TDD

Test Driven Development

TDD

Présentation

Historiquement :

- les tests sont écrits **après le développement**
- les tests sont écrits **en fonction du code**
- 2 développeurs ne vont pas coder à l'identique la même fonction
- ils ne vont donc pas tester leurs codes respectifs de la même manière
- et pourtant ils doivent tous 2 proposer un logiciel qualitatif
- est-ce logique de procéder ainsi : **coder puis tester ?**

Kent Beck

- fondateur de l'**Extreme Programming** (1999)
- a pour objectif une meilleure **qualité du code**
- crée l'approche Test-First
- propose de changer **coder puis tester** en **tester puis coder**

TDD

Présentation

Processus du TDD

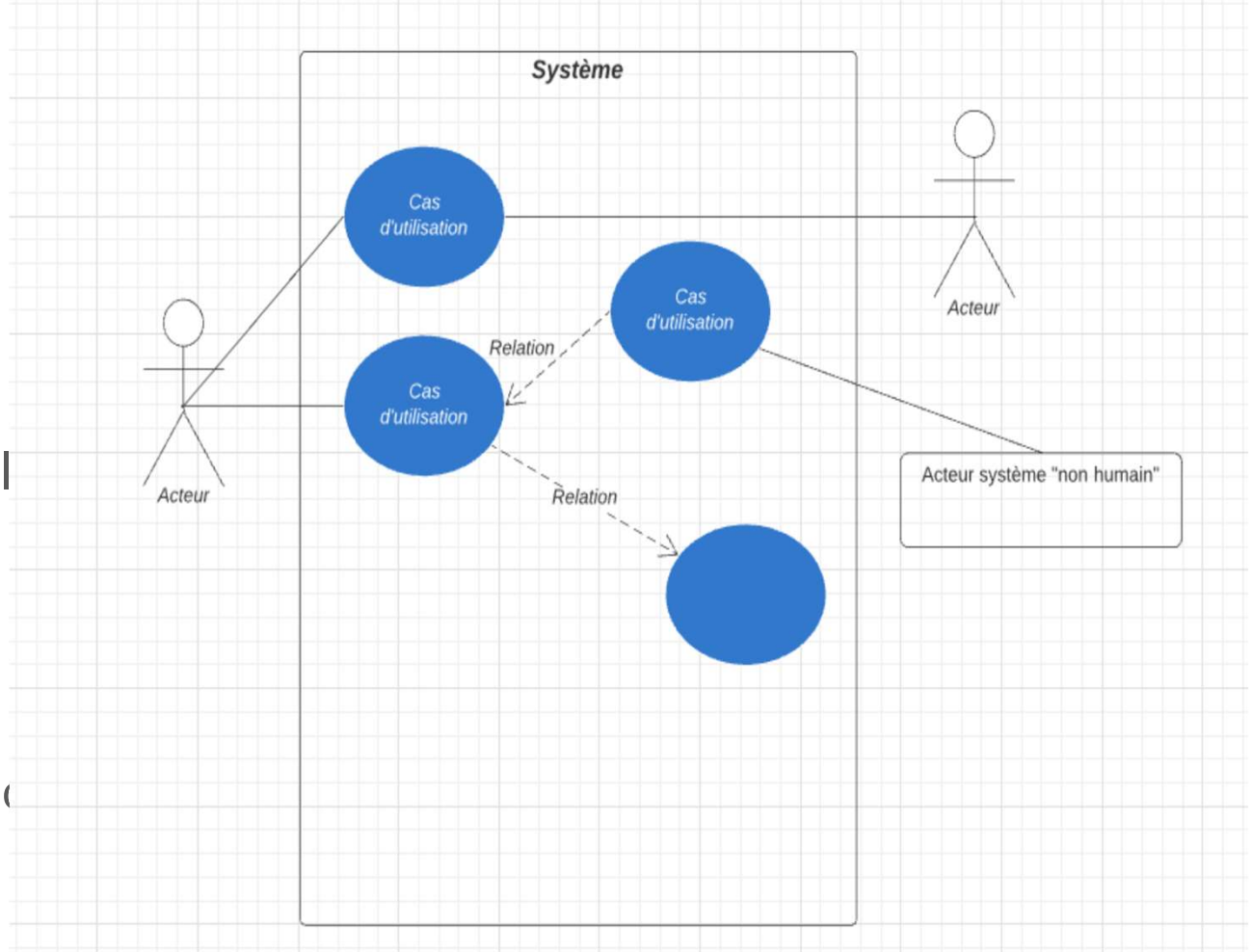
- on part d'une **User Story** (US)
- **on écrit un test** qui décrit un cas de l'US et **qui échoue**
- **on écrit le code nécessaire** et **suffisant** pour que le test réussisse
- on **recommence** jusqu'à ce que toutes les conditions de validation de l'US soient remplies
- c'est un développement **progressif → incrémental** et **cyclique**
- c'est à priori contre-intuitif, inhabituel
- nous n'avons pas été formés à cela

