Logiciels et codes sources : comprendre et accompagner

Violaine Louvet

ANF DDOR, 4 juillet 2023

Plan

Plan

Code source, logiciel, algorithme, de quoi parle-t-on ?

- Eléments issus du rapport Bothorel
  - « Un code source peut être défini comme un ensemble d'instructions exécutables par un ordinateur. »
    En fait, le code source est lisible par l'humain et doit être transformé pour être exécutable par l'ordinateur
  - « Un algorithme n'est pas nécessairement informatisé. »
  - « De manière simplifiée, l'algorithme est une recette de cuisine, et le code sa réalisation concrète »
- Code source vs exécutable
  - Un logiciel fonctionne grâce à du code exécutable, compréhensible uniquement par des machines (binaire).
  - L'« âme »d'un logiciel est dans son code source, c'est-à-dire dans les instructions telles qu'elles sont rédigées pour être lisibles par l'humain.
  - Seul le code source permet d'accéder aux informations techniques et scientifiques

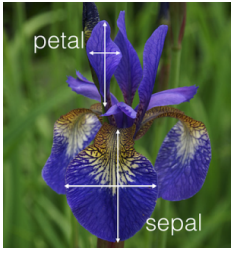
En résumé

- Algorithme : décrit le déroulé pour la résolution d'un problème posé.
- Code source : mise en oeuvre et formalisation de l'algorithme dans un langage informatique (par exemple python, C++, java ...). C'est un (ou plusieurs) fichier(s) texte.
- Exécutable : traduction du code source (en général via un compilateur ou un interpréteur) en code binaire compréhensible par l'ordinateur.
- Logiciel : en général, l'ensemble global comprenant le code source et/ou l'exécutable, et le plus souvent la documentation, des exemples d'utilisation, éventuellement les dépendances ... et évidemment la licence associée.

Définition du collège Codes sources et logiciels du Comité pour la science ouverte (COSO)

Les logiciels de recherche sont développés pour répondre à des besoins spécifiques de la science. Ils sont conçus, maintenus, et utilisés par des scientifiques (chercheurs et ingénieurs) et institutions de recherche, éventuellement dans une dimension internationale.
Ils peuvent découler de travaux de recherche comme ils peuvent les favoriser, notamment par des publications avant/sur/autour/avec le logiciel.
Ceux-ci peuvent se formaliser de différentes façons (une plateforme, un intergiciel, un workflow ou une bibliothèque, module ou greffon d'un autre logiciel) et être ainsi en interaction dans un écosystème ou au contraire plus autonomes.

Illustration par l'exemple : petit code python



- Base de données de caractéristiques d'iris (tailles des sépales et des pétales)
- Entraînement d'un modèle d'apprentissage non supervisé par l'algorithme de clustering du kmeans

Illustration par l'exemple : petit code python

```
#!/usr/bin/env python
# coding: utf-8
# On charge les  bibliotheques dont on a besoin
import pandas as pd
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# On charge les donnees
# Reparti entre donnees d'entrainement
# et donnees de test
iris = datasets.load_iris()

# Stocker les donnees en tant que DataFrame Pandas
x = pd.DataFrame(iris.data)
# Definir les noms de colonnes
```

```
x.columns=['Sepal_Length','Sepal_width',
           'Petal_Length','Petal_width']
y = pd.DataFrame(iris.target)
print(y.columns == ['Targets'])

# Cluster K-means
model = KMeans(n_clusters=3)
# Adapter le modele de donnees
model.fit(x)

# Afficher les differents clusters
colormap=np.array(['Red','green','blue'])
plt.scatter(x.Petal_Length, x.Petal_width,
            c=colormap[model.labels_],s=40)
plt.show()
```

Illustration par l'exemple : petit code python

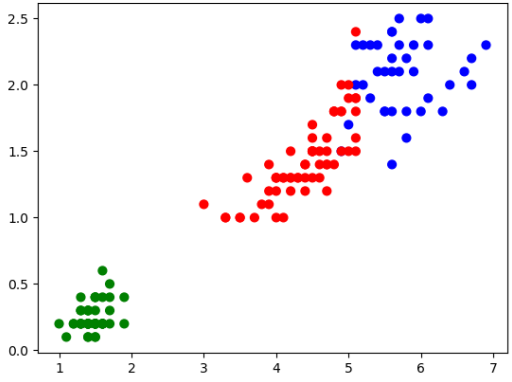
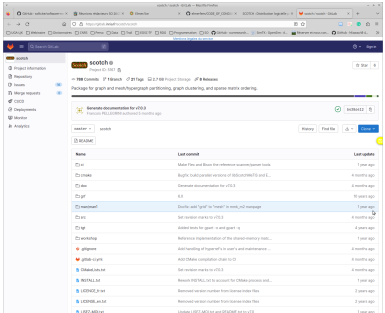


