



Résumé du cours

1I2AC1 : Génie logiciel et Conception orientée objet

Régis Clouard, ENSICAEN - GREYC

Génie logiciel / Software engineering

2

- Le génie logiciel est une science de génie industriel qui étudie les **méthodes** de travail et les **bonnes pratiques** des ingénieurs pour développer des logiciels de grande taille (plusieurs années - hommes)
- L'ambition est de développer des logiciels :
 - Satisfaisant les attentes du client
 - Fiables
 - Performants
 - Respectant les délais et les coûts
 - À faible coût de maintenance

Génie logiciel aujourd'hui

- **Composants**

- 1 **Paradigme** : Conception orientée objet
- 2 **Formalisme** : UML
- 3 **Méthode** : Agilité

Paradigme : Conception par objet

4

- Vision duale du paradigme procédural
 - ▶ On s'intéresse aux données et plus aux procédures.
- **Objet** : entité animée capable de réaliser des services
- **Programme** : ensemble d'objets coopérant par échange de services
- **Programmation** : quels objets sont nécessaires à la réalisation du logiciel et comment s'échangent-ils des services ?

Paradigme : Conception par objet

5

- 6 concepts de base :

- 1/ **Modularité** → fonctions, classes, paquets, nœuds

- 2/ **Abstraction** → simplification du problème à l'essentiel.

- 3/ **Encapsulation** → protéger l'implémentation d'une classe du regard extérieur

- ▶ **Classe** : un ensemble de services (méthodes publiques).

- ▶ **Attribut** : n'existe qu'au moment de la programmation de la classe (nommé *donnée membre*) et parce que la réalisation d'un service le réclame.

- Un attribut n'existe pas en conception orientée objet.

Paradigme : Conception par objet

6

- 6 concepts de base (suite) :

- 4/ **Association** → relation entre classes

- ▶ Types : normal, agrégation, composition.

- ▶ NB : En conception, une association n'est pas un attribut.

- 5/ **Héritage** → construction d'une classe par généralisation / spécification d'une autre classe.

- 6/ **Polymorphisme** → dans une hiérarchie, le même service peut avoir un comportement différent selon l'objet réel qui est appelé (décidé au moment de l'exécution).

- programmation au niveau le plus abstrait.

Génie logiciel aujourd'hui

7

- 1 Paradigme : conception orientée objet
- 2 **Formalisme : UML**
- 3 Méthode : Agilité

Formalisme : UML

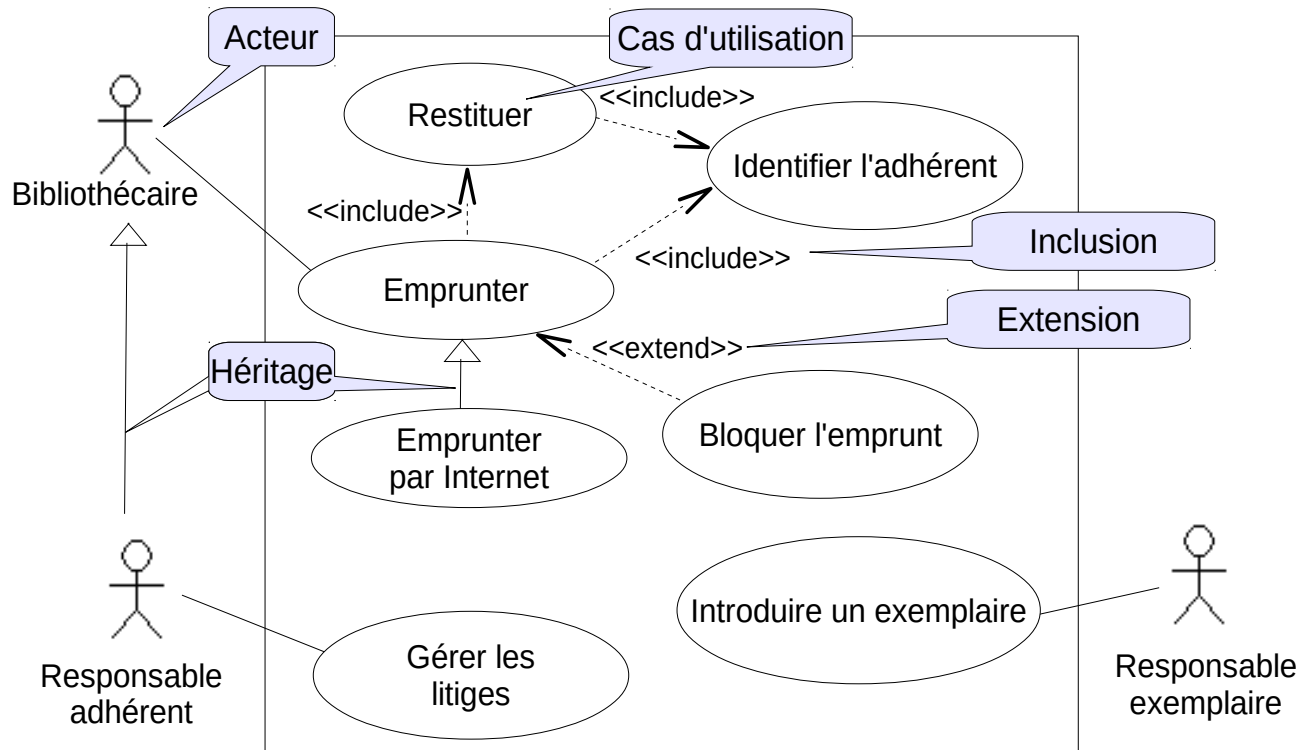
- Langage qui permet de parler de la conception d'un logiciel :
 - **Haut niveau** : plus abstrait que l'algorithmique. Il permet d'exprimer les besoins et les solutions.
 - **Diagrammatique** : peu de syntaxe.
 - **Normalisé** : partagé par tous les développeurs du monde.
 - **Translangage** : y compris les langages non orientés objet (p. ex. langage C).

