Université Pierre et Marie Curie
Master Informatique Spécialité SAR **Prise de notes de cours**

Tuteur universitaire : Yann Thierry Mieg

Ingéniérie Logiciel - De l'analyse à la conception

GALLARDO Kévin

Paris, 75005, le 9 décembre 2013

Table des matières

Chapitre 1

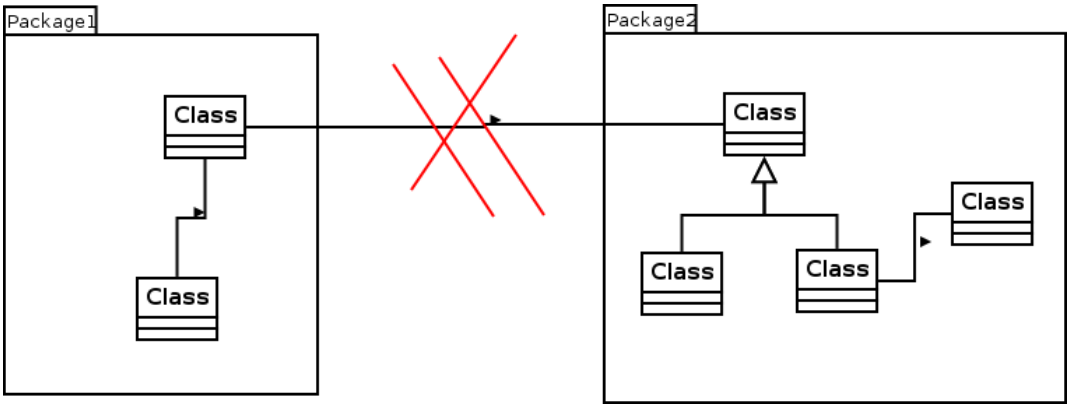
Première étape

Pour effectuer la découpe des données et fonctions on va naviguer dans le diagramme des classes métier à la recherche d'unités réutilisables, maintenables, evolutives.

1.1 Découpe des données

Représ.

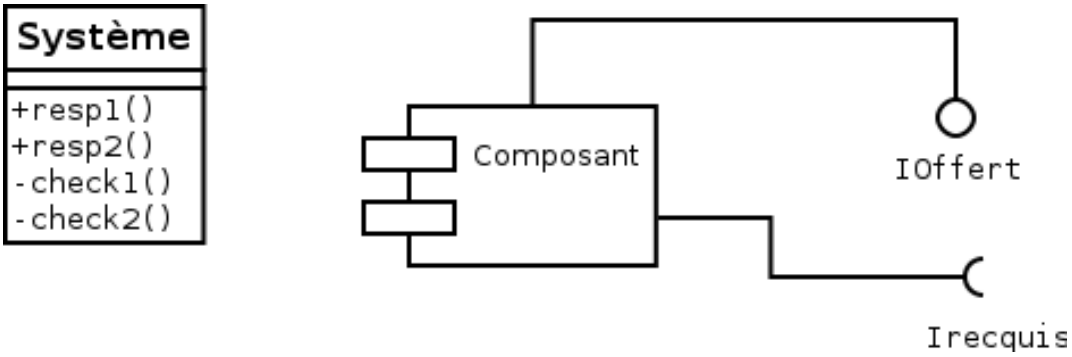
Figure1 : Découpe des données



1.2 Découpe des fonctions

Représ.

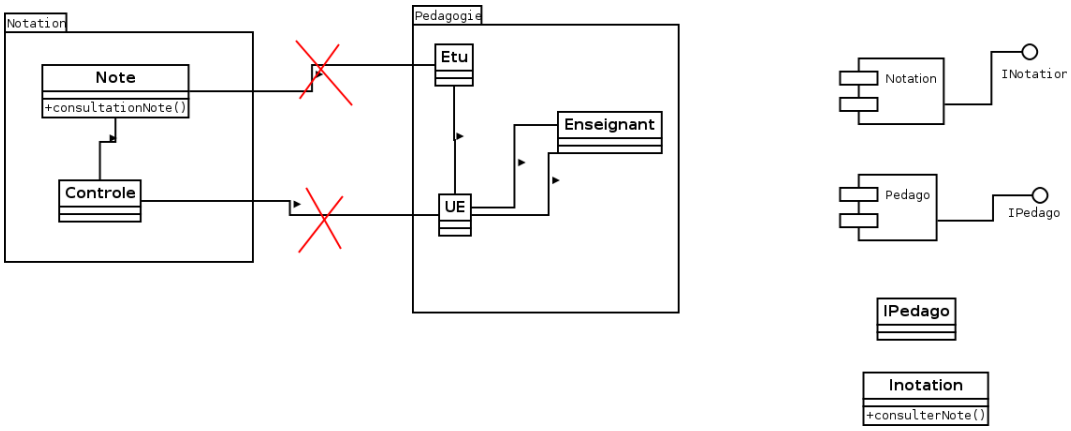
Figure2 : Découpe des fonctions



1.3 Exemple

Exemple

Figure3 : Exemple sur DBUFR



Chapitre 2

Modèle de composants

Le modèle de composants apporte une granularité superieure à l'objet.
il faut savoir gérer :

1. Hétérogénéité
2. Déploiement
   — cycle de vie(instantiation du composant)
   — chargement
   — nommage(enregistrer un nouveau composant, tester la présence d'un composant)
   — distribution

2.1 Les modèles implémentés

— Microsoft : COM, DCOM puis .NET
— Java : EJB puis JEE (JavaRMI, EJB distribué)
— OSGT : Base d'Eclipse (XML) -> chargement à chaud des composants
— SOAP : SOAP+html+IDL (XML et webservice)
— CORBA : CCM (langauge pour définir des interfaces)

Idées de base :

— Favoriser la réutilisation
— Favoriser la substituabilité(maintenance et évolutions)
— Favoriser la portabillité

2.2 Langage de Déscription d'Architecture (ADL)

L'ADL est une spécification séparée.

1. Composants eux-mêmes
2. Connecteurs
3. Topologie

2.2.1 Diagramme de structure interne

Représente des instances et leurs connexions au sein d'un contexte englobant.

Exemple

Figure 6 : Diagramme de structure interne (moteurPédale, moteurExplosion, etc..)

Figure 7 : Diagramme de structure interne des composants

24/10 : figure

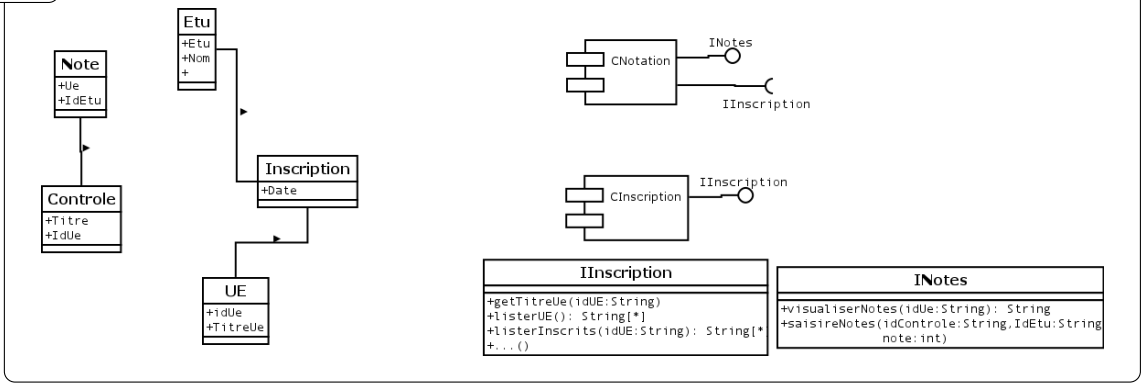
Chapitre 3

Conception Détaillée

Implémentation de composants, plusieurs choix d'approche :
— Framework : EJB, .Net, OSGI....
— J2SE
— Java + Relationnel

