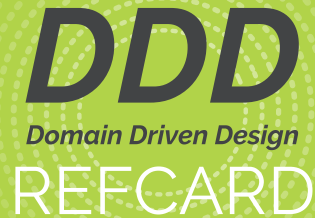


Sepehr NAMDAR FARD & Romain DENEAU

Juillet 2018

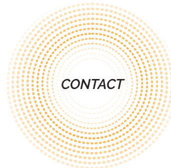
Cabinet de conseil IT et Agilité | SOAT

Problématiques de conception des logiciels complexes

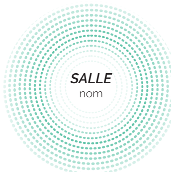
Absence de langage commun

L'ÉQUIPE DE DÉVELOPPEMENT — **L'EXPERT MÉTIER**

VS



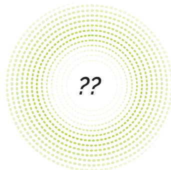
La même signification mais des termes différents



VS

Même terme mais des significations différentes

VS



Terme inconnu du métier

Architecture avant tout technique

- src
  - controller
    - EntretienController.java
  - dao
    - EntretienDao.java
  - dto
    - EntretienDto.java
  - service
    - EntretienService.java

⚠ Métier relégué en seconde zone

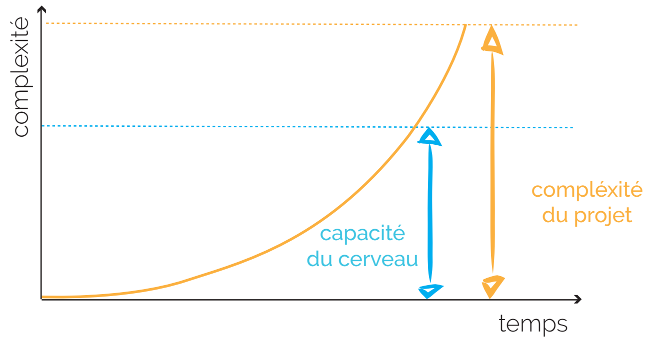
> *"We're not code writers, we're problem solvers.*"*
> Michael Feathers

Une bonne communication est la clé pour réussir un projet.

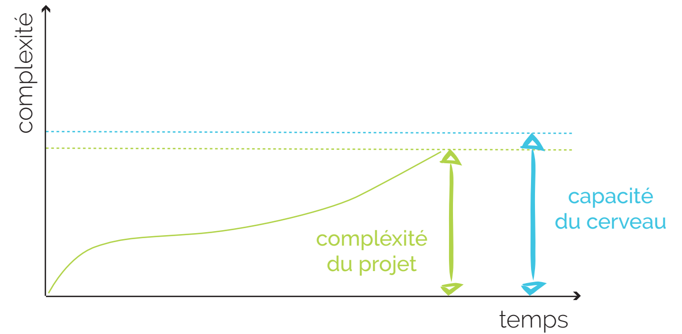
Nous ne faisons pas qu'écrire du code, nous apportons des solutions.

Comment DDD peut nous aider

La complexité sans DDD



La complexité avec DDD



DDD est une aide pour :

- ✓ Garder la complexité à un niveau abordable et compréhensible.
- ✓ Conserver un produit maintenable dans le temps.

Outils adaptés selon la complexité

> *"Critical complexity of most software projects is in understanding the domain itself*"*
>
> Eric Evans

Complexité essentielle → domaine métier**

Knowlegde Crunching

- **Quoi :**
  - Collecte des connaissances métier pertinentes
  - En relation avec le projet
- **Comment :**
  - Collaboration entre développeurs et experts métier
  - Discussions autour du métier
  - Dans la durée
- **Exemples :**
  - User Story Telling, User Story Mapping
  - Event Storming d'*Alberto Brandolini*

Création de l'Ubiquitous Language du projet

→ Terminologie unique, précise, non équivoque et partagée

Découpage du domaine

→ En plusieurs parties, en sous-domaines

Complexité accidentelle → technique, legacy

Architecture centrée sur le domaine

- **Objectif :**
  - Isoler le domaine métier
  - Le séparer des problématiques techniques
- **Exemples :**
  - Ports and Adapters d'*Alistair Cockburn*
  - Clean architecture de *Robert C. Martin*

