

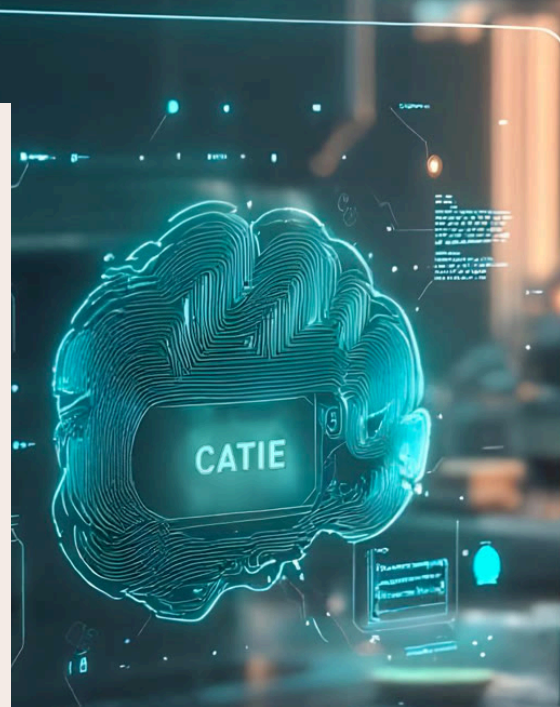


# CATIE

## Outils d'imagerie pour la robotique

Introduction au module

05/10/2026



# Qui suis-je ?

## Sébastien Delpeuch

- Ingénieur en informatique au CATIE, Bordeaux
- DevOps, Python et robotique
- Diplômé de l'ENSEIRB-MATMECA, Informatique, spé Robotique et Apprentissage

# Ce que je fais au quotidien

---

## DevOps

- Workflows GitHub Actions mutualisés
- Runners de CI auto-hébergés
- Cluster Kubernetes interne : Helm, Prometheus, Grafana

---

## Python

- Outils internes (FastAPI, React)
- Template de projet Python : uv, Ruff, pytest

---

## Robotique

- Coordination de projets clients et R&D
- Jumeau numérique (ROS 2, Gazebo, Docker)
- Équipe CATIE Robotics (RoboCup@Home)

# Le CATIE

- Créé en 2014, association loi 1901
- Centre de Ressources Technologiques labellisé
- Gouvernance : Région Nouvelle-Aquitaine, entreprises et acteurs académiques
- Mission : transférer la recherche vers les entreprises



Du prototype de labo au produit : le CATIE fait le pont.

# Trois domaines d'expertise

---

## Science des données

### **Tirer le meilleur des données.**

Construire, développer et valoriser les données, prédire.

---

## Facteurs humains

### **L'utilisateur avant la technologie.**

Évaluer l'acceptabilité, la désirabilité et l'engagement.

---

## Systèmes embarqués

### **Maîtriser le moteur du numérique.**

Concevoir, prototyper et tester logiciels et cartes électroniques.

# Le module TS341

Concevoir, de bout en bout, un logiciel de traitement d'images pour un besoin réel : comprendre le besoin, le découper, développer une solution propre, rapide et déployable.

---

## Imagerie

Clément Pinet  
Acquisition, traitement et analyse  
d'images  
5 séances de 2h

## Outils et méthode

Sébastien Delpéuch  
Python, packaging, qualité,  
conteneurisation  
5 séances de 2h

## Projet

Les deux volets réunis  
Travail en autonomie, encadré  
6 séances de 4h

Intervention extérieure : à venir, date et intervenant annoncés en cours de module

# Deux volets, un projet

## Imagerie · Clément

### Comprendre l'image

- Comment un capteur la produit
- Comment la rendre lisible
- Comment y trouver ce qu'on cherche



## Projet

### Un besoin client réel

- Décoder des fichiers bruts
- Les traiter vite
- Livrer un outil qui tourne ailleurs que sur votre machine



