Logiciels et codes sources, tour d'horizon

Violaine Louvet

Logiciels et codes sources, tour d'horizon

Violaine Louvet

23 mai 2023

Contexte et objectifs

Contexte

- Les codes sont un des piliers de la recherche scientifique dans toutes les disciplines
- La dynamique de la science ouverte renforce la nécessité d'ouvrir les codes, ce qui est déjà le cas de la majorité des développements

Objectifs

- Comprendre le cadre de développement des codes de recherche
- Identifier les points clés essentiels pour se faciliter la vie mais aussi celle des autres utilisateurs et développeurs
- Savoir valoriser son développement

Tour de table

Merci de vous présenter en quelques phrases :

Laboratoire, sujet de thèse

Attendus pour ce cours

Plan

Plan

Plan

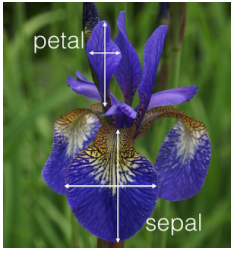
Code source, logiciel, algorithme, de quoi parle-t-on ?

- 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.

Code source, logiciel, algorithme : illustration par l'exemple



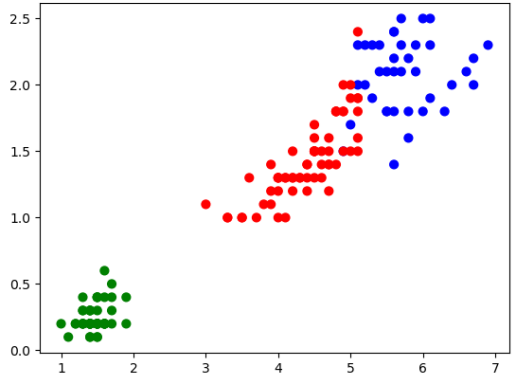
- Base de données de caractéristiques d'iris (tailles des sépales et des pétales)
- Entraînement d'un modèle d'apprentissage non supervisé par l'algorithme de clustering du kmeans

