[FLOSS_ESR] Rencontres sur les enjeux du logiciel libre dans la recherche (2022)

Synthèse atelier 1

Version 0.2

Sommaire

Version	Date	Auteurs	Commentaires
0.1	20/01/2022	Célya Gruson-Daniel	Rédaction atelier 1 (review Maya Anderson-Gonzalez) + envoi Bastien Guerry
0.2	26/01/2022	Célya Gruson-Daniel	Mise à jour ressources et typographie

1 | Atelier #1 – Contribution à la production de codes sources en recherche : quelles pratiques, quels enjeux ?

L'atelier s'est déroulé le vendredi 14 janvier 2022 de 9h à 12h30 avec une cinquantaine de participant·e·s dans le cadre des rencontres FLOSS_ESR

1.1 Rappel des ressources partagées lors de l'atelier aux participant·e·s

1.1.1 Informations pratiques

- [FLOSS_ESR#1] nformations utiles
- [FLOSS_ESR#1] Tutoriel de connexion & pratiques collaboratives
- [FLOSS_ESR] FAQ

1.1.2 Présentations courtes initiales

- Présentation des ateliers par Camille Maumet et Violaine Louvet (membres du collège logiciels libres et open source du Comité pour la science ouverte)
- Projet Dynare par Michel Juillard sous Licence CC-BY-NC-ND 4.0
- Évaluation des logiciels de recherche : protocole CDUR par Teresa Gomez-Diaz sous Licence CC-BY 4.0
- Pratiques de reproductibilité par Pierre-Antoine Bouttier sous Licence CC-BY-NC-SA 4.0

1.1.3 Notes des groupes de travail et des sessions plénières

- Notes session plénière
- Groupe 1 : Contribution aux logiciels libres et reconnaissance : évaluation, carrière
- Groupe 2 : Ouverture des codes sources et qualité en recherche/reproductibilité : pratiques, outils
- Groupe 3 : Intégration des codes sources dans l'écosystème de l'édition et de l'archivage scientifique
- Groupe 4 : Des premiers scripts à la naissance d'un projet open source : leviers techniques et humains
- Groupe 5 : De la sortie d'un projet open source à sa pérennisation : leviers techniques et humains

1.2 Résumé des constats et propositions d'action

1.2.1 Constats

1. Cadre conceptuel : Le logiciel est aujourd'hui reconnu comme un élément clef de la démarche scientifique et comme partie intégrante des productions scientifiques.

2. Cadre institutionnel : Le mouvement de la science ouverte souligne l'importance des logiciels libres/open source et de la reproductibilité mais sans qu'un accompagnement et des incitations concrètes de la part des institutions soient mis en place.

3. Pratiques culturelles : On assiste encore à un manque de reconnaissance de la part des institutions et des communautés scientifiques dans les carrières par rapport aux pratiques associées au logiciel (développement, contribution, aide à la diffusion, reproductibilité, etc.)

4. Contexte de création : Les codes sources et les logiciels en recherche représentent des objets protéiformes avec différents degrés de maturité et de soutiens, d'objectifs associés et d'intégration dans des communautés données (discipline, organisation, domaine). Une approche fine est nécessaire pour déterminer des modalités d'évaluation, de valorisation et de reconnaissance de ces objets.

5. Pratiques de production et d'évaluation : La production de logiciels est un processus dynamique (versions successives, etc.) qui implique une communauté mêlant des rôles et des types de contributions diverses. Cela s'inscrit dans des principes et des modalités d'évaluation spécifiques par rapport aux publications et aux données

6. Stratégie de pérennisation et valorisation : L'identification, le référencement et le dépôt des codes posent un ensemble de problématiques à adresser/affiner (articulation avec l'évaluation des publications, modération des dépôts, format de diffusion et règles juridiques).

7. Modalités de pérennisation et valorisation : Passer d'un script à un projet open source valorisable et perenne repose sur des leviers financiers et de ressources humaines pour faciliter le développement et le maintien de communautés en prenant en considération les spécificités des projets de recherche (souvent de niche).

1.2.2 Propositions d'actions

1. Penser le script dès sa création comme un projet qui sera un jour potentiellement ouvert et qui s'intègre dans une logique de reproductibilité.

2. Mettre en place de bonnes pratiques collaboratives de développement logiciel en s'appuyant sur un noyau d'outils existants.

