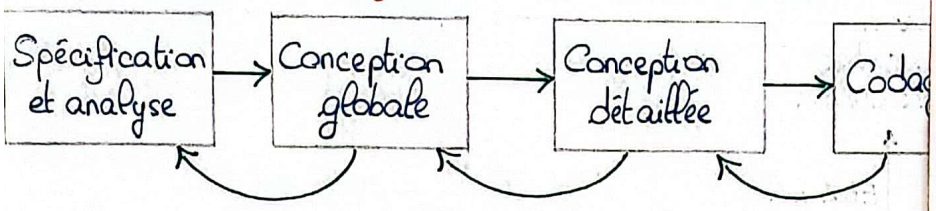
Modèles de cycle de vie logiciel et méthodes de développement

* Cycle de vie logiciel :
  - Ensemble d'étapes qui composent le processus de développement et d'utilisation du logiciel.
    → Donner une structure de manière graphique et physique.

I / Cycles de vie classiques :

* Modèle de cycle de vie en cascade :

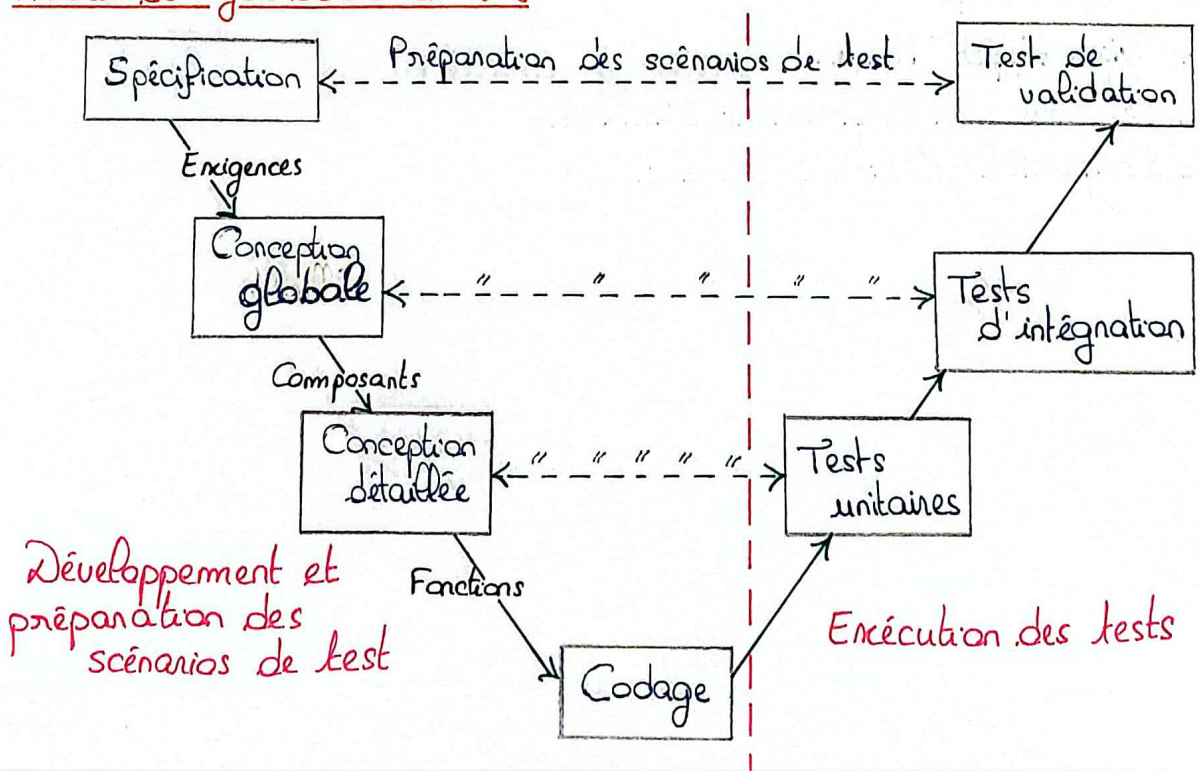


Méthodes classiques :
- Etapes clairement définies
- Documentation très fournie
- Pour les grands projets et les projets gouvernementaux

- Des documents / programmes sont produits à la fin de chaque étape à une date précise.
- Le résultat de chaque étape est soumis à un examen. Si ce dernier est satisfaisant, on passe à l'étape suivante.
- Une étape ne remet en cause que la précédente.