Formalisme : UML

- 6 diagrammes principaux :
 - 1) **Vision statique** : trouver les classes et leur organisation
 - 1) Cas d'utilisation
 - 2) Classe
 - 3) Paquet
 - 2) **Vision dynamique** : trouver les méthodes et leur communication
 - 1) Activité
 - 2) Séquence
 - 3) États-transitions

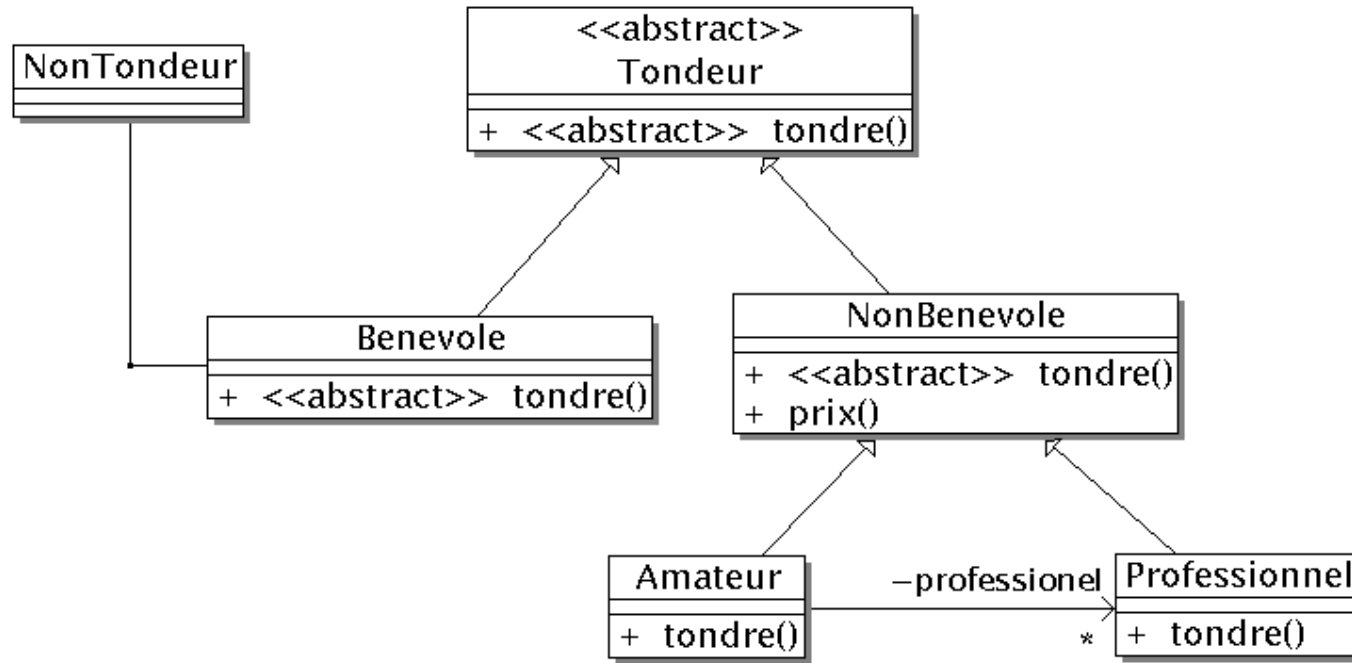
1- Diagramme des cas d'utilisation

- Définir le problème par les interactions entre les acteurs extérieurs et le système à développer.



2- Diagramme de classes

