Préambule : adresses utiles

- LIF7 : Conception et développement d'applications
  http://licence-info.univ-lyon1.fr/LIF7
  (renvoi vers http://liris.cnrs.fr/alexandre.meyer/wiki/doku.php?id=lif7)
- Responsable d'UE : Alexandre MEYER (cours, TD, TP)
  alexandre.meyer@univ-lyon1.fr
- Autres intervenants :
  - Julien MILLE (cours, TD, TP)
    julien.mille@univ-lyon1.fr
  - Elsa FLECHON (TP)
    elsa.flechon@liris.cnrs.fr
  - Jean-David GENEVAUX (TP)
    jean-david.genevaux@liris.cnrs.fr

Gestion de projet

LIF7 – Semestre printemps 2013

Julien MILLE
julien.mille@univ-lyon1.fr

LIF7 : Objectifs

- Développer un projet complet
- Outils
- Méthodes
- Travail en groupe
- Règles de programmation

LIF7 : Organisation

- 4 x 2h cours
- 5 x 2h TD
- 11 x 4h TP (soutenances lors de la dernière séance)
- Contrôle continu
  - Interrogation en cours/TD
  - TP à rendre
  - Examen en amphi
  - Projet

Bilan des UE

- LIF1 : initiation
  - notions de bases, fonctions, arguments, etc.
- LIF5 : programmation et structures de données
  - solutions classiques à quelques problèmes (boîte à outils).
- LIF7 : développement d'un projet
  - expérience et mise en application des notions acquises dans les UE précédentes.
- LIF9 : algorithmes et complexité
  - évaluer les qualités et les défauts des algorithmes.

Pour aujourd'hui…

- Notions de gestion de projet
- Programmation modulaire
- Outils et règles de programmation

Définition de projet

- **Projet**
  - l'ensemble des **actions** à entreprendre
  - afin de répondre à un **besoin**
  - défini dans des **délais** fixés (début, fin)
  - mobilisant des **ressources** identifiées (humaines et matérielles)
  - possède également un **coût** : budgétisation
  - on appelle «**livrables**» les résultats attendus du projet
- Maître d'ouvrage (demandeur du projet) ≠ maître d'œuvre (réalisateur du projet)

7

Problèmes classiques en développement logiciel (1/3)

- On passe son temps à réinventer la roue :
  - Perte de temps, perte d'argent
- Génération après génération, les informaticiens refont les mêmes erreurs:
  - Mauvaise transmission du savoir-faire
- La maintenance de la plupart des logiciels est difficile à faire
- Presque tous les développements logiciels coûtent cher

8

Problèmes classiques en développement logiciel (2/3)

- Les logiciels requièrent plus de flexibilité que jamais
  - Les technologies évoluent rapidement:
    - Langages
    - Intergiciels (en anglais, *middleware*)
    - Composants
    - Protocoles
    - …
  - La plupart des applications écrites aujourd'hui devront survivre à des changements technologiques (plateforme, système d'exploitation, langage, …)

9

Problèmes classiques en développement logiciel (3/3)

- De nombreux projets logiciels échouent
  - Dépassement de budget
  - Echéances manquées
  - Projets arrêtés avant que le logiciel ne soit livré à un seul client (ou utilisateur)

10

Logiciel de très grande taille (LTGT)

Tentative de définition

- Quantitativement:
  - C: > 100'000 lignes de code
  - Java: > 1'000 classes
- Qualitativement:
  - Pas possible d'avoir tout le code en tête
  - Nombreux développeurs
  - Longue durée de vie
  - Investissement élevé (argent, années-hommes)
  - Coût du *reengineering* prohibitif

11

Plusieurs types de LTGT

- Scientifique:
  - Ex.: modèles météorologiques, astronomiques
  - Beaucoup de calculs, peu d'utilisateurs
  - Centralisé ou fortement distribué (MPP, Grid)
  - Fortran, C
- E-business:
  - Ex.: serveur d'application e-business
  - Systèmes distribués
  - Très grand nombre de clients
  - Java, C#, XML
- …

12

Problèmes de conception critiques pour les LTGT

- Passage à l'échelle (*scalability*):
  - Serveur Web: 10'000, 100'000 ou 1'000'000 requêtes par jour?
- Robustesse
- Hétérogénéité :
  - A l'instant t
- Fréquents changements de technologie :
  - Au fil du temps
- Changements de cahier des charges:
  - Le *reengineering* complet n'est pas possible

