IFT159
Analyse et programmation
Chapitre 3 — Conception descendante

Gabriel Girard

Département d'informatique
UNIVERSITÉ DE SHERBROOKE

7 septembre 2015

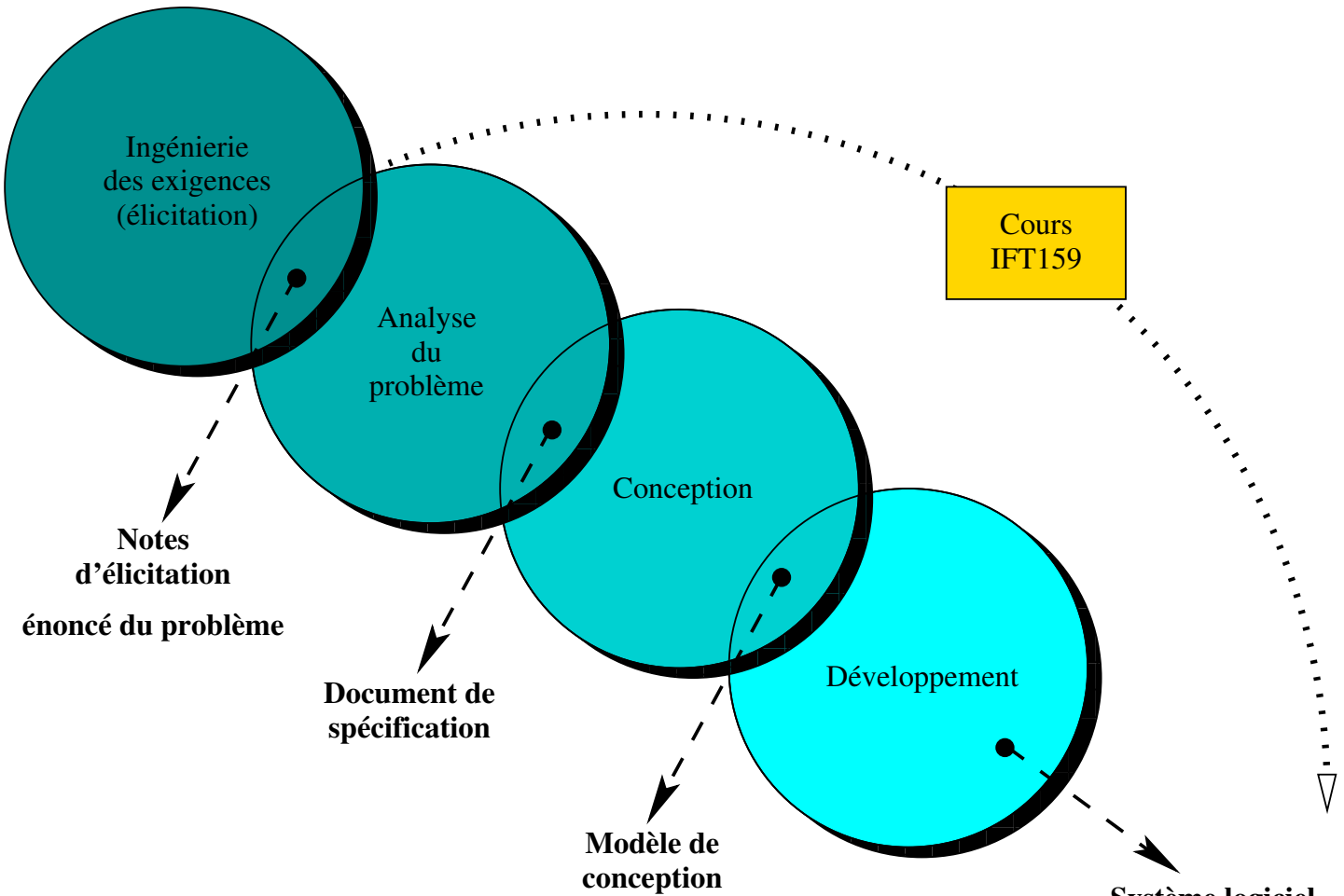
Chapitre 3 — Conception descendante

UNIVERSITÉ DE SHERBROOKE

Conception descendante

UNIVERSITÉ DE SHERBROOKE

Phases de développement



Conception descendante

Énoncé du problème

- Texte qui décrit le problème à résoudre

Énoncé du problème

- Texte qui décrit le problème à résoudre
- Doit permettre la compréhension du problème par soi-même

Énoncé du problème

- Texte qui décrit le problème à résoudre
- Doit permettre la compréhension du problème par soi-même
- Décrit les caractéristiques comportementales, fonctionnelles et opérationnelles du système.

