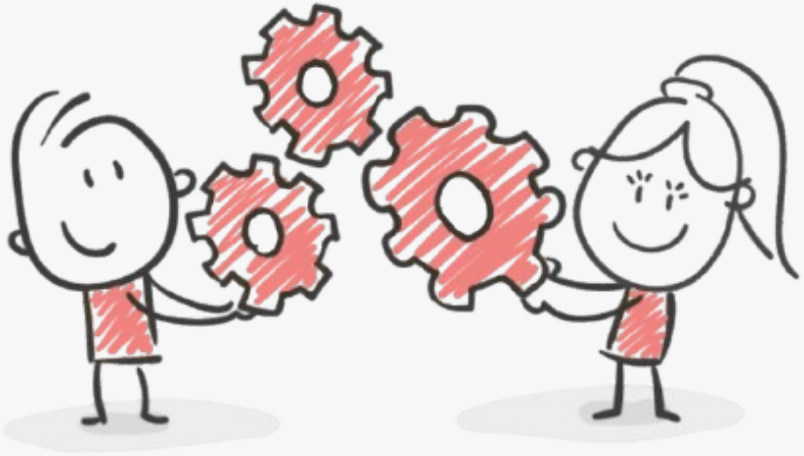
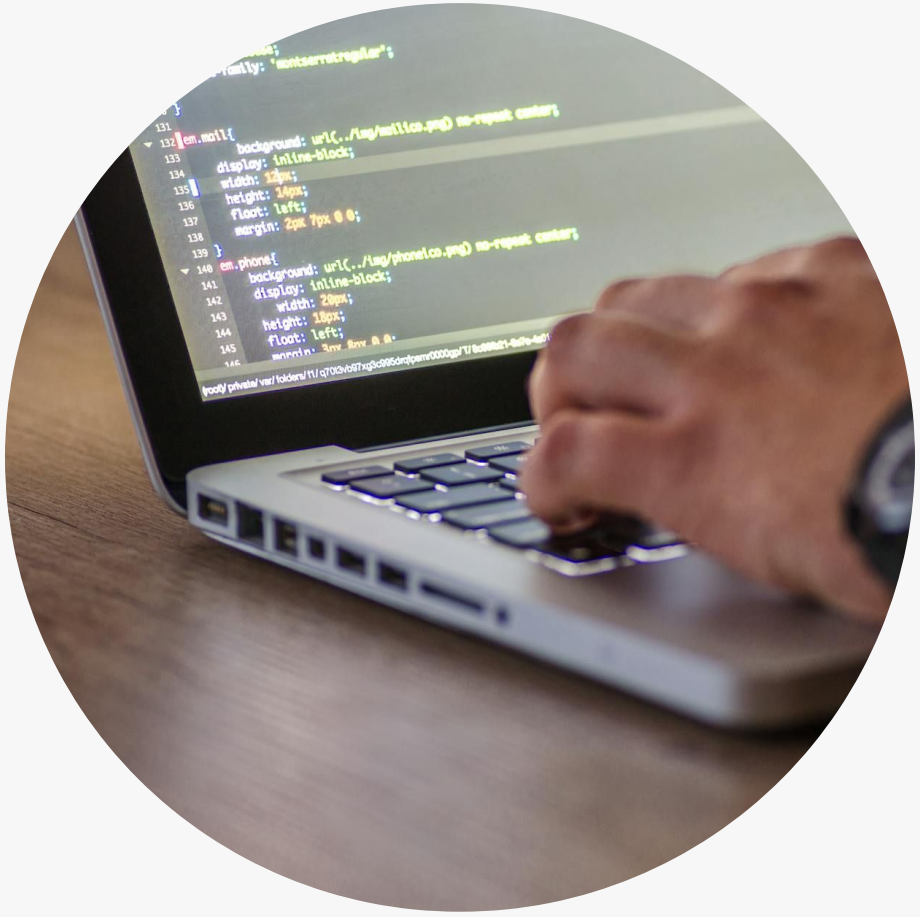
GÉRER

Les algorithmes, codes et logiciels tout au long de leur cycle de vie, du plan de gestion de logiciel à la sécurité, en passant par l'installation et le versioning.

