Le cycle de vie d'un logiciel : Conception

- ***La phase d'analyse*** ***(définition des besoins et spécification fonctionnelle et techniques)*** est **suivie de la phase de conception**.

- Généralement ***la conception*** est décomposée en deux phases successives :

  - ***Conception générale*** ou conception architecturale (preliminary design ou architectural design)

  - **Conception détaillée (*detailed design*)**

Le cycle de vie d'un logiciel : Conception

- ***Conception générale :***

Décrire l'architecture de la solution, c'est-à-dire son organisation en entités, les interfaces de ces entités et les interactions entre ces entités.

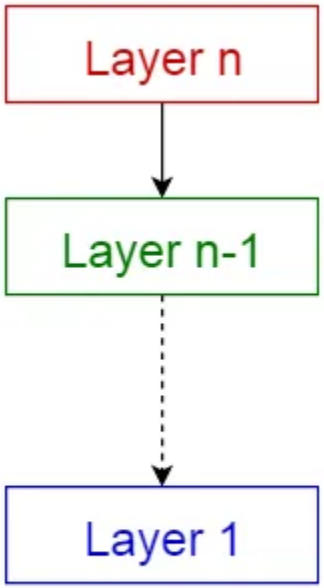
Le cycle de vie d'un logiciel : Conception

- ***Conception générale :*** Architecture logicielle courantes

❑ **Modèle en couches (Layered pattern)**

Les 4 couches les plus courantes d'un système d'information général sont les suivantes. -

- Couche de présentation (également appelée couche d'interface utilisateur) .
- Couche d'application (également appelée couche de service)
- Couche de logique métier (également appelée couche de domaine) .
- Couche d'accès aux données (également appelée couche de persistance).



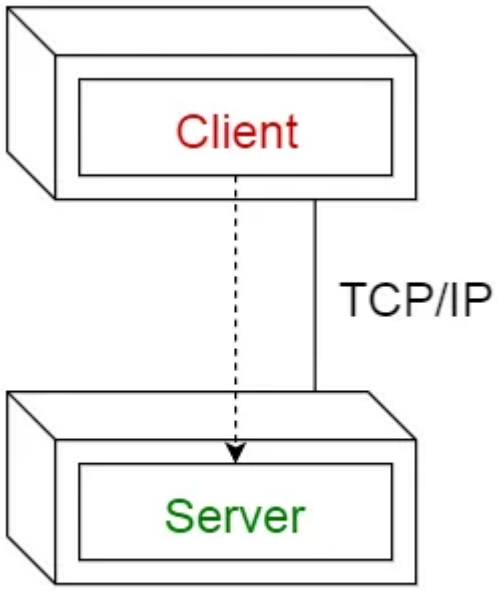
Layered pattern

Le cycle de vie d'un logiciel : Conception

- ***Conception générale :*** Architecture logicielle courantes

❑ **Modèle client-serveur (Client-server pattern)**

Ce modèle se compose de deux parties; un serveur et plusieurs clients.
Les clients demandent des services au serveur
Le serveur fournit des services pertinents à ces clients.



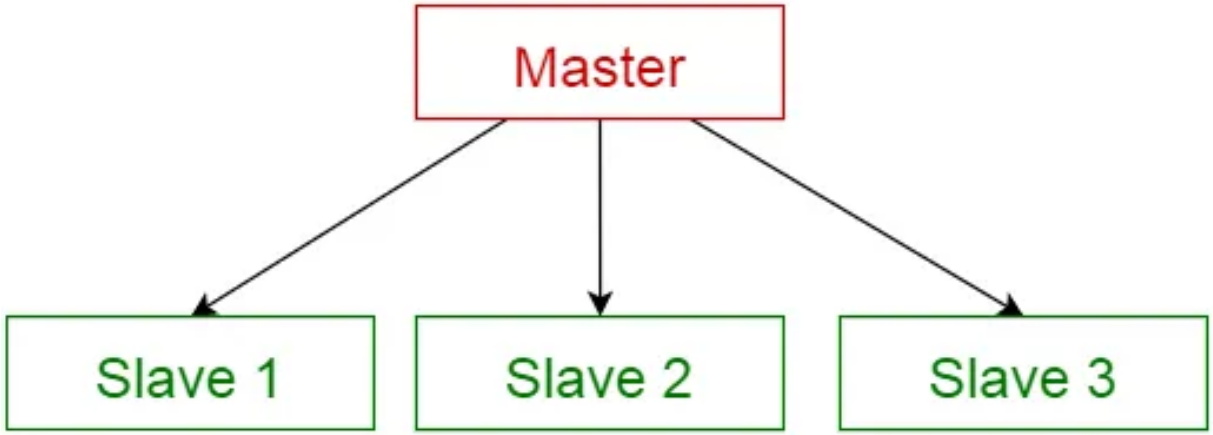
Client-server pattern

Le cycle de vie d'un logiciel : Conception

- ***Conception générale :*** Architecture logicielle courantes

❑ **Modèle maître-esclave (Master-slave pattern)**

Ce modèle se compose de deux parties: maître et esclaves.
Le composant maître distribue le travail entre des composants esclaves identiques et calcule un résultat final à partir des résultats renvoyés par les esclaves.