13

Méthodes de conception

Cycle de développement

- Cycle en cascade
- Cycle itératif
- Cycle en V
- Cycle en spirale
- Méthode AGILE
- Extreme programming
- … Scrum et d'autres

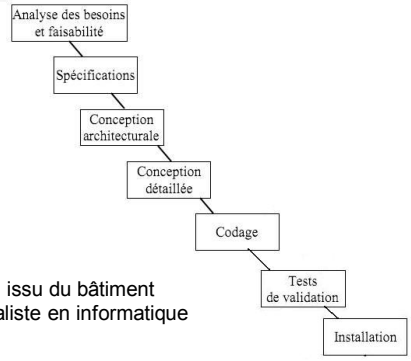
14

Cycle en cascade

- Analyse des besoins (quoi), spécification
- Conception architecturale, de haut niveau (comment)
- Conception détaillée (comment en détail)
- Codage
- Débogage (proc. d'assurance qualité)
- Déploiement
- Tests/Utilisation

15

Cycle en cascade



Modèle issu du bâtiment
Peu réaliste en informatique

16

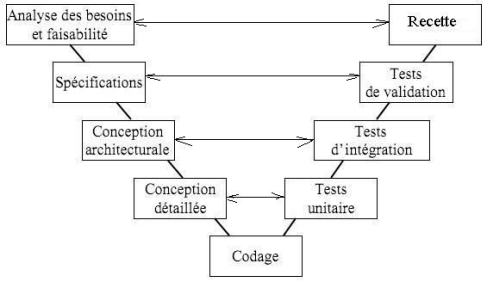
Cycle itératif

1. Analyse des besoins (quoi), spécification
2. Conception architecturale, de haut niveau (comment)
3. Conception détaillée (comment en détail)
4. Codage
5. Débogage (proc. d'assurance qualité)
6. Déploiement
7. Tests/Utilisation → Version 1.0

→ on recommence en 1 pour la version 2.0
…

17

Cycle en V



Cascade, itératif ou en V
Problème : globalement il n'y a que la version finale qui tourne

18

Cycle en spirale

- Cycle => version beta 1 (v0.1)
- Cycle => version beta 2 (v0.2)
- …
- Cycle => version 1.0

Par l'implémentation de versions successives, le cycle recommence en proposant un produit de plus en plus complet et dur.

Remarque : la méthode que nous allons utiliser … en l'appliquant à l'informatique ▪ AGILE

19

Méthode *AGILE*

- Suis un cycle en spirale
- Méthode de développement informatique
- Impliquant au maximum le demandeur (client)

▪ permet une grande réactivité à ses demandes

▪ plus pragmatiques que les méthodes traditionnelles

▪ visent la satisfaction réelle du besoin du client, et non d'un contrat établi préalablement.

20

Méthode *AGILE*

- Valeurs
  - **L'équipe**
    - La communication est une notion fondamentale dans l'équipe.
    - \+ une équipe de développeurs moyens qui communique
    - \- Une équipe de très bons développeurs sans communication
  - **L'application**
    - Il est vital que l'application fonctionne.
    - Documenter le code (et mettre à jour la documentation)
  - **La collaboration**
    - Le client doit collaborer avec l'équipe et fournir un feed-back continu
  - **L'acceptation du changement**
    - planification et structure du logiciel doivent être flexibles
    - afin de permettre l'évolution de la demande du client tout au long du projet

21

Méthode *AGILE*

Principes (une douzaine)

- Satisfaire le client en livrant tôt et régulièrement des logiciels utiles
- Le changement est bienvenu
- Les gens de l'art et les développeurs doivent collaborer quotidiennement au projet
- Bâtissez le projet autour de personnes motivées
- La méthode la plus efficace de transmettre l'information est une conversation en face à face
- Un logiciel fonctionnel est la meilleure unité de mesure de la progression du projet
- Les processus agiles promeuvent un rythme de développement soutenable
- La simplicité - l'art de maximiser la quantité de travail à ne pas faire - est essentielle
- Les meilleures architectures, spécifications et conceptions sont issues d'équipes qui s'auto-organisent
- À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace
- …