Code source, logiciel, algorithme : illustration par l'exemple

```
#!/usr/bin/env python
# coding: utf-8
# On charge les  bibliothèques 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 données
# Réparti entre données d'entrainement
# et données de test
iris = datasets.load_iris()

# Stocker les données en tant que DataFrame Pandas
x = pd.DataFrame(iris.data)
# Définir 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 modèle de données
model.fit(x)

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

Code source, logiciel, algorithme : illustration par l'exemple



Plan

L'objet logiciel de la recherche

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.

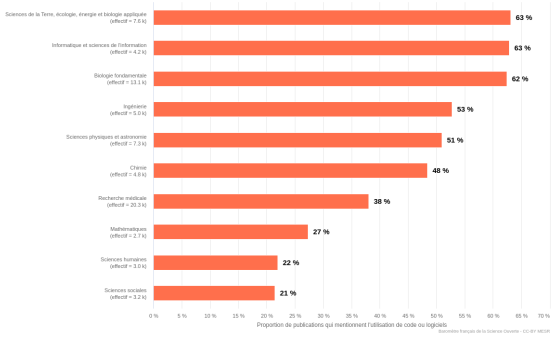
Code de recherche $\neq$ données de recherche

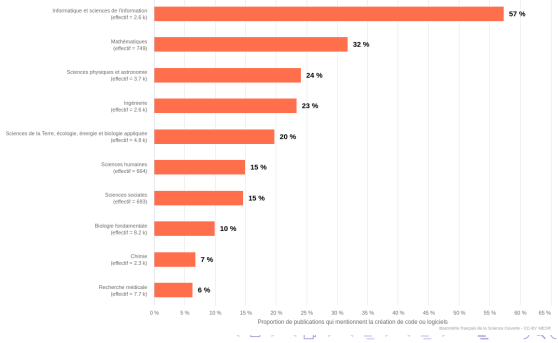
- Les données de recherche sont plutôt passives, les codes sont intrinsèquement vivants
  - On ne change en général pas les données, collectées dans un contexte bien défini
  - On change éventuellement la façon dont on les traite et on les analyse (grâce à des codes)
  - Les codes sont associés à une (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

Des usages variés

- Comme outils de recherche : collecte, traitement de données, tests de modèles ... dans de nombreuses communautés disciplinaires
- Comme résultat ou objet de recherche : en informatique ou en mathématique par exemple, en tant que preuve d'existence d'une solution algorithmique efficace à un problème donné





Différents contextes de développement

- Individuel : un chercheur / ingénieur (souvent un doctorant) développe seul un code adressant une question de recherche.
- 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.
- Communauté : organisation du développement d'un code à l'échelle de toute une communauté scientifique
- Partenariat avec une structure privée : développement dans le cadre d'une collaboration avec une entreprise qui donne lieu à un contrat spécifique.

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

Cas du développement individuel

- 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
- Un niveau de complexité pas forcément très élevé (nombre de lignes de code, dépendances ...)
- Une durée de vie pas toujours très longue
- Forts enjeux autour de la reproductibilité
- Audience attendue : personnelle ou interne à une petite équipe. Mais peut intéresser en dehors de ce petit cercle.

Cas du développement individuel

Points d'intérêt

- Grande souplesse dans le développement
- Sur des questions scientifiques pointues
- Moins de complexité peut impliquer une installation plus facile

Points de vigilance

- Assurer la réutilisabilité y compris pour l'auteur lui-même (bonnes pratiques)
- Eviter de perdre le code
- Pouvoir y accéder et le récupérer
- Capitaliser sur ce type de développement

Cas du développement dans une équipe de recherche

- Souvent dans le cadre d'un projet de recherche partagé par toute une équipe, dans un laboratoire ou sur un thème scientifique bien défini
- 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
- Un niveau de complexité qui peut être élevé en terme de dépendances et de taille du code
- Audience attendue : l'équipe, ses étudiants, et la communauté qui travaille sur des thématiques proches. Eventuellement, selon le sujet, une communauté plus large.

Cas du développement dans une équipe de recherche

Points d'intérêt

- Développement qui capitalise autour des recherches d'une équipe
- Facilite l'appropriation et l'enrichissement par tous les membres y compris les nouveaux arrivés
- Partage des outils sans plus value et souvent chronophage à développer (E/S, pré et post-traitement ...)

Points de vigilance

- Intégrer des bonnes pratiques et des règles de développement