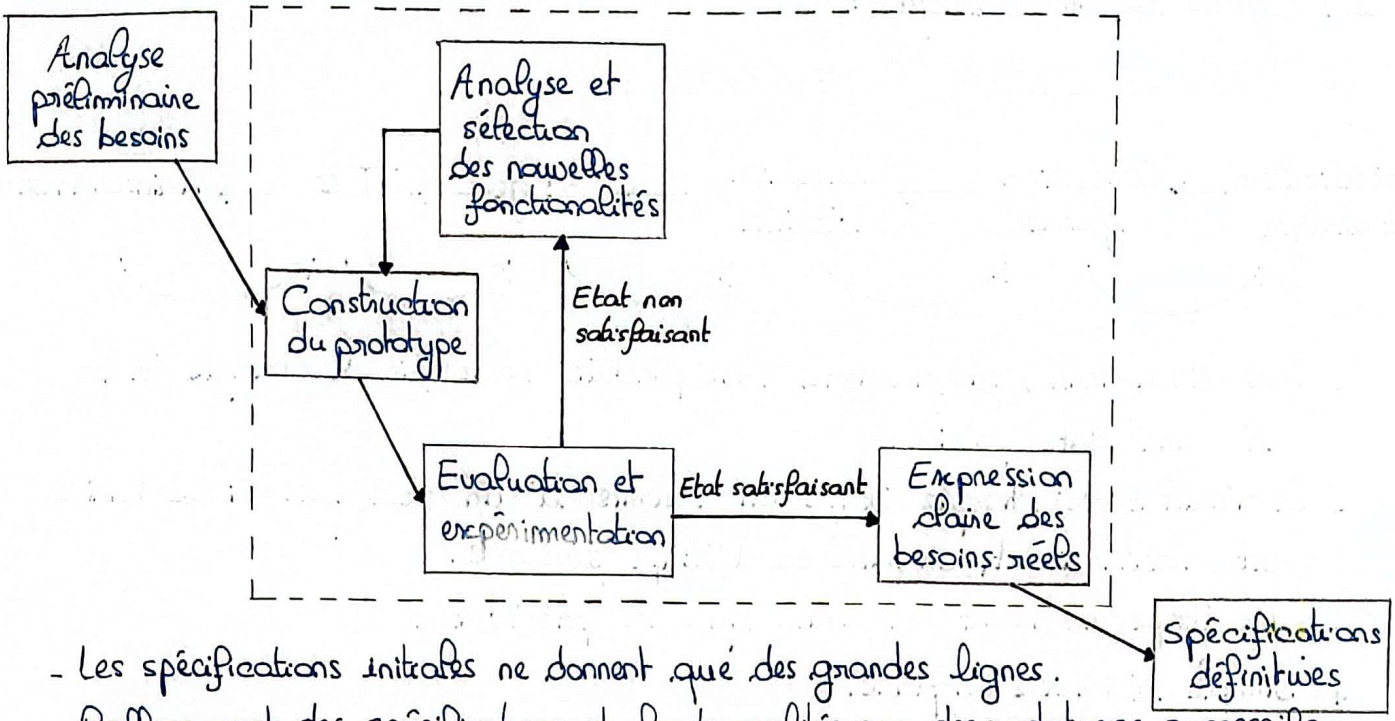
\+ Simple à mettre en place.
\- Détection tardive des erreurs.
\- Inefficace pour les projets qui exigent la prise en considération de risques.

* Modèle de cycle de vie en V :



- Enchaînement séquentiel : les résultats des étapes de départ sont utilisées par les étapes d'arrivée.
  + Vérification objective des spécifications.
  + Le plus utilisé pour les grands projets.
  - Pas de prise en considération de risques.
  - Inefficace lorsque les besoins ne sont pas stables.

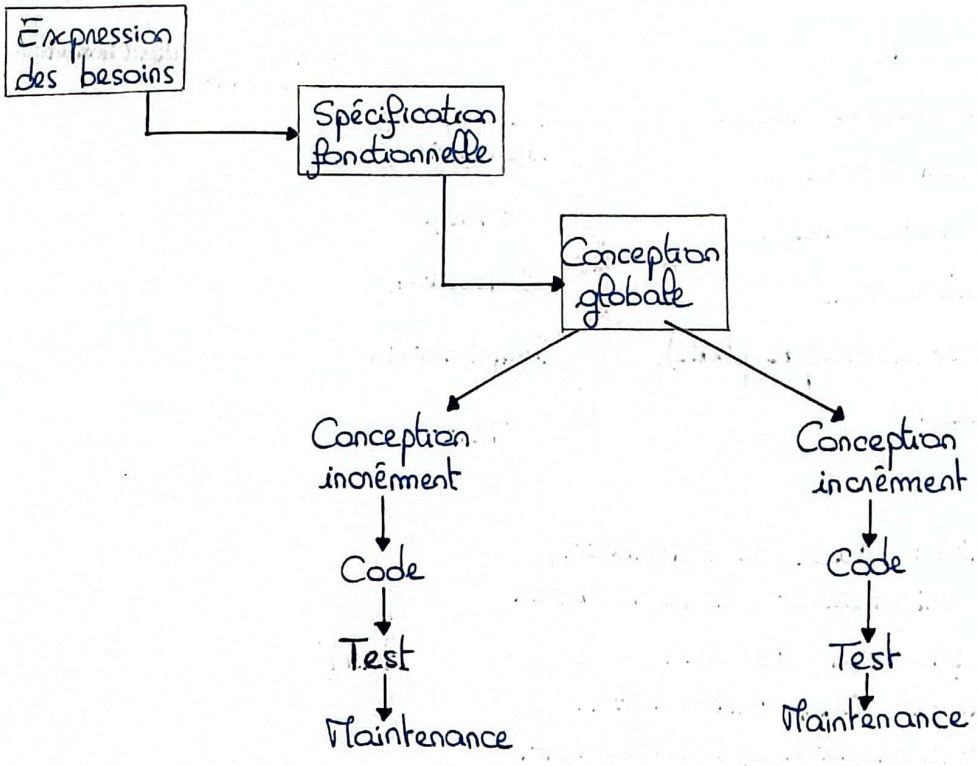
* Modèle de cycle de vie par prototypage :



- Les spécifications initiales ne donnent que des grandes lignes.
- Raffinement des spécifications et fonctionnalités par des prototypes successifs.
- Implication du client à intervenir lors de l'évolution du prototype.

+ Participation du client.
+ Réception rapide de résultats tangibles.
+ Eviter l'effet Tunnel.
- Le coût peut rapidement s'éclater.
- Impossible d'estimer le coût.

