Comment bien initier un projet de développement ouvert ?

Violaine Louvet & Tuğçe M. Demir

Colloque sciences ouvertes 2024, 28 novembre 2024

Contenu de l'atelier

Atelier d'1h

- Cycle de vie du logiciel
- A travers le parcours d'un plan de gestion logiciel
- En approfondissant certaines problématiques

En pratique

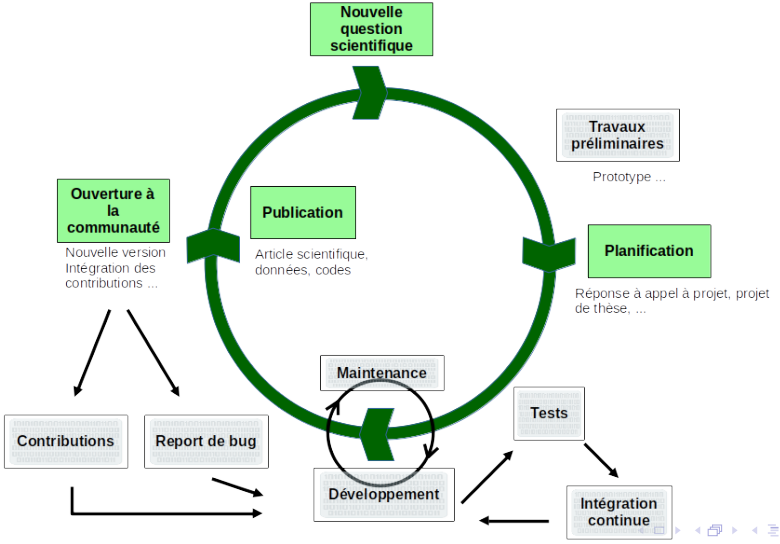
- Des apports « théoriques »
- Des échanges et des discussions

Les objectifs

- Identifier les bonnes pratiques
- Pour bien démarrer un développement

Déroulé

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 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

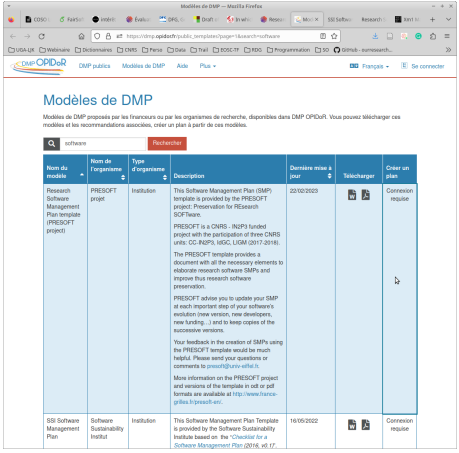
- La nécessité de prévoir et d'anticiper le déroulement d'un projet de recherche en ce qui concerne les logiciels développés dans ce cadre est aussi importante que pour les données
- Pas encore exigé par les financeurs mais aide très utile pour les projets
- Les questions ne se posent pas de la même façon que pour les données
  - Les données de recherche sont plutôt passives, les codes sont intrinsèquement vivants
  - Les codes s'appuient sur des dépendances et tout un environnement logiciel et matériel qui évolue sans cesse
  - Les codes représentent un travail de création, et correspondent à un cadre juridique différent de celui des données

Modèles de Plan de Gestion Logiciel

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

- PRESOFT project - également sur Hal
- Software Sustainability Institut

Un modèle est en cours de finalisation dans le cadre des ateliers de la donnée en partenariat avec DMP Opidor.



3 Les sections du plan de gestion logiciels
- Description du logiciel
- Outils et environnement d'exécution
- Préservation du logiciel
- Questions juridiques
- Valorisation scientifique

Description du logiciel

- Son nom !
  - Vérifier que le nom n'est pas déjà utilisé dans un contexte proche (voire que la marque n'est pas déposée auprès de l'INPI)
- Une description des objectifs du logiciel mais aussi le public potentiel visé.
- l'url du site web du logiciel ou du projet associé si elle est disponible
- Son année de début de développement
- Le domaine d'application / la thématique scientifique
- Des mots-clés
  - Idéalement issus de vocabulaires contrôlés ou de thésaurus
- La catégorie du logiciel (autonome, bibliothèque, plug-in, ...)