- Utiliser une forge logicielle pour faciliter le travail collectif, voir inciter aux contributions extérieures
- S'assurer a minima de la bonne intégration des contributions (pas de régression, ...)

Cas du développement dans une communauté de recherche

- Regroupement des forces d'une communauté pour développer un outil commun 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)
- Audience attendue : la communauté dans sa globalité (internationale), éventuellement d'autres communautés scientifiques selon la thématique

Cas du développement dans une communauté de recherche

Points d'intérêt

- Mutualisation à l'échelle de toute une communauté
- Code très visible, et intéressant potentiellement beaucoup plus largement
- Pérennisation des développements
- Point d'ancrage pour toute une communauté

Points de vigilance

- S'assurer de la gouvernance du projet
- Gestion de l'attribution des droits
- Intégrer la question du support (dont la formation), de la maintenance ...
- Nécessite en général l'investissement d'un organisme ou d'un établissement principal pour assurer la pérennité

Cas du développement en partenariat avec une structure privée

- Dans le cadre d'une collaboration de recherche avec une entreprise
- 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
- Différents cas de figure peuvent se présenter : développement fermé, partiellement ou complètement ouvert
- Audience attendue : l'entreprise, et selon le niveau d'ouverture, l'équipe de recherche ou une communauté plus large

Cas du développement en partenariat avec une structure privée

Points d'intérêt

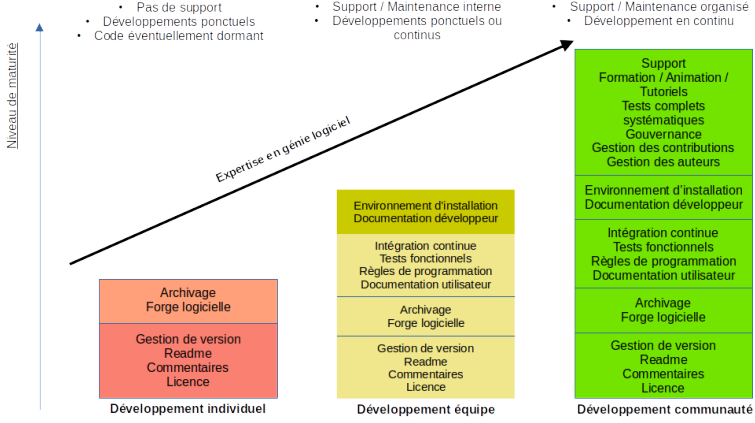
- Une voie de transfert intéressante
- Un moyen de financement pour consolider les développements d'une équipe de recherche

Points de vigilance

- S'assurer de l'accord de consortium pour garder un certain contrôle sur les développements

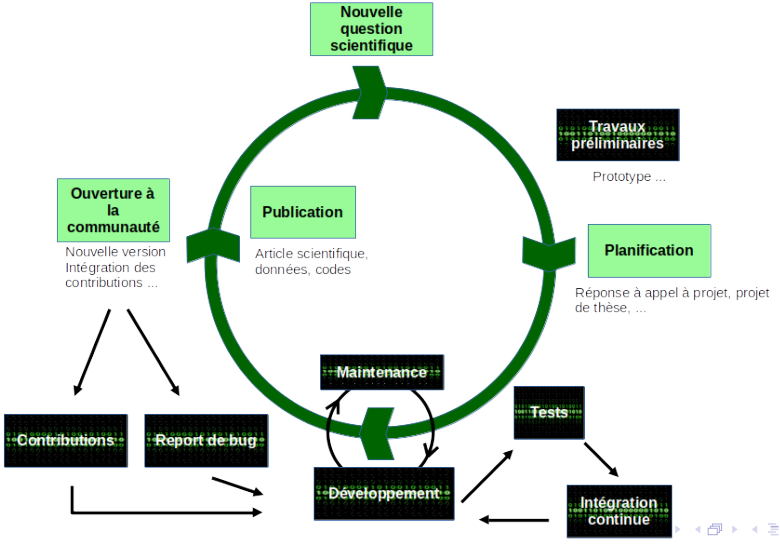
Maturité de développement

- Les différents contextes de développement impliquent souvent des niveaux de maturité très divers



Plan

Cycle de vie d'un code de recherche



La vie d'un code de recherche

- Les questions scientifiques orientent le développement qui est complètement intégré au processus de recherche
- Les cycles et sous-cycles sont itératifs et imbriqués
- Selon le contexte, certaines étapes peuvent ne pas exister ou être juste esquissées (par exemple les tests et l'intégration continue)
- La temporalité est très variable : le cycle peut s'interrompre sur une durée plus ou moins longue (code dormant, voir mort) et reprendre si l'intérêt scientifique renaît

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)

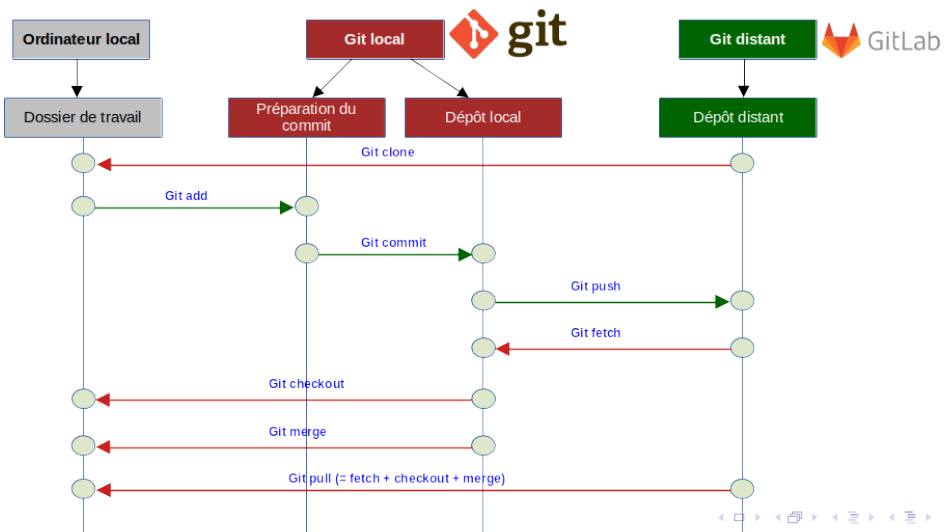
Git et gitlab en deux slides

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.

- Le système de gestion de versions a comme objectif de pouvoir retracer des modifications physiques dans la mémoire de façon intelligente
- Ces changements sont stockés dans un dépôt (repository)
- Git fonctionne en créant deux dépôts des changements :
  - Le premier se trouve sur la même machine des fichiers de travail
  - Le deuxième se trouve dans une autre machine, souvent un serveur ou une plateforme cloud comme GitLab, qui s'occupe de centraliser les changements

Git et gitlab en deux slides



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

Forges disponibles [1]

- Forges de l'ESR
  - De nombreux établissements, organismes voir laboratoires ont déployé leur propre forge (en général gitlab mais il en existe quelques autres)
  - Liste non exhaustive : CNRS, INRIA, Huma-Num, INRAE, UGA, U Bordeaux, ...
  - SourceSup est la forge nationale hébergée par Renater. Elle s'appuie sur FusionForge (et pas gitlab)