Rappels

1 - Rédiger des User Stories

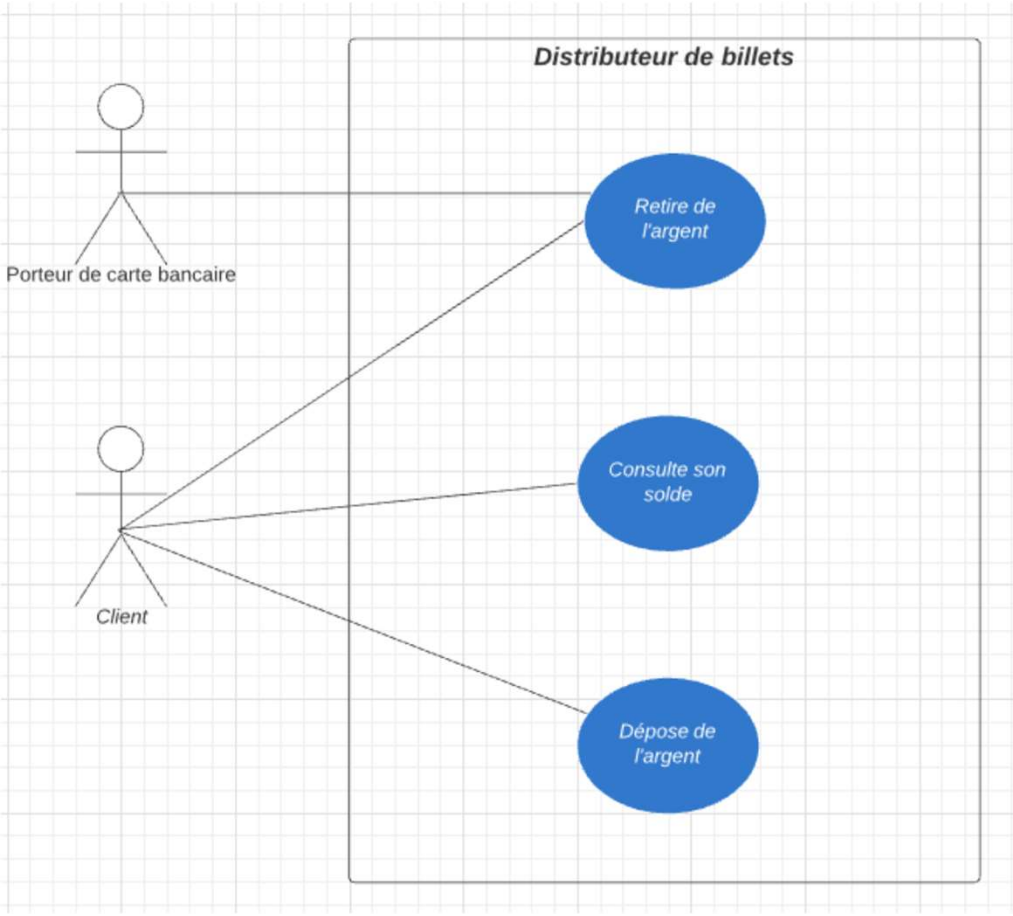
Proposition : partir du cas d'utilisation (use case)

- il représente un ensemble de **séquences d'actions** réalisées par l système et observables par un **acteur** particulier
- chaque cas d'utilisation décrit un comportement du système et correspond à une **fonction métier** système



Voici un exemple très simple de **cas d'utilisation** illustrant l'usage d'un distributeur de billets par 2 acteurs :

- un simple porteur de carte bancaire (qui n'est pas client de la banque où il retire de l'argent)
- un client de la banque où il retire de l'argent

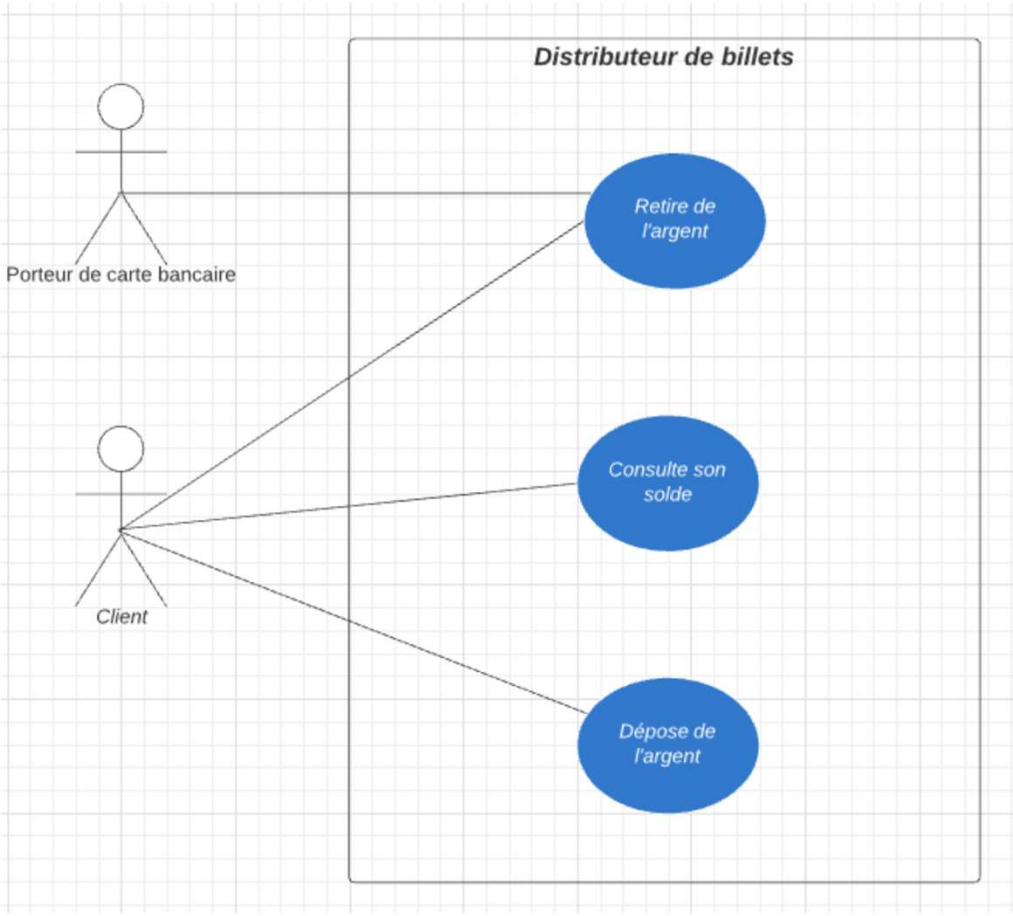


En tant que **non client** de la banque :

