ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

04
Chapitre

Méthode agile : une première approche

1I2AC1 : Génie logiciel et Conception orientée objet

Régis Clouard, ENSICAEN - GREYC

« Je ne suis pas un grand programmeur.
Je suis juste un bon programmeur avec de bonnes habitudes. »
Kent Beck (créateur de la méthode X-Programming)

Objectif du chapitre

- Une première introduction à l’agilité dans le développement logiciel.
  - *Les méthodes de développement seront vues en détail au semestre 8.*
  - Ici, nous présentons une approche pragmatique utilisable pour les TP et les projets.
- À l’issue de ce chapitre, vous serez sensibilisé :
  - à l'importance d’un cycle de développement itératif et incrémental.
  - à la nécessité d'un dialogue permanent avec les futurs utilisateurs sur la base d’une version opérationnelle du futur logiciel.
  - à l’obligation de tester son code.
- Cela doit vous permettre de changer votre façon de développer en TP et en projet pour aborder plus sereinement le travail de conception et de programmation.

Plan du chapitre

1

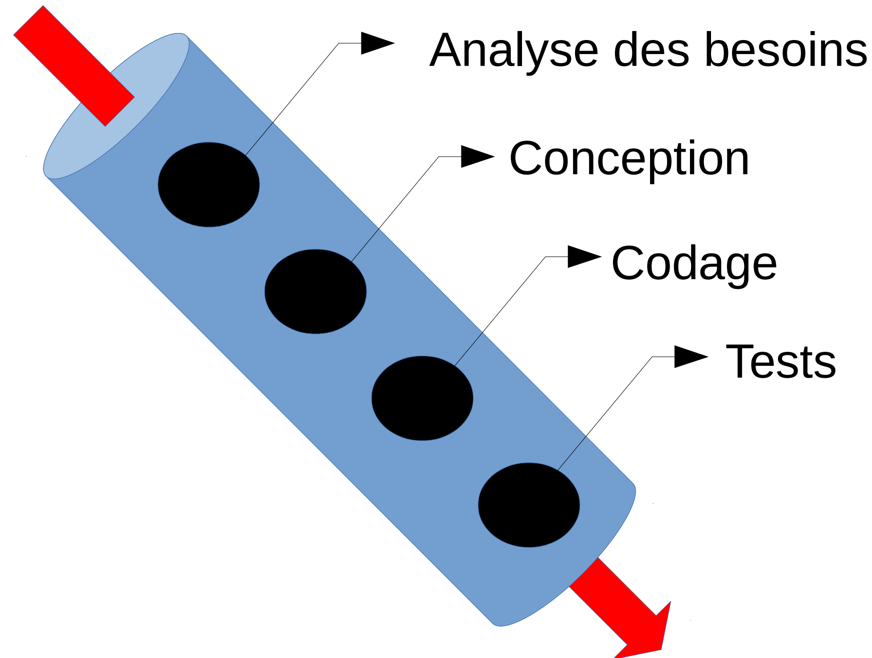
Mauvaises pratiques du développement

Développement en TP

- Pratique recherchée et parfois enseignée :
  1. Je code tous les algorithmes demandés dans le sujet.
  2. Je compile.
  3. J'exécute pour tester le programme.
- Ça ne marche pas.
  - Aie, ça ne compile pas !
  - Le compilateur génère un flot de messages d'erreur que j'ai du mal à comprendre.
  - Quand enfin ça compile, je teste le programme en vérifiant visuellement les sorties.
  - Si le programme ne fonctionne pas correctement, je truffe le code d'affichages intermédiaires pour identifier la cause du dysfonctionnement.
- Cette pratique porte un nom en génie logiciel : le cycle de développement en cascade.

