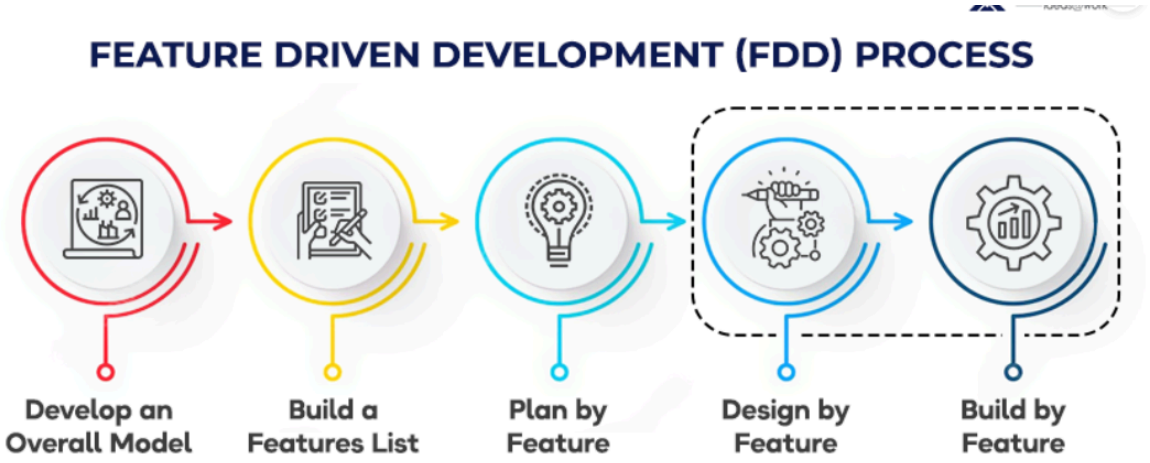
Feature-Driven Development (FDD)

Métaphore **maison**. Au lieu de construire toute la maison en une seule fois, vous commencez par les éléments essentiels (fondations, puis les murs, les toits et l'intérieur). Idem en Feature Driven Development (FDD) - une approche étape par étape qui se concentre sur une caractéristique spécifique à la fois et qui progresse graduellement vers le produit complet.
Méthode agile utilisée en info repose développement **itératif** et **incrémental** et **cycle très court**. Pratique pour des **gros projets** avec une **grande équipe**

D'ailleurs conçu par Jeff De Luca pour mission de 15 mois, 50 personnes en 1997 pour un projet de développement de logiciel dans une grande banque de Singapour.
Résultat : 5 processus



1. Développer un **modèle global**
Avec une compréhension claire de la vision du produit et de la portée du projet, l'architecte en chef et les membres de l'équipe collaborent pour créer un modèle d'objet qui répond au problème du domaine. Ce modèle sert de base fondamentale sur laquelle les systèmes d'application seront construits et les fonctionnalités seront développées au fur et à mesure.

2. Dresser la **liste des fonctionnalités**
Les membres de l'équipe de projet collaborent et dressent la liste des « caractéristiques » en tenant compte de la vision du produit et de la valeur qu'il offre aux utilisateurs potentiels. Ces caractéristiques doivent être traduites en petites tâches gérables pour les développeurs, qu'ils doivent être en mesure de réaliser en 2 à 10 jours. (très différent de SCRUM dans laquelle un sprint dure entre deux et quatre semaines. Si une feature nécessite plus de deux semaines, elle est divisée en plusieurs features jusqu'à atteindre moins de deux semaines par feature)

3. **Planifier par fonctionnalité**
Les fonctionnalités sont maintenant évaluées en tenant compte de divers facteurs tels que la création de valeur, la largeur de bande des ressources nécessaires, les délais prévus, les

risques et les dépendances. Par la suite, les fonctionnalités sont organisées de manière appropriée et attribuées à des équipes de développeurs spécifiques appelées « feature owners ».

4. **Conception par fonctionnalité**
Le programmeur en chef analyse et finalise les fonctionnalités qui doivent être conçues et construites en priorité. Des paquets de conception pour chaque fonctionnalité seront créés et examinés en vue du développement et contribueront également à affiner le modèle global.

5. **Construction par fonctionnalité**
C'est à ce stade que se déroulent toutes les activités de construction des fonctionnalités, telles que le développement, l'intégration et les tests. Le programmeur en chef examine ensuite la fonctionnalité pour s'assurer qu'elle est conforme aux objectifs ou qu'elle nécessite des itérations. La fonctionnalité sera alors ajoutée à la version principale.

Membres de l'équipes :
- Chef projet : superviseur
- Architecte en chef : conception globale et modélisation système
- Responsable du dev : dirige et encadre l'équipe de dev, supervise les act de programmation quotidiennes
- Programmeur principal : participe à l'analyse et conception, peut être chargé de diriger de pttes équipes de dev
- Proprio de la classe : membre des pttes équipes de dev. conception, codage, teste et documentation des fonctionnalités
- Expert en domaine : membre d'1 équipe qui comprend le pb que le client doit résoudre. appui pour les dev en terme de connaissances pour s'assurer qu'ils oeuvrent et fournissent les élémts qui comptent le plus pour le client

Avantages :
- Donne à l'équipe une très bonne compréhension de la portée et du contexte du projet.
- Nécessite **moins de réunions** qui est remplacée par de la documentation.
- Utilise une approche **centrée sur l'utilisateur**. Avec le FDD, le client est l'utilisateur final contrairement à la méthodologie Scrum où le product owner est l'utilisateur final.
- Fonctionne bien avec des **projets à grande échelle**, à long terme ou en cours.
- Le suivi et la **correction des bugs** est facilitée
- T**ravail efficace** par petites étape

Inconvénients :
- pas idéal pour les **petits projets** car il est difficile pour un petit groupe de personnes d'assumer les différents rôles.
- **Dépendance accrue sur le programmeur principal** qui a plusieurs casquettes.

- Ne fournit aucune documentation écrite au client, bien qu'il y ait beaucoup de communication documentée entre les membres de l'équipe pendant les cycles de développement du projet.
- Met l'accent sur la **propriété individuelle du code** au lieu d'une propriété partagée par l'équipe.

Bonnes pratiques :

- Modélisation objet de domaine : les équipes créent des diagrammes de classes pour décrire les objets d'un domaine et leurs associations. Ce processus vous fait gagner du temps en vous aidant à identifier la fonction à ajouter pour chaque élément.
- Développement par fonctionnalité : si une fonctionnalité ne peut être mise en œuvre en deux semaines, elle doit être décomposée en éléments plus petits et gérables.
- Propriété individuelle d'une classe (d'un code) : chaque classe ou groupe de codes est attribué à un seul propriétaire.
- Équipes de fonctionnalité : bien qu'une personne soit responsable des performances et de la qualité de chaque classe, une fonctionnalité peut impliquer plus d'une classe, de sorte que tous les membres de l'équipe de fonctionnalité contribuent aux décisions concernant la conception et la mise en œuvre.
- Inspections : les équipes FDD effectuent des inspections pour détecter les défauts et garantir la meilleure qualité.
- Gestion de la configuration : cette pratique consiste à identifier le code source de toutes les fonctionnalités et à en consigner les modifications.
- Calendrier de réalisation périodique : cette bonne pratique permettra à l'équipe de disposer en permanence d'un système à jour qu'elle pourra présenter au client.

- Rapports d’avancement : les chefs de projet doivent fournir fréquemment des rapports attestant du travail terminé.