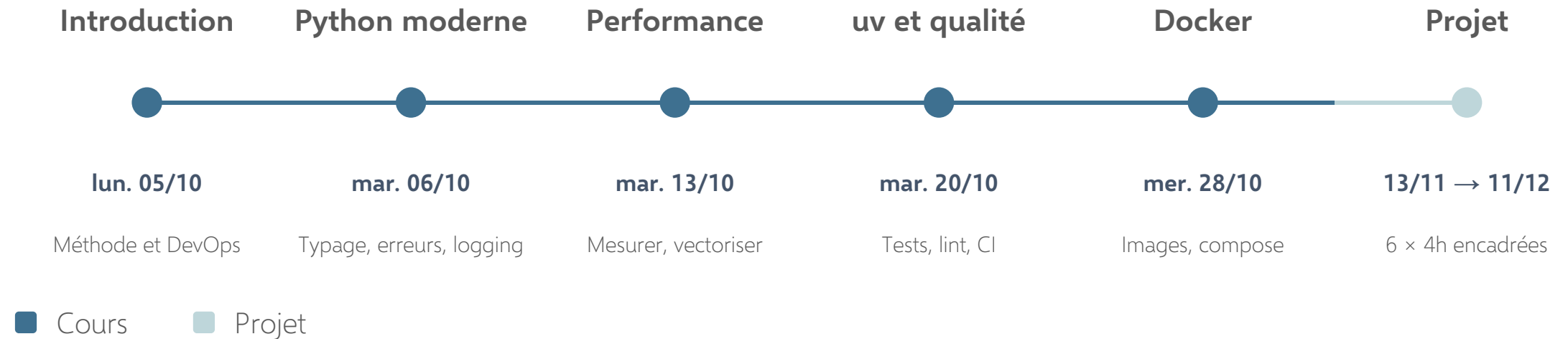
## Outils et méthode · Sébastien

### En faire un logiciel

- Lisible, typé, testé
- Rapide : mesuré, puis optimisé
- Livré : packagé, conteneurisé, vérifié en CI

Aucun des deux volets ne suffit seul : un décodage juste mais trop lent, ou un code propre qui produit de mauvaises images, ne répond pas au besoin du client.

# Mes séances



Les TP suivent un **fil rouge** : un même petit programme, amélioré d'une séance à l'autre. À la fin, vous avez le squelette propre de votre projet.

# Le site du cours

<https://ts341.tail8b1283.ts.net>

---

## Cours

Tous les diaporamas, de Clément et de moi, à lire en ligne ou en PDF

---

## TP

Le fil rouge à télécharger sous chaque séance, puis son corrigé

---

## Projet

Le planning, le sujet 2026 et ceux des années précédentes



# 1. Projet 2026

Le besoin, les jalons, et ce que vous rendez : un rapport de projet.

# Le projet

Un client fait de l'inspection visuelle automatisée de véhicules. Ses caméras produisent des fichiers bruts, sans aucune documentation. Il lui faut un logiciel qui les transforme en une archive d'images propre.

---

## Phase 1 · Décodage

Comprendre seuls ce que contiennent les fichiers bruts, et les découper en une archive propre

---

## Phase 2 · Optimisation

→ La spécification de l'archive attendue vous est donnée : produisez-la le plus vite possible

---

## Bonus · Floutage

→ Flouter les visages présents sur les images

La spécification est donnée à tous en phase 2 : un groupe bloqué en phase 1 peut quand même avancer.

# Ce que vous recevez

```
donnees/
├── 20260604073700_1_1FTEW3LP1TFB21247.eop
├── 20260604073700_1_1FTEW3LP1TFB21247/
│   ├── LFC1/
│   │   └── LFC1_79_1920x31600.raw
│   ├── LFD1/
│   │   └── LFD1_47_1920x37600.raw
│   ├── ...
│   └── UNDER2/
│       └── UNDER2_94_1920x37600.raw
├── 20260604072651_1_1FTFW4L58TFB35557.eop
├── 20260604072651_1_1FTFW4L58TFB35557/
└── ...
```

---

## Par dossier

Un fichier **.eop** à côté

44 sous-dossiers

Un **.raw** dans chacun,  $\approx 3$  Go en tout

---

## Documentation

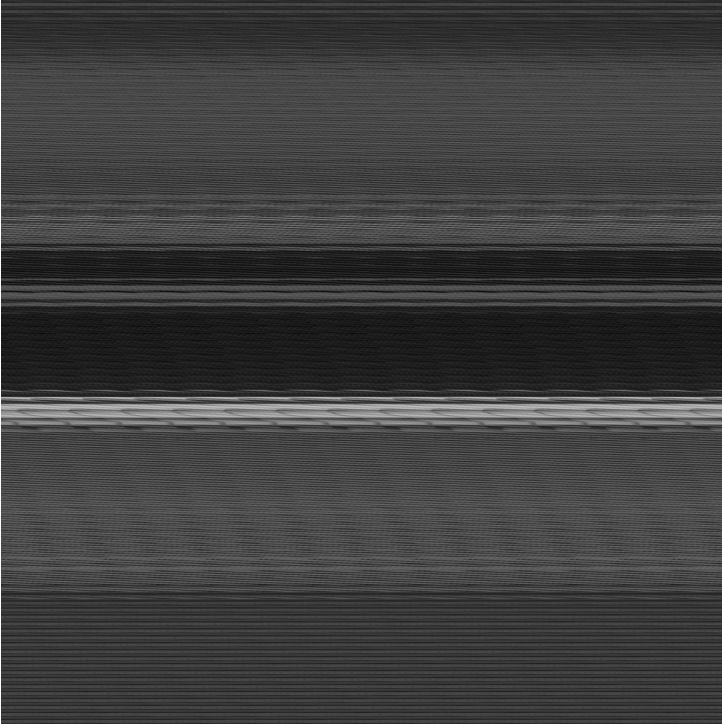
Aucune. Le nom des fichiers est le seul indice.