3. Clarifier les différents types de contribution et y associer des modes de valorisation propres.

4. Sensibiliser/former les communautés à la culture libre et aux bonnes pratiques de développement logiciels, de préparation/mise à disposition/diffusion du code et de reproductibilité.

5. Développer des stratégies de pérennisation et de valorisation fondées sur l'identification et le référencement des logiciels, le maintien et soutien des communautés grâce à un cadre juridique, financier et de ressources humaine adaptés

1.3 Synthèse détaillée des constats et propositions d'action

1.3.1 Constats

1. Cadre conceptuel : Le logiciel est aujourd'hui reconnu comme un élément clef de la démarche scientifique et comme partie intégrante des productions scientifiques.

- À partir des années 2010, il y a une prise de conscience de l'appartenance du logiciel à la démarche scientifique.
- Aujourd'hui avec le numérique, les logiciels sont importants pour toutes les sciences.
- Le logiciel représente une connaissance exécutable qui peut avoir un impact en dehors de sa communauté pour la société.
- Dans ce contexte, l'open source permet d'avoir des contributions extérieures et sont les seuls logiciels à pouvoir répondre aux enjeux de reproductibilité.

2. Cadre institutionnel : Le mouvement de la science ouverte souligne l'importance des logiciels libres/open source et de la reproductibilité mais sans qu'un accompagnement et des incitations concrètes de la part des institutions soient mis en place.

- La mise à disposition des logiciels libres et open source doit se faire en articulation avec l'écosystème existant de l'édition et des principes FAIR.
- Le deuxième Plan National de la Science Ouverte met en avant la production et la diffusion des codes sources, mais il manque des incitations fortes de la part des employeurs et des organisations.
- La reproductibilité ne peut pas se faire sans logiciels libres et open source.

3. Pratiques culturelles : On assiste encore à un manque de reconnaissance de la part des institutions et des communautés scientifiques dans les carrières par rapport aux pratiques associées au logiciel (développement, contribution, aide à la diffusion, reproductibilité, etc.)

- Aujourd'hui, mis à part le domaine de l'informatique (INRIA, CNU section 27, etc.), le développement logiciel est encore souvent considéré comme un atout technique et non pas comme une vraie pratique scientifique numérique à mettre en avant.
    - Les développeurs de logiciels libres sont considérés souvent comme des techniciens notamment dans les secteurs non informatiques.
    - Il y a peu de reconnaissance dans la plupart des disciplinaires mais aussi dans le cas du secondaire.
- Souvent les développements de logiciels libres se font en plus des autres missions et ne sont pas pris en compte dans la carrière et dans le temps de travail, car ces tâches ne font pas partie de la fiche de poste.
    - Contre-exemple : INRIA section 6 production logiciel reconnu et ATD mis en place au sein de l'INRIA.

- Le développement logiciel, les activités de contribution, d'identification, de diffusion des logiciels (dépôts, etc.) tout autant que l'aide à la reproductibilité ne sont pas valorisés malgré le temps conséquent nécessaire à ces tâches parfois rébarbatives (par exemple le dépôt dans des archives)…
    - Les chercheurs ne voient pas comment ces activités peuvent les aider dans leur carrière.
- Beaucoup de communautés sont encore aujourd'hui réfractaires à la mise à disposition de codes sources de peur d'être jugé·e sur leurs codes et les erreurs éventuelles.
    - Il manque encore aujourd'hui un cercle vertueux à la publication du code et d'incitation à refaire des expériences d'un article existant.

4. Contexte de création : Les codes sources et les logiciels en recherche représentent des objets protéiformes avec différents degrés de maturité et de soutiens, d'objectifs associés et d'intégration dans des communautés données (discipline, organisation, domaine). Une approche fine est nécessaire pour déterminer des modalités d'évaluation, de valorisation et de reconnaissance de ces objets.

