Cours Génie Logiciel (GL 2)

3ème Année LMD

Les partons de conception (Design patterns)

Définition

Est une solution à un problème récurrent de conception objet. Les patterns simplifient la conception du logiciel et définissent un vocabulaire commun. Ils sont formulés de façon abstraite, indépendante du contexte ou de l'application.

En français, on dit aussi « motif de conception », voire « patron de conception ».

Est une structure qu'on réutilise à chaque fois qu'on en a besoin

Les partons de conception (Design patterns)

Description d'un patron de conception

Un patron est généralement donné par :

- Un nom: vocabulaire de conception
- La description du problème qu'il résout et son contexte;
- Les bénéfices qu'il apporte;
- La solution elle-même: description des éléments, de leurs relations/coopérations et de leurs rôles dans la résolution du problème;
- Les conséquences : Effets résultant de la mise en œuvre du patron, et éventuellement des remarques concernant l'implémentation.

Les partons de conception (Design patterns)

- **Les avantages**
  - ❑ Répondre à un problème de conception grâce à une solution éprouvée et validée par des experts.
  - ❑ La rapidité et la qualité de conception ce qui diminue également les coûts.
  - ❑ Mettre en avant les bonnes pratiques de conception grâce à la réutilisabilité.
  - ❑ facilite la communication.

Les partons de conception (Design patterns)

- **Les patrons de création :** ils permettent d'instancier et de configurer des classes et des objets.
  - Abstract factory, Factory method, Singleton,
  - Prototype, Builder
- **Les patrons comportementaux :** ils permettent d'organiser les objets pour qu'ils collaborent entre eux.
  - Iterator, Strategy, State, Observer, Command,
  - Visitor, Chain of responsiblity, Interpreter, Mediator, Memento, Template method

Les partons de conception (Design patterns)

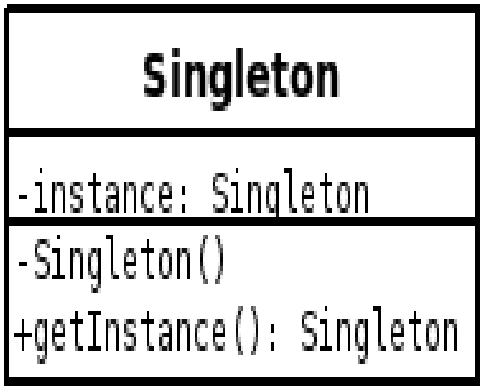
➢**Les patrons structuraux :** ils permettent d'organiser les classes d'une application

- ❑ Decorator, Adapter, Facade, Composite
- ❑ Bridge, Flyweight, Proxy

Les partons de conception (Design patterns)

- **Pattern Singleton**
  - ❑ Problème résolu : permet d'assurer qu'une seule instance d'une classe donnée est utilisée.

  - ❑ Solution : le constructeur de la classe est déclaré privé, afin d'empêcher la création d'autres instances, tandis qu'une instance est stockée dans la classe elle-même et accessible par une méthode. L'instance est créée de façon statique, ou au premier appel de getInstance().

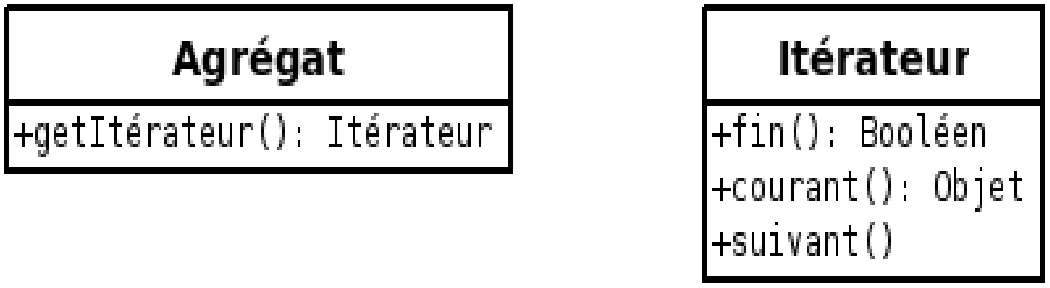



```
public static synchronized Singleton getInstance(){
if (_singleton == null)
_singleton = new Singleton();
return _singleton;
}
```

Les partons de conception (Design patterns)

- **Pattern Itérateur**
  - ❑ Problème résolu : permet de découpler l'itération sur les éléments d'une structure de la structure elle-même. Permet en particulier d'effectuer plusieurs itérations simultanément sur la même structure. Autrement dit fournir un accès séquentiel aux éléments d'un agrégat d'objets indépendamment de l'implémentation de l'agrégat (liste, tableau, ...)

  - ❑ Solution : la structure fournit des itérateurs sur elle-même.



Les partons de conception (Design patterns)

- **Pattern Itérateur**

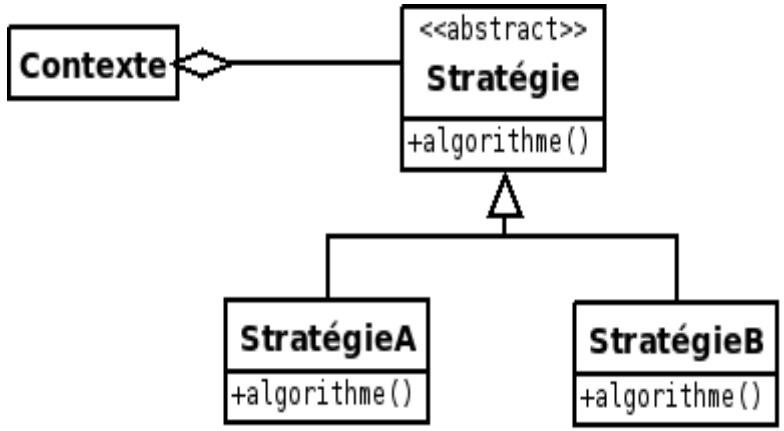
Exemple :

```
class Exemple{
private:
    vector<int> collection;
public:
    int somme(){
        int somme = 0;
        vector<int>::iterator it = collection.begin();
        while (it != collection.end()){
            somme += *it;
            it++;
        };
        return somme;
    }
};
```

Les partons de conception (Design patterns)

- **Pattern Stratégie**
  - ❑ Problème résolu : permet à un objet de modifier dynamiquement l'implémentation d'un comportement (par exemple, d'algorithme pour un problème) sans changer de classe.

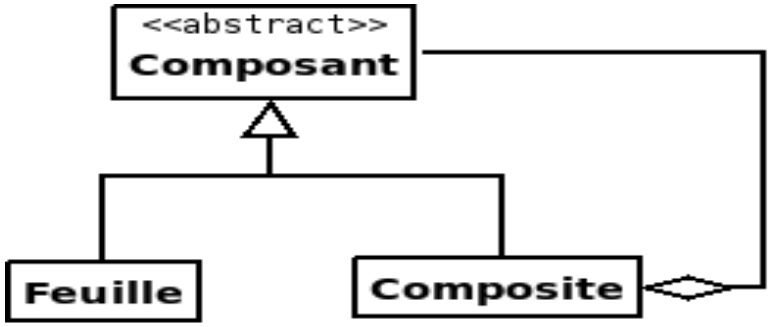
  - ❑ Solution : l'algorithme n'est pas implémenté dans la classe de l'objet. L'objet contient une stratégie, dont il peut changer dynamiquement l'instance concrète. Les différentes stratégies sont donc implémentées dans différentes classes concrètes.



Les partons de conception (Design patterns)

- **Pattern Composite**
  - ❑ Problème résolu : permet à une classe « cliente » de traiter de manière indifférente des objets ou des ensembles d'objets. Autrement dit représenter des hiérarchies composant/composé et traiter de façon uniforme les composants et les composés

  - ❑ Solution : une classe abstraite est dérivée en feuilles et en composites, qui contiennent eux-mêmes des composants (abstraits).

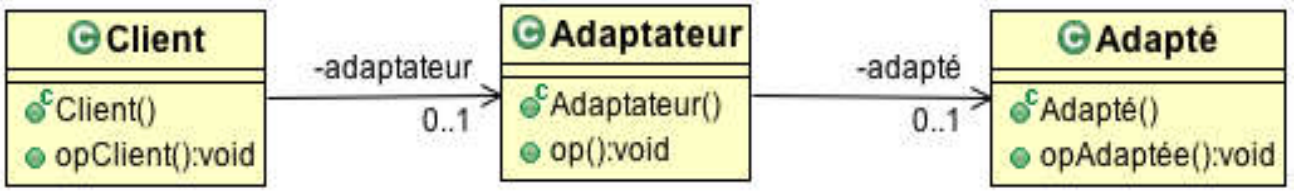


En général, les opérations pourront être implémentées dans le composite par de simples boucles sur les éléments qu'il contient.

Les partons de conception (Design patterns)

- **Pattern Adaptateur**
  - ❑ Problème résolu : permet d'« unifier » une interface requise par un client et l'interface d'une classe fournissant les services attendus. autrement dit fournir une interface stable(Adaptateur) à un composant dont l'interface peut varier (Adapté).

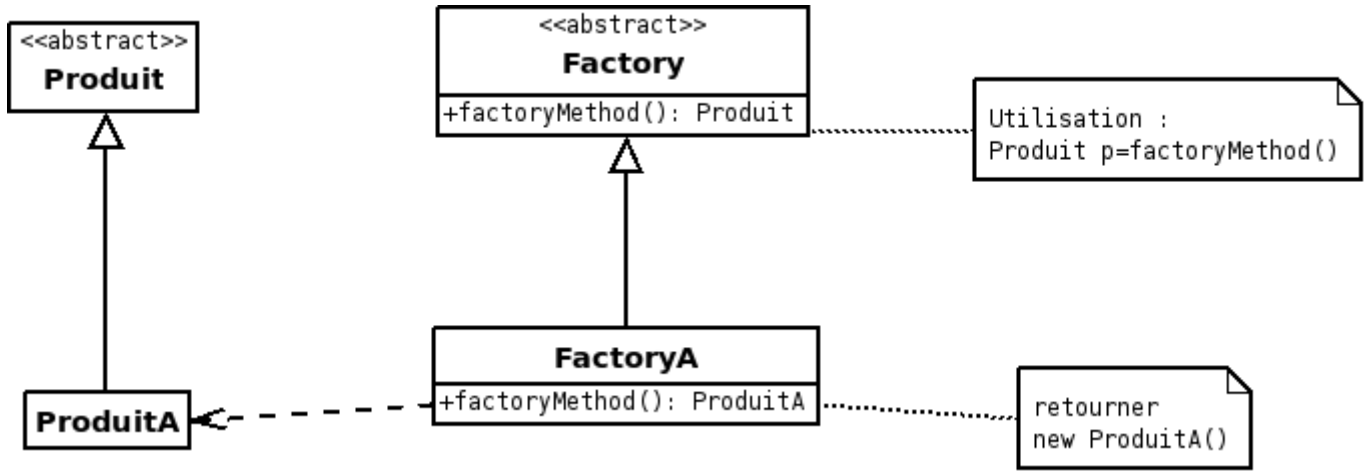
  - ❑ Solution : ce motif se décline en deux variantes. L'adaptateur de classe consiste à faire hériter une nouvelle classe de la classe fournissant les services, en lui faisant implémenter l'interface requise tout en utilisant les méthodes de la surclasse. L'adaptateur d'objet consiste à écrire une nouvelle classe, implémentant l'interface requise et utilisant une instance du fournisseur par délégation.



Les partons de conception (Design patterns)

- **Pattern Factory**
  - ❑ Problème résolu : permet à un client de créer des objets sans savoir leur type précis.