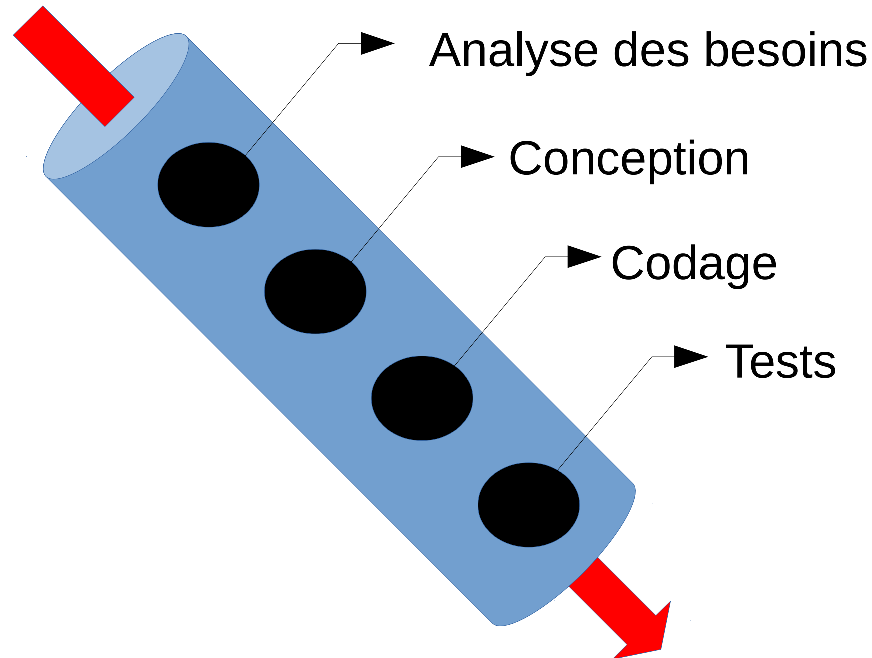
Développement en cascade

- Le cycle de développement en cascade a longtemps été enseigné et utilisé en entreprise.
  - Il est la cause de l'échec de nombreux projets professionnels.
  - Toutefois, il a une vertu pédagogique : il identifie les 4 étapes du développement.



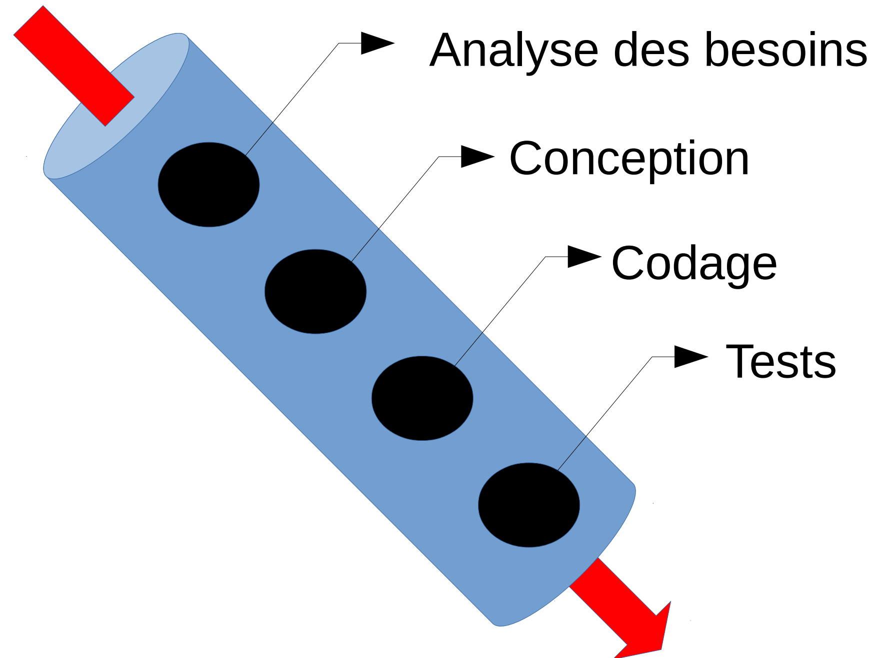
Cascade et effet tunnel

- Ne marche pas en pratique :
  - On se met d'accord avec le client à l'entrée du tunnel sur la liste des fonctionnalités à développer.
  - Et quand on ressort du tunnel, les besoins du client ont changé !



