

Ingénierie du Logiciel

Master 1 Informatique – 4I502

Cours 1 : Introduction Génie Logiciel

Yann Thierry-Mieg
Yann.Thierry-Mieg@lip6.fr

Structure de l'UE et Organisation

Plan des Séances

Objectif : Méthode de développement, Analyse, Conception Architecturale

1. Introduction : organisation du développement logiciel

Arc 1 : QUOI ? La phase d'analyse.

2. Analyse I : cas d'utilisation, classes métier

3. Analyse II : fiches détaillées

4. Analyse III : séquences, tests de validation, bilan analyse

Examen Réparti 1 : Analyse d'un cahier des charges.

Arc 2 : COMMENT ? Conception

5. Conception Architecturale : transition, architecture, composants, séquences

6. Conception Détaillée : plateformes à composant, réalisation en J2SE

7. Tests : composants bouchons, configurations de test, test unitaire, bilan cycle V

Arc 3. D'autres approches

8. Agilité : pratiques agiles, SCRUM, XP...

9. Ingénierie des modèles : méta-méta-modélisation, Domain Specific Language

10. Bilan de l'UE, design pattern architectural, ouverture.

Examen Réparti 2 : Conception orienté composant

Références

- Hypothèse / pré-requis :
 - Niveau intermédiaire en Java ou autre langage O-O
 - Des bases en UML, en Design Pattern peuvent être utiles
- Page web : <https://pages.lip6.fr/Yann.Thierry-Mieg/IL/>
 - Slides du cours, Enoncés TD et TME
 - Fiches de cours : non redondant
 - Vidéos de l'amphi et d'un groupe de TD.
 - Exemple/Guide pour le projet de TME : Sudoku
 - **Annales** : une ressource importante pour préparer l'examen.
- Références biblio :
 - Ian Sommerville, « Software Engineering »
 - Gamma, Helm, Vlissides, Johnson, « Design Patterns »
 - Freeman&Freeman, « Head First in Design Patterns »
 - Larman, « Applying UML and Patterns »
 - Roques, « UML par la pratique » (ou d'autres du même auteur)
 - Se choisir des livres en bibliothèque et les **lire**

Organisation

- 2h cours, 2h TD, 2h TME par semaine
- Examen réparti 1 (novembre) sur feuille
- Examen réparti 2 (janvier) sur feuille
- Répartition : 10% note de TME, 40% Exam 1, 50% Exam 2.
- En TD :
 - Appliquer les étapes de la démarche avec un enseignant sur un Cahier des Charges (CdC)
- En TME :
 - Equipes de 6 à 8 étudiants (4 à 5 équipes par groupe de TD)
 - Appliquer la démarche en équipe sur un CdC :
 - LDVH le Livre Dont **Vous** êtes le Héros

Gestion des Equipes

- Groupes de 6 à 8 étudiants
 - Problème : s'organiser pour utiliser cette ressource, dynamique de groupe
 - Jeu de rôle : SSII répondant à un appel d'offre
- Communication
 - Support écrit avec traces consultables par tous les membres du groupe : discord, google groups, whatsapp...
 - Git pour les sources
 - Decoupage et affectation de taches
 - Réunions
- Outils
 - IDE Java+UML : IBM Rational Software Architect Designer (RSAD)
 - Disponible à la ptpi + à télécharger et active avec la clé sur le site

Gestion d'équipe

- Idéalement prendre des personnes avec des affinités
 - Mais l'exercice correctement appliqué doit permettre de collaborer avec tout le monde
 - Rester poli et professionnel dans les échanges
- Réunions efficaces
 - Time-box : au bout de 15 minutes c'est fini. 4*2 binomes produisent plus que 8 ensemble en général.
 - Ordre du jour : par écrit avant la réunion, qu'on y ait réfléchi un minimum. Typiquement, tour de table, avancement sur mes tâches, si besoin synchronisation, découpage du travail suivant, affectation.
 - Rapport : décisions prises lors de la réunion, principalement définition et affectation de tâches
- Outils importants
 - Trace écrites consultables (par tous) préférables, éviter les échanges perso autour du projet
 - Organisation et découpage en tâches, des outils peuvent aider e.g. Trello
 - Pas de contrainte sur l'organisation, mais si un membre veut faire CP de façon consensuelle cela peut bien fonctionner

Outils recommandés

- Un gestionnaire de versions de code :
 - Github, Gitlab...
- Un outil de communication avec traces écrites de préférence
 - Discord (en texte au moins quelques lignes de CR si on fait en vocal), google groups, ...
- Un outil pour rédiger les livrables (slides, rapports, ...)
 - Google docs
- Notre outil UML
 - Rational Software Architect
- Un outil pour organiser le découpage et l'affectation des tâches
 - Trello est spécialisé sur ce créneau, mais google groups ou même discord peuvent faire l'affaire. On peut aussi utiliser le github/gitlab en détournant un peu les « issues ».
- Un outil de collaboration partage d'écran etc...
 - Discord, zoom,...

Projet de TME : évaluation

Deux soutenances :

- Slides/powerpoint + présentation en groupe
- Séance 5 : bilan de l'analyse
- Séance 10 : conception, bilan du projet, démo

Evaluation :

- Le groupe a bien appliqué la démarche proposée
- Qualité et exhaustivité des diagrammes et artefacts produits
- Qualité de la présentation et cohésion de l'équipe

Le projet n'est pas centré sur la réalisation/codage:

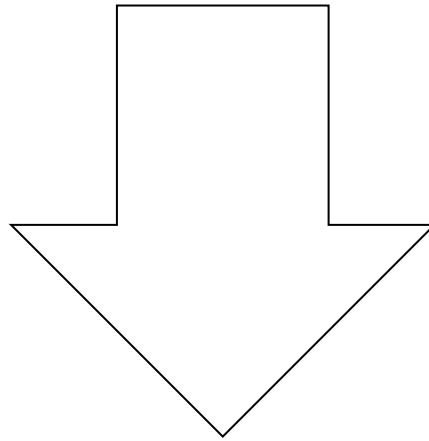
- Pas une UE de programmation
- Si le projet permet une démonstration d'un prototype réalisant un sous-ensemble réduit des fonctionnalités, mais conçu et construit en appliquant la démarche, on est satisfait.

I. Méthode de Développement

Du besoin informel à la solution

- Besoin informel :

« Je souhaite un véhicule sans chauffeur qui me permette de me déplacer sans permis de conduire »



- Solution :



Google Car

Le besoin d'une méthode

- « Petit » programme, isolé, non critique, jetable, durée de développement limité
 - Exo de TME en binome, page ou formulaire web, ...
 - 1. Ouvrir IDE. 2. Coder.
- Large echelle, communicant, critique, temps reel, embarqué réutilisable, coût de développement en dizaines d'homme/an...
 - Nombreuses sources de complexité
 - Défis techniques
 - Défis d'organization

Complexité technique

Solution complexe :

- Interfaces composants physique embarqué
- Temps réel
- Connecté / accès données carte
- Synchronisation cameras/direction
- Interface homme machine
- Langages et technologies/framework techniques multiples
- Prise de décision / système critique
- Système de systèmes : le système est un assemblage complexe de sous-systèmes hétérogènes...

La solution ne se limite pas au code, considérer l'ensemble des documents et artefacts nécessaires pour le développement d'un tel système.