  - ❑ Solution : les objets à créer (produits) sont représentés par une classe abstraite dont dérivent toutes les classes d'objets. Une « usine » abstraite permet aux clients de fabriquer des produits. Chaque usine concrète crée un produit du type concret correspondant. En général, la méthode de création prend des arguments.



Les partons de conception (Design patterns)

- **Pattern Façade**
  - ❑ Problème résolu : permet à un client d'utiliser de façon simple et transparente un ensemble (complexe) de classes qui collaborent pour réaliser une tâche courante.

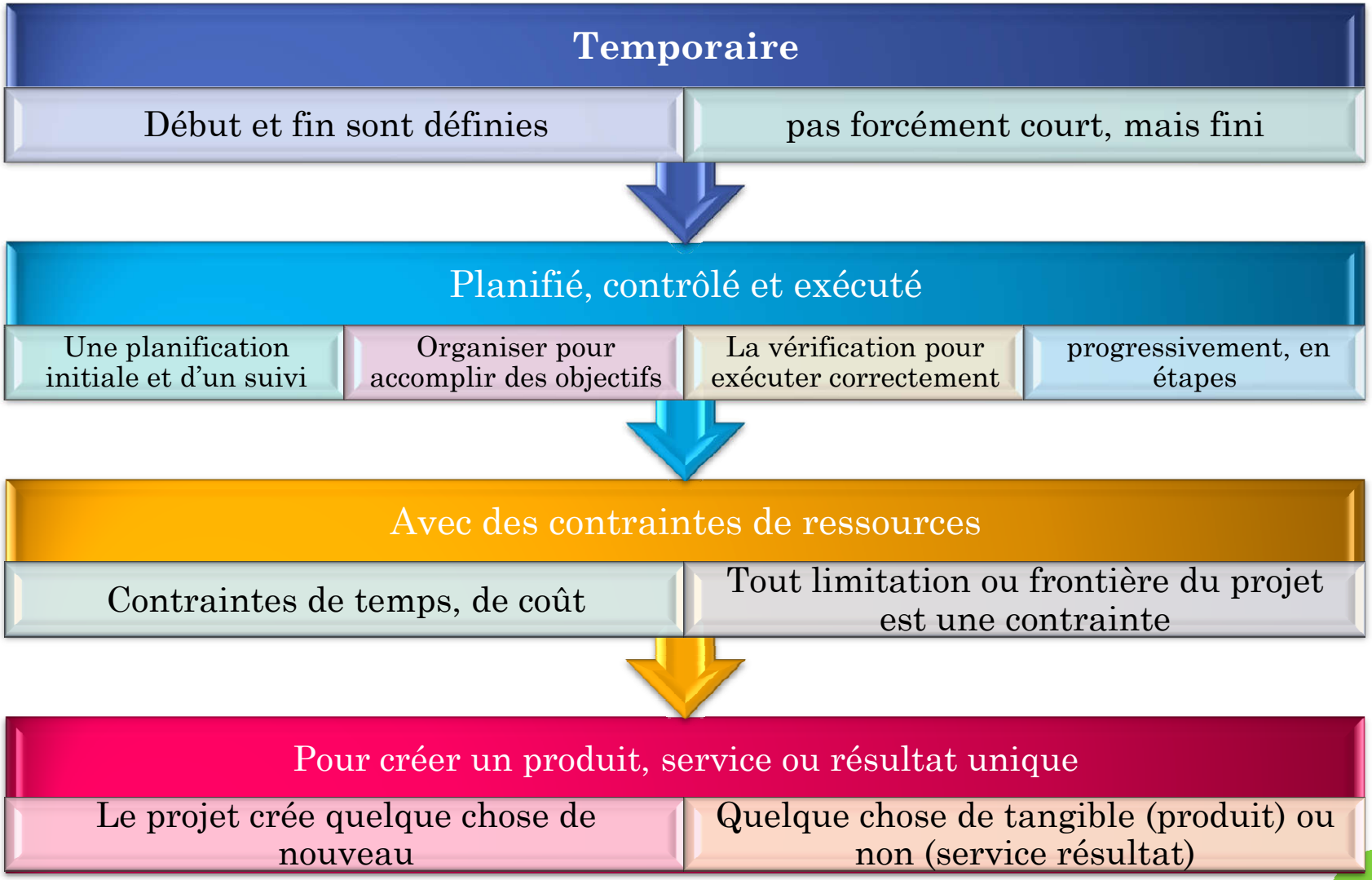
  - ❑ Solution : une classe fournit une façade de l'ensemble de classes, en proposant des méthodes qui réalisent les tâches usuelles en utilisant les autres classes.

Gestion de projets

Qu'est qu'un projet ?

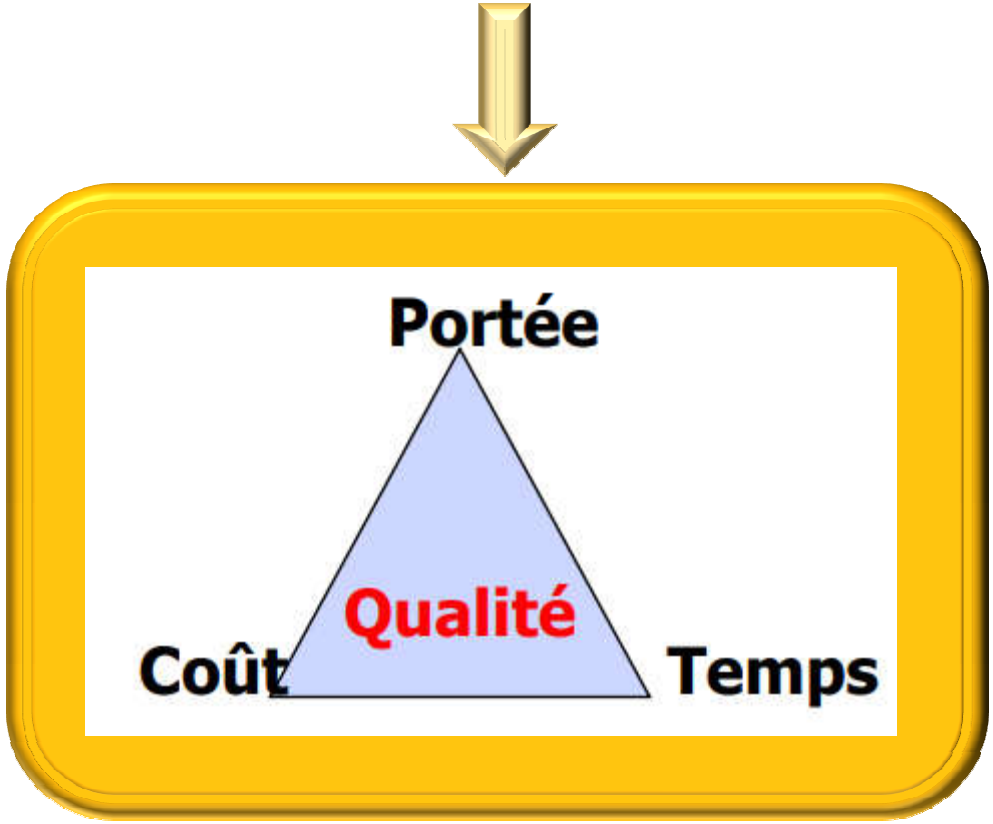
Un effort temporaire qui est progressivement planifié, contrôlé et exécuté par des personnes travaillant avec des contraintes de ressources pour créer un produit, service ou résultat unique.

Gestion de projets



Gestion de projets

Gérer un projet, c'est essentiellement gérer continuellement ces contraintes, pour atteindre des critères de qualité prédéfinis



Gestion de projets

➢ Naissance du projet

- ❑ Pourquoi démarre---t---on Un projet ?
  - ◆ Besoin, demande, idée, inspiration…
- ❑ Besoin organisationnel
  - ◆ Amélioration dans le processus métier ou création d'un nouveau
- ❑ Demande du marché
  - ◆ Opportunité Pour un Produit ou un service
- ❑ Demande d'un client
- ❑ Avance technologie (ou obsolescence)
- ❑ Nécessité légale
- ❑ Besoin social

Gestion de projets

- **Fin du projet**
  - Les objectifs sont atteints
  - Il devient clair qu'on ne pourra atteindre les objectifs
  - Le besoin n'existe plus

Gestion de projets

- **Problèmes souvent humains**
  - Planifier la progression
  - Motiver et coordonner un groupe de professionnels
- Techniques souvent communes à la gestion de projet en général
- Problème particulier de la visibilité
  - Un projet logiciel apparaitra souvent a ses développeurs

comme presque achevé alors qu'il ne l'est qu'a 90%

Gestion de projets

➢ Pratiques du chef de projet

- Opter pour une gestion des risques continue
  - Prévoir des étapes reparties sur l'ensemble du cycle de vie consacrées a l'identification et a l'évaluation des risques, ainsi que des tâches pour y remédier

- Estimer les coûts et planifier le projet a partir de données empiriques
  - Prévoir une étape au début du cycle de vie pour évaluer le coût du projet et une série d'étapes ultérieures pour raffiner cette estimation. Au cours de chaque étape, les données devront être archivées pour les évaluations ultérieures

Gestion de projets

➢Pratiques du chef de projet

- Utiliser des métriques pour la gestion du projet
  - Choisir des métriques et prévoir des étapes pour enregistrer les valeurs de celles-ci, et d'autres pour analyser les progrès en fonction de ces résultats

- Suivre l'évolution de la valeur acquise

- Rechercher les défauts en fonction des objectifs de qualité
  - Déterminer des objectifs pour le nombre de rapports d'erreurs et prévoir des étapes pour communiquer ces résultats

- Considérer les employés comme la ressource la plus importante
  - Contrôler l'ensemble du processus logiciel pour estimer son impact sur le programmeur

Gestion de projets

➢ Pratiques du chef de projet

- Utiliser un outil de gestion de configuration
  - Assurer que les modifications du logiciel sont effectuées de manière a minimiser les coûts globaux
  - Garder la trace des différences entre les versions pour contrôler les nouvelles versions
- Gérer et suivre l'évolution des besoins
  - Prévoir des étapes pour recueillir les besoins des utilisateurs
- Orienter la conception en fonction du système vise
- Définir et contrôler les interfaces

Gestion de projets

➢Pratiques du chef de projet

- Concevoir plusieurs fois pour ne coder qu'une seule
  - Prévoir des étapes pour contrôler la conception
- Identifier les éléments potentiellement réutilisables
- Contrôler les spécifications
- Organiser les tests comme un processus continu
  - Prévoir des étapes de tests dans toutes les phases

Gestion de projets

➢ Analyse de la valeur acquise

❑ Mesures de base

- CBT : coût budgété du travail
  - Quantité de travail estimée pour une tâche donnée
- CBTP : coût budgété du travail prévu
  - Somme des quantités de travail estimées pour l'ensemble des tâches devant être achevées à une date donnée
- CBA : coût budgété a l'achèvement
  - Total des CBTP et donc l'estimation de la quantité de travail pour le projet entier
- VP : valeur prévue
  - Proportion de la quantité de travail totale estimée attribuée à une tâche donnée
  - VP = CBT/CBA

Gestion de projets

➢ Analyse de la valeur acquise

❑ Mesures de base

- CBTE : coût budgété du travail effectué
  - Somme des quantités de travail estimées pour les tâches achevées à une date donnée

- CRTE : coût réel du travail effectué
  - Somme des quantités de travail réelles pour l'ensemble des tâches du projet