3.1 En J2SE

Exemple



3.1.1 Tables associatives

Map <Key, Value>
— get(Key) : Value
— put(Key, Value) : void

Utilisation :

```
1  Map <K, V> implements HashMap
```

3.1.2 Serialisation

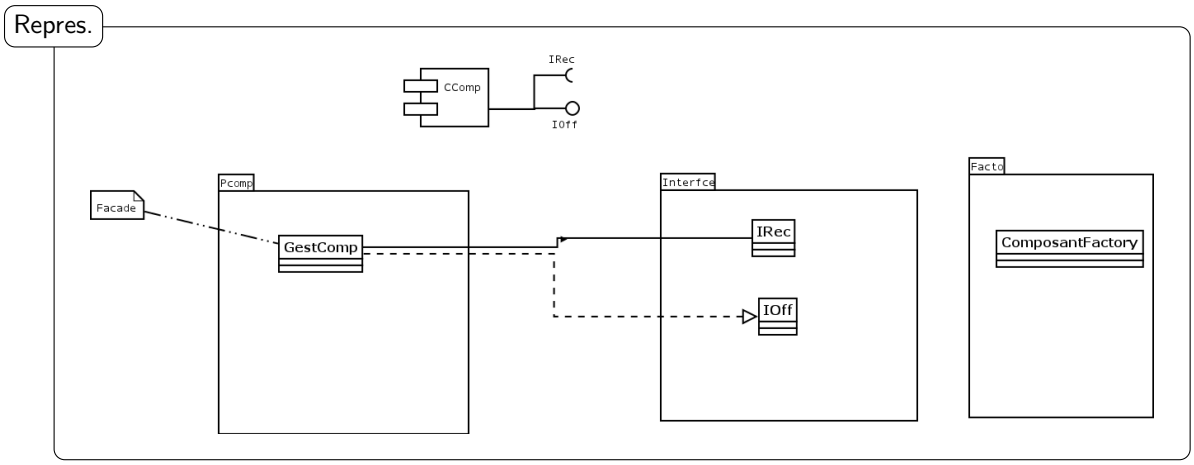
Permet de stocker un objet dans un fichier pour pouvoir être re-ouvert plus tard. Textuel :

— XML
— JSON

Autre :

— Java : 'implements Serializable'

3.1.3 Modèle Physique de Données




```
1 getNotesEtu(String idEtu){
    Connection conn = new Connection(conn);
3   String query = 'Select * From Note n Where n.idEtu' + idEtu +';';
    ResultatSet ns = conn.executeStatement(query);
5 }
```

3.1.4 Objet Relational Mapping (ORM)

Frameworks

— Hibernate
— DAO(Microsoft)

Principes

On crée un mapping vers l'objet etc....

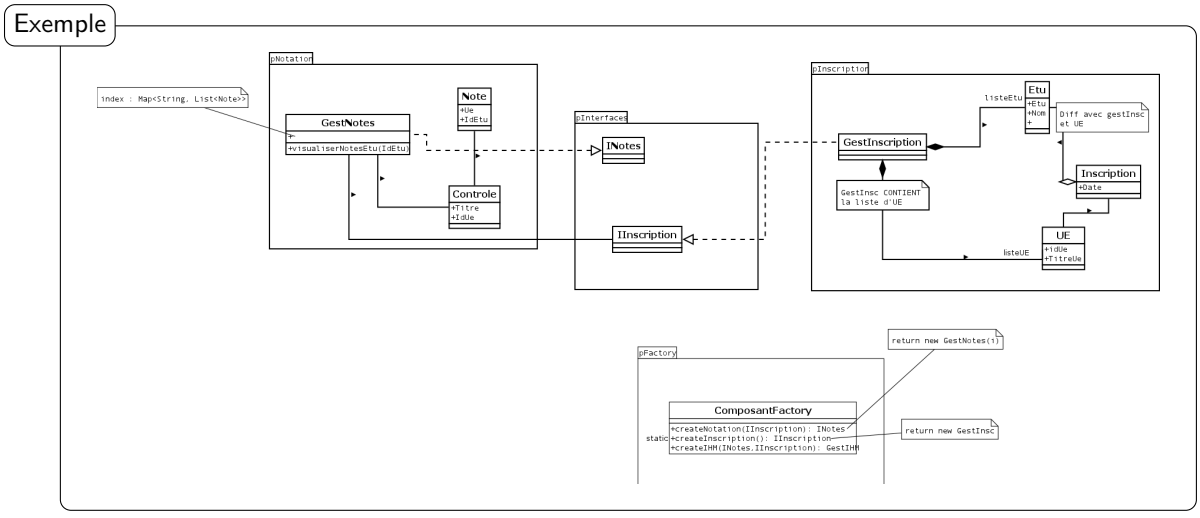
Chapitre 4

Tests d'intégration

C'est une étape de contrôle pour composants. L'objectif :
— tester les interaction entre composants, c'est l'équipe de développement qui s'occupe d'effectuer ces test. Ce sont des test éxécutables
— figer les interfaces
— Permettre le développement en parallèle

4.1 Tester les composants

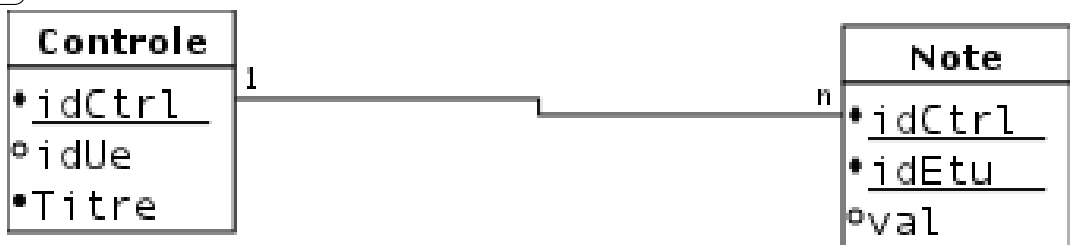
— tester les opérations offertes
— attention à l'état interne (séquences d'intéractions)

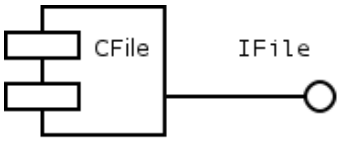


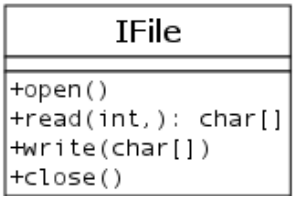
4.2 Test d'intégration d'un composant

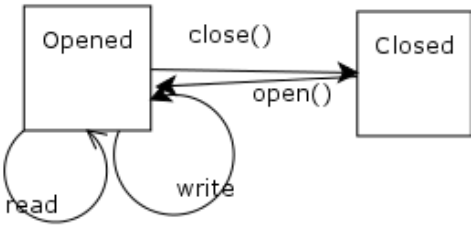
C'est une séquence d'invocation d'operations de ses interfaces offertes. Ce sont des séquences pertinentes (tirée des diagrammes de séquence inter-composant).

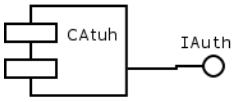
Exemple

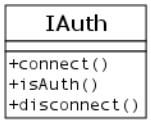

Controle
idCtrl
idUe
Titre
1
n
Note
idCtrl
idEtu
val

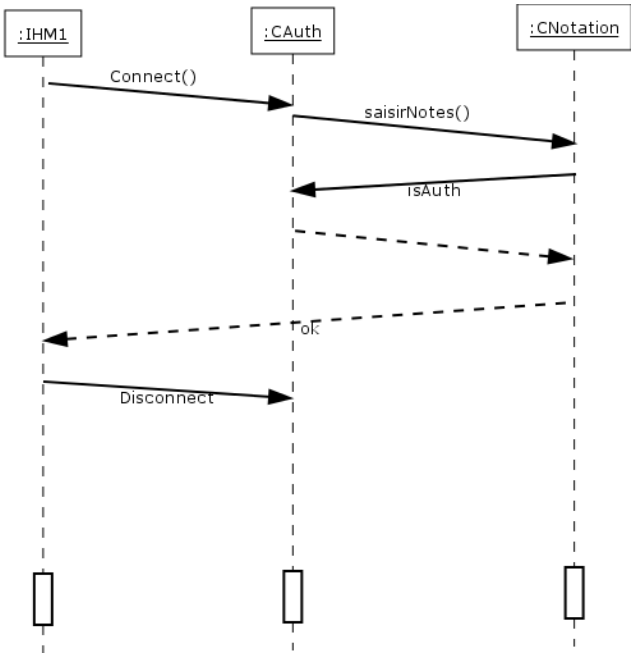

CFile
IFile


IFile
+open()
+read(int,): char[]
+write(char[])
+close()


Opened
close()
Closed
open()
read
write


CAtuh
IAuth


IAuth
+connect()
+isAuth()
+disconnect()


:IHM1
:CAuth
:CNotation
Connect()
saisirNotes()
isAuth
ok
Disconnect

Chapitre 5

Les Design Pattern

Éléments de solution orienté objets à des problèmes récurents.

5.1 Historique

Gang Of Four (GOF) -> Gamma, Helm, Vlissides, Johnson, '95, défini 20 patterns.