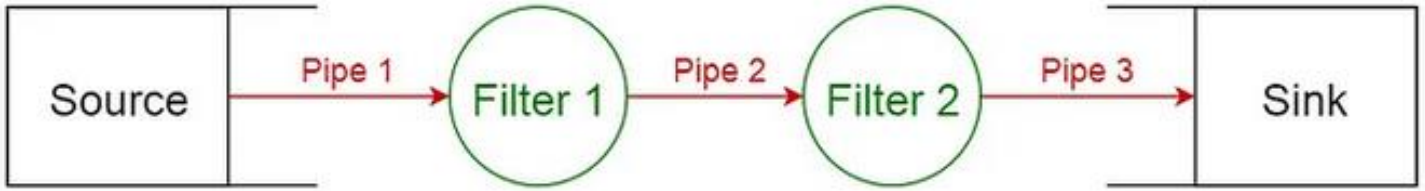
Master-slave pattern

Le cycle de vie d'un logiciel : Conception

- ***Conception générale :*** Architecture logicielle courantes

❑ **Modèle de canal-filtre (Pipe-filter pattern)**

Ce modèle peut être utilisé pour structurer des systèmes qui produisent et traitent un flux de données.
Chaque étape de traitement est enfermée dans un composant de filtre.
Les données à traiter passent par des canaux.
Ces canaux peuvent être utilisés pour la mise en mémoire tampon ou à des fins de synchronisation.



Le cycle de vie d'un logiciel : Conception

- ***Conception générale :*** Architecture logicielle courantes

❑ ***Modèle de courtier (*Broker pattern)**

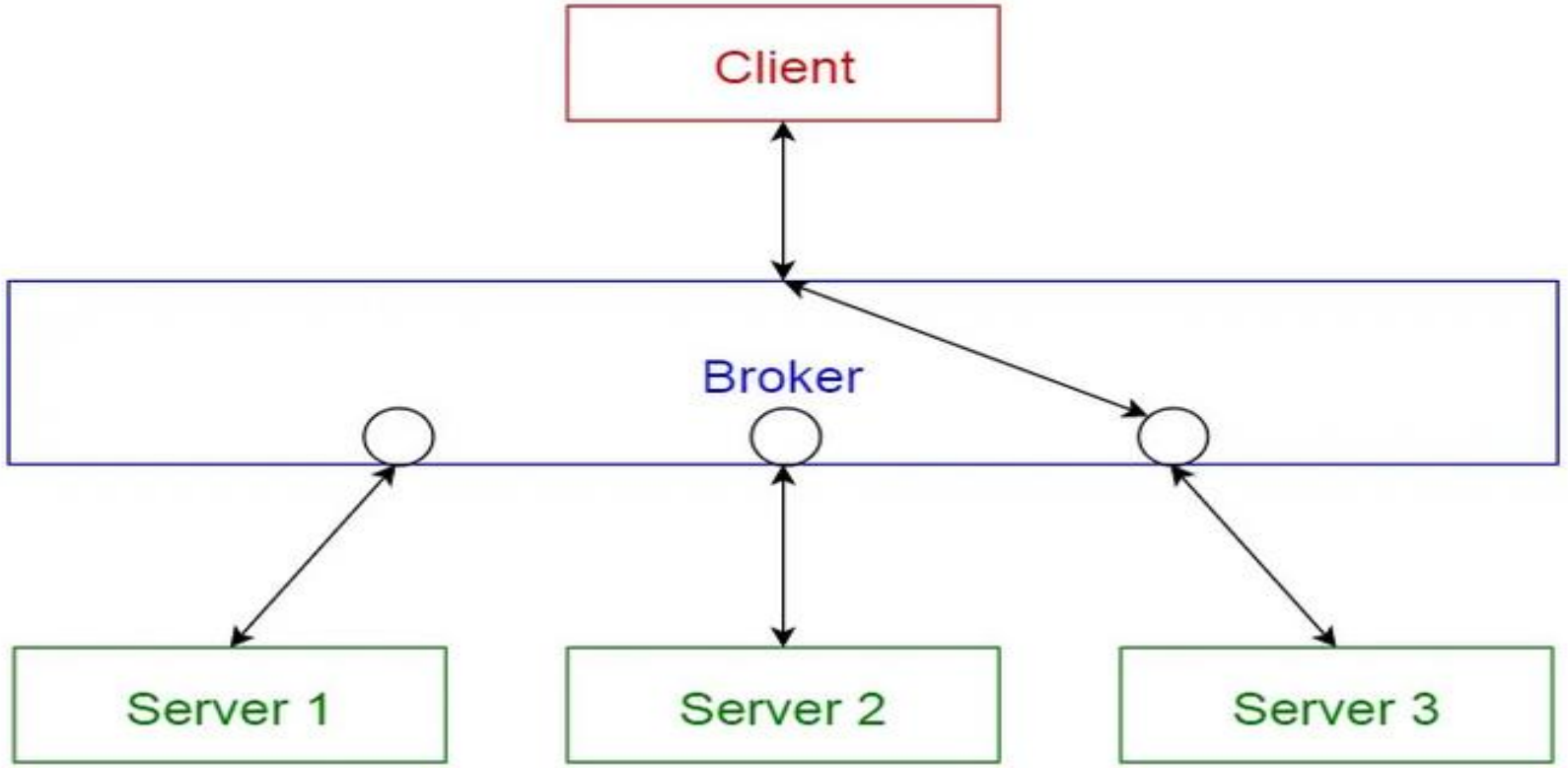
- Ce modèle est utilisé pour structurer des systèmes distribués avec des composants découplés.
- Ces composants peuvent interagir les uns avec les autres par des appels de service à distance.
- Un composant courtier est responsable de la coordination de la communication entre les composants.
- Les serveurs publient leurs capacités (services et caractéristiques) à un courtier.