22

Extreme Programming

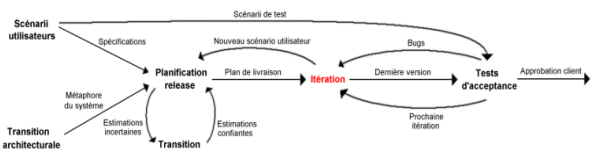
- est une méthode *AGILE*
  - la revue de code est une bonne pratique, elle sera faite en permanence (par un binôme) ;
  - les tests sont utiles, ils seront fait systématiquement à chaque implémentation ;
  - la conception est importante, elle sera faite tout au long du projet
  - la simplicité permet d'avancer plus vite, nous choisirons toujours la solution la plus simple
  - la compréhension est importante, nous définirons et ferons évoluer ensemble des métaphores
  - l'intégration des modifications est cruciale, nous l'effectuerons plusieurs fois par jour
  - les besoins évoluent vite, nous ferons des cycles de développement très rapides pour nous adapter au changement

23

Extreme Programming

Cycles rapides de développement (des itérations de quelques semaines) :

- Phase d'exploration avec le client détermine les scénarios ;
- L'équipe : scénarios →planifient des tâches et tests fonctionnels ;
- Développeur s'attribue des tâches et les réalise en binôme;
- Tous les tests fonctionnels OK→ produit



Premier bilan

- Que le cycle soit long ou court, ces méthodes se basent toutes sur des documents formalisés
  - Même si, avec les méthodes « agiles », la forme du document a moins d'importance que le bon fonctionnement du logiciel
  - L'oral ne suffit pas ! Car si changement de développeurs en cours de dev, etc.
- Une équipe de développement produit
  - Le cahier des charges avec des diagrammes de modules
  - Le code géré par une base de données de code (svn)
  - La documentation du code → type doxygen
- Vous ferez pareil!

25

Cahier des charges

- Un **cahier des charges**
  - un document visant à définir exhaustivement les spécifications de base d'un produit (même si version beta) ou d'un service à réaliser.
  - modalités d'exécution
  - les objectifs à atteindre et vise à bien cadrer une mission.
  - le prestataire doit anticiper ou freiner les exigences du client (non-informaticien), selon le cas

Le cahier des charges est un document contractuel, il fait office de contrat entre le client et le prestataire de service.

26

Cahier des charges

- Un plan possible
  - **Chapitre 1 – Présentation du projet**
    - Contexte, qui, historique, etc.
  - **Chapitre 2 - Description de la demande**
    - Définir les résultats que le projet doit atteindre
    - Définir les fonctionnalités du produit
  - **Chapitre 3 - Contraintes**
    - Coût, durée de développement, budget, etc.
  - **Chapitre 4 - Déroulement du projet**
    - Planification : définir les grandes étapes du projet
    - Chaque étape est découpée en tâches
    - Pour chaque tâche on doit
      - Définir les ressources nécessaires: humaines, matérielles, etc.
      - Expliquer ce qu'elle doit faire et comment
      - Souvent définir un « livrable »=un résultat concret= du code, un test, etc.
  - **Annexes**

27

Cahier des charges : exemple 1/4

- **Chapitre 1 – Présentation du projet**
  - Qui? SNCF
  - Contexte? S'occupe des trains en France depuis toujours, société publique, etc.
- **Chapitre 2 - Description de la demande**
  - Résultats visés : construire un train rapide entre Lyon et Milan pour tous et tout le temps
  - Fonctionnalités :
    - durée < 2h, => vitesse>300km/h
    - Un train toutes les heures
    - Coût du billet < 50euros par voyage
    - 2 classes de wagon
- **Chapitre 3 - Contraintes**
  - Coût : budget < 10 Meuros sur fond propre + 50Meuros de l'état
  - Durée de développement : prêt dans 2 ans pour les JO