Énoncé du problème

- Texte qui décrit le problème à résoudre
- Doit permettre la compréhension du problème par soi-même
- Décrit les caractéristiques comportementales, fonctionnelles et opérationnelles du système.
- Contient de l'information redondante, inutile, parfois contradictoire.

Questions

- Est-on familier avec le domaine d'application ?

Questions

- Est-on familier avec le domaine d'application ?
- Le problème à résoudre est-il bien défini ?

Questions

- Est-on familier avec le domaine d'application ?
- Le problème à résoudre est-il bien défini ?
- Les besoins du client peuvent-ils être modifié avec le temps ?

Questions

- Est-on familier avec le domaine d'application ?
- Le problème à résoudre est-il bien défini ?
- Les besoins du client peuvent-ils être modifié avec le temps ?
- Les besoins du client sont-ils bien exprimés ?

Questions

- Est-on familier avec le domaine d'application ?
- Le problème à résoudre est-il bien défini ?
- Les besoins du client peuvent-ils être modifié avec le temps ?
- Les besoins du client sont-ils bien exprimés ?
- Qui peut répondre à mes questions ?

Questions

- Est-on familier avec le domaine d'application ?
- Le problème à résoudre est-il bien défini ?
- Les besoins du client peuvent-ils être modifié avec le temps ?
- Les besoins du client sont-ils bien exprimés ?
- Qui peut répondre à mes questions ?
- Existe-t-il une solution au problème (faisabilité) ?

Questions

- Est-on familier avec le domaine d'application ?
- Le problème à résoudre est-il bien défini ?
- Les besoins du client peuvent-ils être modifié avec le temps ?
- Les besoins du client sont-ils bien exprimés ?
- Qui peut répondre à mes questions ?
- Existe-t-il une solution au problème (faisabilité) ?
- Quelles sont les ressources disponibles ?

Questions

- Est-on familier avec le domaine d'application ?
- Le problème à résoudre est-il bien défini ?
- Les besoins du client peuvent-ils être modifié avec le temps ?
- Les besoins du client sont-ils bien exprimés ?
- Qui peut répondre à mes questions ?
- Existe-t-il une solution au problème (faisabilité) ?
- Quelles sont les ressources disponibles ?
- Quels sont les résultats à produire ?

Questions

- Est-on familier avec le domaine d'application ?
- Le problème à résoudre est-il bien défini ?
- Les besoins du client peuvent-ils être modifié avec le temps ?
- Les besoins du client sont-ils bien exprimés ?
- Qui peut répondre à mes questions ?
- Existe-t-il une solution au problème (faisabilité) ?
- Quelles sont les ressources disponibles ?
- Quels sont les résultats à produire ?
- Les données d'entrée sont-elles disponibles ?

Questions

- Est-on familier avec le domaine d'application ?
- Le problème à résoudre est-il bien défini ?
- Les besoins du client peuvent-ils être modifié avec le temps ?
- Les besoins du client sont-ils bien exprimés ?
- Qui peut répondre à mes questions ?
- Existe-t-il une solution au problème (faisabilité) ?
- Quelles sont les ressources disponibles ?
- Quels sont les résultats à produire ?
- Les données d'entrée sont-elles disponibles ?
- Quels est la durée de vie prévue du logiciel ?

Questions

- Est-on familier avec le domaine d'application ?
- Le problème à résoudre est-il bien défini ?
- Les besoins du client peuvent-ils être modifié avec le temps ?
- Les besoins du client sont-ils bien exprimés ?
- Qui peut répondre à mes questions ?
- Existe-t-il une solution au problème (faisabilité) ?
- Quelles sont les ressources disponibles ?
- Quels sont les résultats à produire ?
- Les données d'entrée sont-elles disponibles ?
- Quels est la durée de vie prévue du logiciel ?
- etc.

Exemples d'énoncé de problème (1)

Écrivez un programme qui effectue une analyse bidimensionnelle ou profilométrique du relief de la peau humaine basée sur une décomposition spectrale respectant la réalité, c'est-à-dire l'existence de deux familles de plis, l'une liée au derme profond, l'autre au stratum corneum. Une analyse statistique permet de mesurer l'anisotropie du relief et ainsi de suivre le vieillissement cutané. Une étude tridimensionnelle permet à la fois la quantification de la distribution des hauteurs du relief d'un point de vue statistique et morphologique et celle de la répartition et de la densité des plis dans différentes directions de la surface.

Exemples d'énoncé de problème (2)

- Écrivez un programme qui accepte en entrée un programme écrit en C++ et qui retourne *vrai* si le programme s'arrête ou *faux* autrement.