# Un raw : les données brutes

```
$ xxd LFS4_104_1920x41600.raw | head -12
00000000: 272a 2727 2728 2528 252a 2728 2c28 2c29  '***'(%('%*' (, (, )
00000010: 282a 292a 2c2a 272b 262c 222a 272a 2629  (*)*,*'+'&,"***&)
00000020: 272a 2a2c 292c 2c30 292e 2f28 282b 2728  '**, ),,0)./(('+(
00000030: 2a2d 282a 292e 2c2c 2a2c 292b 2c2d 2c2a  *-(*).,,*,),+,-,*
00000040: 2a2e 2c2d 2a2c 2c2b 2f2a 292d 2a2c 2c2f  *.,-*,,+/*)-*,,/
00000050: 2f2d 2c2d 2a30 2c2e 2e30 2c2e 2c32 2a2e  /-,-*0,..0,..,2*.
00000060: 2e2d 312b 2c2f 2e30 2c2e 2e2e 2c2f 2c2e  .-1+,/.0,...,/,
00000070: 2e2f 2c2d 2c2c 2e2d 2c2f 2a32 2f2e 312d  ./,-,...-,*2/.1-
00000080: 2c32 2e34 3131 2f2e 2e2d 2f31 3031 2a2f  ,2.411/...-/101*/
00000090: 2e32 3130 3232 3330 2f2e 352f 3330 3331  .2102230/.5/3031
000000a0: 3133 3131 3133 3132 3134 2c30 3133 3232  1311131214,01322
000000b0: 3133 2f35 3132 3331 2e33 3333 3134 3433  13/51231.3331443
```

Les 192 premiers octets d'un fichier qui en compte 79 872 000.

# Un raw : lecture naïve



Le même fichier, lu comme une image carrée : un octet par pixel,  $8937 \times 8937$ .

Il y a quelque chose dedans. À vous de trouver comment le lire.

# Confidentialité des données

---

## Vos obligations

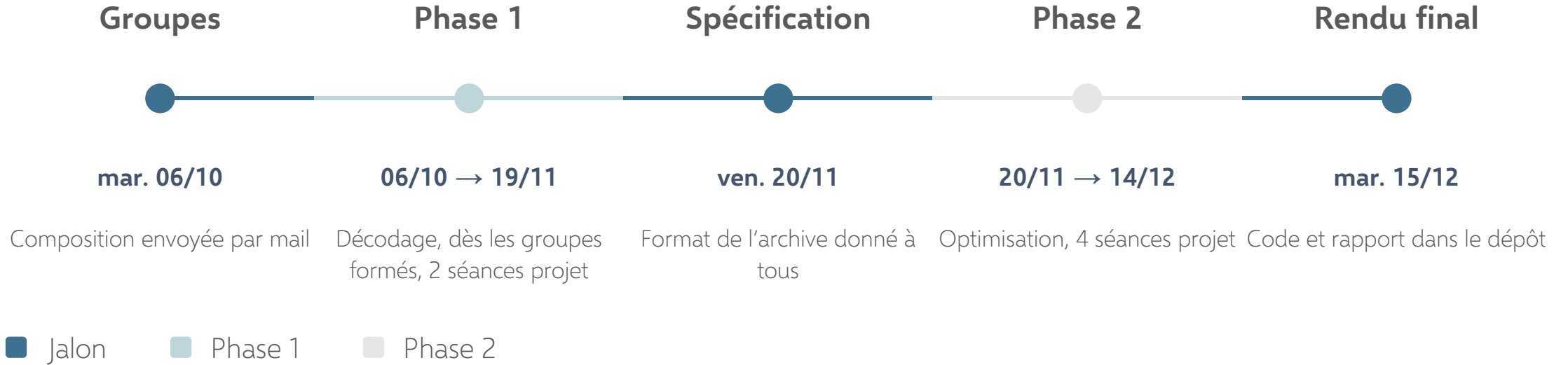
- Aucun partage hors du groupe et des enseignants
- Aucun dépôt ailleurs : Git, cloud, messagerie, outils en ligne
- Aucune image publiée, ni pendant ni après
- Dépôt GitHub **privé**
- Données supprimées à la fin du projet