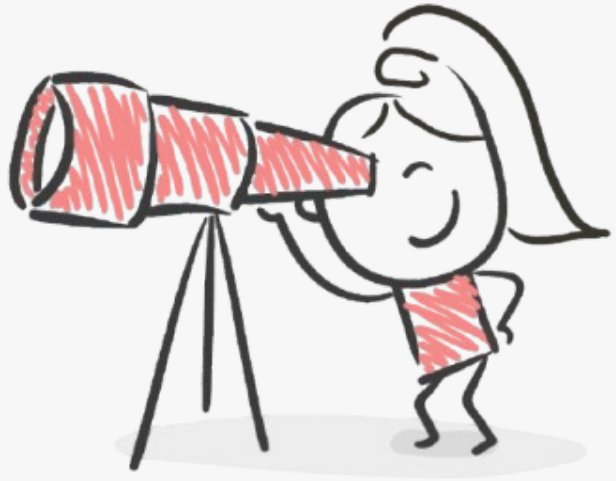
COMMENCER



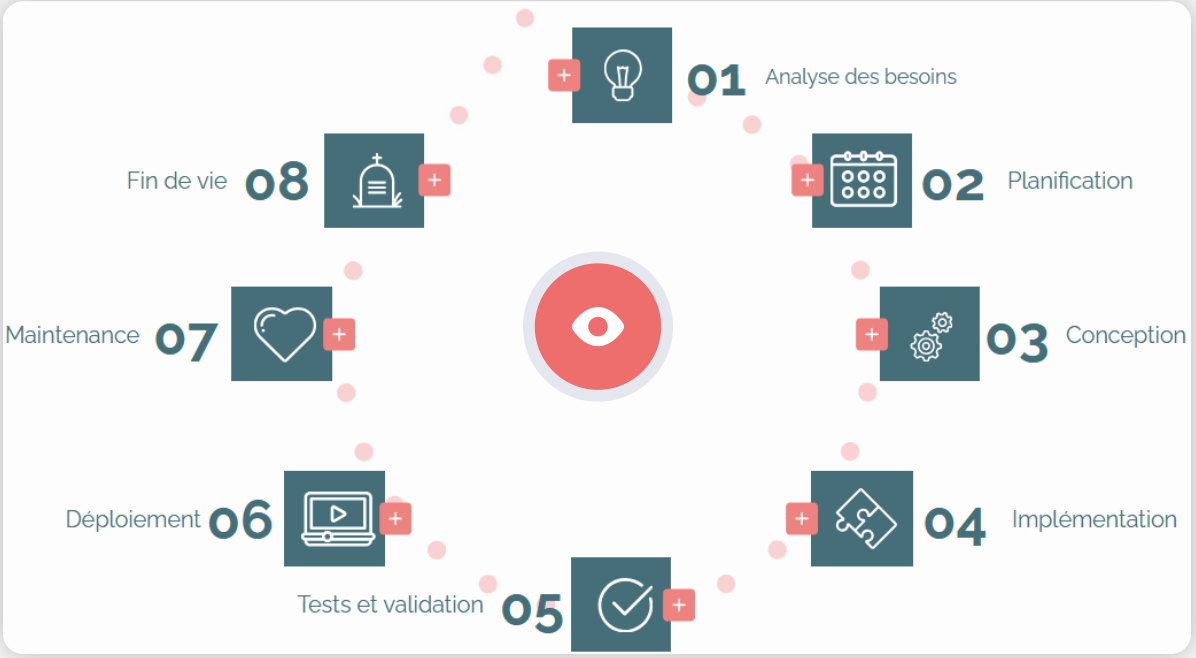


SOMMAIRE

Cliquez pour accéder directement aux parties.



Lorsqu'on veut s'atteler à **l'écriture d'un logiciel**, il est indispensable de **procéder par étapes**, en faisant appel à de la **gestion de projet spécifique au domaine de l'ingénierie logicielle**. C'est ce qu'on nomme communément le **cycle de vie du logiciel**. La construction d'un logiciel se déroule au travers de **diverses phases depuis sa conception jusqu'à sa fin de vie** rendant compte du cycle complet de développement :



Cycle de vie du logiciel

Cliquez sur les + et le ? pour en savoir plus

01 Analyse des besoins

02 Planification

03 Conception

04 Implémentation

05 Tests et validation

06 Déploiement

07 Maintenance

08 Fin de vie

Implémentation

Communément appelé développement, il s'agit d'écrire le code source du logiciel en regard de la conception précédemment établie, en réalisant également des tests garants de la cohérence et du bon fonctionnement des composants individuels du logiciel.