Gestion de projets

- **Analyse de la valeur acquise**

- **Indicateurs d'avancement**
  - VA : Valeur acquise
    - VA = CBTE/CBA
      = somme des VP pour les tâches achevées
      = PA (pourcentage achevé)
  - IPT : indicateur de performance temporel
    - IPT = CBTE/CBTP
  - VE : variance par rapport à l'échéancier
    - VE = CBTE-CBTP
  - IC : indicateur d'écart sur les coûts
    - IC = CBTE/CRTE
  - VC : variance par rapport aux coûts
    - VC = CBTE -CRTE

Gestion de projets

- **Suivi des erreurs**
  - Conserver une trace des :
    - Erreurs qui se sont produites
    - Durées entre deux erreurs successives
  - Permet de :
    - Mieux déterminer une date de livraison
    - Motiver les testeurs et les développeurs

Gestion de projets

- **Suivi des erreurs**
  - **Taux d'erreur**
    - TE : Inverse du temps qui sépare deux erreurs successives
      - Ex : si une erreur se produit tous les deux jours, TE = 0.5 erreurs par jour
    - Taux d'erreur instantané : estimation du taux d'erreur courant
    - Taux d'erreur cumule : bonne estimation des taux d'erreur à venir
      - Somme de toutes les erreurs divise par le temps total

Gestion de projets

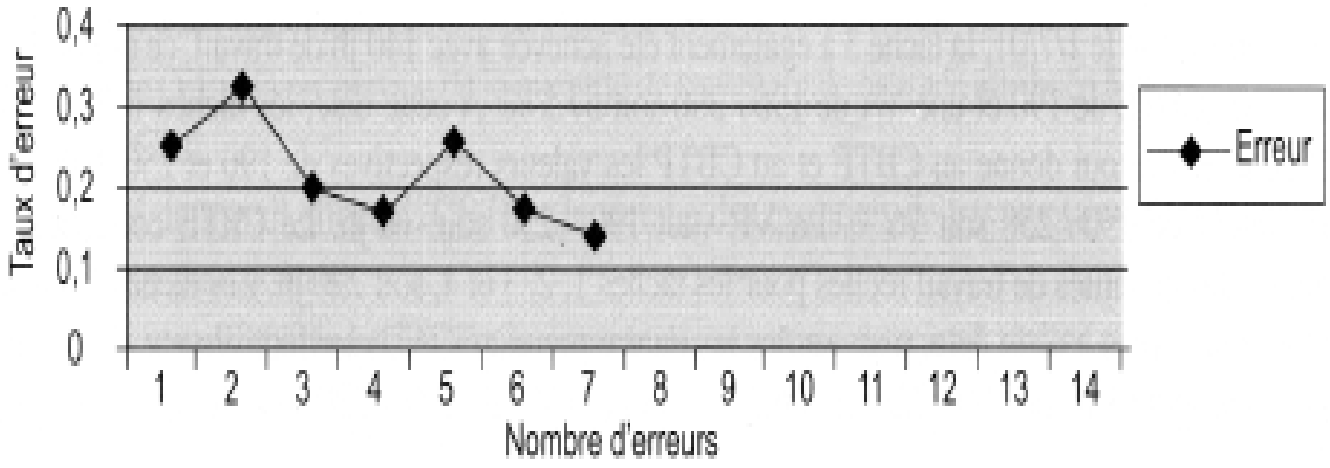
- **Suivi des erreurs**
  - **Taux d'erreur**
    - Une représentation graphique permet d'estimer les tendances des taux d'erreur par régression linéaire
      - y : taux d'erreurs
      - x : deux possibilités :
        - nombre d'erreurs : donne une estimation du nombre d'erreurs Restantes
        - temps écoule : donne une estimation du temps avant livraison

Gestion de projets

- **Suivi des erreurs**
  - **Exemple :** les durées entre une série d'erreurs sont les suivantes : 4, 3, 5, 6, 4, 6, 7.
    - Calcule de taux d' erreurs

Durée	4	3	5	6	4	6	7
Taux	0.25	0.33	0.20	0.17	0.25	0.17	0.14

    - Représentation graphique



Planification de projets

- **Planification**
  - Elément indispensable de la conduite de projets
    - Déterminer les tâches a accomplir
    - Déterminer l'ordre des tâches
    - Décider des ressources allouées a chaque tâche

Planification de projets

- **Suivi de la planification**
  - Réaliser des réunions d'avancement du projet de façon périodique;
  - Évaluer les résultats de toutes les revues;
  - Déterminer si les jalons du projet ont été atteints;
  - Comparer les dates de fin réelles et prévues;
  - Discuter avec les gens.

Planification de projets

➢ Organigramme technique

- ❑ Objectif : Diviser les tâches principales en tâches plus petites
- ❑ Nécessite de :
  - ◆ Pouvoir identifier leurs différentes parties
  - ◆ Trouver des livrables et des jalons qui permettront de mesurer l'avancement du projet
- ❑ WBS Work Breakdown Structure : organigramme technique
  - ◆ Fondement essentiel de tout projet, permet de découper l'ensemble du travail à accomplir en sections gérables.
  - ◆ Structure arborescente
  - ◆ Le premier niveau de décomposition correspond souvent au modèle de cycle de vie adopte

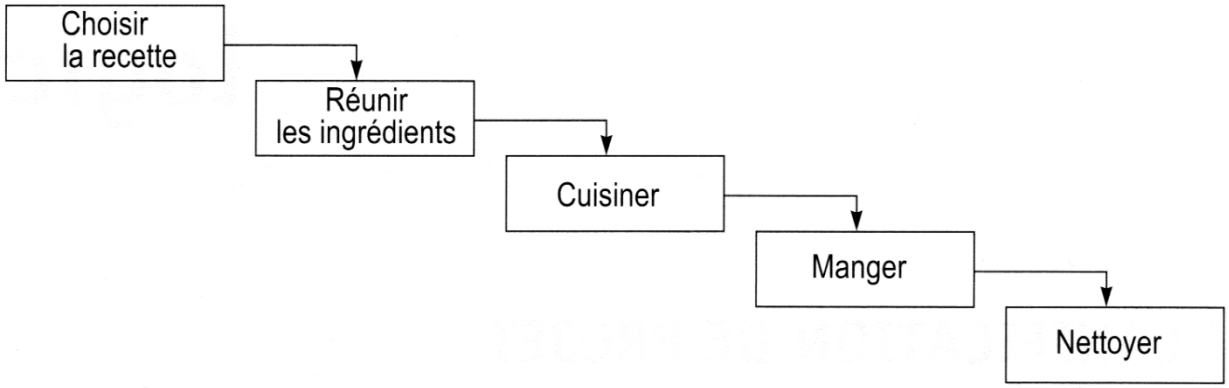
Planification de projets

- **Règles d'obtention d'un organigramme technique (WBS)**
  - Structure arborescente
    - Pas de boucle
    - Les actions itératives apparaissent dans le modèle de processus ou dans le modèle de cycle de vie
  - Les descriptions des tâches et des livrables doivent être claires et sans ambigüité
    - Chaque livrable doit être associe à une tâche
  - Chaque tâche doit avoir un critère d‘ achèvement
    - Le plus souvent un livrable
  - L‘achèvement de toutes les sous-tâches doit entrainer l‘achèvement de la tâche

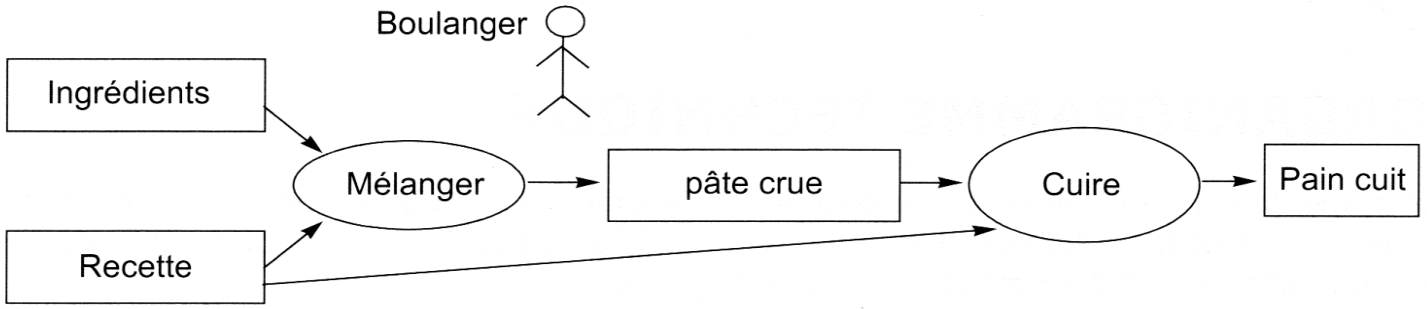
Planification de projets

➢ Exemple

- Modèle de cycle de vie pour la consommation de tartines

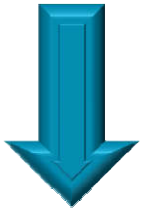


- Modèle de processus pour cuisiner du pain



Planification de projets

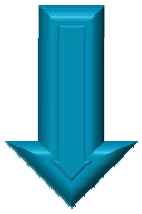
- **Exemple**
  - Choisir la recette
  - Réunir les ingrédients → Ingrédients réunis
  - Cuisiner
  - Manger
  - Nettoyer



  - Choisir la recette
    - Choisir les ingrédients → Liste des ingrédients
    - Vérifier leur disponibilité → Liste de courses
  - Réunir les ingrédients Ingrédients réunis
  - Cuisiner
  - Manger
  - Nettoyer

Planification de projets

- Choisir la recette
- Réunir les ingrédients ⟶ Ingrédients réunis
- Cuisiner
  - Mélanger
  - Cuire ⟶ Pain cuit
- Manger
- Nettoyer



