

Concevoir : organiser , optimiser, s'inspirer, créer, modéliser

→ Répondre aux critères de qualité

→ Comment ? Concevoir

→ Doit être alignés avec les BF et les BNF

Spécification → Quoi ?

Atelier 1 : Conception architecturale

Elaborer la conception architectural d'une maison ayant les caractéristiques suivantes :

2 étages

1 sous sol

1....

Différentes interprétation (BF, BNF) → importance du processus de spécification

Différents choix architecturaux → doivent être alignés avec BNF et BF → Critères de qualité

Différentes vues (**logique et physique**) architecturales

Différents niveaux de détails (**Boit noir / Boite blanche**)

Différents styles architecturaux

Différents types de vues ou différentes structures d'une même vue ??

Une architecture peut être exprimée sous forme de différents types de vue et chaque type de vue différentes structures.

Architecture d'un logiciel :

définition d'archi logique et physique (à voir)

Structure d'un logiciel

Décomposition en éléments structuraux (interconnectés les uns des autres à travers des interfaces)

Architecture logique → packages

(type 1 de → classes

v. architecturale) → composants

1. types d'architectures

2.structure

3.niveau de détails (raffinement hiérarchique de l'architecture)

3.1 niveau 0 : Archi globale : boite noire (haut niveau de description)

3.2 niveau 1 : Archi détaillée : boite blanche

→ Critère de qualité de l'architecture logicielle :

-Faible couplage

-Forte cohésion

Pas de forte dépendance: Pourquoi ?

- en cas d'erreur dans un composant , une répercussion sur tous les composants dépendants
- en cas de changement d'un composant, tous les composants dépendants doivent être modifiés

Atelier 2 : Conception architecturale 1 (16/03/2016)

Proposer la conception architecturale de votre application PiDev (%sprint Web) suivant les contraintes suivantes :

- La conception architecturale doit véhiculer une vue logique à base de composants.
- La conception architecturale doit traduire un niveau global de description

Critère de qualité de la conception architecturale :

Forte cohésion(regrouper les éléments qui sont fortement liés ensemble) } GRASP (Pattern)
Faible couplage }

➔ réduire les indépendances ,faible couplage

GRASP : General Responsibility Assignment Software Patterns(Assignation des responsabilités)

Recherche 1 : relative au chapitre "Conception architecturale"

Les styles architecturaux : (Principe de fonctionnement + avantages + inconvénients + exemple d'application "web, interface graphiques...")

Modèle en 3 couches

Modèles en 5 couches (**phoenix**)

Modèle MVC

Les patrons de conception GoF (Gang of Four)

Catégories du GoF

Principe et exemple d'application :

1-Abstraction Factory

2-Prototype (**phoenix**)

3-Façade

4-Adapter

5-Observer

6-Chain of responsibility

Recherche 2: relative au chapitre "Gestion de configuration logicielle GCL"

1- Intérêt de la GCL

2- Principe de base de la GCL (Scénarii)

2.1 Versioning

2.2 Journalisation

2.3 Gestion de conflits

tableau les patrons (styles architecturaux) 3 couches 5 couches MVC

Style	Principe	Règles de communication	Avantages	Inconvénients	Type d'applications
MVC	Isoler la donnée elle-même de sa présentation Distinguer la consultation de la modification	Les 3 composantes suivantes d'une donnée sont <i>distinguées et séparées</i> Le modèle (sa structure) La vue (sa représentation pour affichage) Le contrôleur (les moyens de modifier la valeur)	+Une conception clair e et efficace grâce à la séparation +Un gain de temps de maintenance et d'évolution du site +Une plus grande souplesse pour organiser le développement	-la séparation des différentes couches nécessite la création de plus de fichiers - pas convenable aux petits projets	Les applications Web
5 couches	Décomposition logique en 5 couches	Chaque couche a ses propres responsabilités , elle utilise la couche en dessous d'elle , et appelle la couche suivante (via des interfaces	+ Maitriser la complexité +Optimiser le tps de développement +Séparer les responsabilités	-une expertise est requise -difficulté de conception - cout élevé	Applications manipulant des données de persistances ➔ les application de gestion utilisateurs BD + GUI
3 couches	C'est une extension du modèle client-serveur.	1. La couche présentation 2. La couche fonctionnelle liée au serveur 3. La couche de données liée au serveur de base de données (SGBD)	+ plus de flexibilité	- expertise requise -coût élevé	