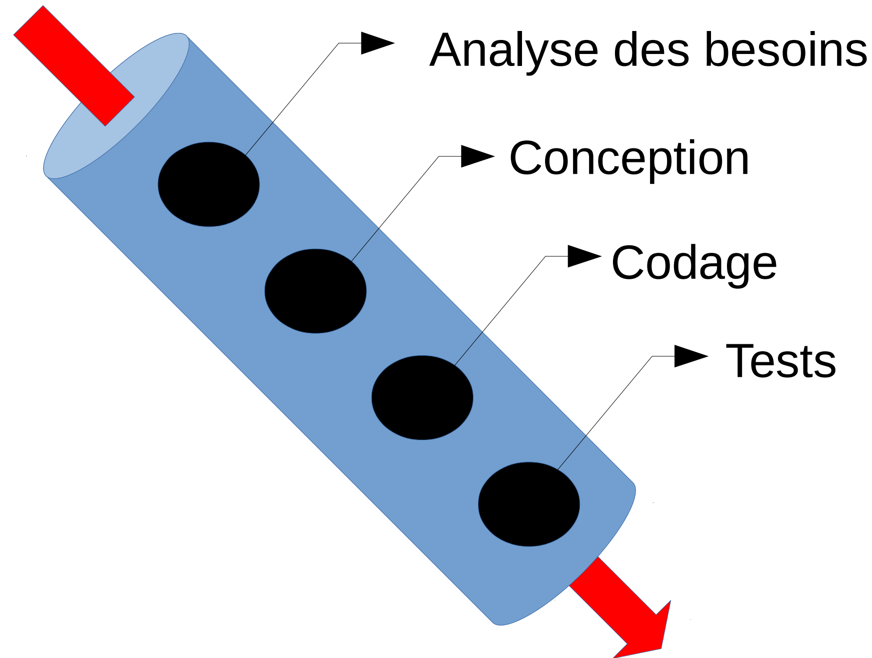
Conception puis codage

- Ne marche pas en pratique :
  - On peut très bien découvrir au moment du codage un problème qui remet en cause la conception.
  - Il faut recommencer, ce qui peut mettre le projet en péril.