- Une question se pose sur la différence à faire entre logiciel et code source et le degré de maturité de projets.
    - Le code source contient intrinsèquement les algorithmes, et peut être relu. Par contre le logiciel peut contenir des données, un environnement, une documentation et peut plus largement faire référence à une communauté.
    - Il existe des projets avec différents degrés de maturité, avec d'un côté des "petits" logiciels et de l'autres des projets issus de plusieurs années de travail. Il s'agirait de les distinguer tout comme c'est le cas pour les différents types de publications aujourd'hui (*preprint,* publiés, etc.).
- Le cadre de développement d'un projet logiciel peut aussi bien être à l'échelle individuelle non financée tout autant que dans le cadre d'un projet ANR ou européen avec une plus grande visibilité.
    - Néanmoins, des initiatives individuelles peu visibles peuvent avoir un impact important à terme. Il s'agit donc de rester inclusif.
- Les objectifs de la mise à disposition du code peuvent être divers (enjeux de reproductibilité, création de communauté, travail dans une équipe restreinte ou individuelle, etc.)
    - Suivant les périmètres (communautés, laboratoires, etc.), les objectifs de la reproductibilité peuvent varier : entre refaire tourner un script, obtenir le résultat identique ou repartir du travail réalisé pour le dériver/étendre.
    - L'objectif de la mise à disposition est-elle pour une ouverture minimale (un README.md) ou bien un code ouvert pour attirer des personnes et des collaborations

5. Pratiques de production et d'évaluation : La production de logiciels est un processus dynamique (versions successives, etc.) qui implique une communauté mêlant des rôles et des

types de contributions diverses. Cela s'inscrit dans des principes et des modalités d'évaluation spécifiques par rapport aux publications et aux données

- Il faut distinguer différents types de contribution.
- À la différence d'une publication, il est difficile d'évaluer un logiciel qui peut-être amené à se développer (travail jamais fini).
- Pour la mesure de l'activité de production logiciels dans l'évaluation, il est important de ne pas retomber dans les travers de la bibliométrie. Les impacts de la production logiciels peuvent varier dans le temps, être décalés. Il s'agit de s'intégrer dans les démarches actuelles autour de l'évaluation qualitative (cf. DORA).
- Pour l'évaluation, il peut être difficile de trouver les évaluateurs qui ont les compétences et l'expertise pour évaluer le logiciel si les personnes sont d'un domaine différent.
- Il existe souvent une confusion entre logiciels et données ou logiciels et publications (avec l'application de principes FAIR qui n'a que peu de sens appliqué aux logiciels).
- Une question se pose sur le type d'évaluation lié aux publications : s'agit-il de faire une évaluation du logiciel au même titre qu'une publication ou bien comme une partie de l'évaluation d'une publication ?
    - Est-ce qu'il s'agit d'évaluer un simple dépôt sur une archive, un code utilisé/utilisable par d'autres personnes que ces auteurs, ou bien d'évaluer un code en profondeur (comme nous distinguons poster vs. publication dans une conférence et publication dans une revue).
    - Le code est considéré comme une métadonnée attachée à une publication. On évalue la publication éventuellement on regarde le code, mais il n'y a pas d'évaluation du code. Au mieux, quelqu'un cherche à vérifier que les résultats de la publication peuvent être regénérés avec le code référencé. Le code n'est pas évalué en lui-même.

6. Stratégie de pérennisation et valorisation : L'identification, le référencement et le dépôt des codes posent un ensemble de problématiques à adresser/affiner (articulation avec l'évaluation des publications, modération des dépôts, format de diffusion et règles juridiques)

- L'identification et le référencement des logiciels est une question complexe. Il faut distinguer l'identification d'un projet" (e.g. Octave) d'un "artefact logiciel" (e.g. la version précise utilisée pour une expérience). cf. https://www.softwareheritage.org/2020/07/09/intrinsic-vs-extrinsic-identifiers/.
    - Dans le cas de « projet » il s'agit d'utiliser des identifiants extrinsèques (et fragiles) comme :
        - une notice bibliographique modérée (HAL) ;
        - un objet identifié par un DOI, avec une liste d'auteurs extraite sans discernement de la liste des contributeurs d'un projet github (par exemple) ;
        - les ARK.

