Les outils du développeur pour le calcul scientifique

Anne Cadiou

Laboratoire de Mécanique des Fluides et d'Acoustique

Séminaires et ateliers pour le calcul scientifique
PMCS2I - LMFA
Vendredi 2 juin 2017

Table des matières

1 Un peu d'histoire...

2 Développer en calcul scientifique

3 Synthèse

4 Proposition

Table des matières

1 Un peu d'histoire...
2 Développer en calcul scientifique
3 Synthèse
4 Proposition

Un peu d'histoire...

- **fin des années 50** : développement des premiers langages de progammation exploitant des compilateurs
- **fin des années 60** : diversification des champs d'application de l'in formatique et production de logiciels
- **début des années 70** : émerge un besoin de méthodologies et d'outils pour les développements logiciels. promotion d'une nouvelle discipline : le génie logiciel.

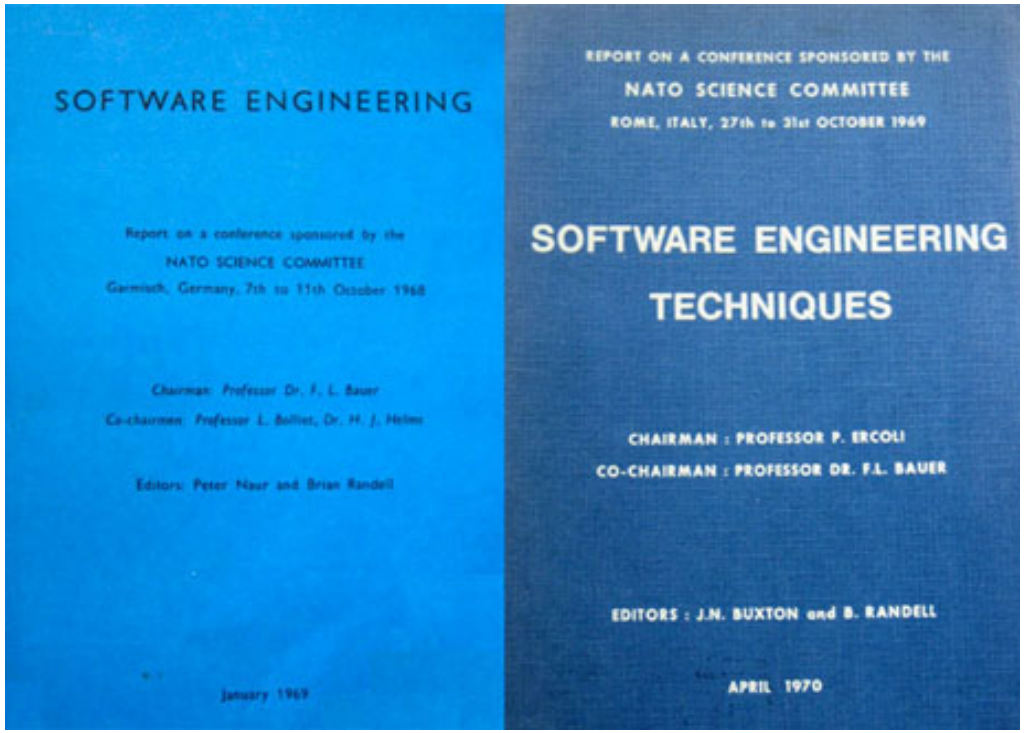


Table des matières

1 Un peu d'histoire...

2 Développer en calcul scientifique

3 Synthèse

4 Proposition

Cycle de vie d'un code de calcul

- Formulation mathématique d'un problème physique (équations, modèles, ...)
- Formulation calculable (formulation discrète, espace d'approximation, algorithme, ...)
- Choix des langages
- Adoption d'une méthode de développement
- Écriture du code
- Construction (compilateurs, interpréteurs)
- Validation, débogage, optimisation
- Production (réalisation des campagnes de calcul, analyse, ...)
- Distribution (licence, publication, ...)