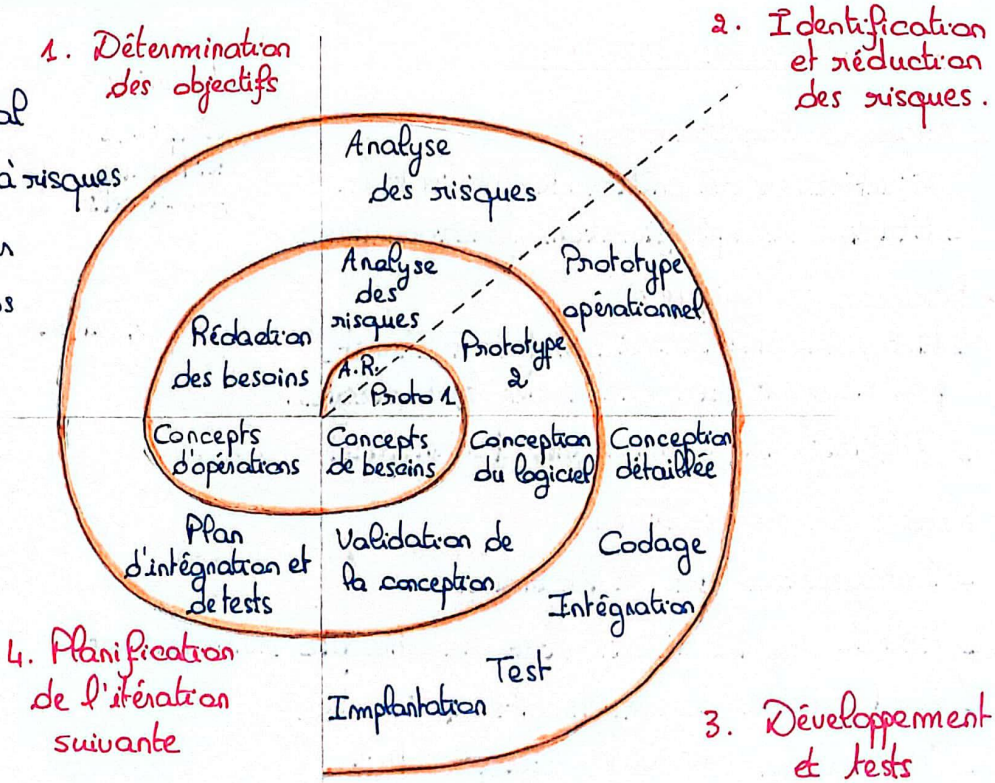
* Modèle de cycle de vie incrémental :



- Diviser le projet en incréments.
- Chaque incrément est testé comme un produit final.
  - + Développement plus simple.
  - + Optimisation du temps et partage des tâches.
  - − Les incréments doivent être indépendants.

* Modèle de cycle de vie en spirale :

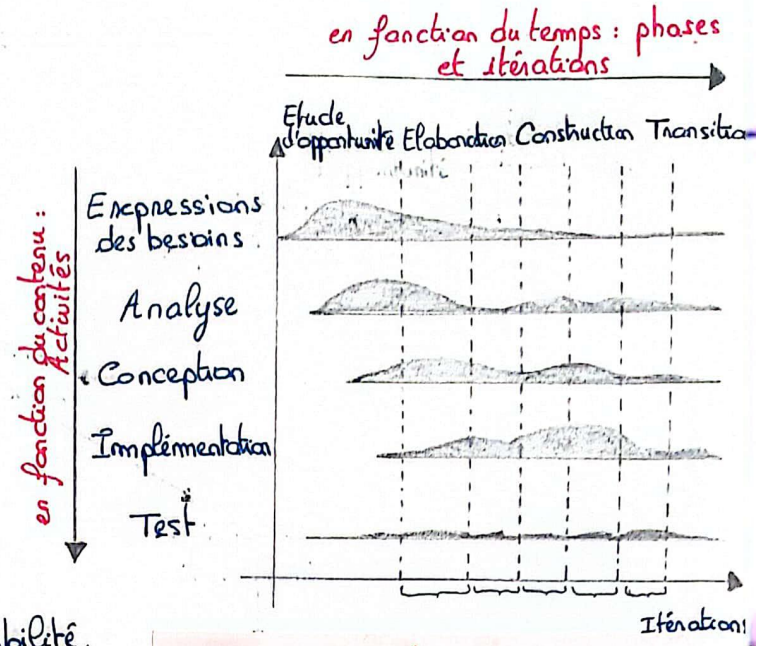
+ Analyse de risques
+ Complet et général
+ Projets innovants et à risques
− Difficile à appliquer
− Beaucoup de temps
− Coût élevé



II/ Méthodes lourdes :

* Le processus unifié UP :

- Piloté par les C.U.
  (Adopter le point de vue user).
- Centré sur l'architecture
  (décrite sur différentes vues).
- Itératif et incrémental
  (feedbacks et test à chaque phase).
- Orienté risques.



- Etude d'opportunité : étude de faisabilité, de risques, des exigences et des contraintes.
- Elaboration : spécification des cu.
- Construction : tous les cu + version bêta.
- Transition : Test et correction → Déploiement.

Inconvénients des UP :
- Font tout mais lourds.
- Parfois difficile à mettre en oeuvre.
- Beaucoup de temps

* Le processus RUP :

- RUP est une instance / implémentation de U
- Livrables gérés et contrôlés tout au long du d
- Modifications prises en charge.
- Intégration au fur et à mesure (pour les
- Centré sur l'architecture logicielle.