Exemples d'énoncé de problème (2)

- Écrivez un programme qui accepte en entrée un programme écrit en C++ et qui retourne *vrai* si le programme s'arrête ou *faux* autrement.
- Écrivez un programme qui calcule le salaire moyen des travailleurs au noir.

Exemples d'énoncé de problème (3)

- Écrivez un programme qui calcule la moyenne de trois nombres entiers positifs. On veut que la moyenne soit une valeur entière.

Exemples d'énoncé de problème (3)

- Écrivez un programme qui calcule la moyenne de trois nombres entiers positifs. On veut que la moyenne soit une valeur entière.
- Écrivez un programme qui calcule l'aire et la circonférence d'un cercle.

Conception descendante

UNIVERSITÉ DE SHERBROOKE

Analyse

Objectifs

1. Comprendre et clarifier le problème : ambiguïté, complétude, cohérence ?

Analyse

Objectifs

1. Comprendre et clarifier le problème : ambiguïté, complétude, cohérence ?
2. Déterminer les données d'entrée et de sortie du problème.

Analyse

Objectifs

1. Comprendre et clarifier le problème : ambiguïté, complétude, cohérence ?
2. Déterminer les données d'entrée et de sortie du problème.
3. Déterminer grossièrement le traitement à faire :

Analyse

Objectifs

1. Comprendre et clarifier le problème : ambiguïté, complétude, cohérence ?
2. Déterminer les données d'entrée et de sortie du problème.
3. Déterminer grossièrement le traitement à faire :
   - fixer des hypothèses ;

Analyse

Objectifs

1. Comprendre et clarifier le problème : ambiguïté, complétude, cohérence ?
2. Déterminer les données d'entrée et de sortie du problème.
3. Déterminer grossièrement le traitement à faire :
   - fixer des hypothèses ;
   - déterminer les formules, équations et constantes ;

Analyse

Objectifs

1. Comprendre et clarifier le problème : ambiguïté, complétude, cohérence ?
2. Déterminer les données d'entrée et de sortie du problème.
3. Déterminer grossièrement le traitement à faire :
   - fixer des hypothèses ;
   - déterminer les formules, équations et constantes ;
   - déterminer les cas d'exception et d'erreurs.

Analyse

Objectifs

1. Comprendre et clarifier le problème : ambiguïté, complétude, cohérence ?
2. Déterminer les données d'entrée et de sortie du problème.
3. Déterminer grossièrement le traitement à faire :
   - fixer des hypothèses ;
   - déterminer les formules, équations et constantes ;
   - déterminer les cas d'exception et d'erreurs.
4. Déterminer le comportement du système.

Analyse

Méthodologie

- Comprendre le domaine du problème ; (données)

Analyse

Méthodologie

- Comprendre le domaine du problème ; (données)
- Définir les fonctions du logiciel ; (fonctionnelle)

Analyse

Méthodologie

- Comprendre le domaine du problème ; (données)
- Définir les fonctions du logiciel ; (fonctionnelle)
- Représenter le comportement du logiciel ; (comportementale)

Analyse

Méthodologie

- Comprendre le domaine du problème ; (données)
- Définir les fonctions du logiciel ; (fonctionnelle)
- Représenter le comportement du logiciel ; (comportementale)
- Hierarchiser le modèle d'analyse contenant l'information, la fonction et le comportement du logiciel ;

Analyse

Méthodologie

- Comprendre le domaine du problème ; (données)
- Définir les fonctions du logiciel ; (fonctionnelle)
- Représenter le comportement du logiciel ; (comportementale)
- Hierarchiser le modèle d'analyse contenant l'information, la fonction et le comportement du logiciel ;
- Aller de l'information essentielle vers les détails du système.

Analyse

Une bonne analyse

1. Peut servir de documentation à la solution implémentée

Analyse

Une bonne analyse

1. Peut servir de documentation à la solution implémentée
2. Détermine :

Analyse

Une bonne analyse

1. Peut servir de documentation à la solution implémentée
2. Détermine :
   - le type des données en entrée et en sortie ;

Analyse

Une bonne analyse

1. Peut servir de documentation à la solution implémentée
2. Détermine :
   - le type des données en entrée et en sortie ;
   - comment les données seront utilisées ;

Analyse

Une bonne analyse

1. Peut servir de documentation à la solution implémentée
2. Détermine :
   - le type des données en entrée et en sortie ;
   - comment les données seront utilisées ;
   - les fonctions les modifiant ;

Analyse

Une bonne analyse

1. Peut servir de documentation à la solution implémentée
2. Détermine :
   - le type des données en entrée et en sortie ;
   - comment les données seront utilisées ;
   - les fonctions les modifiant ;
   - les constantes utilisées ;

Analyse

Une bonne analyse

1. Peut servir de documentation à la solution implémentée
2. Détermine :
   - le type des données en entrée et en sortie ;
   - comment les données seront utilisées ;
   - les fonctions les modifiant ;
   - les constantes utilisées ;
   - l'interaction avec l'environnement.