5.2 Type de patterns

5.2.1 Création

Comment créer les objets :
— Simple Factory
— Abstract Factory
— Factory Method
— Prototype
— Singleton

5.2.2 Structurels

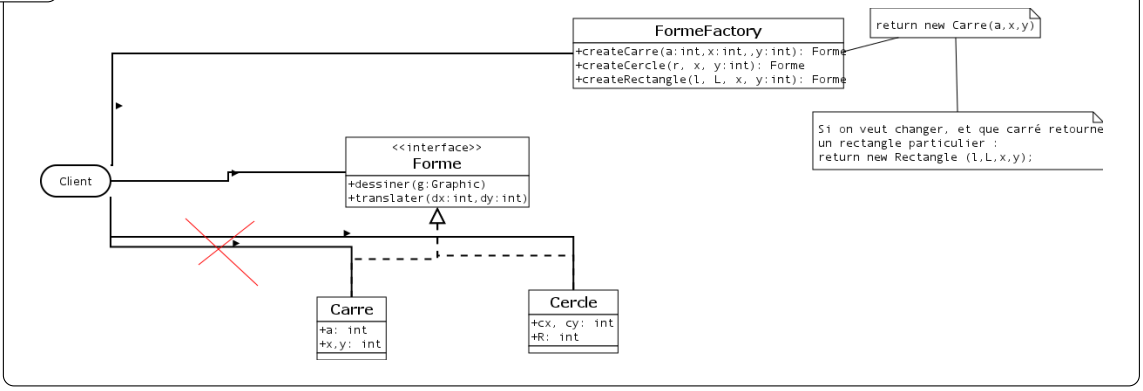
— Facade
— Adapter
— Proxy : plus d'éfficacité, fait semblant d'être le vrai objet mais que ne l'est pas tout a fait
— Composite : permet de gérer des structures arborescentes

5.2.3 Comportement

— Strategy
— Iterator : homogénéiser l'accès a la structure de donnée à travers une API très simple(itérator sur les feuilles d'un arbre)
— Command : Creer une classe qui représente un traitement
— State : mettre en pace des objets qui ont un état interne et les réactions de cet objet dépend de cet état interne

5.3 Factory

Exemple



Sans la factory

```
1 import Forme;
  import Carre;
3 import Cercle:
  Forme f1 = new Carre(20,30,30);
5 Forme f2 = new Cercle(20, 50, 40);
  ....
7 ....
  f1.translater(10,10);
9 f1.dessiner(g) .....
```

Nouvelle version

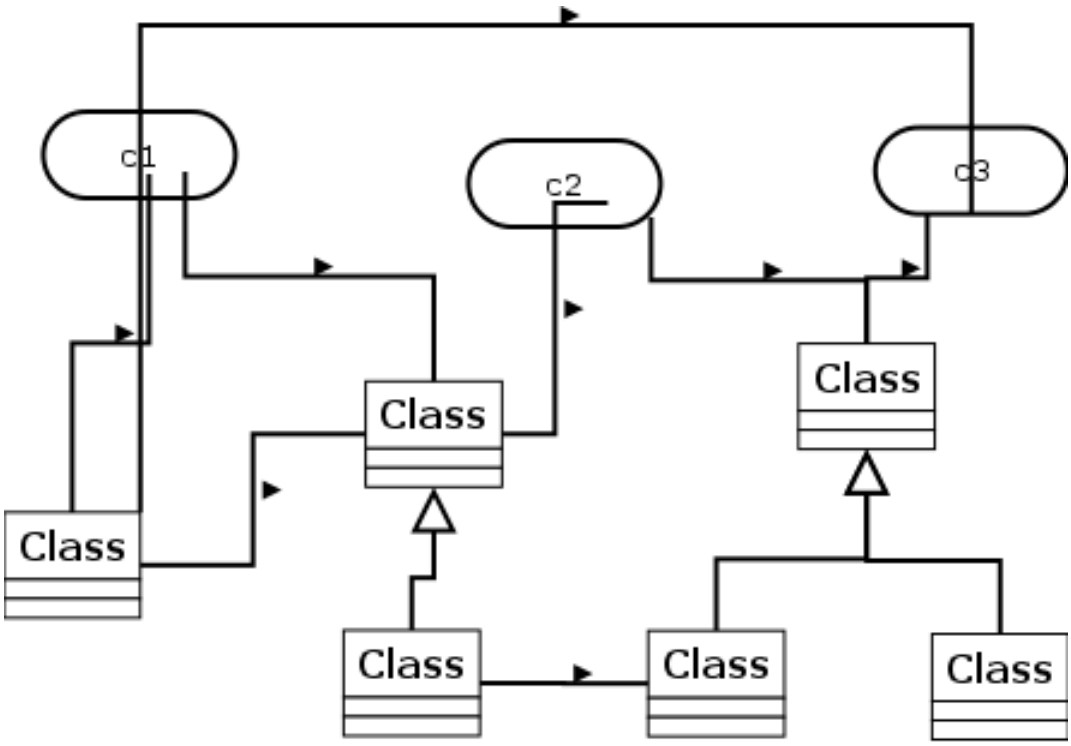
```
1 import FormeFactory;
  import Forme;
3 FormeFactory fac = ----------;
  Forme f1 = fac.createCarre(20,30,30);
5 Forme f2 = fac.createCercle(20, 50, 40);
  ....
7 ....
  f1.translater(10,10);
9 f1.dessiner(g) .....
```

Plus de problème d'ajout de nouvelle classe etc.. La factory permet d'instancier des abstractions tout en cachant les implémentations.

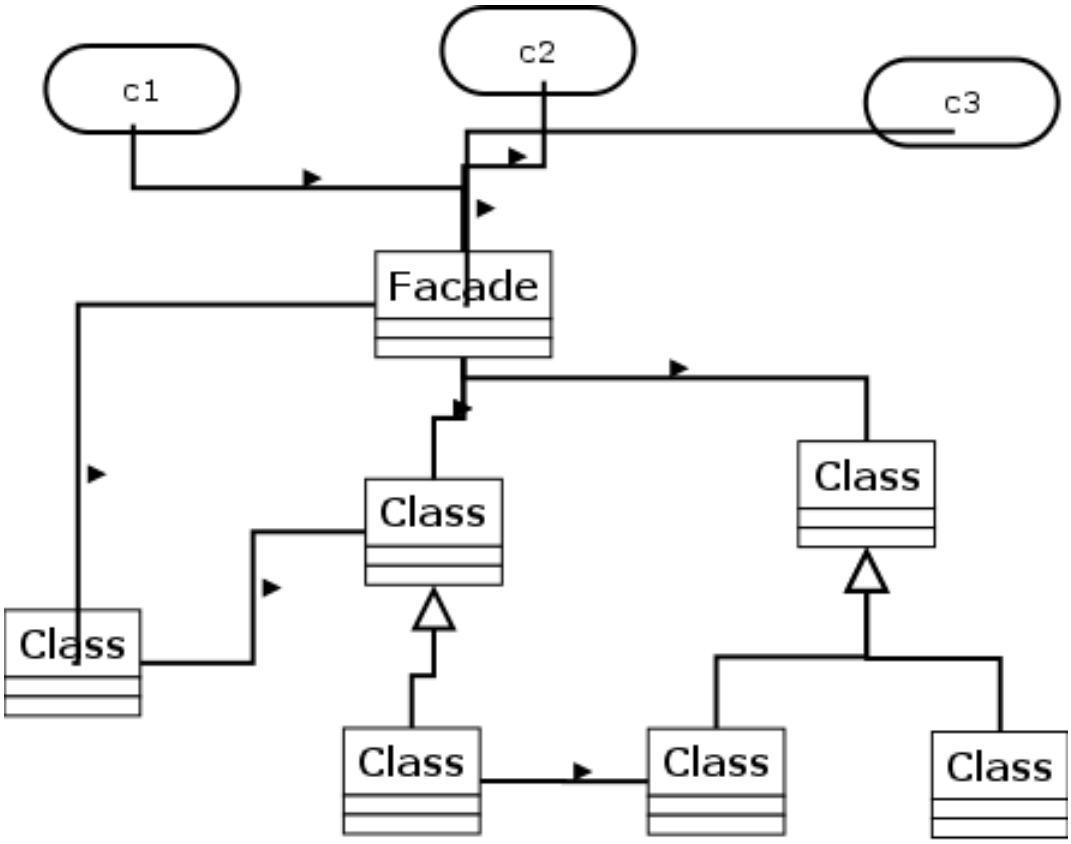
5.4 Abstract Factory

C'est l'identique d'une factory mais la classe factory est elle même abstraite ce qui permet encore plus de condifugration.