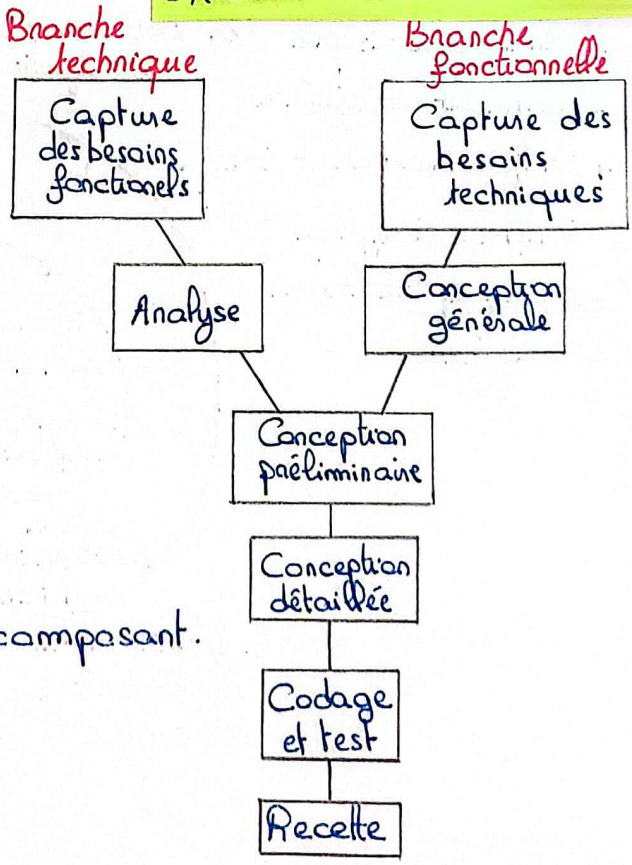
*** Rôles & Activités**
- Concepteur : Définir les opérations
- Rédacteur : Détailler le D. utilisation
- Analyste Système : Trouver acteurs et C.U.
- Développeur : Réaliser les tests
- Architecte : Concevoir

* Le processus 2TUP :

*** Branche fonctionnelle :**
- S'intéresser au métier de l'utilisateur.
- Etudier les spécifications fonctionnelles.

*** Branche technique :**
- Lister les obstacles à prendre en compte par rapport aux choix des technologies.
- Définir les composants techniques.

*** Phase de réalisation :**
- Intégration des 2 branches.
- Etude de la réalisation de chaque composant.
- Codage des composants + Tests.
- Validation et vente du produit.

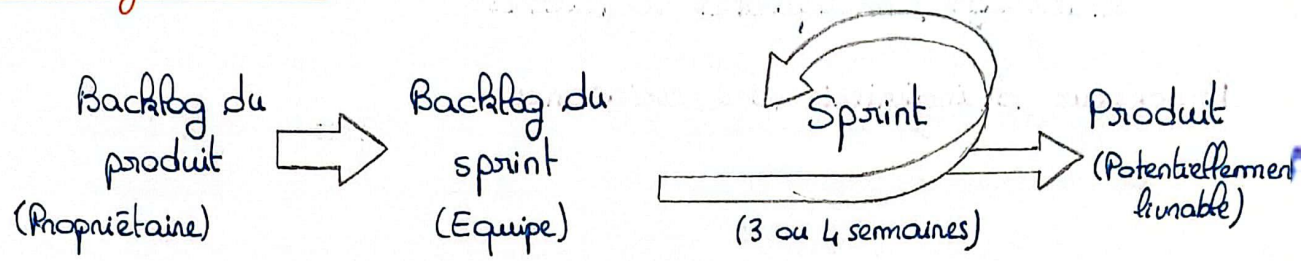


III / Méthodologies agiles :

4 valeurs :

- Priorité aux personnes et aux interactions sur les procédures et outils.
- Priorité aux applications fonctionnelles sur la documentation pléthorique
- Priorité à la collaboration avec le client sur la négociation de contrat.
- Priorité à l'acceptation du changement sur la planification.

* Méthodologie Scrum :



- Composé d'une suite d'itérations courtes : sprints.
- A la fin d'un sprint, l'équipe livre la partie au client.
- Le projet peut être modifié par le client à la fin de chaque sprint.

Les acteurs :

- Product Owner : Porteur de la vision globale du produit.
  Gère le Backlog du produit / Accepte ou rejette les livrables.
- Scrum Master : Gardien des pratiques de Scrum / Facilitateur.
- L'équipe : 5 à 9 personnes

Les artéfacts :

- Backlog du produit : Listes des fonctionnalités attendues.
- Backlog de sprint : Sous-ensemble de user-story du Backlog du produit sur le sprint courant.
- Burndown Chart : Représentation graphique de l'évolution du travail.

Les réunions :

- Réunion de planification du sprint : planification.
- La mêlée quotidienne : Evaluer l'avancement / identifier les obstacles
- Revue de Sprint : Evaluation finale du sprint.
- Rétrospective : Améliorer le processus pour le prochain sprint.