Planification de projets

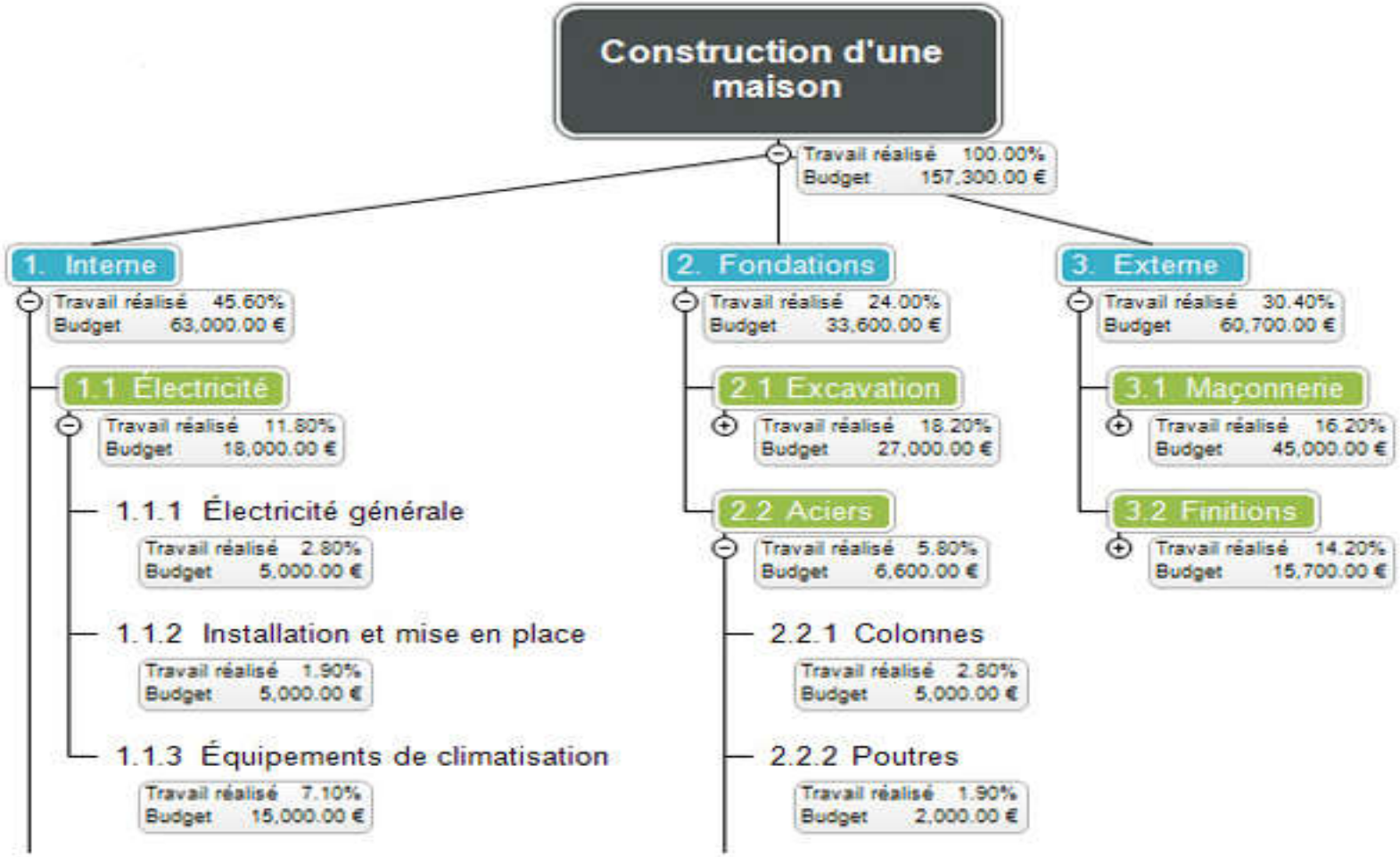
- Choisir la recette
- Réunir les ingrédients ! Ingrédients réunis
- Cuisiner
  - Mélanger
    - Ajouter l'eau ! Saladier d'eau
    - Ajouter la levure et la farine ! Mélange
    - Faire la pâte → Pâte
    - Laisser lever 1 → Pâte levée
    - Ajouter le reste de farine et pétrir → Pâte pétrie
    - Laisser lever 2 → Pâte pétrie et levée
    - Former des miches → Miches
    - Laisser lever 3 → Miches levées
  - Cuire ! Pain cuit
- Manger
- Nettoyer

Planification de projets

- Choisir la recette
- Réunir les ingrédients ⟶ Ingrédients réunis
- Cuisiner
- Manger
  - Découper en tranches ⟶ Tranches de pain
  - Beurrer ⟶ Tartines beurrées
  - Manger ⟶ Goût satisfaisant
- Nettoyer
  - Nettoyer ustensiles ⟶ Ustensiles propres
  - Nettoyer cuisine ⟶ Cuisine propre

Planification de projets

➢ **Exemple de WBS pour la construction d'une maison**



Planification de projets

➢ **Caractéristiques de la WBS**

❑ Elle permet au chef de projet d'établir le graphe PERT et de faire un suivi budgétaire

- Doit être complète, pour élaborer un graphe PERT correct
- Doit être non ambiguë, pour budgéter correctement le projet et contrôler les coûts par la suite
- Les résultats des activités doivent être mesurables pour évaluer l'avancement général

❑ Certaines activités sont toujours présentes :

- Elaboration des documents, inspections...
- Construction d'outils, apprentissage...

La semaine prochaine

inchaa lah

La méthode PERT

❑ Program Evaluation and Review Technique (Technique d'Estimation & Révision de Programme)

❑ Est une technique de planification permettant de gérer l'ordonnancement dans un projet.

❑ Permet de représenter sous forme d'un graphe de dépendances, un réseau de tâches dont l'enchaînement permet d'aboutir à l'atteinte des objectifs d'un projet et cela en :

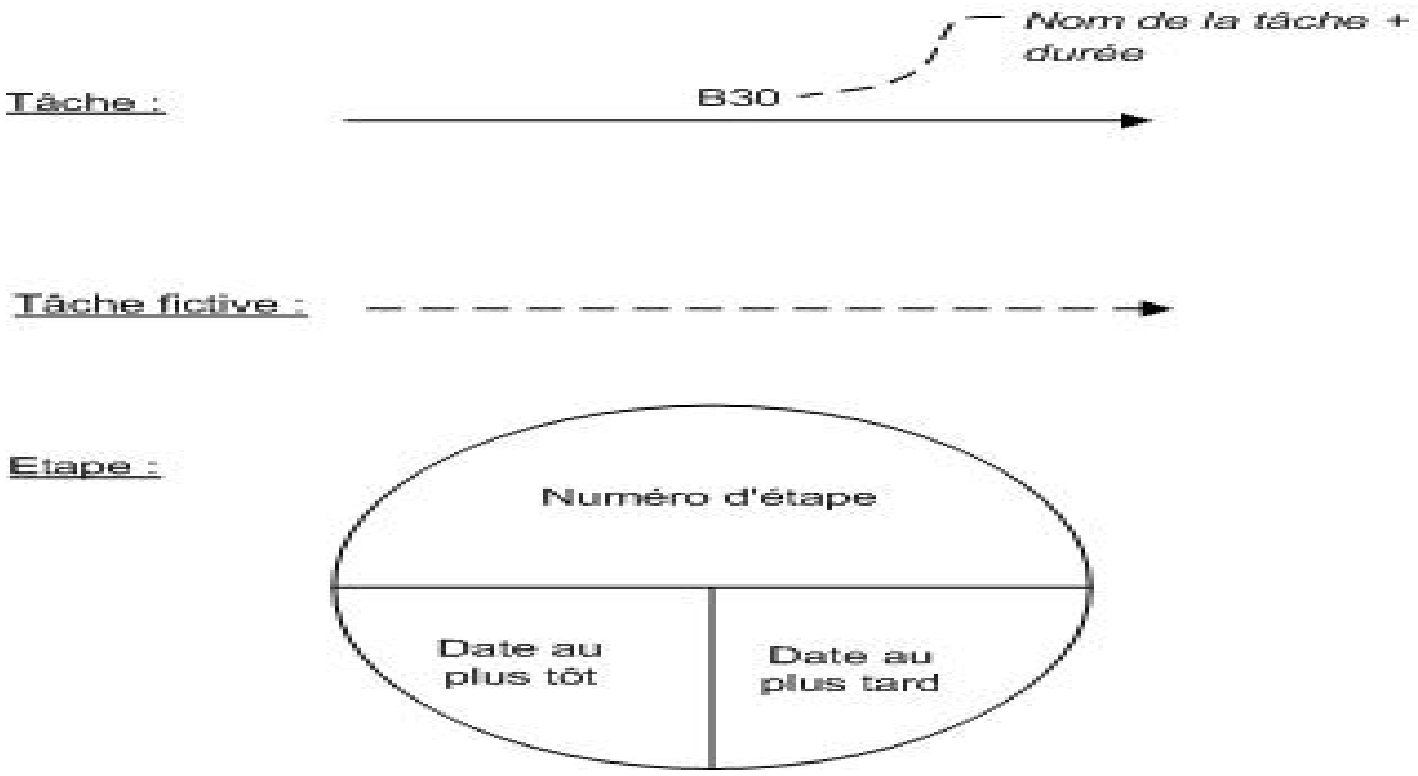
- Identifiant les tâches et estimer leur durée;
- Ordonnant les tâches;
- Construisant le réseau et l'exploiter.

Les éléments de la méthode PERT

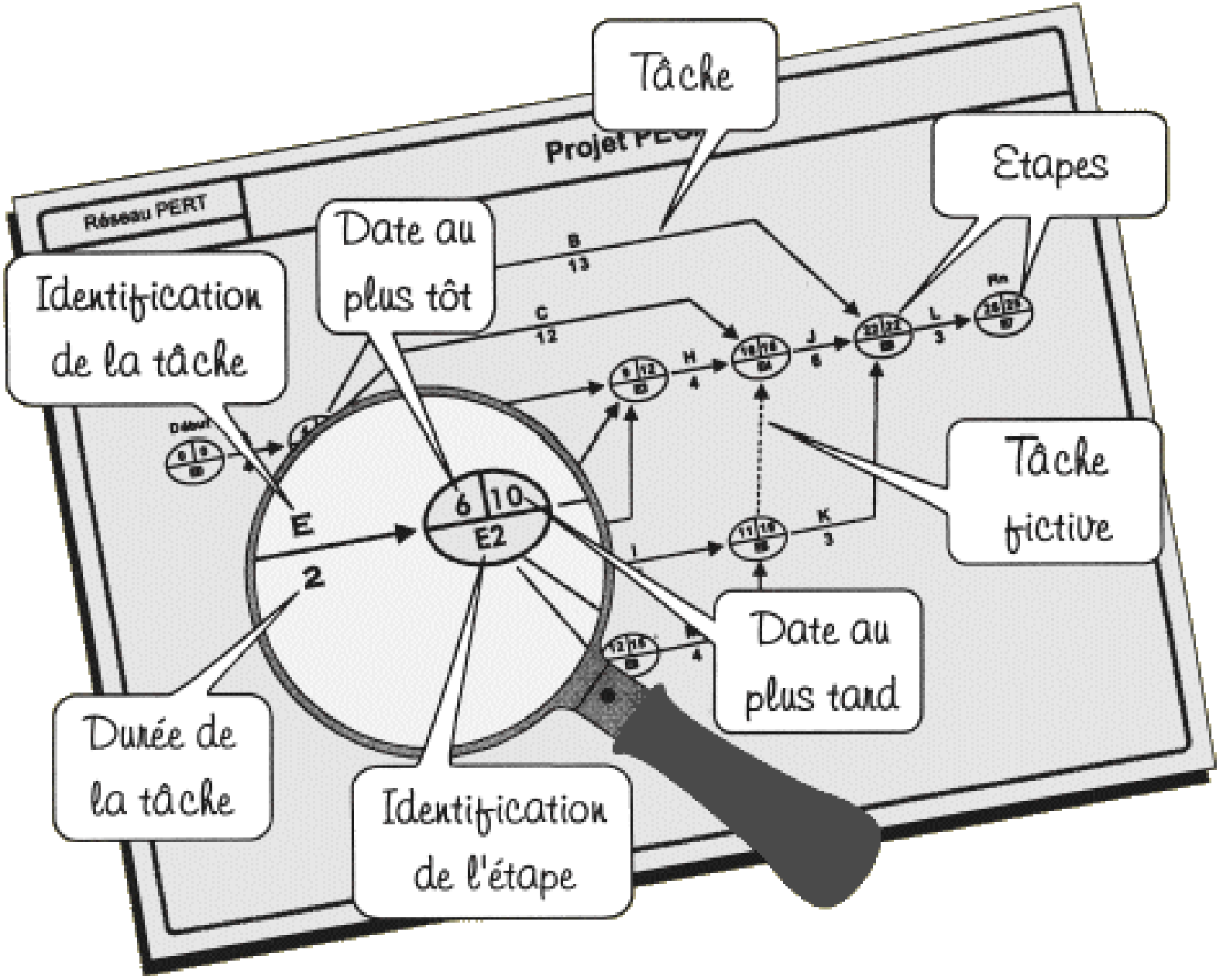
- **Tâche** (parfois activité ou étape), représentée par une flèche. A chaque tâche correspond un code (identifiant) et une durée.