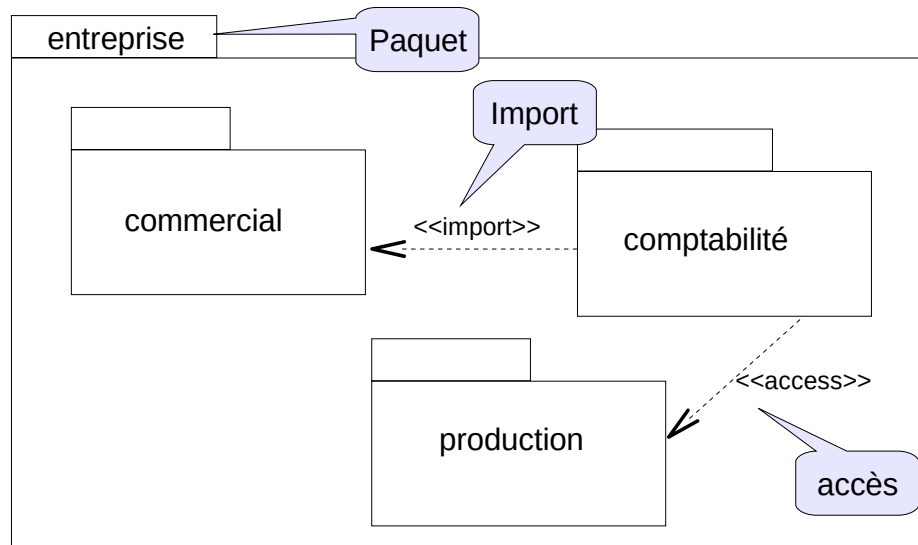
- **Vision statique structurelle du logiciel** → modéliser les classes qui seraient nécessaires à la réalisation du logiciel (commencer par les classes métier - classes du domaine)



- NB : Ne pas mettre les attributs → rompt l'encapsulation

3- Diagramme de paquets

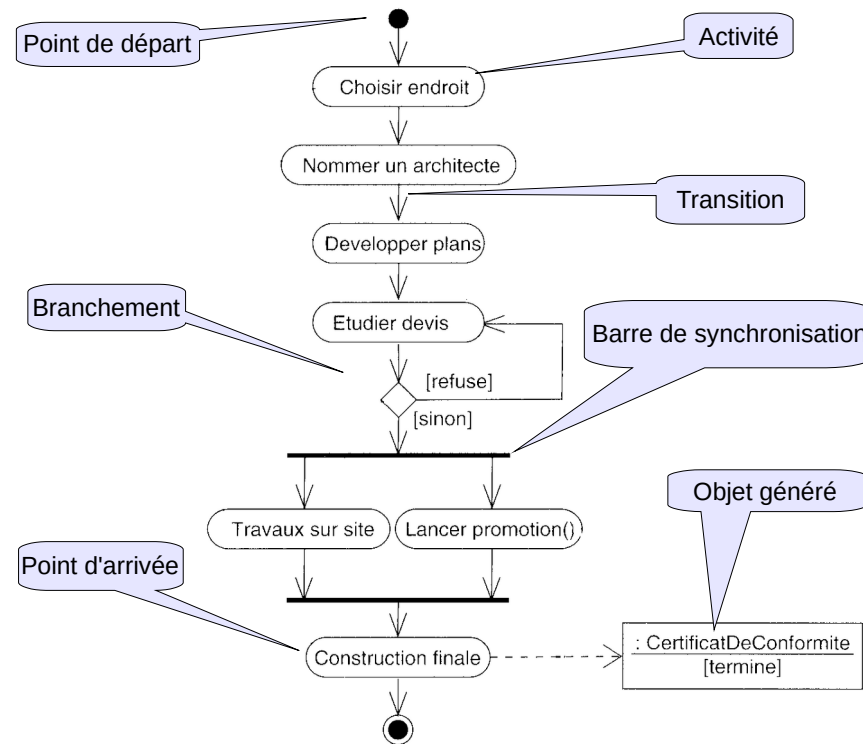
- **Vision statique organisationnelle du logiciel** → importance pour la vision et l'organisation du développement notamment en équipe.