Dimension Humaine

Intervenants Divers => visions diverses

- Chef de Projet : avancement projet, affectation ressources

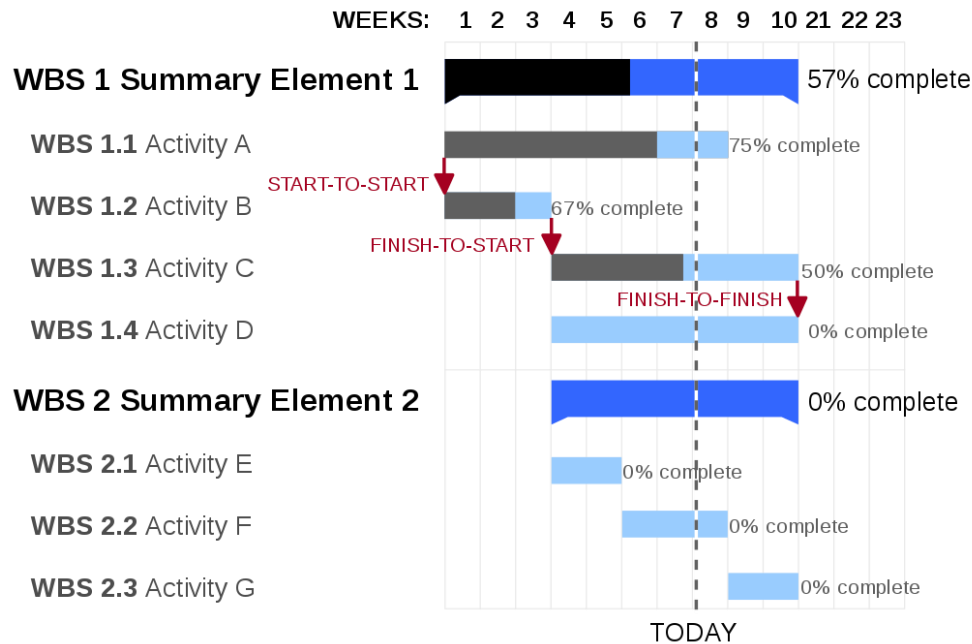
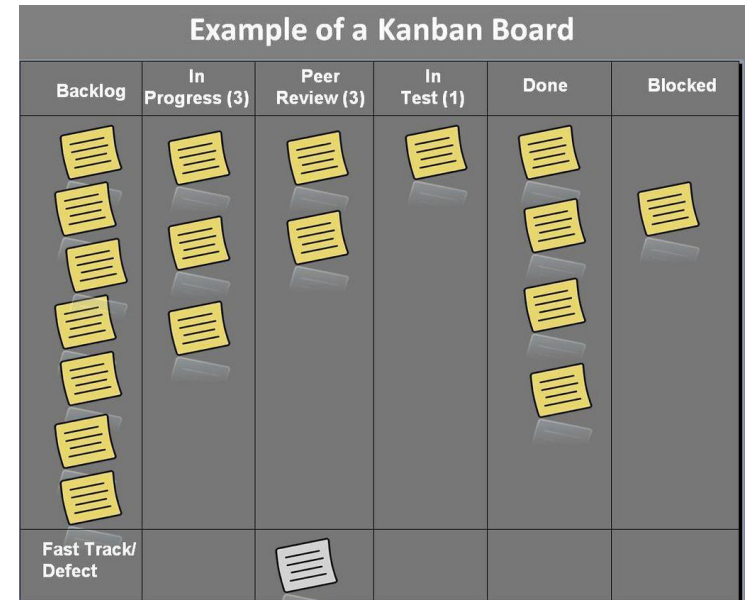
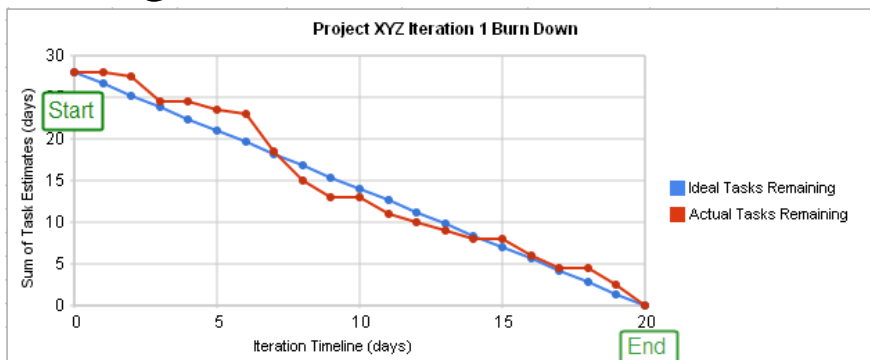


Diagramme de Gantt

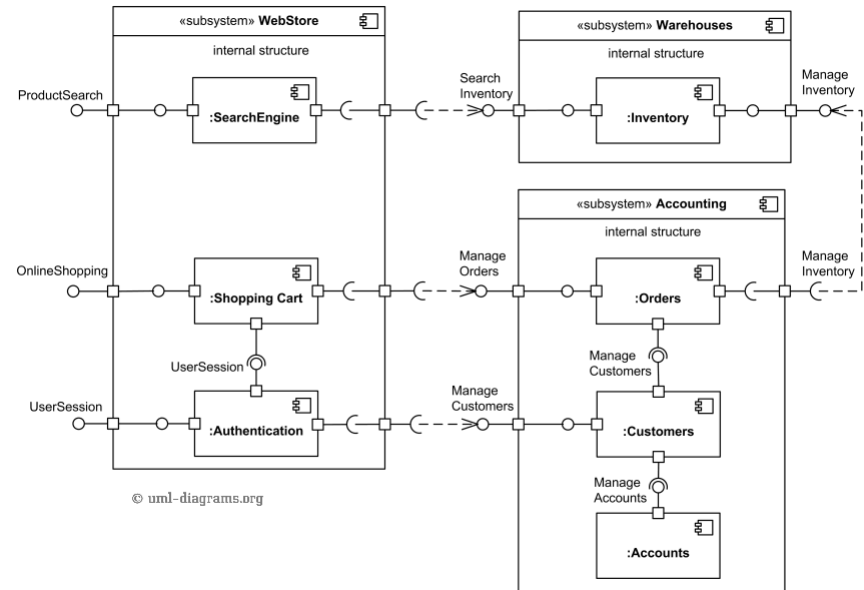
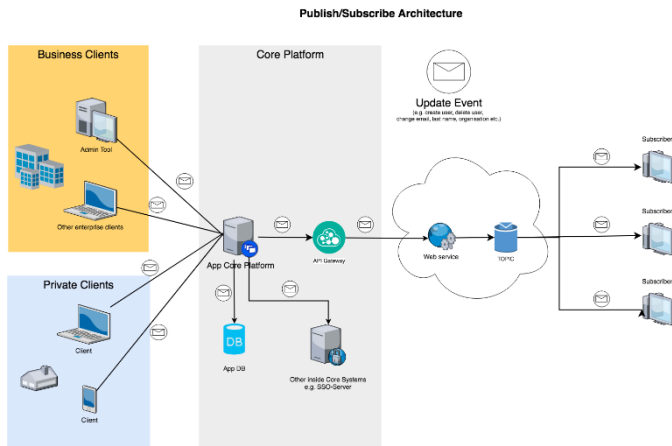


