LE CODE : UNE CONSTRUCTION COLLECTIVE

Forge | Codage | Workflow | Annonces

Antoine Pérus, IJCLab

La qualité d'un projet informatique ne se résume pas à la qualité du codage, mais dépend également de la *qualité des interactions collectives* au sein du projet et de leur mise en œuvre.

Comment améliorer notre pratique et notre code en renforçant notre conscience de l'importance du collectif ?

Il y a beaucoup de langues différentes.

Pour échanger et mieux se comprendre, il faut

- soit traduire, et c'est compliqué (traducteurs),
- soit partager un langage commun, des conventions communes.

C'est essentiellement l'objectif de ces "bonnes pratiques" :

- conventions communes
- règles de formatage communes
- design pattern
- framework

Langage commun ne signifie pas pensée unique !

Le langage commun est là pour faciliter les échanges.

Langage commun ne signifie pas pensée unique !

Le langage commun est là pour faciliter les échanges.

Collectif ne signifie pas nécessairement Commun

et

Commun ne veut pas dire Unique

Il faut "penser" le collectif et donc, la diversité du collectif.

Il faut que les contraintes inhérentes au processus soient :

- réfléchies,
- discutées,
- comprises,
- admises …

Remarque :

- Le "groupe" peut se résumer à une personne !
- Tout le monde en a fait l'expérience : on a parfois du mal à se relire ou à comprendre ce qu'on a fait …

  … quelques semaines ou quelques mois plus tard

Application à quelques outils, quelques pratiques …

Forge | Codage | Workflow | Annonces

LA FORGE

La forge "sociale" est l'outil central aujourd'hui dans un développement logiciel.

C'est bien le 1er espace dans lequel on doit parler un langage commun …

WIKI !!

Zone d'expression libre, oui … mais, pour être utile, doit être réfléchie et structurée collectivement …

LABELS

- Ce sont les étiquettes (couleur et texte) qu'on peut associer aux tickets, aux Merge/Pull Request (MR/PR) et qui permettent de catégoriser, et donc de structurer, ces derniers.
- Ils facilitent la lecture, le partage des tâches et donc la vision collective.
- Techniquement, ils peuvent participer à la définition du *dashboard* (Gitlab).
- Ils peuvent être partagés entre projets.

L'ensemble proposé par défaut est une bonne base, mais ça vaut vraiment le coup de l'enrichir de façon réfléchie selon l'environnement du projet …

Il existe des outils pour les importer et les gérer en ligne de commande.
(gitlab-standard-labels par exemple)

TICKETS

Un ticket peut être vu comme l'ouverture d'une discussion … qu'on va donc formuler (poliment 😉) et de façon compréhensible par tous ses lecteurs !

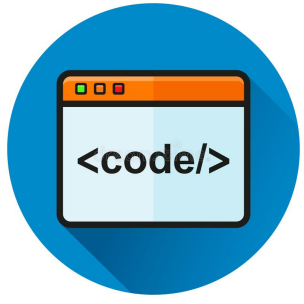
Le 1er cercle des lecteurs, ce sont les membres du projet; il est naturel de se mettre d'accord sur la façon de se parler de manière efficace !

Les tickets sont un espace privilégié pour échanger tant au sein du projet qu'entre les membres du groupe et le monde extérieur.

La conscience qu'on peut avoir de cet espace de discussion peut en permettre un usage collectif particulièrement riche et productif.

Pour aider à amorcer cette discussion à l'ouverture d'un ticket, on peut utiliser le mécanisme de templates proposé par la plupart des forges :

- cela permet de rédiger les tickets de façon similaire et cela facilite une rédaction complète.
- il est possible de définir différents templates selon le type de ticket que l'on veut ouvrir …



CODAGE

FORMATAGE

Une des façons les plus simples d'échanger du code entre développeurs est certainement de "parler" un dialecte commun, avec un "accent" partagé.

C'est l'objectif des conventions de code :

- formatage et présentation similaire
- règles de nommage

Ce besoin est tellement fort, que la communauté python, par exemple, utilise de plus en plus un outil comme black qui reformate d'une manière unique …

(la quasi seule liberté étant la longueur d'une ligne autorisée !)

TESTS

Les tests sont aussi (en plus de tester !) une excellente façon de montrer comment ça marche !

Typiquement en python, les doctests sont des exemples qui valent la peine d'être bien écrits, avec des noms compréhensibles … Souvent, ils peuvent remplacer efficacement les commentaires.

```
# calculations.py
# ...

def divide(a, b):
    """Compute and return the quotient of two numbers.

    Usage examples:
    >>> divide(84, 2)
    42.0
    >>> divide(15, 3)
    5.0
    >>> divide(42, -2)
    -21.0
    >>> divide(42, 0)
    Traceback (most recent call last):
```

```
# calculations.py

"""Provide several sample math calculations.

This module allows the user to make mathematical calculations

Module-level tests:
>>> add(2, 4)
6.0
>>> subtract(5, 3)
2.0
>>> multiply(2.0, 4.0)
8.0
>>> divide(4.0, 2)
2.0
"""
```

WORKFLOWS

BRANCHES / COMMITS

Démarrer une branche de développement, c'est commencer une nouvelle histoire en proposant quelque chose !