Cycle de vie d'un code de calcul

- Formulation mathématique d'un problème physique (équations, modèles, ...)
- Formulation calculable (formulation discrète, espace d'approximation, algorithme, ...)
- Choix des langages
- Adoption d'une méthode de développement
- Écriture du code
- Construction (compilateurs, interpréteurs)
- Validation, débogage, optimisation
- Production (réalisation des campagnes de calcul, analyse, ...)
- Distribution (licence, publication, ...)

Choix du langage

Langages largement utilisés en calcul scientifique
Fortran, C/C++, Python, R, matlab, bash, (julia, scilab)

Langages

- compilés vs. interprétés
- haut vs. bas niveau

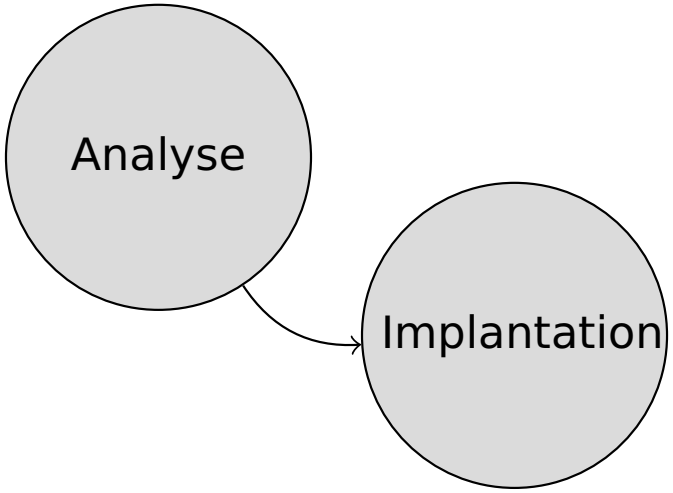
Typologie

- statique vs. dynamique
- procédural vs. objet

Compromis entre interopérabilité, performance, habitudes de la communauté, contraintes du projet, ...

Adoption d'une méthode de développement

De l'abscence de méthode (coder puis corriger)
vers l'identification d'étapes simples