Kanban

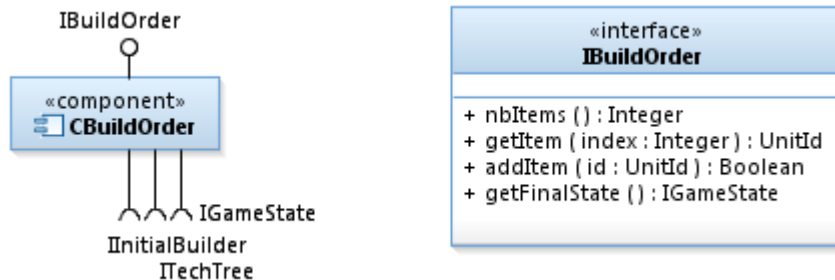
Burndown Chart

Dimension Humaine

- Architecte : architecture logicielle et matérielle, composants, interactions

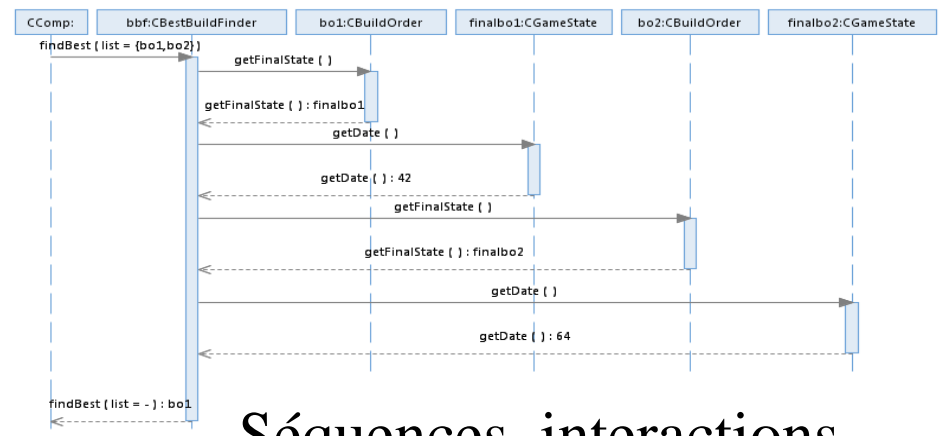


Vision architecturale



API, Interfaces

Composants, Instanciation



Séquences, interactions

Dimension Humaine

Développeur : réaliser le code

UC02 : Choisir Grille

Description : Permet au joueur de charger une grille existante, ou à partir d'une photo.

Acteurs : Joueur, (secondaire) Reconnaissance Formes

Hypothèse : L'application est démarrée

Préconditions : aucune

Scénario Nominal :

1. Le joueur choisit l'option « charger Fichier »
2. Le système affiche un sélecteur de fichier, positionné dans le répertoire « grilles » de l'application.
3. Le joueur choisit un fichier extension .sdk
4. Le système charge le contenu du fichier dans une grille.
5. Le système affiche la grille

Post-conditions :

Le système a chargé et affiche la grille choisie.

Alternatives :

A1 : Charger Image

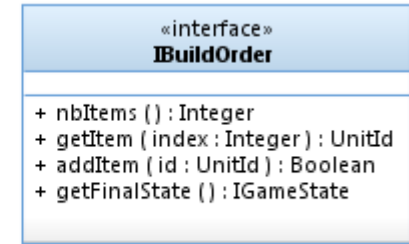
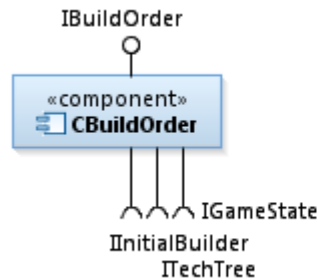
En SN1, le Joueur choisit plutôt de charger une photo.

- A1.1 le système affiche un sélecteur de fichier, positionné dans le répertoire « grilles » de l'application.
 A1.2 Le joueur choisit un fichier extension .png ou .jpg
 A1.3 Le système demande à l'acteur Reconnaissance Formes de lire la grille
 A1.4 Le système charge le résultat dans une grille
 A1.5 retour en SN5

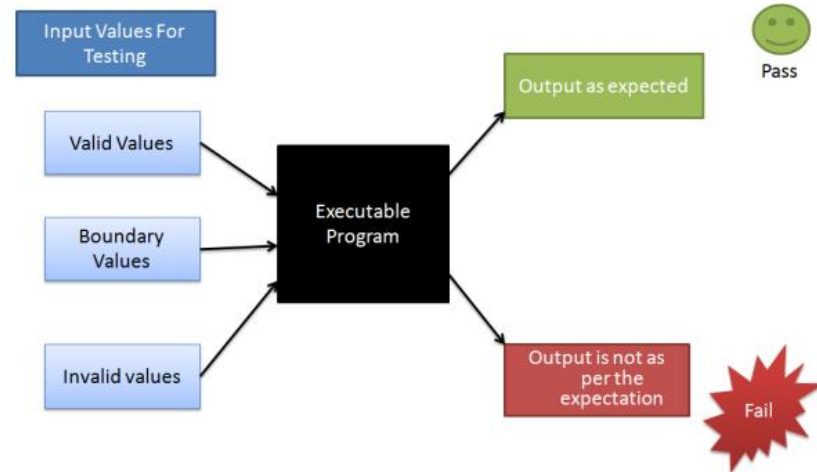
Exceptions

E1 : Fichier Invalide

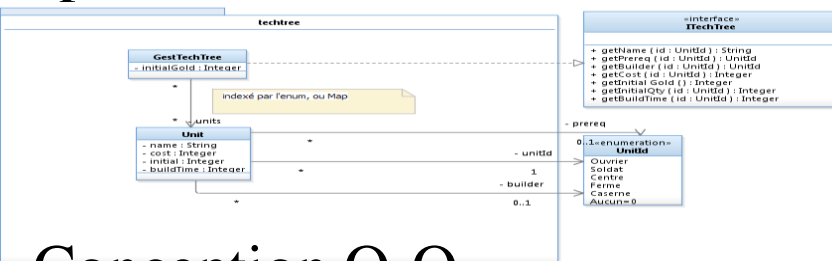
En SN4 ou A1.3, si la lecture du contenu du fichier échoue, le système notifie l'utilisateur et l'interaction est terminée.