- je peux retirer de l'argent
- je ne peux rien faire d'autre

En tant que **client** de la banque :

- je peux retirer de l'argent
- je peux consulter mon compte
- je peux déposer de l'argent



TDD

Présentation

Autre exemple de « User Stories »

Plaçons-nous dans le contexte d'un **site de E-Commerce**

En tant que **futur client**, je dois pouvoir créer un compte

En tant que **client inscrit** :

- je peux modifier mon compte
- je peux rechercher des produits qui m'intéressent
- je peux passer des commandes

En tant qu'**administrateur du site** :

- je peux vérifier l'identité de nos clients
- je peux suspendre des comptes clients
- je peux créer des produits

Diagramme de séquences

Rappels

Un **diagramme de séquences** peut être complété par :

- les **scénarios alternatifs** qui correspondent à d'autres hypothèses que le scénario principal, par exemple si la carte bancaire n'est pas valide ou pas lisible
- les **messages d'erreurs** : on modélise les cas d'erreurs de fonctionnement du distributeur (panne ou défaillance de la connexion entre le distributeur et le système bancaire central), ou encore les erreurs de saisie du porteur de cartes et le traitement informatique qui en serait fait (blocage de carte)

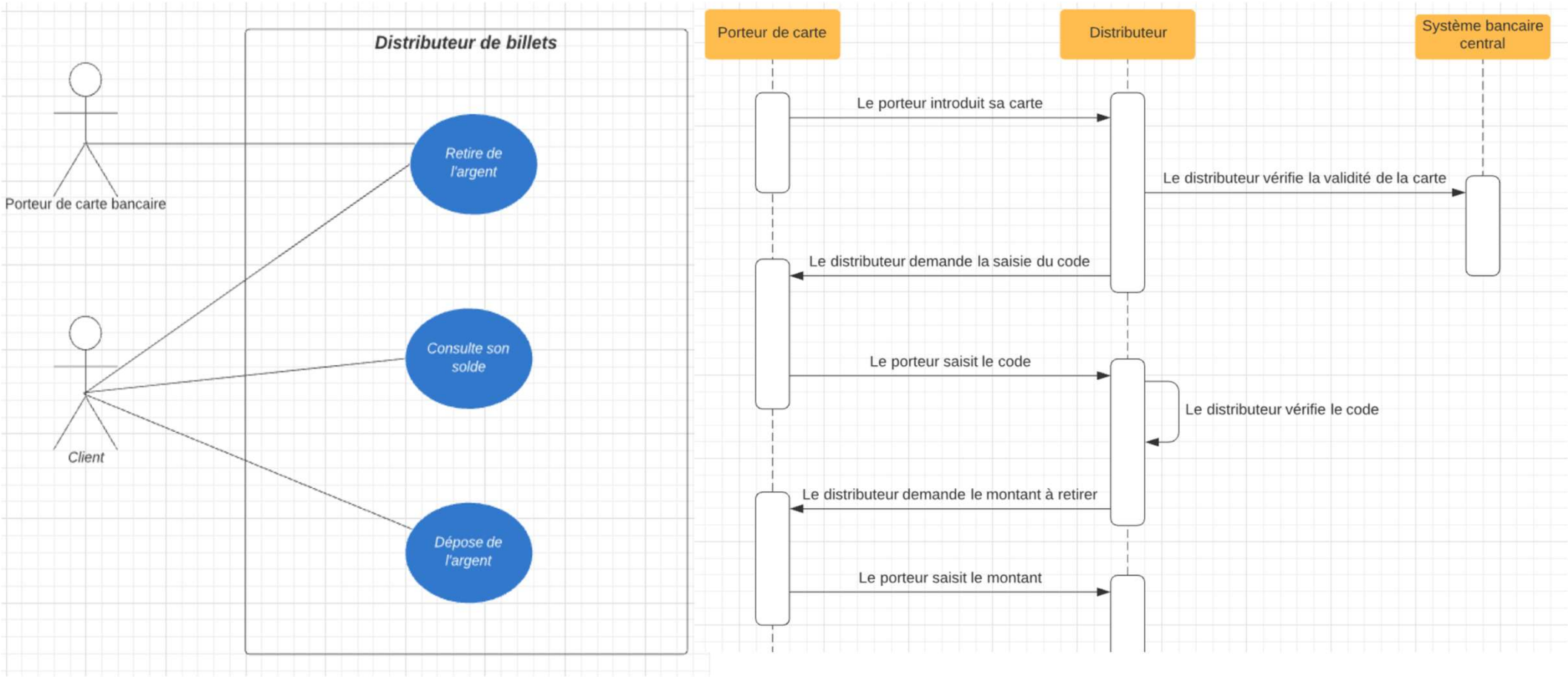

Distributeur de billets
Porteur de carte bancaire
Retire de l'argent
Consulte son solde
Client
Dépose de l'argent
Porteur de carte
Distributeur
Système bancaire central
Le porteur introduit sa carte
Le distributeur vérifie la validité de la carte
Le distributeur demande la saisie du code
Le porteur saisit le code
Le distributeur vérifie le code
Le distributeur demande le montant à retirer
Le porteur saisit le montant