Analyse des besoins

Pour s'assurer d'un développement logiciel réussi, il est essentiel de définir de façon claire et précise les objectifs du projet. On parle ici d'expression des besoins ou des exigences des utilisateurs et des utilisatrices du logiciel en accord avec les concepteurs ou les conceptrices.

Fin de vie

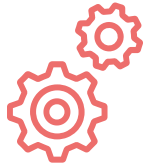
Il peut arriver qu'un logiciel puisse être mis au rebus ou décommissionné, en raison de son obsolescence ou de l'émergence de nouvelles technologies. Cette étape ne concerne pas tous les logiciels, et peut être optionnelle, car très liée au contexte.

Maintenance

Un logiciel utilisé conduit nécessairement à des ajustements qu'ils soient d'ordre correctif et/ou évolutif. Cela revient à traiter les dysfonctionnements, effectuer des mises à jour et intégrer les évolutions du logiciel, en réponse aux retours des utilisateurs et des utilisatrices et aux nouveaux besoins.

Déploiement

Cette étape représente la distribution officielle du logiciel auprès des usagers, qui peut s'opérer progressivement ou de manière simultanée, selon les impératifs opérationnels.

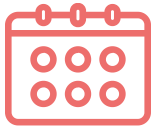


Conception

Cette phase sert à spécifier les fonctionnalités du logiciel, ainsi que la planification du développement à suivre.

Tests et Validation

Cette étape permet de vérifier de manière exhaustive l'opérationnalité du logiciel ; c'est à dire de vérifier son adéquation aux exigences précédemment établies. Il s'agit notamment d'établir divers tests, qu'on appelle d'intégration (tests fonctionnels, tests de performance), ainsi que d'autres validations.



Planification

Une étude de faisabilité permet de fixer une temporalité en accord avec les besoins et de définir le budget et les ressources utiles à la réalisation du logiciel.

LE PLAN DE GESTION LOGICIEL (PGL) OU SOFTWARE MANAGEMENT PLAN (SMP)

Concrètement, ça ressemble à quoi ?

Pourquoi rédiger un PGL ?

Tout comme le PGD (Plan de Gestion de Données), le PGL permet de prendre du recul, et peut constituer une aide à la réflexion et à la planification pour la mise en place des bonnes pratiques de développement et de gestion de code.

Globalement, c'est quoi ?

« Un document unique de référence qui décrit les (bonnes) pratiques mises en place lors du développement d'un logiciel. Il permet de centraliser les informations sur un logiciel dans un document unique. Ce document doit être régulièrement révisé pour vérifier que l'on suit bien les bonnes pratiques que l'on s'était fixées au départ. »
Source Openscience.pasteur

Quand faut-il commencer un PGL ?

Dès le début de votre projet de développement !
Il permettra de définir de bonnes pratiques qui pourront être communes à toutes les personnes du projet.

Comment le mettre en oeuvre ?

Différents modèles sont disponibles

→ ELIXIR

→ PRESOFT

→ SSI

Le modèle PRESOFT

PREservation of REsearch SOFTware du CNRS

Disponible dans DMP OPIDoR

Le modèle d'ELIXIR

ELIXIR est une **organisation intergouvernementale** qui rassemble des ressources en **sciences de la vie** de toute l'Europe.

https://elixir-europe.org/

SMP Elixir

Le modèle du Sofware Sustainability Institute SSI

Première organisation au monde dédiée à l'amélioration des logiciels de recherche

https://zenodo.org/records/2159713

Un **plan de gestion logiciel** bien structuré est indispensable pour **organiser et coordonner les activités liées au développement, à la maintenance et à l'amélioration continue du logiciel.**
Ce plan sert de **guide** pour toutes les parties prenantes impliquées dans le projet.

Le **modèle d'ELIXIR** est très simple et est donc utilisable pour tous types de projets logiciels dans le domaine des sciences de la vie. Il permet au scientifique de se poser les bonnes questions sur les bonnes pratiques à mettre en œuvre lors du développement et de la diffusion du logiciel.