Conception descendante

Conception

Définition

Étape qui consite à élaborer, à partir de l'analyse, une solution au problème (algorithme)

Conception

Définition

Étape qui consite à élaborer, à partir de l'analyse, une solution au problème (algorithme)

Stratégie générale

Conception

Définition

Étape qui consite à élaborer, à partir de l'analyse, une solution au problème (algorithme)

Stratégie générale

- Décomposer le problème (tâche) en sous-problèmes (sous-tâches) et identifier les problèmes pertinents (modules)

Conception

Définition

Étape qui consite à élaborer, à partir de l'analyse, une solution au problème (algorithme)

Stratégie générale

- Décomposer le problème (tâche) en sous-problèmes (sous-tâches) et identifier les problèmes pertinents (modules)
- Deux façons de décomposer

Conception

Définition

Étape qui consite à élaborer, à partir de l'analyse, une solution au problème (algorithme)

Stratégie générale

- Décomposer le problème (tâche) en sous-problèmes (sous-tâches) et identifier les problèmes pertinents (modules)
- Deux façons de décomposer
  - selon le traitement (conception fonctionnelle)

Conception

Définition

Étape qui consite à élaborer, à partir de l'analyse, une solution au problème (algorithme)

Stratégie générale

- Décomposer le problème (tâche) en sous-problèmes (sous-tâches) et identifier les problèmes pertinents (modules)
- Deux façons de décomposer
  - selon le traitement (conception fonctionnelle)
  - selon les données à manipuler (conception objet)

Conception

La conception peut-être

Descendante (*top-down*)

- décomposer le problème initial en sous-problèmes et
- décomposer chaque sous-problème en de nouveaux sous-problèmes
- . . . jusqu'à des problèmes primitifs.
- Approche « diviser pour mieux régner »

Conception

La conception peut-être

Ascendante *(bottom-up)*

- construire à partir d'opérations primitives des opérations plus complexes
- . . . jusqu'à une opération globale qui résout le problème initial.

Décomposition fonctionnelle descendante

Description

Décomposition fonctionnelle descendante

Description

- Décomposition selon le traitement

Décomposition fonctionnelle descendante

Description

- Décomposition selon le traitement
- La décomposition fonctionnelle descendante est basée sur la stratégie de résolution de problèmes *diviser pour régner*.

Décomposition fonctionnelle descendante

Description

- Décomposition selon le traitement
- La décomposition fonctionnelle descendante est basée sur la stratégie de résolution de problèmes *diviser pour régner*.
- Chaque étape de la décomposition est suivie par l'élicitation de sous-problèmes.