Les patrons de conception

Nom du patron	Catégorie	Objectifs de la catégorie	Problèmes traités par le patron	Solution proposées par le patron
1 - Façade	Structure	Camoufler les fonctionnalités du système	-Sys.complexes - classes très nombreuses -nombre de fonctionnalités relativement faible	Découpler le client du sys complexe .
2-Adapter	Structure	Etendre les fonctionnalités -> Extension de structure	- Un sys qui a les bonnes données, les bons services mais la mauvaise interface (problèmes d'incompatibilité)	Créer une nouvelle classe adapter qui fournit l'interface, fait le lien entre l'utilisateur et le système
3-Prototype	Création	Gérer le problème de création de nouveau objets	-La création d'une instance est couteuse en terme de temps et ressources	Clonage / Copie
4-Abstract Factory	Création	Camoufler le processus d'instanciation au client (nouvelle solution de creation)	-Le sys doit être indépendant de la création du projet -Créer des objets de la même famille	Fournit une interface unique pour instancier des objets
5-Pattern observer(MVC)	Comportement	- gère les problèmes d'interactions entre objets	-Le besoin qui consiste à modifier tout changement de l'état d'un objet dépendant	- les observateurs peuvent représenter le changement notifié de façon dynamique
6-Chain of responsibility	Comportement	- Maitriser l'interaction entre objets	- partitionner le traitement de la même requête entre différents destinataires	- isoler les différents parties d'un traitement

Exemple d'applications (Diagramme de classes)

1-Façade : objectif -> offrir un point d'accès unique dans un système complexe (sys. avec un fort couplage)

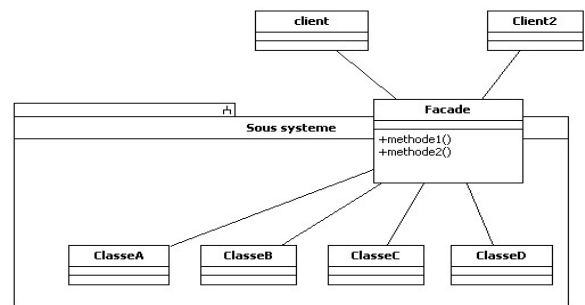


Figure 1 Façade

2- Adapter :

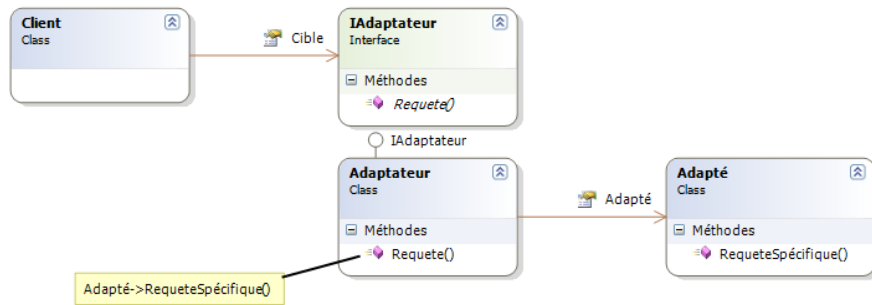


Figure 2 Adapter

3-Prototype :