- Les clients demandent un service au courtier, et le courtier redirige ensuite le client vers un service approprié à partir de son registre.

Le cycle de vie d'un logiciel : Conception

- ***Conception générale :*** Architecture logicielle courantes

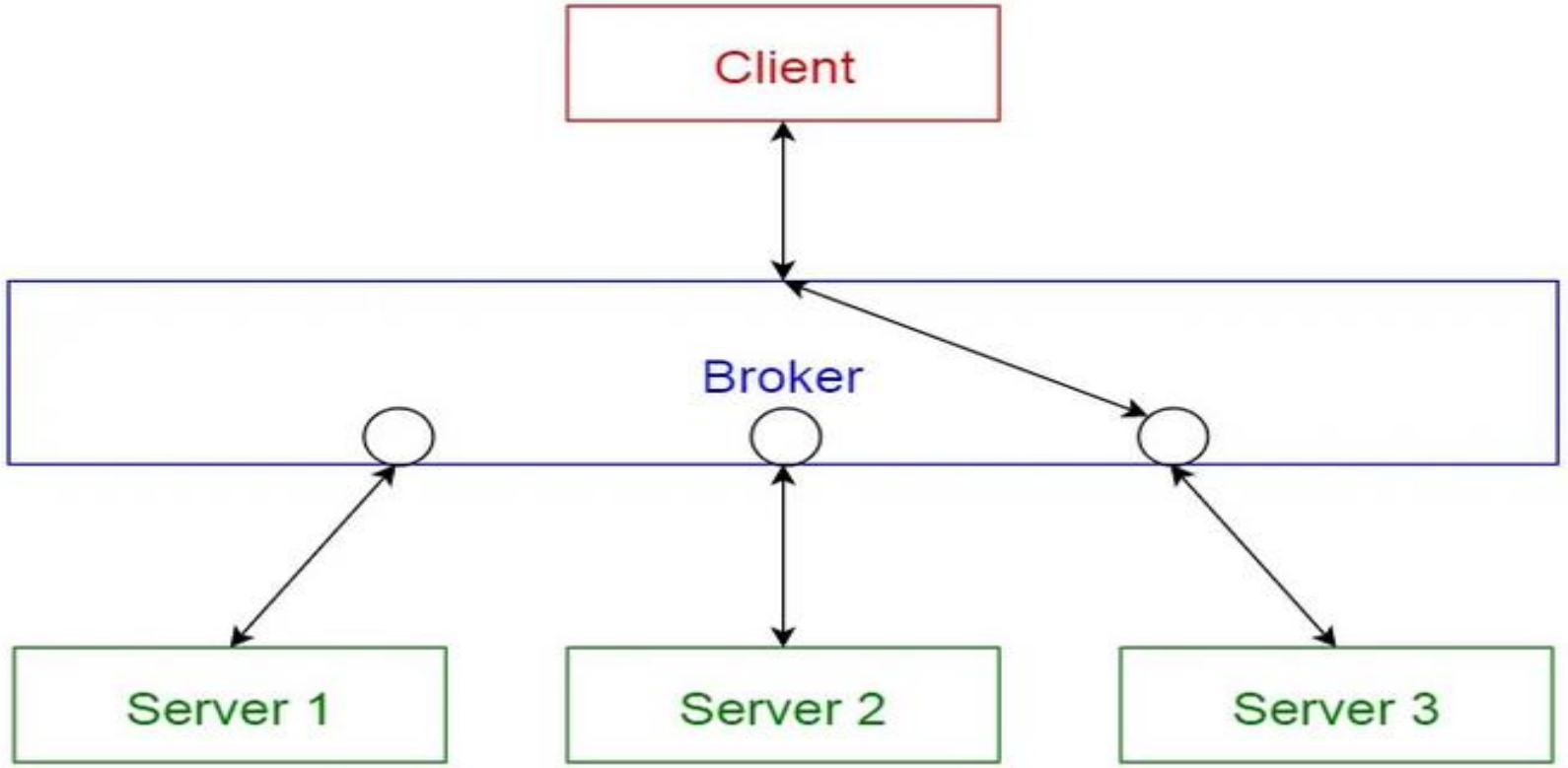
❑ ***Modèle de courtier (*Broker pattern)**