- Les paquets en Java : **fr.ensicaen.ecole.projet**

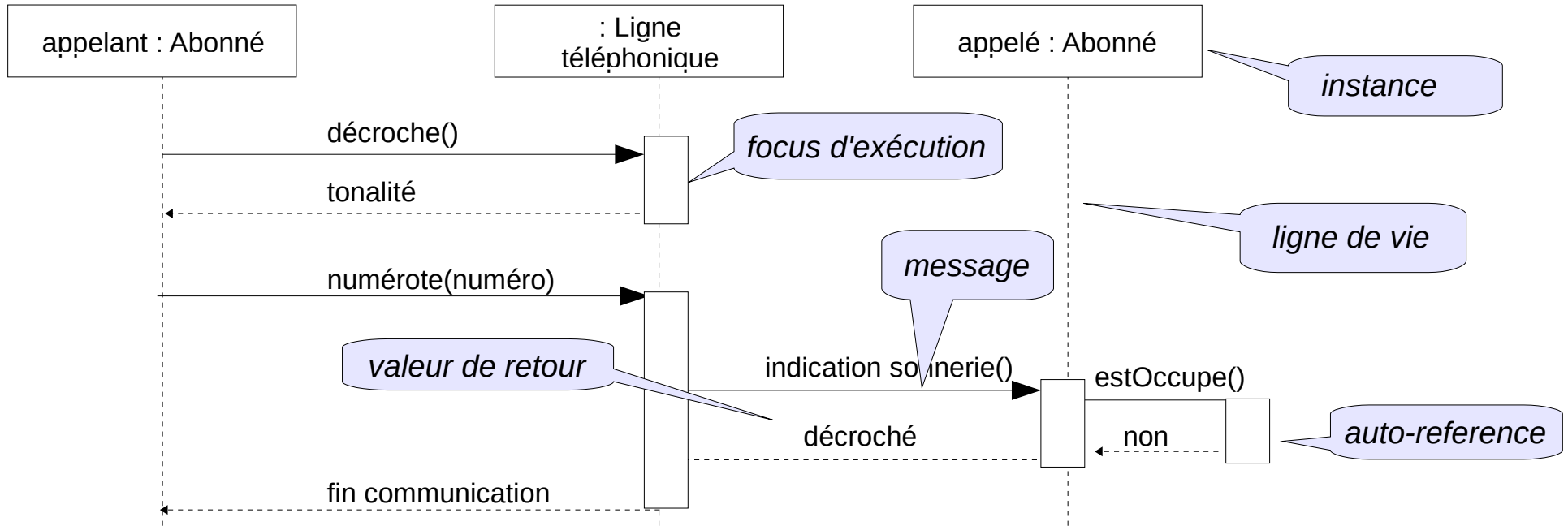
4- Diagramme d'activités

- **Vision dynamique** → décrire les activités liées à la réalisation d'un cas d'utilisation ou d'un algorithme.
 - NB : ne fait pas intervenir les classes.



