Domain Driven Design
&
Architecture hexagonale

Responsable : MARIN Geoffrey

Dernière modification : 21/09/2025

Université Claude Bernard Lyon 1

Utilisation de OpenAI pour la reformulation de phrases et vérification du plan.

Sommaire

Objectif(s)

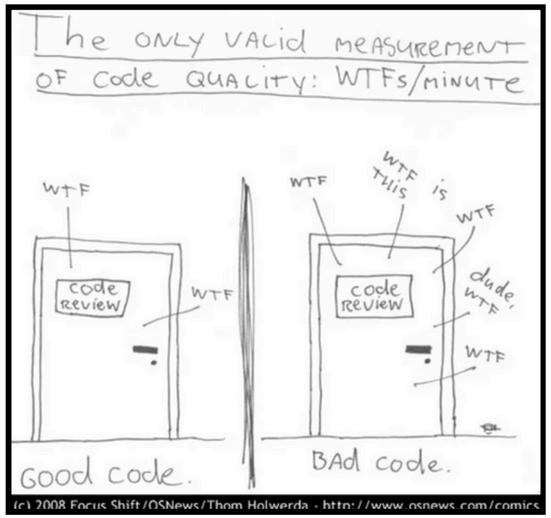
A l’issue de cette présentation vous serez à même de :

- Améliorer votre prise de recul sur les outils utilisés;
- Comprendre l’intérêt des architectures logicielles;
- Comprendre que le coeur du développement, ce sont les bonnes pratiques;
- Comprendre que les outils ne sont pas une fin en soi, mais adaptés à une situation;
- Adapter une architecture logicielle à une architecture de SI;

Pourquoi concevoir une architecture ?

Les symptômes de mauvaises pratiques

- ***"Code spaghetti"*** est une conséquence de l'**accumulation de dettes techniques** :
  - Prolifération de **logiques métier** dupliquées
  - Absence de **conventions de développement** (nommage, documentation, revue de code, etc...)
  - Présence de classes "Dieu" qui gère trop de **responsabilités**. (=> Cohésion faible)
  - Absence de stratégie de **tests automatisés**
  - Violation des **principes S.O.L.I.D** (=> Couplage fort)
  - Absence de suivi des **mises à jours des librairies** utilisées



"From spaghetti to spaghetto ou comment ne pas emmêler votre code", Romain Boniface, Blog WexIt.fr, 14/05/2020

"What is Spaghetti Code (And Why You Should Avoid It)", Stephen Watts, Blog BMC.com, 17/06/2020

"From spaghetti code to professional code", Vahab Najari, Medium, 20/11/2020

Les impacts

Utilisateur(s)

- Bugs
- Ruptures de services
- Peu ou pas, de mises à jour

Projet

Chef de projet :

- Plus de visibilité
- Communication tendue

Développeurs:

- Stress
- Travail peu intéressant
- Perte d'intérêt

Business

- Perte de productivité
- Perte de rentabilité
- Perte de crédibilité

"Introduction à l'Architecture & la Clean Architecture", Harold Cohen, Youtube TheTribe, 17/11/2021

Qu’est ce que l’architecture logicielle ?

> **“The set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both.”**
>
> Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord et Judith Stafford
> *“Documenting Software Architectures: Views and Beyond (2nd Edition)”*
> *Addison-Wesley, 2010*

- **Protéger le domaine métier (la valeur de l’application).**
- **Contenir l’entropie.**
- **Rendre l’application évolutive et pérenne.**

L’architecture logicielle est l’expertise de concevoir des logiciels (ou applications).

Elle permet donc de définir des systèmes, leurs connexions et leurs frontières autour d’un besoin métier, dans l’objectif de faciliter leur évolution.

"Introduction à l'Architecture & la Clean Architecture", Harold Cohen, Youtube TheTribe, 17/11/2021
"What is your definition of software architecture ?", Carnegie Mellon University - Software Engineering Institute, sei.cmu.edu, 22/01/2017
"Adoptez la clean archigonale", Christophe Breheret-Girardin, Youtube BreizhCamp, 22/07/2023

Les enjeux de l’architecture logicielle

Modularité

Scalabilité

Maintenabilité

Performance

Sécurité

"Introduction à l'Architecture & la Clean Architecture", Harold Cohen, Youtube TheTribe, 17/11/2021

"Adoptez la clean archigonale", Christophe Breheret-Girardin, Youtube BreizhCamp, 22/07/2023

"5 key software architecture principles for starting your next project", Andrey Stepanov, Blog ByteMinds, bytemind.co.uk

Comment réaliser ces enjeux ?

Quels sont vos outils ?

Principes S.O.L.I.D

Principes introduits par Robert C.MARTIN en 2002 dans son livre "Agile Software Development, Principles, Patterns, and Practices"

