Logiciel Libre
Cours 3 — Fondements: Génie Logiciel

Stefano Zacchiroli
zack@pps.univ-paris-diderot.fr

Laboratoire PPS, Université Paris Diderot

2013–2014

Outline

1. Algorithmique
2. Génie logiciel
3. Réutilisation, combinaison, coopération

Rappel du cours 1

- Code source : programme écrit par un programmeur dans un langage de programmation (C, Pascal, OCaml, ...).
  Nécessaire pour comprendre ou modifier le programme.
- Binaire, Exécutable : programme dans le langage de la machine (dépend de l'architecture : Pentium, AMD64, ...).
  (Presque) incompréhensible pour des lecteurs humains.
  Nécessaire pour exécuter le programme.
- Compilateur : logiciel qui traduit code source en binaire.

Contenu Cours 2

- Qu'est-ce que c'est l'algorithmique
- Principes du génie logiciel
- Réutilisation, combinaison, et coopération de logiciels

Outline

1. Algorithmique
2. Génie logiciel
3. Réutilisation, combinaison, coopération

L'idée algorithmique

Problème :

Trier une liste d'entiers différents, par exemple

$$1, 17, 5, 12$$

On veut la mettre en ordre ascendant :

$$1, 5, 12, 17$$

Comment résoudre ce problème avec un programme ?

Le programme doit

- prendre en entrée (par exemple lire d'un fichier) une liste quelconque d'entiers ;
- sortir (par exemple, écrire sur un autre fichier) la liste triée.

Algorithme

- Décrire comment organiser le travail pour résoudre une tache
- Utilise les éléments communs de la programmation
  - instructions primitives, boucles, etc.
- Indépendant d'un langage de programmation concret
- Peut laisser ouvert certains détails du programme
  - p.ex. : les structures de données ou l'ordre d'exécution des instructions quand il ne sont pas importants
- Intuition : "la recette derrière le programme"
- D'un algorithme au programme : codage / implémentation

Exemple d'un algorithme

- Première solution du problème de tri : tri à bulles (*bubble sort*).
- Tant qu'il y a deux éléments successives dans la liste qui ne sont pas dans le bon ordre les inverser.

Example (tri à bulles)

Étapes :

0. 1, 17, 12, 5

Exemple d'un algorithme

- Première solution du problème de tri : tri à bulles (*bubble sort*).
- Tant qu'il y a deux éléments successives dans la liste qui ne sont pas dans le bon ordre les inverser.

Example (tri à bulles)

Étapes :

0. 1, 17, 12, 5

Exemple d'un algorithme

- Première solution du problème de tri : tri à bulles (*bubble sort*).
- Tant qu'il y a deux éléments successives dans la liste qui ne sont pas dans le bon ordre les inverser.

Example (tri à bulles)

Étapes :

0. 1, 17, 12, 5
1. 1, 12, 17, 5

Exemple d'un algorithme

- Première solution du problème de tri : tri à bulles (*bubble sort*).
- Tant qu'il y a deux éléments successives dans la liste qui ne sont pas dans le bon ordre les inverser.

Example (tri à bulles)

Étapes :

0. 1, 17, 12, 5
1. 1, 12, 17, 5
2. 1, 12, 5, 17

Exemple d'un algorithme

- Première solution du problème de tri : tri à bulles (*bubble sort*).
- Tant qu'il y a deux éléments successives dans la liste qui ne sont pas dans le bon ordre les inverser.

Example (tri à bulles)

Étapes :

0. 1, 17, 12, 5
1. 1, 12, 17, 5
2. 1, 12, 5, 17
3. 1, 5, 12, 17

Les questions que se posent les informaticiens . . .

. . . quand ils voient cet algorithme :

Terminaison Est-ce que l'exécution s'arrête pour n'importe quelle liste d'entrée ?

Correction Est-ce que le résultat est bien trié quand l'exécution s'arrête ?

Non-déterminisme Il y a des choix à faire (quand il y a plusieurs paires en désordre), est-ce que le choix pris a une conséquence pour le résultat ?