---

## Ce que le client et le CATIE garantissent

- Données prêtées de manière désintéressée
- Votre code reste votre propriété
- Analysé par le CATIE, à des fins pédagogiques uniquement
- Le client n'y aura jamais accès

# Jalons



# Le rapport de projet

Un fichier **Markdown** dans le dépôt du groupe, court, qui couvre ces six points :

---

## 1. Besoin

Reformulez le besoin du client avec vos mots

---

---

## 2. Contraintes

Performances, matériel, formats, délais

---

---

## 3. Découpage

Les grandes briques et leurs interfaces

---

---

## 4. Choix techniques

Langages, bibliothèques, outils

---

## 5. Risques

Ce qui a bloqué, et comment vous l'avez contourné

---

## 6. Répartition

Qui a fait quoi, et quand

# Ce qui a manqué en 2025

---

## Constat

Cahiers des charges vides, ou choix techniques sans aucune justification

« OpenCV plutôt que scikit-image, car déjà vu avec Clément et plus rapide sur nos mesures » vaut mieux que « on utilise OpenCV ».

Rendu du rapport : **mardi 15/12**

---

## Attendu

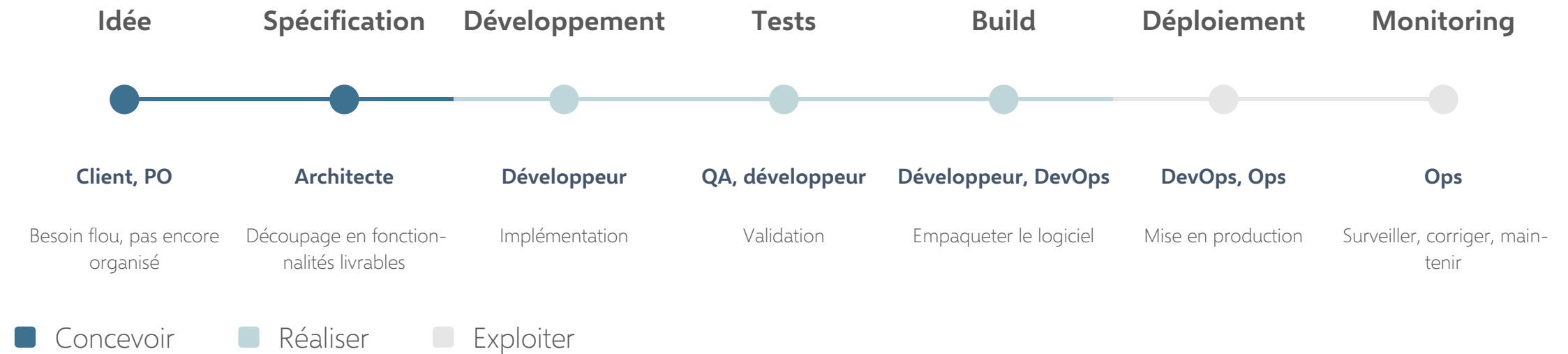
Chaque choix est comparé à au moins une alternative, avec un critère explicite



## 2. Processus de développement

Comment on passe d'une idée à un logiciel en production, comment on conçoit son code, et comment Git organise le travail à plusieurs.

# Du besoin à la production



Dans le projet, votre groupe tient **tous** ces rôles.

# Concevoir : les design patterns

Un **design pattern** (patron de conception) est une solution éprouvée à un problème de conception qui revient souvent. Ce n'est pas du code à copier, c'est un plan à adapter. Le catalogue de référence en décrit 23, en trois familles :

## Création

### Comment fabriquer les objets

- Factory Method : choisir la classe à créer au moment de l'exécution
- Builder : construire un objet complexe étape par étape

## Structure

### Comment assembler les objets

- Adapter : faire parler deux interfaces incompatibles
- Decorator : ajouter un comportement sans modifier l'objet

## Comportement

### Comment les objets coopèrent

- Strategy : rendre des algorithmes interchangeables
- Observer : prévenir des abonnés d'un événement

Catalogue illustré : [refactoring.guru/design-patterns](http://refactoring.guru/design-patterns)