S : Single Responsability Principle

O : Open/Closed Principle

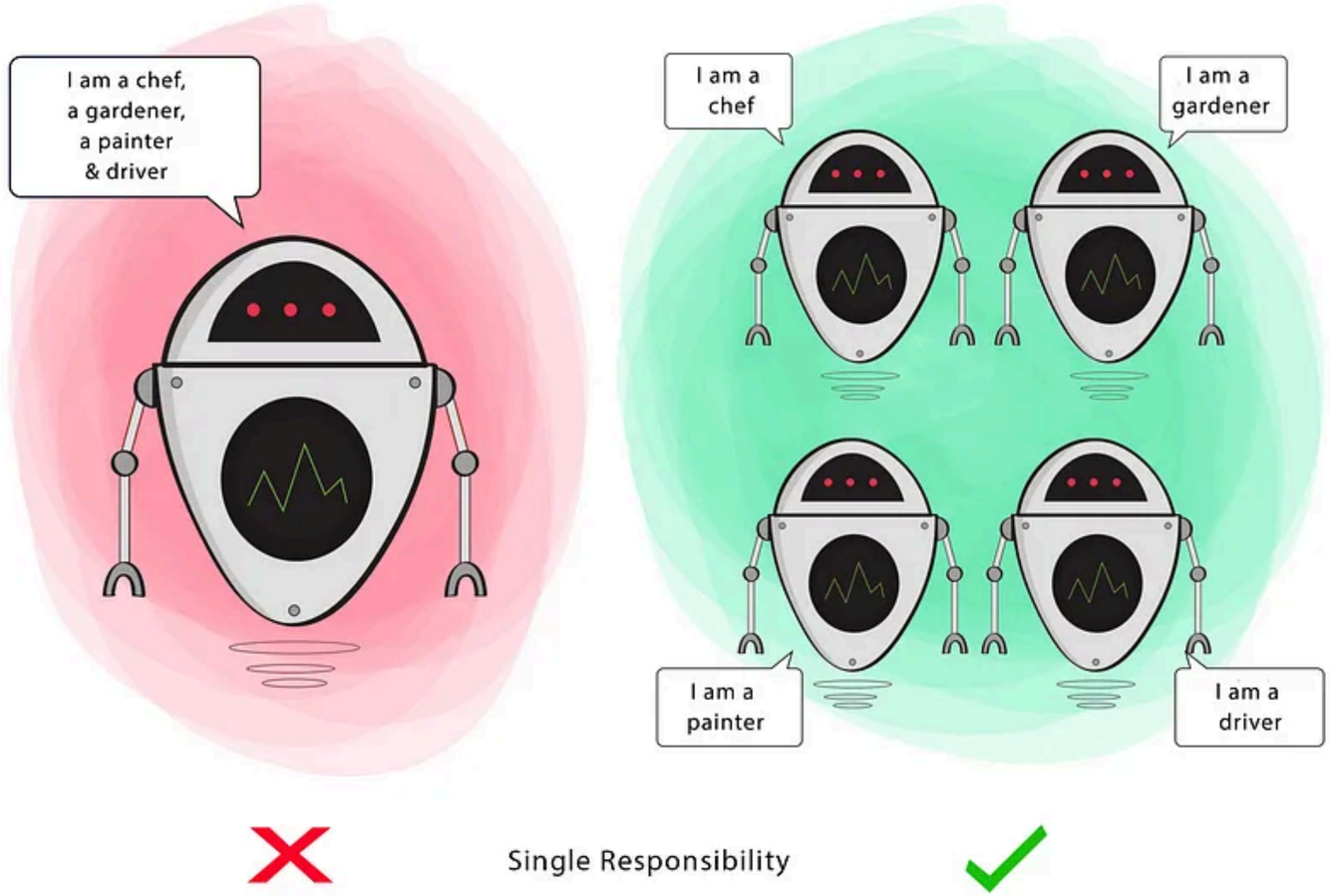
L : Liskov Substitution Principle

I : Interface Segregation Principle

D : Dependency Inversion Principle

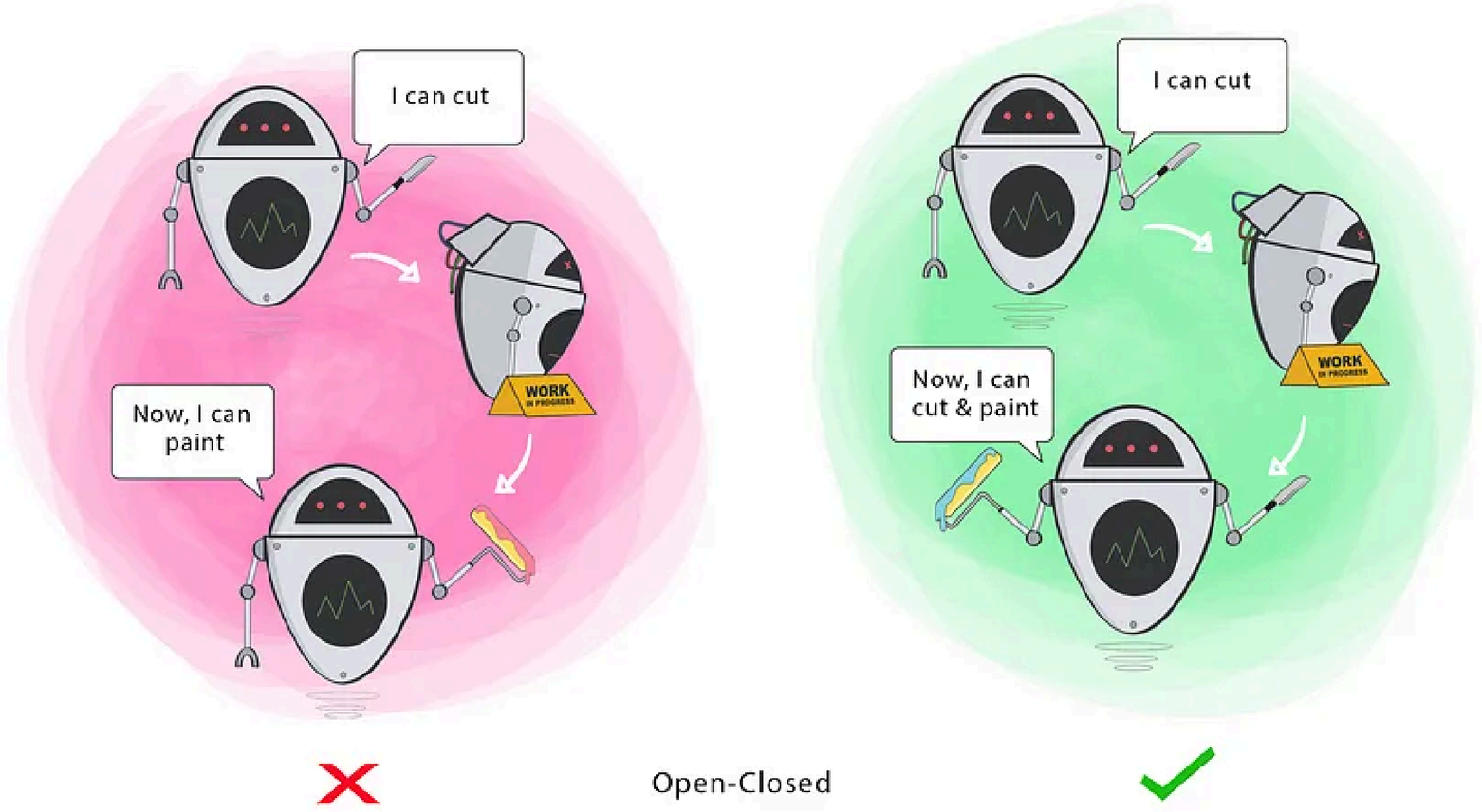
"The S.O.L.I.D Principles in Pictures", Ugonna Thelma, Medium, 18/05/2020

Single Responsability Principle



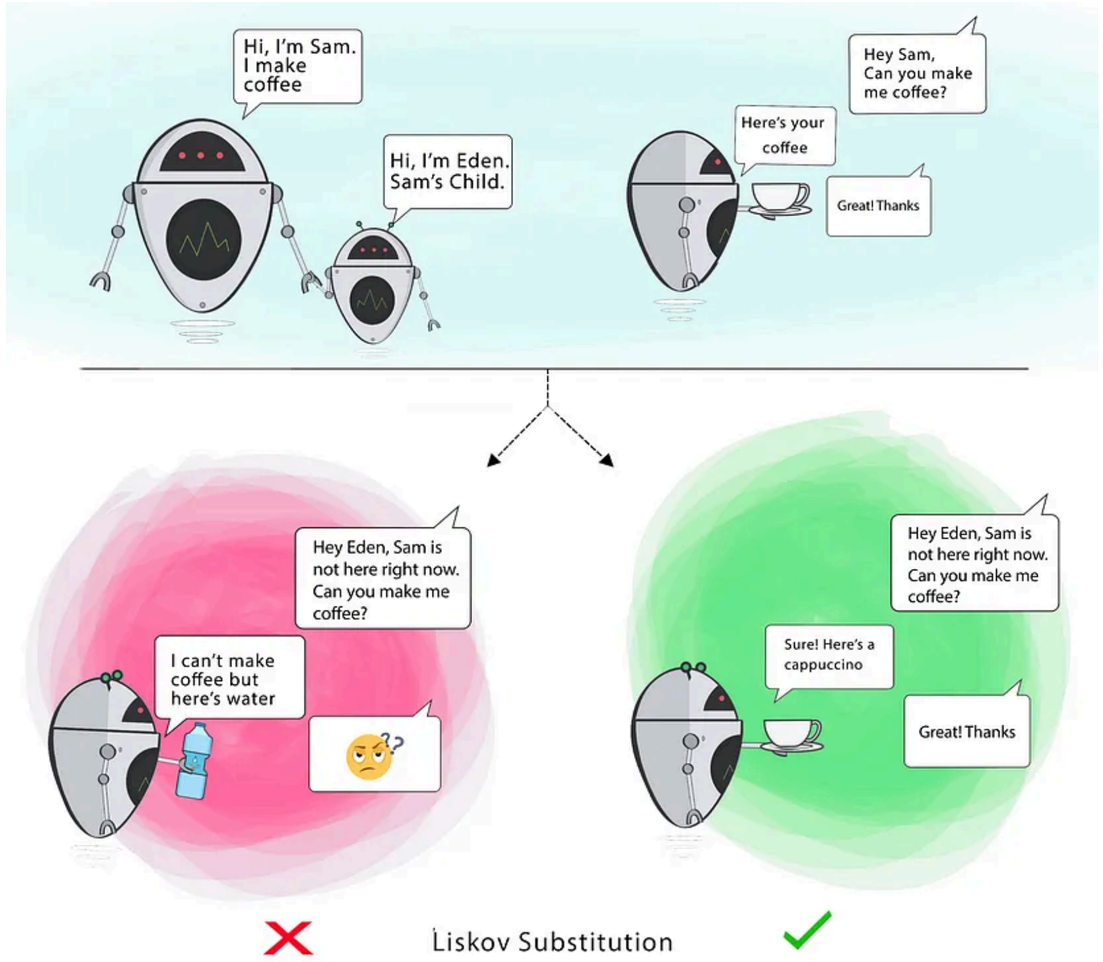
"The S.O.L.I.D Principles in Pictures", Ugonna Thelma, Medium, 18/05/2020