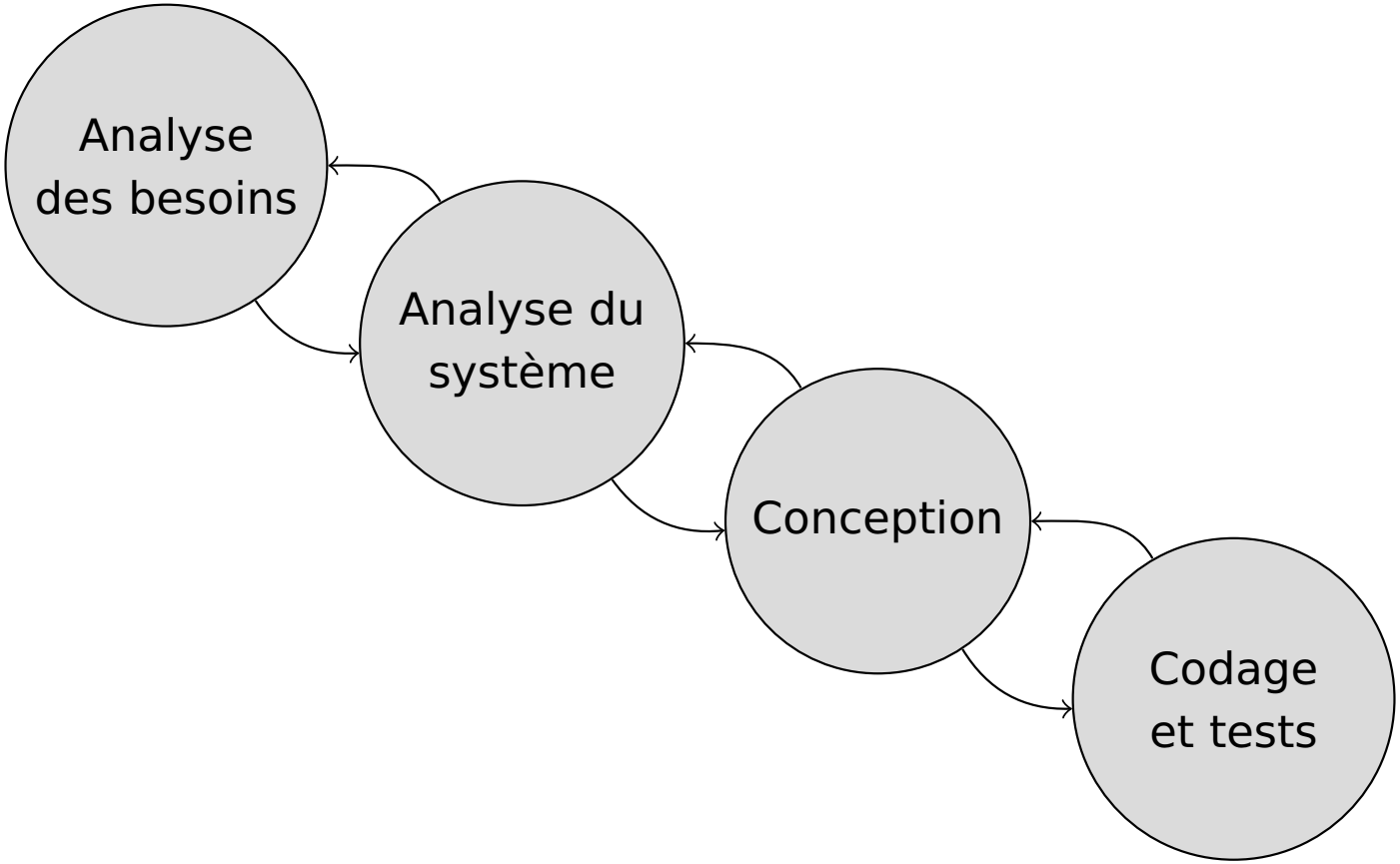


S'avère en principe insuffisant (corriger, itérer, ...)
Principales famillles de méthodes

- Méthodes **linéaires** : cascade, cycle en V, spirale
- Méthodes **agiles** : XP, pilotage par les tests