Le cycle de vie d'un logiciel : Conception

- ***Conception générale :*** Architecture logicielle courantes

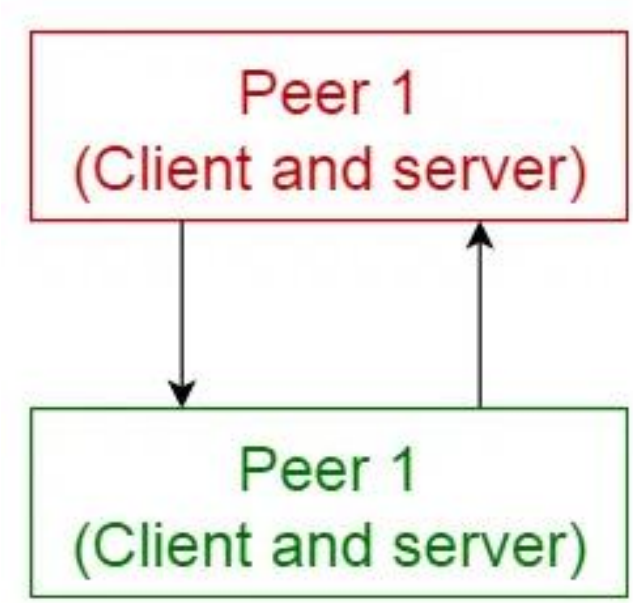
❑ ***Modèle de courtier (*Broker pattern)**



Le cycle de vie d'un logiciel : Conception

- ***Conception générale :*** Architecture logicielle courantes

***Modèle* Peer-to-Peer**



Peer-to-peer pattern

Le cycle de vie d'un logiciel : Conception

- ***Conception générale :*** Architecture logicielle courantes

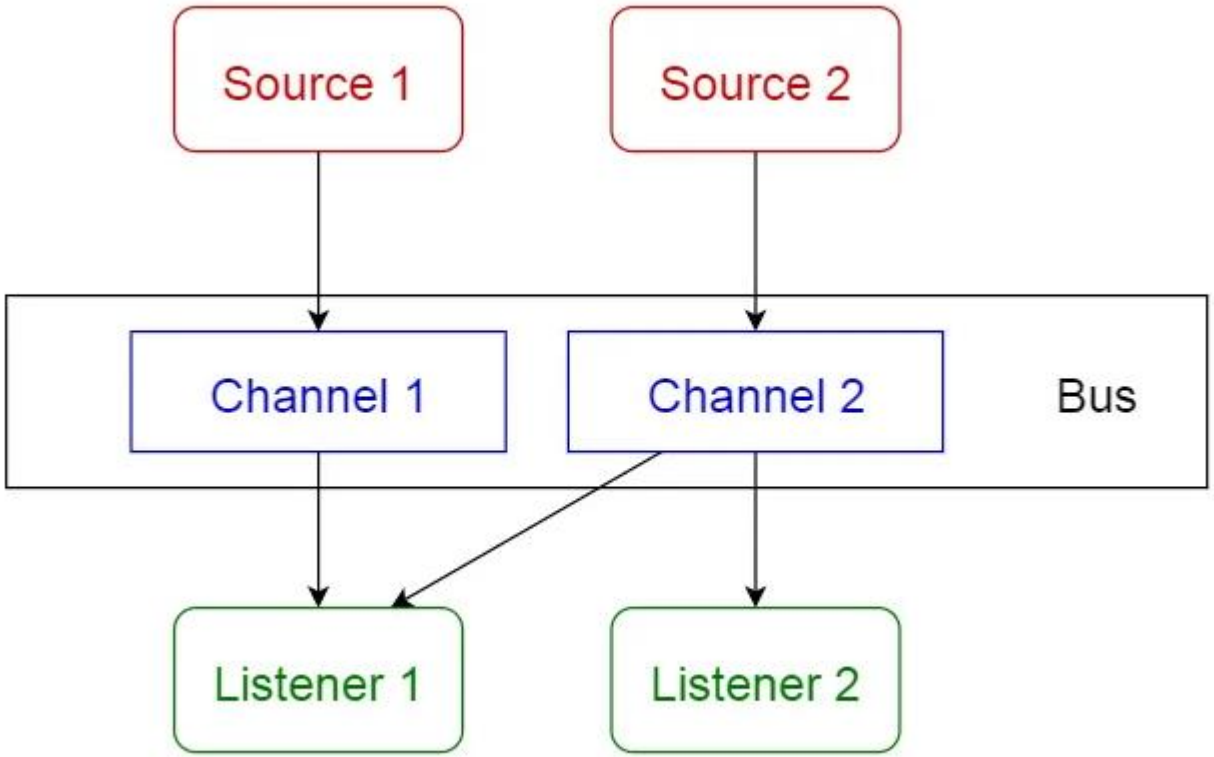
❑ ***Modèle Evénement – Bus (*Event-bus pattern)**