Open/Closed Principle



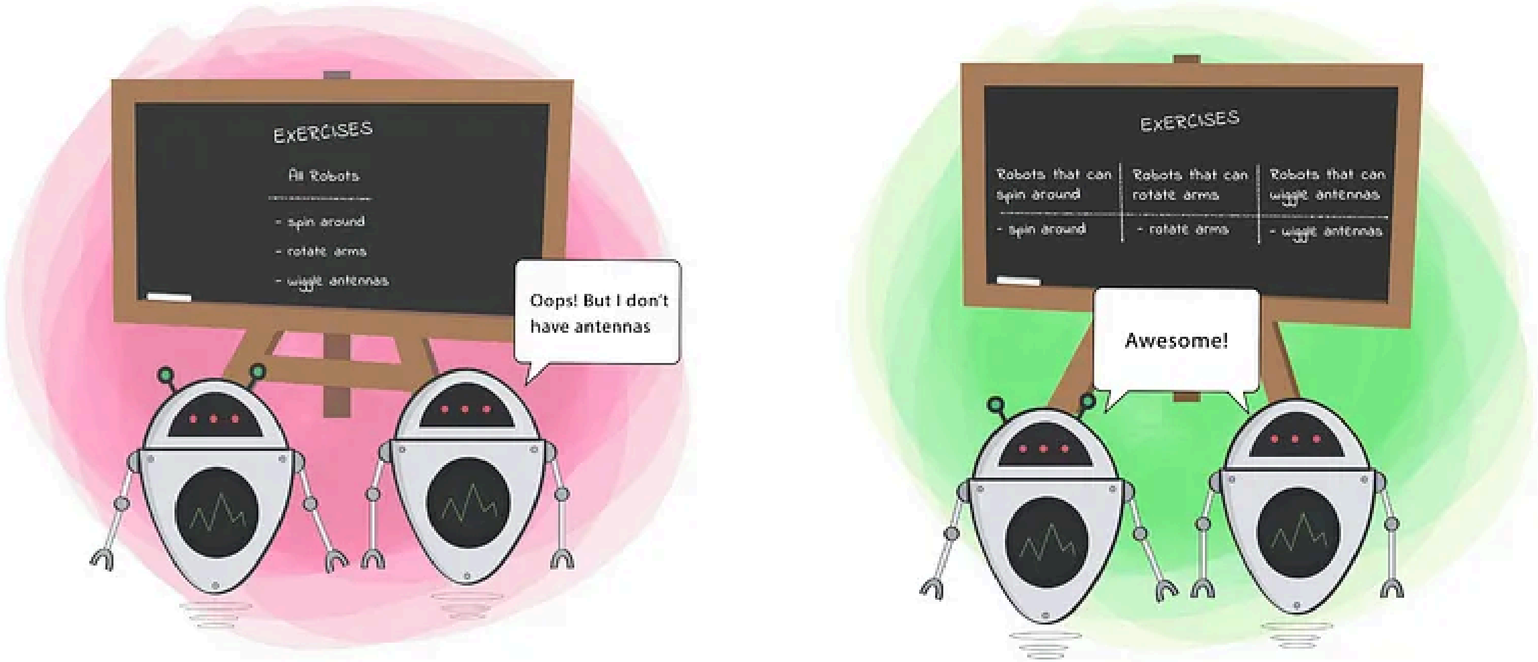
"The S.O.L.I.D Principles in Pictures", Ugonna Thelma, Medium, 18/05/2020