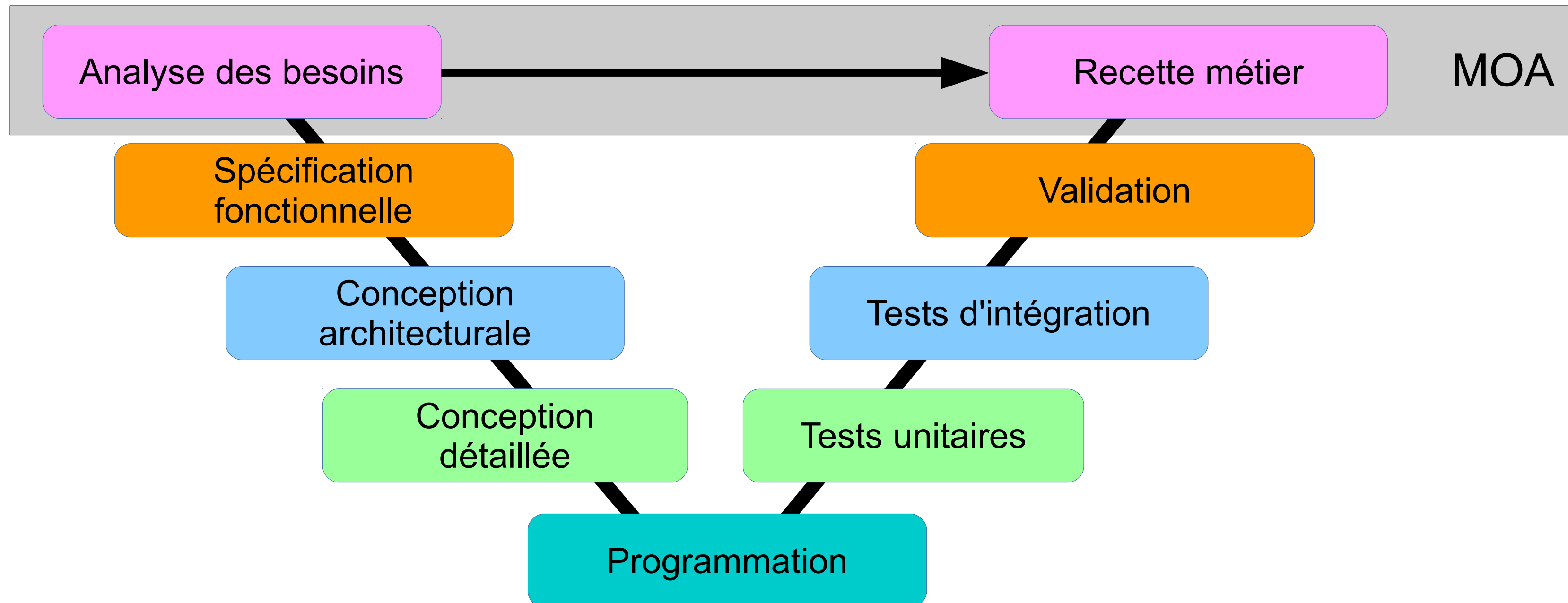
Test après le codage

- Très inefficace
  - Les bugs et erreurs doivent être détectés le plus tôt possible.
  - La définition des tests en fin est difficile parce que l'on a perdu les objectifs de ce que l'on doit tester.
  - Les tests sont en général court-circuités pour tenir les délais.



Une amélioration : le cycle en V

- L'apport du cycle en V se situe au niveau des tests.
  - Le meilleur moment pour définir les tests, c'est lors la spécification des éléments à tester.
- Mais globalement, l'amélioration reste modérée.
  - p. ex. du point de vue MOA, le cycle en V est toujours vu comme un tunnel.



Plan du chapitre

1
Mauvaises pratiques du développement

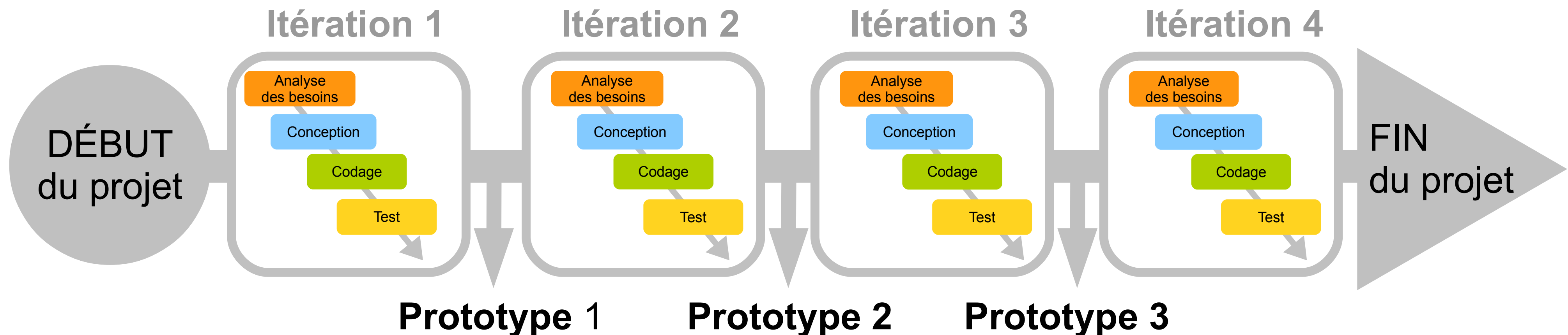
2
Bonnes pratiques du développement

Pratique actuelle : l'agilité

- Méthodes agiles (ici principalement *Extreme Programming*).
  - Le développement est centré sur le code.
  - Le logiciel est développé **itérativement** par **incréments**.
  - Il s'agit de produire rapidement des versions opérationnelles successives et de plus en plus complète du logiciel : des prototypes.
    - « Un intellectuel assis ira toujours moins loin qu'un con qui marche » - Michel Audiard.
  - L'intégration du travail des développeurs doit être faite en continu (quasi-quotidienne).

Cycle de développement itératif

- Itération
  - Courte durée (environ 2 semaines)
  - Reproduit un cycle en cascade complet.
  - Se termine par la livraison d'un prototype opérationnel.
  - Le prototype est soumis à l'évaluation du client.
  - Considérée comme une fin en soi (comme si le projet s'arrêtait à l'issue).
    - Pas de tâche s'étalant sur plusieurs itérations ; si nécessaire la découper.