Complexité Est-ce "efficace" ?

La stratégie *diviser pour régner*

latin : *divide et impera*

- Diviser un problème en plusieurs problèmes :
  - de la même nature,
  - qui sont chacun plus petit que le problème de départ.
- Résoudre chacun des petits problèmes.
- Combiner des solutions des petits problèmes pour obtenir la solution du problème du départ.
- Dire comment résoudre un problème trivial.

La stratégie *diviser pour régner*

latin : *divide et impera*

- Diviser un problème en plusieurs problèmes :
  - de la même nature,
  - qui sont chacun plus petit que le problème de départ.
- Résoudre chacun des petits problèmes.
- Combiner des solutions des petits problèmes pour obtenir la solution du problème du départ.
- Dire comment résoudre un problème trivial.

Il s'agit d'un algorithme récursif.

Un peu plus concrètement : trier

Recette pour trier :

1. Diviser : on choisit une valeur de *pivot* p et divise une liste donnée l dans deux parties :
   - la liste l_1 des valeurs $< p$
   - la liste l_2 des valeurs $> p$
2. Trier l_1 donne r_1, trier l_2 donne r_2.
3. Combiner les résultats : La liste consiste en r_1 d'abord, puis p, puis r_2 (pourquoi ?)
4. Une liste triviale (longueur 0 ou 1) est déjà triée !

Comment est-ce que ça marche ?

- Ici : noir : reste à trier, red : déjà trié.
- Lors de la division en sous-problèmes on écrit l'élément pivot directement entre les deux parties, ça facilite la combinaison des résultats !

10 3 1 5 17 12 55

Comment est-ce que ça marche ?

- Ici : noir : reste à trier, red : déjà trié.
- Lors de la division en sous-problèmes on écrit l'élément pivot directement entre les deux parties, ça facilite la combinaison des résultats !

10 3 1 5 17 12 55
3 1 5 10 17 12 55

Comment est-ce que ça marche ?

- Ici : noir : reste à trier, red : déjà trié.
- Lors de la division en sous-problèmes on écrit l'élément pivot directement entre les deux parties, ça facilite la combinaison des résultats !

10	3	1	5	17	12	55
3	1	5	10	17	12	55
1	3	5	10	12	17	55

Comment est-ce que ça marche ?

- Ici : noir : reste à trier, red : déjà trié.
- Lors de la division en sous-problèmes on écrit l'élément pivot directement entre les deux parties, ça facilite la combinaison des résultats !

10	3	1	5	17	12	55
3	1	5	10	17	12	55
1	3	5	10	12	17	55
1	3	5	10	12	17	55

Discussion de cet algorithme

- Terminaison ? est-ce qu'on peut diviser infiniment ?

Discussion de cet algorithme

- Terminaison ? est-ce qu'on peut diviser infiniment ?
- Correction ? Est-ce que le résultat est forcement trié ?

Discussion de cet algorithme

- Terminaison ? est-ce qu'on peut diviser infiniment ?
- Correction ? Est-ce que le résultat est forcement trié ?
- Conséquence du choix de l'élément pivot ?

Discussion de cet algorithme

- Terminaison ? est-ce qu'on peut diviser infiniment ?
- Correction ? Est-ce que le résultat est forcement trié ?
- Conséquence du choix de l'élément pivot ?
- Efficacité ?

Discussion de cet algorithme

- Terminaison ? est-ce qu'on peut diviser infiniment ?
- Correction ? Est-ce que le résultat est forcement trié ?
- Conséquence du choix de l'élément pivot ?
- Efficacité ?

Il s'agit ici de quicksort, un des algorithmes de trie les plus rapide (en moyenne), dû à *Tony Hoare*.