Liskov Substitution Principle



"The S.O.L.I.D Principles in Pictures", Ugonna Thelma, Medium, 18/05/2020

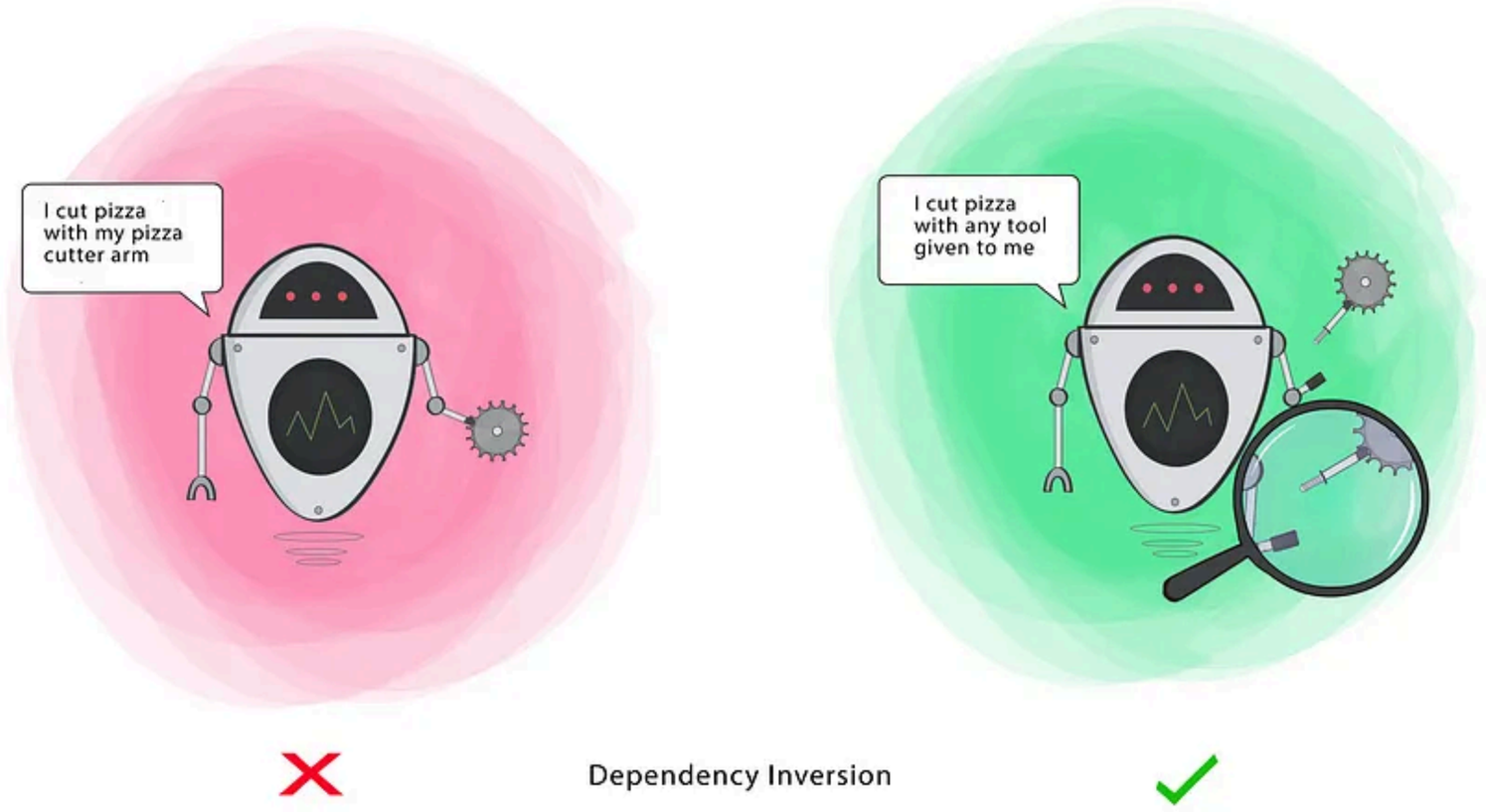
Interface Segregation Principle



Interface Segregation

"The S.O.L.I.D Principles in Pictures", Ugonna Thelma, Medium, 18/05/2020

Dependency Inversion Principle

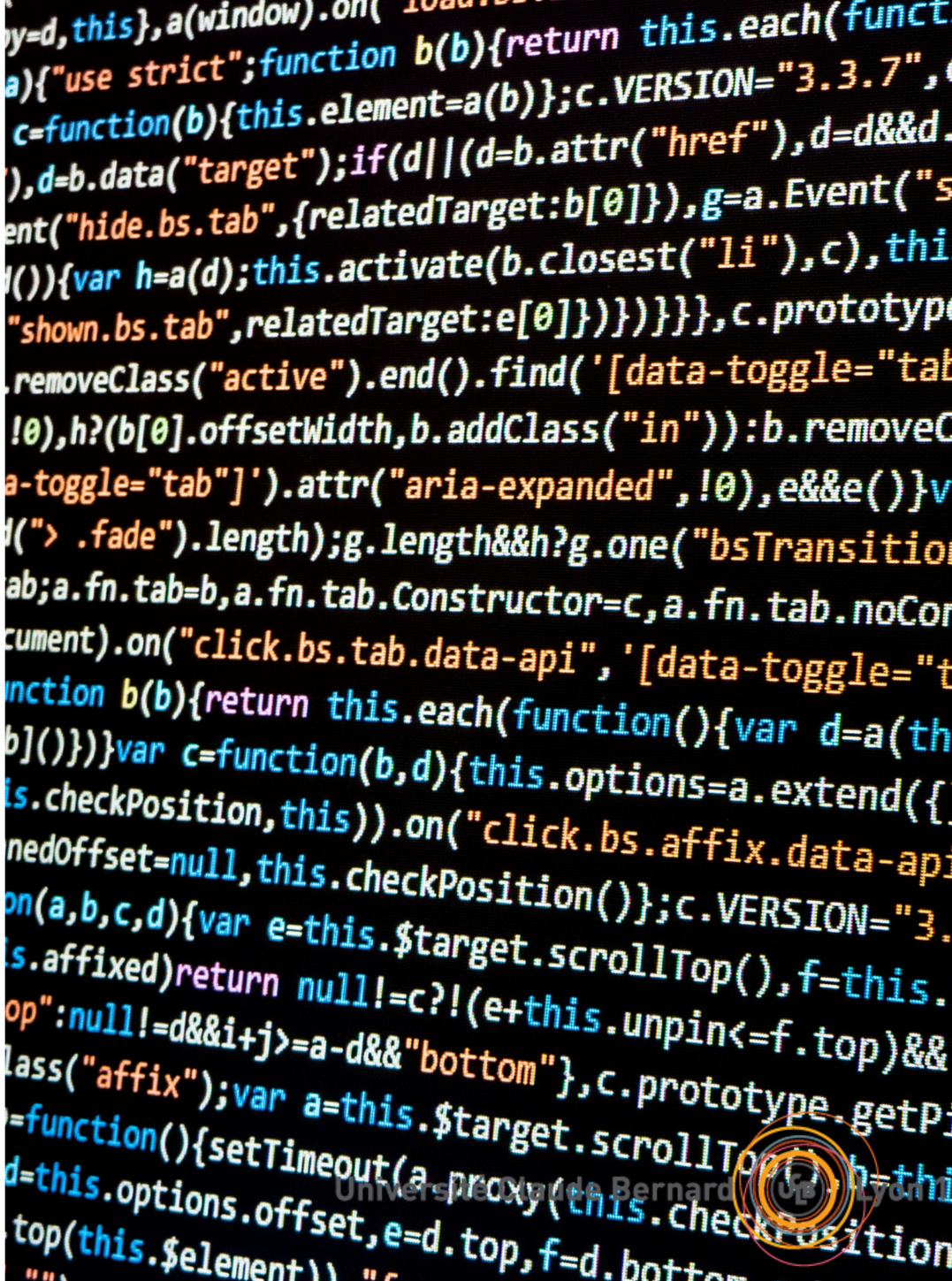


"The S.O.L.I.D Principles in Pictures", Ugonna Thelma, Medium, 18/05/2020

Patron(s) de conception connu(s)

Les patrons de conception utilisés et mis en place lors de vos travaux pratiques universitaires.

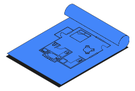
- Singleton/Prototype
- Data Access Object
- Data Transfer Object
- Chain of Responsability
- Dependency injection



"What is Refactoring", Refactoring Guru

Architecture(s) connue(s)

Architectures logicielles utilisées ou mises en place durant vos travaux pratiques universitaires.



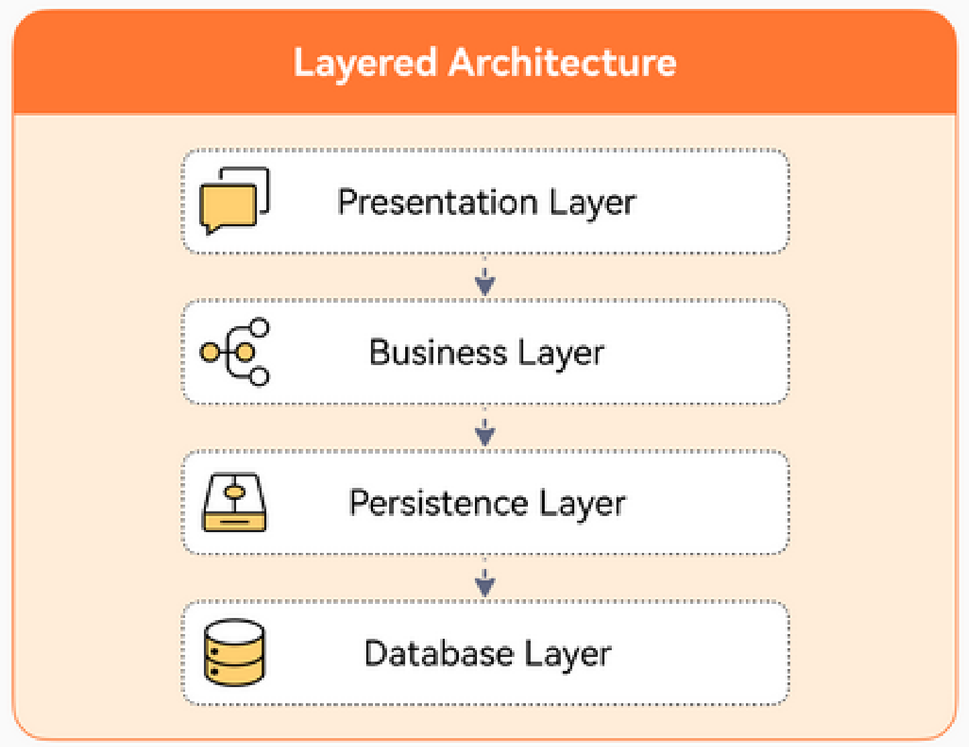
- Architecture Modèle-Vue-Contrôleur
- Architecture en couches

"Layered Architecture (Couches)", Quartz

"What is Refactoring", Refactoring Guru

🎯 Architecture en couches

En réponse, aux problèmes soulevés par des projets ne respectant aucune architecture, ni bonne pratique, l'architecture en couche propose un modèle de séparation des responsabilités. Elle est le modèle d'architecture le plus répandu dans les projets informatiques des années 2000/2010.

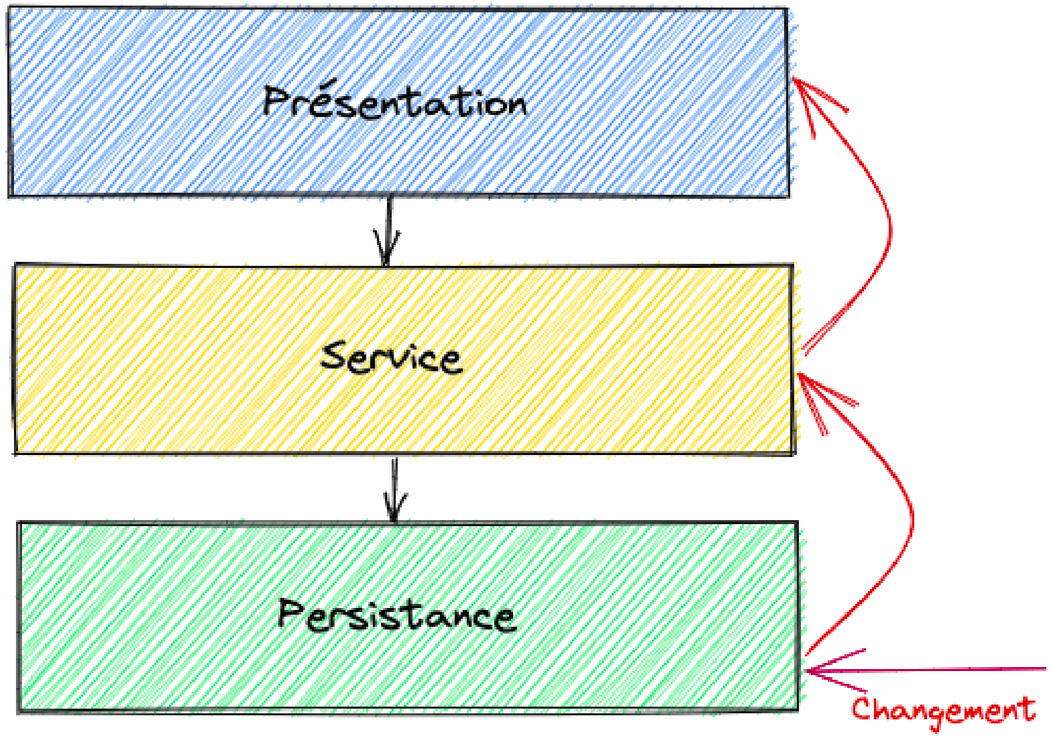


- **Séparation des responsabilités**
- **Structure isolée en couche**

"Layered Architecture (Couches)", Quartz
"L'Architecture En Couches, Toujours La Norme ?", Simon Chareyron, InfoQ, 12/09/2022
"6 Software Architectural Patterns You Must Know ", ByteByteGo

⚠ Problématique(s)

- La couche persistance guide le développement, la technique passe avant le métier.
- Maintenabilité et isolation des couches qui évoluent vers du couplage fort.
- Dans un contexte métier riche, les services finiront par devenir des “God” classes.



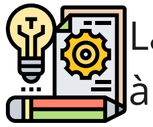
Comment empêcher l’entropie ?

