M1 info

GINF41B2 (Conception et Programmation Orientée Objet)

Cours #1
GL et COO

Pierre Tchounikine

Rappels / GL
Approche COO *vs.* approche fonctionnelle

(Cours #2 = approche COO en tant que telle)

Intro GL

Définition

Le Génie Logiciel est l'ensemble des activités de conception et de mise en œuvre des produits et procédures tendant à rationaliser la conception et le suivi du logiciel.

C'est l'art de spécifier, concevoir, réaliser et faire évoluer, avec des moyens et dans des délais fixés, des composants logiciels et leur documentation.

Problématique

Le génie logiciel est né d'un certain nombre de constats :

- il est extrêmement difficile de construire des gros logiciels et, notamment, il est extrêmement difficile de gérer
  - le coût du logiciel
  - le temps de développement du logiciel
  - la fiabilité du logiciel
- construire des gros logiciels ne pose pas les mêmes problèmes que construire des petits logiciels

le pont au dessus du ruisseau ⟷ *le pont au dessus du Rhône*

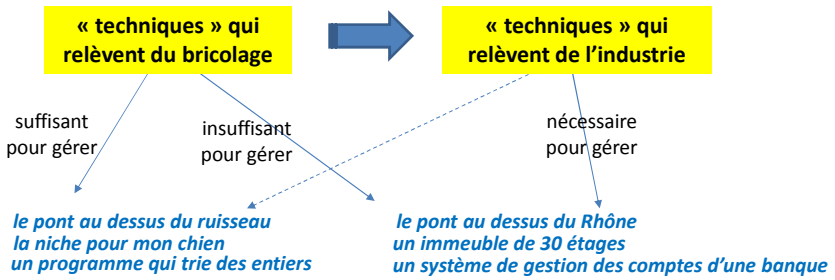
la niche pour mon chien ⟷ *un immeuble de 30 étages*

un programme qui trie des entiers ⟷ *un système de gestion des comptes d'une banque*

Objectif du Génie Logiciel

L'objectif général du Génie Logiciel est de permettre le développement de « bons » logiciels de façon systématique (« ingénierie »)

- dans des délais connus à l'avance et respectés (ou, au moins, gérés)
- dans des coûts connus à l'avance et respectés (ou, au moins, gérés)



Qualités d'un « bon logiciel »

- adéquat : qui correspond au cahier des charges, qui propose les fonctionnalités attendues
- maintenable : qui peut être modifié facilement et avec un coût raisonnable (un logiciel a une durée de vie)
- fiable : qui fait ce qui est prévu, dans tous les cas
- robuste : qui continue à fonctionner en environnement dégradé
- exploitable : qui est facile à apprendre et à utiliser
- adaptable : qui peut être utilisé dans d'autres contextes (mutatis mutandis)
- intègre : qui permet de gérer les restrictions d'accès des utilisateurs, où l'accès aux données et à la mémoire est contrôlé, etc.
- souple : qui facilite les modifications liées à la maintenance et l'évolution
- réutilisable : dont certains composants peuvent être réutilisés
- lisible : dont le code source est facilement lisible et compréhensible

Améliorer la production de logiciels

- Organisation et gestion de projet (modèle en V, etc.)
- Méthode de conception (préceptes, notations, etc.)
- Architecture (structures réutilisables)

cf. cours Ph. Lalanda

- **Conception-Modélisation / Programmation**

focus du cours

Travailler au niveau modélisation / programmation

Niveau modélisation/programmation

- Le niveau d'abstraction et puissances des langages
  - plus la différence entre les concepts identifiés lors de la conception et les structures du langage sont importantes, plus il y a de risques d'erreur (d'où le problème d'utiliser des langages comme C)
  - si un langage permet de faire des choses incohérentes, il y aura des choses incohérentes (d'où le problème d'utiliser des langages comme C)
- Les programmeurs

  on apprend à programmer et on prend des (mauvaises) habitudes de programmation en travaillant seul et sur des petits programmes
  - la programmation ne doit pas être considérée comme un « jeu » mais comme un travail rationnel : les entreprises n'ont pas besoin de « programmeurs fous » mais de « programmeurs compétents »
  - il faut travailler sur la qualité du code : lisibilité, maintenance, évolution, documentation du code
  - il faut apprendre à travailler en équipe et sans ego
  - …