5.5 Facade



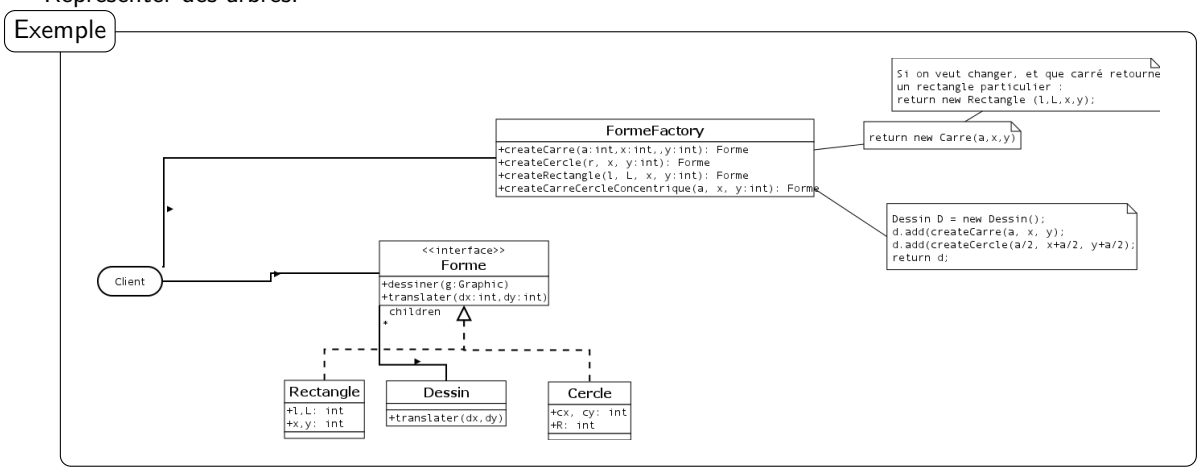
Vers :



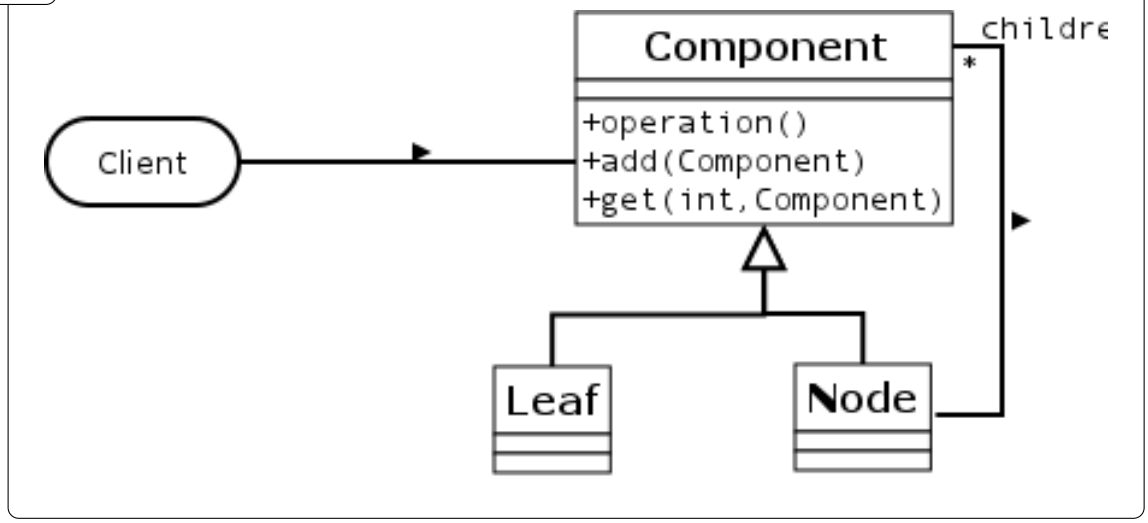
5.6 Composite

Représenter des arbres.

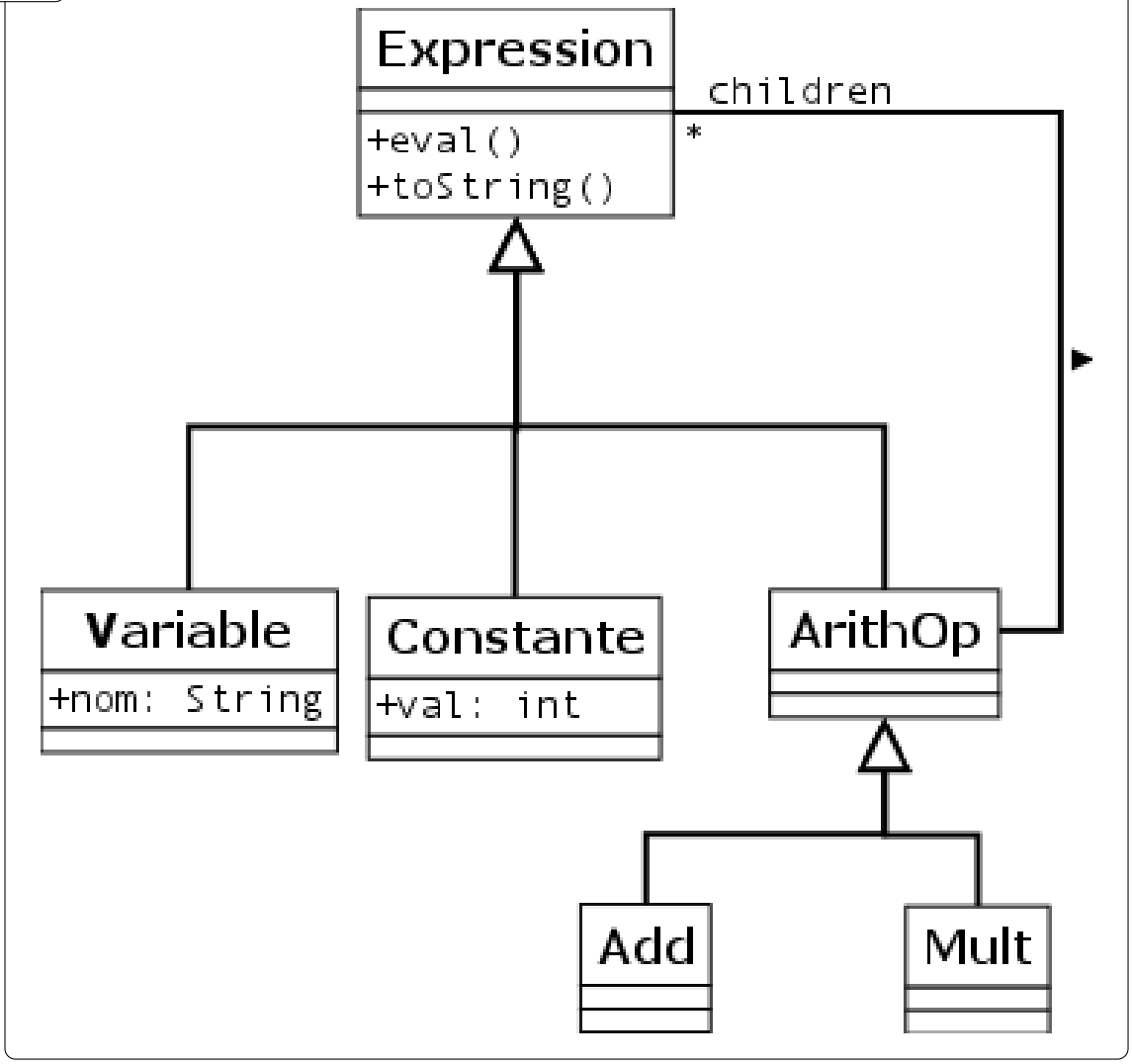
Exemple



Repres.



Exemple2



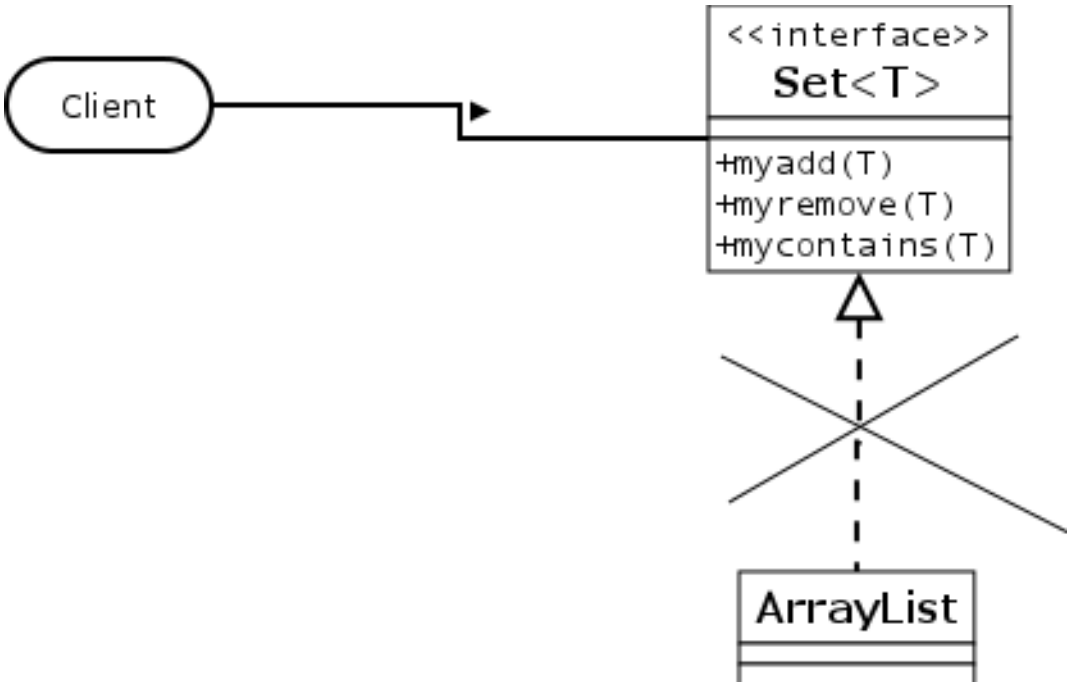
Dès que l'on parle de strucure arbre, il faut penser à composite car cela facilite grandement le traitement

