# Formation « Calcul parallèle et applications aux plasmas froids »

–

# Valorisation, licences et logiciels libres

Violaine Louvet [1]

[1] Institut Camille jordan - CNRS

10-14/10/2011

# Sommaire

# Quel rapport avec mon travail ? !

- J'utilise la simulation numérique pour ma recherche
- Le travail de développement de code me prend beaucoup de temps ainsi qu'à mes étudiants
- Il n'est pas/peu reconnu comme travail de recherche
- Cependant, il est indispensable à ma recherche
- Chacun de mes étudiants redéveloppe les mêmes choses
- Je suis incapable de reprendre le code d'un thésard
- J'aimerais bien collaborer avec un collègue sur ce code
- J'ai écris ce bout de code l'année dernière mais impossible de comprendre pourquoi j'ai fait ça comme ça
- Dommage, j'aurais bien aimé le réutiliser
- On utilise finalement toujours les mêmes trucs : résolution de systèmes linéaires ...
- ...

# Peut-on vraiment changer tout ça ?

# Peut-on vraiment changer tout ça ?

- Ecrire du code réutilisable, modulable, partageable, diffusable
  - homogénéiser les développements
  - gagner peu à peu du temps sur les développements en réutilisant ce qui a déjà été fait
  - s'approprier relativement facilement un code
  - travailler à plusieurs sur un même code

# Peut-on vraiment changer tout ça ?

- Ecrire du code réutilisable, modulable, partageable, diffusable
- Utiliser des outils adaptés
  - Faciliter le portage
  - Gérer la doc
  - Modifier et pouvoir revenir en arrière, gérer plusieurs versions ...

# Peut-on vraiment changer tout ça ?

- Ecrire du code réutilisable, modulable, partageable, diffusable
- Utiliser des outils adaptés
- Documenter
  - de façon systématique
  - sans être obligé de faire des doublons
  - à jour !

# Peut-on vraiment changer tout ça ?

- Ecrire du code réutilisable, modulable, partageable, diffusable
- Utiliser des outils adaptés
- Documenter
- Ne pas réinventer la roue
  - utiliser ce qui existe déjà

# Peut-on vraiment changer tout ça ?

- Ecrire du code réutilisable, modulable, partageable, diffusable
- Utiliser des outils adaptés
- Documenter
- Ne pas réinventer la roue
- Diffuser sous licence libre
  - Partager pour améliorer

# Sommaire

# Sommaire

# Règles de programmation

## Définition

Une règle de programmation est la description de contraintes à respecter quand on écrit un logiciel. La vérification d'un ensemble de règles de programmation peut être réalisée soit manuellement par lecture du code, soit automatisée par un outil qui localise les violations des règles.

## Objectifs

Faciliter une compréhension universelle et simplifier la maintenance.

# Bon code et mauvais code

## Mauvais code

- Difficile à comprendre et à lire
- Difficile à maintenir
- Difficile à faire évoluer
- Difficile à optimiser
- Difficile à partager et à réutiliser

## Bon code

- Peut être lu et amélioré par un développeur autre que l'auteur d'origine
- Est facile à lire et à comprendre
- A des dépendances minimales
- Est ciblé : ne fait qu'une chose et la fait bien

# Règles de nommage

Définissent la façon de nommer les identificateurs (noms de variables, types, fonctions, classes ...) utilisés de façon à faciliter la lecture du code et à aider à détecter des erreurs.

- Choisir des noms révélateurs des intentions : un nom qui nécessite un commentaire n'est pas auto suffisant.
- Eviter les possibilités d'amalgames et d'ambiguités :
  - Info, Data sont des mots parasites vagues
  - argc, argv sont des identificateurs quasiment standardisés en C
- Choisir des noms prononçables : cela facilite le travail en équipe
- Choisir des noms facilitant les recherches lexicales
- Préférer les identificateurs en anglais (plus universels)
- Choisir un mot par concept : par exemple, get ou fetch

# Notation

Choisir ses propres règles et rester cohérent :

- Certains préfèrent utiliser une majuscule pour le début de chaque mot significatif dans le nom d'une variable, les autres restant en minuscule (ex : StructuredMesh)
- D'autres commencent par une minuscule et suivent ensuite la même règle (ex : structuredMesh)
- Ou encore, utiliser les underscores pour séparer les mots (ex : Structured_Mesh)

# Commentaires

Mal nécessaire : pallient notre incapacité à exprimer nos intentions par le code.
Mais la lisibilité d'un programme n'augmente pas avec le nombre de lignes de commentaires.
La seule source d'information absolument juste est le code.