Niveau modélisation/programmation

- Le niveau d'abstraction et puissances des langages
  - plus la différence entre les concepts identifiés lors de la conception et les structures du langage sont importantes, plus il y a de risques d'erreur (d'où le problème d'utiliser des langages comme C)
  - si un langage permet de faire des choses incohérent…

utilisation pertinente des notions, langages, composants et environnements de programmation

- Les progra…

  on apprend à progra… de programmation en travaillant seul et sur des petits programmes
  - la programmation ne doit pas être considérée comme un « jeu » mais comme un travail rationnel : les entreprises n'ont pas besoin de « programmeurs fous » mais de « programmeurs compétents »
  - il faut travailler sur la qualité du code : lisibilité, maintenance, évolution, documentation du code
  - il faut apprendre à travailler en équipe et sans ego
  - …

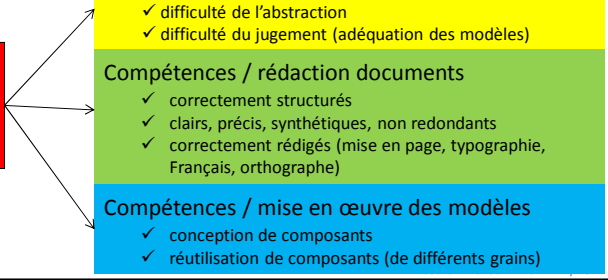
Idée importante à bien comprendre

Le logiciel est un produit immatériel que l'on ne « voit » que très (trop) tard → **rôle essentiel des documents** :

- documents de définition des besoins
- documents de conception
- documents de modélisation
- documents de développement
- etc.

et des prototypes
et des livraisons incrémentales

Le Génie Logiciel relève des sciences de l'artificiel, l'outil de base est la modélisation (et la documentation des modèles)



Introduction à la conception orientée objet

La conception

- La conception du logiciel est le processus qui permet de passer de l'expression des besoins à la phase de développement
- La conception est
  - un processus créatif
    → pas de recettes, on apprend par expérience
  - un processus itératif
    on ne produit pas une conception en un seul jet, on travaille toujours pas affinements successifs (une première esquisse informelle, puis on précise, souvent avec des retours arrières)
- La conception consiste à décrire le système
  - selon différents angles
  - à différents niveaux d'abstraction (en précisant de plus en plus)

La conception

- On distingue 2 grandes approches de conception :
  - La conception fonctionnelle
    - le processus est centré sur les fonctions que doit assurer le logiciel (le logiciel est vu comme un ensemble de fonctions)
    - on identifie les fonctions principales, puis on les décompose (récursivement)
  - la conception objet
    - Le processus est centré sur les objets et leurs relations

Quelle que soit l'approche adoptée, on constate que le fait de se focaliser trop tôt sur les structures d'implantation (de travailler trop tôt au niveau des langages informatiques) :
- ✓ contraint la conception
- ✓ limite indûment l'éventail des possibilités
- ✓ induit des coûts de maintenance élevés

La conception fonctionnelle

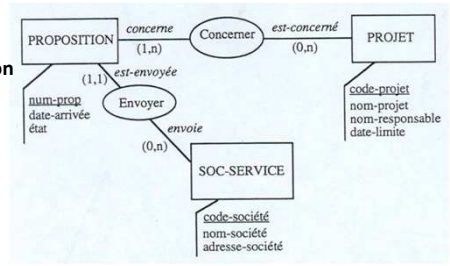
- Principes de base :
  - analyser les données
    par exemple, utilisation de schéma entités – associations
  - analyser le système du point de vue des transformations que subissent les données
    en général, utilisation de techniques de flot de données
  - structurer petit à petit le système (architecture)
    en général, utilisation de diagrammes de structure

➡ un système est perçu comme un ensemble de fonctions

La conception fonctionnelle

Précis de génie logiciel. M.C. Gaudel, B. Marre, F. Schlienger, G. Bernot. Ed. Masson