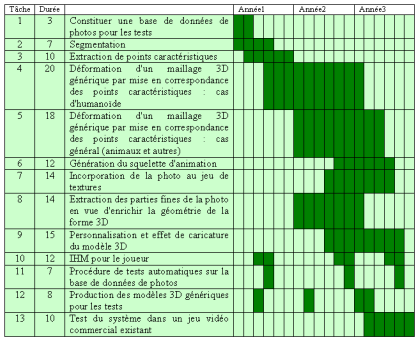
28

- **Chapitre 4 - Déroulement du projet**
  - Liste des tâches
    - Tâches 1
      - Construction des rails.
      - Livrable = les rails
      - Réalisé quand les rails seront dans l'entrepôt X.
    - Tâche 2 :
      - pose des rails.
      - Livrable=500km de rails entre Lyon et Paris
      - Validé quand rail posées en continue entre Lyon et Gare de Lyon.
    - Tâche 3 :
      - conception de la locomotive.
      - Validé quand le plan de la loco aura respecter les contraintes
      - Livrable=plan de la locomotive
    - Tâche 4 : construction de la locomotive. …. Livrable=1 locomotive
    - Tâche 5 : conception des wagons. … Livrable=2 plans de wagon.
    - Tâche 6 : construction des wagons. ….
    - Tâche 7 : test de la locomotive à grande vitesse sur rail. Validé quand la loco ira à 300km/h
    - … tâche 129
  - Les tâches dans le temps
    - Les milestones : après 3 mois nous aurons la version beta 1, après 5 mois la beta 2, etc.
    - + le schéma suivant

Cahier des charges : exemple 3/4

Schémas récapitulatifs du déroulement des tâches

Exemple de Diagramme de Gantt



30

Cahier des charges : exemple 4/4

Chapitre 5 – aspects financier, budget prévisionnel

- **Projet sur 48 mois**
- **SNCF**
  - **Personnels présents + environnements** — **2 014 039,80 €**
  - **Ingenieurs** — **2 056 500,00 €**
    dont salaires 1 042 500,00 €
  - **Fonctionnement (hors personnel)** — **230 342,00 €**
  - **Coût complet** — **5 000 515,48 €**
  - **Aide demandé** — **1 072 475,68 €**
- **Alstom**
  - **10 techniciens, 10 ingénieurs** — **3 064 500,00 €**
  - **Fonctionnement** — **3 010 096,00 €**
  - **Equipement** — **9 060 876,00 €**
  - **Coût complet** — **5 001 430,04 €**
  - **Aide demandé à l'état** — **2 030 874,02 €**
- **Coût complet** — **100 001 945,52 €**
- **Aide demandé** — **4 003 349,70 €**

31

Programmation modulaire

Règle d'intégrité
Diagramme de module

32

Programmation modulaire

- Votre code est découpé en **modules** (=.h/.c)
  - Les structures et les fonctions qui interviennent sur ces structures doivent être regroupées
  - Vue de l'extérieur : seules importent les fonctions déclarées dans le .h = la partie visible de l'iceberg
  - Règle d'intégrité : on ne touche pas directement aux champs de la structure, on passe par les fonctions
    - Simplifie l'utilisation
    - Recherche de bug
    - Maintient la cohérence éventuelle entre les champs
- Exemple : métaphore de la voiture
  - Le conducteur ne trifouille pas les câbles du moteur
  - Il utilise les fonctions : tourneVolant, changeVitesse, etc

33

Programmation modulaire

- Exemple : un module Voiture
  - Fichier **Voiture.h**

```
/* Directives pour éviter les inclusions multiples */
#ifndef VOITURE_H
#define VOITURE_H

/* Inclusions des en-têtes utilisées */
#include "Moteur.h"
#include "Volant.h"
#include "Roue.h"

/* Définitions des structures publiques (à usage
  externe au module) */
typedef struct
{
    Moteur mot;
    Volant vol;
    Roue roues[4];
    ...
} Voiture;
```

34

Programmation modulaire

```
/* Déclaration des fonctions publiques (à usage
  externe) */
void initVoiture(Voiture *);
void avancerVoiture(Voiture *, int);
void tournerVolant(Voiture *, int);
int getNiveauEssence(const Voiture *);
void fairePleinEssence(Voiture *);
...

#endif /* Termine le #ifndef VOITURE_H */
```

35

Programmation modulaire

- Fichier **Voiture.c**

```
/* Inclusion de l'en-tête du module */
#include "Voiture.h"

/* Inclusion d'en-têtes standards  */
/* Pour avoir sqrt(), cos(), sin(), ... */
#include <math.h>

/* Pour printf(), scanf(), ... */
#include <stdio.h>

/* Définitions de constantes */
const float _pi = 3.14159;
```

36

Programmation modulaire

```
/* Définitions des fonctions internes */
float convDegreRadian(int deg)
{
    float rad;
    rad = deg*_pi/180.0;
    return rad;
}
...

/* Définitions des fonctions publiques */
void initVoiture(Voiture *pVoit)
{
    initMoteur(pVoit->mot);
    initVolant(pVoit->vol);
    ...
}
void avancerVoiture(Voiture *pVoit)
{
    ...
}
...
```

37

Avant de coder...