inconvénient : on est limité par le prototype crée

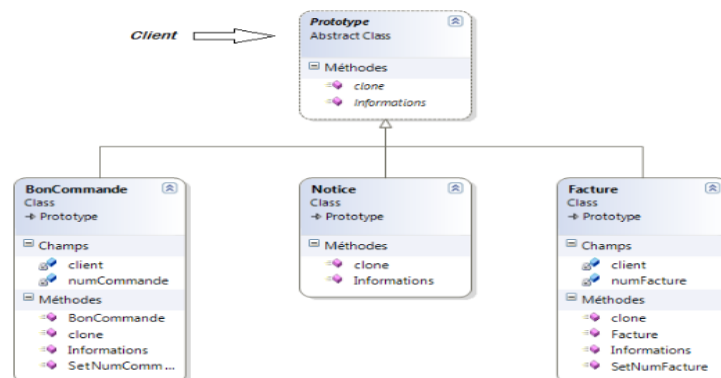


Figure 3 Prototype

4- Abstract Factory

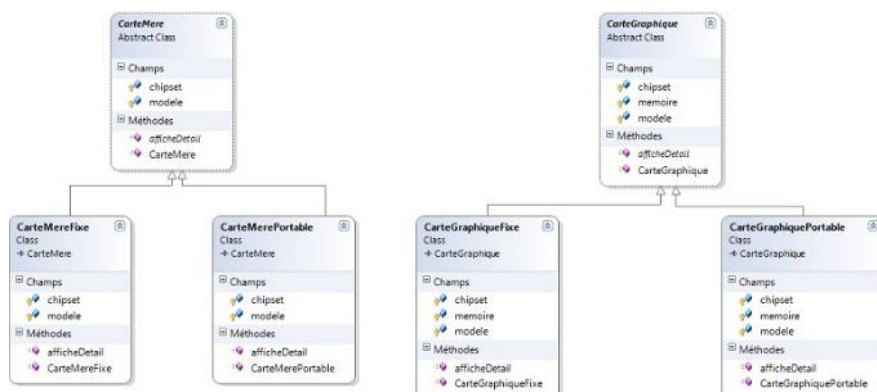
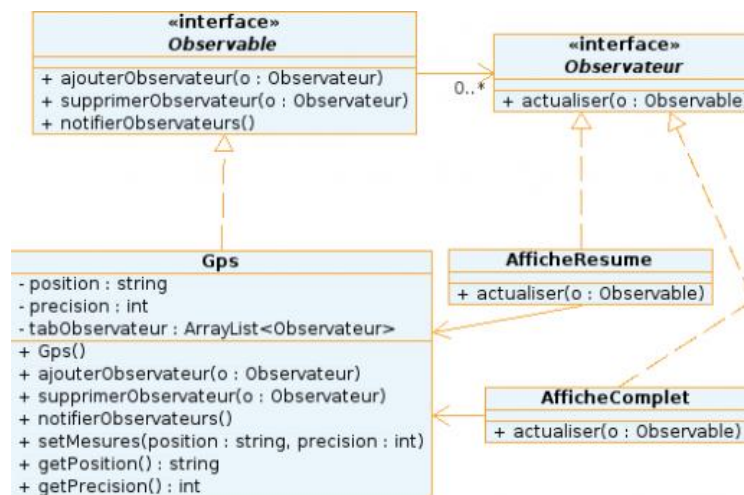


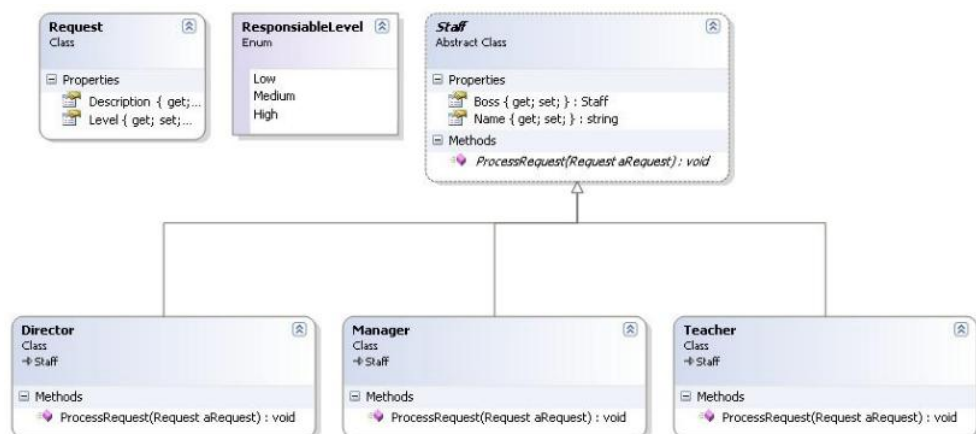
Figure 4 Abstract Factory

5- Observer (MVC)

Modèle observé , Vue et Contrôleurs observables



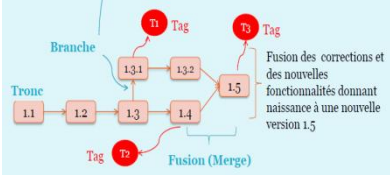
6- Chain of responsibility



l'intérêts de la GCL :

- travail collaboratif
- plusieurs développeurs sur le même projet

➔ Outils GCL : Centralisés (SVN,CVS...) ou décentralisés (GitLab,GitHub...)