- Dans le cas d'artefact logiciel, il faut utiliser des identifiants intrinséques. cf. RDA/Force11 %20https://www.rd-alliance.org/group/rdaforce11-software-source-code-identification-wg/outcomes/use-cases-and-identifier-schemes.
    - Une référence vers le code archivé sur Software Heritage avec un SWHID. Les SWHID (*Software Heritage Identifiers)* généralisent et standardisent les "commit hash ».
- Aujourd'hui, la diversité des contributions possibles (développement, documentation, remontée de bugs, participation à la vision du logiciel, etc.) n'est souvent pas prise en compte lors du dépôt.
    - Pour Zenodo et github, une liste d'auteurs est générée à partir de l'ensemble des contributeurs, automatique, sans discernement sur l'importance de leur contribution ou la désirabilité de les mentionner comme auteur.
    - Se pose aussi la question dans les publications, de la citation à la fois de ceux qui ont contribué aux développements logiciels et de citer les outils employés.
    - Il faut travailler la notion d'auteurs, qui n'est pas la même selon les identifiants choisis (contributeurs mineurs, majeurs et participants à la direction de projets).
- Des plateformes existent pour déposer/archiver un code souvent sans processus de vérification/modération :
    - Sur HAL, il y a une modération des métadonnées du logiciel -> on va vers une vérification de la qualité de la notice a minima ;
    - Sur Zenodo, Github, SWH, pas de vérification .
- Q : Faut-il une complémentarité entre plusieurs outils HAL, SH, les revues ? Ou bien des plateformes de catalogage interne ESR (type PLUME / RELIER / FENIX) ?
- Se pose la question de ce qu'il est pertinent de diffuser et sous quel format et avec quelles licences :
    - Q : S'agit-il obligatoirement d'employer des licences libres ou bien un code librement copiable et utilisable dans un contexte de recherche est-il accessible de manière suffisante pour une démarche de science ouverte ?

7. Modalités de pérennisation et valorisation : Passer d'un script à un projet open source valorisable et perenne repose sur des leviers financiers et de ressources humaines pour faciliter le développement et le maintien de communautés en prenant en considération les spécificités des projets de recherche (souvent de niche).

- Les logiciels sont souvent produits par des doctorants ou post-doctorants sans perspective de pérennisation ce qui rend difficile l'évolution dans le temps du projet et la gestion du logiciel, son évaluation et financement.
    - PhDware : logiciel développé par un doctorant mais sans perspective de pérennisation.
    - Difficulté de l'évolution dans le temps avec les membres des équipes qui partent (PhD).
    - Difficulté gestion de logiciels, évaluation/financement / prise en compte dans les carrières.

- Le passage d'un script ou ensemble de scripts, de « coin de table » (plus ou moins élaboré) à un projet open source – où le code est diffusable et utilisable par tous – est associé à un ensemble de difficultés :
    - les programmes ont été écrits rapidement ;
    - les scripts sont dédiés à un cas spécifique ;
    - les programmes s'accompagnent souvent de peu de documentation, de test ;
    - le temps de la « robustification » n'a pas été valorisé.
- La réécriture du code avant diffusion peut-être chronophage et chère.
    - Cela est très chronophage notamment lorsqu'on fait appel à des externes spécialisés en programmation, car il y a un besoin de comprendre les principes scientifiques sous-jacents.
    - Il est plus « rentable » d'accompagner les chercheurs en amont pour débuter avec des bonnes pratiques.
- Les logiciels sont des objets qui évoluent en fonction des besoins de la communauté. Cela rajoute une difficulté à la maintenance sur le long terme.
- Le développement logiciel et sa pérennisation nécessite de prendre en considération le haut degré de spécialisation de logiciels en recherche avec des développements de niche
    - À la différence de projets associatifs qui peuvent être précaires mais avec un flux de contributrices et contributeurs, les logiciels en recherche s'adressent à des communautés très spécialisées.
    - Un champ d'application de niche peut rendre la création de communauté plus complexe. Le nombre d'experts est souvent réduit (avec une problématique de compétitivité). Les utilisateurs et utilisatrices ne sont pas forcément des contributeurs et contributrices et manque de compétences techniques.

1.3.2 Propositions d'actions

1. Penser le script dès sa création comme un projet qui sera un jour potentiellement ouvert et qui s'intègre dans une logique de reproductibilité.

- Inscrire le projet dans une infrastructure dès le début. Avoir au minimum une forge logicielle sur laquelle diffuser le travail avec si possible des commits réguliers
    - Quelle forge utiliser ? Forge institutionnelle ? Github/Gitlab ?
    - Question de l'accès aux extérieurs à ces forges ? (Doublons interne/externe ? Coûteux).
- Penser deux niveaux d'ouverture du code :
    - ouverture minimale du code avec un README ;
    - code ouvert pour attirer les gens, collaborer, diffuser. Ce qui implique un certain nombre de prérequis techniques :
        - instruction d'installation ;

- détails des fonctionnalités ;
- gestion de version propre ;
- branches séparées propres : *master/devel/feature ;*
- version (nécessaire pour reproductibilité, figer des versions) ;
- *release notes ;*
- système de tickets/*issues ;*
- *roadmap* : présenter où on veut aller et l'échéance.