- **Étape**, c'est-à-dire le début et la fin d'une tâche. Chaque tâche possède une étape de début et une étape de fin. A l'exception des étapes initiales et finales, chaque étape de fin est étape de début de la tâche suivante. Les étapes sont en règle générale numérotées et représentées par un cercle, mais elles peuvent parfois avoir d'autres formes (carré, rectangle).
- **Tâche fictive**, représentée par une flèche en pointillés, permet d'indiquer les contraintes d'enchaînement entre certaines étapes.

Les éléments de la méthode PERT



Planification de projets



Exploitation d'un réseau PERT

❑ Calcul des dates au plus tôt

- **La date au plus tôt** : il s'agit de la date à laquelle la tâche pourra être terminée au plus tôt, en tenant compte du temps nécessaire à l'exécution des tâches précédentes. Se calculent de gauche à droite en partant de 0 et rajoutant la durée de la tâche à la date précédente. En cas de convergence, on prend la valeur la plus élevée.
- DTO : date de début au plus tôt
- FTO : date de fin au plus tôt
- **FTO = DTO + durée**

Exploitation d'un réseau PERT

❑ Calcul des dates au plus tard

- **La date au plus tard** : il s'agit de la date à laquelle une tâche doit être terminée à tout prix si l'on ne veut pas retarder l'ensemble du projet. Elles se calculent de droite à gauche en partant de la date de fin au plus tard et en retranchant la durée de la tâche à la date précédente. En cas de convergence on prend la valeur la plus faible.
- DTA : date de début au plus tard
- FTA : date de fin au plus tard
- DTA = FTA - durée

Exploitation d'un réseau PERT

❑ Calcul des marges et du chemin critique

- **Les marges :** la différence entre la date au plus tard et la date au plus tôt d'une tâche s'appelle la marge totale.

- Il peut y avoir que certaines étapes ont une marge nulle et que pour d'autres on est plus libre.

- La marge libre d'une tâche T est le délai de retard maximum que l'on peut apporter à la mise en route de cette tâche, sans pour autant que les tâches suivantes en soient affectées. Elle est égale à la différence entre :
  - La plus petite date au plus tôt des tâches suivantes ;
  - La date au plus tôt de la tâche T, à laquelle on rajoute sa durée.

Exploitation d'un réseau PERT

- Calcul des marges et du chemin critique
  - Exemple

Etape	Date au plus tôt	Date au plus tard	Marge
1	0	0	0
2	300	483	183
3	3	483	480
4	300	483	183
5	3	3	0
6	33	33	0
7	330	513	183
8	633	633	0
9	450	633	183
10	643	643	0
11	653	653	0
12	773	773	0

Exploitation d'un réseau PERT

❑ Calcul des marges et du chemin critique

- Le chemin critique : l'ensemble des activités qui s'enchainent en continu et dont la somme des durées est égale à la durée totale.

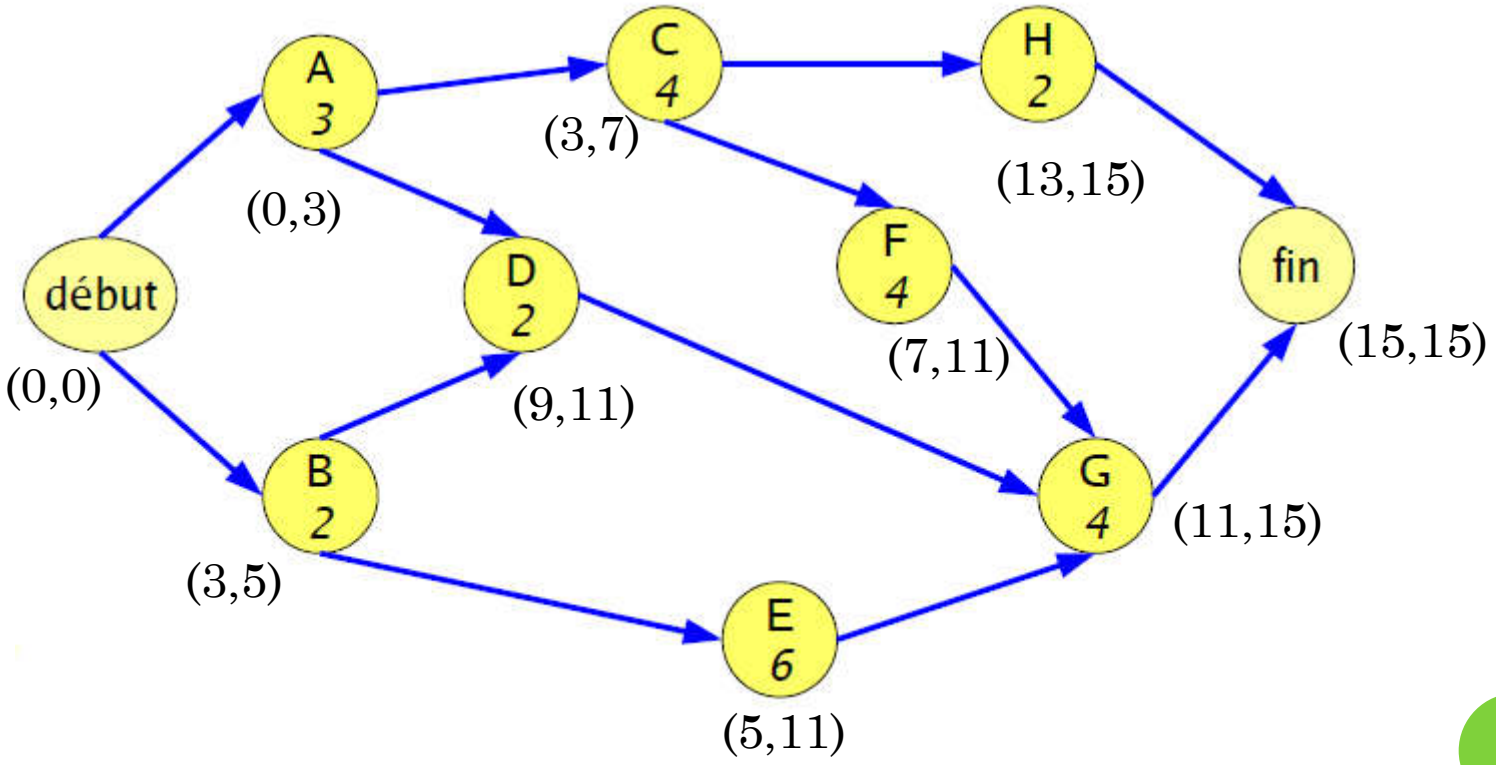
- Particulièrement surveillé par le chef de projet puisqu' 'un retard sur ce chemin entraine de faire un retard sur le projet. Une réduction de la durée du projet passe forcément par la réduction de la durée du chemin critique du projet . Les activités du chemin critique n'ont ni marge totale, ni marge libre autrement dit le chemin critique relie les étapes qui n'ont aucune marge.

- Détermine la durée minimale du projet,

Planification de projets

Exploitation d'un réseau PERT

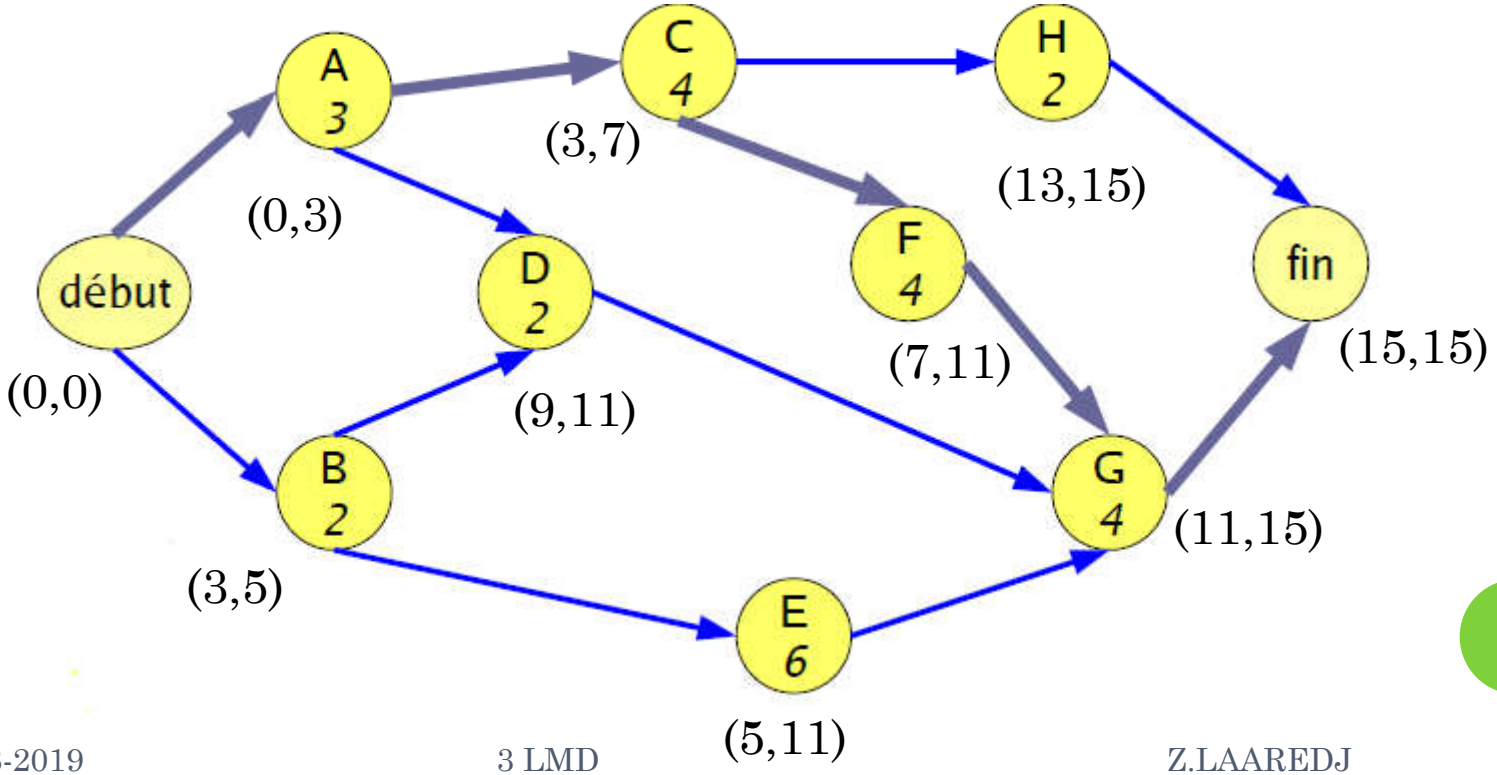
- **Calcul des marges et du chemin critique**
  - **Exemple:** quel est le chemin critique et la durée minimale ?



Planification de projets

Exploitation d'un réseau PERT

- **Calcul des marges et du chemin critique**
  - **Exemple:** quel est le chemin critique et la durée minimale ?
  - Durée minimale = 15, le chemin critique est Début, A,C,F, G,Fin.