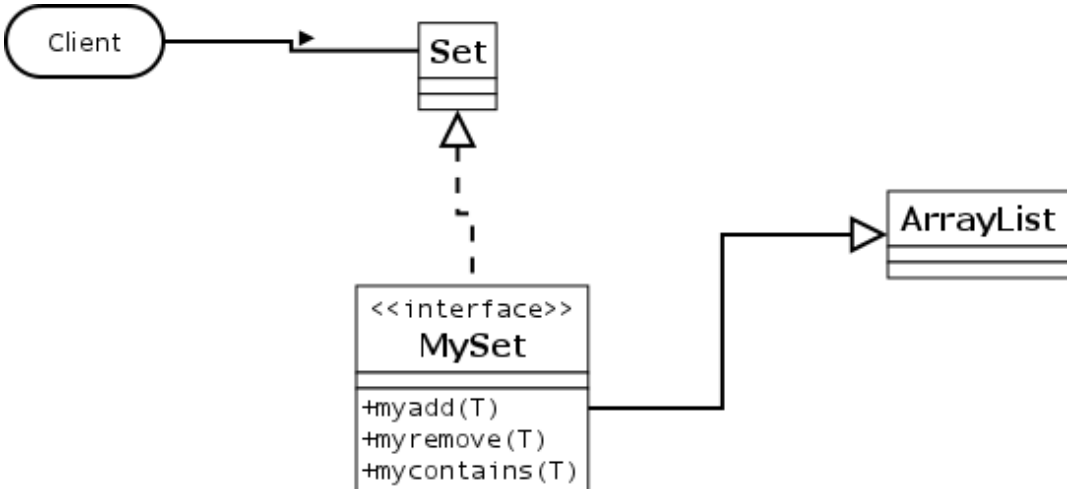
5.7 Adapter

Réutiliser l'existant sans modification. Permet de rajouter du comportement sur l'existant sans le modifier.

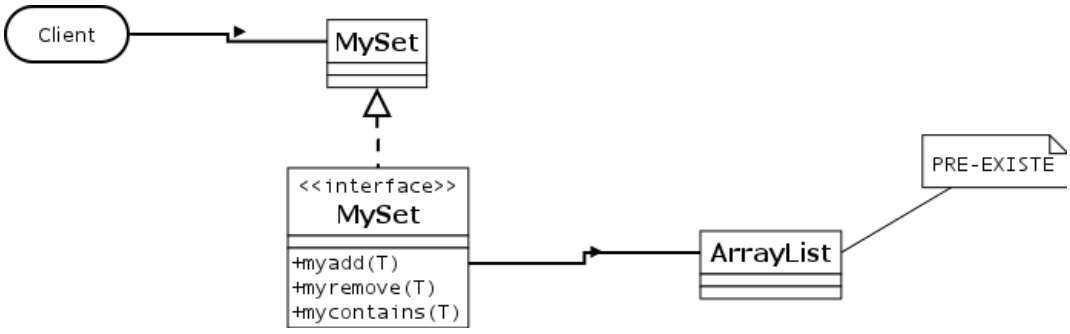
Exemple

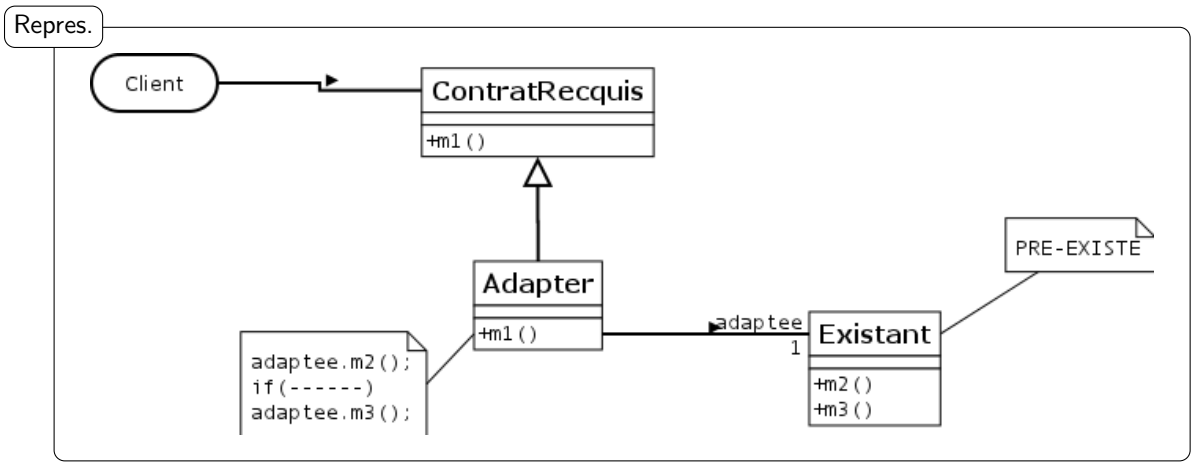
Problème :





Solution :




Repres.
Client
ContratRecquis
+m1()
Adapter
+m1()
adaptee
1
Existant
+m2()
+m3()
PRE-EXISTE
adaptee.m2();
if(------)
adaptee.m3();

5.8 Biblio

Chapitre 6

Tests d'intégration (suite)

6.1 Objectifs

— Permettre de dev en parallèle les composants
— Figer les interfaces des composants
— Préparer l'intégration

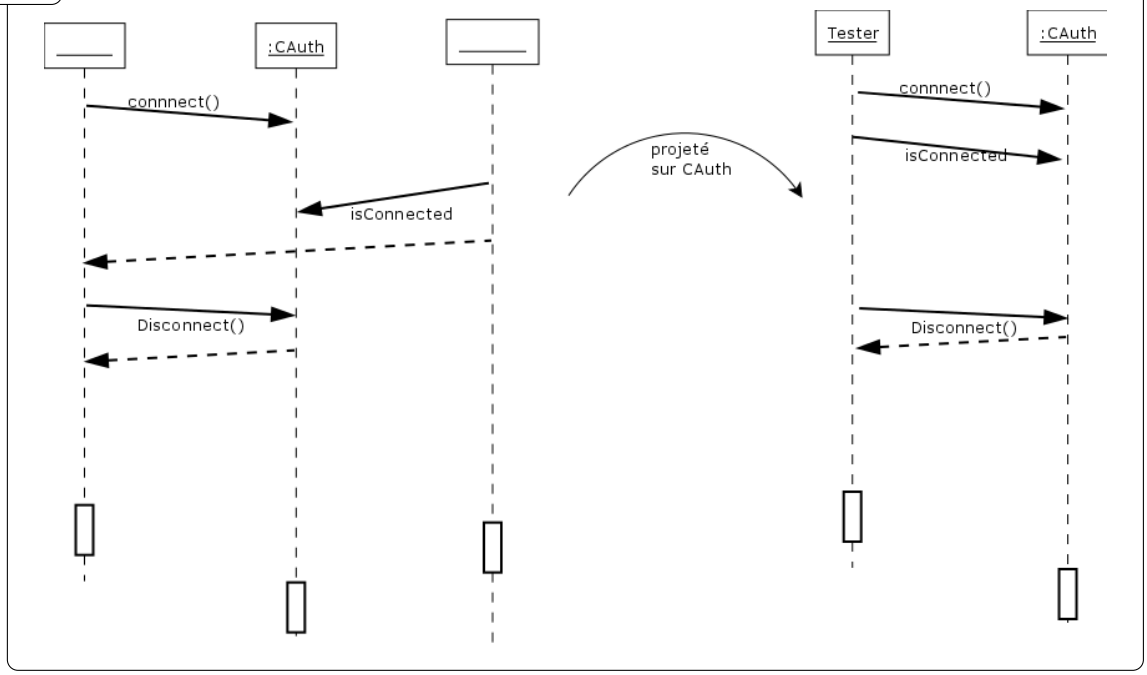
6.2 Nature

Séquence d'invocation à des opérations d'une interface offerte par le composant + résultat attendu.