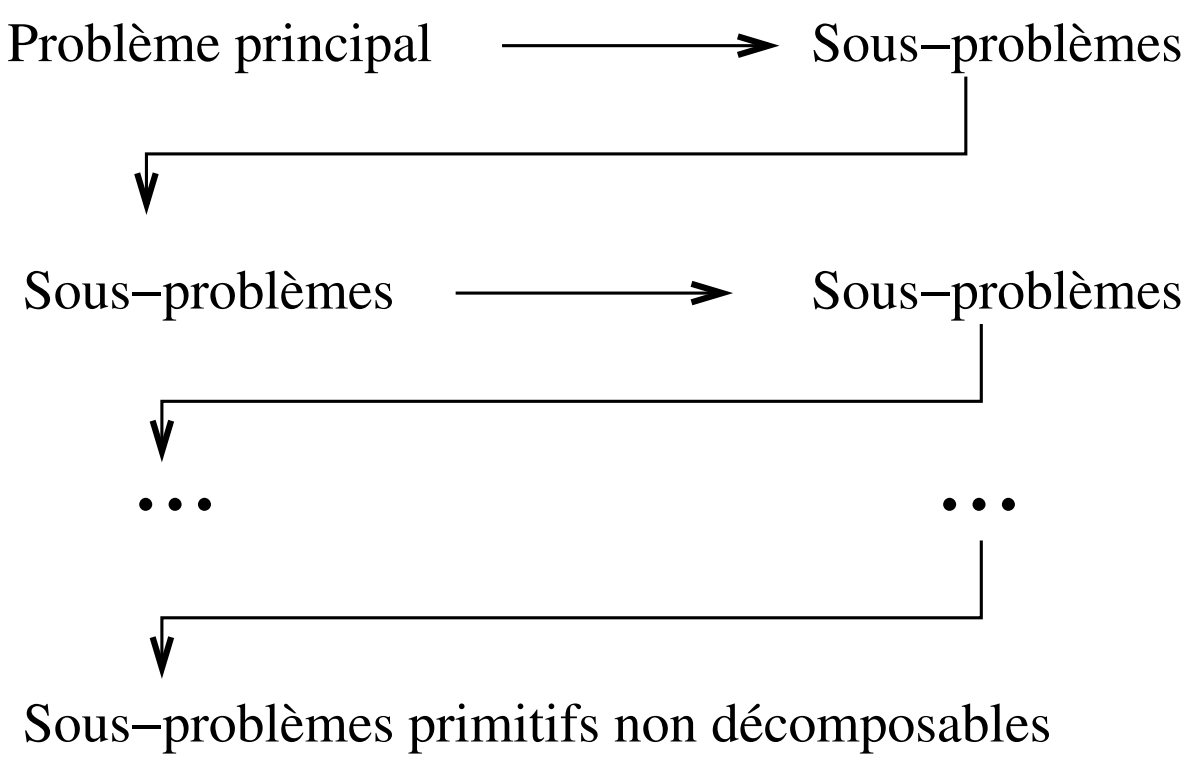
Décomposition fonctionnelle descendante

Description

- Décomposition selon le traitement
- La décomposition fonctionnelle descendante est basée sur la stratégie de résolution de problèmes *diviser pour régner*.
- Chaque étape de la décomposition est suivie par l'élicitation de sous-problèmes.
- On obtient des unités de traitement relativement indépendantes (modules)

Décomposition fonctionnelle descendante

Tâche répétitive (approche itérative) :



Décomposition fonctionnelle descendante

Techniques

Les techniques utilisées dans cette méthode sont :

Décomposition fonctionnelle descendante

Techniques

Les techniques utilisées dans cette méthode sont :

- Le *raffinement successif*, c'est-à-dire que chaque étape de la décomposition fait intervenir une seule décision de conception.

Décomposition fonctionnelle descendante

Techniques

Les techniques utilisées dans cette méthode sont :

- Le *raffinement successif*, c'est-à-dire que chaque étape de la décomposition fait intervenir une seule décision de conception.
- Le *masquage d'information* (critère de décomposition), c'est-à-dire que les décisions propres à un module sont cachées aux autres modules. Les données et opérations accessibles aux autres modules le sont seulement à travers une interface bien définie. Les données et les opérations non utiles aux autres modules sont inaccessibles.

Les modules

Un module

Les modules

Un module

- est une boîte exécutant une tâche précise

Les modules

Un module

- est une boîte exécutant une tâche précise
- essaye de généraliser le processus si possible

Les modules

Un module

- est une boîte exécutant une tâche précise
- essaye de généraliser le processus si possible
- n'attaque qu'un problème à la fois

Les modules

Un module

- est une boîte exécutant une tâche précise
- essaye de généraliser le processus si possible
- n'attaque qu'un problème à la fois
- décrit précisément les étapes de résolutions du problème

Les modules

Un module

- est une boîte exécutant une tâche précise
- essaye de généraliser le processus si possible
- n'attaque qu'un problème à la fois
- décrit précisément les étapes de résolutions du problème
- . . . en utilisant d'autres modules

Les modules

Un module

- est une boîte exécutant une tâche précise
- essaye de généraliser le processus si possible
- n'attaque qu'un problème à la fois
- décrit précisément les étapes de résolutions du problème
- . . . en utilisant d'autres modules
- doit pouvoir communiquer avec d'autres modules

Les modules

Un module

- est une boîte exécutant une tâche précise
- essaye de généraliser le processus si possible
- n'attaque qu'un problème à la fois
- décrit précisément les étapes de résolutions du problème
- . . . en utilisant d'autres modules
- doit pouvoir communiquer avec d'autres modules
- . . . via des interfaces précises

Les modules

Un module

- est une boîte exécutant une tâche précise
- essaye de généraliser le processus si possible
- n'attaque qu'un problème à la fois
- décrit précisément les étapes de résolutions du problème
- . . . en utilisant d'autres modules
- doit pouvoir communiquer avec d'autres modules
- . . . via des interfaces précises
- Notion d'exactitude, de robustesse et de fiabilité (plus tard)

Conception des modules

L'interface

Conception des modules

L'interface

1. Les questions de l'analyse
   $\Longrightarrow$ Interface du module

Conception des modules

L'interface

1. Les questions de l'analyse
   $\Longrightarrow$ Interface du module

L'algorithme

Conception des modules

L'interface

1. Les questions de l'analyse
   $\Longrightarrow$ Interface du module

L'algorithme

1. Quelles sont les grandes étapes à réaliser ?
   $\Longrightarrow$ Premier niveau de l'algorithme

Conception des modules

L'interface

1. Les questions de l'analyse
   $\Longrightarrow$ Interface du module

L'algorithme

1. Quelles sont les grandes étapes à réaliser ?
   $\Longrightarrow$ Premier niveau de l'algorithme
2. Quelles sont les étapes qui devraient être résolues à part ?
   $\Longrightarrow$ Identification des modules