Le contexte d’une entreprise

Notion de valeur

Dans un contexte de projet informatique, la notion de valeur est liée à la notion d'information qui est corrélée aux enjeux du système d'information dans lequel elle évolue :

- Assurer la cohérence des informations
- Construire un réseau informationnel
- Aider à la prise de décision



La donnée est au centre des intérêts, car elle constitue une valeur à laquelle on peut rattacher une information, un sens.

"What is Data vs. What is Information", Sanjay Jain, Blog Bloomfire, bloomfire.com, 10/02/2025
"Data, Information and Knowledge in Information Systems", Jenna Doucet, July 2009

Bibliothèque : Donnée vs Information

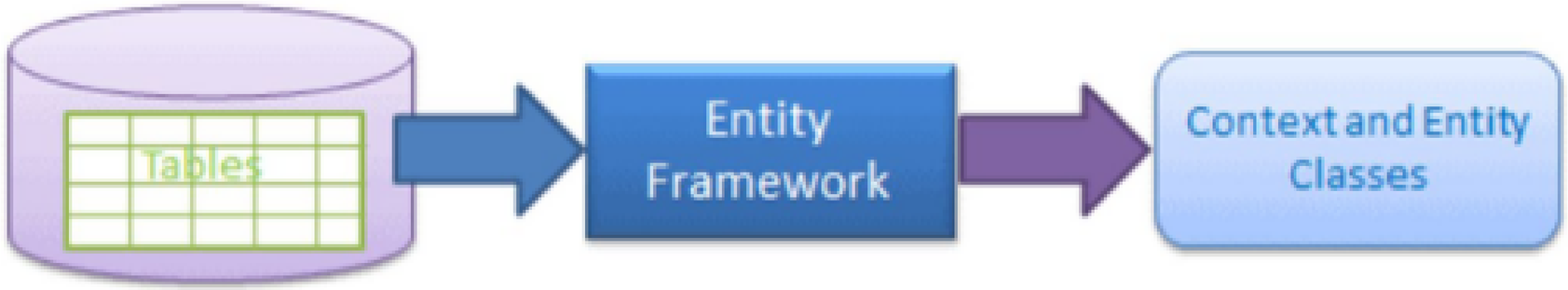
{"titre": "Le Petit Prince", "auteur": "Antoine de Saint-Exupéry", "date_emprunt": "2025-09-21", "date_retour": "2025-10-05"}

Le livre Le Petit Prince est actuellement emprunté par un lecteur et doit être rendu le 5 octobre 2025.

Quel(s) impact(s) sur le développement ?

“Database-First” design

Une philosophie de conception logicielle où la structure des données (les tables, le schéma de la base de donnée) guide la manière de structurer son application.



Par expérience, une majorité des projets dans les années 2000 suivait cette approche de développement.
On élabore des schémas UML, puis on génère les classes à partir de ce schéma.
La donnée, écrite dans notre base de donnée, qui sera exploitée, est donc l’origine de tout.

C’est une philosophie de conception qui n’est adaptée que pour des petits projets où le besoin est essentiellement du CRUD (create, read, update et delete) sur un **domaine métier** (modèle de données) stable et prédictible.

"Development Approaches with Entity Framework", Entity Framework
"Domain-first vs. Schema-first Architecting", Brian Lawler, Medium, 17/08/2016

Où se situe le problème ?

Domain Driven Design

Domain Driven Design

Le Domain Driven Design, D.D.D, est une philosophie de conception décrite dans le livre “**Domain-Driven-Design, Tackling Complexity in the Heart of Software”, publié en 2003 d’Eric Evans**, avec pour objectif que **le domaine métier soit au centre du développement**.
Cela se traduit côté organisationnel, par une collaboration entre les experts métiers et les experts techniques et côté développement, par une recherche de l’isolation de la logique métier (domain métier), elle ne dépendra d’aucune autre composante.

> **“Domain-Driven Design is an approach to software development that centers the development on programming a domain model that has a rich understanding of the processes and rules of a domain. ”**
>
> Martin Fowler

Cette philosophie voit le métier comme la vraie valeur d’une application.

Pour parvenir à faire en sorte, que le code développé reflète au mieux les besoins métiers exprimés, on va utiliser ce qu’Eric Evans décrit comme des **Strategic Patterns** et des **Tactical patterns.**

"Comment aligner votre architecture avec vos besoins métier grâce au Domain Driven Design ?", Nicolas Barlogis, InsideGroupe, 03/09/2024
"DDD : Domain-Driven Design (patterns techniques + stratégiques)", Alex so Yes, 16 mars 2024
"Domain Driven Design", Martin Fowler, 22/04/2020

Anaemic Domain Model vs Rich Domain Model

Anaemic Domain model

Une classe métier qui ne possède pas ou peu de règles métiers. Elle est majoritairement composée de mutateurs (getter/setter). Les règles métiers qui lui sont liées, sont implémentées dans des Services, Helpers, etc... C'est un **modèle orienté service (procedural design)**.

Avantage(s):

- Simple à mettre en place et à utiliser.
- Met en avant la séparation des responsabilités
- Réutilisabilité des services, etc...

Inconvénient(s) :

- Violation du principe d'Encapsulation de la programmation orientée objet
- Augmente le risque de tendre vers un code "Spaghetti"

Rich Domain model

Une classe métier qui porte sa logique métier et ne permets pas l'accès à ses données. Un modèle parfaitement aligné avec le **Domain Driven Design**.

Avantage(s):

- Forte cohésion
- Exprime clairement le métier

Inconvénient(s) :

- Complexité de mise en place
- Violation du principe S.R.P (Single Responsability Principle)

"Rich vs. Anaemic Domain Models in Clean Architecture: Principles, Trade-offs, and Practical Guidance (Part 1)", Alex Tjahjana, Medium, 23/05/2025
"Anemic Domain Model vs Rich Domain Model", Adrien Caubel, 23/03/2025

Bibliothèque : Modèle anémique

```java
public class Book {
    public String title;
    public String author;
    public boolean isAvailable = true;
}

@Service // pour Spring ou @ApplicationScoped pour CDI (ex : Quarkus)
public class BookService {
    public final BookRepository bookRepository;

    public BookService(BookRepository bookRepository){
        this.bookRepository = bookRepository;
    }

    public void emprunterLivre(Livre livre) throws Exception {
        if (!livre.disponible) {
            throw new Exception("Livre déjà emprunté");
        }
        livre.disponible = false;
        bookRepository.emprunterLivre(livre);
    }

    public void retournerLivre(Livre livre) {
        if (livre.disponible) {
            throw new Exception("Livre pas emprunté");
        }
        livre.disponible = true;
        bookRepository.retournerLivre(livre);
    }
}
```

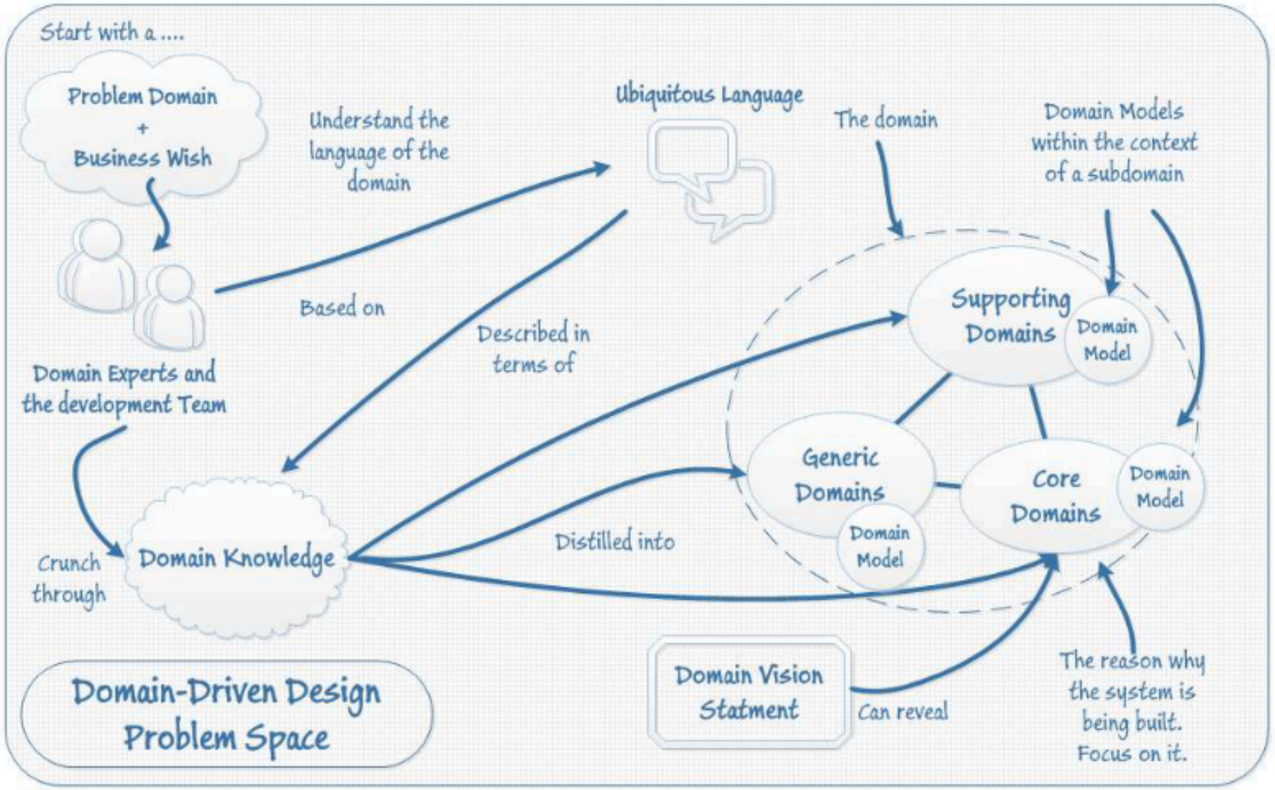
Bibliothèque : Modèle riche

```java
public class Book {
    private String title;
    private String author;
    private boolean isAvailable;

    public Livre(String title, String author) {
        this.title = title;
        this.author = author;
        this.isAvailable = true;
    }

    public boolean isAvailable() {
        return isAvailable;
    }

    public void emprunter() throws Exception {
        if (!disponible) {
            throw new Exception("Livre déjà emprunté");
        }
        disponible = false;
    }

    public void retourner() {
        if (disponible) {
            throw new Exception("Livre pas emprunté");
        }
        disponible = true;
    }
}
```