- Réfléchir aux structures (types de données qui vont être manipulées dans l'application) et aux liens entre les différentes structures (ex: une voiture va contenir un moteur, un volant, ...)
- Réfléchir à l'organisation des structures en module (souvent, un module par structure)
- Réfléchir aux fonctions (que va t'elle faire ? que va t'elle retourner ? quels paramètres va t'elle prendre en entrée et en sortie ?) **sans pour autant écrire le corps des fonctions**
- Modéliser l'application sous forme **graphique** : nécessité d'avoir un modèle de **diagramme**

38

UML : Unified Modeling Languages

Comment s'y retrouver avec un programme comportant de nombreux modules ?

- **UML** = formalisme permettant de :
- définir et de visualiser un modèle à l'aide de **diagrammes**
- 13 types de diagramme existent
- Combinés, les différents types de diagrammes UML offrent une vue complète des aspects statiques et dynamiques d'un système.
- Nous ferons une utilisation très simplifiée d'UML
  - 1 diagramme
- Vous verrez plus d'UML en Génie Logiciel en M1
- Remarque : UML n'est pas une **méthode de conception** (≠ MERISE, SADT, etc.)

39

Diagramme des modules

Comment s'y retrouver avec un programme comportant de nombreux modules ? → Diagramme des modules

- Chaque module est décrit par une boîte contenant
  - Liste des types, structures
    - Pour chaque structure, la liste des champs (noms et types)
  - Liste des fonctions, procédures avec
    - Usage interne(-) / externe(+) au module
      - Interne=(-) n'apparaît pas dans le .h
      - Externe=(+) apparaît dans le .h (la partie visible de l'iceberg)
    - Pour chaque fonction, la liste des paramètres (types et modes d'accès) et le type retourné
- Une boîte = en quelque sorte un .h indépendant du langage

40

Diagramme des modules

Représentation d'un module

Module NombreComplexe
  structure NbComplexe {Re, Im : Réel}
+ Fonction somme(IN NbComplexe , IN NbComplexe) →NbComplexe
+ Fonction produit(IN NbComplexe , IN NbComplexe) →NbComplexe
+ Procédure affiche(IN NbComplexe)
  …
- Procédure reduire(IN-OUT NbComplexe)

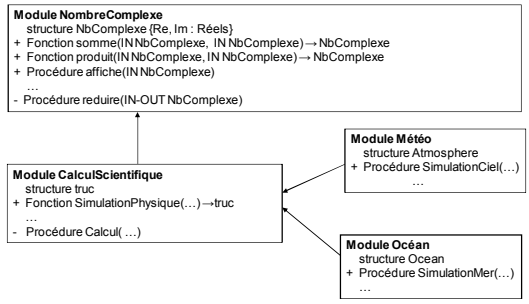
Notations
Accès aux fonctions :
+ : public → pour un usage externe au module
- : privé → pour un usage interne au module

Mode d'accès des paramètres :
IN : donnée → paramètre en entrée (lecture)
OUT : résultat → paramètre en sortie (écriture)
IN-OUT : donnée résultat → paramètre utilisé en entrée/sortie (lecture/écriture)

41

Diagramme des modules

Dépendances entre modules



⟶ La flèche signifie « utilise » ou « à besoin de »

42

Diagramme des modules

- Dépendances entre modules
  - La flèche signifie « utilise » ou « à besoin de »
  - Le module B dépend du module A si
    - B utilise au moins une fonction, une variable, un type déclaré dans le module A
    - Si la partie externe de A change en cours de vie du projet, il faut faire des modifications dans B
  - Les modules ayant de nombreuses flèches pointant dessus constituent **le noyau du projet → à écrire (et tester si possible) au début**