Schéma entité-relation



Flot de donnée

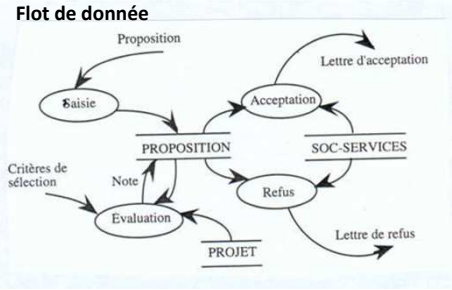
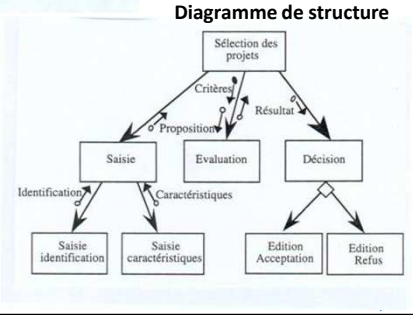


Diagramme de structure



La conception fonctionnelle

- Points forts de l'approche fonctionnelle :
  - analyse apparemment « simple et intuitive »
  - factorisation de fonctionnalités : si on découpe « bien » les choses, on peut réutiliser les mêmes fonctions à différents endroits
- Attention, le processus n'est jamais linéaire
  - on définit les choses à un premier niveau
  - on affine ...
  - ... et, souvent, on détecte de nouveaux points qui amènent à reconsidérer les structures des cycles précédents

Cf. modèles en V, en cascade, etc.

Limites de la conception fonctionnelle

- L'approche fonctionnelle est généralement considérée comme « naturelle » et intuitive car :
  - un système est perçu comme un ensemble de fonctions
  - la séparation fonction / donnée est cohérente avec le fonctionnement d'un ordinateur
- En fait
  - c'est une vision inspirée de la structure des ordinateurs
    → pourquoi est-ce qu'elle serait « naturelle » pour analyser les choses ?
  - c'est ainsi que l'on apprend à programmer, et c'est donc un schéma que l'on assimile très tôt ... et dont on a du mal à sortir

Limites de la conception fonctionnelle

- Les deux problèmes essentiels que posent l'approche fonctionnelle sont
  - la maintenance
  - la gestion de la complexité

l'approche objet a été « boostée » pour ses avantages / gestion (de l'évolution) des applications complexes

- On va mettre en évidence les limites de l'approche fonctionnelle à 2 niveaux :
  - d'un point de vue « pragmatique »
  - d'un point de vue plus « fondamental »

Vision « pragmatique » des limites de la conception fonctionnelle

Limites de la conception fonctionnelle

- La conception fonctionnelle mène à une structuration du système essentiellement liée aux fonctions que doit assurer le logiciel
- Lors des phases de maintenance on est amené à
  - reconsidérer des fonctions (ajouts, modifications, etc.)
  - travailler sur de nouvelles données
    - → de nouvelles structures
    - → de nouvelles façons de réaliser une fonction

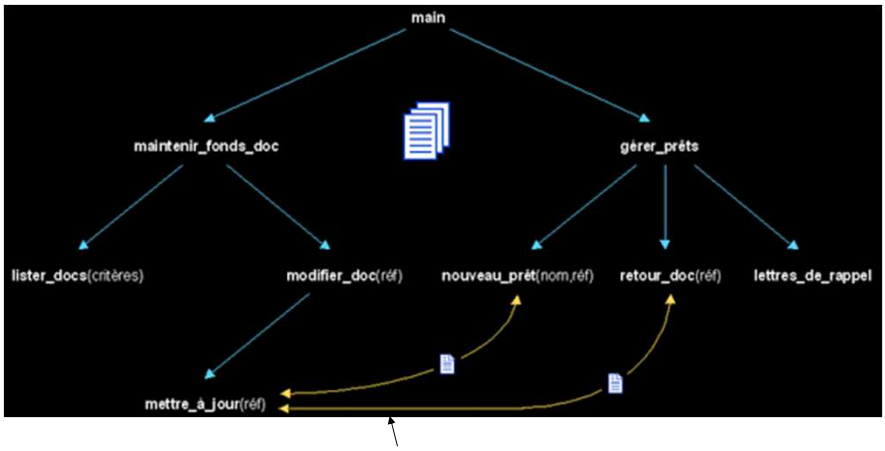
➡ modifications importantes car « en cascade »

la modification d'une structure de donnée entraîne la modification de toutes les fonctions liées à cette donnée : il peut y en avoir partout dans le système, et les modifications se répercutent en cascade