Vue globale du DDD d’Eric Evans

"Part 1: Domain Driven Design like a pro", Anders Gill, Medium, 09/11/2019

D.D.D : Problem Space



"Part 1: Domain Driven Design like a pro", Anders Gill, Medium, 09/11/2019

"Practicing Domain-Driven design", Scott Millet, 14/01/2014

Patterns stratégiques (1/2)

Langage ubiquitaire

Définit un vocabulaire unifié pour aider à clarifier les concepts et à s'assurer que chacun parle le même langage, qu'il soit technique ou métier.

Contextes délimités

Un contexte délimité représente un concept métier qui répond à une problématique.

Modèle de Domaine

Le modèle de domaine est une abstraction des processus et de la logique métier au sein du logiciel.

"Comprendre le Domain-Driven Design : guide pratique pour les développeurs", Philippe Aubertin, Axopen, 08/01/2025
"DDD : Domain-Driven Design (patterns techniques + stratégiques)", Alex so Yes, 16 mars 2024

Patterns stratégiques (2/2)

Anti-Corruption Layer

Un bounded context utilise les services d'un autre et n'est pas un stakeholder, mais cherche à minimiser l'impact des changements de ce contexte dont il dépend en introduisant un ensemble d'adaptateurs – une couche anti-corruption.

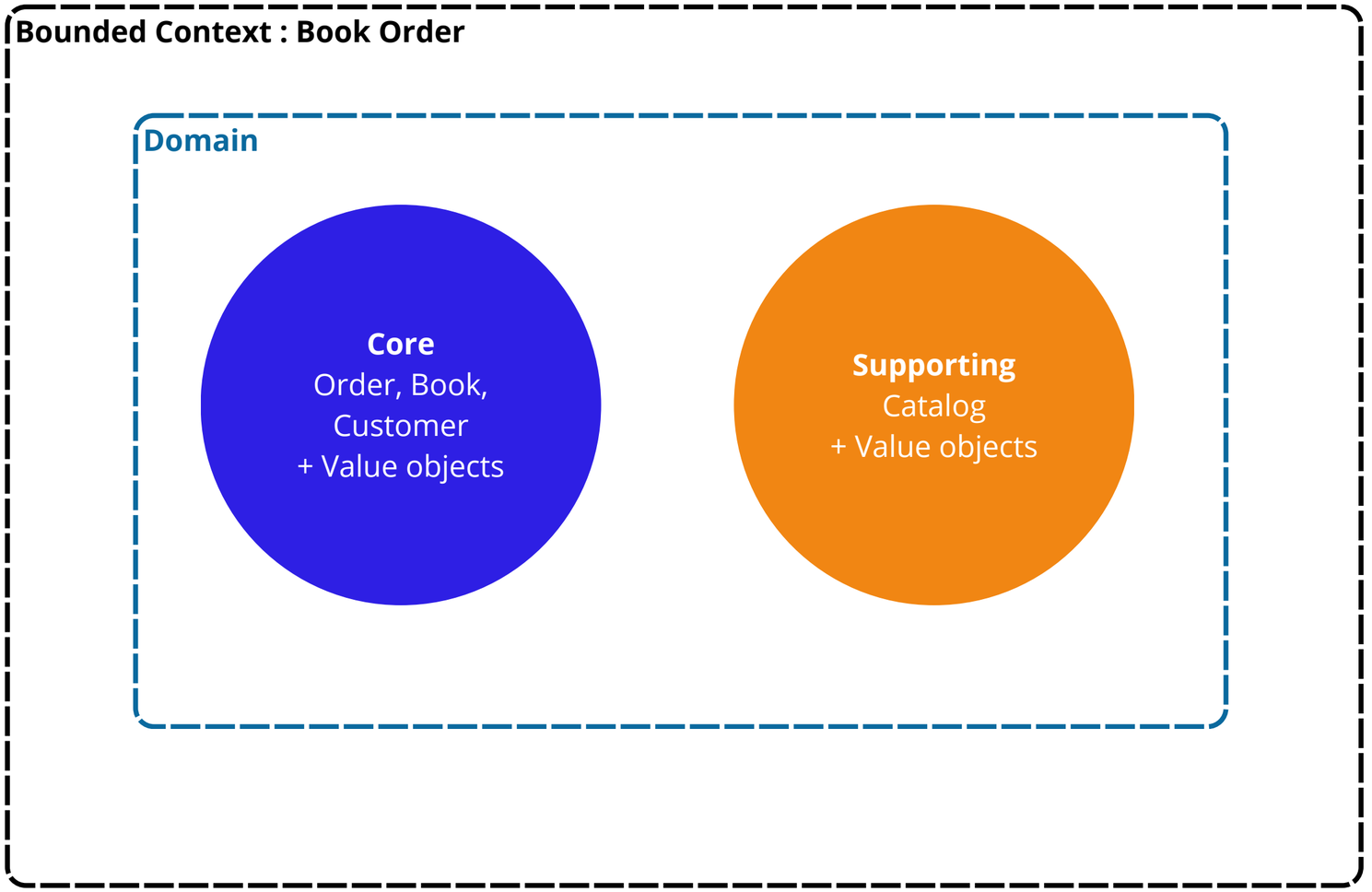
Shared kernel

Deux bounded contexts partagent un noyau commun de code (par exemple une librairie) qui sert de langage commun, mais pour le reste, chacun fait les choses à sa manière.

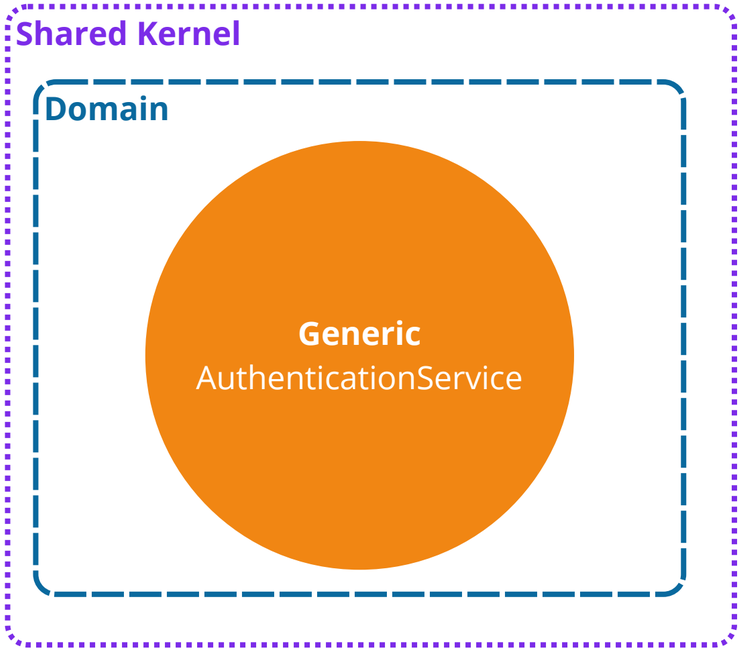
"Activity/Technique: Strategic Domain-Driven Design (DDD)", Olaf Zimmermann