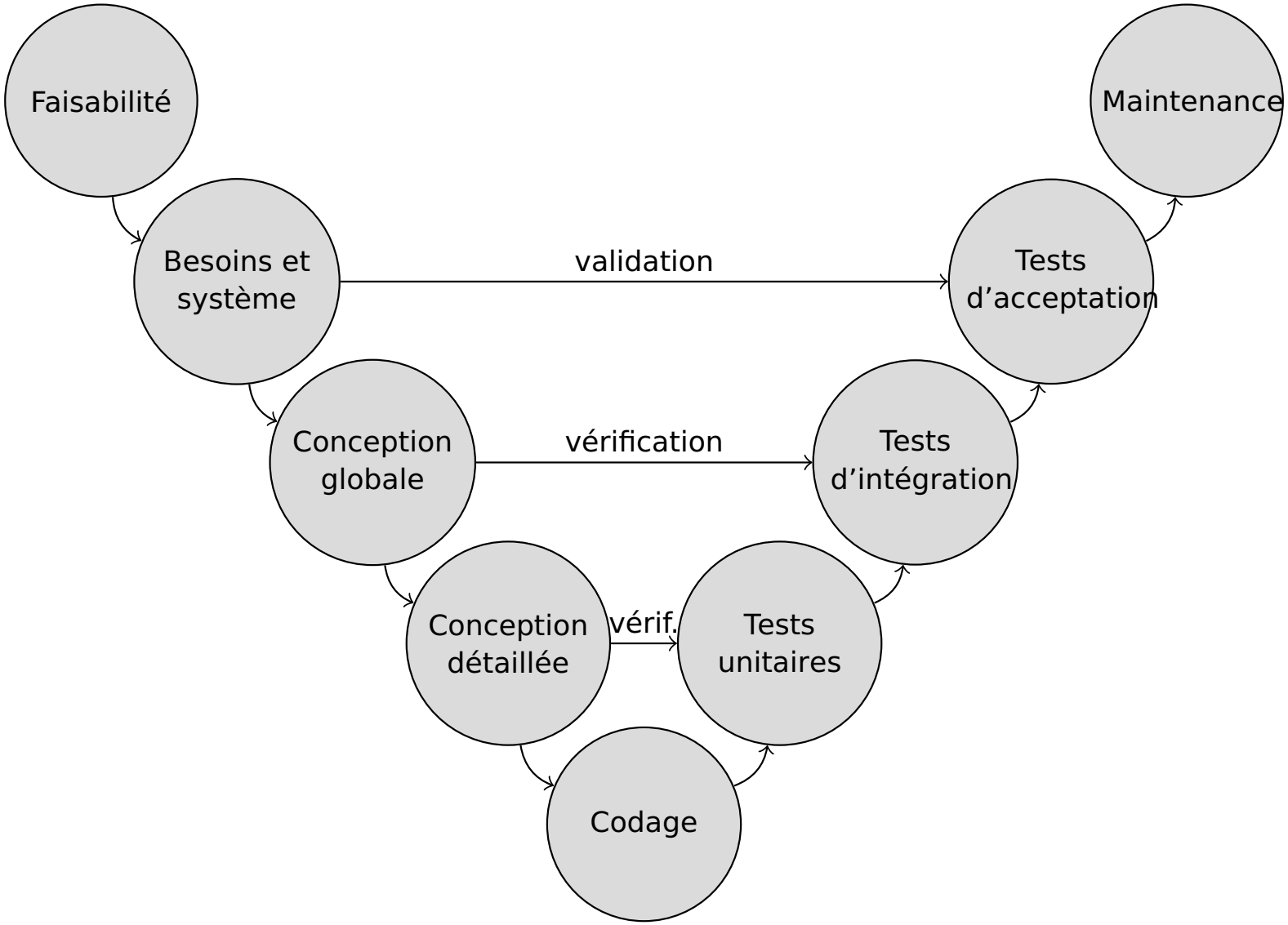
Méthodes linéaires

Modèle en **cascade itérative** (exemple avec quelques étapes)
Sur une proposition de Royce (1970)



Méthodes linéaires

Modèle de **cycle en V**



Méthodes linéaires

- Facile à comprendre
- Facile à planifier dans une gestion de projet
- Modèle en cascade à utiliser
  - quand les besoins sont connus et stables
  - pour faire une nouvelle version d'un code existant
  - pour faire du portage
- Modèle en V à utiliser
  - pour mettre l'accent sur la vérification et la validation
- Modèle en spirale à utiliser
  - pour intégrer l'analyse des risques
  - pour intégrer du prototypage à chaque cycle

Méthodes linéaires adaptées à un cahier des charges fixé ou discuté itérativement

Méthodes agiles

Proposition d'un collectif de développeurs au milieu des années 90
S'adaptent à des projets de petite ou moyenne taille, avec une équipe réduite

- **les individus et leurs interactions**, plus que les processus et les outils
- des **logiciels opérationnels**, plus qu'une documentation exhaustive
- la **collaboration** avec les utilisateurs, plus que la négociation contractuelle
- l'**adaptation** au changement, plus que le suivi d'un plan

Méthodes agiles

Modèle d'extreme programming (**XP**)
Le codage est au coeur du modèle

- implication massive et quotidienne de l'utilisateur
- itérations courtes et livraisons fréquentes
- tests unitaires et intégration continue

Intégration continue

Aide à la **validation** des développements, en permettant de **vérifier** que chaque modification du code source ne produit pas de régression dans l'application du code, en termes de

- fonctionnalité
- performance (informatique, numérique)
- portage (quel que soit l'environnement)

Principe :

- **développement de tests** (unitaires, intégration, ...)
- intégration quotidienne **par les développeurs**

Avantages :

- test immédiat des codes développées
- prévention rapide en cas de code incompatible ou manquant
- détection et réparation continue des problèmes d'intégration

Outils d'intégration continue

- scripts (bash, python, perl, ...),
- Makefile,
- CTests, ...

Serveurs :

- Jenkins,
- Travis CI,
- gitlab-ci,
- CDash, ...