Souvent, elle est associée à un ticket, avec donc un dialogue probablement déjà entamé dans celui-ci.

Si on ouvre simultanément une Merge Request (marquée Draft:) associée à cette branche, le dialogue peut se poursuivre !

Si on ouvre simultanément une Merge Request (marquée Draft:) associée à cette branche, le dialogue peut se poursuivre !

On y participe que ce soit :

- pour demander conseil,
- pour expliquer au fur et à mesure le travail, l'avancement,
- pour proposer son point de vue.

Les messages de commit sont des éléments essentiels de la narration soutenue par la branche.

Les messages de commit sont des éléments essentiels de la narration soutenue par la branche.

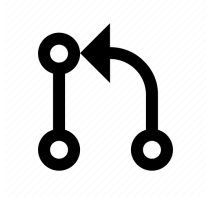
L'outil de gestion de version saura dire :

- *Qui ?*
- *Quand ?*
- *Quoi ?*

mais il ne pourra jamais raconter le Pourquoi de chaque évolution, de chaque détail de l'histoire !

Les conventions et les templates sont ici, particulièrement utiles.

À tel point, que certains projets font valider (par des outils) le format de ces messages de commit et n'acceptent ceux-ci que s'ils respectent le format et les conventions convenus.



PULL/MERGE REQUEST

Qu'il ait été ouvert en même temps que la branche associée ou pas, la *Merge Request* est d'abord un espace de discussion collective.

C'est une proposition de contribution qui reste ouverte à la discussion !

On peut donc y échanger et y défendre son point de vue.

Par courtoisie, et par souci d'une discussion civilisée, on veille à ne discuter que d'un seul sujet dans son *Merge Request* … pour éviter la cacophonie !

Par courtoisie, et par souci d'une discussion civilisée, on veille à ne discuter que d'un seul sujet dans son *Merge Request* … pour éviter la cacophonie !

Si le besoin de diverger ou de rebondir sur un autre sujet se présente, les outils modernes permettent d'ouvrir une nouvelle *Issue* directement depuis un fil de discussion dans une MR.

REVUES DE CODE

On dit que la *Revue de Code* est indispensable, fondamentale, comme peut l'être le test, dans la chaine qualité … Pourquoi pas !

Ce qui fait qu'une *Revue de Code* est efficace et n'est pas un simple formalisme, voire une perte de temps, c'est de la concevoir comme une discussion (fructueuse) …

L'objectif d'une bonne *Revue de Code* :

- ✅ améliorer la qualité finale
- ✅ partager la connaissance du projet
- ✅ augmenter l'expertise de chacun
- ✅ prévenir l'augmentation de la dette technique, voire la diminuer

passe par des échanges les plus explicites possibles, cadrés par des choix collectifs.

Cette discussion, entre 2 acteurs au départ, peut cependant être partagée le plus collectivement possible.

Le code écrit dans le projet est une œuvre partagée, les *Revues de Code* sont l'un des outils qui permettent ce partage.

En ayant à l'esprit qu'il s'agit d'une réflexion collective, on trouvera naturel, par exemple, de résumer la *Merge Request* à l'origine de la *Revue de Code* avec :

- ✅ la description du contexte pour comprendre ce qu'il se passe
- ✅ la description du bug ou de la proposition
- ❗ qu'est-ce que modifient les changements proposés
- ✅ comment sont-ils testés (description, plan de tests) ?
- ⚠️ les limitations de la proposition, qu'est-ce qui manque ?

Là encore, utiliser des templates !

ANNONCES

Lorsqu'on présente un projet, on a tout un tas de moyens pour raconter notre histoire de développement et convaincre de l'utiliser le plus agréablement possible !

Ce sont, par exemple, les fichiers "conventionnels" :

- README
- CHANGELOG
- LICENCE
- CONTRIBUTING

Tous, peuvent être vue comme une présentation du projet par l'équipe du projet.

README

C'est souvent l'un des points d'entrée qui signe l'accueil !

- Soit, on souhaite la bienvenue courtoisement avec toutes les informations utiles, mais pas plus,
- Soit, on risque de passer pour un projet inachevé, pas mûr …

Des templates existent, il y a même des sites pour rédiger un README complet, exhaustif, efficace …

CHANGELOG

Une bonne façon pour qu'il soit "intéressant" et utile à la collectivité est de le générer automatiquement à partir des messages de commit,

… si bien sûr, ces derniers sont à la hauteur de l'enjeu et ont été rédigés en suivant des conventions décidées collectivement et appliquées.

Pour GitLab, voir :

- Changelogs
- Changelog entries

Voilà !

Ce tour d'horizon de quelques outils et pratiques utilisés dans des projets de développement logiciel a permis d'insister sur l'aspect discursif collectif qui est toujours sous-jacent derrière leur utilisation.

L'idée étant que la qualité d'un projet informatique ne se résume pas à la qualité du codage ni à la quantité d'outils pointus utilisés, mais dépend également de la qualité des interactions collectives au sein du projet.

Qu'il faut y veiller,

et que ça en vaut la peine !

UN DÉFAUT DE DISCUSSIONS PRÉALABLES ??

MERCI !