"DDD : Domain-Driven Design (patterns techniques + stratégiques)", Alex so Yes, 16 mars 2024

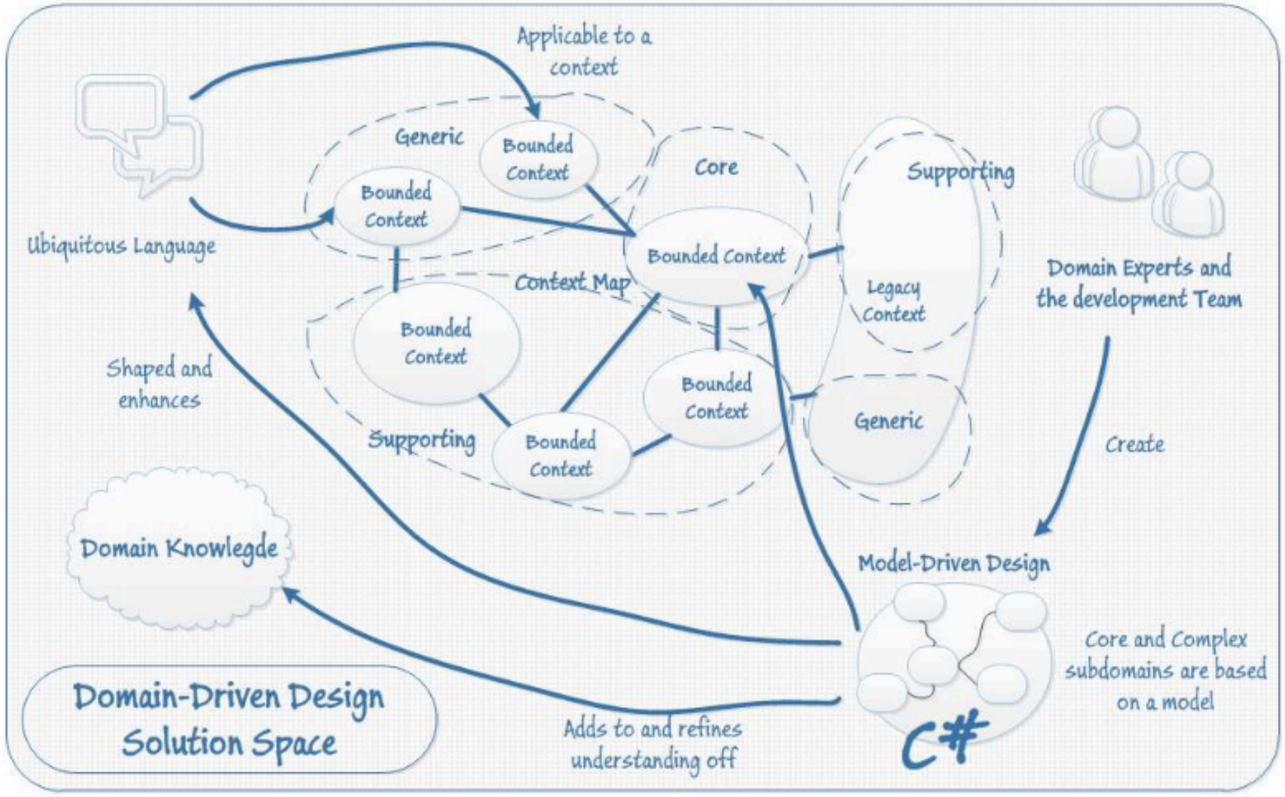
Bibliothèque : Context Map




```java
public class Catalog {
    private List<Book> books;

    public List<Book> searchByAuthor(String author) {
        return books.stream()
                    .filter(b -> b.getAuthor().equals(author))
                    .collect(Collectors.toList());
    }
}
```



D.D.D : Solution Space



"Part 1: Domain Driven Design like a pro", Anders Gill, Medium, 09/11/2019

"Practicing Domain-Driven design", Scott Millet, 14/01/2014

Patterns tactiques

Entity

Objets ayant une identité unique, traqués tout au long de leur cycle de vie.

Value Object

Objets sans identité propre, définis par leurs attributs.

Aggregate

Groupes d'entités et de valeurs, traités comme une unité cohérente.

"Comprendre le Domain-Driven Design : guide pratique pour les développeurs", Philippe Aubertin, Axopen, 08/01/2025
"DDD : Domain-Driven Design (patterns techniques + stratégiques)", Alex so Yes, 16 mars 2024

Bibliothèque : Contexte Order

```java
public class Customer {
    private CustomerName name;
}
public record CustomerName (String firstName, String lastName){}

public class Book {
    private Title title;
    private AuthorName author;
    private boolean isAvailable;
    /**
     * Logique métier
     */
}
public record Title (String title){}
public record AuthorName (String name){}

public class Order {
    private Book book;
    private Customer customer;
    private LocalDate borrowDate;

    public Order(Book book, Customer customer){
        if (!book.isAvailable()) throw new Exception("Livre déjà emprunté");
        this.book = book;
        this.customer = customer;
        this.borrowDate = LocalDate.now();
        book.emprunter();
    }
}
```

```java
public class OrderService {
    private BookRepository bookRepository;
    private CustomerRepository CustomerRepository;
    private OrderRepository orderRepository;

    public void placeOrder(Long bookId, Long customerId) throws Exception {
        Book book = bookRepository.findById(bookId);
        Customer customer = customerRepository.findById(customerId);
        Order order = new Order(book, user);
        orderRepository.save(order);
    }
}
```

Value objects :

CustomerName, Title et AuthorName

Domain:

Book

Entities:

Customer et Order

Aggregate:

Order

Avantage(s) & Inconvénient(s)

Avantage(s)

- Facilité d’échange avec les utilisateurs et les experts techniques.
- Facilité aux nouveaux développeurs d’appréhender l’aspect fonctionnel du projet avec le code.
- Facilité pour tester le code et le maintenir.

Inconvénient(s)

- Manque de liberté de conception, le code est guidé par le métier.
- Peu adapté à une équipe de développeurs peu expérimentés.
- Un coût plus élevé de développement, le code est plus long à écrire (beaucoup de composants et de réflexion).
- Demande de l’investissement pour tous les membres de l’équipe.

"Domain-Driven Design : quel apport dans une architecture microservices ?", SoftFluent

"DDD : Domain-Driven Design (patterns techniques + stratégiques)", Alex so Yes, 16 mars 2024

Quel(s) impact(s) sur le choix de l’architecture ?

Architecture hexagonale

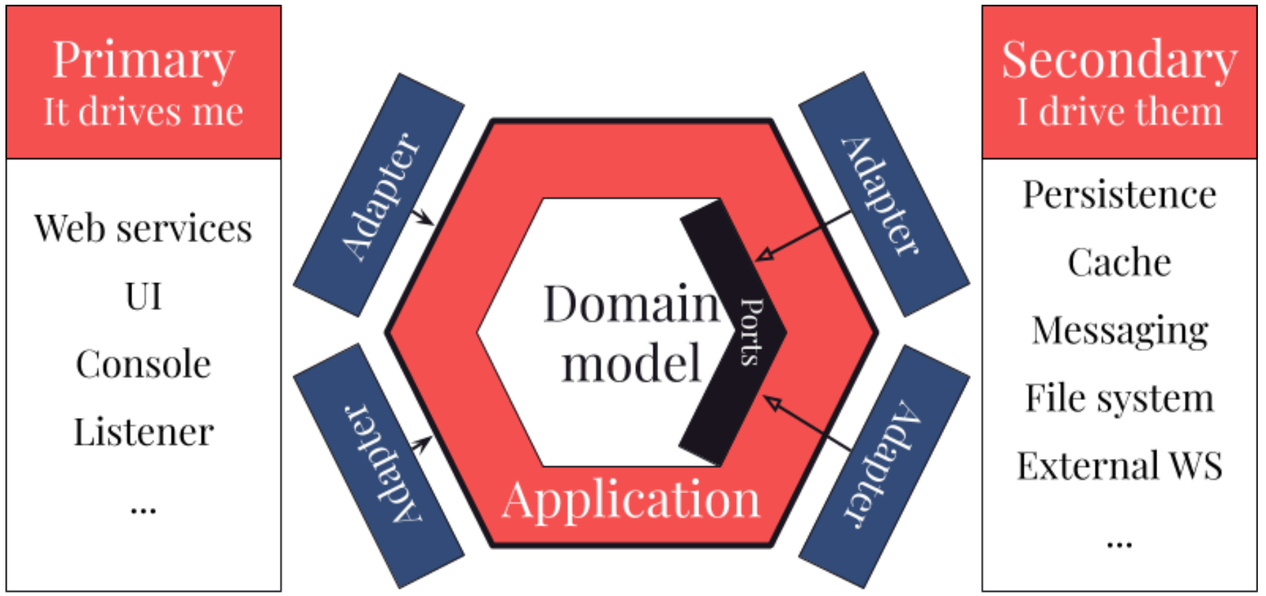
Architecture hexagonale

L'architecture hexagonale, décrite par **Alistair Cockburn en 2005**, vise à découpler la logique métier des détails techniques.
Elle repose sur le principe que **toutes les dépendances doivent pointer vers le domaine** (inversion de dépendances).