API, Interfaces



Spec. Fonctionnelle



Conception O-O

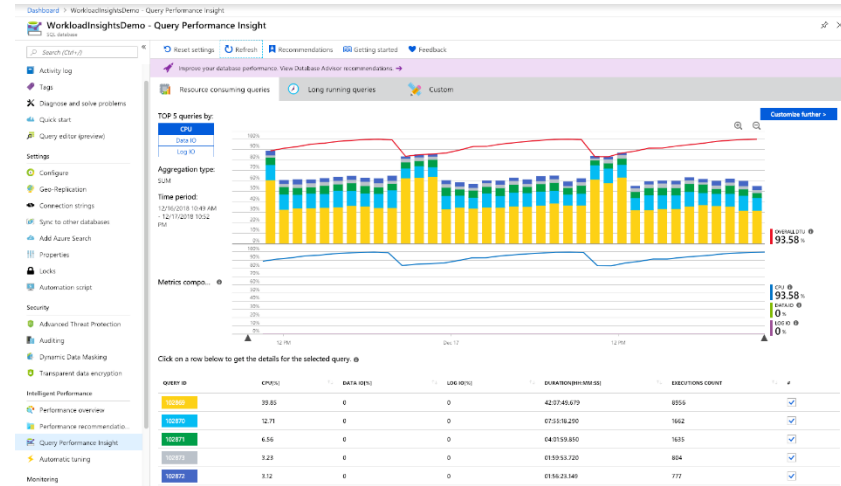
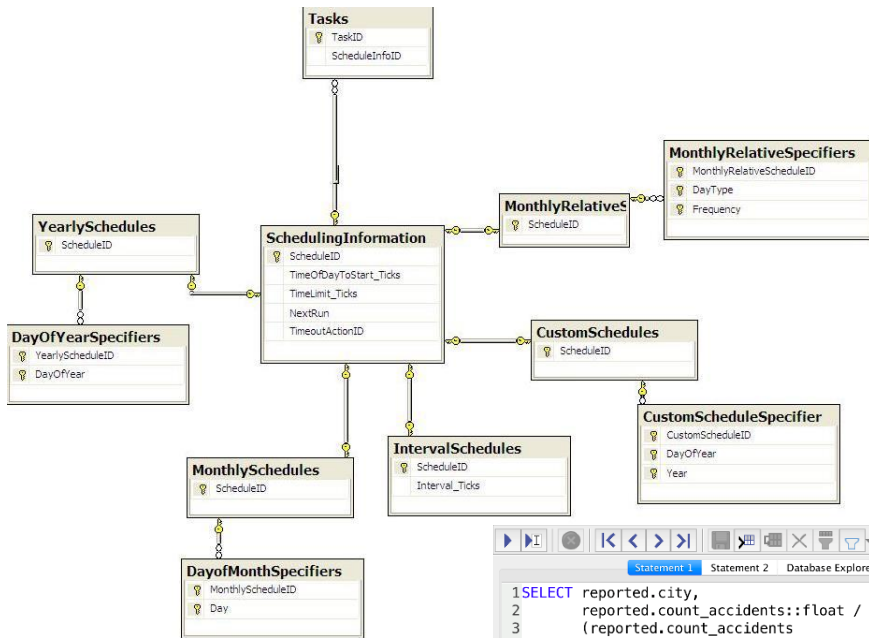
Tests



Code (!)

Dimension Humaine

Administrateur Base de Donnée : entretenir et gérer les données, optimiser les accès



Requêtes et volumes, perfs

```
Statement 1 Statement 2 Database Explorer 3
1 SELECT reported.city,
2   reported.count_accidents::float /
3   (reported.count_accidents
4    + not_reported.count_accidents)::float AS ratio
5 FROM
6   (SELECT city, COUNT(*) AS count_accidents
7    FROM accidents
8    WHERE reported = 'true'
9    GROUP BY city) as reported
10 JOIN
11   (SELECT city, COUNT(*) AS count_accidents
12    FROM accidents
13    WHERE reported = 'false'
14    GROUP BY city) as not_reported
15 ON reported.city = not_reported.city
16 ORDER BY ratio DESC
17 LIMIT 3;
```

city	ratio
ROSTOCK	0.67
MAGDEBURG	0.67
BREMEN	0.55

Requêtes SQL

Dimension Humaine

Intervenants Divers => visions diverses

- Testeur : définir et exécuter les tests
 - Specifications fonctionnelle, tests, api
- Client/Investisseur : vue d'ensemble, définition du besoin
 - use case, tests de validation, maquettes/protos
- Designer : ergonomie, graphismes
 - charte graphique, maquettes, use case
- Responsable qualité : suivi et mesure de la qualité
 - Normes qualité, métriques, instruments

...

La **DIVERSITE** = multitude de points de vue, dédiés à une utilisation particulière

On ne peut/doit pas chercher à tout fusionner

Le code n'est qu'un artefact parmi d'autres, peu abstrait et inadapté pour beaucoup d'intervenants.

La méthode de développement : objectifs

La méthode a pour but d'améliorer :

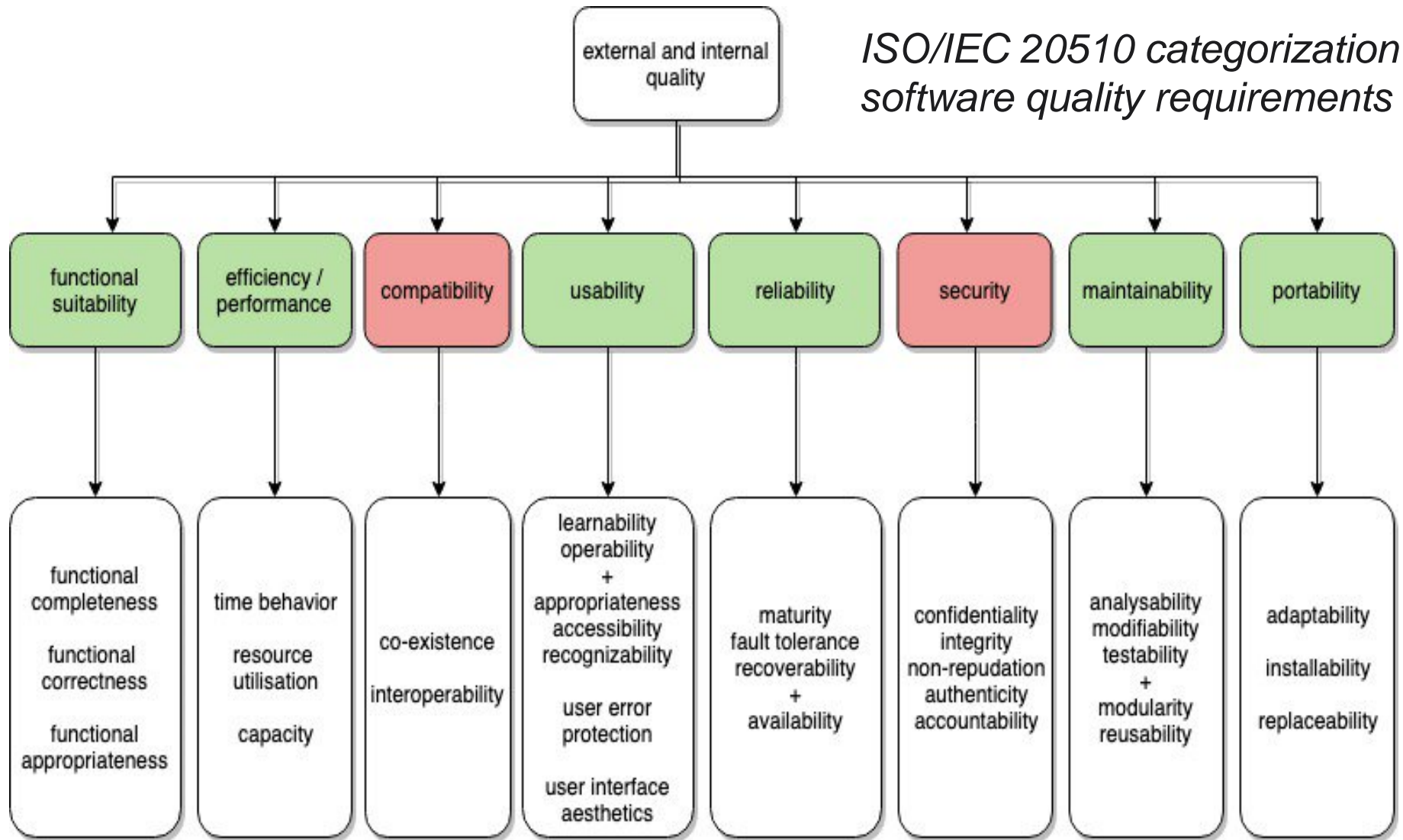
- La productivité des équipes :
 - Time to market, délais, coûts
- La qualité des logiciels produits
 - Portabilité, robustesse, sécurité...