  - La plupart de ces forges ont un accès restreint à leur communauté / personnels avec un degré d'ouverture plus ou moins important
- Forges commerciales (github, gitlab.com ...)
  - Très utilisées dans le cadre de collaborations internationales ou pour assurer plus de visibilité
  - Pas de contraintes d'accès comparé à la plupart des forges ESR
  - Mais attention aux conditions d'utilisation ! Et à la pérennité de ces plateformes

1. Travail du GT2 du COSO Codes Sources et Logiciels

Plan

Pourquoi s'intéresser à la qualité logicielle ?

L'objectif est d'assurer pour soi, pour ses collègues, pour sa communauté, pour tous de :

- comprendre ce que fait le logiciel et comment il le fait : les sources doivent être accessibles dans un environnement pérenne, le code doit être commenté ...
- installer le logiciel sans problème (avec une licence adaptée) dans différents contextes via l'utilisation d'outils adaptés (gestionnaire de paquets, outils de compilation, conteneur, ...)
- utiliser le logiciel, pouvoir identifier clairement les dépendances (de confiance et suffisamment pérennes), existence d'une documentation, d'exemples ...
- contribuer, faire évoluer les fonctionnalités grâce à la mise en place de règles de programmation claires, un code modulaire, un nommage explicite ... et une documentation développeur
- maintenir sur le long terme, intégrer des tests, pouvoir corriger les bugs sans dégrader le logiciel, pouvoir revenir dessus après plusieurs mois/années ...

Forte variabilité de l'exigence de qualité

La qualité logicielle est un sujet très vaste

- Une très large littérature sur les modèles, les indicateurs, les normes ...
- L'exigence est très dépendante du logiciel : sa cible, son utilisation, son périmètre ...
- Plus l'exigence est forte, plus les compétences en terme de génie logiciel doivent être présentes dans l'équipe de développement

Les attendus de la qualité des logiciels

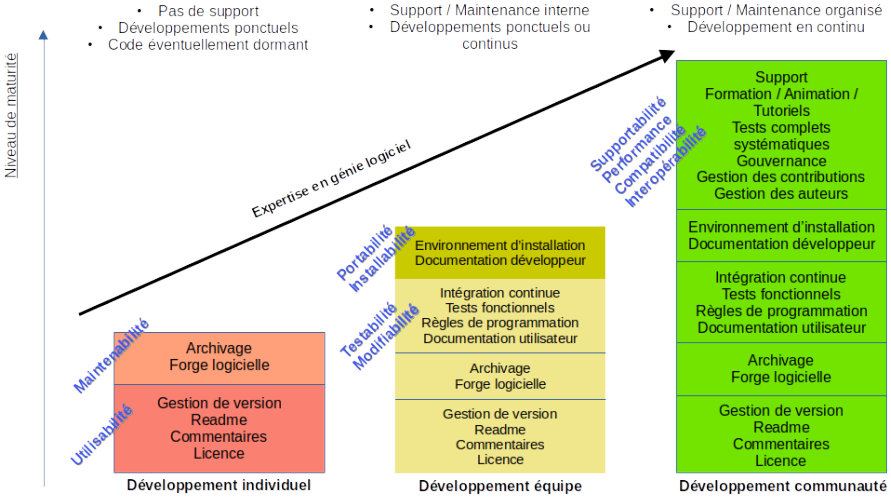
Liste non exhaustive et non priorisée !

- Performance / Utilisation de ressources
- Utilisabilité / facilité d'utilisation
- Maintenabilité
- Interopérabilité
- Compatibilité
- Installabilité
- Portabilité / adaptabilité
- Modifiabilité
- Supportabilité
- Testabilité

Tout n'est pas forcément pertinent pour les codes de recherche !!

D'autres éléments de qualité comme la sécurité, l'attractivité (esthétique de l'interface utilisateur), la tolérance aux fautes sont encore moins appropriées à la plupart des codes développés dans les labos.

Qualité minimale attendue



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

Dater la création

- Le droit d'auteur s'applique dès la création du logiciel, il est donc essentiel de pouvoir dater cette création
  - La preuve peut être faite par tout moyen, y compris des choses simples : enregistrement des versions, cahier de laboratoire, envoi de mail à soi-même avec la dernière version ... tout ce qui a un horodatage
  - Preuve aussi par le dépôt APP (Agence pour la Protection des Programmes) : se rapprocher de son service valorisation

- S'il n'y a pas de droit explicitement donné à travers une licence, utiliser un logiciel relève de la contrefaçon.

Pas de licence $\equiv$ tous droits réservés

Les types de licences

Deux grands types de licence :

- Licences libres ou Open Source, termes plus ou moins similaires
  - Les licences dites « libres »(traduction bancale de l'anglais « royalty-free ») viennent de la Free Software Foundation
  - Open Source vient de l'Open Source Society
  - Ils n'ont pas exactement la même philosophie
  - Une licence libre ne veut pas dire libre de droit, bien au contraire !

- Licences propriétaires, donc non libre c'est-à-dire que seul l'auteur ou l'ayant droit du logiciel peut le modifier.

- Liste des licences existantes : https://spdx.org/licenses/ (Software Package Data Exchange)