Les **deux autres modèles** sont plus complets. Ils incluent également toutes les informations nécessaires pour permettre à d'autres de bien comprendre et réutiliser le logiciel : son objectif, ses fonctionnalités, sa version, sa durée de vie, les solutions prévues pour la maintenance…

LES DIFFÉRENTES DOCUMENTATIONS : NOTICES, GUIDES, ETC. 1/2

Installation, fonctionnement et utilisation d'un logiciel

Lorsqu'on gère un logiciel, il faut penser également à **son utilisation par une communauté d'utilisateurs et d'utilisatrices**. Les différentes documentations sont essentielles car elles fournissent des instructions détaillées sur l'installation, le fonctionnement et l'utilisation du logiciel. Afin de garantir une installation correcte et une utilisation optimale, elles sont cruciales pour accompagner le logiciel : **le guide d'installation, la notice de fonctionnement et le guide d'utilisation.**

Parfois, notamment lorsqu'on souhaite ouvrir le logiciel pour une communauté de collaborateurs et de collaboratrices, on s'emploie à créer un seul fichier nommée **README** (lisez-moi) qui devient la porte principale de la documentation d'un logiciel pour les utilisateurs, les utilisatrices et les développeurs et développeuses.

LE FICHIER README

Le fichier README

Ce fichier texte, que l'on trouve communément à la racine du projet, résume l'objectif du logiciel, son installation, son utilisation, à l'aide d'exemples simples. Il donne des informations sur les personnes ayant contribué à son développement et informe sur la licence d'utilisation.

Ce fichier possède une structure communément employée par les développeurs et développeuses, ce qui facilite la prise en main de nouveaux utilisateurs et utilisatrices.

Le README peut être utilisé dans d'autres contextes que la documentation de logiciels, comme par exemple pour documenter un projet, une donnée.

C'est un concentré d'informations à lire absolument !

Son format doit être numérique et ouvert (txt, markdown).

L'ART DU README

Exemple d'une structure de base d'un fichier README

Chacun des éléments ci-dessous peut être extrait dans un fichier à part, c'est souvent le cas de la licence, du journal des modifications (changelog) qui est généré de façon régulière. Tous ces éléments peuvent être accessibles à la racine du projet déposé sur une forge.

Overview : Présentation/Objectifs

Install : description de la procédure d'installation et des dépendances logicielles et matérielles requises (requirements)

Usage : cas basiques d'utilisation, avec des exemples fournis dans le logiciel. Un lien vers une documentation plus complète peut être fourni.

License : licence choisie et explicitation des conditions d'utilisations, de modification et de redistribution.

Contributing : éléments permettant d'indiquer si le projet accepte des contributions, de définir les règles mises en œuvre pour participer, pour remonter un bug...

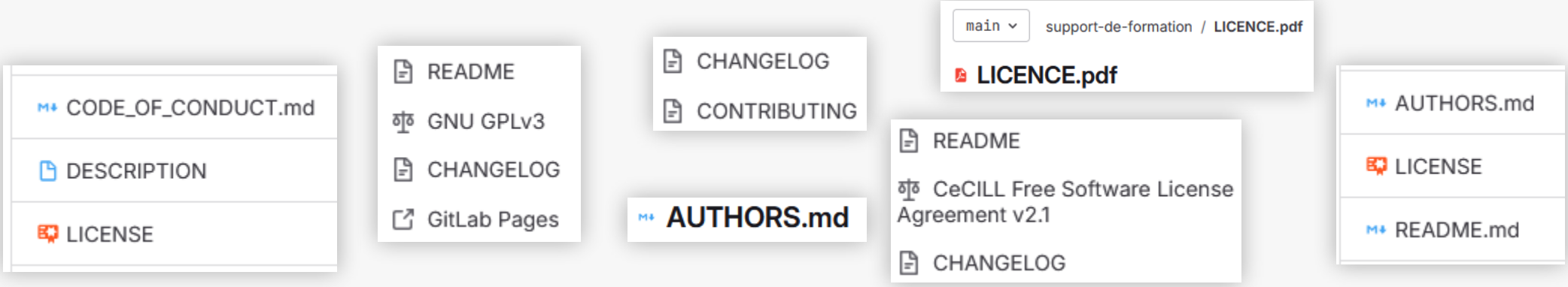
Changelog : fichier permettant d'expliciter et de suivre les modifications apportées dans une nouvelle version, afin de garder un historique des différentes phases de programmation du logiciel ayant amené à en sortir des versions distinctes.