Illustration par l'exemple : code communautaire

Scotch

- Logiciels et bibliothèques séquentiels et parallèles pour le partitionnement de graphes, le placement statique, et la renumérotation par blocs de matrices creuses, et le partitionnement séquentiel de maillages et d'hypergraphes
- Exemple du dépôt



Code de recherche $\neq$ données de recherche

- Les données de recherche sont plutôt passives, les codes sont intrinsèquement vivants
  - On ne change en général pas les données, collectées dans un contexte bien défini
  - On change éventuellement la façon dont on les traite et on les analyse (grâce à des codes)
  - Les codes sont associés à une (ou des) action(s) : création de connaissances, transformation d'informations, visualisation, ...
  - Le code peut être réutilisé tel que, en reproduisant son environnement et toutes ses dépendances mais on a surtout envie de le modifier pour l'adapter à nos besoins propres ou l'enrichir de nouvelles fonctionnalités
- Les codes s'appuient sur des dépendances et tout un environnement logiciel et matériel qui évolue sans cesse
  - Cela complexifie les questions de reproductibilité
- Les codes représentent un travail de création, et correspondent à un cadre juridique différent de celui des données

Plan

Différents contextes de développement

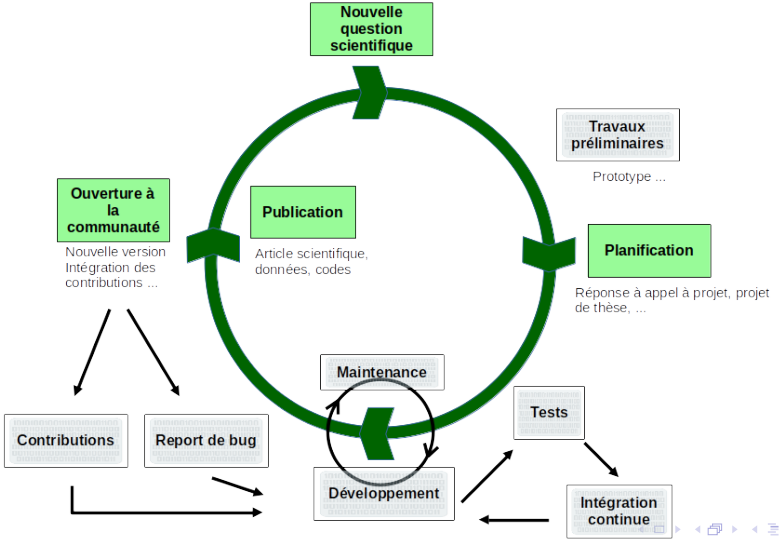
- Individuel : un chercheur / ingénieur (souvent un doctorant) développe seul un code adressant une question de recherche.
  - Sans doute la grande majorité de la production logicielle dans les laboratoires de recherche
  - Développement d'un code adressant une question de recherche particulière et dont les résultats produisent une/des publication(s)
  - Souvent réalisé dans le cadre d'une thèse
  - Une grande question : capitaliser sur ce type de développement
- Equipe de recherche : quelques chercheurs / ingénieurs, en général géographiquement (ou thématiquement) proche (même laboratoire) développent et partagent un code sur le sujet sur lequel ils travaillent.
  - Souvent dans le cadre d'un projet qui peut être financé et donc soumis à des contraintes d'ouverture selon le financeur
  - Développement en continu qui s'inscrit dans la durée avec plusieurs développeurs
  - Avec un objectif de mutualiser et de pérenniser les développements, en particulier permet aux doctorants de bénéficier d'un socle de base solide et d'apporter leurs contributions

