codage
Tests unitaires
Tests d'intégration
Validation
Vérification/Validation
Retour si erreur/problème
Fonctionnement normal

↪ Tests unitaires 1/2

Définition :

- Étape du cycle de développement consistant à vérifier le bon fonctionnement d'une partie d'un logiciel ou d'une unité
  - classe, module, ...

Objectif :

- Le but du test unitaire et de s'assurer que la réalisation, i.e. l'implémentation, correspond complètement à sa spécification

Méthode :

- le test permet de définir :
  - un état de sortie à l'issue de l'exécution du code testé, fonction de l'état d'entrée
  - un résultat attendu d'après la spécification
- le résultat du test correspond à la comparaison entre la sortie obtenue et la sortie attendue.

Quand :

- Dans un cycle de développement classique,
  - les tests unitaires viennent après la réalisation de tout ou partie du code
  - l'écriture des tests unitaires est souvent considérée comme une étape secondaire ...

↪ Tests unitaires 2/2

Avantages :

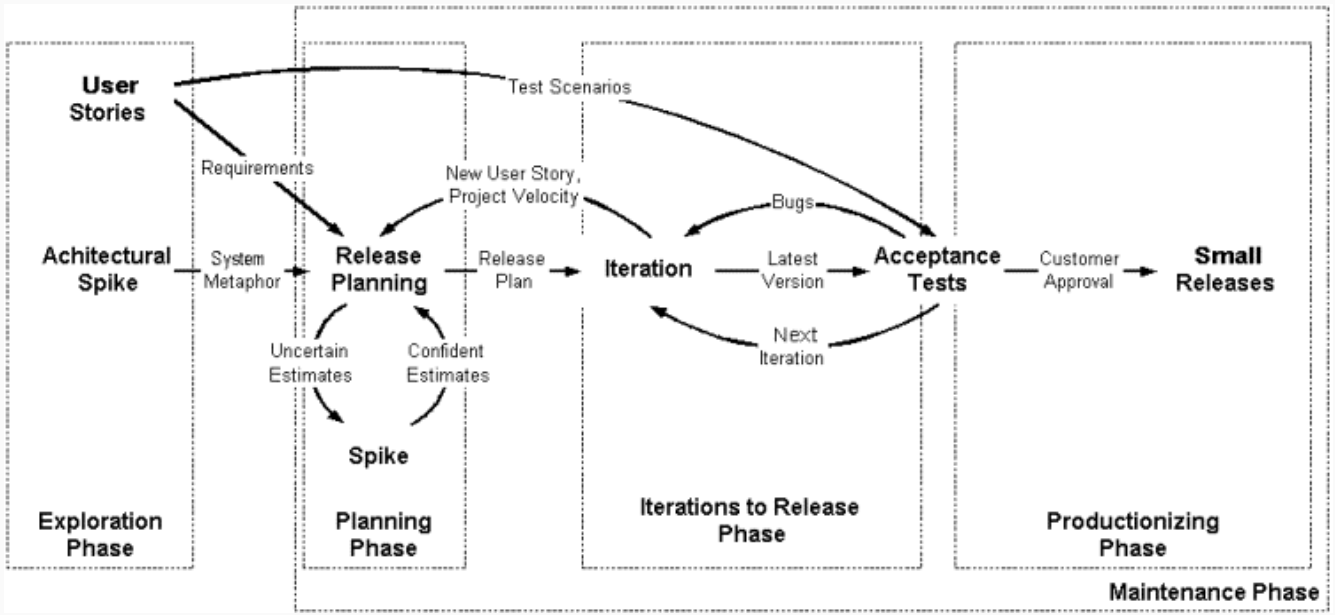
- Essentielle dans les applications critiques
- permet de mener en même temps des tests de couverture de code
  - vérification que lors du test l'ensemble du code de l'unité testée est exécuté
- assure la non-régression
  - à chaque modification du code du logiciel, l'ensemble de la batterie de tests doit être rejoué
  - permet ainsi de détecter très tôt l'introduction d'erreur(s) lors de la modification et donc la régression du code existant

L'outillage :

- Pourquoi des outils pour les tests :
  - éviter la pollution du code du logiciel par du code de test
  - diminuer le temps d'écriture des tests
  - proposer un environnement d'exécution et de gestion des tests
- Nombreux frameworks :
  - Il existe des frameworks de tests pour de nombreux langages
  - on utilise souvent le terme de xUnit ou x représente le langage visé par le framework
    - JUnit : Java
    - CUnit : C
    - NUnit : .NET
    - ...

↪ Définition

- Le développement dirigé par les tests est apparue avec la méthode d'Extreme Programming
  - l'objectif premier de cette méthode est de réduire les coûts dus aux changements en poussant à l'extrême des étapes déjà réalisées dans des méthodes classiques :
    - revue de code faite en permanence par un binôme
    - tests unitaires réalisés avant l'implémentation
    - conception tout au long du projet : c'est le refactoring
    - intégration des modifications au plus tôt
    - cycle très court
    - construction de "produit" partiel



↪ Fonctionnement

- Le cycle de réalisation d'une unité comporte 5 étapes
  1. Écrire le test
  2. Vérifier que le test échoue
     - le code n'existant pas, le test doit forcément échoué
  3. Écrire le code juste nécessaire (et pas plus) pour faire passer le test
  4. Vérifier que l'exécution du test passe
  5. Refactoriser le code
     - Amélioration du code sans changer les fonctionnalités

↪ Spécifications : Rappels

- Les spécifications doivent contenir :
  - la définition du domaine d'entrée
    - conditions pour chacun des paramètres
  - la définition du domaine de sortie
    - expression de ce que doit effectuer l'unité spécifiée
  - les éventuelles exceptions pouvant être levées
- Ce sont ces spécifications qui doivent être implémentées par la(les) méthode(s) de test