# Les design patterns dans le cours

| Pattern        | Le problème qu'il résout                                              | Où vous le verrez                             |
|----------------|-----------------------------------------------------------------------|-----------------------------------------------|
| Strategy       | Changer d'algorithme sans toucher à ceux qui l'utilisent              | Les opérations du pipeline (Python moderne)   |
| Decorator      | Ajouter un comportement (mesure du temps, logs) autour d'une fonction | <code>@timed</code> (Python moderne)          |
| Iterator       | Parcourir une collection sans la charger en entier                    | Les générateurs (Python moderne)              |
| Factory Method | Choisir l'objet à créer d'après la configuration                      | <code>--executor threads</code> (Performance) |

Un pattern est d'abord un **vocabulaire** : « les opérations suivent un Strategy » se comprend en une phrase. Nommez-les dans la partie **Découpage** du rapport, et n'en mettez pas là où une simple fonction suffit.

# Gestion des sources avec Git

## Ce que ça apporte

- Historique complet des modifications
- Travail à plusieurs sur le même code
- Retour en arrière possible à tout moment
- Traçabilité : qui a changé quoi, et pourquoi

## Ce qu'on attend dans le projet

- Un dépôt GitHub par groupe
- Des commits petits et explicites
- Une branche par fonctionnalité
- Une Pull Request relue avant chaque merge

# Git en cinq mots

## Bases

| Mot     | Ce que c'est                                                          | Commande                                     |
|---------|-----------------------------------------------------------------------|----------------------------------------------|
| Dépôt   | Le dossier du projet et tout son historique                           | <code>git clone, git init</code>             |
| Commit  | Une photo du code à un instant, avec un message qui dit pourquoi      | <code>git add, git commit</code>             |
| Branche | Une suite de commits parallèle, pour travailler sans gêner les autres | <code>git switch -c</code>                   |
| Merge   | Réunir le travail d'une branche dans une autre                        | <code>git merge</code> , ou une Pull Request |
| Remote  | La copie partagée du dépôt, ici sur GitHub                            | <code>git push, git pull</code>              |

- Chaque commit pointe vers son parent : l'historique est un graphe, c'est ce que dessinent les schémas suivants (un point = un commit, une ligne = une branche)
- Tout reste sur votre machine tant que vous n'avez pas fait `git push`
- Le fichier `.gitignore` liste ce que Git ne doit jamais suivre : `donnees/`, `.venv/`, les images produites. À écrire **avant** le premier `git add .`, sinon les données confidentielles partent sur GitHub

# Écrire un commit : Conventional Commits

`type(portée):` ce que fait le commit

Pourquoi ce changement, si ce n'est pas évident.

| Type       | Usage                              |
|------------|------------------------------------|
| feat       | Nouvelle fonctionnalité            |
| fix        | Correction de bug                  |
| refactor   | Restructuration, même comportement |
| test, docs | Tests, documentation               |
| ci, chore  | Pipeline, outillage, dépendances   |

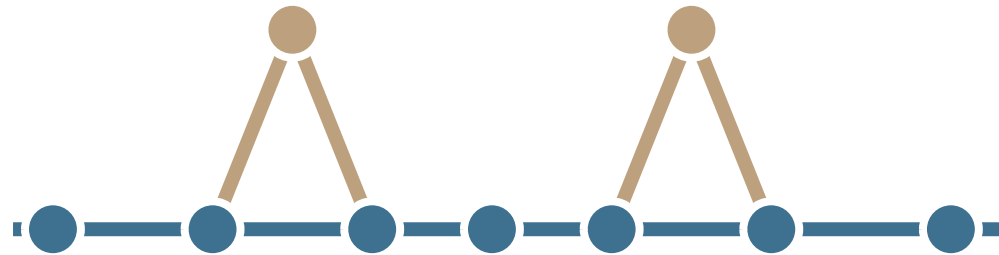
## Ce qu'on lit dans l'historique

- modifs, fix, wip, ça marche enfin
- + `feat(decodage):` lire un `.raw` dans un tableau NumPy
- + `fix(archive):` nommer les images par caméra
- Un commit = une idée : petit, et qui compile
- Le type donne la version suivante de **1.4.2** :  
`fix` → 1.4.3  
`feat` → 1.5.0  
**BREAKING CHANGE** → 2.0.0