Différents contextes de développement

- Communauté : organisation du développement d'un code à l'échelle de toute une communauté scientifique
  - Regroupement des forces d'une communauté pour développer un outil commun très visible permettant à chacun de l'enrichir en fonction de ses besoins de recherche
  - Implique des personnes issues d'établissements différents et géographiquement éloignées y compris à l'international
  - En général, implique la mise en place d'un consortium définissant les accords de Propriété Intellectuelle et la question des licences
  - En général, suppose un/des personnels permanents ou pas dédiés à la gestion du développement (besoin essentiel en génie logiciel)
  - En général, existence d'un processus d'implication de la communauté dans l'orientation des développements (gouvernance du logiciel)
- Partenariat avec une structure privée : développement dans le cadre d'une collaboration avec une entreprise qui donne lieu à un contrat spécifique.
  - Fait l'objet d'un contrat entre les chercheurs concernés (et leurs établissements) et l'entreprise
  - Les conditions de Propriété Intellectuelle et de licence sont définies dans le contrat

→ Tous sont liés au processus de recherche et orientés vers un même objectif : la production de connaissances scientifiques

Cycle de vie d'un code de recherche

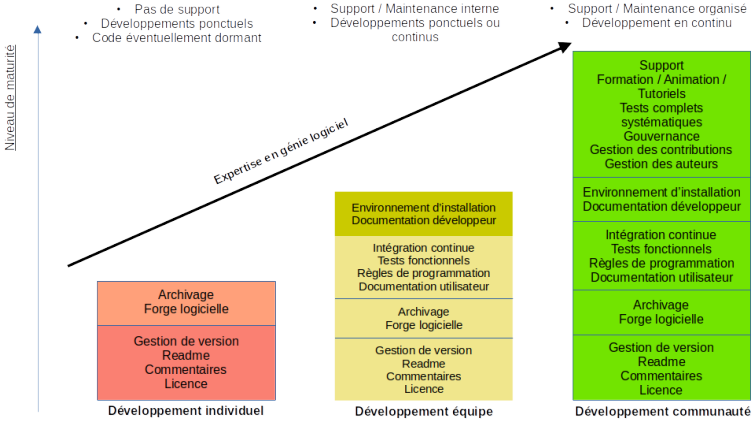


Quelles problématiques se posent lors du développement d'un code de recherche ?

En vrac, les points importants, selon le contexte :

- Développement : travailler à plusieurs, sur du long terme, sur des sites géographiques éventuellement éloignés, voir à l'international
- Qualité :
  - s'assurer que ce qu'on fait est reprenable par (soi et par) d'autres (au sens à la fois compréhensible / lisible et pour la continuité des développements)
  - s'assurer de comprendre ce qu'on a fait dans plusieurs mois / années
  - s'assurer qu'on ne casse pas tout un jour où on est fatigué : tests + intégration continue
  - règles de programmation, documentation
- Points de vigilance :
  - ne pas réinventer la roue
  - aspects juridiques : licences, droits
  - essayer de capitaliser des développements à une échelle plus grande (ou pas ?)
  - Si le code prend de l'ampleur : gérer les contributions, gouvernance ...

Maturité de développement



Plan

Au début était l'historique

Pourquoi a-t-on besoin de gérer l'historique ?

- *Ah, ce code marchait avant que je ne fasse des modifs !!*
- *Tu as modifié quelle version ? Ah mais non, c'était pas la dernière, je te l'ai envoyée par mail !*

- Besoin primaire de gérer l'historique des versions quand on développe (seul ou à plusieurs) !
- Les gestionnaires de version datent du début des années 70

Le cas de git

- Créé en 2005 pour le développement du noyau Linux par Linus Torvalds
  - $>$ 20 millions de lignes de code
  - $\sim$ 14 000 développeurs

Forges logicielles

- Au delà de la gestion de versions, une forge logicielle est un système complet en ligne de gestion, de partage et de maintenance collaborative de textes (et donc en particulier de codes sources)
- Intègre de nombreux outils :
  - système de gestion des versions (par exemple, via Git)
  - outil de suivi des bugs
  - gestionnaire de documentation
  - gestion des tâches
  - intégration continue ...