Le domaine définit des ports (interfaces), qui sont implémentés par des adapters situés en périphérie :

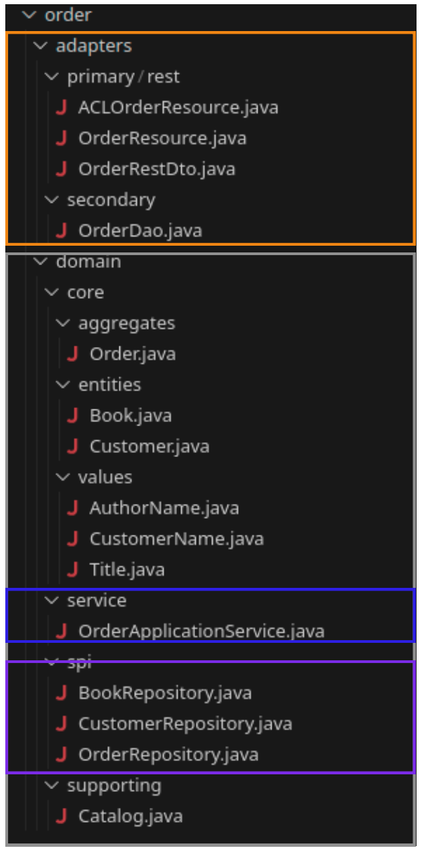
- **Adapters primaires (driving adapters - API)** : ceux qui déclenchent un cas d'usage (ex: REST).
- **Adapters secondaires (driven adapters - SPI)** : ceux qui fournissent au domaine les services techniques dont il a besoin (ex: DB).

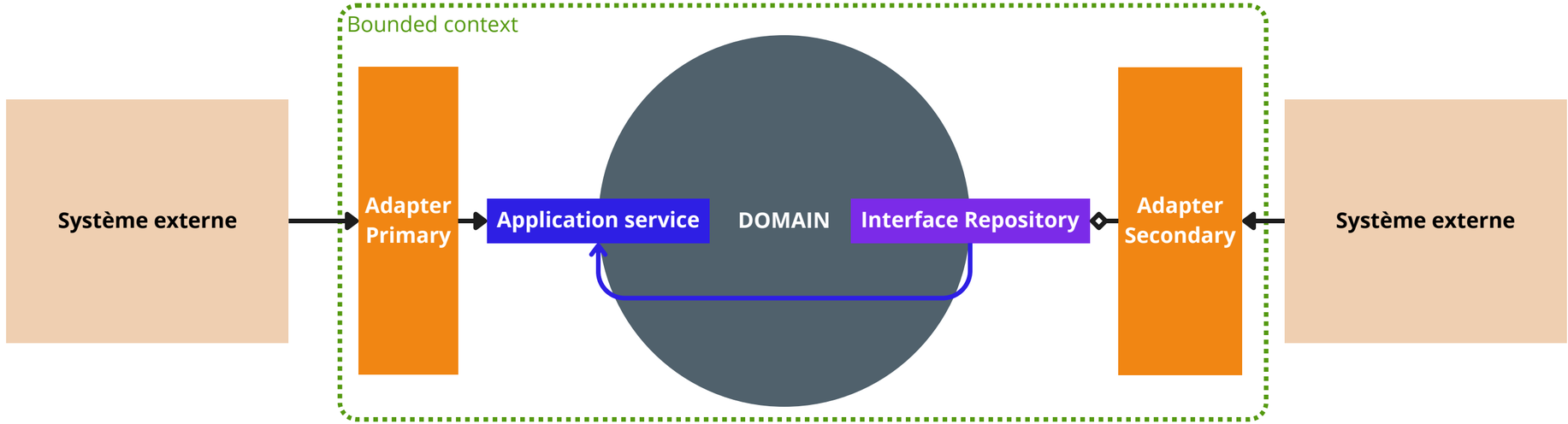
Cette architecture, souvent décrite comme un **"pattern Ports & Adapters"**, est particulièrement adaptée au Domain Driven Design, car elle protège le domaine des dépendances techniques.



"Spring Boot Java dans l'hexagone", Colin Damon, Ippon, 17/02/2021
"Architecture Hexagonale et Clean Architecture", TheTribe, 04/08/2022

Bibliothèque : Mise en place d’un hexagone





Avantage(s) & Inconvénient(s)

Avantage(s)

- Facilité pour tester le code et le maintenir.
- Ouverture du code à l’évolution.
- Retarder la décision des choix technologiques.

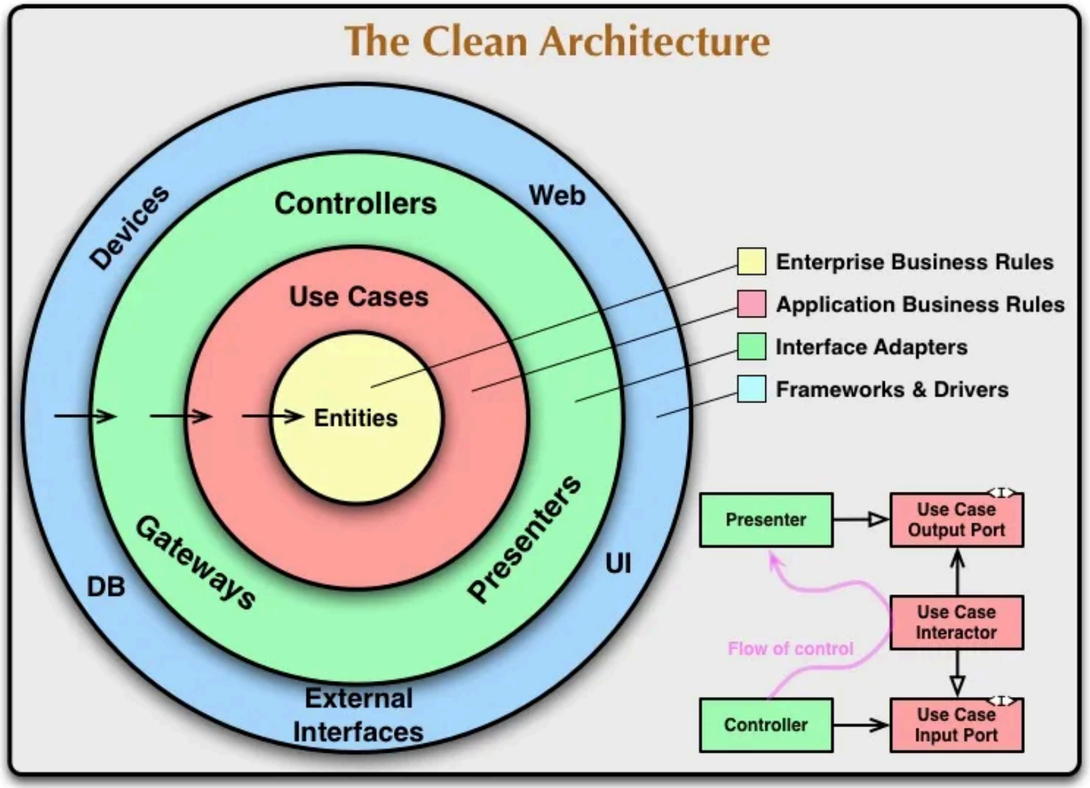
Inconvénient(s)

- Complexité amené par la création de beaucoup de classe pour découpler.

"Hexagonal Architecture: What Is It and Why Do You Need It?", Alex Hyett, 17/02/2023
"Architecture hexagonale : Avantages/Inconvénients", Quartz

Pour aller plus loin…

Clean Architecture



"Clean Architecture Part One: What is Clean Architecture?",Cagri Tezel, Medium, 06/01/2023
"The Clean Architecture", Robert C.Martin, The Clean Code Blog, 13/08/2012

Question(s) ?