De l'importance des métadonnées

Les plans de gestion (données ou logiciels) « machine actionable » pourront permettre à terme de pré-remplir les fichiers de description des logiciels en particulier celui qui permet de faire le lien entre HAL et Software Heritage pour l'archivage et le référencement (voir la dernière partie)

3 Les sections du plan de gestion logiciels

- Description du logiciel
- Outils et environnement d'exécution
  - Dépôt du code source
  - Intégration continue
  - Documentation
  - Langages de programmation
  - Dépendances
- Préservation du logiciel
- Questions juridiques
- Valorisation scientifique

Outils et environnement d'exécution

- Faire les bons choix techniques
  - pour se faciliter la vie
  - pour travailler facilement avec les autres membres du projet
  - pour inciter à des contributions extérieures
- au niveau des outils, services mais aussi langages de programmation, dépendances ...
- Mais aussi anticiper des règles communes (nommage, architecture ...)

→ Réfléchir et décider d'un cadre technique, organisationnel, de gouvernance pour inscrire le développement dans la durée et éventuellement dans un contexte plus large que le projet de départ.

Gestionnaire de version

- Permet la gestion de l'ensemble des versions d'un ou plusieurs fichiers texte (donc en particulier du code source)
- Différents outils existants : git est le plus courant, mercurial, subversion ...

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 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/merge request)
- Simplifie les interactions entre les développeurs (tickets) et les utilisateurs (forums)
- Facilite la gestion des tests automatiques (intégration continue)

Forges disponibles

Rapport du collège logiciels et codes sources du COSO sur les forges

- Forges de l'ESR
  - De nombreux établissements, organismes voire 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

La question du choix de la forge

- La plupart des logiciels libres de l'ESR à périmètre international et/ou sociétal utilisent une forge commerciale.
- Facilite tous les types de contributions par tout le monde.
- **MAIS**
  - Pas de garantie de pérennité.
  - Souvent soumises à des juridictions extra-européennes.
  - Utilisation des sources sans réel contrôle pour l'apprentissage des outils d'aide à l'écriture de code (OpenAI Codex/GitHub copilot, GitLab suggestions, etc.).
  - Flou juridique sur la question du respect des licences pour ces outils.
- Il y a un enjeu majeur de **souveraineté**.
- Le collège logiciels et codes sources du COSO travaille actuellement sur des préconisations pour répondre à ces enjeux.

Intégration continue

Automatisation

L'intégration continue facilite la gestion de nombreuses tâches répétitives lors du processus de développement.

- Permet de détecter rapidement les erreurs grâce à des tests systématiques
- Permet d'améliorer la qualité du code par de l'analyse statique
- Permet de générer automatiquement la documentation ou le packaging de nouvelles versions ...

- Le choix de la forge contraint souvent le choix de l'outil de CI
- Anticiper l'organisation des tests, mais aussi les exigences de qualité des développements pour choisir les bons outils

Documentation

- Importance du README : c'est souvent la première page sur laquelle on tombe en général
- Le readme n'est pas la documentation de votre logiciel. Il ne doit donc pas être trop long, mais assez complet pour trouver de l'information rapidement.
- Différents types de documentation à destination d'acteurs différents :
  - documentation utilisateur (avec par ex un getting started, un tutoriel, des exemples, la description de l'installation ...)
  - documentation développeur (voire contributeur)
- Il existe pas mal d'outils pour faciliter l'écriture et la mise à disposition de la documentation (outils d'extraction automatique, gitlab/github pages, ...)
- Penser au choix de la langue utilisée !