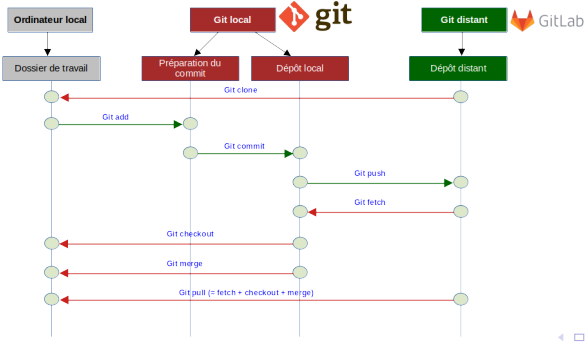
Une ressource essentielle

- Favorise les contributions et facilite leur intégration (pull request)
- Simplifie les interactions entre les développeurs (tickets) et les utilisateurs (forums)
- Facilite la gestion des tests automatiques (intégration continue)

Git et gitlab

Git n'est pas gitlab

- Git est un logiciel de gestion de versions décentralisé (logiciel libre sous GPLv2)
- GitLab est un logiciel libre de forge basé sur git. A noter que gitlab est scindé en deux versions : l'une libre, l'autre propriétaire. Gitlab est une plateforme qui peut être déployée dans les établissements ou laboratoires.



Forges $\neq$ archives

- Ne pas confondre les forges logicielles et les archives de codes
- Software Heritage n'est pas une forge logicielle, c'est une archive de codes sources
  - Pour la pérennisation des codes sources
  - mais pas pour leur développement collaboratif
  - Par contre SWH s'appuie sur les forges existantes
- A l'inverse, les forges logicielles n'assurent pas la pérennisation des codes sources

La forge est le coeur névralgique de tout développement

- Facilite la mise en oeuvre de bonnes pratiques de développement
- Absolument indispensable quand on développe à plus de un mais particulièrement utile aussi pour les développements individuels
- Simplifie aussi la diffusion du logiciel, son référencement, son archivage ...
  - En particulier, c'est le seul endroit où les informations sont constamment à jour

Plan

Archivage et signalement des codes sources

Pourquoi archiver ?

- Le code source est fragile :
  - Obsolescence des formats, problème matériel, dépendances à des outils (forge par exemple) qui disparaissent ...
  - La perte des codes ayant été utilisés pour de la production scientifique arrive malheureusement régulièrement
- Les logiciels sont un des piliers des processus de recherche, au côté des publications et des données et il est essentiel de les préserver

Pourquoi signaler ?

- Assurer la description
- Faciliter la recherche (par domaine scientifique par exemple)
- Permettre la citation
- Valoriser les logiciels

Software Heritage et HAL

Software Heritage

- Initiative dont l'objectif est de construire une archive universelle des codes sources
- En les collectant, les préservant et les partageant sur le long terme
- Lancée en 2016 par INRIA et soutenue par l'UNESCO
- Collecte de l'intégralité des logiciels disponibles publiquement sous forme de code source.
- Depuis des plateformes d'hébergement de codes, comme GitHub, GitLab.com ou Bitbucket, et des archives de paquets, comme Npm ou Pypi ...

- Un nouveau type de dépôt sur HAL : le logiciel
- Complémentarité des deux plateformes
  - Grande visibilité des logiciels dans une démarche de science ouverte via HAL
  - Archivage pérenne via Software Heritage

Publication de logiciels

- Tout comme pour les données pour lesquelles il existe une forme de publication : les data papers, il est possible de publier spécifiquement sur du code
- Liste de journeaux dans lesquels les soumissions sur les logiciels sont acceptées : https://www.software.ac.uk/which-journals-should-i-publish-my-software
- Exemple de process de review du JOSS (Journal of Open Source Software) :
  - Général : dépôt, licence, auteurs et contributeurs
  - Fonctionnalité : installation, confirmation des fonctionnalités annoncées, performances éventuelles
  - Documentation : objectifs, installation, exemples, documentation fonctionnelle, tests, recommandations pour la communauté
  - Article : description claire des fonctionnalités et du problème résolu, positionnement par rapport à d'autres logiciels proches, qualité de la rédaction, références

Plan

Droits d'auteur appliqués au logiciel

Le logiciel est protégé par le droit d'auteur avec des règles spécifiques :

- Droits moraux attachés à l'auteur
- Droits patrimoniaux, qui régissent les modalités d'exploitation, sont propriétés de ou des institutions qui emploient le ou les auteurs
- Contrairement aux autres droits d'auteurs, il y a une « dévolution automatique des droits patrimoniaux à l'employeur »
- Depuis le 15 décembre 2021, c'est vrai aussi pour les stagiaires
- L'algorithme, considéré comme une suite d'idées, ou le modèle mathématique ne peuvent pas être soumis au droit d'auteur
- La documentation est protégée par le droit commun du droit d'auteur