Limites de la conception fonctionnelle

http://uml.free.fr/

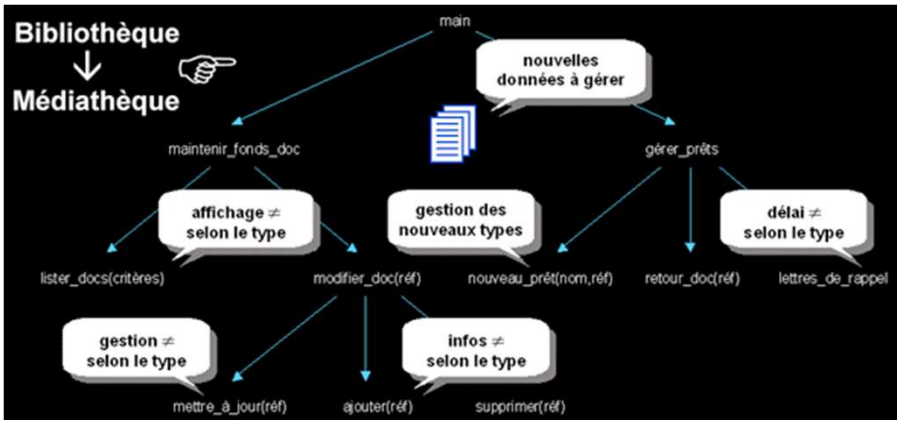
idée séduisante : structure du logiciel = ensemble de fonctions



idée séduisante : « factorisation » de fonctions

Limites de la conception fonctionnelle

mais si on décide de gérer modifier les objets / gérer de nouveaux objets …



Limites de la conception fonctionnelle

L'approche objet est une réponse à ce problème de « modification en cascade » :

en objet, on regroupe
- les données
- les traitements associés

{structure de donnée + fonctions} = 1 entité

notion de classe

⇒ quand les données évoluent, les perturbations sont
- localisées
- limitées

notion d'encapsulation

Vision « fondamentale » des limites de la conception fonctionnelle

Limites de la conception fonctionnelle

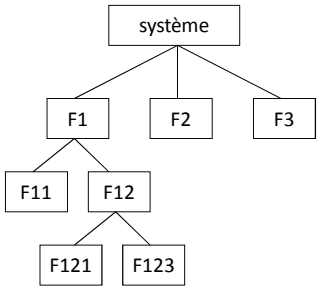
- conception fonctionnelle : on modélise ce que le système fait (les fonctions)

  ⇒ **couplage { fonctions / architecture } statique**
  (l'architecture est essentiellement liée aux fonctions)

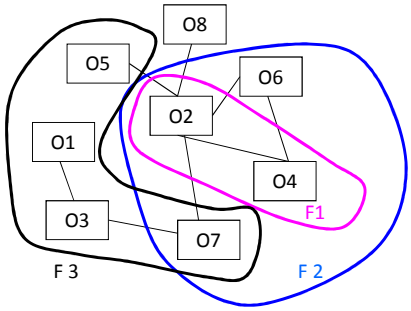
- conception objet : on modélise ce que le système est ET ce qu'il fait
  - on modélise la perception du monde que l'on a, les propriétés statiques et dynamiques de l'environnement (les objets, leurs relations, leurs comportements)
  - les fonctions sont le résultat de collaborations entre les objets qui composent le système

  ⇒ **couplage { fonctions / architecture } dynamique**
  (les évolutions du logiciel n'imposent pas nécessairement des changements de structures)

Conception fonctionnelle / objet



architecture de fonctions



réalisation de fonctions par collaboration d'objets

Avantages de la COO

- Avec l'approche objet la conception du système est essentiellement une démarche d'intégration de composants élémentaires

  « diviser pour construire en recomposant »

  → facilite la gestion de la complexité
  → facilite l'évolution et la maintenance (pendant et après la conception)
  → facilite la réutilisation

- En COO on s'appuie essentiellement sur un **modèle du domaine** et un modèle du domaine est **plus stable** que les besoins fonctionnels

  (les besoins fonctionnels correspondent à un problème dont la vision et la solution évoluent)

  « les fonctions changent plus souvent que les objets »

Limites de la conception fonctionnelle

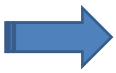
Par ailleurs,

- en conception fonctionnelle on est rapidement sur des aspects techniques
- en conception objet l'accent est mis sur **l'analyse et la modélisation des concepts** (les objets, les relations)

or

c'est au niveau de l'analyse que l'on peut faire les gains les plus appréciables

(quand on aborde les problèmes d'un point de vue technique on a beaucoup plus de problèmes)



on va bosser sur la COO / POO