2. **Mettre en place de bonnes pratiques collaboratives de développement logiciel en s'appuyant sur un noyau d'outils existants.**

- Bonnes pratiques de développement :
  - documentation minimale dès le début ;
  - tests dès le début qui représentent un gain de temps pour les développements futurs ;
  - processus de *code review* (pour produire du code "propre" au fil de l'eau) ;
  - processus de « c*oding style* » dès le début (ex : python pep8) ;
  - figer des versions de manière régulière pour des soucis de reproductibilité (Une publication = une version) ;
  - importance de référencer le code (telle branche, tel commit) utilisé.
- S'appuyer sur un noyau d'outils existants : git, forges logicielles, documentation, conteneurs, gestion d'environnement logiciel nix, etc.
  - Pour la reproductibilité, présence d'outils matures tels que Software Heritage, citation d'une *release* ou d'un morceau de code avec le SWHID, Zenodo, Nix-Guix pour la partie "environnement".
  - Prendre en considération la barrière d'entrée élevée à certains outils (e.g. git). Importance de simplifier les interfaces d'utilisation pour certaines communautés. Un *notebook Jupyter* est à la limite de ce qui est utilisable (déjà trop complexe pour certains utilisateurs).
  - Privilégier des outils avec une facilité d'installation et d'utilisation pour la reproductibilité, et une modification pour le long terme (dette technique).

3. **Clarifier les différents types de contribution et y associer des modes de valorisation propres.**

- Proposer quelques pistes de valorisation à destination de celles et ceux qui développent :
  - statut ambassadeur/dev ;
  - invitation à des évènements ;
  - organisation de prix ;
  - entretien carrière, poste à profil ;

- gratuité pour des formations ;
- heure de décharges.

- Distinguer différents types de contribution et les manières de les citer en lien avec les plateformes de dépôts, archivages :
  - développement ;
  - documentation ;
  - remontée de bugs ;
  - participation à la vision stratégique.
- Mettre en avant les logiciels dans le cadre des publications :
  - citation des personnes ayant contribué au développement logiciel ;
  - citation des outils utilisés.

4. Sensibiliser/former les communautés à la culture libre et aux bonnes pratiques de développement logiciels, de préparation/mise à disposition/diffusion du code et de reproductibilité.

- Sensibiliser à la culture du logiciel libre et de la mise à disposition des codes sources
  - Valoriser le temps consacré à la « robustification » (auprès des instances).
  - Promouvoir le libre est aussi une façon pour des personnes non informaticiennes de se familiariser avec les dynamiques communautaires et de les observer sur des projets importants.
  - Importance de communiquer sur les bienfaits du partage et de la mise à disposition.
  - Arriver à lutter contre la peur d'être jugé sur ses erreurs, qu'il ne faut justement pas les cacher. Ne pas avoir peur de publier / montrer son code.
  - Entraîner un regard bienveillant des communautés, car il manque encore un cercle vertueux de la publication du code.
  - Promouvoir des exemples qui marchent bien pour diffuser les bonnes pratiques à une communauté plus large que les développeurs.
  - Valoriser la reproduction de résultats scientifiques.
  - Sensibiliser au fait qu'il s'agit d'un investissement pour l'avenir (gain de temps pour la réutilisation si personne qui a écrit le code est parti par exemple).
- Mettre en place des formations et autres accompagnement avec les moyens humains nécessaires pour aider les chercheurs. Possibilité de formations :
  - aux bonnes pratiques de développement (documentation, *coding style*, tests, etc.) ;
  - aux outils employés (forge, *notebook*, etc.) en adaptant aux communautés (formations de base ou avancées) ;