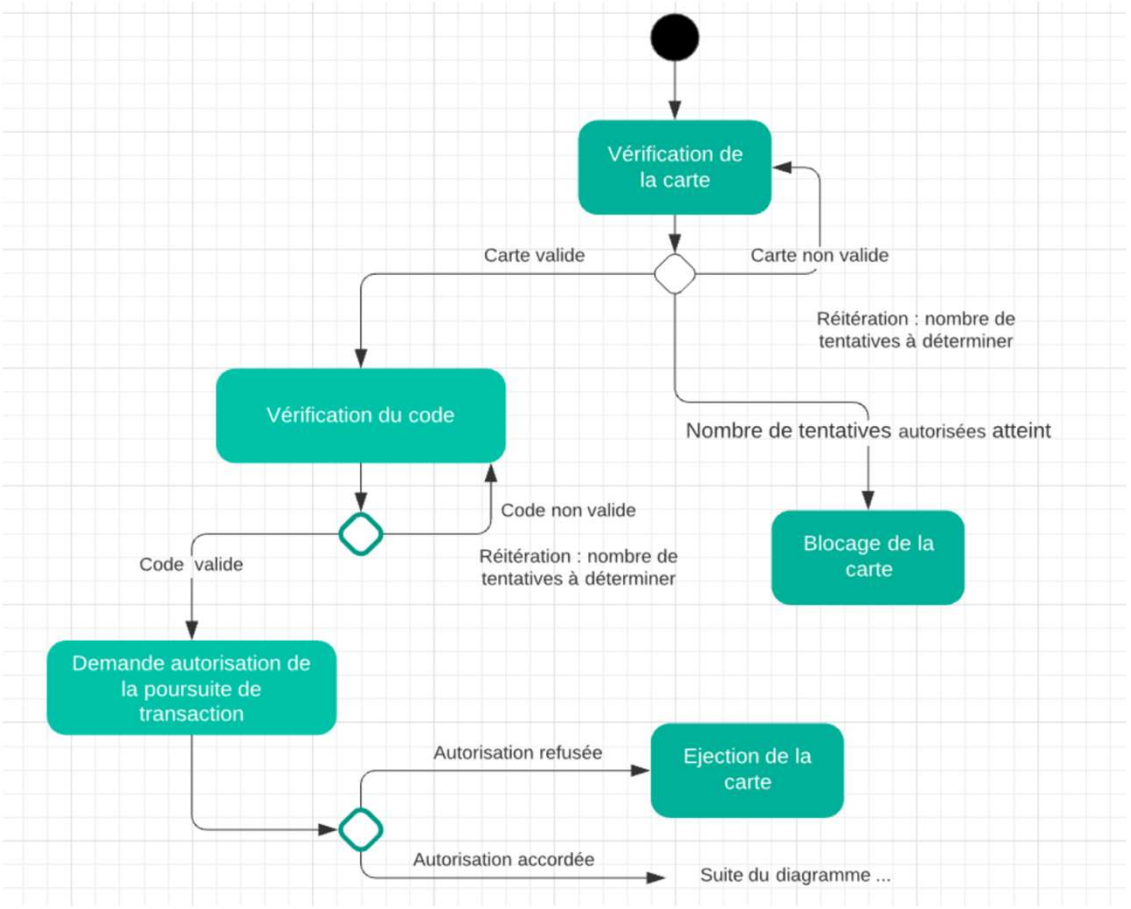
Diagramme d'activités

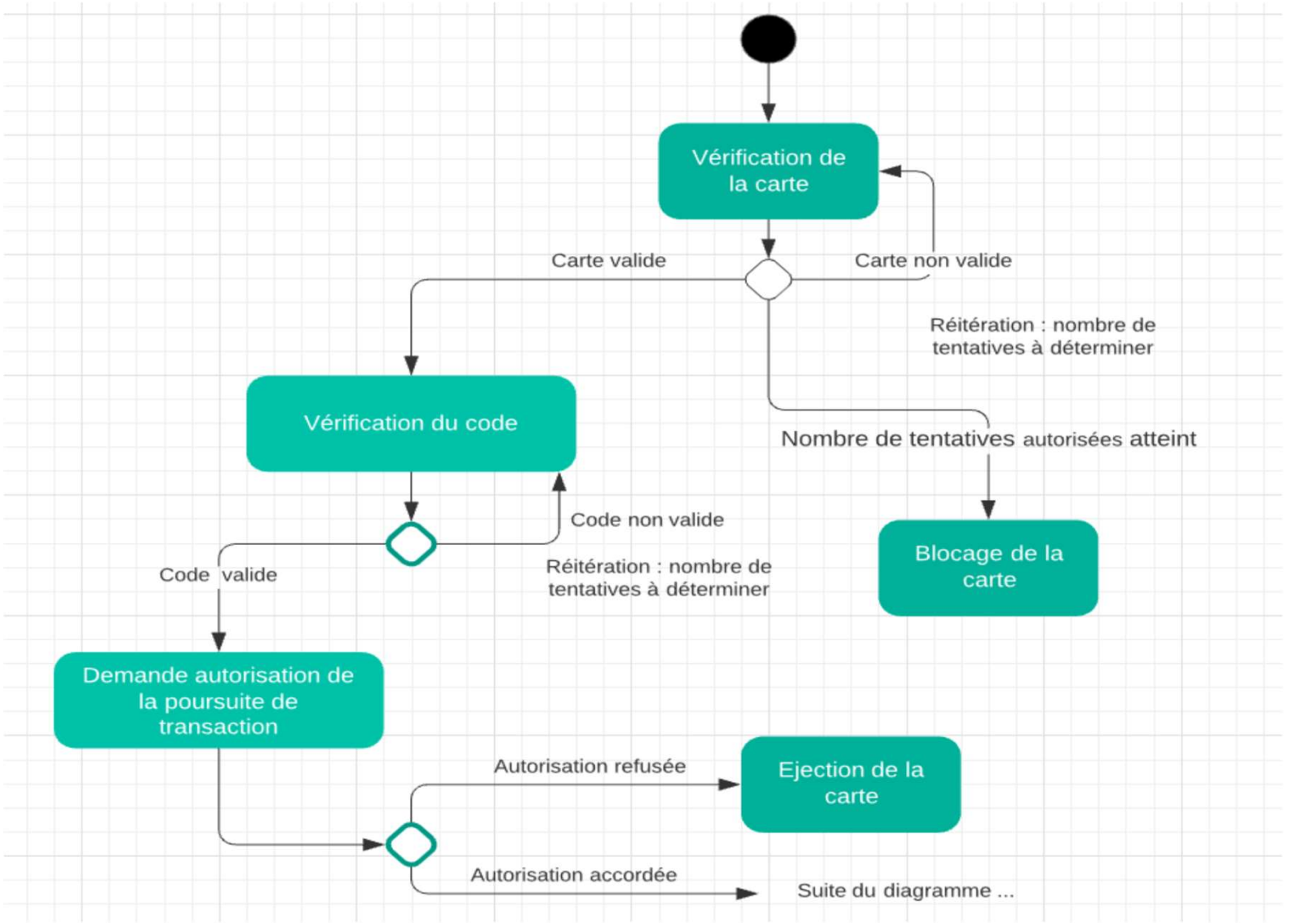
Rappels

- Le **diagramme d'activité** présente les **actions** effectuées dans un cas d'utilisation et affiche le flux de travail d'un point de départ à un point d'arrivée en détaillant tous les chemins possibles
- Il est souvent comparé au terme **organigramme**
- Le parallèle est possible avec un modèle conceptuel des traitements (MCT) de la méthode Merise

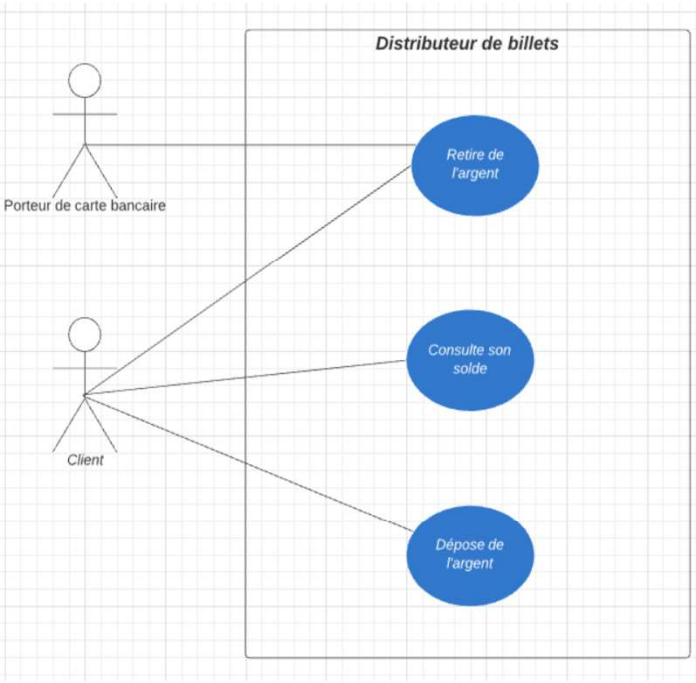
On y trouve :

- des **arcs** (circuits, flux) symbolisés par les flèches orientées
- des **nœuds** :
  - les nœuds, début et fin