Souvent une opposition entre ces deux critères

Quelles qualités doit posséder le logiciel ?

Qualités du logiciel

ISO/IEC 20510 categorization of software quality requirements



Source: ISO20500.com

Cf. aussi ISO9126

Qualités du logiciel

De nombreuses qualités souhaitables selon le domaine d'application et les contraintes

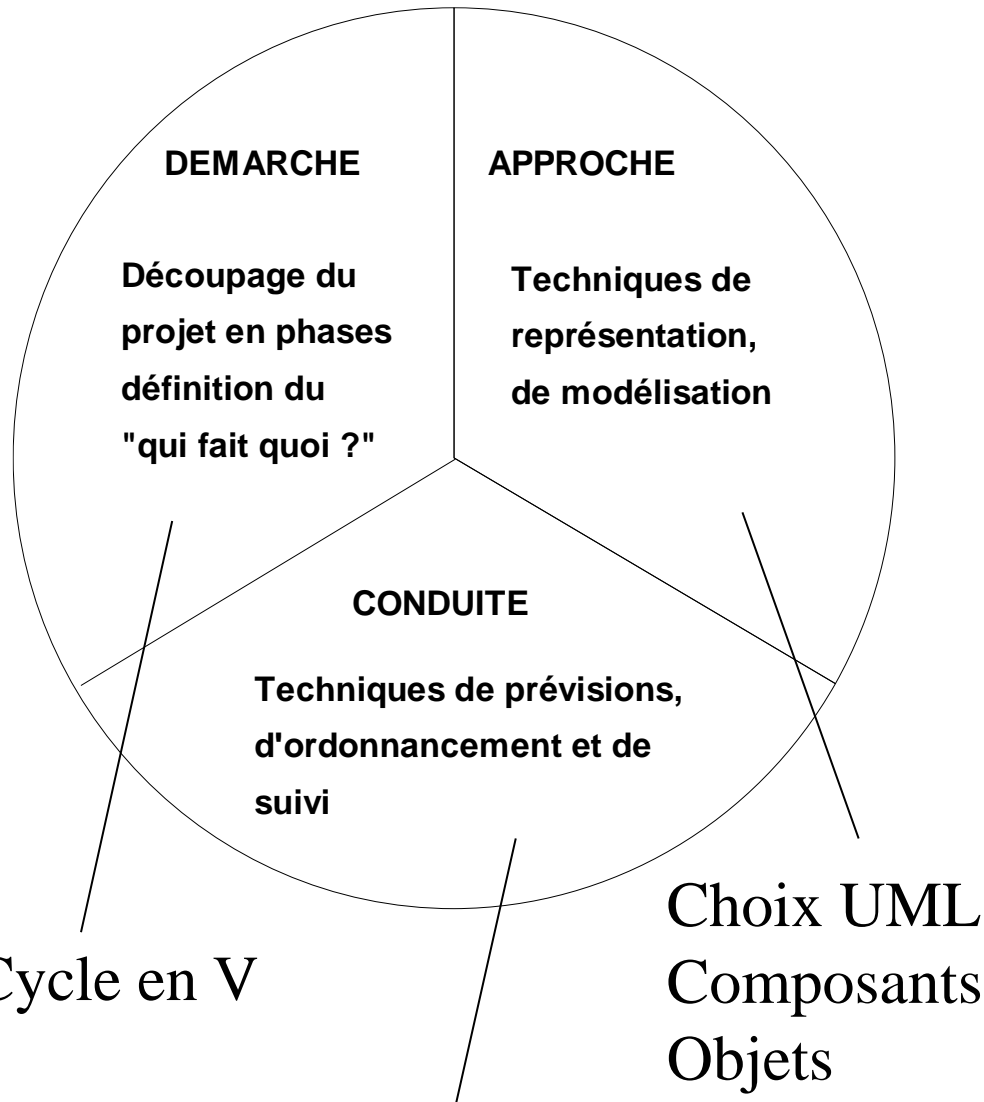
La qualité est un métier en soi (peu abordé dans l'UE)

- Métrique : indicateur quantitatifs aidant à évaluer l'évolution d'une dimension de la qualité logicielle
 - Loc, loc/classe, classes/package, profondeur d'héritage, couverture de test, ...
- Maîtrise du risque : identification, classification, mesures de mitigation...

Dans l'UE, on recherche en particulier :

- Portabilité : minimiser les hypothèses sur la plateforme technique, permettre la réalisation sur toute plateforme
- Réutilisabilité, modularité : accent sur la construction d'une application à l'aide de composants réutilisables
- Aspects non fonctionnels peu abordés : performance, sécurité...

Composantes d'une méthode



La démarche décrit les actions à mener, leur enchaînement et les dispositifs nécessaires pour atteindre les objectifs du projet.

La conduite est la technique de prévision et d'ordonnancement des travaux. Elle permet le suivi du projet.

L'approche est la technique de modélisation utilisée au cours du projet.

La technique de modélisation comprend :

- un formalisme,
- une heuristique.

Le formalisme offre une façon de s'exprimer : du texte, des modèles.

L'heuristique est la technique de découverte : c'est la manière de trouver et de formuler les éléments nécessaires au modèle.

Méthode

Qu'attend-t-on d'une méthode ?

- Quand ?
 - Découpage en étapes, manière d'aborder le problème, de découper le raisonnement
- Qui ?
 - Définition des rôles des intervenants, organisation
- Comment ?
 - Stratégies et outils de modélisation préconisés, traitements nécessaire pour chaque étape
- Quoi ?
 - Spécification des objectifs et besoins, traçabilité des exigences

Quelles sont les qualités recherchées d'une méthode ?

- Gains de production, Gains de qualité, Réduction des coûts, Gestion des risques

Méthode de Développement : une définition

Un méthode doit expliquer quand ? comment ? qui ? doit faire des modèles et des schéma (lesquels ? dans quel objectif ? à quelle granularité d'abstraction) pour concevoir une application.

Une méthode doit permettre de bons résultats (gain de productivité).

Elle doit être facile à apprendre et à mettre en œuvre.

Toute méthodologie de développement a pour objectif de produire un logiciel répondant au mieux au cahier des charges, tout en améliorant la productivité.

Elle définit :

- ☐ des étapes (e.g. Analyse, Tests Validation, Conception...), le plus souvent aussi une découpe en itérations plus ou moins longues.
- ☐ des moyens (e.g. diagrammes UML, tests, fiche détaillée, génération de code...),
- ☐ des produits intermédiaires du développement (c.à.d. les livrables : e.g. document d'analyse, tests de validation...).

Elle peut aussi définir une structure générale pour la gestion de projet (petites ou grosses équipes, organisation du travail...