Le logiciel libre

- L'ouverture des logiciels développés dans le cadre de la recherche publique est un point essentiel de la science ouverte

- Le logiciel libre définit 4 types de libertés pour l'utilisateur :
  - 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.

Les principales licences libres

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.
- copyleft faible : la licence initiale reste, des ajouts peuvent avoir une autre licence.
- copyleft fort : la licence initiale s'impose sur tout. Licence dite contaminante.

Type	**Exemple de licences**
Sans copyleft	BSD license, Apache License 2 MIT
Copyleft faible	GNU library or « Lesser »General Public License (LGPL)
Copyleft fort	GNU General Public, License EUPL

Exemple de licences libres : GNU GPL

- Exemple de la licence GNU GPL = GNU General Public License :
  - C'est la licence la plus connue et la plus répandue dans le monde du libre. Elle autorise, sans l'accord de l'auteur et sans crainte d'une action en contrefaçon :
    - l'utilisation du logiciel,
    - l'étude du fonctionnement du logiciel,
    - l'adaptation du logiciel aux besoins de l'utilisateur,
    - la copie et la diffusion auprès d'amis ou de collègues,
    - l'amélioration du logiciel par l'utilisateur et la distribution du logiciel modifié au public.
  - ATTENTION : il s'agit d'une licence dite contaminante !! Elle impose à l'utilisateur de rediffuser ces modifications sous licence GPL $\rightarrow$ l'ensemble du logiciel réutilisant un bout de code GPL est contaminé par la GPL.

Exemple de licences libres : BSD

- Exemple de la licence BSD = Berkeley System Distribution licence :
  - C'est une licence très peu restrictive : les logiciels diffusés sous licence BSD peuvent être librement copiés ou modifiés. Une seule contrainte : faire figurer sur tous les travaux dérivés, les documentations et les publicités relatives à ces travaux une mention apparente faisant référence à la licence elle-même, et mentionnant les auteurs du logiciel original.

  - A noter : dans le cas de logiciels très populaires, le nombre des auteurs est souvent très important. La mention de l'intégralité des contributeurs est donc complexe et peu lisible. Pour supprimer ce désagrément, la version 2 de la licence BSD supprime les obligations de mentions des auteurs.

Licence multiple

- Principe : distribuer un même logiciel sous plusieurs licences différentes, en général du dual-licensing

- Intérêt : concilier un aspect financier et la possibilité de contribuer au logiciel
  - Soit l'utilisateur utilise une licence libre et peut contribuer en mettant à disposition son travail
  - soit l'utilisateur achète une licence d'utilisation qui va lui permettre d'intégrer le logiciel dans un autre projet propriétaire.

- En pratique : la gestion des licences multiple est délicate. En particulier, s'assurer de la complète paternité sur le logiciel.
  $\rightarrow$ Contacter votre service de valorisation

Licence multiple : exemple

Exemple : MySQL

- Système de gestion de bases de données relationnelles.
- Logiciel libre et open source.
- Distribué sous une double licence GPL et propriétaire.
- Ce qui permet de le distribuer dans un produit propriétaire.

En pratique, mettre une licence sur un logiciel

- Anticiper le choix de la licence !
  - Dépend des conventions et contrats éventuels
  - En accord avec les établissements concernés, et l'ensemble des co-auteurs
  - Avant l'ouverture et la diffusion
  - Tenir compte des licences des dépendances incluses
- En pratique :
  - Intégrer le fichier texte de la licence dans le dépôt du code (fichier LICENSE ou COPYING)
  - Intégrer les entêtes de la licence dans tous les fichiers source
  - Intégrer un fichier AUTHORS avec le nom des auteurs