  - le nœud d'action : symbolisé par un rectangle aux bords arrondis, c'est un nœud exécutable qui représente une action, une transformation ou un calcul. Par exemple, la vérification du code de la carte est un nœud d'action.

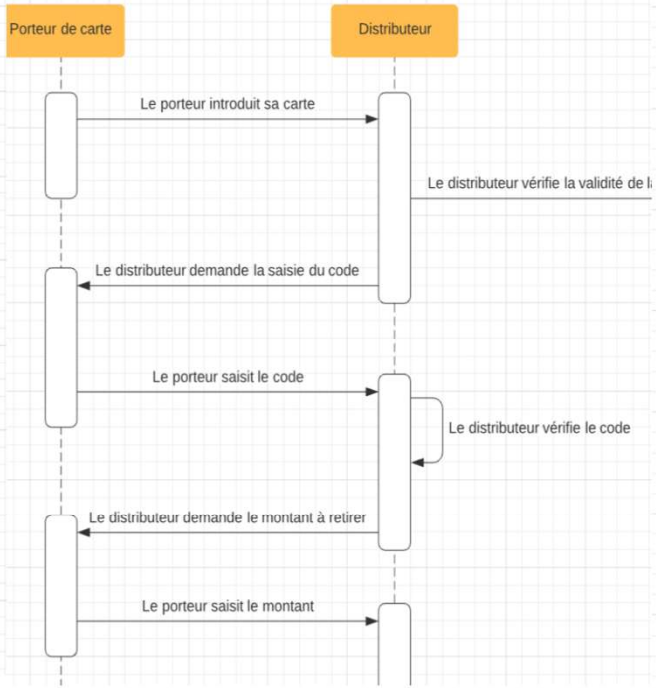



Vérification de la carte
Carte valide
Carte non valide
Réitération : nombre de tentatives à déterminer
Vérification du code
Nombre de tentatives autorisées atteint
Code non valide
Code valide
Réitération : nombre de tentatives à déterminer
Blocage de la carte
Demande autorisation de la poursuite de transaction
Autorisation refusée
Ejection de la carte
Autorisation accordée
Suite du diagramme ...