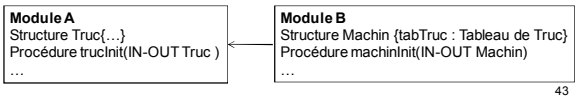


Diagramme des modules

- Cas des pointeurs
  - Le diagramme des modules doit, autant que possible, être indépendant du langage → éviter les pointeurs C
  - Attention : le logiciel Umbrello utilisé en TD utilise la notation *
  - Rôle des pointeurs en C et équivalent dans le diagramme
    - Passage de paramètre par adresse → à remplacer par les modes d'accès OUT ou IN-OUT dans le diagramme
    - Champ de structure représentant un lien / une relation vers une autre structure → indiquer explicitement 'lien'
    - Champ de structure destiné à être alloué en tableau (allocation dynamique) → indiquer explicitement 'tableau '
  - Remarque : un double pointeur (**) peut être un lien sur un tableau, un tableau de liens, un tableau 2D (matrice) → à préciser !

Diagramme des modules

- Cas des pointeurs
  - Exemple : structure en C et son équivalent dans le diagramme des modules

Code : Personne.h

```
struct Personne {
   char *nom;
   char *prenom;
   int age;
   ...
   struct Personne *ptrPere;
   struct Personne *ptrMere;
   struct Personne
            **tabPtrEnfants;
};
```

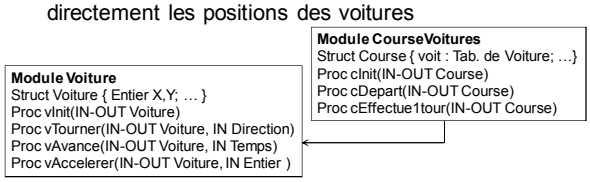
Equivalent diagramme

Module Personne
Structure Personne {
nom, prenom : chaîne
age : entier
…
ptrPere, ptrMere : lien sur Personne
tabEnfants : tableau de liens sur Personne
}

  - voir diapos 56 et 57 du 2ème cours

Intégrité d'un module

- Même si le compilateur le permet on ne modifie pas les champs d'une structure directement !
- On passe par l'intermédiaire des fonctions définies par le module
  - Fonction de type « accesseurs » et « mutateurs »
- Exemple : le module CourseVoiture ne modifie pas directement les positions des voitures



Dépendances : cas de l'IHM

- IHM = Interface Homme Machine
  - Menus déroulants, boîtes de dialogue, etc.
  - Souvent basée sur une librairie
- On doit pouvoir changer la librairie avec des modifications mineures des modules non-IHM
  - Par exemple : passer d'un PC à un téléphone portable
- Conséquence
  - Aucun module n'a besoin du module traitant l'IHM
  - Diagramme des modules
    - Aucune flèche n'arrive sur le module d'IHM
  - Le module d'IHM peut avoir besoin de tout le monde
  - Cf. cours librairie SDL

Pour résumer : qu'est ce qu'on attend de vous ?

- Un cahier des charges initial
  - Fixant les fonctionnalités à atteindre
    - Ex: Je veux que mon produit final fasse ça, ça et ça
  - Découpant le projet en sous-tâches
    - Il y aura N outils : un qui fait ça, un qui fait ça, etc.
    - Pour l'outil 1, je dois faire telle tâche et telle tâche
    - La tâche t sera considérée comme finie quand ceci marchera
    - La tâche t produira un livrable = du code, ou un test, ou une version du soft, etc.
  - Agençant la réalisation des tâches dans le temps = diagramme de Gantt
    - Après 1 mois les tâches 1,2 et 4 seront finis, etc.
- Diagramme de module : à maintenir pendant la durée du développement
- 1, 2 voir 3 cycles de productions du produit
  - A chaque fois
    - Définir la liste des sous-tâches et des tests à réaliser
    - Répartir les tâches entre chaque développeur
    - Consigner ceci dans un document court et précis
    - Réaliser les tâches et tests en
      - documentant le code (doxygen)
      - entrant votre code dans la base régulièrement (SVN)
  - Programmation modulaire

Applications

Exemples

Exercice : la FST

- La scolarité de la Faculté des Sciences et Technologies de Lyon 1 souhaite informatiser la gestion des étudiants, des enseignants et des UEs. Vous êtes chargés de développer une application en C.
- Il faut notamment pouvoir représenter :
  - l'ensemble des UEs proposées (code, intitulé, contenu, n° séquence, nombre de crédits ECTS, nombre de CM/TD/TP, …)
  - l'ensemble des enseignants intervenants (coordonnées, n° employé, statut, …)
  - la liste des étudiants inscrits dans l'UFR (coordonnées, n° étudiant, …)