```
/*
 * Copyright (c) 2017 Alice Commit <alice@example.com>
 *
 * SPDX-License-Identifier: BSD-2-Clause
 * License-Filename: LICENSES/BSD-2-Clause_Alice.txt
 */
```

Plan

Enjeux de la reproductibilité

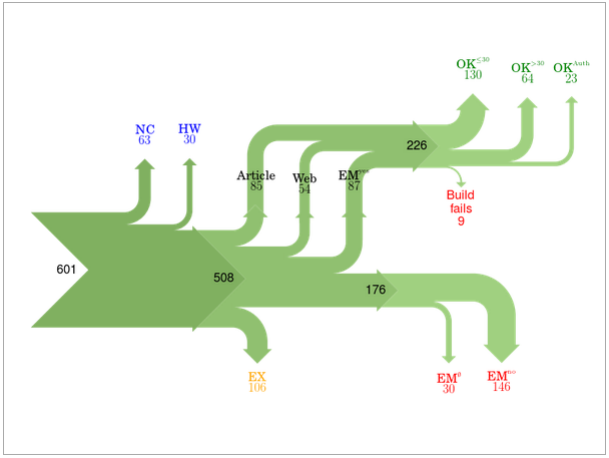
Intégrité scientifique

Décret no 2021-1572 du 3 décembre 2021 : « l'ensemble des règles et valeurs qui doivent régir les activités de recherche pour en garantir le caractère honnête et scientifiquement rigoureux »

- L'ouverture des publications, des données et des codes sources est une condition nécessaire à la reproductibilité des résultats scientifiques
- Suffisante ?

La recherche reproductible (computationnelle) est encore aujourd'hui l'exception
il vaut mieux parler de transparence

Exemple : Repeatability in Computer Science [2]



Legend				
Classification		Code Location		Build Results
BC	Paper where the results are backed by code.	Article	Code is found from link in the article itself.	OK≤30	We succeed in building the system in ≤30 minutes.
NC	Paper excluded due to results not being backed by code.	Web	Code is found from a web search.	OK>30	We succeed in building the system in >30 minutes.
HW	Paper excluded due to replication requiring special hardware.	EMyes	Code is provided by author after email request.	OK>Author	We fail to build, but the author says the code builds with reasonable effort.
EX	Paper excluded due to overlapping author lists.	EMno	Author responds that the code cannot be provided.	Fails	We fail to build, and the author doesn't respond to survey or says code may have problems building.
		EMno	Author does not respond to email request within 2 months.		

2. http://reproducibility.cs.arizona.edu/

Définitions [3]

Pas mal de concepts différents dont les définitions ne font pas consensus : répétabilité, reproductibilité, réplicabilité

Rerunnable Can you re-run your program ? One day, one week, one month, one year (just kidding) apart ?

Repeatable Can you re-run your program and get same results ? Did you save everything, including random seed ?

Reproducible Can someone re-run your program and get same results ? Did you save the software stack ?

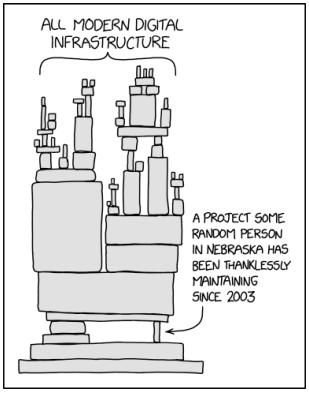
Replicable Can someone reimplement your model and get same results ? Did you describe everything ?

Reusable Can someone reuse your program using different data ? Is your software data-dependent ?

3. Konrad Hinsen, Nicolas P. Rougier. ReScience 2017. hal-01573262

Vraie et fausse reproductibilité [4]

- Le code ouvert permet et facilite l'inspection critique du source
- Mais il ne suffit pas de rendre le code public
- Il faut aussi ouvrir son environnement
- Et évaluer la fiabilité des dépendances
- Sans parler des dépendances possibles au hardware



a

a. https://xkcd.com/2347/

4. Open Science Days UGA 2022, K. Hinsen

Complexité liée au logiciel

Chaque résultat numérique d'un code dépend de :

- les données d'entrée
- le code source
- les bibliothèques (dépendances) utilisées par le code
- les compilateurs / interpréteurs
- les options de compilation
- le système d'exploitation de l'ordinateur
- le hardware de la machine

Tout un environnement non stable, pas toujours documenté, qui évolue en permanence et qu'on ne contrôle pas.

Reproductibilité et bonnes pratiques

Le respect de bonnes pratiques de développement facilite le cheminement vers la reproductibilité :

- Versionning : être capable d'accéder à la version du code qui a permis la publication
- Tests : permet d'assurer la reproductibilité du comportement au cours de la vie du code
- Revue de code : s'assurer que le code est lisible et compréhensible par d'autres que soi
- Documentation : décrire ce que fait le logiciel (approche scientifique), comment l'utiliser, ...