Langages de programmation

- Le choix du ou des langages de programmation doit se faire de façon collective et va dépendre :
  - du type de logiciel (outil de simulation, application web, ...)
  - de l'expertise dans l'équipe
  - de l'usage dans la communauté concernée
  - du besoin en disponibilité et en portabilité sur différents systèmes d'exploitation
  - des besoins en performance
  - de l'écosystème et des bibliothèques disponibles
- Ne pas forcément se limiter à un seul langage : l'interfaçage entre un langage de bas niveau pour la performance et de haut niveau pour la faciliter de développement est souvent pertinent !

Classement des langages les plus utilisés

https://www.tiobe.com/tiobe-index/

Dépendances

- On n'écrit pas de code sans utiliser l'existant !
- L'utilisation de dépendances externes est intrinsèquement liée au développement de code
  - Ne pas réinventer la roue
  - Garantir une certaine fiabilité et qualité
- Il faut cependant les choisir avec discernement, identifier leur pérennité, assurer les mises à jour
- Bien documenter : sous forme de *requirements* et dans le processus d'installation
- Faciliter au maximum l'installation via l'utilisation d'outils dédiés : systèmes de packaging (conda, ou pour aller plus loin dans la reproductibilité, nix ou guix), conteneurs

3 Les sections du plan de gestion logiciels
- Description du logiciel
- Outils et environnement d'exécution
- Préservation du logiciel
  - Archivage pérenne
  - Référencement
- Questions juridiques
- Valorisation scientifique

Forges logicielles $\neq$ archives

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

MAIS

- Ce n'est pas une archive de codes
- Cela ne sert pas à pérenniser sur le long terme

Software Heritage : archive des codes sources

- Collecter l'intégralité des logiciels disponibles publiquement sous forme de code source
- Préserver : les codes sont stockés et accessibles sur le long terme
- Partager : les codes collectés sont organisés et indexés de façon à être référencés de manière optimale

Initiative à but non lucratif lancée en 20216 par INRIA et soutenu par de nombreux organismes, dont l'UNESCO
Membres fondateurs : Roberto Di Cosmo et Stefano Zacchiroli

Archivage et référencement

- Une collaboration entre Software Heritage et l'archive ouverte HAL a permis de développer le dépôt de logiciels pour favoriser le référencement et la description des codes sources.
  - Le dépôt y est simplifié en utilisant l'identifiant SWHID si le logiciel est déjà archivé dans Software Heritage
  - Il est possible de déposer un fichier sous format compressé du code qui sera ensuite pérennisé dans SWH si ce n'est pas le cas

- La notive HAL permet d'assurer la description du logiciel
  - Avec une modération des dépôts
  - Avec des métadonnées de qualité et vérifiées

Métadonnées : CodeMeta

- Format de métadonnées qui joue un rôle de pivot / table de concordance entre différents vocabulaires disponibles
- 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/` (mais aussi d'outils spécifiques pour les paquets R et Python).

3 Les sections du plan de gestion logiciels

- Description du logiciel
- Outils et environnement d'exécution
- Préservation du logiciel
- Questions juridiques
  - Auteurs et contributeurs
  - Licences
- Valorisation scientifique

Le cadre juridique du 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
  - Implication au niveau scientifique
  - 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.

3 Les sections du plan de gestion logiciels
- Description du logiciel
- Outils et environnement d'exécution
- Préservation du logiciel
- Questions juridiques
- Valorisation scientifique

Valorisation scientifique

- Penser à citer le logiciel dans toutes les publications qui y sont liées et, inversement, indiquer les DOI des publications dans le dépôt, le site web du logiciel ou sa notice HAL.
- De façon similaire, faire des liens croisés avec les autres produits de recherche liés (données en particulier).
- Etudier la possibilité d'une publication spécifique sur le logiciel (software paper).

Conclusions

- Nécessité d'inscrire dans le temps les développements réalisés dans les laboratoires : poser un cadre de développement et faire les bons choix techniques dès le départ !
  - Cela peut paraître contraignant mais consolide le socle du développement
  - Et permet de gagner du temps et de bien cadrer les travaux pour la suite
- Le plan de gestion logiciels permet d'anticiper les problématiques en amont
- Et de se projeter dès le début dans l'objectif de faciliter les contributions extérieures et la constitution d'une communauté autour du logiciel