- ✓ La cohérence entre commentaires et code est à l'entière responsabilité du programmeur
- ✓ Ne pas compenser le mauvais code par des commentaires : il est préférable de le réécrire
- ✓ S'expliquer directement dans le code plutôt que dans un commentaire

Pas Bon

```
! Test si la maille est sur
! le bord ou pas
if (maille%i .eq. 1 .or.&
  maille%i .eq. nx .or. &
  maille%j .eq. 1 .or. &
  maille%j .eq. ny) then
...
```

Bon

```
if (maille.isOnBoundary()) then
...
```

# Spécificités du FORTRAN

La casse n'est pas prise en compte dans le nom des identificateurs : utilisation de l'« underscore » pour définir des noms descriptifs.

- Mots clés fortran, fonctions intrinsèques, types utilisateurs : en majuscule.
- Utilisation du « : : » dans les déclarations, séparant le type du nom de la variable.
- Eviter les mots-clés EQUIVALENCE, SAVE, GOTO, ENTRY, COMMON, PAUSE (issus du FORTRAN 77).
- Utiliser les fonctions intrinsèques du langage.
- Respecter le nombre de dimensions des tableaux dans le passage des arguments des subroutines.
- Utiliser IMPLICIT NONE pour assurer la déclaration de toutes les variables.
- Spécifier la vocation des arguments d'une procédure avec l'attribut INTENT.
- Utiliser les modules pour structurer le programme. D'une façon générale, utiliser les nouvelles fonctionnalités du FORTRAN90/95 pour réaliser des implémentations pseudo-orientées objet.

# Sommaire

# Standards et normes

Quand il n'y a pas de norme, tout est permis :

- le langage BASIC en est un exemple flagrant, étroitement dépendant du compilateur pour lequel il est développé. L'apparition de standards ANSI 14 ans après la naissance du langage n'a pas suffit à modifier le comportement des utilisateurs.
- Le langage C a aussi commencé avec une multiplication chaotique des compilateurs. Afin de remédier à ce chaos l'ANSI (American National Standards Institute) décida de fixer une norme, encore valide aujourd'hui : on appelle cette norme le C ANSI.

## Norme

La norme est une définition universellement valable d'un langage :

- définit comment le compilateur devra être fait ;
- définit les conditions d'application du compilateur au programme.

Quel que soit le système d'exploitation et le compilateur que l'on utilise, le programme écrit suivant une norme pourra fonctionner.

**Essentielle à la portabilité des programmes**

# Normes

## Portabilité

- La notion de norme est centrale dans les questions matérielles aussi bien que logicielles, car c'est elle qui permet la portabilité dans l'espace.
- Portabilité dans le temps : il est nécessaire que ces normes s'accompagnent de procédures ou de protocoles d'évolutivité, permettant aussi bien une compatibilité ultérieure qu'une rétrocompatibilité.

## Norme IEEE

- Problématique des nombres flottants : arrondis, calculs en virgule flottante non associatifs, ...
- Norme IEEE 754 : type des langages, unités de calcul des processeurs, compilateurs, systèmes d'exploitation

# Sommaire

# Sommaire

# Outils de construction de programme

Logiciels permettant d'automatiser (ordonnancer et piloter) l'ensemble des actions (préprocesseur, compilation, éditions des liens, etc.) contribuant, à partir de données sources, à la production d'un logiciel.

- Essentiel dès qu'on a plus de 2 fichiers ... donc pour tous !
- Essentiel à la portabilité des codes

## Outils

- Basés très souvent sur l'outil « make ».
- Les Makefile seuls sont un peu dépassés : complexes, pas portables, pas de gestion simple de fonctionnalités avancées (test, intégration continue ...)
- Autotools : très puissant mais pffff pas très simple ...
- CMake : moins flexible mais plus simple avec des ajouts intéressants (CPack, Ctest). Le petit qui monte, qui monte ...

# Sommaire

# Gestionnaire de documentation

## Objectif

Générer automatiquement (ou presque...) une documentation technique

## Quelles informations ?

- Prototype de fonctions, Classes
- Graphes d'appels, Diagrammes (de classses, ...)
- Liens vers les fichiers sources ...
- Différents formats de sortie (en fonction des possibilités de l'outil) : html, pdf, latex, ps, XML, . . .

→ **doxygen**

# Sommaire

# Gestionnaire de version

Logiciel permettant de gérer l'historique des modifications d'un ensemble de documents.

- Typiquement : les codes source d'un logiciel.
- Mais aussi : article, documentation, site web, fichiers de configuration ...