Ce modèle traite principalement des événements et comporte 4 composants principaux :
Source d'événement, écouteur d'événement, canal et bus d'événement.
Les sources publient des messages sur des canaux particuliers sur un bus d'événements.
Les auditeurs s'abonnent à des chaînes particulières.
Les auditeurs sont informés par des messages qui sont publiés sur une chaîne à laquelle ils se sont déjà abonnés.

Le cycle de vie d'un logiciel : Conception

- ***Conception générale :*** Architecture logicielle courantes

❑ ***Modèle Evénement – Bus (*Event-bus pattern)**



Event-bus pattern

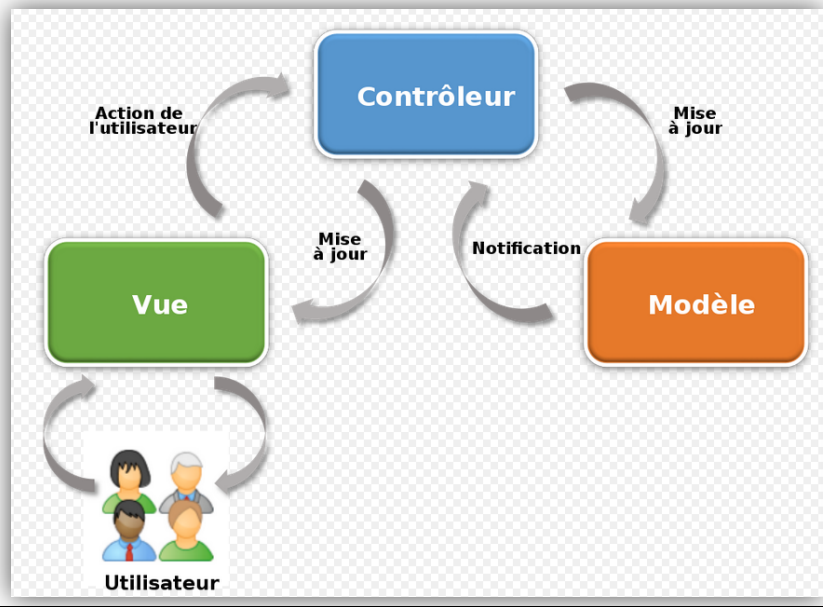
Le cycle de vie d'un logiciel : Conception

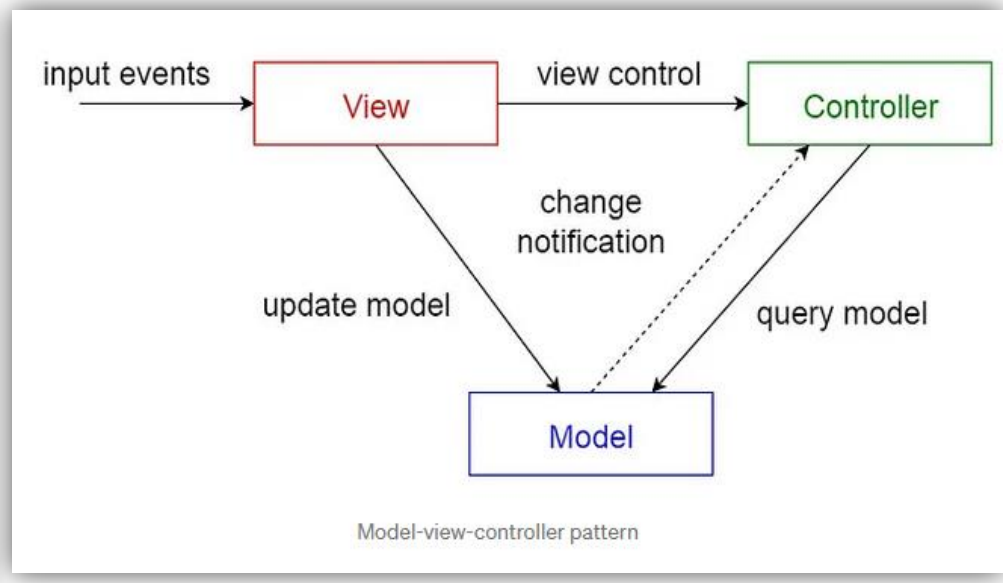
- ***Conception générale :*** Architecture logicielle courantes