Bubble sort vs quicksort

Video : visualization of quicksort
http://www.youtube.com/watch?v=aXXWXz5rF64
Credits : YouTube user udiprod

Outline

1 Algorithmique

2 Génie logiciel

3 Réutilisation, combinaison, coopération

Problèmes du génie logiciel

- Comment s'organiser pour écrire un (grand) logiciel.
  Grand : $>$ 1.000.000 lignes de code
- Organisation du travail parmi un grand nombre de programmeurs travaillant sur un projet (éventuellement à des endroits différent)
- Un logiciel n'est pas écrit une fois pour l'éternité, il évolue toujours
  - ajout de nouvelles fonctionnalités, erreurs, adaptation à des nouveaux environnements, ...

C’est un problème très difficile

- Une citation provocante :

  > *Software Engineering is the part of computer science that is too difficult for computer scientists.*

- concerne le processus de création d’un logiciel
- n’est lui même pas toujours complètement formel
- peut quand même profiter d’un support par des outils informatiques

Très grands projets informatiques

Exemple : Windows Vista

- Début du développement : mai 2001
- Date de livraison initialement prévue : fin 2003
- Date réelle de livraison : janvier 2007
- $>$ 50 millions lignes de code
- plusieurs milliers de programmeurs

Le modèle traditionnel

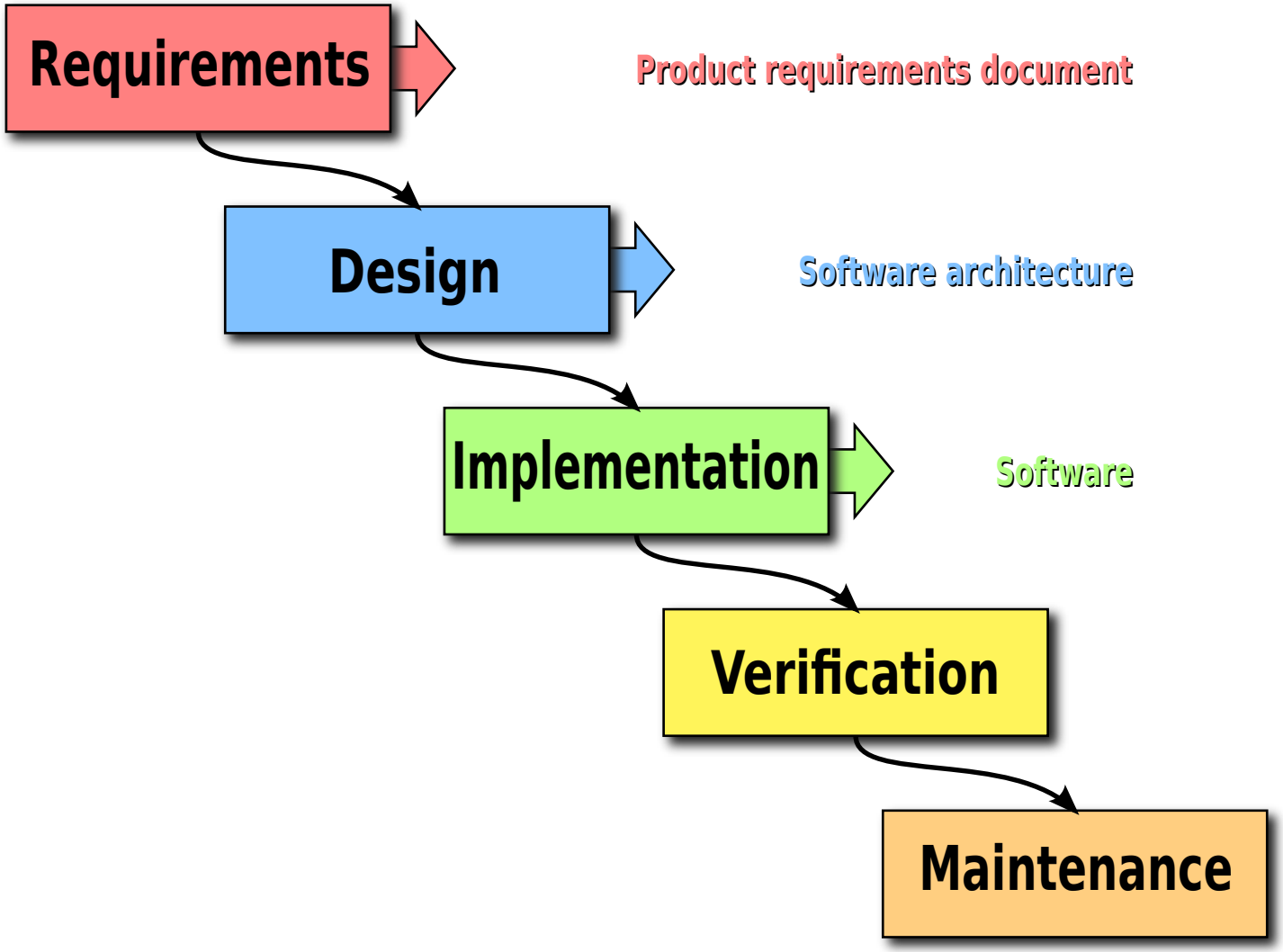
- Comprendre le problème, établir un cahier de charges, établir une spécification du logiciel
  - La nature de la spécification peut varier de très informelle (écrite en langue naturelle) à très formelle (écrite dans un système mathématique).
- Conception de la structure du logiciel, découpage en unités.
- Codage (ne prend qu'une petite partie du temps total du projet).
- Tester :
  - Peut servir à détecter des problèmes, mais jamais à assurer l'absence de problèmes car on ne peut jamais couvrir tous les cas d'utilisation.
  - Il peut être utile de faire tester le programme par une équipe indépendante des programmeurs.