## Fonctions de base

- ✓ conserver un historique des modifications
- ✓ permettre de travailler à plusieurs (verrous, gestion des conflits)
- ✓ permettre les modifications en parallèle (branches)
- ✓ garantir la sécurité (intégrité, disponibilité, confidentialité)

→ **subversion**

→ **git**

# Sommaire

# Forges logicielles

## Qu'est-ce qu'une forge ?

### Outil de travail collaboratif

- Mise en relation de personnes d'horizons différents (développeurs, experts métiers, ...) au sein de projets.
- Mise à disposition d'un ensemble d'outils : gestionnaire de versions, de bugs, listes de diffusion ...
- Mais aussi : vitrine pour la visibilité d'un projet, pérennisation du projet ...

## Accès à une forge ?

- ✓ Il existe pas mal de forges académiques (SourceSup, INRIA, INRA ...)
- ✓ mais conditions d'accès et limitations
- ✓ Pas de solution miracle

# Sommaire

# Sommaire

# Utiliser l'existant

Ce qu'on ne veut pas redévelopper :

- → parce qu'on n'en a pas le temps
- → pour ne pas repartir de zéro
- → parce que l'écriture de certains codes est très spécialisée
- → parce qu'on fera pas mieux que des personnes qui ont passées des années à optimiser le code
- → parce que c'est un domaine technique/scientifique qu'on ne connaît pas/mal

**Attention aux licences des briques utilisées dans les codes !**

# Sommaire

# Les questions à se poser avant de diffuser

Vous voulez diffuser un logiciel ?

- Qui sont tous les auteurs ? Pour qui travaillent-ils ? Dans quel cadre ont-ils contribué (professionnel, personnel) ?
- Contexte du développement : contrat privé (appel d'offre), collaboration (avec une entreprise, un organisme de recherche ...), contrat européen, ...
- Qui sont les détenteurs (propriétaires) des droits patrimoniaux ?
- A-t-on des documents constituant des preuves d'antériorité, permettant d'identifier les auteurs ?
- Briques logicielles intégrées ? Quelles licences ?
- Quelle licence choisir ? Tous les détenteurs des droits patrimoniaux sont-ils d'accord ?

# Qu'est-ce qu'un logiciel libre ?

Un logiciel publié sous une licence garantissant les 4 libertés suivantes (définition de la Free Software Fondation, FSF) :

- ✓ liberté d'exécuter le logiciel (utilisation à l'infini),
- ✓ liberté d'étudier le fonctionnement (disponibilité du code),
- ✓ liberté de redistribuer des copies,
- ✓ liberté d'améliorer le logiciel et de publier ces améliorations.

## Logiciel propriétaire/commercial

- Un logiciel dont la licence interdit une des quatre libertés mentionnées n'est pas libre : logiciel propriétaire.
- logiciel commercial : logiciel dont le contrat de licence prévoit une contrepartie financière à l'utilisation.

# libre $\neq$ libre de droits

- L'auteur d'un logiciel libre n'abandonne pas ses droits d'auteur mais concède à chacun le droit d'utiliser son logiciel.
- Attention : un logiciel libre n'est pas forcément un logiciel gratuit, et un logiciel gratuit n'est pas forcément libre.
- Un logiciel libre peut être un logiciel commercial.
- Un logiciel distribué en code source n'est pas forcément un logiciel libre.
- Un logiciel diffusé sans licence n'est pas un logiciel libre, au contraire.

# Sommaire

# Propriété du code

Qui est propriétaire du code (informations de copyright) ?

- ✓ Le logiciel relève usuellement du droit d'auteur à condition qu'il soit original.
- ✓ Copyright $\sim$ droits patrimoniaux
- ✓ Dans la fonction publique, le propriétaire des droits patrimoniaux sur le code est l'employeur du développeur.
- ✓ Du point de vue juridique, seul le titulaire des droits patrimoniaux peut décider d'une diffusion du logiciel en libre ou non.
- ✓ Les auteurs du logiciel conservent leurs droits moraux.

Cas de stagiaires ou de thésards (avec financement exterieur à l'établissement) : jurisprudence pas claire en ce qui concerne les droits patrimoniaux ! !

# Sommaire

# Licences

## Qu'est-ce qu'une licence ?

→ Une licence est un contrat entre les auteurs du logiciel et les utilisateurs.

## A quoi sert une licence ?

→ Indique qui peut utiliser un logiciel et pour quelle utilisation.

→ Protége les auteurs (et les éventuels collaborateurs au développement), les propriétaires des droits patrimoniaux, les utilisateurs.

→ Peut entraîner des obligations sur l'utilisation du code comme brique de nouveau code.