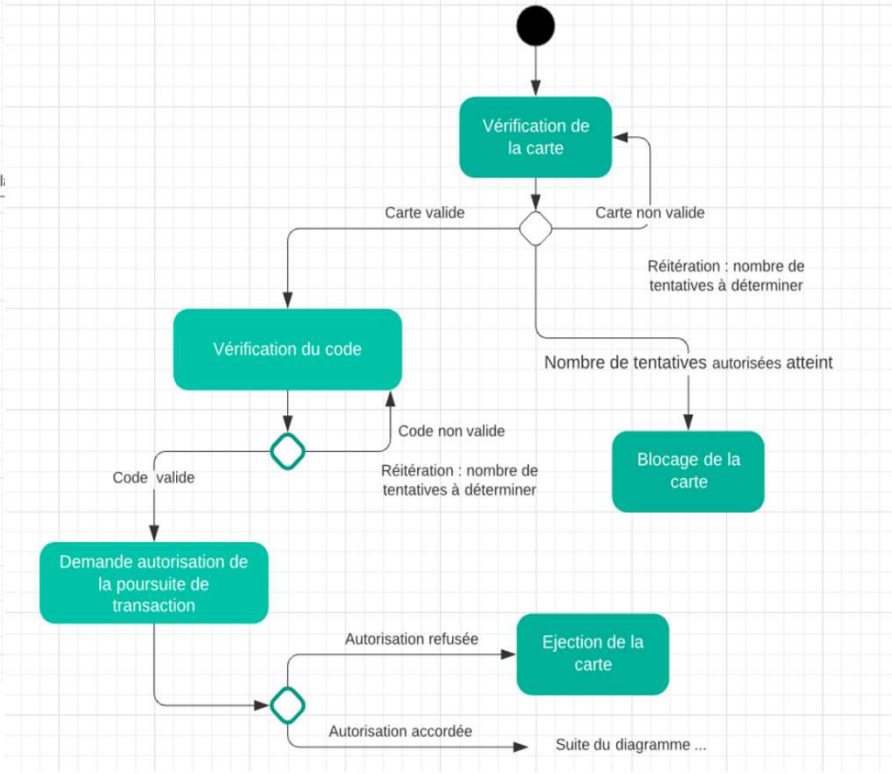
Cas d'utilisation



Séquences



Activités



Cas d'utilisation → Séquences → Activités

Mise en évidence des **user stories** et des **tests**

TDD

Red
Green
Refactor

Red

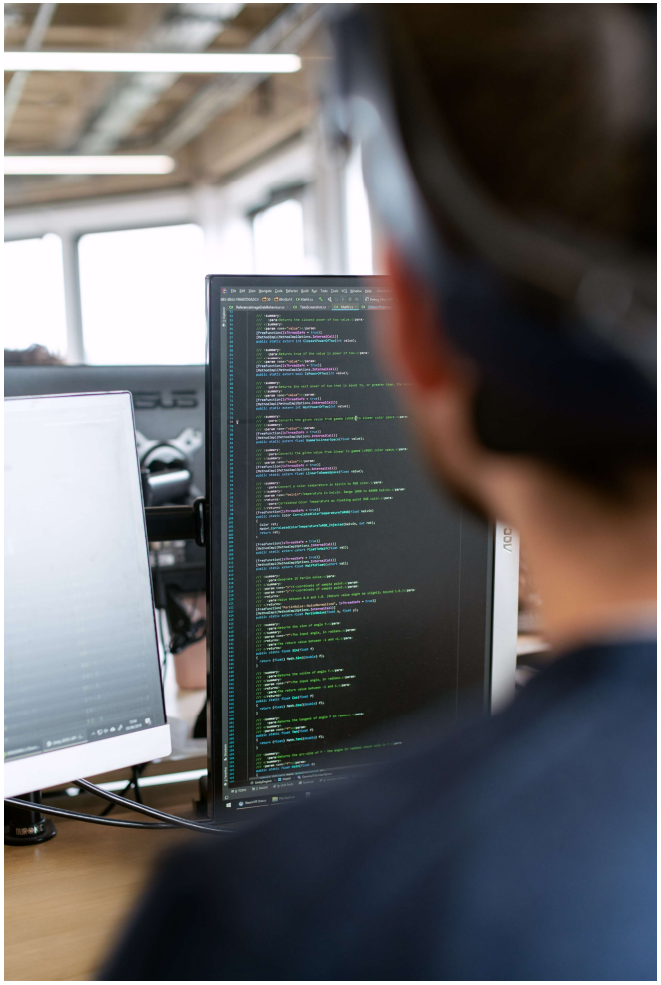
- positionnement : utilisateur
- action : écriture d'un **test**
- le code correspondant à ce test n'existe pas encore donc **ce test va échouer**

Green

- positionnement : développeur
- action : écriture du **code strictement nécessaire** pour **passer le test écrit en Red**