Décomposition en composants découplés

→ Design patterns DDD : Aggregate, Value Object, Entity…

Utilisation de l'Ubiquituous Language

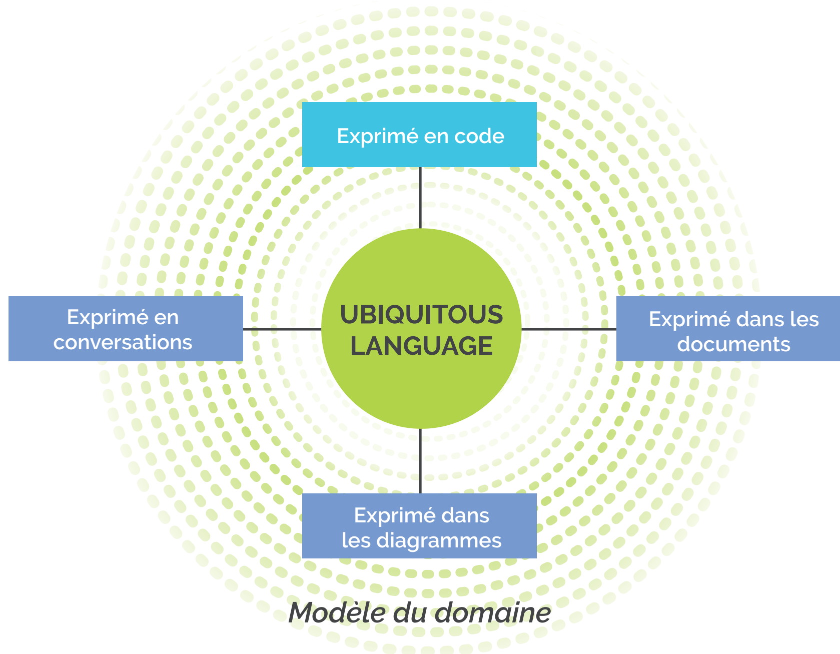
→ À tous les niveaux dans le code : nom des packages/ namespaces, des classes/interfaces, des méthodes, des propriétés, des événements, des variables traitant du métier

---

* *La complexité critique de la plupart des projets IT réside dans la compréhension du domaine fonctionnel.*

** *Sauce secrète, terminologie riche, processus complexes*

Création d’un langage commun



- Langage parlé par tous les acteurs du projet (Devs, Product Owner, Stakeholders, Commerciaux, etc.)
- Implémenté dans la partie technique (le code, les tests, etc.)
- Fédère les acteurs du projet
- Des méthodes de *Knowledge Crunching* peuvent nous aider à le construire

Prérequis à la mise en place du DDD

Investissements importants

ARGENT & TEMPS

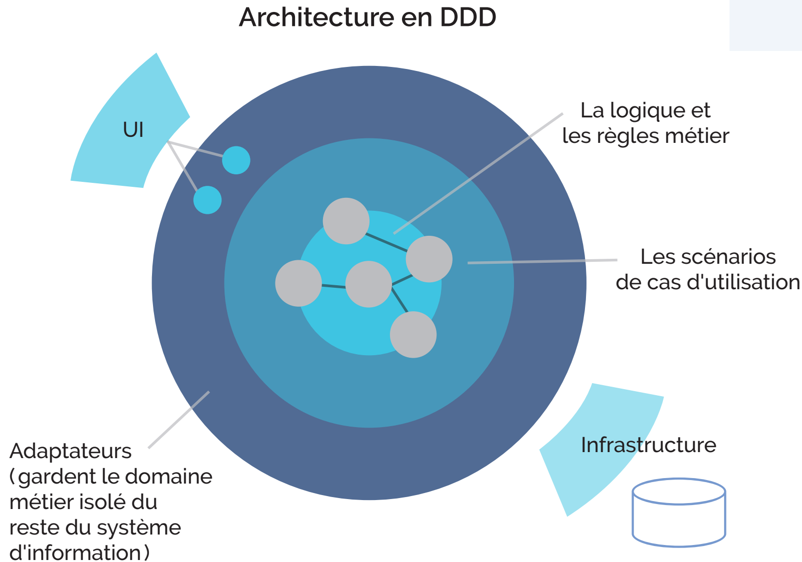
- ✓ Complexité métier
- ✓ Experts métier disponibles
- ✓ Produit évolutif

- ✗ Métier “simple”
- ✗ Absence de sachants
- ✗ Logiciel “one shot”

DDD n’est pas une solution miracle

Une application conçue avec DDD

> *"Any fool can write code that a computer can understand. Good programmer writes code that humans can understand.*[*]*"*
>
> Martin Fowler,
> Refactoring : improving the Design of Existing Code




```
◢ src
    ▷ infrastructure
    ◢ model
        ▷ Candidat.java
        ▷ ConsultantRecruteur.java
        ▷ Creneau.java
        ▷ Entretien.java
    ◢ use_case
        ▷ AnnulerEntretien.java
        ▷ ConfirmerEntretien.java
        ▷ PlanifierEntretien.java
        ▷ ReporterEntretien.java
  test
```

- ✔ Architecture centrée sur le domaine
- ✔ Le code et son organisation sont compréhensibles par les experts métier