Conception des modules

L'interface

1. Les questions de l'analyse
   $\Longrightarrow$ Interface du module

L'algorithme

1. Quelles sont les grandes étapes à réaliser ?
   $\Longrightarrow$ Premier niveau de l'algorithme
2. Quelles sont les étapes qui devraient être résolues à part ?
   $\Longrightarrow$ Identification des modules
3. Pour chaque étape qui ne sont pas des modules, on se repose les questions de la conception.
   $\Longrightarrow$ Niveau subséquent de l'algorithme

Conception des modules

Pour chaque module

Conception des modules

Pour chaque module

1. On se repose les questions pertinentes de l'analyse
   $\Longrightarrow$ Analyse du module

Conception des modules

Pour chaque module

1. On se repose les questions pertinentes de l'analyse
   $\Longrightarrow$ Analyse du module
2. On se repose les questions pertinentes de la conception
   $\Longrightarrow$ Identification des modules
   $\Longrightarrow$ Algorithme

Conception — Outils

Les deux outils utilisés

Conception — Outils

Les deux outils utilisés

- Diagrammes structurels
  Permettent de visualiser les liens entre un sous-problème et les sous-problèmes générés par la décomposition.

Conception — Outils

Les deux outils utilisés

- Diagrammes structurels
  Permettent de visualiser les liens entre un sous-problème et les sous-problèmes générés par la décomposition.
- Pseudo-code ou organigramme
  Permet d'illustrer le traitement fait par une tâche particulière (algorithme).

Conception descendante

UNIVERSITÉ DE SHERBROOKE

Réalisation

- Choix du langage (en théorie)

Réalisation

- Choix du langage (en théorie)
- Type de programmation

Réalisation

- Choix du langage (en théorie)
- Type de programmation
  - Programmation modulaire

Réalisation

- Choix du langage (en théorie)
- Type de programmation
  - Programmation modulaire
  - Programmation structurée

Réalisation

- Choix du langage (en théorie)
- Type de programmation
  - Programmation modulaire
  - Programmation structurée
  - Programmation typée

Réalisation

- Choix du langage (en théorie)
- Type de programmation
  - Programmation modulaire
  - Programmation structurée
  - Programmation typée
  - Programmation impérative

Réalisation

- Choix du langage (en théorie)
- Type de programmation
  - Programmation modulaire
  - Programmation structurée
  - Programmation typée
  - Programmation impérative
  - Programmation procédurale (dans la majeure partie du cours)

Réalisation

- Choix du langage (en théorie)
- Type de programmation
  - Programmation modulaire
  - Programmation structurée
  - Programmation typée
  - Programmation impérative
  - Programmation procédurale (dans la majeure partie du cours)
  - Programmation orientée-objet (à la fin du cours)

Réalisation — Programmation modulaire

Cette méthode divise un système complexe en plusieurs parties, appelées modules, qui :

- sont utilisées pour généraliser les tâches, ajouter un niveau d'abstraction et éviter les multiples instances d'une même fonctionnalité
- peuvent être développées simultanément par plusieurs programmeurs.

Réalisation — Programmation modulaire

Les principes de base de la programmation modulaire sont les suivants :

- Chaque module implémente une seule tâche indépendante des autres.

Réalisation — Programmation modulaire

Les principes de base de la programmation modulaire sont les suivants :

- Chaque module implémente une seule tâche indépendante des autres.
- Chaque module a un seul point d'entrée et un seul point de sortie.

Réalisation — Programmation modulaire

Les principes de base de la programmation modulaire sont les suivants :

- Chaque module implémente une seule tâche indépendante des autres.
- Chaque module a un seul point d'entrée et un seul point de sortie.
- La taille d'un module est petite.

Réalisation — Programmation modulaire

Les principes de base de la programmation modulaire sont les suivants :

- Chaque module implémente une seule tâche indépendante des autres.
- Chaque module a un seul point d'entrée et un seul point de sortie.
- La taille d'un module est petite.
- Chaque module est conçu pour être codé et testé séparément.

Réalisation — Programmation modulaire

Les principes de base de la programmation modulaire sont les suivants :

- Chaque module implémente une seule tâche indépendante des autres.
- Chaque module a un seul point d'entrée et un seul point de sortie.
- La taille d'un module est petite.
- Chaque module est conçu pour être codé et testé séparément.
- Le système est composé de modules (intégration des modules et essais intégrés).