Le modèle traditionnel (cont.)

- Documenter :
  - Documentation destinée à des utilisateurs
  - Documentation destinée à des administrateurs de système (p.ex. installation, mise à jour, . . .)
  - Documentation destinée à des programmeurs qui veulent modifier le programme (éventuellement pas publiée)
  - Il peut être utile de documenter avant de coder, ou même avant la conception du système.
- Déploiement
- Maintenance
  - p.ex. mise à jour, *porting*, failles de sécurité, changement des besoins, . . .

Modèle Cascade

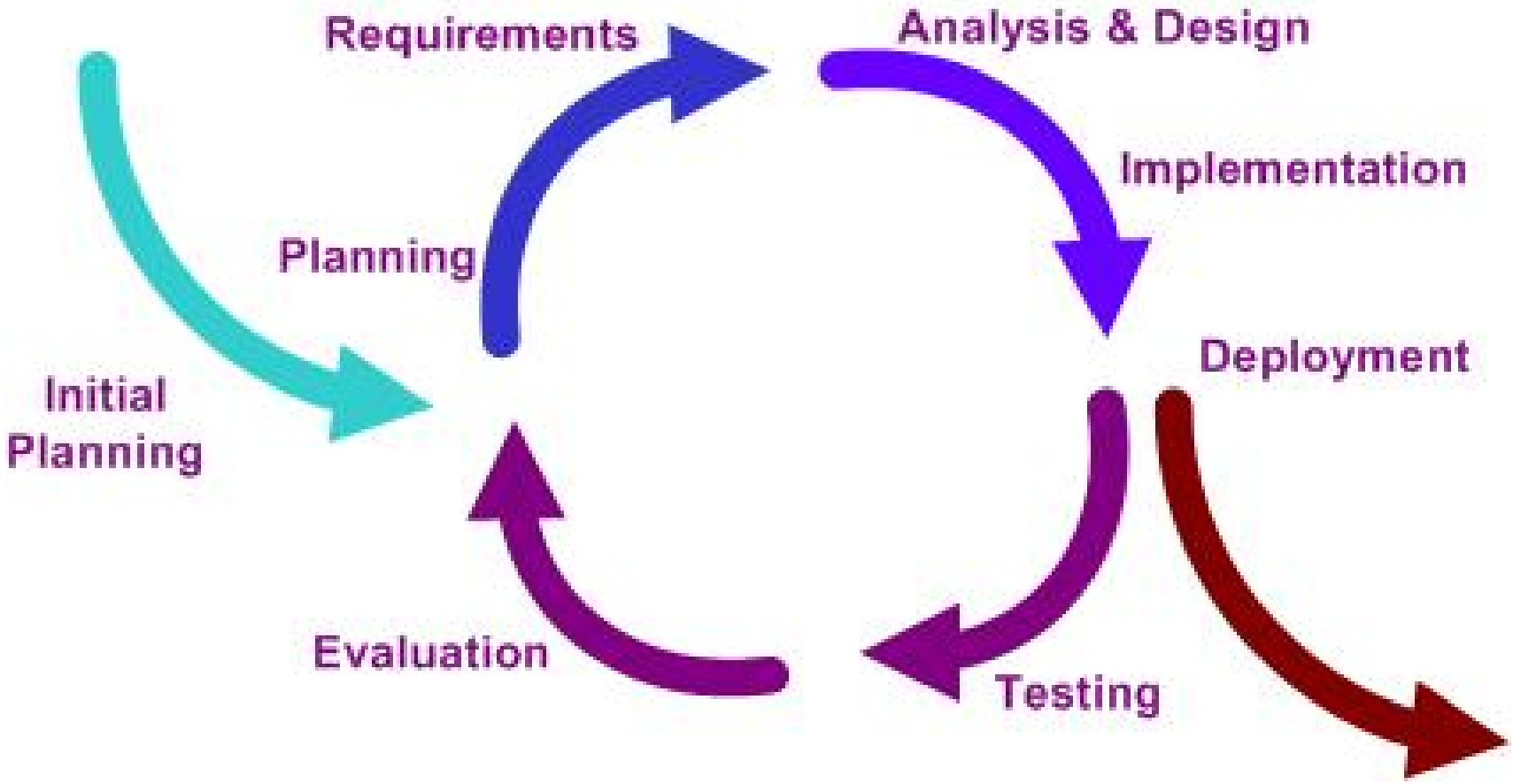
Angl. : *waterfall model*



https://en.wikipedia.org/wiki/File:Waterfall_model.svg

Modèle itératif

Modèle plus flexible avec retour en arrière.



https://en.wikipedia.org/wiki/File:Iterative_development_model_V2.jpg

Organiser un programme en unités

- Nécessité pour les programmes de grande taille.
  Bonne idée en tout cas.
- Des personnes différentes peuvent travailler sur les parties différentes (même si elles ne sont pas indépendantes).
- Compilation séparée.
- Analyse descendante : Organiser un programme en une hiérarchie d'unités, étapes de raffinement successif (*separation of concerns*).
- Les langages de programmation différents offrent des moyens différents pour organiser les unités
  - modules, classes, *namespaces*, ...

Outline

1. Algorithmique
2. Génie logiciel
3. Réutilisation, combinaison, coopération

Réutilisation de code (par copier-coller)

Utiliser une partie d'un programme existant pour la rédaction d'un nouveau programme.

Des nains sur des épaules de géants
— Bernard de Chartres, XII siècle

- Nécessite le code source du programme existant.
- Nécessite le même langage de programmation.
- Résultat : un nouveau programme *monolithique*.
- Problème de licence si on est pas propriétaire du programme existant (voir plus tard le chapitre sur les licences de logiciels).
- Seulement possible si le programme existant est bien structuré, et permet d'extraire facilement un morceau avec une certaine fonctionnalité.

Réutilisation de code (par copier-coller) — exemple

- On dispose d'un programme écrit (par exemple en C) pour trier une liste d'entiers.
- On souhaite écrire un programme qui lit un fichier de données numériques et qui produit un tableau formaté contenant les données dans un ordre ascendant, et en plus leur moyenne.
- Le programmeur lance son éditeur de texte pour rédiger le programme, puis :
  - Écrire le code pour lire les données d'un fichier
  - Inclure textuellement (copier-coller) le programme de trie, éventuellement supprimer les parties inutiles
  - Mettre le code pour calculer la moyenne, et formater le résultat.