Planification de projets

➢ Extensions des réseaux PERT

- ❑ **PERT Charge :** pour prendre en compte les ressources affectées au projet
  - Ressource : moyen nécessaire au déroulement et à l'aboutissement d'une tâche;
  - Les tâches sont caractérisées par des durées et des intensités de ressources
- ❑ **PERT Cost :** pour gérer les coûts
  - Permet d'optimiser l' échéancier des paiements en
    - Jouant sur les surcoûts affectant les tâches critiques
    - Jouant sur les économies possibles sur les tâches non critiques

Planification de projets

- **Le diagramme Gantt**
  - Utilise les dates au plus tôt et des dates au plus tard
  - Permet d' établir un planning en fonction des ressources
  - Son but est de faire apparaître
    - la répartition des activités dans le temps;
    - l'affectation des individus.
  - Il donne une description détaillée
    - des coûts (en hommes*mois);
    - des dates pour chaque tâche et pour chaque phase.
  - A chaque tâche sont attribués
    - un objectif pour repérer la terminaison de l'activité;
    - une durée pour atteindre cet objectif;
    - des ressources nécessaires à son accomplissement

Planification de projets

➢ Le diagramme Gantt

❑ Il faut d'abord estimer les durées et les ressources;

❑ Pour harmoniser le diagramme de Gantt, il faut utiliser la même unité de temps;

❑ Les ressources peuvent être humaines ou matérielles;

❑ Après avoir ordonnées les tâches à l'aide d'un PERT :

- en abscisse, l'échelle des temps
- en ordonnée, la liste des tâches
- des rectangles sont tracés proportionnellement à la durée de la tâche, avec l 'affectation des ressources nécessaires.

Planification de projets

➢ Estimation des coûts

❑ Le modèle COCOMO de base

- COnstructive COst Model;
- Développé à l'entreprise TRW (organisme du DoD, USA) par Boehm et son équipe en 1981;
- Fondé sur une base de données de plus de 60 projets différents.

❑ Modèle d'estimation

- du coût de développement d'un logiciel en nombre de mois-hommes (E : effort)
- du temps de développement en mois (TDEV) en fonction du nombre de lignes de codes (instructions) en milliers (KLOC, KDSI)

Planification de projets

➢ Estimation des coûts

❑ Le modèle COCOMO de base

- 3 types de projet sont identifiés :
  - Organique : innovation minimale, organisation simple et petites équipes expérimentées (ex : petite gestion…)
  - Médian (semi-detached) : degré d ’innovation raisonnable (ex: banque, compilateurs…)
  - Imbriqué (embedded) : innovation importante, organisation complexe, couplage fort et nombreuses interactions (ex : gros systèmes, avioniques...)

Planification de projets

- **Estimation des coûts**
  - **Le modèle COCOMO de base**

    - **Calcul de l'effort**
      - Formule générale : $E = a \times KLOC^b$
      - a et b estimes en fonctions de données empiriques

    - **Calcul du temps de développement**
      - Formule générale : $TDEV = a \times KLOC^b$
      - a et b estimes en fonctions de données empiriques

Planification de projets

- **Estimation des coûts**
  - **Le modèle COCOMO de base**

Projet	MM	TDEV
Organique	$E = 2.4 * KLOC^{1.05}$	$TDEV = 2.5 * KLOC^{0.38}$
Médian	$E = 3.0 * KLOC^{1.12}$	$TDEV = 2.5 * KLOC^{0.35}$
Imbriqué	$E = 3.6 * KLOC^{1.20}$	$TDEV = 2.5 * KLOC^{0.32}$

Planification de projets

- **Estimation des coûts**
  - **Le modèle COCOMO intermédiaire**
    - Estimation modifiant l'estimation brute fournie par le modèle COCOMO de base en se servant des attributs logiciel, matériel, projet, personnel.

    - Les attributs du logiciel
      - Besoin en fiabilité
      - Taille de la Base de Données
      - Complexité du produit

    - Les attributs du matériel
      - Contraintes sur le temps l'exécution
      - Contraintes sur la mémoire
      - Contraintes sur le stockage
      - Contraintes du temps de passage entre deux processus (synchronisation)

Planification de projets

- **Estimation des coûts**
  - **Le modèle COCOMO intermédiaire**
    - **Les attributs du projet**
      - Techniques de programmation moderne
        - Programmation Orientée Objet
        - Programmation Evénementielle
      - Utilisation l'Ateliers de Génie Logiciel
      - Contraintes de développement
        - Délais
        - Budget

Planification de projets

- **Estimation des coûts**
  - **Le modèle COCOMO intermédiaire**
    - **Les attributs du personnel**
      - Compétence de l'analyste
      - Compétence du programmeur
      - Expérience dans l'utilisation du langage de programmation
      - Expérience dans le domaine de l'application
      - Expérience dans l'utilisation du matériel

Planification de projets

- **Estimation des coûts**
  - **Le modèle COCOMO intermédiaire**

Projet	MM
Organique	$E = 3.2 * KLOC^{1.05}$
Médian	$E = 3.0 * KLOC^{1.12}$
Imbriqué	$E = 2.8 * KLOC^{1.20}$

Les estimations obtenues par la formule sont multipliées par les 15 facteurs de coût liées aux attributs du logiciel, du matériel, du projet et du personnel

Planification de projets

- **Estimation des coûts**
  - **Le modèle COCOMO expert**
    - Inclue toutes les caractéristiques du modèle intermédiaire
    - Ajouts :
      - L'impact de la conduite des coûts sur chaque étape du cycle de Développement
      - Le projet est analyse comme une hiérarchie : module, sous-système et système
    - COCOMO expert permet une véritable gestion de projet
      - Utile pour de grands projets
      - Ajustement des paramètres possibles en fonction de données locales portant sur les habitudes de développement
      - Problème : nécessite une estimation de la taille du projet en KLOC

Planification de projets

➢ **Estimation des coûts**

❑ **Analyse en points de fonction**

- Plutôt que d'estimer le nombre de lignes de code, il peut être plus judicieux d'estimer des points de fonction
- Les éléments les plus courants à prendre en compte sont les :
  - Interrogations : paires requête-réponse
  - Entrées : les champs individuels ne sont généralement pas comptes séparément (nom, prénom...)
  - Sorties (comme les entrées)
  - Fichiers internes : fichiers tels que le client les comprend
  - Interfaces externes : données partagées avec d'autres programmes

Assurance qualité

- **Définition de « Qualité »**
  - ISO 8402 : « l'ensemble des caractéristiques d'une entité qui lui confèrent l'aptitude à satisfaire des besoins exprimes et implicites ».
  - Un logiciel est de qualité lorsqu'il fonctionne comme il est suppose le faire.
  - Il est plus facile de mesurer les défauts de qualité
    - Mécontentement du client;
    - Nombre de rapports d'erreurs.

Assurance qualité

- **Inspections formelles**
  - Activité formelle et planifiée :
    - Un concepteur présente des documents sur un projet à un groupe d'autres concepteurs qui en évaluent les aspects techniques avec pour objectif de trouver les erreurs;
    - Contrôle effectué par des personnes techniquement compétentes;
    - Participation active de l'auteur;
    - Porte sur un produit fini;
    - Inspection périodique au cours du processus de développement.

Assurance qualité

- **Rôles pour une inspection**
  - **Le modérateur :** c'est lui qui choisi l' équipe ainsi il dirige l'inspection.
  - **Le lecteur :** Il n'est généralement pas l'auteur du produit, Il guide l'équipe dans la structure du produit.
  - **Le secrétaire :** Il consigne le déroulement de l'inspection et note toutes les erreurs trouvées.
  - **L'auteur :** Il est a l'origine du produit examiné permet de répondre aux questions et corriger les erreurs et fait un rapport au modérateur.

Assurance qualité

- **Etapes de l'inspection**
  - **Présentation générale :** fait par l'auteur au reste de l‘ équipe
  - **Préparation :** les membres de l‘ équipe étudient le produit dans la limite d'un temps calcule en fonction du nombre de LOC (Lines Of Code), Ils peuvent s'aider d'une liste de contrôles.
  - **Réunion pour l'inspection :** organisée par le modérateur, dans laquelle le lecteur conduit l'inspection et le secrétaire consigne les problèmes dans un rapport. En cas de désaccord, il est possible de produire des rapport individuels.
  - **Intégration des remarques :** l'auteur corrige les erreurs.
  - **Suivi :** le modérateur contrôle le rapport et les corrections et si les critères sont satisfaits, l'inspection prend fin.

Technique de génie logiciel

➢ Métriques

❑ **Définition :** représentent les différentes mesures qui sont primordiales en génie logiciel. Utilisé pour mesurer un logiciel ainsi le processus de développement de ce logiciel. Les métriques logicielles sont utilisé afin d'améliorer le processus de développement d'un projet.

❑ Les métriques peuvent être classées en trois catégories :

- Celles mesurant le processus de développement
- Celles mesurant des ressources
- Et celles de l'évaluation du produit logiciel

❑ Les métriques de produits mesurent les qualités du logiciel. Parmi ces métriques, on distingue :

- Les métriques traditionnelles
- Les métriques orientées objet

Technique de génie logiciel

➢ Métriques

❑ **Les métriques orientées objet :** prennent en considération les relations entre éléments de programme (classes, méthodes).

❑ **Les métriques traditionnelles :** se divisent en deux groupes :

- Les métriques mesurant la taille et la complexité les plus connus sont les métriques de ligne de code ainsi que les métriques de Halstead
- Les métriques mesurant la structure du logiciel. comme la complexité cyclomatique de McCabe se basent sur des organigrammes de traitement ou des structures de classe.

Technique de génie logiciel

➢ Métriques

❑ Les métriques des lignes de code pour quantifier la complexité d'un logiciel, les mesures les plus utilisées sont les lignes de code (LOC « lines of code ») puisqu'elles sont simples, faciles à comprendrent à compter.

❑ Les types de métriques de lignes de code suivants :

- LOCphy: nombre de lignes physiques (total des lignes des fichiers source) ;
- LOCpro: nombre de lignes de programme (déclarations, définitions, directives, et code;
- LOCcom: nombre de lignes de commentaire ;
- LOCbl: nombre de lignes vides.

➢ Métriques

❑ Métriques de Mac Cabe

- Complexité structurelle selon Mc Cabe
- Métrique la plus utilisée après les lignes de code
  - Met en évidence la complexité structurelle du code
  - Produit un graphe de contrôle qui représente un code
  - Le nombre de faces du graphe donne la complexité structurelle du code

Technique de génie logiciel

➢ Métriques de Mac Cabe

❑ Nombre cyclomatique de Mc Cabe

$$C = a - n + 2p$$

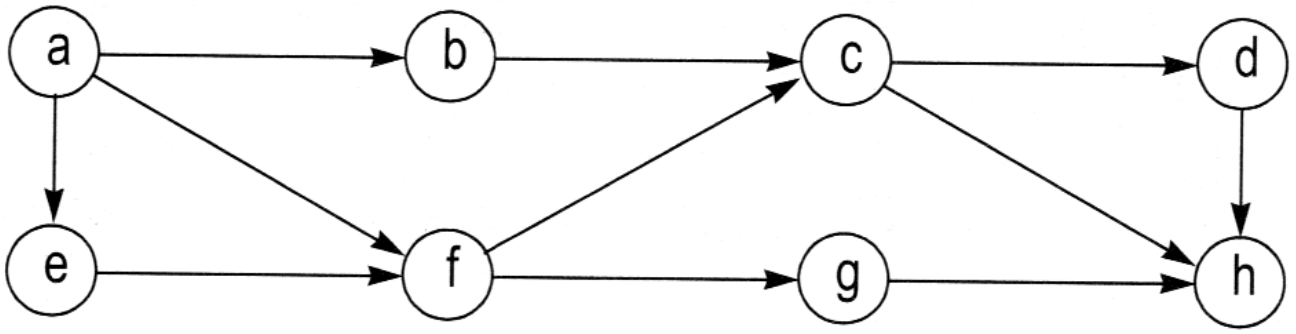
avec

a = nombre d'arcs du graphe de contrôle

n = nombre de nœuds du graphe de contrôle

p = nombre de composantes connexes (1 le plus souvent)