Attribution des droits

- Il est essentiel de pouvoir tracer les différentes contributions pour identifier à qui appartiennent les droits
- En général, différents rôles à considérer :
  - Participation essentielle au développement
  - Participation anecdotique (codage d'exemple, corrections, ...)
  - Participation à la diffusion (script d'installation, création d'un paquet ...)
- L'identification des auteurs doit se faire en amont et de façon concertée

Les types de licences

Deux grands types de licence :

- Licences libres ou Open Source, termes plus ou moins similaires qui définit 4 types de liberté :
  - Liberté d'exécuter le programme, pour tous les usages.
  - Liberté d'étudier le fonctionnement du programme, et de l'adapter à vos besoins. Accès au code source condition requise.
  - Liberté de redistribuer des copies.
  - Liberté d'améliorer le programme et de publier vos améliorations, pour en faire profiter toute la communauté. Accès au code source condition requise.
- Il existe différents types de licences libres :
  - sans copyleft : la licence initiale ne s'impose pas. Permission de redistribuer et de modifier, mais aussi d'y ajouter des restrictions (ex : BSD).
  - copyleft faible : la licence initiale reste, des ajouts peuvent avoir une autre licence (ex : LGPL).
  - copyleft fort : la licence initiale s'impose sur tout. Licence dite contaminante (ex : GNU GPL).
- Licences propriétaires, donc non libre c'est-à-dire que seul l'auteur ou l'ayant droit du logiciel peut le modifier.

Plan

Codes de recherche et principes FAIR

Les principes FAIR sont-ils adaptables au logiciel ?

- Les objectifs des principes FAIR sont de rendre les objets de recherche réutilisables
- Problématique proche de la reproductibilité
- Or la reproductibilité en matière de logiciel est un idéal difficile à atteindre
- Questions ouvertes :
  - Comment définir les contributions et donc les citations ? En particulier quand il y a un grand nombre d'auteurs. Et des types de contributions très différents.
  - Comment intégrer l'environnement, les dépendances ?
  - Comment considérer la problématique de l'installation qui peut être très complexe ?
  - Comment prendre en compte la dynamique du code dans un identifiant pérenne et unique ?
  - ...

Plan

Organiser l'accompagnement : réflexions niveau COSO

Travaux du collège Codes sources et logiciels

- GT1 : Identification et mise en avant de la production logicielle de l'ESR
  - Travaux pour des recommandations pour aller vers un catalogue des logiciels de l'ESR
  - Réflexions autour de l'accompagnement des communautés scientifiques

- Articulation au sein d'un écosystème codes - données - publications
- Harmoniser, fluidifier et accompagner les processus, généraliser les bonnes pratiques de développement et de signalement
- S'appuyer sur la dynamique autour des données :
  - Maillage national avec les ateliers de la donnée au plus proche des communautés scientifiques
  - Intégrer la dimension codes et logiciels dans cet accompagnement
  - Soutenir la mise en place d'un GT des ateliers sur le sujet

Organiser l'accompagnement : le niveau opérationnel

GT5 des ateliers de la donnée autour des codes et logiciels

Objectifs :

- Acculturer sur l'objet, former, sensibiliser
- Proposer des éléments concrets sur différents sujet : plans de gestion logiciel, licences, ...
- Espace d'échanges et de retour d'expérience

- Évolution du PGD vers un plan de gestion des produits de recherche incluant les codes et les logiciels
- Recommandations spécifiques aux logiciels à intégrer sur DMPOpidor
- Recommandations sur les forges à utiliser
- Bonnes pratiques pour le partage des codes et logiciels
- Formations à destination des autres GTs

Conclusions

- Le logiciel est un objet complexe qui doit être considéré dans sa globalité :
  - Nécessité d'associer les bons partenaires sur ces questions (en particulier, centres de calcul)
  - Comme pour les données, la diffusion n'est qu'une (toute) petite partie de la problématique : il faut être en mesure d'accompagner bien au-delà
- Importance de la formation et de la sensibilisation pour bien appréhender tous les aspects
- Le logiciel est devenu depuis peu très apparent dans la science ouverte
  - rester vigilant à ce que toutes les actions soient cohérentes,
  - bien prendre en compte les spécificités du logiciel dont l'historique de l'ouverture est bien plus ancré que pour les données