## Pourquoi choisir une licence ?

→ Pas de licence = droits d'auteur stricts (personne ne peut utiliser ce logiciel sans le consentement de son propriétaire)

→ Logiciel disponible en téléchargement : libre que s'il est accompagné d'une licence libre.

## Quand choisir une licence ?

→ Avant la diffusion ! !

# Qui décide du type de licence libre ?

- → Tous les détenteurs (propriétaires) des droits patrimoniaux et/ou les auteurs. Choix réalisé en fonction de la politique du laboratoire et de ses tutelles.
- → La réutilisation de codes sous diverses licences peut en limiter le choix.
- → Le choix doit être fait avec les services de valorisation des tutelles du laboratoire.
- → Une licence est associée à une version précise du logiciel. Une nouvelle version du logiciel peut être diffusée sous une licence différente des versions précédentes.

## Notion de copyleft, exigence de réciprocité

Un programme sous licence copyleft peut être copié, utilisé, étudié, modifié et distribué dans la mesure où ces possibilités restent préservées. Le concepteur n'autorise donc pas que son travail puisse évoluer avec une restriction de ces possibilités. Les programmes réalisés à partir d'éléments sous copyleft restent copyleft.

- Copyleft fort
  - Licence initiale s'impose sur tout.
  - Licence contaminante.
  - Exemple : GNU GPL, CeCILL v2
- Copyleft faible
  - Licence initiale reste.
  - Ajouts peuvent avoir autre licence.
  - Exemple : MPL, GNU LGPL, CeCILL-C
- Sans Copyleft
  - Licence initiale ne s'impose pas.
  - Les dérivés peuvent avoir n'importe quelle licence.
  - Exemple : Apache, BSD, MIT, CeCILL-B

# Les erreurs à ne pas commettre !

- → Diffuser le logiciel sans licence.
- → Engager une coopération avec un partenaire exterieur autour d'un logiciel sans formaliser la coopération par un contrat.
- → Utiliser des briques logicielles sans connaître leur origine, leur licence.
- → Diffuser sans licence des éléments glanés sur le net, ou les diffuser avec une licence incompatible avec celle d'origine, y compris après modification.

# Sommaire

# Informer

- choisir un nom pour le logiciel (en évitant d'utiliser un nom déjà utilisé, surtout s'il s'agit d'une marque déposée)
- établir la liste des auteurs (avec un pourcentage de participation)
- établir l'originalité du logiciel avec un document descriptif du logiciel et de ses fonctionnalités
- établir la liste des briques logicielles intégrées, avec leurs licences
- choisir une licence
- faire signer un accord de licence par tous les auteurs
- informer la direction des laboratoires impliqués

# Dans le code

→ Indiquer les détenteurs des droits patrimoniaux :

- *Copyright* (© ou *Droits patrimoniaux*) + année(s) + nom de la personne morale ou physique

→ Mettre en place la licence couvrant le code source :

- Indiquer le type de licence dans chaque fichier du code
- Ajouter un fichier de licence (avec le texte complet ou une URL) aux fichiers source du code
- Mentionner la licence dans la documentation, dans le site web
- Mentionner les briques logicielles utilisées et leurs licences

# Sommaire

# Conclusions

D'une façon générale, si on veux que les développements ne soient pas inutilisables à moyen terme :

- ✓ Suivre quelques règles de bon sens
- ✓ Utiliser des outils adaptés, notamment la gestion de version
- ✓ Documenter et instrumenter pour extraire la doc

Et si on souhaite partager et diffuser son travail :

- ✓ Faire les choses dans le bon ordre !
- ✓ Attention au choix de la licence

# Références

- http://www.projet-plume.org/ressource/faq-forge
- http://www.projet-plume.org/ressource/diffuser-logiciel-recomm-juridiques-admin
- http://www.projet-plume.org/fr/ressource/faq-licence-copyright
- http://www.projet-plume.org/ressource/supports-de-cours-et-de-tp-envol-2010
- Normes C : http://www.open-std.org/jtc1/sc22/wg14/. Nouvelle version : http://en.wikipedia.org/wiki/C1X

# Références

- Fortran : `http://www.nag.co.uk/sc22wg5/`.
  Nouvelle version : `http://fortranwiki.org/fortran/show/Fortran+2008`
- Normes IEEE 754 :
  `http://grouper.ieee.org/groups/754/`
- Papier très intéressant sur le problème des nombres flottants :
  `http://www.validlab.com/goldberg/paper.pdf`
- Standards ouverts : `http://open-std.org/`
- Normes C++ :
  `http://www.open-std.org/jtc1/sc22/wg21/`.
  Nouvelle version :
  `http://fr.wikipedia.org/wiki/C%2B%2B0x`