❑ Exemple

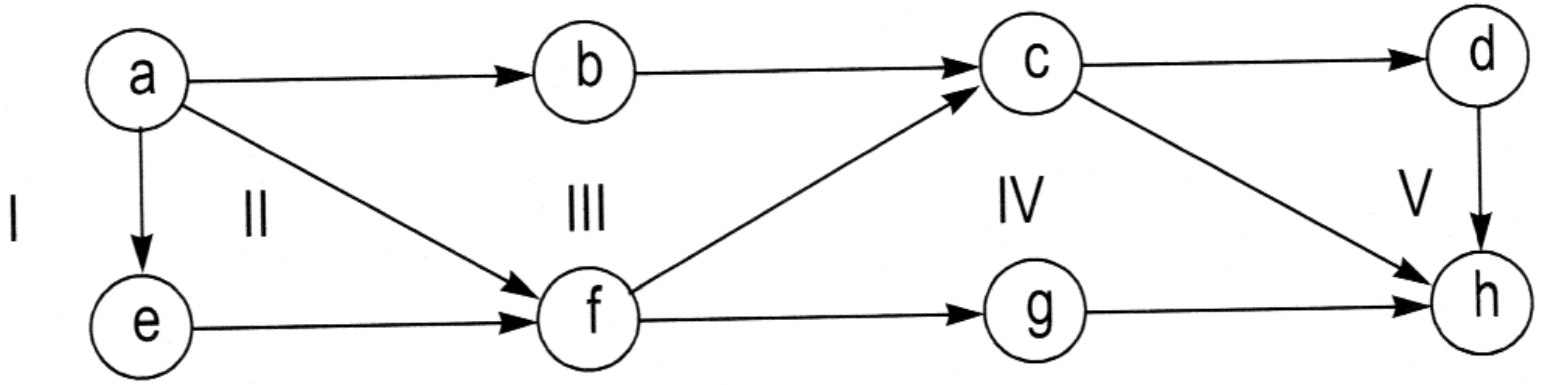


n = 8, a = 11 et p = 1 donc

C = 11-8 + 2 = 5

Technique de génie logiciel

- **Métriques de Mac Cabe**
  - **Nombre cyclomatique de Mc Cabe**



C = 11-8 + 2 = 5

C correspond au nombre de faces (en comptant la face extérieure)

Technique de génie logiciel

➢ Métriques de Mac Cabe

❑ Calcul direct du nombre de Mc Cabe

- Produire un graphe de contrôle et l'analyser peut s‘ avérer long dans le cas de programmes complexes
- Mc Cabe à introduit une nouvelle manière de calculer la complexité structurelle

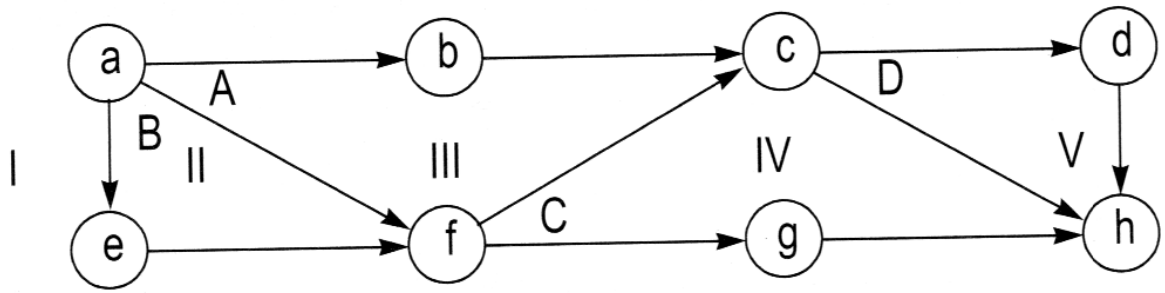
$$C = \pi + 1$$

π le nombre de décisions du code

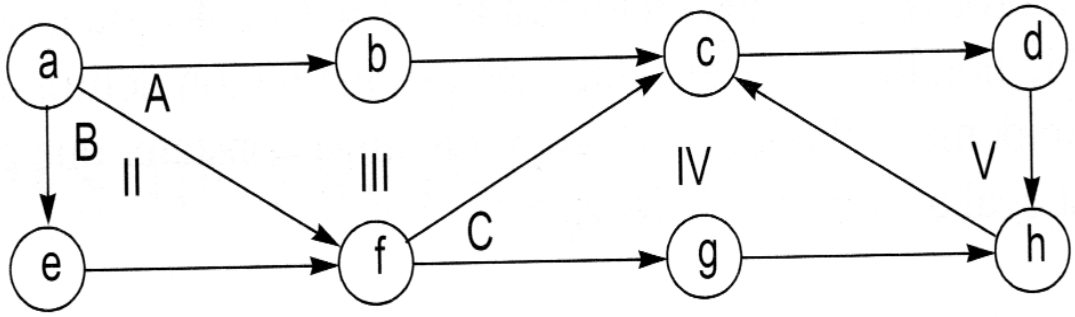
- Une instruction **IF** compte pour 1 décision
- Une boucle **FOR** ou **WHILE** compte pour 1 décision
- Une instruction **CASE** ou tout autre embranchement multiple compte pour une décision de moins que le nombre d'alternatives

➢ Métriques de Mac Cabe

❑ Nombre de faces et formule de Mc Cabe



4 décisions donc
C = 4 + 1 = 5



3 décisions donc
C = 3 + 1 = 4

- Pourtant, même nombre de faces : la formule de Mc Cabe serait incorrecte ? Non
- Le second graphe est invalide parce qu'il ne possède pas de nœud d'arrivée

Technique de génie logiciel

➢ Métriques de Halstead

❑ Les métriques de complexité de Halstead ont été développées par l'américain Maurice Halstead et procurent une mesure quantitative de complexité. Ils sont basées sur l'interprétation du code comme une séquence de marqueurs, classifiés comme un opérateur ou une opérande.

❑ Toutes les métriques de Halstead sont dérivées du nombre d'opérateurs et d'opérandes :

- Nombre total des opérateurs uniques (n1)
- Nombre total des opérateurs (N1)
- Nombre total des opérandes uniques (n2)
- Nombre total des opérandes (N2)

Technique de génie logiciel

- **Métriques de Halstead**
  - **Science informatique de Halstead**
    - Métriques pour la complexité d'un programme
      - Fondées empiriquement
      - Toujours considérées comme valides, contrairement à ses formules complexes de prédiction
    - Entités de base
      - Opérandes : jetons qui contiennent une valeur
      - Opérateurs : tout le reste (virgules, parenthèses, opérateurs arithmétiques...)

Technique de génie logiciel

- **Métriques de Halstead**
  - **Taille du vocabulaire**
  - Mesures de base n1 et n2 (eta)
    - n1 : nombre d'operateurs distincts
    - n2 : nombre d‘ opérandes distincts
    - n=n1 +n2 : nombre total de jetons distincts
    - Opérandes potentiels
      - Ensemble de valeurs minimal pour n'importe quelle implémentation
      - Pour comparer différentes implémentations du même algorithme
      - S'obtient généralement en comptant les valeurs qui ne sont pas initialisées à l‘ intérieur de l'algorithme
        - Valeurs lues par le programme
        - Valeurs passées en paramètres
        - Valeurs globales appelées depuis l'algorithme

➢ Métriques de Halstead

❑ Exemple

```
z = 0;
while x > 0
    z = z + y;
    x = x - 1;
end-while
print (z);
```

- Opérateurs :
  = ; while/end-while > + -
  print ()
  $\eta_1 = 8$
- Opérandes : = z 0 x y 1
  $\eta_2 = 5$

Technique de génie logiciel

➢ Métriques de Halstead

❑ Longueur d'un programme

- N1 : Nombre total d'opérateurs
- N2 : Nombre total d'opérandes
- N = N1 + N2 : Nombre total de jetons

❑ Exemple

```
z = 0;
while x > 0
    z = z + y;
    x = x - 1;
end-while
print (z);
```

Opérandes		Opérateurs	
=	3	Z	4
;	4	0	2
w/ew	1	X	3
>	1	Y	1
+	1	1	1
-	1		
Print	1		
( )	1		

N1 = 13, N2 = 11
donc N = 24

➢ Métriques de Halstead

❑ Estimation de la longueur

- Estimation de N a partir de n1 et de n2

$$N^{est} = n1 + \log2(n1) + n2 \times \log2(n2)$$

- Estimer la longueur de l'exemple :

$N^{est} = 8 \log2(8) + 5 \log2(5) = 8 \times 3 + 5 \times 2.32 = 35.6$

donc $N^{est} >> N$

- **Remarque :** pratiquement , si la différence entre N et N^{est} est supérieure à 30%, il vaut mieux renoncer à utiliser les autres mesures de Halstead

➢ Métriques de Halstead

❑ Volume

- Estimation du nombre de bits nécessaires pour coder le programme mesure
- A partir de là, on obtient le volume du Programme(V) en multipliant la taille du vocabulaire par le logarithme 2

$$V = N \times \log_2(n1 + n2)$$

$$V = N \times \log_2(n)$$

- Dans notre exemple : V = 25 X log2(13) = 25 X 3.7 = 92.5

Technique de génie logiciel

- **Métriques de Halstead**
  - **Autres mesures de Halstead**
    - **Le Niveau de difficulté (D) :** ou propension d'erreurs du programme est proportionnel au nombre d'opérateurs unique (n1) dans le programme et dépend également du nombre total d'opérandes (N2) et du nombre d'opérandes uniques (n2). Si les mêmes opérandes sont utilisés plusieurs fois dans le programme, il est plus enclin aux erreurs.

$$D = (n1 / 2) * (N2 / n2)$$

    - **Le Niveau de programme (L):** est l'inverse du Niveau de difficulté. Un programme de bas niveau est plus enclin aux erreurs qu'un programme de haut niveau.

$$L = 1 / D$$

    - **L'Effort à l'implémentation (E) :** est proportionnel au volume (V) et au niveau de difficulté (D).

$$E = V * D$$

Technique de génie logiciel

- **Métriques de Halstead**

- **Autres mesures de Halstead**
  - **Temps T :** temps nécessaire pour implémenter un algorithme . Est calculé en appliquant la formule suivante :

$$T = E / S$$

  - ou S est le nombre de Stroud. Dont la valeur retenu par Halstead est S = 18.

Technique de génie logiciel

➢ Métriques de Henry-Kafura

❑ Flux d'informations d'Henry-Kafura

- Mesurer la complexité des modules d'un programme en fonction des liens qu'ils entretiennent
- On utilise pour chaque module i :
  - Le nombre de flux d'information entrant note In_i
  - Le nombre de flux d'information sortant note Out_i
  - Le poids du module noté poids_i calcule en fonction de son nombre de LOC et de sa complexité

$$\mathbf{Hk_i = poids_i \ X \ (Out_i \ X \ In_i)^2}$$

❖ i : index du module dans le code.