6.2.1 Comment les obtenir ?

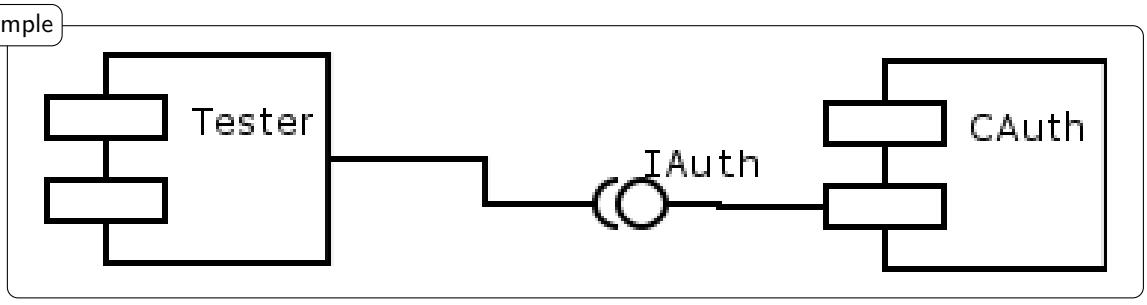
En partant de diagrammes de séquence inter-composants de conception arichitecturaux. On projette les invocations sur une ligne de vie.

Exemple



6.3 Configuration de test

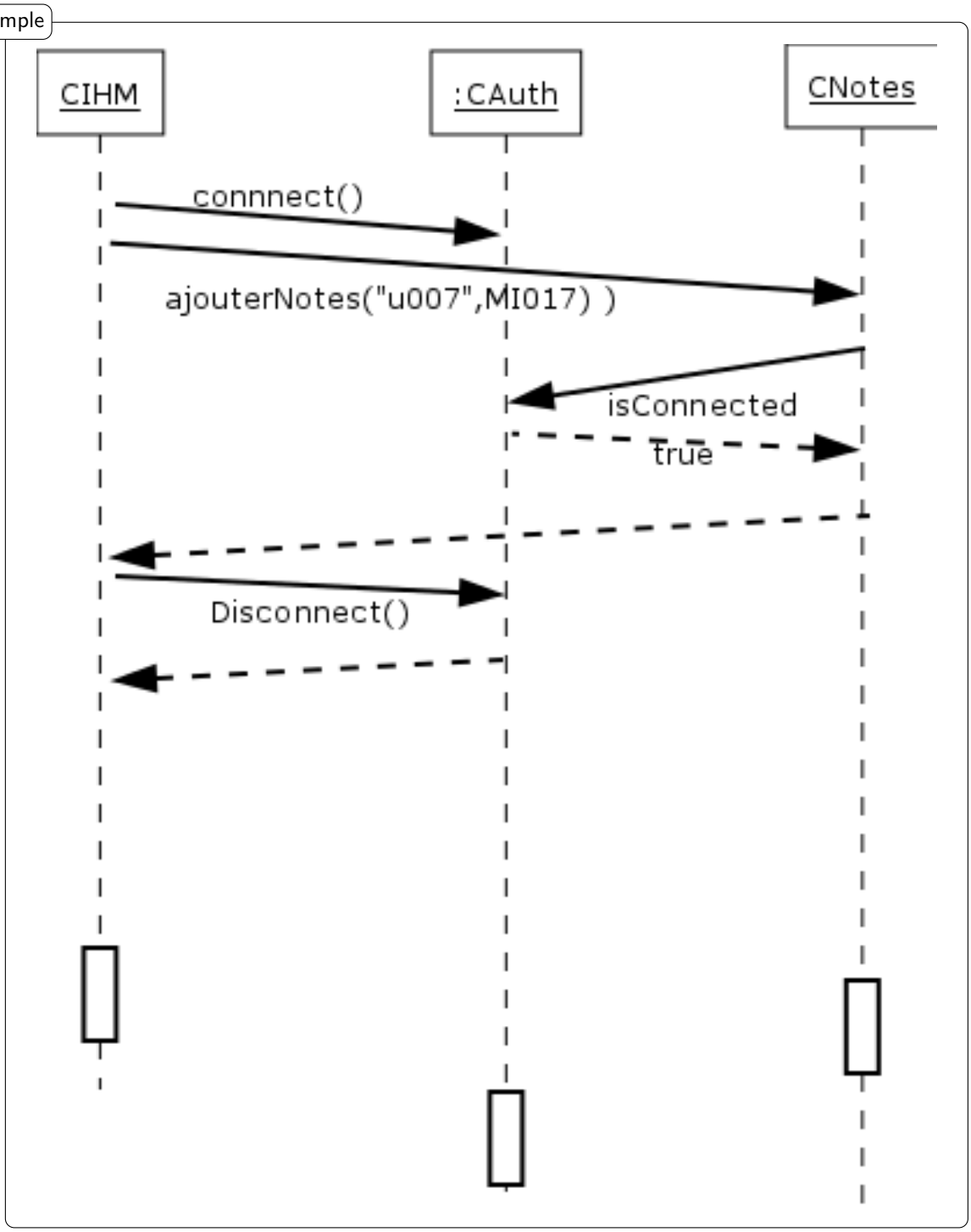
Exemple



On ajoute à ComposantFactory des opérations pour créer des configs de test.

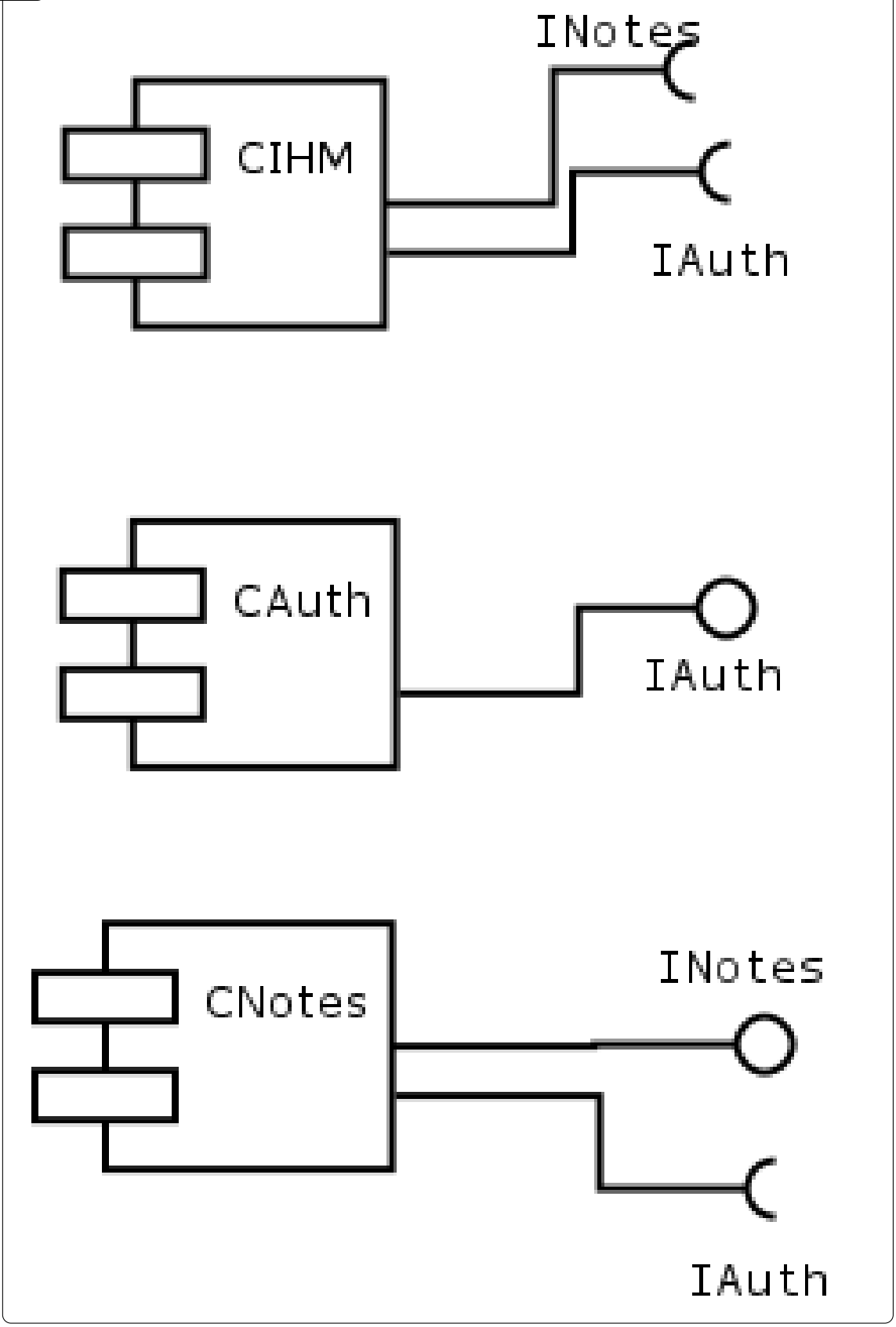
6.3.1 Si le composant requiert une interface ?