# Trois façons de brancher : trunk-based

**courte**

**main**

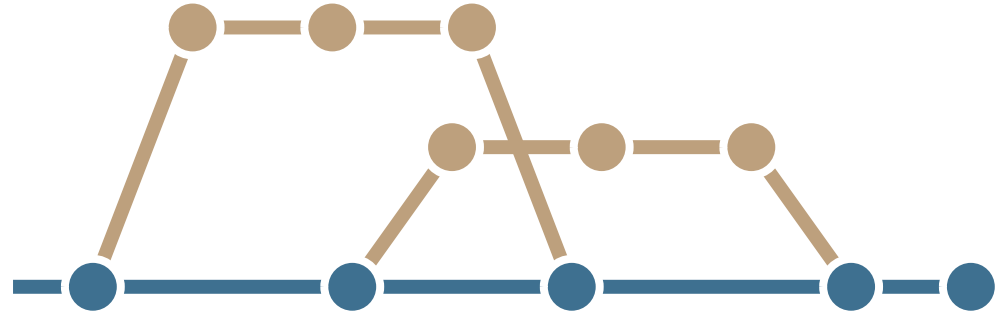


Tout le monde intègre sur **main**, plusieurs fois par jour.

- + Retours rapides, peu de branches
- Exige une CI solide et de la discipline

# Trois façons de brancher : feature-branch

feature  
feature  
main



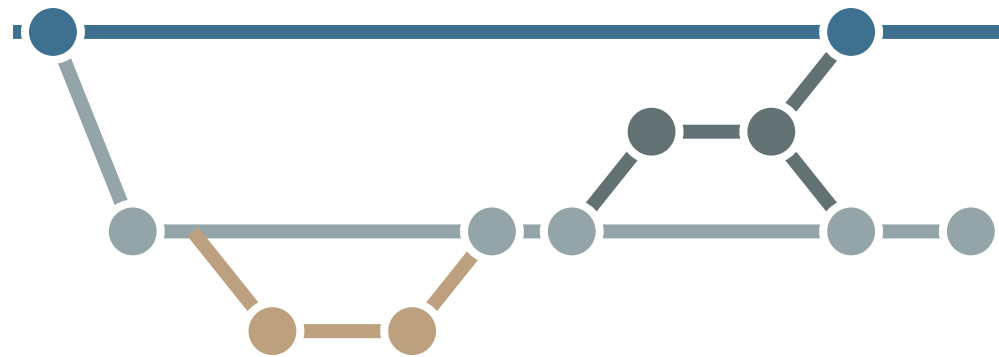
Une branche par fonctionnalité, fusionnée après revue.

- + **main** reste stable, revue avant intégration
- Une branche qui vit trop longtemps devient dure à fusionner

Pour le projet : **feature-branch**, avec des branches courtes.

# Trois façons de brancher : Git Flow

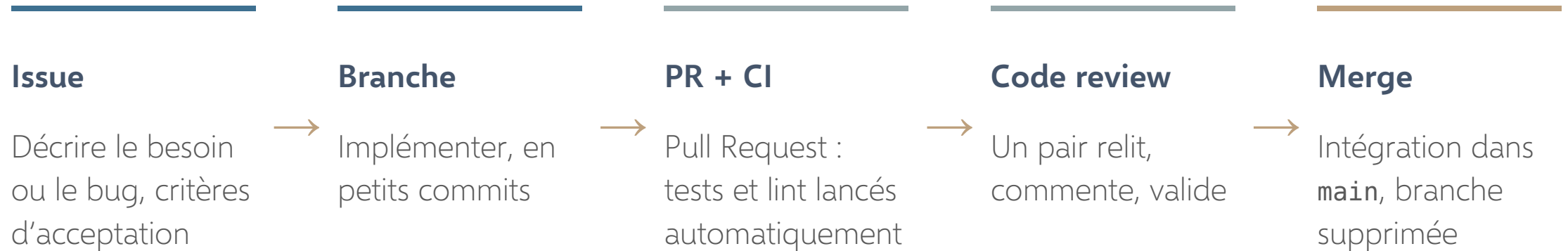
**main**  
**release**  
**develop**  
**feature**



Branches dédiées au développement, aux versions et aux correctifs.

- + Adapté aux logiciels livrés en versions
- Lourd pour une petite équipe

# Une fonctionnalité, de l'issue au merge



Personne ne pousse directement sur **main** : tout passe par une Pull Request, relue et verte en CI.



### 3. Philosophie DevOps

Pourquoi « ça marche sur ma machine » ne suffit pas, et ce que DevOps change.