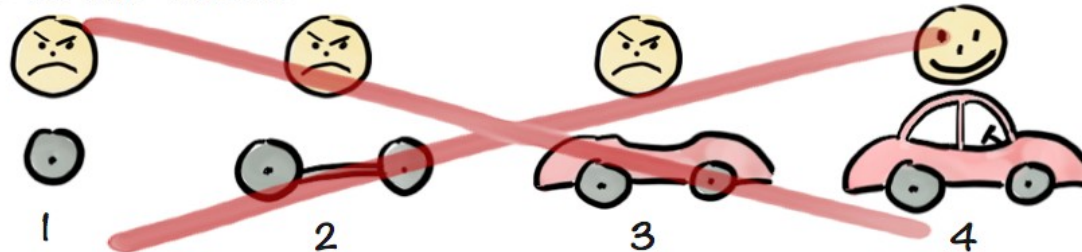


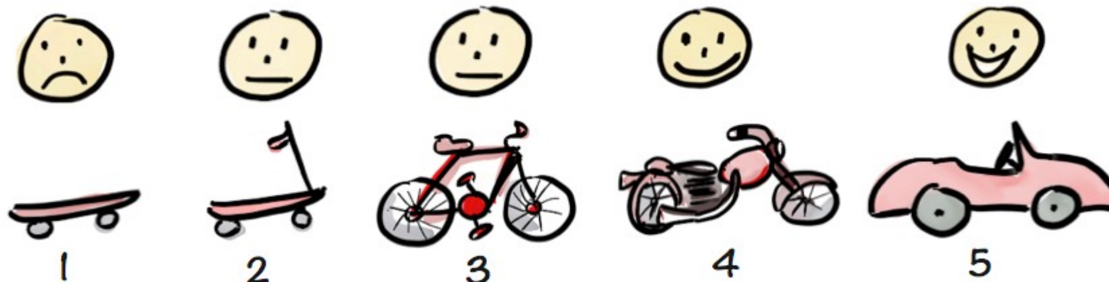
Cycle de développement incrémental

- Le logiciel est construit par **incrémentation**.
  - Le prototype n’est pas jetable.
  - Au contraire, il sera amélioré au cours des itérations suivantes s’il est jugé intéressant par le client.
  - Chaque prototype suivant ajoute de nouvelles fonctionnalités au prototype précédent.
- La règle des 80 – 20 (loi de Pareto)
  - 20 % des fonctionnalités couvrent 80 % des cas d’utilisation du futur logiciel et les 80 % des fonctionnalités restantes ne couvrent que 20 % des cas d’utilisation.
  - Il y a évidemment tout intérêt à commencer par développer les fonctionnalités des 20 %. Ce sont celles qui apportent le plus de valeur ajoutée immédiatement.

Prototype

- MVP : Minimum Viable Product
  - Un prototype avec juste assez de fonctionnalités pour satisfaire les premiers clients et fournir une rétroaction pour poursuivre le développement.





https://blog.crisp.se/2016/01/25/henrikkniberg/making-sense-of-mvp

Intégration continue

- Intégration quasi-quotidienne du travail des développeurs.
  - Il n’existe qu’une version courante du logiciel partagée par tous les développeurs.
  - Il y a une version de travail pour chaque développeur et une version courante commune à tous.
  - Le but est d’éviter l’effet big bang de l’intégration en fin d’itération.
  - La version commune doit toujours être gardée opérationnelle.
  - Chaque développeur en possède une copie sur laquelle il travaille.
- DevOps
  - La version doit être compilable et testée sans intervention humaine.
  - Peut aller jusqu’au déploiement automatique chez le client (déploiement continue).

Avantages

- Le cycle itératif
  - On avance à petits pas testés.
  - Si on rate un pas, on n’a raté qu’un pas, sans grosses conséquences.
  - Chaque pas est validé avec le client.
  - Le manque de temps conduit à un déficit de fonctionnalités et pas à un échec total du projet ni à un projet non testé.
  - On a toujours une version intermédiaire mais opérationnelle du logiciel à donner au client.
  - Réduit le temps de mise sur le marché (*time to market*).
- Incrémentation
  - Donne rapidement de la valeur au produit.
  - Le client voit une évolution rassurante et constante du produit.