Exercice : la FST

- Il faut notamment pouvoir modifier ou récupérer :
  - quel étudiant est (ou a été inscrit) à telle UE pendant tel semestre. On souhaite également faire un suivi des étudiants (notes, absences, remarques éventuelles, …)
  - quel enseignant intervient (ou est intervenu) dans telle UE pendant tel semestre
- Etapes
  - Rédaction du cahier des charges
  - Diagramme des modules
    - Ecriture des structures de données
    - …
  - …

Exercice : médiathèque

Gestion d'une médiathèque

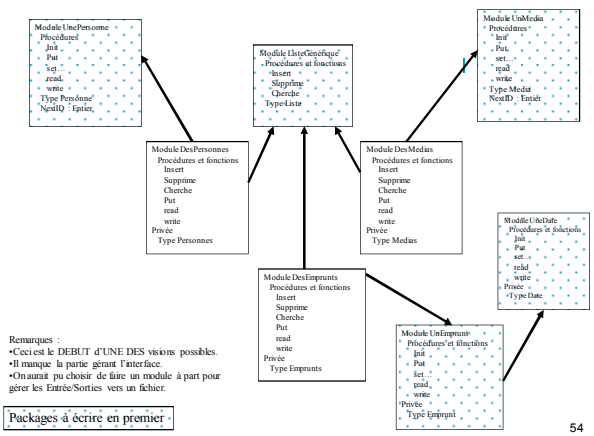
Une médiathèque vous demande de développer son logiciel de gestion des emprunts. Les employés de la médiathèque vous expliquent qu'ils aimeraient pouvoir gérer :

- Une liste d'emprunteurs avec leur nom, prénom, adresse (pour les retrouver en cas de problème) et leur âge (à des fins de statistique).
- Une liste de média avec le titre, l'auteur et le type : CD, livres, cassettes vidéo ou DVD.
- Un historique des emprunts : la personne qui emprunte, le média emprunter, la date de sortie et la date de retour (ou la date de retour prévu si le média est encore sorti).

Vous êtes plusieurs développeurs sur ce projet, en tant que chef de projet vous devez spécifier au mieux les différents modules nécessaires et leurs interactions afin que chaque développeurs puissent commencer à travailler. Ils ont besoin du graphe de dépendance et de la spécification de tous les modules (fonctions/procédures/types ; et leur utilisation interne ou externe).

Exercice : médiathèque

- Cahier des charges en fonction de la discussion avec les utilisateurs de la médiathèque
- Diagramme de modules

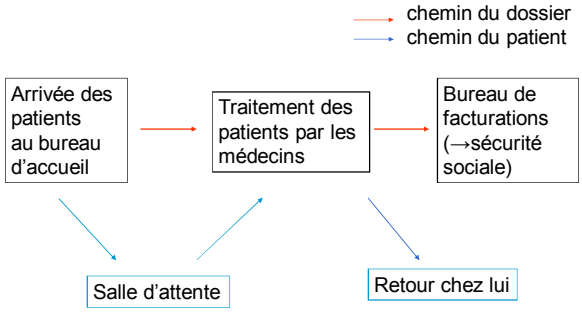


Exercice : gestion des entrées à l'hôpital

- Un hôpital souhaite automatiser le cheminement des patients
- Un informaticien passe quelques jours aux urgences, il
  - observe
  - discute
  - et revient avec un cahier de brouillon

55

Schéma de fonctionnement



56

Informations

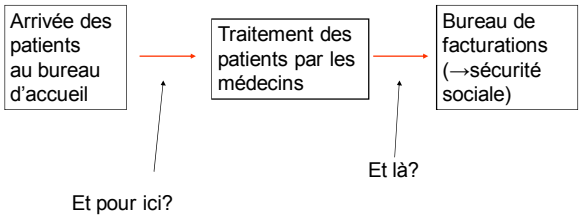
- Quelques mots-clefs
  - Patient : nom, prénom, adresse, numéro de sécurité sociale, …
  - Heure/jour d'entrée
  - Examen : température, tension, liste des symptômes (bras cassé, vomissement, saignements, pertes de conscience, …)
  - Diagnostic
  - Procédure de soins : traitements
    - Plâtre, 3 cachets de médicament X, 1 du médicament Y, etc.

57

Structures de données?

58

Structures de données?



59