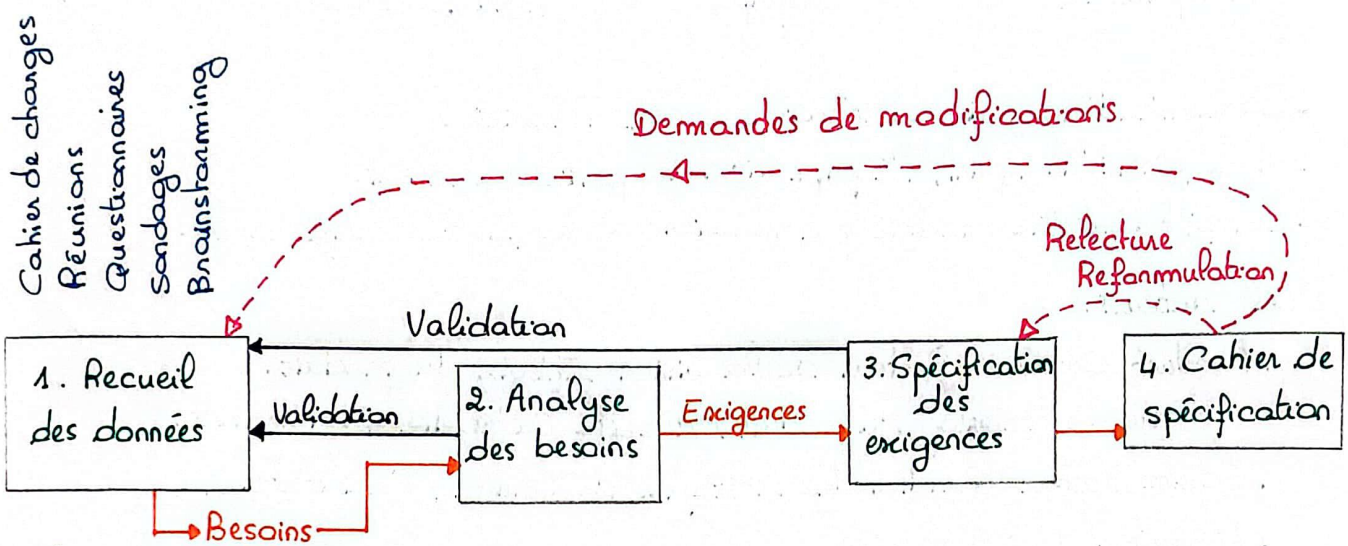
Gestion des exigences

Intérêts de la spécification logicielle :

Langage naturel imprécis, ambigüe, contradictoire, etc.

→ Recours aux diagrammes UML / graphiques et schémas illustratifs / maquettes logicielles

Processus d'ingénierie des exigeances :



Eliminer : - Redondance
- Ambigüité
- Incohérence

Types de documents de spécifications :

- Point de vue interne : description précise (concepteurs, personnels technique)
  → Document de spécification technique.
- Point de vue externe : description des services et contraintes (utilisateurs non informaticiens, décideurs) → Document de spécification des besoins.

Catégories des spécifications :

- Fonctionnelles :
  définit les services du système et ses fonctionnalités.
- Non fonctionnelles :
  définit les contraintes de performance (efficacité, robustesse etc) et contraintes d'interface (logiciel et matériel)
- Liés au processus :
  définit les contraintes liées au développement du produit (coûts, délais) ou à l'exploitation future (prix de vente, marché de distribution).

Architecture et conception de logiciels

Objectifs de la conception :

- Réutilisabilité : (code source ou API) → gagner du temps.
- Maintenabilité : influt les coûts de maintenance.

Niveaux conceptuels :

Conception architecturale globale

- Architecture de haut niveau.
- Décomposer le logiciel en sous-systèmes définis par leurs interfaces et leurs fonctions.

Conception architecturale détaillée

- Détailler la conception générale.
- Fournir pour chaque composant une description.

Types d'architecture :

Architecture logique

- Structure logique de l'application.
- Décomposition en éléments logiques
  - Approche de résolution :
  - Approche Top - Down :

1. Concevoir la structure logicielle.
2. Etudier les éléments du plus bas niveau
3. Détailler tous les éléments de la structure

- Approche Bottom - Up :

1. Identifier les éléments de bas niveau.
2. Prendre des décisions de réutilisation.
3. Décider de l'assemblage des éléments.

- Combinaison des 2 approches :

. Top - Down : bonne structure du système.
. Bottom - Up : s'assurer que les composantes réutilisables soient concevables et réalisables.

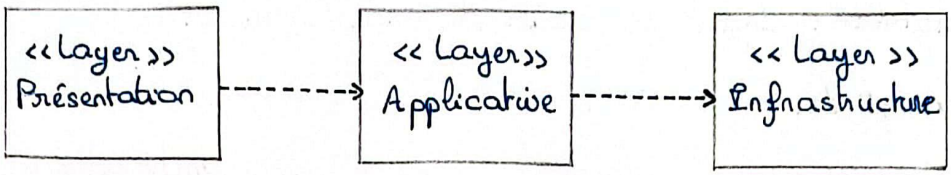
Architecture ~~logicielle~~ physique

- Structure physique de l'application.
- Décomposition en éléments physiques (serveurs, pcs, etc).
- Architecture physique 1 - niveau (1 - tiers) :
  Toutes les couches logiques sont situées sur la même machine.
- Architecture physique 2 - niveaux (2 - tiers) :
  Les couches logiques sont séparées sur 2 sites :
  - Les dispositifs du serveur sur un site (BD, service de messagerie etc).
  - Le reste sur le poste client.
  - Partage d'informations entre utilisateurs.

* Patrons d'architecture :

Modèles standards de structuration qui couvrent les types classiques d'application (solution générale et réutilisable) Ex : MVC, modèles en couches.

* Modèle en 3 couches :



Principe : faire évoluer distinctement les 3 couches sans remettre en question les autres couches.

- Il y a une dépendance entre IHM (présentation) et Métier (applicative)
- Le package Infrastructure doit répondre aux besoins de Métier.

* Couche présentation :
  - Interactions entre l'utilisateur et le logiciel.
  - Visualisation des informations.
  - Traduction des commandes de l'utilisateur en actions sur les autres couches.