Mise en pratique en TP et projet de C

- Démarche qui s'inspire du développement agile :
  - Mise en place de l'environnement de développement
    1. Construire le dossier.
       - Makefile, organisation des sous-dossiers, etc.
    2. Construire l'environnement de test.
       - Fichiers de test ou autres.
    3. Coder la fonction `main()` vide (prototype 0)
       - Compiler et exécuter pour vérifier l'environnement de développement.
  - Développement par itérations
    1. Coder une fonction de l'ensemble
    2. Faire un `main()` qui permet de la tester :
       - Compiler, Exécuter, Tester.

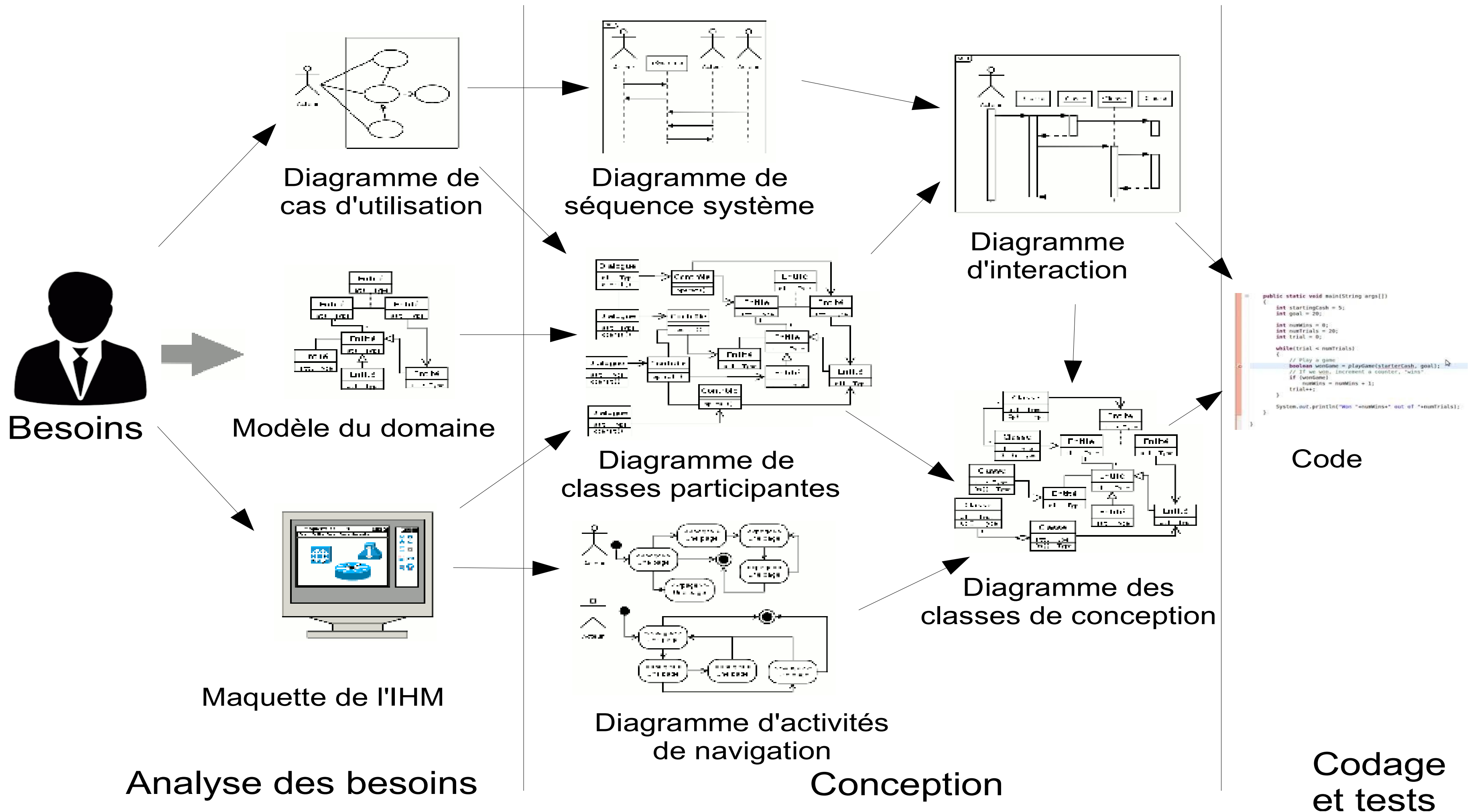
Plan du chapitre

1
Mauvaises pratiques du développement

2
Bonnes pratiques du développement

3
Développement et diagrammes UML

Utilisation des diagrammes UML lors d'une itération



Analyse des besoins

- Diagramme des cas d’utilisation.
  - Poser le problème en termes d’interactions entre les utilisateurs et le système.
    - Identifier les fonctionnalités à réaliser.
- Maquettes des écrans de l’IHM.
  - Les écrans des futures interactions.
    - Papier ou graphique (avec un créateur d’interface eg. *SceneBuilder*).
- Modèle du domaine.
  - Une première analyse du domaine à partir des concepts manipulés dans le domaine.

Conception

- Diagramme de classes
  - Parce qu'il est difficile de trouver les classes du futur logiciel :
    1) S’appuyer sur le « modèle du domaine ».
       - Il fournit les premières classes.
    2) Modéliser les aspects fonctionnels avec les diagrammes dynamiques (séquence, activités, états-transitions).
       - Ils permettent de comprendre comment cela peut fonctionner.
    3) Ajouter les « classes participantes » dans le diagramme de classes.
       - On obtient le diagramme de « classes de conception ».

Conception modulaire

- « Diviser pour régner »
  - Face à des problèmes complexes, rechercher la modularité afin d'obtenir un ensemble de modules plus simples et plus facilement gérables.
- En conception orientée objet, la notion de module se décline en :
  - **Méthode** : implémentation de services.
  - **Classe** : abstraction de données et de services.
  - **Paquet** : groupement de classes dans une seule collection.

Conception modulaire

- **La décomposition d'un système en modules est une tâche difficile.**