II. Modélisation

Retour sur la diversité

Différents intervenants

- Chef de projet (avancée du projet)
- Développeur (méthode, algo, dépendance aux API)
- Testeur (définition de tests, exécution de tests)
- Deploiement (mise en production, tests de montée en charge)
- Qualité (respect des charte, métriques)
- CLIENT (besoin satisfait)

Points de vue différents

Niveaux d'abstraction différents

Pourtant tout cela doit être **cohérent**

Limitation des langages de programmation

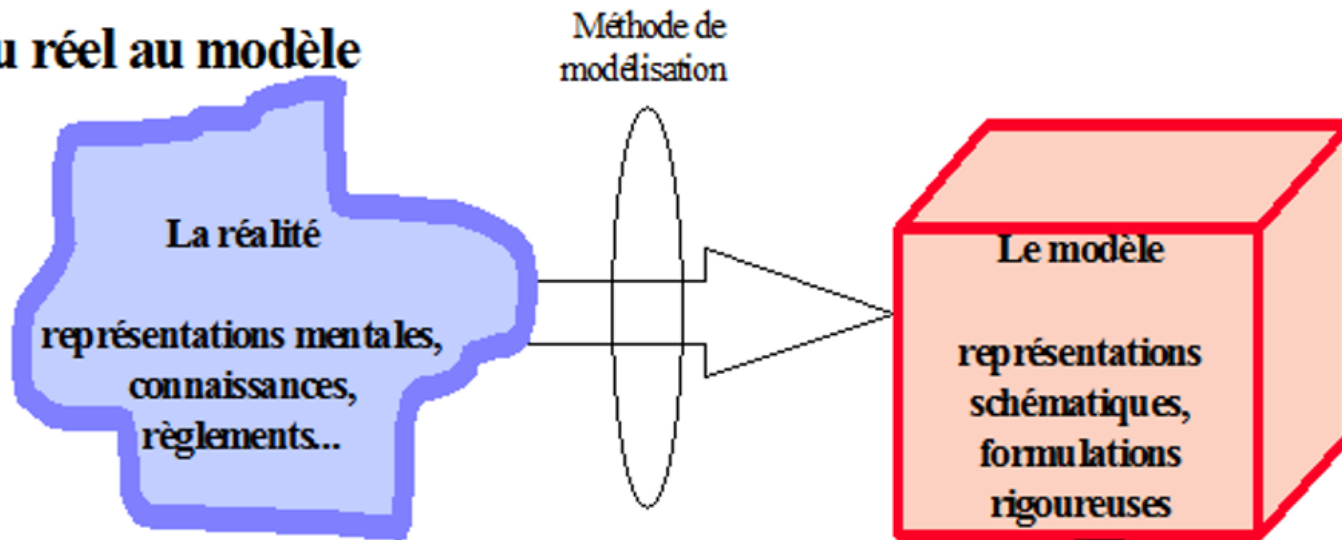
1 seul niveau d'abstraction

1 seul niveau de diversité

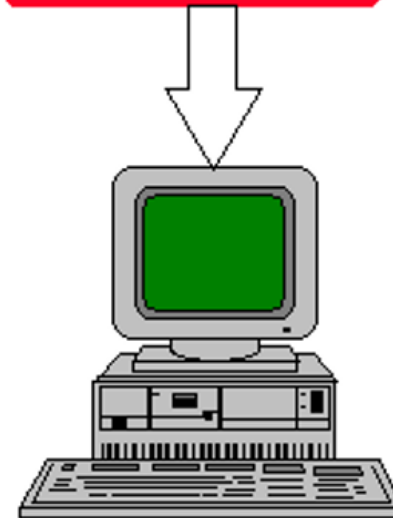
Les modèles offrent une solution pour aider à maintenir la coherence tout en acceptant la diversité

Qu'est-ce qu'un modèle ?

◆ Du réel au modèle



◆ Du modèle au logiciel

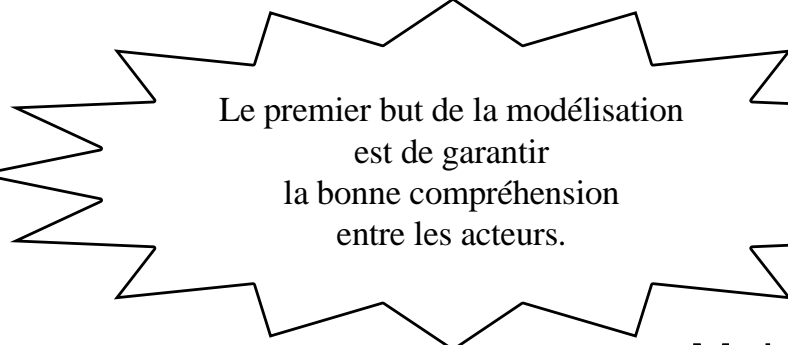


Modèle = représentation simplifiée du réel

Contraintes de représentation facilitant le passage au logiciel

Pourquoi Modéliser ?

COMMUNIQUER



Le premier but de la modélisation
est de garantir
la bonne compréhension
entre les acteurs.

◆ **Entre utilisateurs et informaticiens**

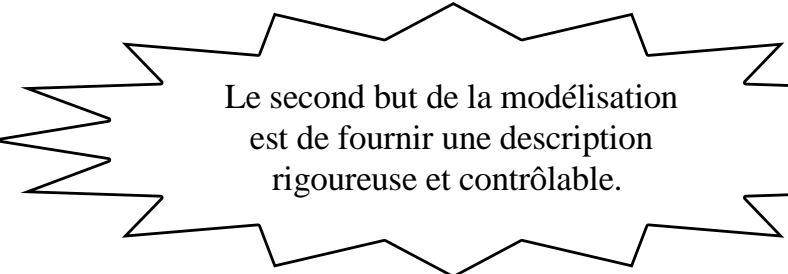
◆ **Entre informaticiens :**

↙ analystes, concepteurs,
développeurs,
mainteneurs...

Mais aussi avec « soi » :

- mieux comprendre le problème
- documenter

CONTROLLER



Le second but de la modélisation
est de fournir une description
rigoureuse et contrôlable.

◆ **Qualité des modèles :**

↙ cohérence,
exhaustivité, couverture,
pertinence,
économie...

Support outillé, génération de code, de documentation

Analyses du modèle (consistance, cohérence)

Communiquer et Contrôler

Comment atteindre ces objectifs ?

Parmi les critères que l'on cherche à garantir en utilisant une méthode de modélisation, on peut citer :

- **la cohérence** : toutes les représentations qui composent le modèle s'articulent entre elles et ne se contredisent pas.
- **l'exhaustivité et la couverture** : le modèle contient toute l'information nécessaire pour atteindre les buts fixés.
- **la pertinence** : toutes les informations rassemblées dans le modèle ont leur utilité.
- **l'économie du modèle** : les représentations sont aussi concises que le permettent les critères précédents. Le modèle possède l'élégance et la structure qui facilite son exploitation.