- Utiliser des formats ouverts
- Définir des conventions de programmation et de nommage
- Utiliser des environnements logiciels reproductibles : guix, nix

Plan

Archivage 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

- Archivage $\neq$ stockage $\neq$ outil de développement
  - Archivage $\equiv$ préservation sur du long terme

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 ...

Collect
Preserve
Share

https://archive.softwareheritage.org/

Signalement des logiciels de la recherche

Pourquoi signaler ?

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

- Importance des métadonnées pour :
  - Décrire de façon précise et contrôlée un logiciel
  - Identifier son usage
  - Spécifier le contexte
  - Créditer plus facilement les créateurs
- Importance des identifiants pérennes pour :
  - Garantir un lien stable à la ressource en ligne

HAL, Hyper Articles en Ligne

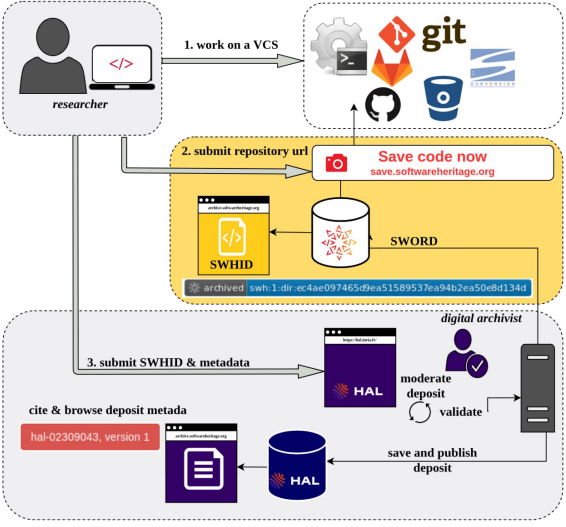
- Archive ouverte pluridisciplinaire
- Initiée en 2000 par le CNRS et exploitée par le CCSD - Centre pour la Communication Scientifique Directe
- Fournit des outils pour l'archivage et la diffusion ouverte des résultats scientifiques.
- Où les chercheurs peuvent déposer leurs résultats académiques dans le respect de leurs droits d'auteur
- Supporte différents types de dépôt :
  - Publications,
  - Documents (par exemple préprints et rapport),
  - Thèses ...
- Pour rendre la recherche aussi accessible et ouverte que possible

https://hal.science/

HAL + Software Heritage

- Un nouveau type de dépôt sur HAL : le logiciel
  - Collaboration initiée en 2018 entre HAL et SWH
  - Après une phase de test avec INRIA, déployé largement aujourd'hui
  - En particulier, le dépôt SWHID (identifiant pérenne de SWH) est désormais en production
- Complémentarité des deux plateformes
  - Grande visibilité des logiciels dans une démarche de science ouverte via HAL
  - Archivage pérenne via Software Heritage
  - Modération des métadonnées pour assurer leur qualité
  - Différents formats d'export pour faciliter la citation

HAL + Software Heritage : en pratique



Métadonnées

En plus des métadonnées spécifiques à la discipline de recherche concernée par le code, il existe des schémas de métadonnées dédiés au logiciel :

- Citation File Format (CFF) : https://citation-file-format.github.io/
- Fichier schema.org : https://schema.org/, schéma de métadonnées généraliste
- Fichier CodeMeta : https://codemeta.github.io/, est un format pour les métadonnées logicielles génériques, qui étend les fichiers schema.org.

Focus sur CodeMeta

- Format privilégié pour HAL + SWH
- La présence d'un fichier codemeta.json dans le dépôt du projet sur SWH est détecté par HAL qui charge automatiquement les métadonnées
- Existence d'un outil en ligne facilitant la création du fichier json : https://codemeta.github.io/codemeta-generator/

Citation

- L'utilisation de HAL et SWH facilite la citation du logiciel :
  - Plusieurs formats d'exports possibles (BibTeX, TEI, DCTERMS, codemeta.json, etc.)
  - Permet de créer les liens nécessaires entre les publications, les données et les codes
  - Indispensable pour la reproductibilité. En particulier, le SWHID désigne précisément un logiciel dans son contexte (version, commit, ...)
  - Important pour créditer correctement les auteurs et contributeurs

Baromètre Science Ouverte

- Le BSO mesure le niveau d'ouverture des publications en France
- Les mentions de logiciels y sont désormais aussi détectées en particulier grâce aux citations.

Plan

Principes FAIR

Findable/Trouvable : Données faciles à trouver.

- possédant un identifiant unique et pérenne
- décrites par des métadonnées riches
- enregistrées ou indexées dans une source interrogeable