❑ Modèle MVC (modèle-vue-contrôleur)

Divise une application interactive en 3 parties

- Un modèle (Model) contient les données à afficher ;
- Une vue (View) contient la présentation de l'interface graphique ;
- Un contrôleur (Controller) contient la logique concernant les actions effectuées par l'utilisateur.





Le cycle de vie d'un logiciel : Conception

- ***Conception générale :*** Architecture logicielle courantes

❑ Modèle de tableau noir (Blackboard pattern)

Ce modèle est utile pour les problèmes pour lesquels aucune stratégie de solution déterministe n'est connue. Le modèle de tableau noir se compose de 3 composants principaux.

- **Tableau noir** :une mémoire globale structurée contenant des objets de l'espace de solution.
- **Source de connaissances** : modules spécialisés avec leur propre représentation.
- **composant de contrôle** : Sélectionne, configure et exécute les modules.

Tous les composants ont accès au tableau noir.
Les composants peuvent produire de nouveaux objets de données qui sont ajoutés au tableau noir.
Les composants recherchent des types particuliers de données sur le tableau noir et peuvent les trouver par correspondance de modèle avec la source de connaissances existante.

Le cycle de vie d'un logiciel : Conception

- ***Conception détaillé* (*detailed design*) *:***

❑ La conception détaillée **affine** la conception générale.

❑ **Décompose** les entités découvertes lors de la conception générale en entités plus élémentaires (des ***composants*** logiciels élémentaires).

❑ Il faut ensuite **décrire chaque composant logiciel en détail** : ***son interface, les algorithmes utilisés, le traitement des erreurs, ses performances, etc***.

L'ensemble de ces descriptions constitue **le document de conception détaillée.**

- Exemple : voir document «Software Detailed Design Template »

Le cycle de vie d'un logiciel : Tests

Tests

Le test est un ensemble d'activités utilisé pour tester le code source afin de découvrir (et corriger) les erreurs avant la livraison du logiciel client.

Objectif : avoir un maximum de garanties sur le bon fonctionnement du système.

Classification des tests :

1. Les modalités de test : **Statique** / **Dynamique**
2. Les méthodes de test : **Structurelle** / **Fonctionnelle**
3. **Manuel** / **Automatique**
4. Les niveaux de tests : **Unitaire** / **Intégration** / **Système** / **Non-régression**

Le cycle de vie d'un logiciel : Tests

Tests : statique ou dynamique

Test dynamique

on exécute le programme avec des valeurs en entrée et on observe le comportement

Test statique

relecture / revue de code et analyse automatique (vérification de propriétés, règles de codage, typage ...)

Le cycle de vie d'un logiciel : Tests

- ***Les niveaux des Tests :***

On peut classer les tests logiciels en différentes catégories :

Tests unitaire :

- **Un test unitaire** est une procédure permettant de vérifier le bon fonctionnement d'une **partie précise d'un programme.**
- En programmation procédurale, une unité est une fonction.
- En programmation orientée objet, une unité est une classe.

Exemple : tester une classe en vérifiant que toutes ces méthodes n' engendrent pas d'erreur d'exécution.

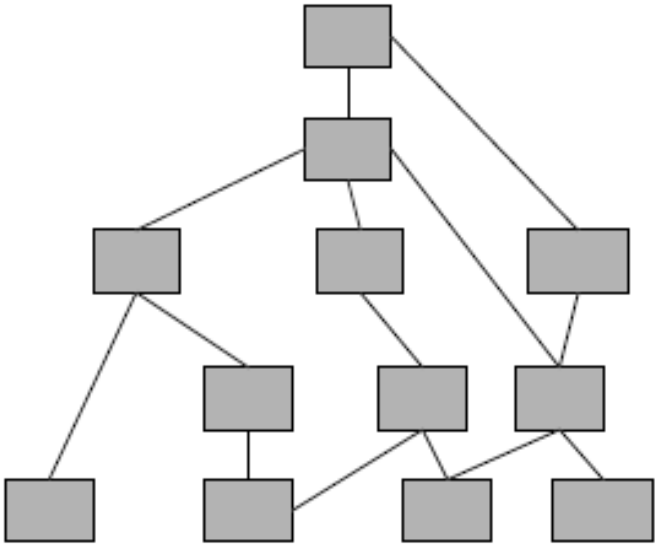
Le cycle de vie d'un logiciel : Tests

Tests d'intégration:

- ***Un test d'intégration*** est une procédure permettant de vérifier le bon fonctionnement d'un logiciel suite à l'assemblage de modules indépendants.