Releases : c'est la publication du logiciel dans une version dite stabilisée.

LES DIFFÉRENTES DOCUMENTATIONS : NOTICES, GUIDES, ETC. 2/2

Par ailleurs, la **documentation technique qui s'adresse aux développeurs et aux développeuses** contribuant au code du logiciel pourra contenir des aspects de modélisation conceptuelle, d'architecture, d'implémentation de code, de construction et déploiement de nouvelles versions, et de tous les commentaires rajoutés au code afin d'en faciliter sa compréhnsion et son évolution.

Il existe **d'autres documents (License, Authors, ChangLog, Code of Conduct, etc.) déposés classiquement à la racine du projet logiciel.**



RENDRE LE CODE PLUS FACILE À INSTALLER ET À RÉUTILISER

La **reproductibilité** des résultats nécessite de donner à la communauté scientifique les moyens de réaliser à nouveau les expérimentations s'appuyant sur des logiciels. Or, il est presque **impossible de garantir qu'un logiciel s'exécute à l'identique**, pour tous les utilisateurs et utilisatrices, sur le long terme. L'utilisation de technologies permettant **le packaging d'application** peut être une partie de la solution. Nous vous proposons de découvrir Docker et Apptainer.

De manière générale, un code correctement conteneurisé sera toujours **plus reproductible** qu'un script laissé à l'abandon. L'usage de Docker, Apptainer ou d'autres solutions (comme GNU Guix et son complément GuixHPC) reste du ressort des mainteneurs, notamment selon l'environnement d'usage, les habitudes, ou les dépendances du projet.

Packaging d'application et conteneurisation...Késako ?

\+  **Docker**

Cliquez sur les technologies pour en savoir plus.

\+  **Apptainer**

Le packaging d'application est le processus qui consiste à assembler le code, ses dépendances et ses configurations dans un format prêt à être installé ou déployé sur un ou plusieurs environnements (serveur, poste de travail, cluster, etc.). Ce processus peut varier selon le langage, la plateforme et les contraintes d'exécution, et vise à garantir une installation fiable, reproductible et maintenable. Le but est de garantir que le logiciel pourra fonctionner correctement sans que l'utilisateur ou l'utilisatrice ait à reconfigurer manuellement l'environnement.

La conteneurisation est une méthode spécifique de packaging qui utilise des conteneurs. Un conteneur est une unité logicielle standardisée regroupant le code et toutes ses dépendances, permettant à l'application de s'exécuter de manière fiable, portable et isolée de son environnement d'exécution.

Packaging d'application et conteneurisation sont des pratiques qui visent à améliorer la reproductibilité et la portabilité des logiciels en isolant le code et ses dépendances dans des environnements contrôlés et cohérents.

L'utilisation d'Apptainer est notamment recommandée pour un déploiement sur des clusters de calcul.

https://apptainer.org/docs/user/latest/

Docker est bien maintenu et dispose d'une communauté large tant dans le monde de la recherche qu'au-delà. Il est relativement simple d'accès et déjà utilisé dans l'Institut pour du packaging de code, de bases de données, de scripts R.

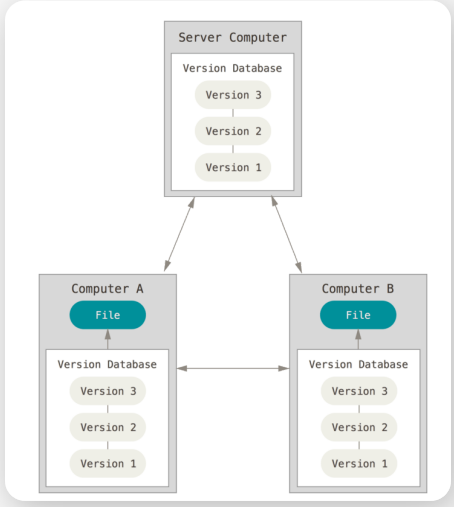
Découvrez ici une présentation de Docker et des mises en pratiques

VERSIONING ET RELEASES

Versioning ? Releases ?... Késako ?