Accessible : Données ou au moins méta-données facilement accessibles.

- entrepôt de confiance, pérenne, certifié
- définir les conditions d'accès et la licence de diffusion
- si embargo ou accès restreint : méta-données accessibles

Interoperable : Facile à combiner avec d'autres jeux de données, par les humains et les systèmes informatiques

- formats libres et ouverts
- mise à disposition du code source si le logiciel de traitement existe
- standards de métadonnées et vocabulaire standardisés

Reusable/Réutilisable : Prêtes à être réutilisables pour une future recherche y compris via des méthodes informatiques

Codes de recherche et 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

Plan de Gestion de Données / Data Management Plan

- Document exigé désormais pour la majorité des projets financés
- Aide concrète à la gestion des données durant tout le projet et au-delà
- Permet de se poser les bonnes questions, et d'anticiper les besoins :
  - Type des données, volumétrie, stockage, sauvegarde, partage ...
  - Problématiques juridiques
  - Diffusion, valorisation et conservation
  - Financement prévu ...

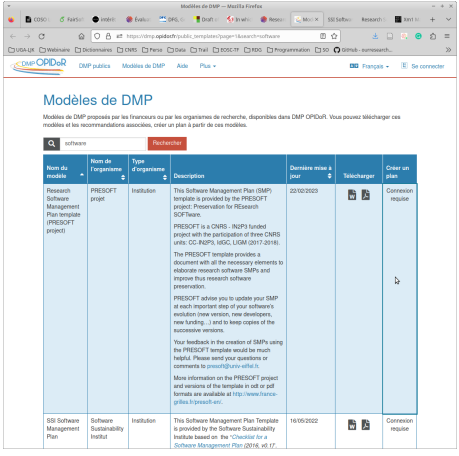
Plan de Gestion Logiciel / Software Management Plan

- La nécessité de prévoir et d'anticiper le déroulement d'un projet de recherche en ce qui concerne les logiciels développés dans ce cadre est aussi importante que pour les données
- Mais les questions ne se posent pas de la même façon
- Quelques éléments clés :
  - Objectif (scientifique) du logiciel
  - Cible en terme d'utilisateurs
  - Aspects techniques : plateforme de développement, langage de programmation, règles de programmation, dépendances, intégration des contributions, intégration continue / tests, documentation, packaging, OS cibles ...
  - Aspects juridiques : impact des dépendances, attribution des droits, gestion des auteurs et contributeurs, copyright assignement, licences, ...
  - Ouverture et diffusion

Modèles de Plan de Gestion Logiciel

Il existe peu de modèles de Plan de Gestion Logiciel.
Deux sont disponibles sur DMP Opidor (en anglais) :

- PRESOFT project - également sur Hal
- Software Sustainability Institut



Plan

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

Le cas de ReScience [5]

- Revue créée en 2015 par Konrad Hinsen et Nicolas Rougier
- Objectif : publier les tentatives des chercheurs de répliquer les calculs effectués par d'autres auteurs, en utilisant des logiciels écrits indépendamment, libres et open-source, avec un processus ouvert d'examen par les pairs

5. http://rescience.github.io/

Plan

Références I

- *Anne Canteaut, Miguel Angel Fernández, Luc Maranget, Sophie Perin, Mario Ricchiuto, et al.*. Software Evaluation. [Research Report] Inria. 2021. hal-03110728
- *Lila Ammour, Anne-Sophie Bonne, Patrick Moreau, Jean-Marc Schmittbiel, Jean-Christophe Souplet.* JE CODE : QUELS SONT MES DROITS ? QUELLES SONT MES OBLIGATIONS ?. 2019. (hal-02399517)
- *Teresa Gomez-Diaz*, Licences et droit d'auteur pour votre logiciel de recherche
- Baromètre science ouverte
- *Arnaud Legrand*, Git, MOOC Recherche Reproductible
- *Roberto Di Cosmo*, Multi Licencing
- ACM, Artifact Review and Badging

Références II

- *Konrad Hinsen, Nicolas P. Rougier.* ReScience. Open science, transparence et évaluation. Perspectives et enjeux pour les chercheurs, URFIST Bordeaux, Apr 2017, Bordeaux, France. hal-01573262
- *Konrad Hinsen.*, Objectif calcul ouvert. Open Science Days@UGA 2022.
- *Chue Hong, N. P., Katz, D. S., Barker, M., Lamprecht, A.-L., Martinez, C., Psomopoulos, F. E., Harrow, J., Castro, L. J., Gruenpeter, M., Martinez, P. A., Honeyman, T., et al. (2021).* FAIR Principles for Research Software (FAIR4RS Principles). Research Data Alliance. DOI : 10.15497/RDA00065