Difficultés :

Construire un **ordre permettant de tester graduellement l'assemblage** des unités (**problème de dépendance**).



Le cycle de vie d'un logiciel : Tests

Tests de validation :

- s'assure que **le produit final** correspond à ce qui a été demandé (dans le document de spécification du logiciel) .
- Lors de ces tests, on valide la globalité du système, i.e. les fonctions offertes, souvent à partir de l'interface.
- Ce genre de tests généralement est manuel mais il peut aussi être automatisé.

Le cycle de vie d'un logiciel : Tests

Tests de non régression :

Un test consiste à vérifier que des modifications apportées au logiciel n'ont pas introduit de nouvelles erreurs.

Quand ?

- Dans la phase de maintenance,
- Après modification du code,
- ajout/suppression de fonctionnalités,
- après la correction d'une faute.

Le cycle de vie d'un logiciel : Tests

Les méthodes de test : Structurelle/Fonctionnelle

- Lors d'un **test *fonctionnel***, on se réfère uniquement à l'observation de la sortie pour certaines valeurs d'entrée.
  Aucune tentative d'analyser le code,***(Test boite noire)***

- Lors d'un **test *structurel***, on exploite la connaissance de la structure et la mise en œuvre du logiciel. ***(Test boite blanche)***
  Possibilité de fixer finement la valeur des entrées pour sensibiliser des chemins particuliers du code;

Le cycle de vie d'un logiciel : Tests

Les méthodes de test : Structurelle/Fonctionnelle

- Exemple de **test *fonctionnel***:

Code python de l'addition

```
def addition(x,y):
    if y==0:
        somme = x
    else:
        somme = x + y
    return somme
```

Exemple de test fonctionnel

```
def test_addition():
  x = 4
  y = 5
  valeur_attendue = 9
  if addition(x,y) == valeur_attendue:
     print("test réussi")
  else:
     print("test échoué")
```

Le cycle de vie d'un logiciel : Tests

Les méthodes de test : Structurelle/Fonctionnelle

❑ Exemple de **test *structurelle***:

Code python pgcd

```
def pgcd(p,q):
  while(p!=q):
   if p>q:
     p=p-q
   else:
     q=q-p
  return p
```