Le versioning (ou gestion des versions d'un logiciel) et la gestion des releases sont des pratiques fondamentales pour assurer un suivi rigoureux des projets logiciels, garantir la traçabilité des évolutions du code et faciliter la collaboration entre les contributeurs et les contributrices. Ces méthodes permettent non seulement de documenter chaque modification apportée au code, mais aussi de structurer le développement de manière transparente et reproductible.

Gestionnaire de versions



Formalisation de releases

4.2.1

MAJOR *Minor* patch

SemVer en bref

Généralement, on distingue 3 niveaux de version d'un logiciel au moyen d'une suite de numérotation dite sémantique MAJEUR.MINEUR.CORRECTIF, qu'il convient d'incrémenter selon les types d'évolution :

1. MAJEUR - changements non rétrocompatibles : 1er nombre X.3.4
2. MINEUR - ajouts de fonctionnalité rétrocompatibles : 2ème nombre 2.Y.4
3. CORRECTIF - corrections d'anomalies ou de dysfonctionnements rétrocompatibles (bugfix) : 3ème nombre 2.3.Z

Par exemple, une version 2.3.4 d'un logiciel indique qu'il est dans sa 2nde version majeure, à laquelle 3 fonctionnalités supplémentaires ont été ajoutées, ainsi que 4 correctifs implémentés. Chaque nombre est incrémenté de façon croissante selon l'évolution de versions.

Des libellés supplémentaires peuvent être ajoutés pour les versions de pré-livraison et pour des méta-données de construction sous forme d'extension du format MAJEURE.MINEURE.CORRECTIF.

D'autres alternatives de numérotation existent mais sont toutefois moins utilisées, comme notamment la notation selon le mois et l'année de publication du logiciel (version 24.10 pour une publication du logiciel en octobre 2024).

La **formalisation de releases** (versions numérotées selon une stratégie claire, comme la gestion sémantique de version ou SemVer) marque des étapes clés dans le cycle de vie du logiciel.

Les numéros de release (ex : v1.2.3) permettent de :

- identifier précisément une version stable pour les utilisateurs et les utilisatrices,
- associer des notes de version détaillant les changements majeurs, mineurs ou correctifs,
- garantir la reproductibilité des résultats en liant une release à un état spécifique du code et de ses dépendances.

En intégrant ces pratiques, les équipes s'assurent que leurs algorithmes, codes et logiciels restent accessibles, compréhensibles et maintenables sur le long terme, tout en facilitant leur partage et leur réutilisation par la communauté scientifique.

Le **versioning** est un processus structuré et méthodique de gestion des différentes versions d'un logiciel ou de codes, permettant de suivre efficacement l'évolution d'un projet informatique (voir fiche *Focus* téléchargeable sous cette ressource).

Une **release** correspond à une nouvelle version d'un logiciel ou d'une application. Après un travail approfondi de développement et de perfectionnement incluant l'intégration de nouvelles fonctionnalités et la résolution de problèmes, les équipes de développement ont jugé cette version suffisamment mature pour être **partagée avec le comité utilisateurs.**

Une release permet donc de livrer des améliorations, des corrections ou de nouvelles fonctionnalités **de façon planifiée et contrôlée**. Chaque modification apportée au logiciel est intégrée de manière réfléchie, en suivant un processus rigoureux et structuré, permettant de **garantir la qualité et la fiabilité du produit final.**

Un **gestionnaire de versions** (comme Git, par exemple) joue un rôle clé dans le **suivi rigoureux des projets logiciels.**
Il offre une visibilité complète sur l'historique du projet, en enregistrant chaque modification (ou « commit ») avec des informations précises : auteur, date, message descriptif, et même les raisons des changements.