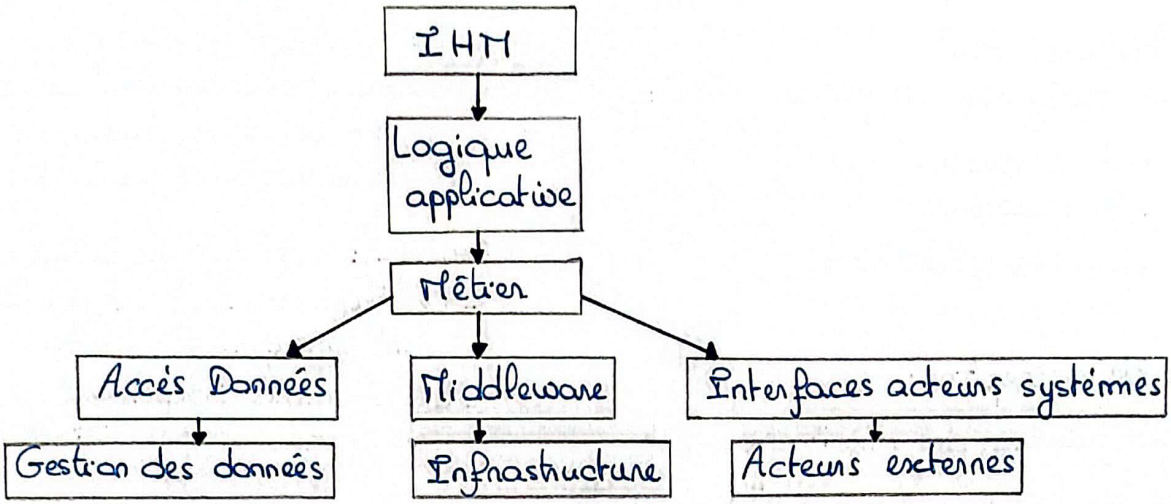
* Couche applicative :
  - Partie fonctionnelle de l'application.
  - Décrit les opérations que l'application opère sur les données.
  - Renvoie à la couche présentation les résultats qu'elle a calculés.

* Couche infrastructure :
  - Partie du code responsable de l'accès aux données
  - Interactions avec la BD.
  - Simplifier la couche métier qui utilise les traitements de cette couche. (La couche métier reste indépendante du code, elle utilise les classes de la couche persistance qui encapsulent ces traitements)

+ Couches ± indépendantes.
+ Sécurité peut être renforcée.
+ Facilite la réutilisation et la substitution.
+ Favorise la standardisation.
- Complexe.
- Problèmes au niveau des performances.
- Plus d'exigence.

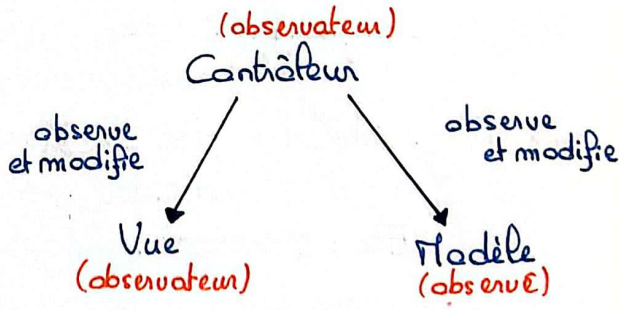
* Modèle en 5 couches :

Modèle en 3 couches + couches intermédiaires
→ Réduire la complexité.
Faciliter la réutilisation et la substitution.



* **Couche présentation (IHM) :**
  - Capter les requêtes de l'utilisateur et les envoyer à la couche applicative.
  - Récupérer la réponse et afficher les résultats.
* **Couche applicative :**
  - Utilise les services métier et les services techniques.
  - Sollicitée par l'IHM / Moyennement réutilisable.
* **Couche Métier :**
  - Implémenter les services métier propres au domaine.
  - Sollicitée par la couche applicative / Potentiellement réutilisable.
* **Couche d'accès aux données :**
  - Gère le stockage de données (CRUD).
* **Couche gestion des données :**
  - Communicateur avec la BD.
  - sas de sécurité entre l'application et la BD.

* Modèle MVC :



* **Modèle :** - Contient les données à afficher.
  - Définit la logique de manipulation.
  - Envoie des événements quand les données sont modifiées.
* **Vue :** Affichage à l'écran.
  - Envoie des événements des actions de l'utilisateur.
  - Ne connaît ni l'API du contrôleur ni le modèle.
* **Contrôleur :** Assure l'interaction entre la vue et le modèle et les connaît.
  - Observe le modèle et modifie la vue.
  - Observe la vue et modifie le modèle.

+ Plusieurs vues sur le m̂ modèle
+ Plusieurs contrôleurs peuvent modifier le m̂ modèle.
+ Les vues seront notifiées.

Qualité de la conception architecturale :

* Application de critères de qualité :
  - Faible couplage, forte cohésion.
* Utilisation de patrons de conception :
  - Patrons GoF.
  - 3 catégories de patrons GoF :
    . de création.
    . de structure.
    . de comportement.

* Patrons de conception :
  La normalisation des architectures est grandement facilitée par l'utilisation de Framework et de patrons (patterns).
  Ils favorisent :
  - La réutilisation.
  - La capitalisation d'expérience.

* Patrons (Patterns) :
  - Solution générique à un problème.
  - Renforce l'abstraction.
  - différents types de patrons :
    * Patrons d'analyse : fournissent des solutions réutilisables pour les étapes d'analyse.
    * Patrons architecturaux (Ex : MVC).
    * Patrons de conception (Ex : Singleton / Observer / Factory)
  - Catégories des patrons GoF :
  . Création :
    - objectif : solution aux problèmes liés à l'instanciation des classes. abstraction du processus d'instanciation.
    - intérêts : plus de flexibilité aux programmes. décider quel objet créer pour un C.U. donné.
    Ex : Prototype / Singleton / Abstract Factory etc.
  . Comportementaux :
    - objectif : solution aux problèmes de communication entre objets et algorithmiques.
    - intérêt : distribution des responsabilités.
    Ex : Observer / Visitor / Interpreter / etc.