Refactor

- positionnement : développeur
- action : **amélioration du code**



TDD : exemple

Red
Green
Refactor

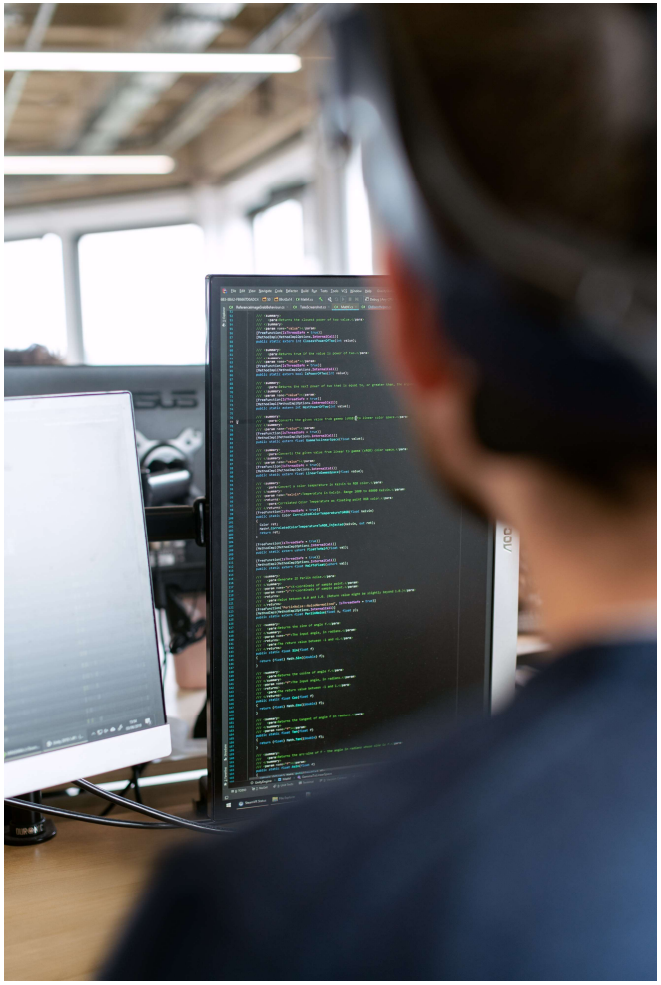
Contexte : connexion à un site web avec un identifiant et un mot de passe (langage php / base de données mysql)

Red

Ecriture de 2 tests :

Test 1 : je vérifie l'existence de l'utilisateur en testant si son identifiant existe dans la base de données

Test 2 : si l'identifiant existe, je vérifie si le mot de passe saisi correspond à celui qui est stocké dans la base de données



TDD : exemple

Red
Green
Refactor

Green

J'écris le code **strictement nécessaire** pour **passer les 2 tests écrits en Red.** Je veux passer de **Red** à **Green.**

Pour le test 1 :

- je me connecte à la base de données
- je vérifie l'existence de l'identifiant de l'utilisateur
- s'il n'existe pas → message d'erreur
- s'il existe → j'exécute le test 2

Pour le test 2 :

- je teste si le mot de passe saisi correspond au mot de passe stocké dans la base de données
- si oui j'autorise l'utilisateur à entrer sur le site
- sinon j'affiche un message d'erreur, je peux aussi proposer à l'utilisateur de renouveler son mot de passe en cas d'oubli

TDD : exemple

Red
Green
Refactor

Refactor

Le code pour passer les 2 tests a été écrit dans la phase **Green.** Mon code fonctionne, l'application pourrait être déployée.

Mais … est-ce suffisant ?

Dans cette phase bleue, je cherche à améliorer mon code sur les 3 critères suivants :

- **lisibilité** : élimination du code inutile, indentation
- **maintenabilité** : choix de noms de variables appropriés, écriture de commentaires strictement utiles
- **efficacité** : optimisation du nombre d'instructions, utilisation de tableaux, de boucles, appels de fonctions …

… tout en conservant bien sûr la même fonctionnalité

TDD - Forces	TDD - Faiblesses

• le code est souvent **plus épuré**
• le code est souvent **mieux structuré**
• le code comporte généralement **moins de bugs**
• un code plus simple et mieux écrit assure une **meilleure maintenabilité**
• approche en **totale adéquation avec l'agilité**
	• il faut des **développeurs de qualité**
	• il faut des développeurs **expérimentés en TDD**
	• nécessite des **pratiques agiles** donc :
	  • une organisation **prête à adopter l'agilité**
	  • des **collaborateurs formés** à ces pratiques

BDD

Behavior Driven Development

BDD

Présentation

- on met en avant le **comportement des logiciels** et le **comportement des utilisateurs** avec leurs logiciels
- le logiciel est considéré du **point de vue de l'utilisateur**
- on veut favoriser :
  - la **collaboration** entre les **développeurs** et les autres **parties prenantes** de l'entreprise (non techniques)
  - la **communication** et l'illustration par des **exemples concrets**
- on utilise un langage **commun**, courant
- cette approche est particulièrement prisée dans le **développement frontend**

BDD

Présentation

Les étapes

- définition des **besoins** et des **objectifs**
- description des **fonctionnalités** attendues à l'aide de **scénarii**
- utilisation d'**exemples** pour expliquer aux développeurs les **comportements attendus**
- **réponses** apportées par les développeurs **à chaque scénario**
- **tests**

BDD

Syntaxe

Le BDD utilise le **vocabulaire de la vie courante** pour détailler **les comportements attendus :**

- **Étant donné** (Given)
- **Quand** (When)
- **Alors** (Then)
- **Et** (And)

Exemple :

- **Étant donné** que je suis connecté au réseau wifi de mon école
- **Et** que j'ai un compte sur la plateforme de l'école
- **Quand** je souhaite voir mon planning de formation
- **Alors** je trouve les modules et les numéros de mes salles de cours
- **Et** je peux m'y rendre facilement

BDD

Langage ubiquitaire

Le langage ubiquitaire

- entre les techniciens et les autres acteurs internes ou externes à l'entreprise
- **langage commun** : « *ubiquitous* » en anglais
- les termes utilisés sont **précis** et **identiques pour tous**
- on peut **créer un dictionnaire** interne ou un **wiki** qui répertorie tous ces termes
- ces termes sont utilisés dans le **langage verbal**, dans les **user stories**, dans les **documentations**, dans le **code** …