Écriture du code

- vim, emacs, nano, joe, mg, ... (mode console)
- gedit, kate, xemacs, notepad++ (windows), ... (mode graphique)

```
int initPAR(int argc,char *argv[])
{
    setlocale(LC_ALL,"C");
    MPI_Init(&argc,&argv);
    //int TT=0;
    //MPI_Init_thread(&argc,&argv,MPI_THREAD_FUNNELED,&TT);
    //PRINT ("MPI_INI_THREAD : " << MPI_THREAD_FUNNELED <<" .vs. "<<TT<<endl);
    initARG(argc,argv);
#ifdef DEBUG
    MPI_Comm_set_errhandler(MPI_COMM_WORLD,MPI_ERRORS_RETURN);
#endif
    // init MPI
    CHECKMPI(MPI_Comm_group(MPI_COMM_WORLD,&GRP_ALL));
    CHECKMPI(MPI_Comm_create(MPI_COMM_WORLD,GRP_ALL,&COMM_ALL));
    CHECKMPI(MPI_Comm_set_name(COMM_ALL,"COMM_ALL"));
    CHECKMPI(MPI_Comm_size(COMM_ALL,&Parallel::Private::_nprocall_));
    CHECKMPI(MPI_Comm_rank(COMM_ALL,&Parallel::Private::_col));
    // attention ici myid=_col rank mpi de 0 a nproc -1
    // Write cerr in file
    if (hadOption("LOGERROR")) {
        char logerr[MAXSTRLEN];
    sprintf(logerr,"nadiaPython_%d.err",Parallel::Private::_col+1);
    PRINT1("Log sortie erreur dans "<<logerr<<endl);
    static std::ofstream Ferr(logerr);
    std::cerr.rdbuf(Ferr.rdbuf());
    }
    // debug avec MPI
    // ==============
    volatile int flag = 0; // 0 ou 1
    int id = Parallel::Private::_col;
    char hostname[256];
    gethostname(hostname, sizeof(hostname));
    char ip[100];
    hostname_to_ip(hostname , ip);
    // catch signal SEGFAULT
    struct sigaction sa;
    memset(&sa, 0, sizeof(struct sigaction));
    sigemptyset(&sa.sa_mask)    ;
    sa.sa_sigaction = segfault_sigaction;
    sa.sa_flags   = SA_SIGINFO;

    sigaction(SIGSEGV, &sa, NULL);
"parallelMPI.C" 929L, 29607C written                                    161,1          18%
[0] 0:src* 1:build-                                                    "plume" 18:32 27-mai-17
```

IDE

geany

codeblocks

netbeans

eclipse

kdevelop

pydev

Environnement de développement intégré

- complétion de code
- marqueurs syntaxiques
- navigation
- compilateur/interpréteur
- débogueur
- gestion de version

Documentation

Documentation automatique

- Log du versionnement
- Doxygen
- ...

Documents aynchrones

- LaTeX,
- wiki,
- Sphinx,
- ...

Gestion de version

Permet :

- d'**enregistrer** et **suivre** les modifications des sources
- de faciliter le **partage** et la **collaboration** sur des sources
- de rester **réversible**
  - pouvoir retourner à une version antérieure
  - comparer à une version antérieure
- de s'organiser pour avoir toujours une version **fonctionnelle** disponible
- de **documenter** de façon légère (et systématique) les modifications (date, auteur, message court de description de la modification)

Outils de gestion de version

- **Systèmes centralisés**
  - RCS (Revision Control System)
  - CVS (Concurrent Versions System)
    dérivé de RCS, les développeurs partagent un dépôt unique
  - Subversion (SVN) dérivé de CVS, fusions facilitées
- **Systèmes décentralisés**
  - Git
    conçu par Linus Torvalds pour le projet du noyau Linux en 2005
  - Mercurial
    anciennement utilisé pour le développement du noyau Linux (2002-2005)
  - Bazaar en Python, depuis 2005
  - Fossill en C, depuis 2006
  - Veracity depuis 2011
    intègre le suivi des bug et les outils de développement agile

Forges et services d'hébergement de projets

Caractéristiques :

- système de gestion des versions
- gestionnaire de listes de discussion (et/ou de forums)
- outil de suivi des bugs
- gestionnaire de documentation (souvent sur le principe du wiki)

- Redmine, Tuleap, Trac,
- Gitlab, Github, Gogs, Coding,
- FusionForge (reprise opensource de GForge, devenue propriétaire),
- SourceForge, SourceSup, ...