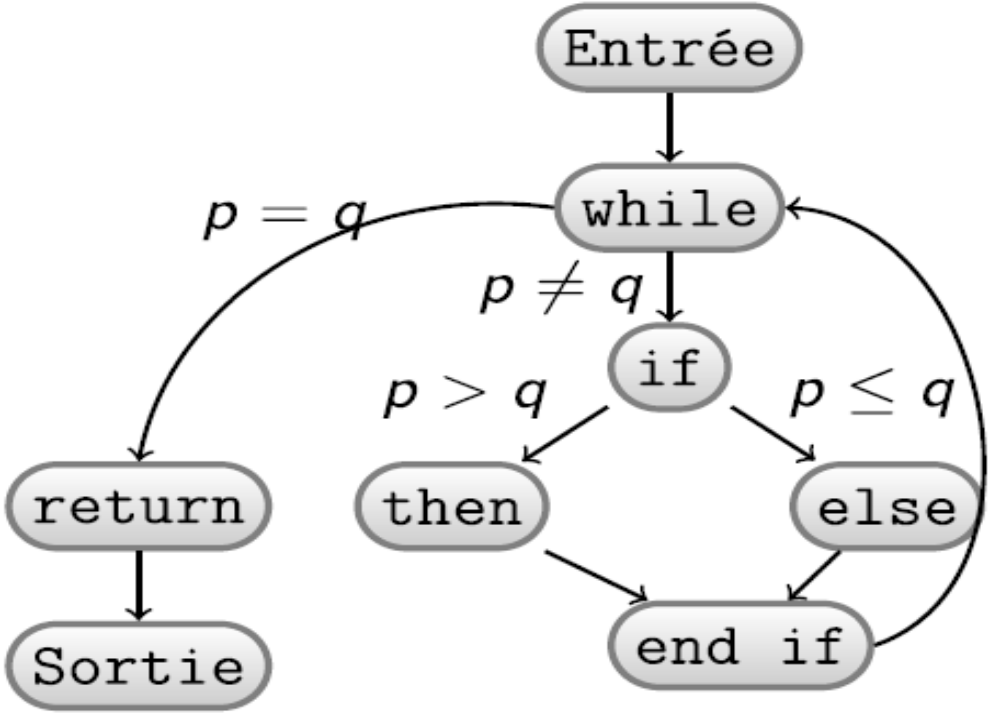


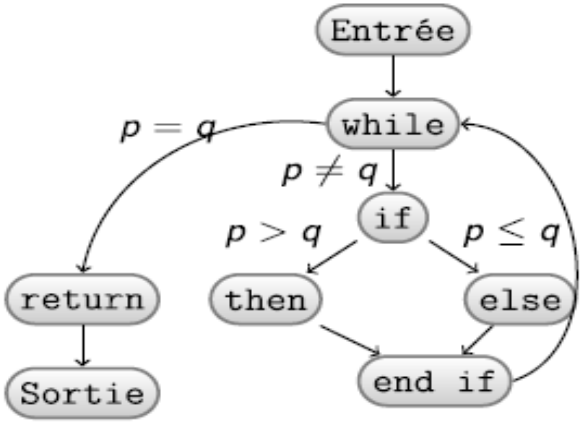
FIGURE – Graphe de contrôle

On cherche des jeux de tests à partir du graphe de contrôle afin de couvrir toutes les instructions, tous les chemins, . . .

Le cycle de vie d'un logiciel : Tests

Les méthodes de test : Structurelle/Fonctionnelle

❑ Test *structurelle*: Niveau de couverture



Tous les noeuds (C0) :

- (Entrée, while, if, then, return, Sortie)
- (Entrée, while, if, else, return, Sortie)

Tous les décisions (C1) :

- Chemins de (C0)
- (Entrée, while, return, Sortie)

Tous les 2-chemins (C-2) :

- Chemins de ($C1$)
- (E, while, if, then, while, if, then,return)
- (E, while, if, then, while, if, else,return)
- (E, while, if, else, while, if, then,return)
- (E, while, if, else, while, if, else,return)

Le cycle de vie d'un logiciel : Tests

Test manuel / test automatisé

Test manuel

- le testeur entre les données de test par ex via une interface;
- lance les tests;
- observe les résultats et les compare avec les résultats attendus;
- fastidieux, possibilité d'erreur humaine;
- ingérable pour les grosses applications;

Test automatisé

- Avec **le support d'outils qui déchargent le testeur** :
- du lancement des tests;
- de l'enregistrement des résultats;
- génération automatique de cas de test

Le cycle de vie d'un logiciel : Tests

Tests automatisés : Exemple d' Outils

JUnit

JUnit est un framework open source pour le développement et l'exécution de tests unitaires automatisables. Le principal intérêt est de s'assurer que le code répond toujours aux besoins même après d'éventuelles modifications. Plus généralement, ce type de tests est appelé tests unitaires de non-régression.

Le cycle de vie d'un logiciel : Tests

Les Tests : **une discipline à part entière.**

Très importants dans le cycle de développement

Les tests unitaires :

Malheureusement ne couvrent pas tous les cas,

Restent néanmoins considérés comme une garantie de qualité

Même si vous couvrez 99% des cas, le 1% restant peut cacher un bug majeur !

« ***Le test de programmes peut être une façon très efficace de montrer la présence de bugs mais est désespérément inadéquat pour prouver leur absence*** » - *Edsger Dijkstra*

Le cycle de vie d'un logiciel : Tests

Le plan de teste **:**

Le plan de test peut être **considéré comme un manuel (document)**. Il **décrit les objectifs des tests** (ce qu'il faut vérifier et/ou valider), **la portée des tests** (ce qui sera et ne sera pas testé), ainsi que **le calendrier général voire détaillé des activités à effectuer** (comment le tester et quand le tester) **la procédure des tests** et les **Contraintes (**environnement de test particulier, installation particulière, intervention humaine spécifique) .

Plan des teste peut être décomposé :

-**Plan de Tests unitaire.**
- **Plan de Tests d'intégration.**
- **Plan de Tests de validation.**

Exemple de plan de teste : « Voir document »

Le cycle de vie d'un logiciel : Tests

***Le plan de teste* :**

Le plan de test peut être **considéré comme un manuel (document)**. Il **décrit les objectifs des tests** (ce qu'il faut vérifier et/ou valider), **la portée des tests** (ce qui sera et ne sera pas testé), ainsi que **le calendrier général voire détaillé des activités à effectuer** (comment le tester et quand le tester) **la procédure des tests** et les **Contraintes (**environnement de test particulier, installation particulière, intervention humaine spécifique) .

Plan des teste peut être décomposé :

-Plan de Tests unitaire.
- Plan de Tests d'intégration.
- Plan de Tests de validation.

Exemple de plan de teste : « Voir document