Exemple

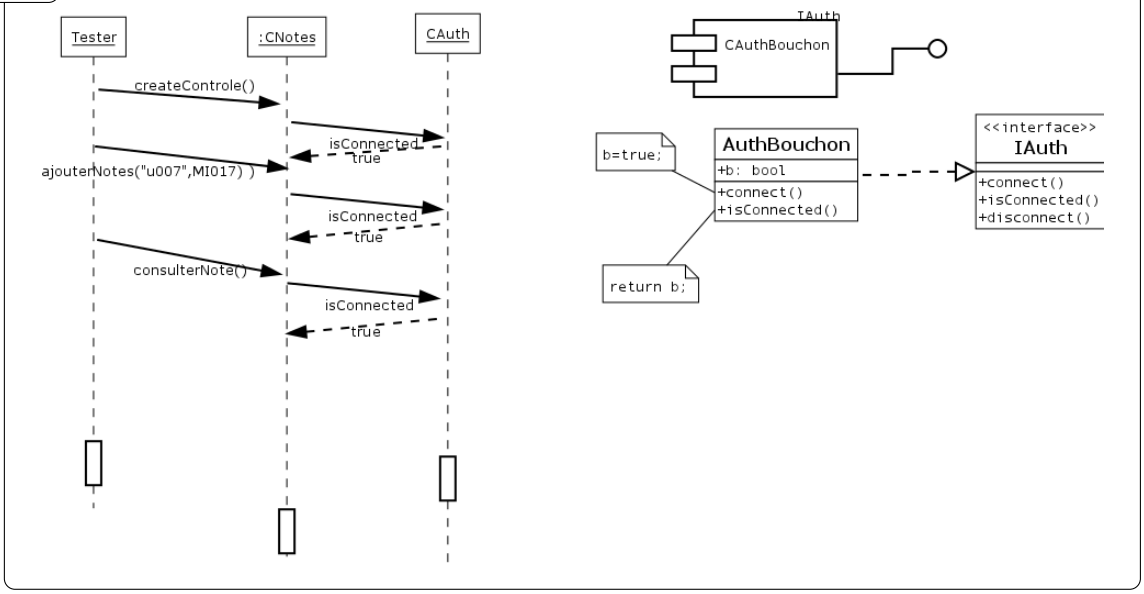


–> Mettre en place des *bouchons*.
Bouchon : implémentation naïve d'une interface qui permet de passer les tests.

Exemple



Exemple


Tester
:CNotes
CAuth
createControle()
isConnected
true
ajouterNotes("u007",MI017))
isConnected
true
consulterNote()
isConnected
true
IAuth
CAuthBouchon
b=true;
AuthBouchon
+b: bool
+connect()
+isConnected()
return b;
<<interface>>
IAuth
+connect()
+isConnected()
+disconnect()

Chapitre 7

Les Cycles de Developpement

7.1 Forces et Faiblesses du V

7.1.1 Avantages

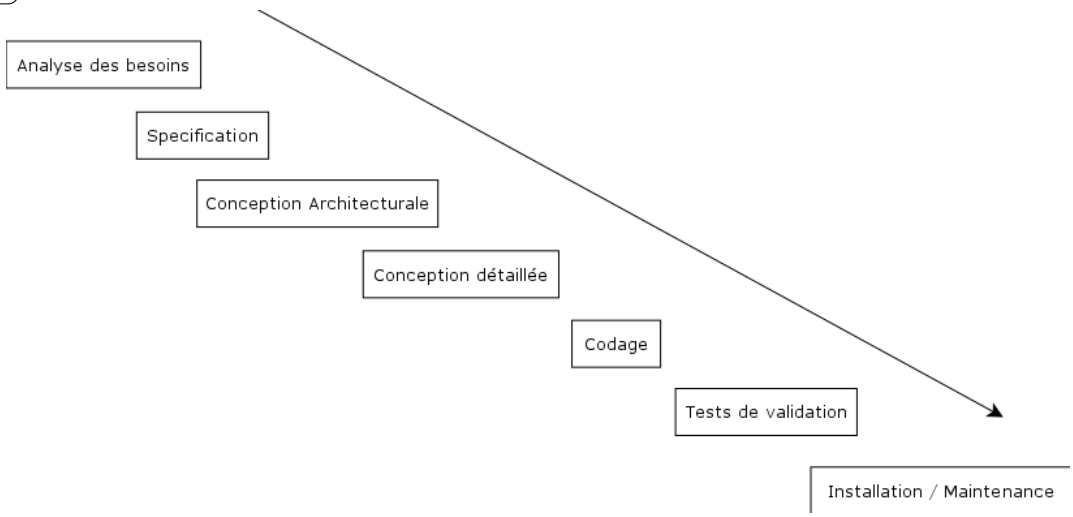
— Bien structuré
  — Branche de controle
  — Branche *Construction*
— Supporte les modèles (Analyse != Conception)

7.1.2 Défauts

— Séquentialité : Les phases s'enchainent et il n'y a pas de retour entre les phases
— Absence de retour client : Effet *Tunnel*, le client n'observe que le résultat final et n'intervient jamais dans le developpement. Gros problème business

7.2 Cycle Cascade

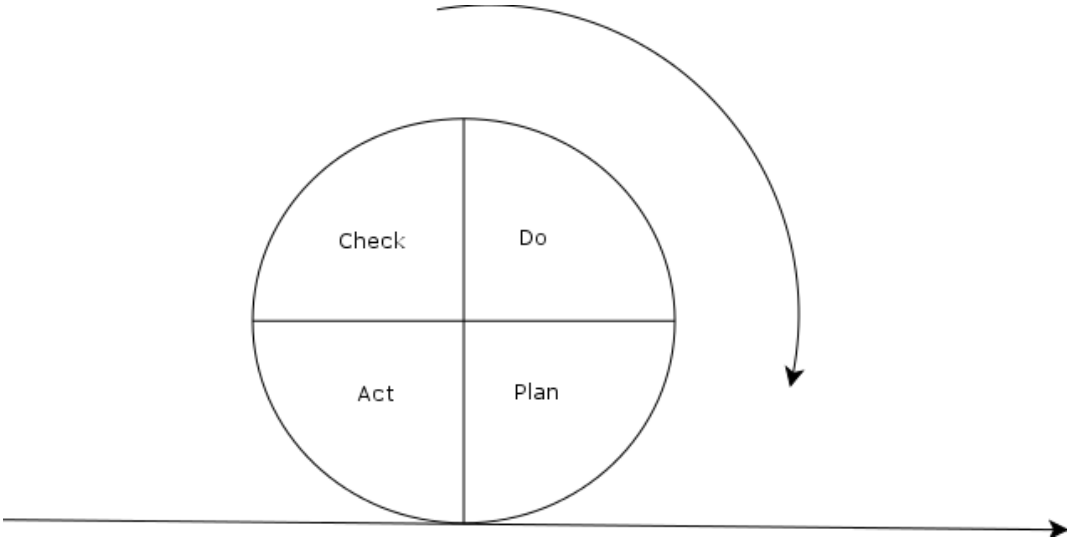
Repres.



7.3 Itération - Cercle Vertueux

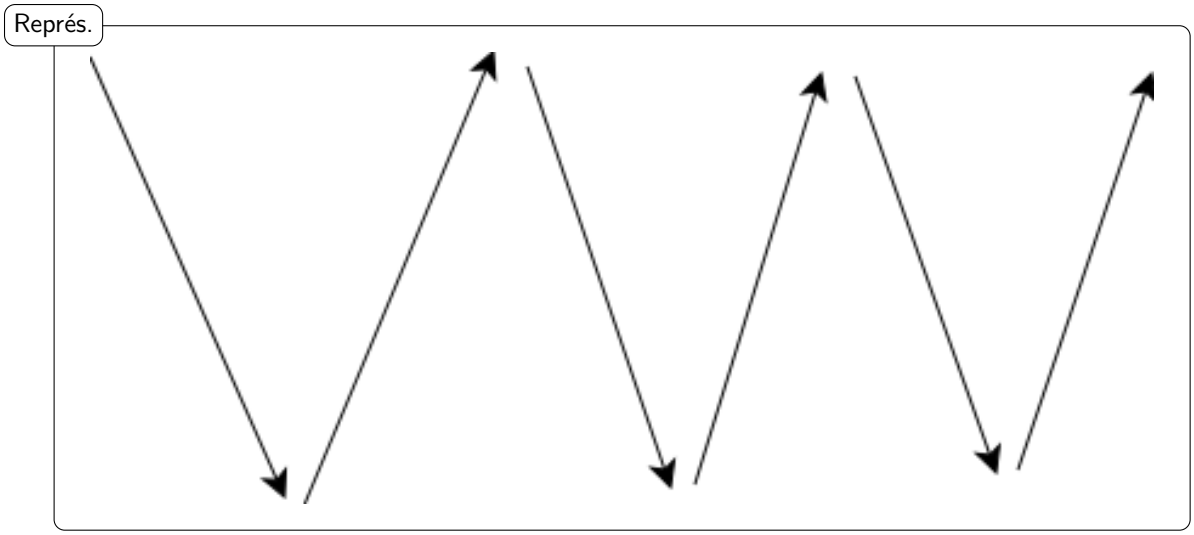
Repres.