  - Pas de théorie.
- Or ces modules jouent un rôle important pour la flexibilité, la fiabilité et la robustesse du système.
- Plusieurs méthodologies à partir du cahier des charges.
  - Débat sur la meilleure approche.

Exemple 1 d'une méthodologie pour déterminer les classes

- Méthode 1 : Analyse de texte (Dennis, 2002)
  - Un nom commun → une **classe**
  - Un nom propre ou une référence directe→ un **objet**
  - Un nom collectif → une **classe**
  - Un adjectif → un **attribut**
  - Un verbe "faire" → une **méthode**
  - Un verbe "être"→ un **héritage** ou une **instanciation**
  - Un verbe "avoir" → une **association**
  - Un verbe transitif → une **méthode**
  - Un verbe intransitif → une **exception**
  - Une phrase verbale prédicative → une **méthode**
  - Un adverbe → un attribut d'une **méthode**

Exemple 2 d'une méthodologie pour déterminer les classes

- Méthode 2 : les « cartes CRC » **C**lasses, **R**esponsabilités et **C**ollaborations
  - 1 post-it.
  - Nom de la classe en haut.
  - Les responsabilités de la classe avec en face les collaborations pour assurer cette responsabilité.
    - On assigne à un objet *o* les responsabilités pour lesquelles *o* possède toutes les informations nécessaires pour remplir ces responsabilités (dans certains cas, il collabore avec d'autres objets pour cela).
  - Les cartes sont collées sur un tableau et forment ainsi l'architecture du logiciel.
- Attention !
  - Attributs : ne pas s'intéresser aux attributs (encapsulation).
  - Méthodes : ne s'intéresser qu'aux services (ie méthodes publiques).

Que retenir de ce chapitre ?

- Les méthodes agiles proposent de nouvelles façons d'aborder le développement logiciel.
  - Le développement suit un cycle itératif et incrémental.
    - Chaque itération est une mini-cascade qui enchaîne les étapes d'analyse des besoins, conception, codage et test.
    - Chaque itération doit se terminer par une version opérationnelle et testée du logiciel en cours.
    - Chaque incrément doit ajouter une valeur au prototype en cours : MVP.
    - Chaque itération est une fin en soi.
- Attention : les méthodes prédictives tel le cycle en V ne sont pas caduques, mais elle sont utiles dans des projets très contraints ou à très long terme.
  - Mais, aujourd'hui le cycle en V est itératif et incrémental