# Le problème : un mur entre deux équipes

## Développement

### **Écrit** l'application

- Ne sait pas où ni comment elle tournera
- Livre un code qui « marche sur sa machine »

## Exploitation

### **Déploie et fait tourner** l'application

- Ne sait pas comment elle fonctionne
- Reçoit un livrable qu'elle ne maîtrise pas

Résultat : des livraisons rares et lentes, et chaque mise en production devient un risque.

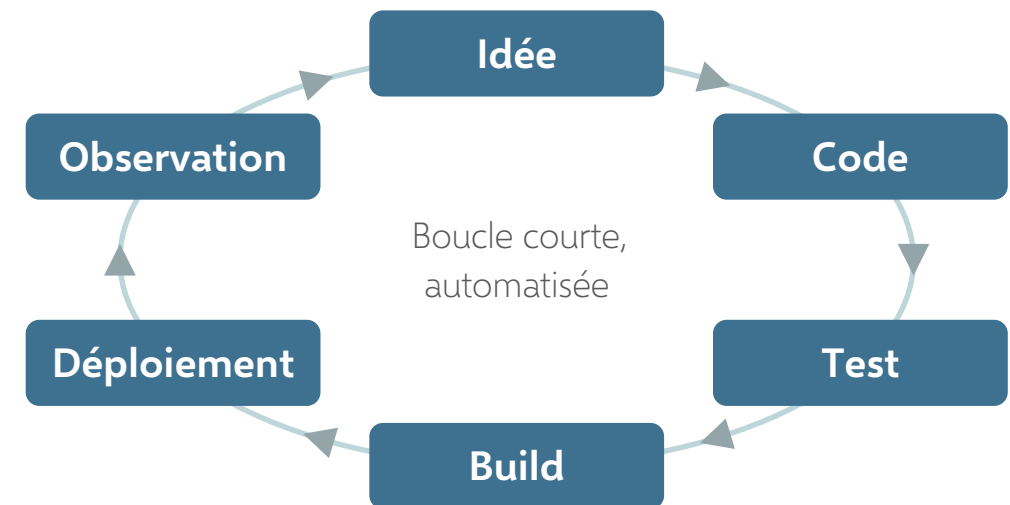
# DevOps : une seule boucle, parcourue souvent

Une même équipe est responsable du logiciel, de l'idée jusqu'à la production.

Objectifs :

- **Livrer vite**
- **Avec peu de bugs**
- **Recommencer souvent**

Le moyen : automatiser chaque étape de la boucle.



# Une culture avant des outils

---

## **You build it, you run it**

L'équipe qui écrit le code le fait aussi tourner, et le répare.

---

## **Automatiser par réflexe**

Ce qu'on fait deux fois à la main, on le script. Une machine ne s'ennuie pas et n'oublie pas d'étape.

---

## **Petits lots, livrés souvent**

Moins de changements par livraison, donc moins de risque et des retours plus rapides.

---

## **Post-mortem sans blâme**

Après un incident, on cherche la cause et on corrige le processus, pas un coupable.

# Tout est du code

| Ce qu'on décrit               | Fichier versionné        | Outil            |
|-------------------------------|--------------------------|------------------|
| L'application                 | src/                     | Python           |
| Les dépendances               | pyproject.toml, uv.lock  | uv               |
| L'environnement d'exécution   | Dockerfile, compose.yaml | Docker           |
| Le pipeline                   | .github/workflows/*.yaml | GitHub Actions   |
| L'infrastructure              | Manifests, charts        | Kubernetes, Helm |
| La configuration des machines | Playbooks                | Ansible          |

**Infrastructure as Code** : tout est dans Git, relu en Pull Request et rejouable. On doit pouvoir tout reconstruire à partir du dépôt.

# CI, livraison continue, déploiement continu

Commit

Build

Tests

Artefact

Staging

Production

**Intégration continue** : chaque commit est construit et testé

**Livraison continue** : chaque version validée est prête, la mise en production reste manuelle

**Déploiement continu** : tout ce qui passe les tests part en production, sans intervention

**Shift-left** : plus un problème est détecté tôt, moins il coûte. Lint, tests, audit des dépendances et recherche de secrets tournent dès la CI, pas la veille de la livraison.

# Un artefact, plusieurs environnements

## Build

Construire **une seule fois** : une image, un tag (v1.4.2)



## Dev

Intégration, données de test



## Staging

Copie de la production, validation finale



## Production

Les vrais utilisateurs

**Build once, deploy many** : on promeut le même artefact d'un environnement à l'autre, seule la configuration change. Reconstruire pour la production, c'est livrer autre chose que ce qu'on a testé.