Roue de Deming



Act : Agir pour corriger les problèmes

7.4 Cycle V itératif



7.5 Rational Unified Process

Framework de processus
- Model, Implem, Test
- Déploiement, Configuration
- Management Equipe

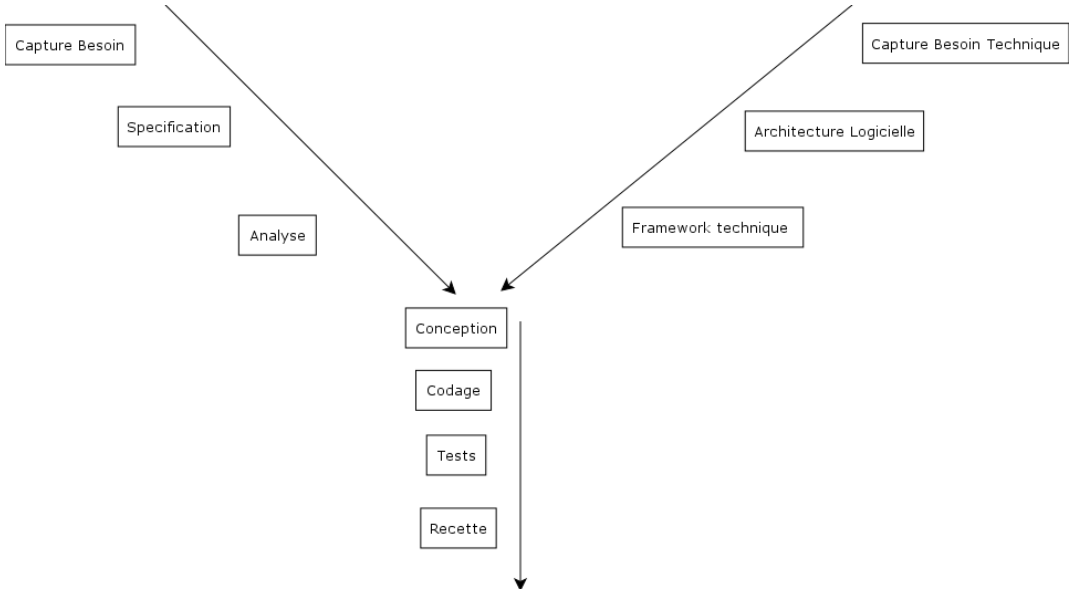
10 activités, 20 rôles
- Architecte
- Testeur, Project Manager
- Responsable Qualité

C'est un cycle itératif appuyé par les modèles UML. Il existe 4 phases :
- Inception
- Elaboration
- (Construction)
- Transition

Dérivables : Specs Techniques des besoins, Descriptions Architecture logicielle etc.

7.6 Cycle en Y

Prise en compte de la plateforme technique. Prendre en compte en framework utilisé lors du developpement.



7.7 Rapid Application Development - Methodes Agiles

Espère résoudre la problématique des *besoins changeants*. On va donc chercher un prototypage rapide. Prototype : Va etre raffiné au fil des itérations, et lorsqu'il comble les exigences, il deviendra une release. Une itération dure 30 jours et on expose la dernière version au client.
Cela implique l'utilisation de beaucoup d'environnements de développement et de framework techniques.

7.7.1 Critique

On critique généralement cette méthode car elle est souvent qualifiée de *bling bling*. On se préoccupe très tôt de l'interface graphique et sans beaucoup penser à la structure même de l'application (inverse de cycle en V).

7.7.2 Différents RAD

Agile, SCRUM, XP etc...
Il existe 4 phases :

- Besoins
- User Design : Conception basée sur les besoins de l'utilisateur, fait fortement participer le client
- Construction : Comprend le code, les test, et le déploiement. Solicite moins le client et plus l'équipe de dev
- Cutover : Etape de consolidation, introspection, prépare la transition pour la prochaine itération

7.7.3 Le manifeste AGILE

Créé en 2002, par une communauté de personnes bien placées dans grandes entreprises.

- On considère que les individus et les intéraction sont plus importantes que les processus de developpement et les outils

— Priviligéier les echanges verbaux et les interactions directes entres personne plutôt que de privilégier les interactions par des délivrables. Un logiciel éxécutable vaut mieux qu'une documentation exhaustive, il vaut mieux présenter au client une application (incomplète) plutot qu'une présentation papier
— Privilégier la coopération avec le client plutôt que la négociation dans un contrat (cahier des charges)
— Répondre au changement, plutôt que de suivre un plan

Permet d'intégrer des changements exterieurs (nouvelle "killer-app") pour produire une application la plus à jour possible.

7.7.4 SCRUM

Equivalent de la mélée de rugby. Est utilisé pour environ 3 à 10 personnes

Roles

— Product Owner : Celui qui veut vraiment voir le logiciel sortir, utilisateur, le client d'une certaine manière
— SCRUM Master : interface, permet de faire la relation entre l'équipe et le monde exterieur, gerer les manques de besoin materiel, technique, etc..
— Equipe : Equipe de developpement, codeurs, tout le monde a le même statut
— Stake holder : Quelqu'un qui finance le projet, pas directement le client, mais intéréssé par le projet, participe aux réunions mais n'intervient pas

Différence PIGS & CHICKEN : Pigs correspond à l'équipe de developpement et est directement impactée sur le bon déroulement du projet, contrairement au chicken (Stake holder), qui n'est pas complètement impliquée par le projet.

Developpement

— Sprint : entre 7 jours et 30 jours, un ensemble d'actions à effectuer (Backlog)
— Début de sprint : Planning : inferieur à une journée, discussion avec le product owner "I as XX, want YY (to ZZ)"<-item du backlog
  Taches + pricing, importance du client, difficulté de developpement
— Backlog du Sprint : ensemble de taches 'réalisables'
— Sprint : cycle quotidient
— Daily Scrum (inferieur à 15 minutes) : Qu'ai-je fait hier ? Que vais-je faire aujourd'hui ? Quels sont mes problèmes ?
— Suivi : Burndown chart
— Sprint Retrospective : réunion de l'équipe : Qu'est ce qui était bien ? Qu'est ce qu'on peut améliorer ?
— Sprint Review : Peut se faire avec les Stake Holder, etat d'avancement et DEMO

Critiques

— Qualité du code et manque de recul sur la structure de l'application
— Défaut du point de vue humain

7.7.5 eXtreme Programming (XP)

Activités

4 activités
— Ecoute du client
— Déroulement des tests, on va être très influencé par les test et leur déroulement
— Codage
— Conception

Valeurs

— Communication : orale, parler avec le client
— Simplicité : Ne pas définir une interface avant d'avoir au moins 2 implémentations, toujours viser quelque chose le plus simple possible(YAGN -> You're not gonna need it, KISS -> Keep it simple stupid)
— Courage : Revenir en arrière, devoir détruire du code déjà écrit
— Feedback : Itérations pas supérieures à la semaine, on a régulièrement des critiques du client et le prendre en compte
— Respect : Ne pas mettre de code qui ne compile pas, respect des autres

Pratiques

— Test Driven : Developpement dirigé par les test, écrire les test avant l'implémentation, lundi réunion, puis du lundi aprem jusqu'au mardi, élaboration des test, puis le reste de la semaine on fait en sorte que les test passent au green. Valeur des test : Red (le test ne passe pas) , Green (le test passe), Refactoring (on réfléchis au code pour améliorer le code)
— Pair Programming : Programmation en binôme
— Intégration continue : lié sur le gestionnaire de versions + un serveur de tests (Jenkins, Teamcity)
— Responsabilité collective : pas de hierarchie, tout le code est à tout le monde, pas de responsable pour une tâche

7.8 Globalement

RAD : bien pour les petites structures (moins de 10 personnes). Pour les grandes équipes ont utilise plus généralement des approches RUP avec délivrables et contrat, car lorsque l'on arrive sur des grands projets, les communications orales sont très difficiles.