5- Diagramme de séquence

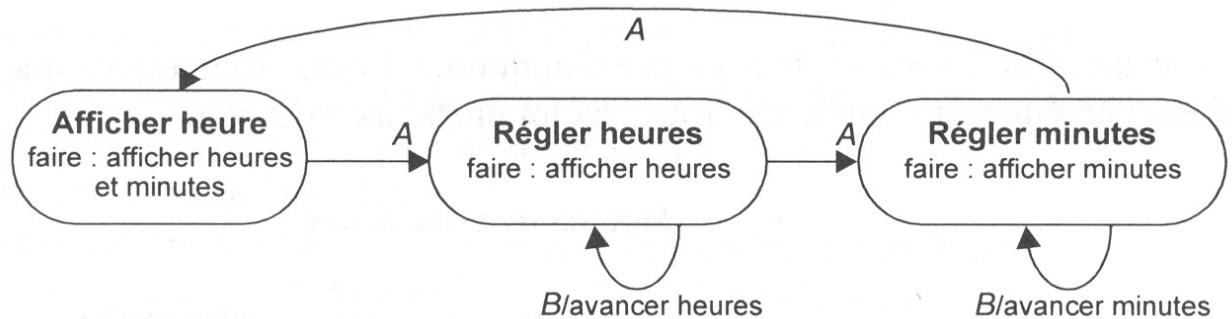
- **Vision dynamique** → identifier l'échange de services entre les classes nécessaires à la résolution d'un cas d'utilisation.



6- Diagramme d'états-transitions

- **Vision dynamique liée à une classe** → modéliser les états que peuvent prendre une classe et qui induisent une variation sur la réalisation de ses services
 - Aide à l'écriture du code des méthodes dont le comportement dépend de l'état de l'objet

Montre
+ appuyerBoutonA() + appuyerBoutonB()



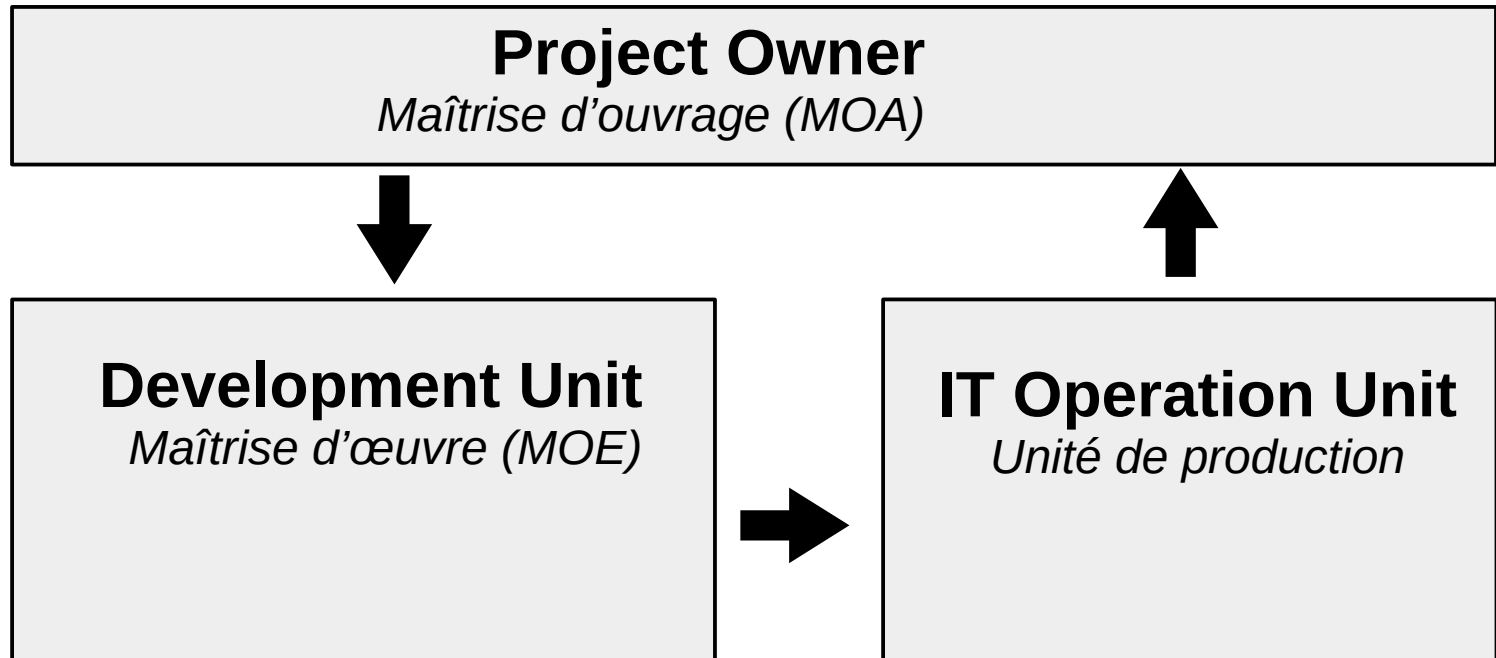
Génie logiciel aujourd'hui

16

- 1 Paradigme : conception orientée objet
- 2 Formalisme : UML
- 3 **Méthode : Agilité**