Concept de la GCL	Description	Exemple
Versionning	<p>Archiver l'évolution d'un logiciel(temps + fonctionnalités</p> <p>Passage de 1.3 a 1.3.1 (on a effectué une modification)</p> <p>1.5 peut devenir 2.0 (nouvelle version)</p>	<p>Numérotation 3 chiffres n°version,n°revision,n°correction</p> 
Gestion de conflits	<p>Si les portions modifiées diffèrent, il n'y a pas de problème.</p> <p>Si des portions modifiées sont communes, il y a problème.</p> <p>Les conflits se résolvent manuellement.</p> <p>Il existe 2 types de gestion de conflits :</p> <ul style="list-style-type: none"> - Pessimiste : impose à chaque utilisateur de demander un verrou avant de modifier une ressource ; ce verrou lui garantit qu'il sera le seul à modifier la ressource (éviter le conflit) - Optimiste : Les développeurs peuvent éditer en même temps le même fichier : le logiciel de gestion de version s'occupe de fusionner toutes les versions du même fichier en un seul, ou de garder une (résoudre les conflits) 	
Journalisation	<p>Permet de conserver et contrôler les modifications apportées aux composants d'un logiciel:</p> <p>Historique d'accès : Qui? Quand?</p> <p>Nature(Création/modification/destruction).</p> <p>Détail de la modification(par des outils intelligents de comparaison des sources).</p>	

Anomalie logicielle

Origine : Erreur au niveau de la spécification , analyse , conception, tests

Présence ou absence d'un composant

Anomalie	Causes	Conséquences
Fuite de mémoire	l'absence de désallocation (de libération) de l'espace utilisé lorsque ces objets ne sont plus référencés.	La saturation de la mémoire de la machine <u>Solution</u> : réduire le risque grâce à la présence d'un ramasse miettes
Crash applicatif	Recherche d'info inexistante Opération impossible à réaliser	Mise hors service le logiciel Défaillance du logiciel/Matériel
Interblocage	Attente mutuelle entre processus	Ressources verrouillées durant les attentes → bloquer d'autres processus ou l'ensemble du système
Faute de segmentation	Problèmes au niveau de manipulation des pointeurs + allocation des @ mémoires	Détection des exceptions (segmentation fault) → mise hors services d'un logiciel défaillant
Buffer overflow	Dépassement de la mémoire tampon	Ecrasement potentiel des informations en mémoire Ecrasement vitales pour les programmes Manipulation des cases mémoires importantes et incidences sécurité
Vulnérabilité	Faibles/faiblesse en terme de sécurité (mécanismes/protocoles)	Système sensible à des attaques malveillantes -Mise hors service -Non disponibilité -Altération des données

Test boîte noire : pas de visibilité sur les détails d'implémentation, ni de code, que les interfaces

Test boîte blanche : repose sur des analyses du code source (visibilité sur les détails)

Test boîte grise : visibilité partielle sur le code + interface (une partie du code)

Type de test	Principe	Méth. de test associée
Test de robustesse	Données entrées volontairement invalide (voir si mn sys traite les exception)	Boîte noire
Test de sécurité	Détecter les failles de sécurité via une vérification périodique de vulnérabilité de sécurité	Boîte noire
Test nominal	Les données qu'on utilise pour effectuer le test doivent être valides	Boîte noire
Test d'administration	Test avec aspects administratifs (sauvegarde/récupération) → évaluer l'efficacité du logiciel	Boîte noire / grise / blanche
Test de non régression	Refaire les mêmes test après modification	Boîte noire / grise / blanche
Test d'utilisabilité	Valider si l'application est facile à utiliser Test par les utilisateurs	Boîte noire
Test unitaire	Test par rapport a des unités de code	Boîte blanche
Test système	Examiner le prog. en entier et détecter les erreurs a travers une vision globale	Boîte noire
Test d'intégration	Créer une version complète et cohérente du logiciel Tester l'intégration des différents modules	Boîte blanche
Test d'acceptation	Permettent de vérifier si le sys marche en totalité (story test)	Boîte noire
Test de performance	Tester le système dans des situations extrêmes (montée en charge: des valeurs haut limites / stress test : ressources anormales)	Boîte noire Boîte blanche
Test automatique	Test automatisé via des outils de test automatique on peut automatiser des tests unitaires ou fonctionnels a travers l'utilisation des scripts	Boîte noire (script) Boîte blanche

concernant l'examen :

Processus unifié (2tup/rup) (illustration des principales caractéristiques , piloté par)

Les phases du rup

Architecture logicielle (phy / logique)

Comment j applique les critères de qualités de la conception architecturale / archi logicielle

Les patrons architecturaux / de conception

Gof

Les objectifs ciblés par chaque patron / comment le patron résout le problème

GCL et T&V