Services d'hébergement externes

`https://sourceforge.net`
`https://www.github.com`

ou interne (ex. Gitlab de votre labo, entité, établissement, ...)

`https://gitlab.mecaflu.ec-lyon.fr`
`https://forge.p2chpd.univ-lyon1.fr`

Construction

Outils de traduction du code en langage machine, basé sur des **compilateurs**ăet des **interpréteurs**

- compilation, édition de liens
- construction des environnements d'exécution

Outils :

- bash,
- Makefile, CMake,
- Autotools (Automake, Autoconf), Ants, Scons, ...
- conteneurs, environnements virtuels, ...

Compilation

Makefile

Scripts (syntaxe propre), utilisés par le programme **make** pour réaliser un ensemble d'actions.
Positionne et déclare les variables d'environnement associées

CMake

- aide à générer le Makefile pour différentes plateformes
  - pour compiler des sources
- possède de nombreuses commandes permettant de localiser les dépendances
  - recherche de bibliothèques
  - facilite le portage ou la gestion de différents compilateurs
- permet l'interface avec des tests unitaires

Outils intégrés

Construction

Autotools, Ant, Scons, ...

suivent une démarche connue de type :

```
./configure; make; make install
```

pour le déploiement et packaging d'application

Options de compilation

Résolution d'une équation de Poisson sur un maillage 65*x*65*x*65 (solveur SOR)

Système	Linux 2.4.7-10smp (Redhat 7.2)	Linux 2.4.7-10smp (Redhat 7.2)	Linux 2.4.7-10smp (Redhat 7.2)
Processeur	Pentium IV Xeon 1.7Ghz Memory : 1024Mb RDRAM ECC PC860 Cache : 8K L1, 256K L2	Pentium IV Xeon 1.7Ghz Memory : 1024Mb RDRAM ECC PC860 Cache : 8K L1, 256K L2	Pentium IV Xeon 1.7Ghz Memory : 1024Mb RDRAM ECC PC860 Cache : 8K L1, 256K L2
Compilateur	GNU g77	Portland Group	GNU g77
Options		-fast	-mcpu=pentiumpro -march=pentiumpro -mpentiumpro -O6 -frerun-cse-after-loop -fno-defer-pop -fschedule-insns -fomit-frame-pointer -fstrength-reduce -fforce-mem -fforce-addr -funroll-loops -freduce-all-givs -Wall
Temps	173 s	88 s	85 s

Débogage

Outils HPC

Allinea DDT, TotalView, gdb, valgrind (memcheck), marmot, scalasca, valgrind (memcheck), VTune, ...

Analyse de performance

Outils de profiling

gprof, Intel Advisor, Linux Trace, memprof,
scalasca, VTune, PAPI, HPCToolkit, TAU, Vampir, Vampirtrace, ...

Métriques

top, free, trace, smem,
systemtap, ...

Production

Réalisation des calculs, traitement des données, ...

Outils de réalisation de campagne de calcul

execo, GC3Pie, OpenMOLE, Sumatra, JUBE, ...

Outils de visualisation

gnuplot, xmgrace, matplotlib (python), matlab, scilab, ...
ParaView, VisIt, MayaVi, Vapor, Medit, xd3d, geomview, gmv, plotMTV, ...
d3, nodejs, OpenDX, Amira, AVS, Plot3D, Scilab, Octave, Yorick, ncview (HDF5), VTK, OpenGL, ...

Licences

- le logiciel est couvert par le droit d'auteur
- pourquoi utiliser une licence ?
  - sans licence, personne ne peut utiliser le logiciel
  - expliciter l'utilisation et la diffusion du programme

Qu'est-ce qu'une licence de logiciels libres ?

- la liberté d'exécuter le logiciel
- la liberté d'étudier le fonctionnement du logiciel
- la liberté de redistribuer des copies du logiciel
- la liberté d'améliorer le logiciel et de publier ses améliorations

Copyright ou copyleft ?
Détermine la réciprocité du code : distribué sous la même licence qu'il a été reçu, héréditaire
GPL, CeCILL, MIT, BSD, Apache, Creative Commons, ...