Réalisation — Programmation structurée

Définition

imposer au programmeur l'utilisation de certains types d'énoncés.

Réalisation — Programmation structurée

Définition

imposer au programmeur l'utilisation de certains types d'énoncés.

Objectif

Réalisation — Programmation structurée

Définition

imposer au programmeur l'utilisation de certains types d'énoncés.

Objectif

- simplifier la logique du programme

Réalisation — Programmation structurée

Définition

imposer au programmeur l'utilisation de certains types d'énoncés.

Objectif

- simplifier la logique du programme
- établir que le programme construit satisfait la spécification

Réalisation — Programmation structurée

La séquence

exécuter une suite d'énoncés

Réalisation — Programmation structurée

La séquence

exécuter une suite d'énoncés

La sélection

exécuter un énoncé choisi selon une condition donnée

Réalisation — Programmation structurée

La séquence

exécuter une suite d'énoncés

La sélection

exécuter un énoncé choisi selon une condition donnée

L'itération

exécuter de façon répétitive un énoncé

Réalisation

De l'analyse/conception à la réalisation

Réalisation

De l'analyse/conception à la réalisation

- les étapes précédantes servent de point de départ (documentation)

Réalisation

De l'analyse/conception à la réalisation

- les étapes précédantes servent de point de départ (documentation)
- à partie de l'analyse

Réalisation

De l'analyse/conception à la réalisation

- les étapes précédantes servent de point de départ (documentation)
- à partie de l'analyse
  - description de nos entrées et sorties

Réalisation

De l'analyse/conception à la réalisation

- les étapes précédantes servent de point de départ (documentation)
- à partie de l'analyse
  - description de nos entrées et sorties
- à partir de la conception

Réalisation

De l'analyse/conception à la réalisation

- les étapes précédantes servent de point de départ (documentation)
- à partie de l'analyse
  - description de nos entrées et sorties
- à partir de la conception
  - chaque module décrit une fonction (module principal $\longrightarrow$ fonction `main`)

Réalisation

De l'analyse/conception à la réalisation

- les étapes précédantes servent de point de départ (documentation)
- à partie de l'analyse
  - description de nos entrées et sorties
- à partir de la conception
  - chaque module décrit une fonction (module principal $\longrightarrow$ fonction `main`)
  - l'algorithme décrit les étapes et les appels

Réalisation

De l'analyse/conception à la réalisation

- les étapes précédantes servent de point de départ (documentation)
- à partie de l'analyse
  - description de nos entrées et sorties
- à partir de la conception
  - chaque module décrit une fonction (module principal $\longrightarrow$ fonction `main`)
  - l'algorithme décrit les étapes et les appels

Bonne habitude

commencer par écrire l'algorithme en commentaire

Exemple

Étape 1

```
// 1. Lire la longueur d'un cote au clavier
// 2. Calculer le volume du cube
// 3. Affiche le volume du cube
```

Étape 2

```
// 1. Lire la longueur d'un cote au clavier
cin >> longueur_cote ;
// 2. Calculer le volume du cube
volume_cube = cube(longueur_cote);
// 3. Affiche le volume du cube
cout << volume_cube ;
```

Exemple : moyenne

```
/** ************************************
   \file moyenne.cpp

   \brief Ce programme calcule a moyenne
          de trois nombres entiers.
   <b>Entrees:</b>
       \li (\c calvier) 3 nombres (\c entiers)
   <b>Sorties:</b>
      \li (\c ecran) moyenne (\c entier)
************************************** */
int main()
{
  // constantes et variables locales
  int val1, val2, val3.  // entree : trois entiers
      moyenne,           // sortie :la moyenne

  // 1.  Obtenir les trois valeurs
  // 2. calculer la moyenne
  // 3. Imprimer le resultat
} // fin main()
```

Exemple : moyenne

```
/**  ...    */
#include <iostream>
using namespace std;
int main()
{
  // constantes et variables locales
  const int nbValeur = 3;
  int val1, val2, val3.  // entree : trois entiers
      moyenne,           // sortie :la moyenne

  // 1.  Obtenir les trois valeurs
  cout << "Entrez les trois entiers : ";
  cin >> val1 >> val2 >> val3;

  // 2. calculer la moyenne
  moyenne = (val1 + val2 + val3) / nbValeur ;

  // 3. Imprimer le resultat
  cout << "La moyenne est  " << moyenne << endl;
  return 0;
}  // fin main()
```

Réalisation — Documentation

On doit documenter notre code grâce à des commentaires qui décrivent :

- Ce que fait le programme ;
- Les entrées et les sorties du programme ;
- l'utilité des variables et constantes ;
- Chaque étape importante de l'algorithme ;
- L'utilité de chaque groupe d'instructions.