Couplage :
un objet A faisant appel aux méthodes d'un objet B a un couplage aux services de B.
* Le couplage est une mesure de degré auquel un élément est lié à un autre, en a connaissance ou en dépend.

Cohésion : un système a une cohésion :
- Les éléments sont inter-reliés sont groupés ensemble.
- Les éléments indépendants sont dans des groupes distincts.

• Structurels :

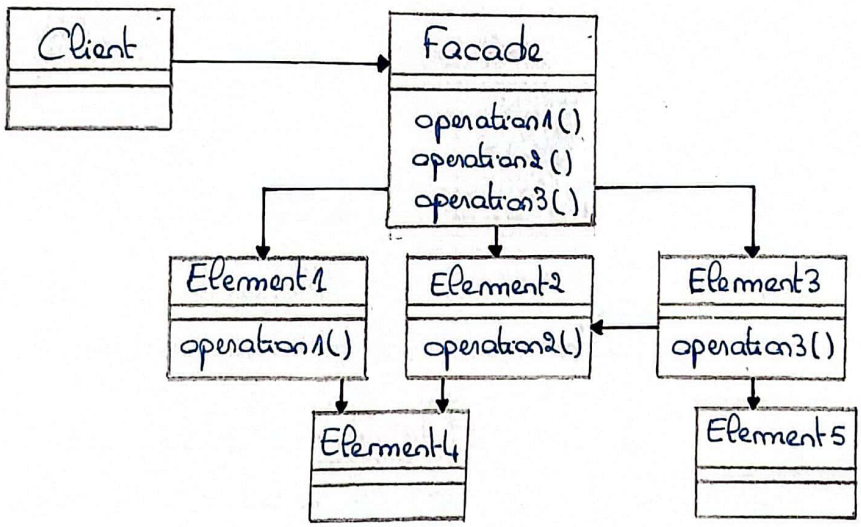
- objectifs : solution aux problèmes de structuration des classes, d'abstraction, de réutilisation.
  composition de classes et d'objets pour des structures plus complexes.
- intérêt : définir des moyens de composer des objets pour obtenir de nouvelles fonctionnalités.

Ex : Facade / Bridge / Proxy etc.

Exemples de patron de conception :

* Facade :

- Cacher la complexité d'un système.
- Minimiser les communications et les dépendances entre les sous-systèmes.
- Fournir au client une interface à travers laquelle il pourra accéder au syst.
- Fournir au client des méthodes simples à travers une classe unique.

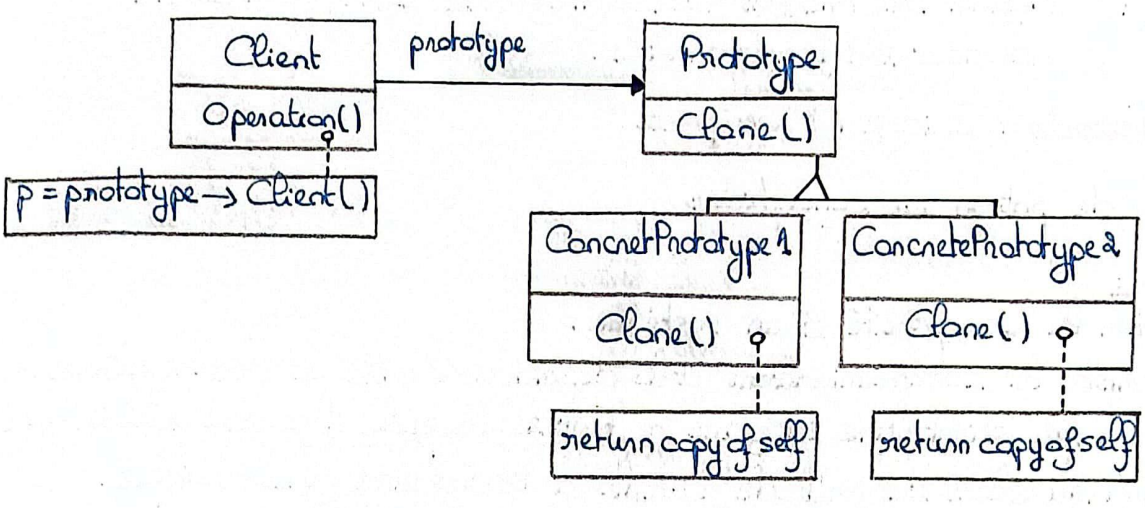


* Observer :

- Définit une dépendance "One-To-Many" entre objets de sorte que lorsque l'état d'un objet change, tous ses objets dépendants sont notifiés et mis à jour automatiquement.
- L'objet observé gère une liste d'observateurs dotés d'une méthode de màj et notifie les changements aux observateurs en appelant leurs méthodes de màj.

* Prototype :

- Spécifie les types d'objets à créer en utilisant un prototype.
- Créer de nouveaux objets en copiant le prototype.
- Fournir de nouveaux objets par la copie d'un exemple plutôt que de produire de nouvelles instances.