- La mesure totale Hk correspond à la somme des Hk_i

➢ Métriques de Henry-Kafura

❑ Exemple

- A partir des In_i et Out_i , calcul des métriques HK en supposant que le poids de chaque module vaut 1

Module	a	b	c	d	e	f	g	h
in_i	4	3	1	5	2	5	6	1
out_i	3	3	4	3	4	4	2	6
HK_i	144	81	16	225	64	400	144	36

HK = 1110

➢ Métriques Objet de Chidamber et Kemerer

- ❑ Ensemble de métriques (Metric Suite for Object Oriented Design)
  - ◆ Evaluation des classes d'un système
  - ◆ La plupart des métriques sont calculées classe par classe
  - ◆ Le passage au global n'est pas clair, une moyenne n‘ étant pas très satisfaisante
- ❑ **M1 : Méthodes pondérées par classe :** C’est la seule métrique à calculer en moyenne sur toutes les classes
  - ◆ WMC : Weighted Methods per Class

$$WMC = \frac{1}{n} \times \sum_{i=0}^{n} c_i \times M_i$$

avec C un ensemble de n classes comportant chacune M_i méthodes dont la complexité (le poids) est note c_i

Technique de génie logiciel

- **Métriques Objet de Chidamber et Kemerer**

- **M2 : Profondeur de l'arbre d'héritage**
  - DIC : Depth of Inheritance Tree
    - Distance maximale entre un nœud et la racine de l'arbre d'héritage de la classe concernée
    - Calculée pour chaque classe

- **M3 : Nombre d'enfants**
  - NOC : Number Of Children
    - Nombre de sous-classes dépendant immédiatement d'une classe donnée, par une relation d'héritage
    - Calculée pour chaque classe

➢ Métriques Objet de Chidamber et Kemerer

❑ M4 : Couplage entre classes

- Dans un contexte OO, le couplage est l'utilisation de méthodes ou d'attributs d'une autre classe.
  - Deux classes sont couplées si les méthode déclarées dans l'une utilisent des méthodes ou instancie des variables définies dans l'autre;
  - La relation est symétrique : si la classe A est couplée à B, alors B l'est à A
- CBO : Coupling Between Object classes
  - Pour chaque classe, nombre de classes couplées;
  - Calculée pour chaque classe.

➢ Métriques Objet de Chidamber et Kemerer

❑ M5 : Réponses pour une classe (RFC)

- {RS}: ensemble des méthodes qui peuvent être exécutées en réponse a un message reçu par un objet de la classe considérée
  - Réunion de toutes les méthodes de la classe avec toutes les méthodes appelées directement par celles-ci
  - Calculée pour chaque classe

$$RFG = |RS|$$

➢ Métriques Objet de Chidamber et Kemerer

❑ M6 : Manque de cohésion des méthodes

- Un module (ou une classe) est < cohesif > lorsque tous ses éléments sont étroitement liés
- LCOM (Lack of COhesion in Methods) tente de mésurer l'absence de ce facteur
- Posons
  - Ii l'ensemble des variables d'instance utilisées par la méthode i
  - P l'ensemble les paires de (Ii ; Ij) ayant une intersection vide
  - Q l'ensemble des paires de (Ii ; Ij) ayant une intersection non vide

$$LCOM = \max(|P| - |Q|, 0)$$

➢ Métriques MOOD

- ❑ Ensemble de métriques pour mesurer les attributs des propriétés
  - Encapsulation
  - Héritage
  - Couplage
  - Polymorphisme

Technique de génie logiciel

- **Métriques MOOD**
  - **Encapsulation**
    - **MHF : Method Hiding Factor**

$$MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC} M_d(C_i)}$$

- $M_d(C_i)$ le nombre de méthodes déclarées dans une classe C_i
- $M_h(C_i)$ le nombre de méthodes cachées
- TC le nombre total de classes.

    - **AHF : Attribute Hiding Factor**

$$AHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)}$$

- $A_d(C_i)$ le nombre d'attributs déclarés dans une classe C_i
- $A_h(C_i)$ le nombre d'atributs cachés

- **Métriques MOOD**
  - **Héritage**
    - **MIF : Method Inheritance Factor**

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

- $M_i(C_i)$ le nombre de méthodes héritées (et non surchargées) de C_i
- $M_a(C_i)$ le nombre de méthodes qui peuvent être appelées depuis la classe i

    - **AIF : Attribute Inheritance Factor**

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

- $A_i(C_i)$ le nombre d'attributs hérités de C_i
- $A_a(C_i)$ le nombre d'attributs auxquels C_i peut accéder

- **Métriques MOOD**
  - **Couplage**
    - **CF : Coupling Factor :** mesure le couplage entre les classes sans prendre en compte celui du à l'héritage

$$CF = \frac{\sum_{i=1}^{TC}\sum_{j=1}^{TC} client(C_i, C_j)}{TC^2 - TC}$$

- $client(C_i, C_j) = 1$ si la classe i a une relation avec la classe j, et 0 sinon
- Les relations d'héritage ne sont pas prises en compte dans les relations

- **Métriques MOOD**
  - **Polymorphisme**
    - **PF : Polymorphism Factor** : mesure le potentiel de polymorphisme d'un système

$$PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} M_n(C_i) \times DC(C_i)}$$

- $M_o(C_i)$ le nombre de méthodes surchargées dans la classe i
- $M_n(C_i)$ le nombre de nouvelles méthodes dans la classe i
- $DC(C_i)$ le nombre de descendants de la classe i

La semaine prochaine

inchaa lah

Gestion des risques

➢ **Définition**

❑ Un risque est la probabilité qu'un événement indésirable ait lieu. Le risque implique des idées de incertitude ou les événements ne se produiront pas de manière certaine ainsi une perte de tel sorte que plus l‘ évènement est indésirable, plus le risque est grand.

❑ Une gestion proactive des risques peut aider a minimiser les effets négatifs d‘ évènements susceptibles de se produire

❑ **Types de risques :**

- Les risques de projet concernent le déroulement du projet
- Les risques techniques portent sur la qualité du produit
- Les risques commerciaux peuvent affecter sa viabilité

Gestion des risques

➢ **Exemple**

Risque	Projet	Tech.	Com.	Propre
Matériel non disponible		×		×
Spécifications incomplètes	×			
Utilisation de méthodes spécialisées		×		×
Problèmes pour atteindre la fiabilité désirée		×		×
Départ d'une personne clé	×			
Sous-estimation des efforts nécessaires	×			
Le seul client potentiel fait faillite			×	×

Gestion des risques

➢ **Calcul des risques**

❑ Utilisation de probabilités élémentaires

- Estimer la probabilité du risque
- Estimer l'impact, le coût des conséquences
- Calculer le risque en multipliant ces deux valeurs

❑ **Exemple :** jeu à deux dés à six faces

- L'obtention d'un 7 fait perdre 60 euros, quel est le risque ?
  - ❖ Il y a 6 façons d'obtenir un 7, soit une probabilité de 6/36 = 1/6
  - ❖ L'impact est de 60 euros
  - ❖ Le risque vaut donc 1/6 X 60 = 10 euros

Gestion des risques

➢ Atténuation des risques

❑ Stratégie proactive pour tenter de diminuer l'impact d'un risque et la probabilité d'un risque

❑ Donc il faut :

- Identifier très tôt les risques les plus importants
- Utiliser un cycle de vie incrémental et fonde sur les risques
- Prototyper autant que possible

Gestion des risques

➢ Atténuation des risques

❑ Stratégie proactive pour tenter de diminuer l'impact d'un risque et la probabilité d'un risque

❑ Donc il faut :

- Identifier très tôt les risques les plus importants
- Utiliser un cycle de vie incrémental et fonde sur les risques
- Prototyper autant que possible

Tests Logiciels

- **Test logiciel :** tester un logiciel c'est exécuter le logiciel avec un ensemble de données réelles
- Il faut confronter résultats de l'exécution et résultats attendus
- Impossibilité de faire des tests exhaustifs
- **Choix des cas de tests :**
  - Il faut couvrir au mieux l'espace d'entrées avec un nombre réduit d'exemples
  - Les zones sujettes à erreurs nécessitent une attention particulière

Tests Logiciels

- **Tests fonctionnels**
  - Identification à partir des spécifications des sous-domaines à tester
    - Produire des cas de test pour chaque type de sortie du programme
    - Produire des cas de test déclenchant les différents messages d'erreur

  - Objectif : disposer d'un ensemble de cas de tests pour tester le programme complètement lors de son implémentation

  - Test « boite noire » parce qu'on ne préjuge pas de l'implémentation
    - Identification possible des cas de test pertinents avant l'implémentation
    - Utile pour guider l'implémentation

Tests Logiciels

➢ Tests structurels

❑ Tests « boite blanche » : détermination des cas de test en fonction du code.

❑ Critère de couverture : règle pour sélectionner les tests et déterminer quand les arrêter.

❑ Au pire, on peut sélectionner des cas de test au hasard jusqu'à ce que le critère choisi soit satisfait.

❑ **Couverture de chaque instruction (C0)**

- Selon ce critère, chaque instruction doit être exécutée avec des données de test
- Sélectionner des cas de test jusqu'a ce qu'un outil de couverture indique que toutes les instructions du code ont été exécutées

Tests Logiciels

➢ Tests structurels

❑ Test de toutes le branches (C1)

- Plus complet que C0
- Selon ce critère, il faut emprunter les deux directions possibles au niveau de chaque décision
- Nécessite la création d'un graphe de contrôle et de couvrir chaque arc du graphe

❑ Test de tous les chemins

- Encore plus complet
- Selon ce critère, il faut emprunter tous les chemins possibles dans le graphe
- Chemin : suite unique de nœuds du programme exécutés par un jeu spécifique de données de test
- Peu adapte au code contenant des boucles, le nombre de chemins possible étant souvent infini

➢ Tests structurels

❑ Couverture des conditions multiples

- Un critère de test de conditions multiples impose que chaque condition primitive doit ^être évaluée a la fois comme vraie et comme fausse et que toutes les combinaisons V/F entre primitives d'une condition multiple doivent être testées
- Très complet et ne pose pas de problème en cas d'itérations

Tests Logiciels

- **Test de flot de données**
  - Test structurel
  - S'appuie sur la circulation des données a l'intérieur du programme

  - Elles circulent depuis l' endroit ou elles sont définies jusqu'à celui ou elles sont utilisées
    - **def** : définition d'une donnée, correspond a l'attribution d'une valeur a une variable
    - **c-use** : utilisation pour calcul, la variable apparaît a droite d'un operateur d'affectation
    - **p-use** : utilisation comme prédicat dans une condition

  - L'utilisation p-use est attribuée aux deux branches conditionnelles

  - un chemin sans définition, note def-free, va de la définition d'une variable a une utilisation sans passer par d'autres définition

- **Test de flot de données**
  - **Critères de tests de flots de données**
    - **dcu** : chaque définition doit être reliée a un c-use en aval par un chemin sans définition
    - **dpu** : chaque p-use doit être reliée a une définition en amont par un chemin sans définition
    - **du** : combinaison des deux précédents
    - **all-du-paths** : le plus exigeant, qui veut que tous les chemins possibles entre une définition et une utilisation doivent être sans définition

Tests Logiciels

- **Tests orientes objet**
  - Couverture des instructions
  - Couverture des branches
  - Couverture du flot de données
  - Ces techniques s'appuient sur un diagramme de contrôle
  - Ces diagrammes sont mal adaptes pour représenter la structure d'un logiciel objet
  - Les complexités de ces logiciels objet résident plutôt dans les interactions entre les méthodes
  - Il faut couvrir les appels de méthodes

Tests Logiciels

- **Tests orientes objet**
  - **Couverture pour les tests objet**
    - **Tests MM (méthode, message) :** tester tous les appels de méthode, si une méthode en appelle une autre plusieurs fois, chaque appel est teste séparément
    - **Couverture des paires de fonctions :** tester tous les enchaînements de deux méthodes possibles avec l'utilisation d'expressions régulières pour identifier les paires à couvrir