Cohérence et Diversité

Il y a un fort besoin de cohérence

Comme toutes les informations visent à développer une application

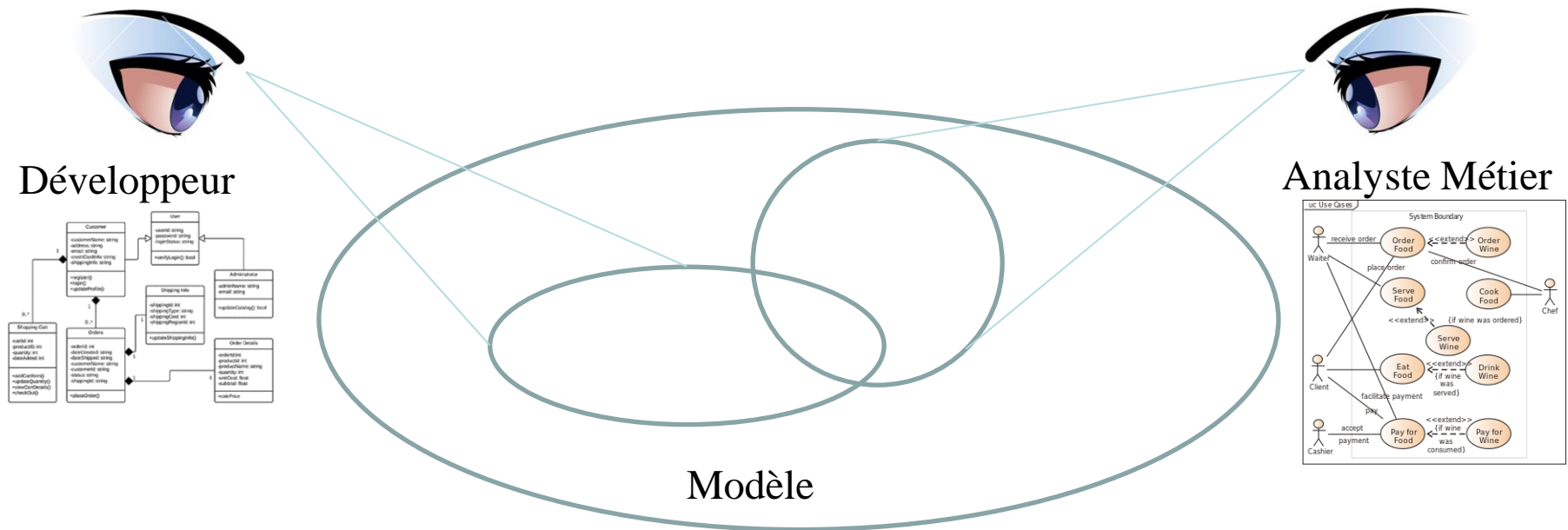
⇒ il serait intéressant de tout centraliser

Mais il ne faut pas uniformiser !

Chaque intervenant a sa façon propre de travailler

On ne peut pas donner un langage de programmation à un client !

Modèle et Diagramme



Modèle :

L'ensemble des informations permettant la conception, le développement et la maintenance du logiciel (diagrammes, code, tests, documentation, ...)

Diagramme ou Vue :

Sous ensemble des informations du modèle (filtré) présentées selon un certain point de vue (selon un axe servant un objectif)

Objet, Composant, Service

Orienté-Objet

Concepts clé facilitant la construction d'applications de grande taille *maintenables*.

Génération de langages :

1. Primitif : asm

Pas de structure, GOTO

2. Procédural : C, Pascal

Structure de contrôle, fonctions, tableaux, struct

3. Orienté Objet : Java, C++, ...

Données (struct) et fonctions sont regroupés en unités, encapsulation

4. Dynamique : Python, Perl, JS, ...

Typage faible (scalaire, list, hash), réflexion avancée

En parallèle, courant fonctionnel (lisp -> ML -> Haskell...),
aujourd'hui assez bien intégré dans les langages OO (lambda).

La classe

La classe

- ✓ Catégorie ou type d'objet.
- ✓ Munie de responsabilités matérialisées par une/des interfaces
- ✓ Définie par ses attributs et méthodes
- ✓ Exemple : classe locomotive

Locomotive
<ul style="list-style-type: none">- modele- puissance- vitesse- couleur
<ul style="list-style-type: none">+ allerEnAvant()+ allerEnArriere()+ demarrer()+ stopper()+ accélérer()

L'objet :instance d'une classe

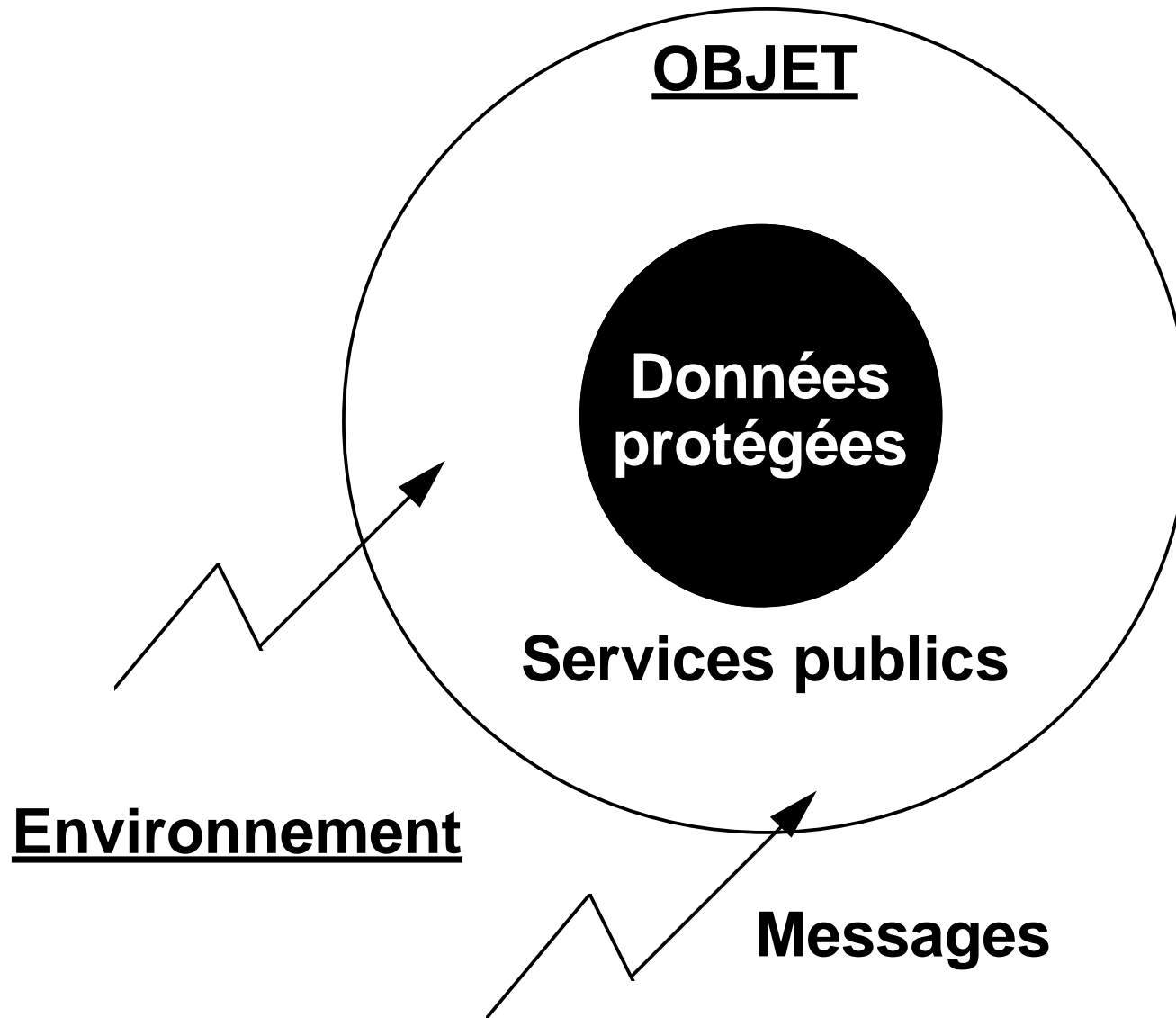
- Les instances d'objets sont définies par la valeur de leurs attributs.
- Les traitements qu'on peut réaliser sur ces attributs sont définis au niveau de la classe à travers les méthodes.