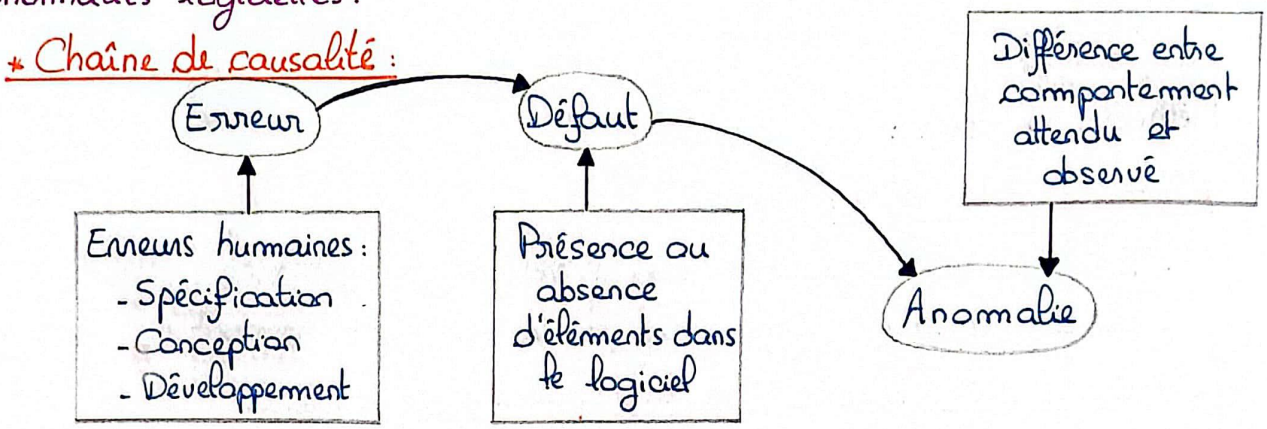
Tests & Validations

- S'assurer que le produit répond aux exigences → Validation
- S'assurer que le produit est construit correctement → Vérification
- Améliorer la productivité des équipes.

* Les tests montrent la présence de défauts mais ne peuvent pas en prouver l'absence.
* Les tests exhaustifs sont impossibles.
- Les tests doivent commencer le plus tôt possible.

Anomalies logicielles :

* Chaîne de causalité :



* Bug :

- Dysfonctionnement de l'ordinateur.

* Crash applicatif ou Deny of Service :

- Déclenchement d'un mécanisme à la fois matériel et logiciel qui met hors service le logiciel défaillant lors de la tentative de ce dernier d'effectuer des opérations impossible à réaliser.

* Fuite de mémoire :

- La quantité de mémoire utilisée par le logiciel défaillant va augmenter et gêne le déroulement des autres logiciels.

* Vulnérabilité :

- Faille de sécurité informatique.

* Faute de segmentation :

- Dysfonctionnement dans des opérations de manipulation de pointeurs.
- Détection des exceptions provoquant la mise hors service.
- Lecture ou écriture des informations dans un emplacement de mémoire qui n'existe pas ou ne lui est pas autorisé.

* Buffer Overflow :

- Lors de l'écriture dans un tampon, un processus écrit à l'extérieur de l'espace alloué au tampon, écrasant ainsi des informations nécessaires au processus.
- Le comportement de l'ordinateur devient impossible. Il en résulte un blocage du programme, voire de tout le système.

* Deadlock ou inter blocage :

- Dysfonctionnement durant lequel plusieurs processus s'attendent mutuellement.
- Ressources vérouillées durant les attentes, pouvant provoquer un blocage dans le système.

Méthodes de tests :

* Méthode boîte noire :

- Tests autour du fonctionnement externe du système.
- Les détails d'implémentation des composants ne sont pas connus.

* Méthode boîte blanche (ou transparente) :

- Tests autour du fonctionnement interne du système.
- Les détails d'implémentation des composants sont tous connus.

* Méthode boîte grise :

Combinaison de la B.N. et B.B. :

- Tests autour du fonctionnement externe du système.
- Quelques détails d'implémentation des composants sont connus.

Types de tests :

* Test nominal (de bon fonctionnement) :

- Entrer des données volontairement valides. → Test to pass.

* Test de robustesse (de défense) :

- Entrer des données volontairement invalides. → Test to fail.

* Test de performance :

- Soumettre l'appli à des demandes de ressources anormales (test de stress) ou à des valeurs aux limites (test de charge).

* Test unitaire :

- Tester les fonctions de code par les programmeurs.

* Test d'intégration :

- Valider le bon fonctionnement d'une ou de plusieurs parties développées indépendamment avec le reste de l'application.

* Test système :

- Tester l'entière application par la méthode boîte noire dans un environnement qui imite la situation réelle d'utilisation de l'application.

* Test de non régression (liés au changement) :

- Reprendre un ensemble de cas de tests après avoir fixé les erreurs.

* Test d'administration :

- Se focaliser sur les aspects d'administration.

* Test d'utilisabilité :

- Valider si l'application est facile à utiliser.
- Test dépendant de l'utilisateur final.
- Les programmeurs et les testeurs ne sont pas impliqués.

* Test de sécurité :

- Vérification de vulnérabilité de sécurité.

* Tests automatique / Tests automatisé :

- Génération automatique de tests utilisant le plus souvent des Frameworks.
- Peuvent être des tests unitaires (sur des fonctions, classes, etc).
- L'analyse des résultats sera rendu au testeur suite à une batterie en mode nuit, week end etc.

* Test & Scrum :

* TDD : Test Driven Development :

- Développement piloté par les tests.
- Ecrire des tests pour vérifier si le test fonctionne.

* ATDD : Acceptance Test Driven Development :

- S'intéresse à la qualité externe du logiciel.
- S'assurer quels besoins doivent être réalisés.

étapes :

1 - Décrire le comportement attendu d'une story.
2 - Transformer les conditions en cas de test appelés Story Test.
3 - Développer le code applicatif qui répond qui répond à la story.
4 - Passer les Story Test sur le code applicatif.