Réutilisation de code (par copier-coller) (cont.)

- Souvent utilisé en pratique.
- Inconvénients :

?

Réutilisation de code (par copier-coller) (cont.)

- Souvent utilisé en pratique.
- Inconvénients :
  - Code doublé : les mêmes morceaux de code existent en beaucoup de copies dans plein de programmes.
  - Les programmes qui réutilisent du code ne profitent pas des améliorations du code de trie (meilleure performance, réparation d'erreurs).
  - Ne marche que pour le même langage de programmation.
  - Pas un bon principe de structuration de code.

Combinaison au niveau binaire

- Bibliothèque : logiciel qui n'est pas autonome, mais qui fournit des fonctionnalités qui peuvent être utilisées par des autres programmes.
- Un programme peut utiliser les fonctionnalités fournies par une bibliothèque à travers des appels de procédures qui sont définies dans le code de la bibliothèque.
- On essaie de fournir des bibliothèques pour des problèmes récurrents.
- La fonctionnalité fournie par une bibliothèque doit être aussi générique que possible.

Bibliothèques et langages de programmation

- Il n'est *a priori* plus nécessaire que le programme qui utilise la bibliothèque, et le code source de la bibliothèque, soient écrits dans le même langage de programmation.
- L'utilisation des bibliothèques écrites dans un autre langage pose en pratique plein de problèmes (qui peuvent être résolus), par exemple la représentation des données.

Bibliothèque — exemple

- Le programmeur a dans son programme besoin de trier une liste d'entiers.
- Il sait qu'il y a une bibliothèque qui fournit des fonctions standard pour traiter des listes d'entiers.
- Le programmeur regarde dans la documentation de la bibliothèque et y trouve une procédure `sort` qui prend en argument une liste d'entiers et qui la trie en ordre ascendant.
- Dans le programme il suffit d'appeler cette procédure, il n'est pas nécessaire de connaître le code de la fonction `sort` dans la bibliothèque.

Généricité (1)

Le programmeur peut avoir des besoins qui sont légèrement différents :

- Trier dans un ordre descendant au lieu d'ascendant.
- Trier des listes de flottants au lieu d'entier.
- Trier des listes de données structurées, par exemple une donnée peut consister en le nom d'un salarié, puis son salaire, puis son ancienneté.
- Trier une liste de données structurée selon de critères différentes et complexes, par exemple une fois selon le nom, une autre fois selon ancienneté d'abord puis salaire quand l'ancienneté est la même.

Généricité (2)

- Solution : Écrire des bibliothèques avec des fonctions génériques :
  - Fonction qui peut trier une liste de données de n'importe quel type (polymorphisme)
  - Fonction de trie qui prend en deuxième argument la procédure qui est utilisée pour comparer deux éléments de la liste (fonction d'ordre supérieure)
- Le degré de généricité possible dépend du langage de programmation.

Bibliothèques statiques et dynamiques

- Bibliothèque statique : La compilation du programme utilise le code source du programme + binaire de la bibliothèque. Le compilateur en produit un seul binaire monolithique en faisant une édition de liens statiques.

- Bibliothèques dynamique : La compilation du programme n'utilise que le code source du programme. Le binaire produit contient des appels à des fonctions de la bibliothèque, il faut que la bibliothèque soit installée chez l'utilisateur du programme.

Bibliothèques statiques et dynamiques

Installation d'un logiciel (en format binaire) :

cas statique il y a un seul binaire à déployer qui contient tout qui est nécessaire pour exécuter le programme. Le programme est autonome.

cas dynamique le binaire est plus petit, mais il faut assurer que toutes les bibliothèques sont installées chez l'utilisateur dans les bonnes versions.

Quoi faire quand la bibliothèque change ?

cas statique il faut recompiler tous les logiciels qui l'utilisent, puis déployer

cas dynamique il suffit d'installer la nouvelle version de la bibliothèque (si l'interface binaire (ABI) n'a pas changé...)