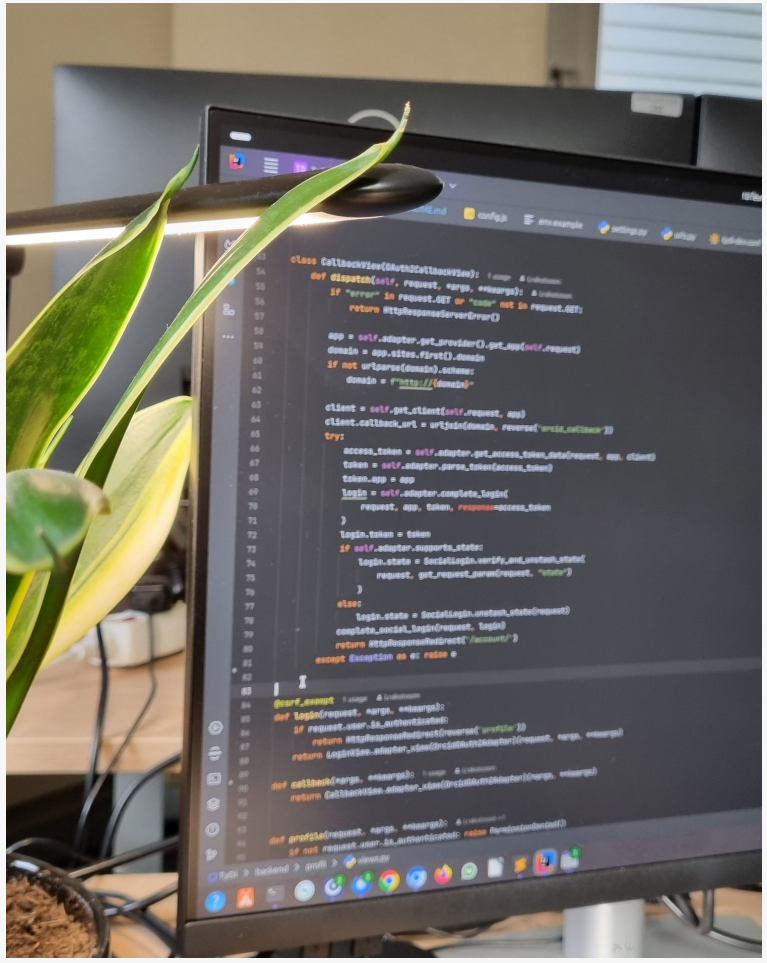
Cette traçabilité fine est essentielle pour :
• retracer l'origine des bugs ou des régressions,
• comprendre les choix de conception à posteriori,
• faciliter la revue de code et la collaboration entre équipes.

De plus, le gestionnaire de versions permet de **gérer des branches parallèles de développement**, ce qui est particulièrement **utile pour tester de nouvelles fonctionnalités ou corriger des bugs sans perturber la version stable du logiciel.**
Chaque branche représente une ligne d'évolution distincte, et leur fusion (« merge ») est documentée, assurant ainsi une traçabilité continue des évolutions.

PRODUCTION D'UN CODE DE QUALITÉ

Le cadre de la Science Ouverte promeut la production et l'ouverture des codes sources logiciels avec un **niveau de qualité permettant leur réutilisation**, conformément aux principes de la FAIRisation (voir la rubrique Ouvrir 3.3.). Pour cela, il est nécessaire de se concentrer sur plusieurs exigences-clés : la disponibilité, le fonctionnement valide dans leur contexte d'application, la reproductibilité des résultats, ainsi que la clarté, la maintenabilité et la pérennité des codes sources.

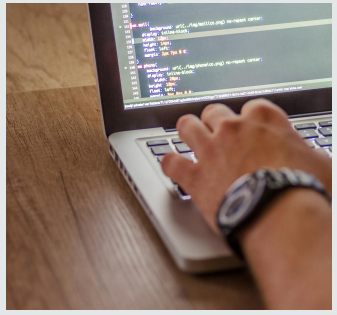
PRODUIRE UN CODE SOLIDE ?



SÉCURITÉ

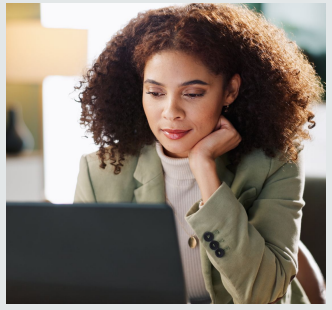
Ouvert ou pas ouvert, **la sécurité de vos algorithmes, codes et logiciels est importante** !

Développeur Développeuse



Utilisateur Utilisatrice

Responsable



En tant que développeur et développeuse, responsable ou simple utilisateur et utilisatrice, prenez connaissance des **diverses recommandations produites par l'équipe RSSI** (responsable sécurité des systèmes d'information) **de votre organisation .**

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

INRAE

Ce support a été créé dans le cadre de la formation à la science ouverte

OSCAR
OUVRIR LA SCIENCE, CONNAISSANCES À ACQUÉRIR

développée par la Direction pour la science ouverte d'INRAE

DipSO