# Observer ce qui tourne

## Logs

Ce qui s'est passé : des événements horodatés et structurés.

Ex. : **logging** (séance 2), Loki

## Métriques

Combien, à quelle vitesse : latence, débit, taux d'erreur, CPU.

Ex. : Prometheus

## Traces

Le parcours d'une requête à travers les composants, étape par étape.

Ex. : OpenTelemetry

Par-dessus : des **tableaux de bord** et des **alertes** (Grafana), pour être prévenu avant l'utilisateur.

Sans logs ni métriques, un logiciel est une boîte noire : on ne corrige que ce qu'on voit.

# Les métriques DORA

Pour aller plus loin

## Fréquence de déploiement

À quelle fréquence on met en production

## Délai de livraison

Temps entre un commit et sa mise en production

## Taux d'échec

Part des déploiements qui provoquent un incident

## Temps de rétablissement

Temps pour rétablir le service après un incident

Deux métriques de vitesse, deux de stabilité. D'après les études DORA (Google), les équipes les plus performantes sont les meilleures sur les quatre à la fois : aller vite ne se fait pas au détriment de la stabilité.

# Panorama des outils

| Étape                           | Outils courants                          | Dans ce module     |
|---------------------------------|------------------------------------------|--------------------|
| Versions et collaboration       | Git, GitHub, GitLab                      | Toutes les séances |
| Dépendances et packaging        | uv, pip, Poetry                          | Séance 4           |
| Qualité et tests                | Ruff, mypy, pyright, pytest, pre-commit  | Séances 2, 4       |
| Intégration continue            | GitHub Actions, GitLab CI                | Séance 4           |
| Conteneurs                      | Docker, Docker Compose                   | Séance 5           |
| Orchestration et infrastructure | Kubernetes, Helm, Ansible, Terraform     | —                  |
| Observabilité                   | Prometheus, Grafana, Loki, OpenTelemetry | —                  |

# Principaux concepts, et où on les voit

| Concept                       | Idée                                                           | Séance |
|-------------------------------|----------------------------------------------------------------|--------|
| Intégration continue          | Chaque modification est testée et vérifiée automatiquement     | 4      |
| Livraison continue            | Chaque version validée est empaquetée et publiable             | 5      |
| Tout est du code              | Dépendances, environnement et pipeline versionnés avec le code | 4, 5   |
| Environnements reproductibles | Mêmes dépendances, même résultat sur toutes les machines       | 4, 5   |
| Qualité du code               | Code lisible, typé, testé, relu                                | 2, 4   |
| Observabilité                 | Des logs exploitables pour comprendre ce qui se passe          | 2      |
| Mesure                        | On optimise ce qu'on a mesuré, pas ce qu'on suppose            | 3      |

## 4. Organisation

Ce qu'il faut installer, comment former les groupes, et comment vous êtes évalués.

# Configuration minimale

À installer **avant la séance 2** :

- Linux (recommandé), macOS, ou Windows avec WSL 2
- **Git** et un compte **GitHub**
- **uv** : il installera lui-même Python  
[docs.astral.sh/uv/getting-started/installation](https://docs.astral.sh/uv/getting-started/installation)
- **Docker** : Engine sous Linux, Desktop sous macOS / Windows  
[docs.docker.com/engine/install](https://docs.docker.com/engine/install) · [docs.docker.com/desktop](https://docs.docker.com/desktop)
- Un éditeur configuré pour Python (VS Code, PyCharm, Neovim...)

---

## Vérification

```
git --version
uv --version
docker run --rm hello-world
```

Les trois commandes doivent passer sans erreur.

# Travail en groupe

---

## Composition

- 3 à 4 personnes
- Au moins un·e **Informatique**
- Au moins un·e **Électronique**
- Au moins un·e **Cognitive (ENSC)**

---

## Constitution

- Groupes à envoyer à [s.delpauch@catie.fr](mailto:s.delpauch@catie.fr)
- Avant le **mardi 06/10**
- Plus de changement possible ensuite

# Évaluation

---

## **5 pts** **Participation**

En cours comme en  
séance projet

---

## **5 pts** **Phase 1 : décodage**

Passer des fichiers bruts  
aux images

---

## **5 pts** **Phase 2 : optimisation**

Traiter le plus vite possible

---

## **5 pts** **Qualité du code**

Typage, logging, tests, lint,  
CI, conteneur

La qualité du code s'évalue sur ce que les séances 2 à 5 vous apprennent.

# QUESTIONS ?

s.delpauch@catie.fr

<https://ts341.tail8b1283.ts.net>