BDD - Forces	BDD - Faiblesses

• langage **ubiquitaire, commun,** simple, accessible à tous : dictionnaire, wiki ...
• **synergie** développeur / utilisateurs : **communication renforcée**
• **approche métier du code**
• **logiciels plus qualitatifs** pour les utilisateurs, notamment **interfaces web** et **applications mobiles**
• approche en **totale adéquation avec l'agilité**
	• les acteurs non-développeurs doivent être **disponibles**
	• les **scénarii** (spécifications, user stories ...) doivent être **bien rédigés**
	• nécessite des **pratiques agiles** donc :
	  • une organisation **prête à adopter l'agilité**
	  • des **collaborateurs formés** à ces pratiques

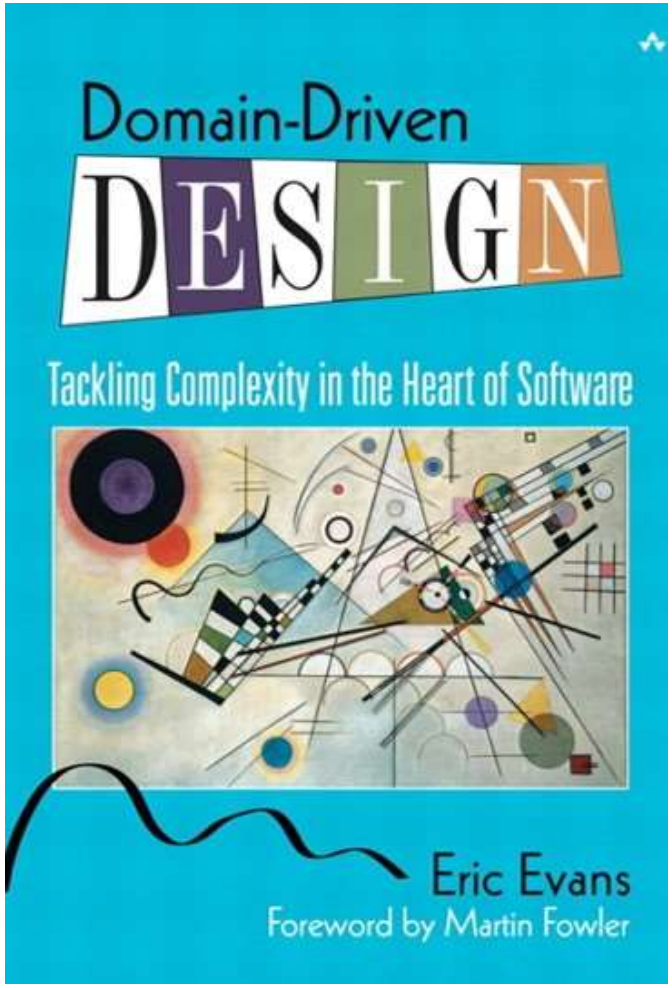
DDD

Domain Driven Design

DDD

Présentation

- **Eric Evans** publie en 2003 un livre intitulé **Domain-Driven Design, Tackling complexity in the Heart of Software**
- c'est une synthèse de son expérience de développeur notamment en programmation orientée objet
- il prône une **conception logicielle pilotée par le métier**
- le code résout les **problématiques métier** par une parfaite connaissance du **domaine** de l'entreprise et du **contexte** de l'utilisateur



DDD

Concepts énoncés par Evans

1 – on retrouve le langage ubiquitaire

- entre les techniciens et les autres acteurs internes ou externes à l'entreprise
- **langage commun** : « *ubiquitous* » en anglais
- les termes utilisés sont **précis** et **identiques pour tous**
- on peut **créer un dictionnaire** interne ou un **wiki** qui répertorie tous ces termes
- ces termes sont utilisés dans le **langage verbal**, dans les **user stories**, dans les **documentations**, dans le **code** …

DDD

Concepts énoncés par Evans

2 – le découpage du contexte *(bounded context)*

- on découpe le contexte global en plus **petits contextes logiques**
- exemples :
  - différents **services** au sein de l'organisation (finances, ventes, achats, production …)
  - différentes **filiales** d'une même entreprise
  - différentes **activités** d'un même groupe …
- **chaque contexte est isolé des autres contextes** et ne traite que ses propres problématiques → on ne s'oblige pas à des interactions, et s'il doit y en avoir, elles seront traitées plus tard

DDD

Concepts énoncés par Evans

3 - la modélisation

Evans propose **3 types d'objets** pour aider à structurer le code :

Entity

- objet défini par une **identité arbitraire** (un identifiant unique …)
- peut évoluer dans le temps

Value object

- objet sans identité, sans évolution
- exemple : l'objet Money = un montant et une devise (euro, dollar, …)

Service

- tout ce qui n'est ni entité ni value object est appelé service
- c'est le code qui applique les règles métier définies par le langage commun

DDD - Forces	DDD - Faiblesses

• langage **ubiquitaire, commun,** simple, accessible à tous : dictionnaire, wiki ...
• **synergie** développeur / utilisateurs : **communication renforcée**
• **approche métier du code**
• **logiciels plus qualitatifs** pour les utilisateurs, notamment **interfaces web** et **applications mobiles**
• approche en **totale adéquation avec l'agilité**
	• **adopter le langage commun**
	• nécessite des **pratiques agiles** donc :
	  • une organisation **prête à adopter l'agilité**
	  • des **collaborateurs formés** à ces pratiques