Qualités d'une documentation

- Concise ;
- Précise ;
- Complète ;
- Simple à maintenir ;
- Uniforme.

Qualités d'une documentation

- Concise ;
- Précise ;
- Complète ;
- Simple à maintenir ;
- Uniforme.

La génération automatique de la documentation se fera en utilisant l'outil `doxygen`

Documentation `doxygen`

Commentaires `doxygen`

- Commentaire sur plusieurs lignes

  `/** commentaire */`
- Commentaire jusqu'à la fin de ligne

  `/// commentaire`

Commentaires classiques

- Commentaire sur plusieurs lignes

  `/* commentaire */`
- Commentaire jusqu'à la fin de ligne

  `// commentaire`

Tests et mise au point du programme

- Les résultats doivent être précis.
- Les résultats doivent être conformes à la spécification du problème.
- Le programme doit fonctionner dans tous les cas (preuve).
- Le programme s'exécute dans les délais prescrits.
- Le programme s'exécute dans l'espace mémoire prescrit.

Tests

Quatre types de tests déjà vus :

- Tests unitaires

Tests

Quatre types de tests déjà vus :

- Tests unitaires
- Tests d'intégration

Tests

Quatre types de tests déjà vus :

- Tests unitaires
- Tests d'intégration
- Tests système

Tests

Quatre types de tests déjà vus :

- Tests unitaires
- Tests d'intégration
- Tests système
- Tests d'acceptation

Tests

Quatre types de tests déjà vus :

- Tests unitaires
- Tests d'intégration
- Tests système
- Tests d'acceptation

Nous nous concentrons sur les deux premières catégories.

Tests unitaires

- Vérifier l'implémentation de la conception (fonction, module, etc.) ;
- Assurer partiellement la logique du programme (complétude et correction) ;
- Principe : isoler l'élément à tester pour qu'un autre élément ne fasse pas d'interférence.

Tests d'intégration

- Vérifier que les éléments interagissent correctement ensembles ;
- S'assurer que les objectifs de la conception sont atteints ;
- Combiner et tester les combinaisons d'éléments jusqu'à intégration du système.

Stratégie de tests

- Tests de type boîte noire
  - Tester des données normales
  - Tester des données limites
  - Tester des combinaisons de données
- Tests de type boîte blanche
  - Tester tous les chemins
  - Tester des cas particuliers (probabilité)

Mise au point — Outils

- Traces (commandes d'impression dans le programme)

Mise au point — Outils

- Traces (commandes d'impression dans le programme)
  - plantage : lorsque le programme produit peu d'affichage, utilisation de `cout` pour voir le moment où le programme plante

Mise au point — Outils

- Traces (commandes d'impression dans le programme)
  - plantage : lorsque le programme produit peu d'affichage, utilisation de `cout` pour voir le moment où le programme plante
  - erreur de logique : ajout de `cout` aux endroits importants pour afficher le contenu de variables clés

Mise au point — Outils

- Traces (commandes d'impression dans le programme)
  - plantage : lorsque le programme produit peu d'affichage, utilisation de `cout` pour voir le moment où le programme plante
  - erreur de logique : ajout de `cout` aux endroits importants pour afficher le contenu de variables clés
- Outils interactifs pour la mise au point : `ddd`

Mise au point — Outils

- Traces (commandes d'impression dans le programme)
  - plantage : lorsque le programme produit peu d'affichage, utilisation de `cout` pour voir le moment où le programme plante
  - erreur de logique : ajout de `cout` aux endroits importants pour afficher le contenu de variables clés
- Outils interactifs pour la mise au point : `ddd`
- IDE (Visual Studio, Code : :Blocks, ...)

Conception descendante

UNIVERSITÉ DE SHERBROOKE

Exemples d'erreurs

- Réacteurs nucléaires (1985) - Cinq réacteurs fermés car un programme testant leur résistance aux tremblements de terre utilisait une somme arithmétique plutôt que la somme des valeurs absolues.
- Test laser sur la navette spatiale (1985) - Altitude en milles nautiques plutôt qu'en pieds.
- Gemini V - Manque son site d'atterrissage par 100 milles (ont oublié de considérer le mouvement de la Terre autour du soleil dans le calcul).

Conclusion

- Tom Cargill (la règle du 90/90)
  "The first 90% of the code accounts for the first 90% of the development time. The remaining 10% of the code accounts for the other 90% of the development time."
- C.A.R. Hoare
  "There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies."
- Edsger W. Dijkstra
  Testing can only prove the presence of bugs, not their absence.