- aux bonnes pratiques d'édition et de diffusion :
  - intégration dans une chaine de publication de corpus : liens entrepôt, bdd, logiciels de publication (mise en place de méthodologie commune) ;
  - aide à faire les dépôts sur HAL ;
  - savoir comment bien référencer son code de façon à ce qu'il soit trouvable et accessible ;
  - *executable papers* = *notebook* utile d'un point de vue pédagogique et de documentation (facilité la compréhension du code et son appropriation) ;

5. Développer des stratégies de pérennisation et de valorisation fondées sur l'identification et le référencement des logiciels, le maintien et soutien des communautés grâce à un cadre juridique, financier et de ressources humaine adaptés

- Assurer la pérennisation de projet logiciel et une augmentation de la reproductibilité par des ressources humaines dédiées.
- Faciliter des recrutements pérennes plutôt que de se baser sur un fonctionnement sur projets avec des contrats de travail à durée limitée et courte.
- Aider aux recrutements de nouveaux contributeurs et à leur montée en compétence.
- Valoriser les productions logicielles et communiquer autour de ses productions pour faciliter l'émergence et le maintien d'une communauté (s'il y a communauté, il y a pérennisation.
- Prendre en considération le besoin d'une taille critique d'un projet et les enjeux de gouvernance qui sont associés au projet.
- Aller chercher des sources de financement auprès d'agence ou de partenaire privé.
- Faire attention aux effets de modes pour les financements (e.g : financement fort pour l'IA en ce moment).
- Prendre en considération l'évolution des filiales de valorisation pour qu'elles accompagnent les projets open source. Cela nécessite un accompagnement juridique sur les modalités de valorisation économique adaptées (transfert des droits, protection des outils appartenant à l'employeur).
- Mettre en avant les gains en terme financier et de temps de la mise en œuvre de projets reproductibles et open source (éviter de reprendre le projet à zéro, s'appuyer sur l'existant, etc.).
- Améliorer les processus d'identification et de référencement des logiciels en prenant en considération les spécificités de production et contribution des communautés.

1.4 Ressources

1.4.1 Exemples de projets et d'initiatives :

- Logiciels qui ont le CNRS comme tutelle : https://haltools.inria.fr/Public/afficheRequetePubli.php?affi_exp=CNRS&typdoc=('SOFTWARE')&format_export=xml&langue=Anglais&CB_accent_latex=oui&CB_auteur=oui&CB_titre=oui&CB_article=oui&Fen=Rech
- Dynare sur :
  - Software Heritage https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://git.dynare.org/Dynare/dynare.git
  - Git : https://git.dynare.org/Dynare/dynare
- OpenMOLE : exploration de modèle Licence AGPL : https://openmole.org/ et https://trempoline.io/fr : propose services avec OpenMole
- Gargantex - analyse de grandes bases de texte (e.g. Twitter) : https://testing.gargantext.org/
  - https://testing.gargantext.org/) :
- OWNTECH – convertisseur électrique programmable sous licence Open hardware + Open software : https://www.owntech.org/) :
- Duniter : écosystème logiciel, crypto-monnaie blockchain -- toute une communauté non technophile : https://duniter.fr/
- Iota2- données spatiales satellites (open data) et publie des cartes de classification en libre. : https://framagit.org/iota2-project/iota2)
- JMLR : https://www.jmlr.org/mloss/)
- Mozilla science Lab : https://wiki.mozilla.org/ScienceLab
- OpenLifeScience : https://openlifesci.org/)
- Elixir europe – fédérer les logiciels et données en bioinfo à l'échelle européenne. FAIRisation des logiciels, statut de maintenance.: https://elixir-europe.org/
- Journeaux Ipol, JOSS avec revue complète du code et de sa réutilisabilité : https://www.ipol.im/ et https://joss.theoj.org/about
- Notion d'écosystème inspirée des travaux du HNLab sur un env. d'écriture et d'édition avec les données de la rech (data papers, executable papers) : https://hnlab.huma-num.fr/blog/2021/05/26/callisto-un-demonstrateur-jupyter/
- intervention lors du colloque DH Nord de 2021 – Mise en place d'une preuve de concept, "Callisto" : https://www.meshs.fr/page/vers_un_ecosysteme_d_ecriture_avec_les_donnees…-----58