[*] *N'importe quel idiot peut écrire du code qu'un ordinateur peut comprendre. Un bon programmeur écrit du code que les humains peuvent comprendre.*

Exemple de pseudo-code en DDD

```
@Scheduler(Chaque.Jour)
const verifierPropaleLe = (propale: Propale, dateJour: Date) => {
   if (propale.estAcceptee()) {
       declencherProcessusSignature(propale)
   }
   else if (propale.estRefusee()) {
       ajouterPropaleDansLeVivier(propale)
   }
   else if (propale.estEchue(dateJour)) {
       notifierChargeeRecrutement(propale)
   }
}

class Propale {
   dateEnvoi: Date
   dureeValidite = 10.Jours
   statut: StatutPropale

   boolean estAcceptee() => statut == StatutPropale.Acceptee
   boolean estRefusee()  => statut == StatutPropale.Refusee
   boolean estEchue(dateJour: Date) =>
       dateJour > dateEnvoi + dureeValidite
```

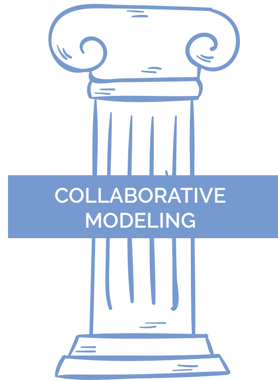
✓ Ubiquitous language du recrutement chez SOAT
- **Chargées de recrutement** : les utilisateurs
- **Propale** : proposition commerciale
- **Vivier** : base de données de CV de candidats

✓ Séparation par aspect
- @Scheduler → AOP

✓ Classe Propale
- ValueObject
- Immuable
- Contient les règles métier associées

✓ DDD agnostique au paradigme de programmation
- Programmation fonctionnelle
- Programmation orientée-objet
- Etc.

Les 3 piliers du DDD



Collaborative Modeling

- Tous les acteurs sont impliqués
- Comprendre et pouvoir juger de la pertinence des informations
- Plusieurs sessions collaboratives et espacées
- Ubiquitous language
- Recherche de sous-domaines

Strategic Design

- Décomposition en sous-modèles : Bounded Contexts
- Établir le modèle adapté
- Définir le Context Mapping

Tactical Design

- Création d'un modèle de domaine efficace
- Building blocks :
  - *Value Objects, Entities, Aggregates*
  - *Repositories, Domain Services, Application Services, Domain Events...*

Conclusion

DDD est une approche de développement logiciel qui remet les besoins métiers au cœur des préoccupations logicielles et répond particulièrement bien aux différentes problématiques rencontrées sur des projets avec des domaines complexes.

Elle est **composée d'un ensemble de principes, de pratiques et de patterns** et est **fondée sur la collaboration des équipes** pour un besoin de maintenir les applications plus facilement et à long terme.

DDD n'est pas une technologie, un framework ou une méthode, et n'est pas adapté à toutes les problématiques. Sa mise en place est coûteuse et nécessite un investissement important en temps et en argent.

La communauté DDD est **en pleine évolution.** De plus en plus de clients se tournent vers cette solution et la prennent comme un investissement à long terme.

Les solutions d'architecture **avec une ampleur métier** comme *CQRS*, *Event Sourcing*, *Microservices* et *Reactive Architecture* gravitent autour de DDD grâce à sa focalisation sur la **compréhension des problématiques métier** et tout ce qu'elle peut apporter.

La communauté Software Craftsmanship et architecture de SOAT serait heureuse de vous accompagner autour de ces sujets, par exemple sur des formations, et vous invite à regarder nos autres publications correspondant à ces sujets qui nous tiennent à cœur.

NOTES

NOTES

Sepehr NAMDAR FARD
Développeur Java et Craftsman - SOAT

Ayant eu l'occasion de travailler sur des missions variées, que ce soit de la pure maintenance ou le suivi complet d'un projet, j'ai pu acquérir des connaissances sur toutes les phases d'un développement logiciel. La réussite d'un projet dépend en grande partie des méthodologies appliquées lors de son déroulement. Celles-ci sont souvent l'Agilité, le Lean, le Behaviour-Driven Development et le Clean Code.

Romain DENEAU
Développeur senior full stack .Net TypeScript - SOAT

Un temps passé chef de projet, je suis désormais revenu à la passion qui m'anime depuis quasiment 18 ans, le développement. Craftsman convaincu, je participe au partage des bonnes pratiques tant Front (TypeScript) que Back (C#) via des articles et des MasterClass Clean Code

SOAT
89 quai Panhard et Levassor 75013 Paris
tèl. : + (33) 1.44.75.42.55
contact@soat.fr - www.soat.fr

Retrouvez-nous sur

Cabinet de conseil IT et Agilité | SOAT