Table des matières

1 Un peu d'histoire...

2 Développer en calcul scientifique

3 Synthèse

4 Proposition

Synthèse

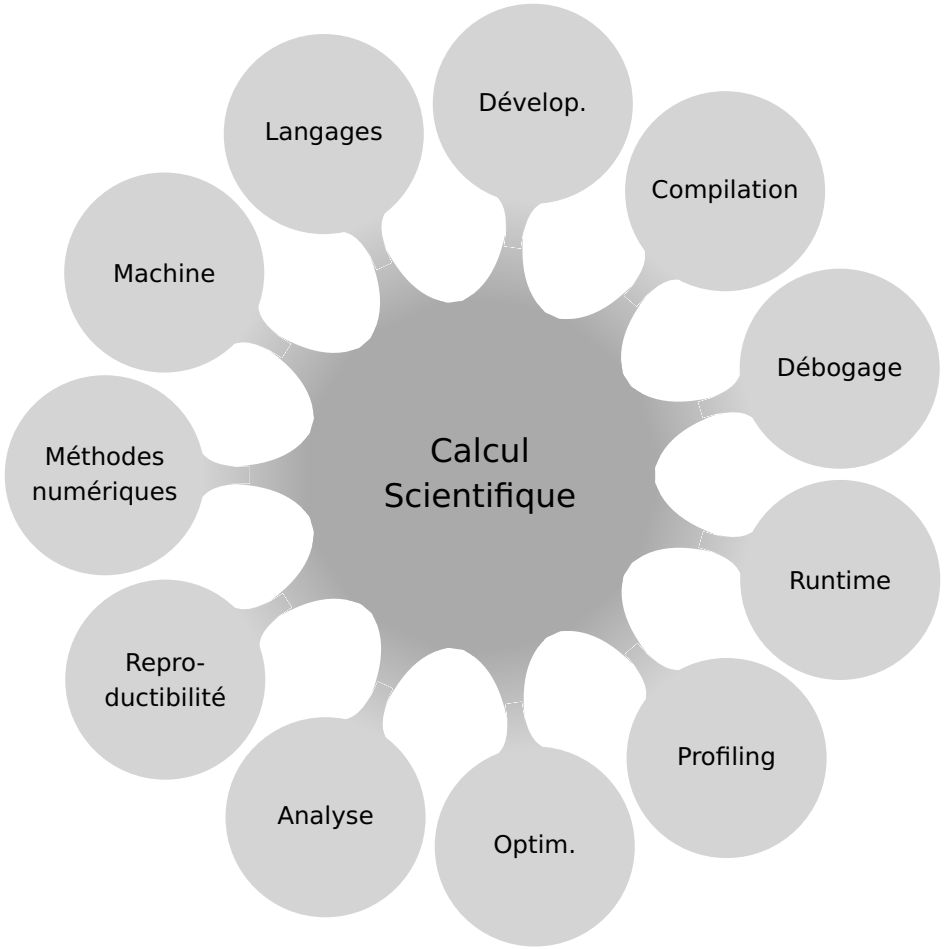

Calcul Scientifique
Dévelop.
Compilation
Débogage
Runtime
Profiling
Optim.
Analyse
Repro-ductibilité
Méthodes numériques
Machine
Langages

Table des matières

1 Un peu d'histoire...

2 Développer en calcul scientifique

3 Synthèse

4 Proposition

Proposition

Faire de la place pour réaliser des sessions de TPs (un peu plus longues)

Nouveaux séminaires

- 06/10/2017 : Choix des langages de programmation
- 10/11/2017 : Architecture
- 01/12/2017 : Compilation
- 02/03/2018 : OpenFoam/ParaView
- 06/04/2018 : Débogage
- 04/05/2018 : Profiling
- 06/07/2018 : Expériences utilisateurs (HDF5, ...)

Sessions de TP sur

- 12/01/2018 : Unix
- 19/01/2018 : Unix
- 02/02/2018 : gitlab et git
- 01/06/2018 : matplotlib
- 08/06/2018 : matplotlib