1.4.2 Thématiques : citations/identifications/évaluation

- Article sur CDUR : https://f1000research.com/articles/8-1353/v2
- Identifiants extrinsèques/intrinsèques : https://www.softwareheritage.org/2020/07/09/intrinsic-vs-extrinsic-identifiers/
- Document de la *task force* RDA/Force11 https://www.rd-alliance.org/group/rdaforce11-software-source-code-identification-wg/outcomes/use-cases-and-identifier-schemes
- Inria pour l'évaluation du logiciel développée par les chercheurs https://hal.inria.fr/LORIA-ALGO/hal-03110728v1. Cet article de 2020 fait un point sur une partie de ces pratiques https://hal.archives-ouvertes.fr/hal-02135891
- https://hal.inria.fr/LORIA-ALGO/hal-03110728v1
- Par exemple, voir la citation proposée pour cette bibliothèque de calcul parallele : https://hal.inria.fr/hal-03516539/ (et le bibtex associe https://hal.inria.fr/hal-03516539/bibtex)
- https://hal.inria.fr/hal-03516539/
- Pour les utilisateurs de LaTeX : https://ctan.org/pkg/biblatex-software
- https://ctan.org/pkg/biblatex-software

1.4.3 Guides /Tutoriels

- Présentation Konrad Hinsen sur la reproductibilité : https://www.canal-u.tv/video/groupe_calcul/introduction_sur_les_principes_generaux_de_la_reproductibilite_dans_le_domaine_du_calcul.59563
- Tutorial : citing software using biblatex-software https://youtu.be/UhQCeAj9yKM
- Guides sur les licences par Etalab : https://www.data.gouv.fr/fr/pages/legal/licences/
- Page de la DINUM sur le sujet des attributions : https://www.numerique.gouv.fr/publications/politique-logiciel-libre/ouverture/#contenu
- Feuille de route 2021-2024 politique des données et algorithmes MESRI : https://www.enseignementsup-recherche.gouv.fr/fr/la-feuille-de-route-2021-2024-du-mesri-sur-la-politique-des-donnees-des-algorithmes-et-des-codes-50534

1.4.4 Publications

- Y. Barborini, R. Di Cosmo, Antoine R. Dumont, M. Gruenpeter, B. Marmol, A. Monteil, J. Sadowska.. La création du nouveau type de dépôt scientifique – Le logiciel. JSO 2018 – 7es journées Science Ouverte Couperin : 100 % open access : initiatives pour une transition réussie, Jan 2018, Paris, France. 2018. https://hal.archives-ouvertes.fr/hal-01688726⟩
- R. Di Cosmo, M. Gruenpeter, B. Marmol, A. Monteil, L. Romary, J. Sadowska. Curated Archiving of Research Software Artifacts : lessons learned from the French open archive. IJDC. 2020 (10.2218/ijdc.v15i1.698). https://hal.archives-ouvertes.fr/hal-02475835)

- R. Di Cosmo, M. Gruenpeter, S. Zacchiroli Referencing Source Code Artifacts : a Separate Concern in Software Citation, CiSE, IEEE, pp.1-9. 2020. (10.1109/MCSE.2019.2963148) https://hal.archives-ouvertes.fr/hal-02446202)
- P. Alliez, R. Di Cosmo, B. Guedj, A. Girault, M.-S. Hacid, et al.. Attributing and Referencing (Research) Software : Best Practices and Outlook from Inria. Computing in Science and Engineering, Institute of Electrical and Electronics Engineers, 2019, pp.1-14. ⟨10.1109/MCSE.2019.2949413⟩. https://hal.archives-ouvertes.fr/hal-02135891)
- A. Monteil, M. Gruenpeter, J. Sadowska, E. Nivault. Garantir la cohérence des données constitue le cœur de notre activité : entretien autour des enjeux descriptifs du code source. Bulletin des bibliothèques de France, École Nationale Supérieure des Sciences de l'Information et des Bibliothèques (ENSSIB), 2021, Dossier BBF 2021-1 • Code source : libérer le patrimoine !. https://hal.archives-ouvertes.fr/hal-03239502⟩
- Gruenpeter, M. et al. Defining Research Software : a controversial discussion. (2021) doi:10.5281/ZENODO.5504016. https://zenodo.org/record/5504016
- Vive le numerique libre", collection numérique pour mettre en avant les auteur.e.s principalement de l'ESR : https://www.amue.fr/fileadmin/amue/systeme-information/documents-publications/la-collection-numerique/N__13_-_Vive_le_Numerique_libre__fevrier_2021_.pdf