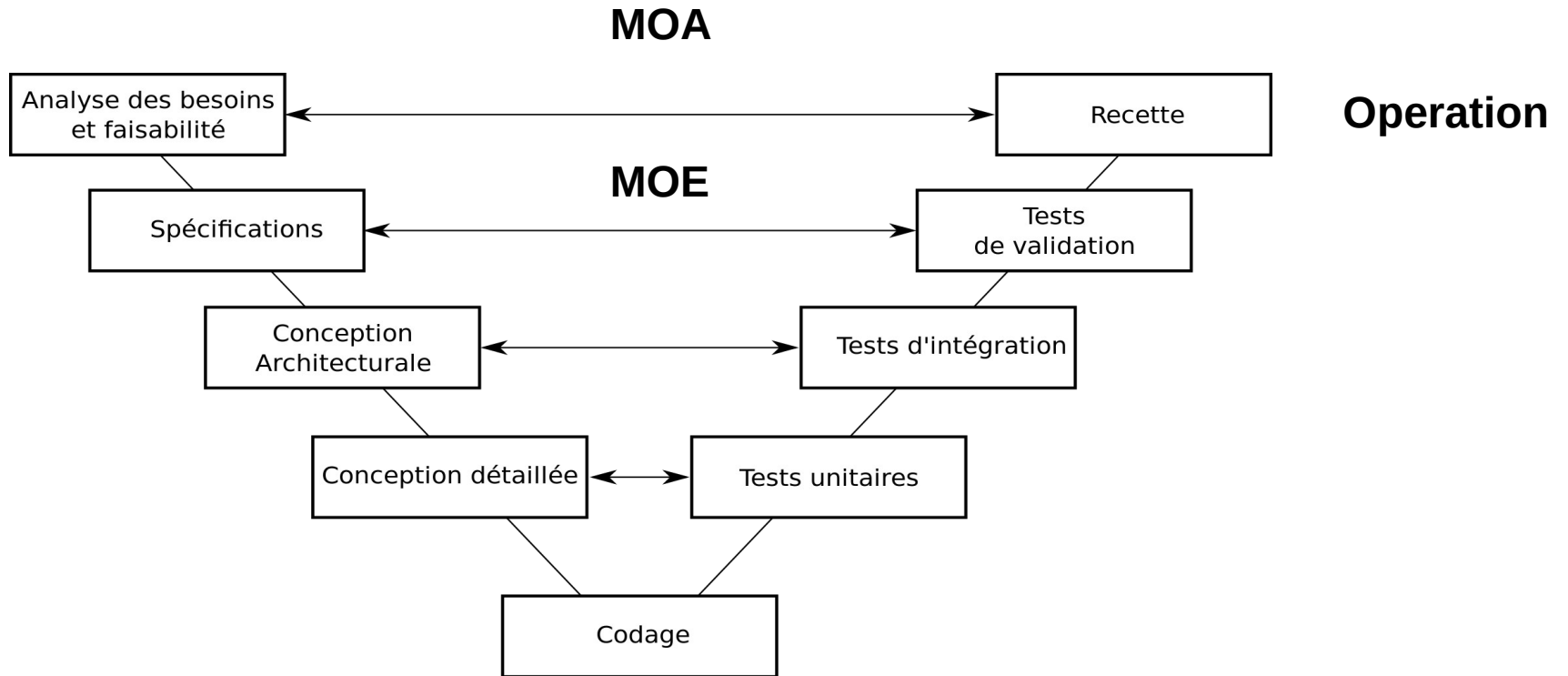
Écosystème du développement logiciel

17



Méthodes Classiques (prédictives) : Cycle en V

18

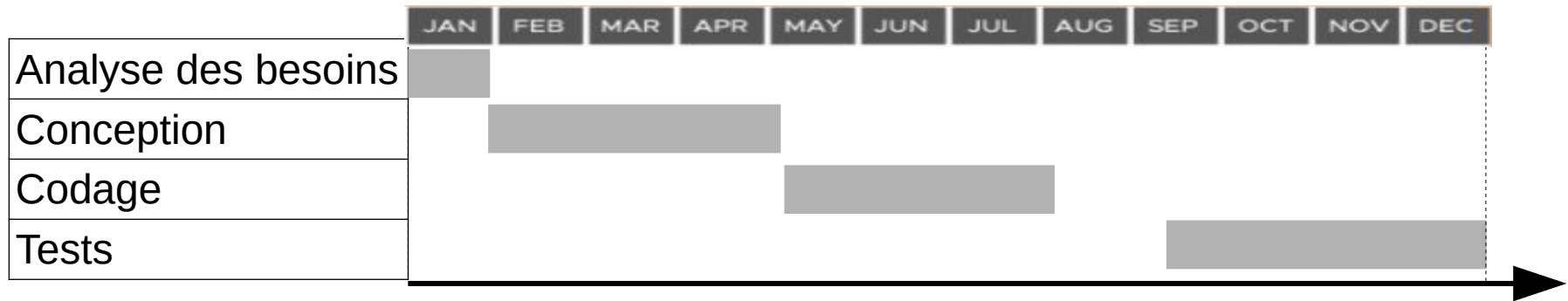


Méthodes agiles (adaptatives) : cycle itératif

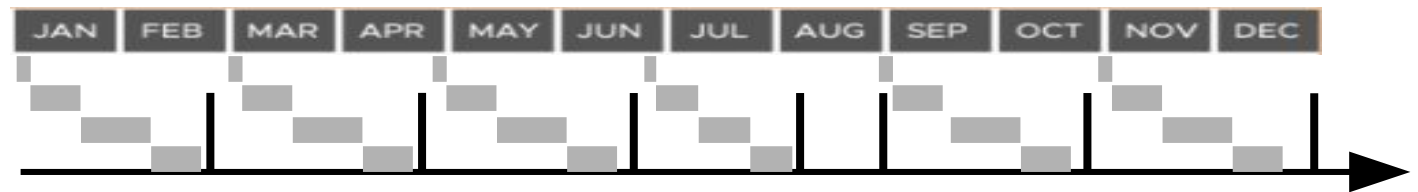
19

■ Changement de perspective

- Cycle en V : Planifier les activités de développement sur toute la durée du projet



- Cycle itératif : Identifier une liste de tâche réalisables en 2 semaines



Qualité logicielle et assurance qualité

20

- Vision externe
 - L'utilisateur est un partenaire du projet
 - Démonstration du produit exécutable (MVP) toutes les fins d'itération
 - Ne construire le contenu de l'itération suivante qu'après la rétrospective de prototype réalisée avec l'utilisateur.
- Point de vue agilité : artisanat du logiciel
 - Qualité du code
 - Tests programmés
 - Auto-documentation

XP : eXtreme Programming

■ Développement

- Centré sur le code.
- Rendre les commentaires de code et la documentation superflue
 - ▶ Code-auto-documenté
 - ▶ Si besoin de documentation utiliser la rétro-ingénierie
- Centré sur les besoins des utilisateurs

■ Tests automatiques

- Principalement tests unitaires
- Limiter les bugs à partir des cas de test liés au contrat de ce que l'on teste.
- Installer des gardes-fous pour se prémunir contre la régression.
 - ▶ Éviter que les mêmes bugs se reproduisent.

XP : eXtreme Programming

- Pratique du développement classique
 - 1) *Quick and dirty*
 - 2) *Test*
 - 3) *Refactoring*
- Développement dirigé par les test : TDD
 - 1) *Test*
 - 2) *Quick and dirty*
 - 3) *Refactoring*
- Développement des tests
 - TDD : quand la fonctionnalité est claire.
 - Classique : quand la fonctionnalité est à découvrir.

Examen

-
- Une feuille A4 manuscrite recto verso
 - Partie cours / partie exercices

Artisan du code

- Être un artisan du code, c'est :
 - Maîtriser son environnement de développement (IDE + outils)
 - ▶ Katas.
 - Amour du code propre.
 - Accepter les critiques sur son code dans le but de s'améliorer.
 - Partager son savoir.
 - Chercher à rendre la documentation et les commentaires superflus dans ses développements.