Loco21:Locomotive

-couleur= rouge
-puissance= 18000
-vitesse= 230
-modele= AL2119

Un objet = instance d'une classe
dans un état particulier

L 'objet : notion d'encapsulation



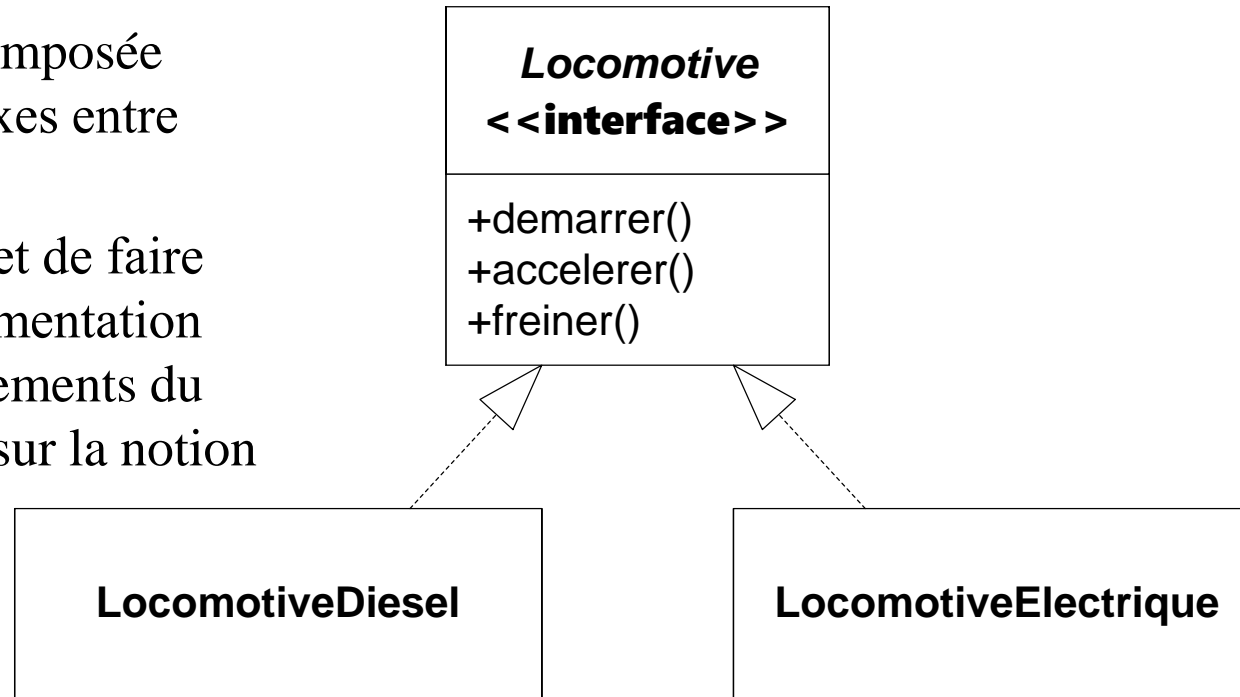
Substituabilité du noyau sans impacter les clients.

L 'abstraction

La modélisation OO pousse à
gagner en abstraction :

- ✓ Une application est composée d'interactions complexes entre composants
- ✓ L'encapsulation permet de faire abstraction de l'implémentation effective des comportements du système pour reposer sur la notion d'*interface*

Une interface définit des
responsabilités,
indépendamment de la façon
dont elles sont réalisées.



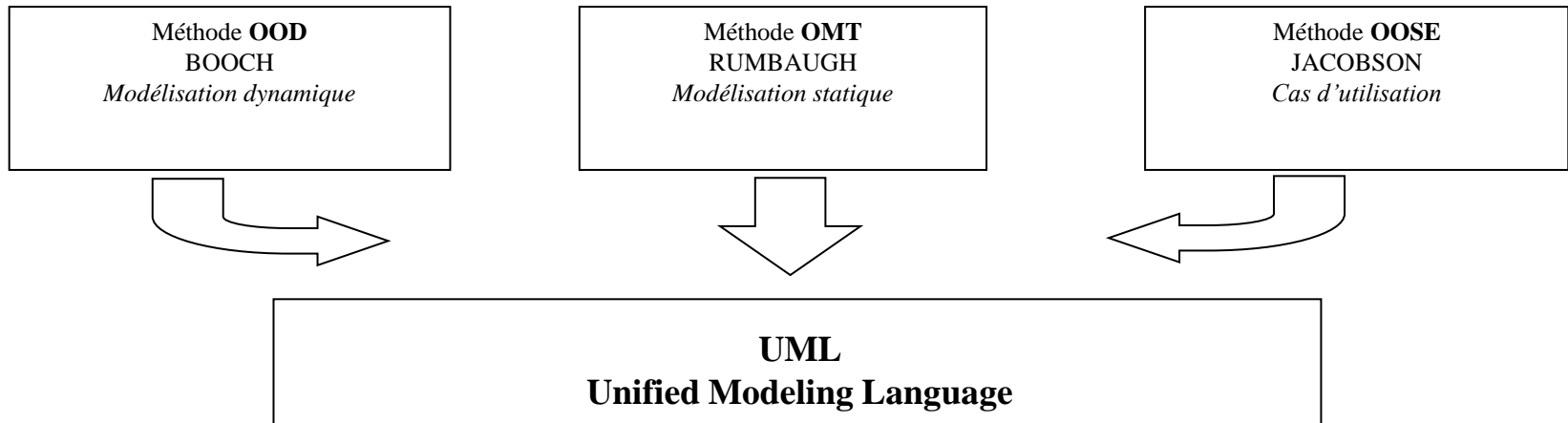
Objets vs Composants

- L'orienté-objet présente de nombreuses qualités MAIS
 - Unité de réutilisation médiocre
 - La classe ne vient pas seule, package entier.
 - Hétérogénéité mal gérée dans un modèle OO pur
 - SGBD relationnel ? langages divers ?
- Au-delà de l'objet :
 - Modèles de composant
 - Présente une API ou interface pour communiquer avec l'extérieur
 - Unité de réutilisation et de déploiement
 - Support de plateformes hétérogènes
 - Support de plateformes distribuées
 - Vision composant promulguée dans l'UE (arc 2)
- Au-delà du composant :
 - SOA : Service Oriented Architecture, SaS Software as a Service
 - Compatible avec la vision orienté-composant
 - Application = chorégraphie entre composants dont on ne connaît **que** les API

Unified Modeling Language UML

Une notation **standard** pour représenter des
modèles d'artefacts logiciels

UML : historique



- 1998 : 1.0, agglomération des diagrammes OMT (~ classes), OOD (~séquences) et OOSE (~use case) : premier standard OMG
- 1998-2003 : évolution rapide vers UML1.4, ajout de diagrammes, sémantique encore vague, représentation interne mal standardisée (échanges entre outils ?)
- 2005-2010 : UML 2.0->2.4, refonte complète de l'infrastructure, publication d'un méta-modèle de référence d'UML, sémantique mieux précisée. Evolution rapide, nombreux soucis révélés par l'intégration
- 2020 : UML 2.5.1 (depuis 2017) le langage est stable, 14 diagrammes/vues

UML est un standard industriel (OMG et ISO) => communiquer++

Selon temps disponible

Diagramme de classe :

- Structure (attributs, opérations), associations, typage
- Expression, Add, Constante, eval():int + implémentations

Diagramme d'objet :

- Instances des classes dans une configuration particulière, snapshot de l'application
- 3+2

Diagramme de Séquence :

- Axe : temps (vertical), interactions entre instances, données échangées
- (3+2).eval()

Intersections non vide entre ces diagrammes :

- Classes -> lignes de vie, opérations->messages, Classe -> instances
- Le modèle UML unifie et assure la cohérence.