# Projet Génie Logiciel

Deuxième Année du Cursus Ingénieur - 2023 - Groupe 1 GL3

—


Jorge Luri Vañó

Nils Depuille

Vianney Vouters

Virgile Henry

Logan Willem

Dorénavant surnommés “membres”

## Valeurs de l'équipe

- Maintenir une bonne ambiance générale pendant l'entièreté du sujet
- S'entraider le plus possible pour faciliter le travail mutuel
- Communiquer toutes les modifications et travaux réalisés
- Droit à l'erreur
- Présence durant les horaires de travail et motivation pour travailler en dehors
- Pouvoir communiquer les inquiétudes sans être jugés
- Transparence dans l'organisation du travail
- Obligation de moyen pour avancer le projet en dehors des heures de travail obligatoires

## Respect de la charte d'équipe

Les membres de l'équipe devront respecter la totalité des articles de la charte, ayant été validée et acceptée par tous avant le début du projet.

Toute infraction des articles de la charte ci-présente implique une sanction au membre infracteur.

Les sanctions seront différentes selon le degré de l'infraction et seront mesurées en nombre de points.

Chaque membre cumule un nombre de points-sanctions. Lors d'une infraction, les autres membres décideront le nombre de points-sanctions à attribuer.

- Tous les points : un café ou un chocolat offert pour chaque membre
- Tous les 3 points : un petit déjeuner offert pour chaque membre
- Tous les 5 points : une tournée de bar offerte
- Tous les 15 points : Menu Tacos offert à tous les membres

Exemples de sanctions :

- Arriver entre 5 et 20 minutes en retard : 1 point
- Arriver entre 20 et 45 minutes en retard : 2 points
- Arriver entre 45 minutes et 3 heures en retard : 3 points
- Arriver plus de 3 heures en retard : 5 points
- Push un commit sur develop ou master qui ne compile pas alors qu'avant ça compilait : 3 points
- Pusher un commit dans develop sans documenter : 1 point

# I. Usage du Git

## 1) Organisation des branches

Le Git sera structuré de la manière suivante :

- master
- develop
- Branches features

Les branches features seront générées par Git Flow. Une feature correspond à un sprint.

La nomenclature des branches features sera la suivante :

etape-<lettre etape>-<nom feature>

## 2) Branches features

Chaque membre devra travailler dans sa branche feature. Le membre est responsable du code présent dans sa branche. Si un autre membre modifie une branche feature, le responsable d'une faille sera toujours le créateur de la branche.

## 3) Modification d'une branche des autres

Aucun membre ne pourra modifier la branche d'un autre membre sans son approbation directe. Si le membre invité accède à une autre branche pour modifier le code, sa modification sera restreinte au préavis annoncé au propriétaire.

## 4) Description des commits

Tout commit dans une branche dont le membre n'est pas propriétaire devra être accompagné d'une description exhaustive des modifications réalisées, afin de tenir informés le membre propriétaire.

## 5) Commit dans Develop et Master

Tout commit dans Develop et Master devra compiler si les commits précédents compilaient. Les nouveaux commits devront, dans la mesure du possible, ne pas perturber les fonctionnalités développées précédemment.

## 6) Debug de Develop

Si lors d'un commit dans Develop une fonctionnalité est endommagée, la prochaine tâche à réaliser sera impérativement le debug de la fonctionnalité en question.

## 7) Commit de develop à master

Aucun membre ne pourra réaliser un commit directement dans la branche master à partir d'une branche personnelle. Les commits devront être réalisés depuis develop. Les commits dans master seront supposés complets, sans erreurs et constitueront la fin d'un sprint du projet.

## 8) Bug dans master

Si un bug est trouvé dans master, la réparation du bug sera prioritaire et impérative pour la totalité des membres de l'équipe.

## 9) Tests unitaire avant commit dans develop

Les membres devront réaliser tous les tests nécessaires pour assurer le bon fonctionnement des caractéristiques implémentées avant de réaliser un commit dans develop. Les commits dans la branche develop seront supposés au moins fonctionnels. Les membres pourront considérer que des erreurs sont encore présentes dans la branche develop mais devront chercher à les résoudre dans les plus courts délais.

## 10) Tests fonctionnels avant commit dans master

Avant de réaliser un commit dans master, la totalité des membres devront réaliser une banque de tests permettant de vérifier le fonctionnement parfait des fonctionnalités implémentées et devront tous valider le commit avant de le réaliser.

## 11) Commits explicites

Les noms des commits seront explicites et définiront les modifications réalisées. Si le nombre de modifications est grand ou le nom du commit est vague, le commit devra contenir une description avec une explication exhaustive des modifications réalisées.

## 12) Difficultés

Si un membre présente une difficulté, dysfonctionnement ou cassure du Git, le membre devra prévenir les autres membres et chercher à résoudre immédiatement l'erreur.

Les autres membres ne réaliseront aucun commit pendant la réparation du Git. Le membre présentant l'erreur devra, dans la mesure du possible, être aidé pour résoudre le problème dans les plus courts délais.

### 13) Sauvegardes

À la fin de chaque journée de travail, les membres réaliseront au moins une sauvegarde du Git sur un ordinateur séparé et indépendant des machines utilisées pour travailler, afin d'assurer une copie de sécurité des codes pour prévenir une perte partielle du projet.

### 14) Tags

Au moment de la fin d'un sprint, le créateur de la feature terminera la branche avec Git Flow et ajoutera un tag au commit avec la nomenclature suivante :

Etape <lettre etape> - <nom feature> - <Numéro version>

Le premier tag commencera avec un numéro de version 1.0.

Une modification mineure (une résolution de bug) augmente le 3º chiffre de la version (ex. 1.0 -> 1.0.1).

Un ajout de fonctionnalité ou de fonction augmente le 2º chiffre de la version et remet à 0 le troisième (ex. 1.0.1 -> 1.1).

Une modification majoritaire ou complète des fonctionnalités implémentées augmente le 1º chiffre et remet à 0 les autres chiffres (ex. 1.1 -> 2.0).

## II. Organisation de l'équipe

### 1) Lieu de travail

Le travail sera réalisé en présentiel pendant les journées, soit à l'Ensimag, soit à la Bibliothèque Universitaire, soit dans une coloc. Un membre pourra télétravailler ponctuellement, si les autres membres l'acceptent.

Par défaut, nous nous retrouverons en présentiel à l'ENSIMAG en E300.

### 2) Horaires

Les horaires de travail seront les suivants :

- Jours de travail : 9h - 12h30 + 14h - 17h30

Les horaires pourront être modifiés ponctuellement si les membres l'approuvent.

Pendant les week-ends et en dehors des horaires de travail, les membres ont une obligation de moyen. Les membres devront avancer dans les parties qui ont été acceptées au préalable.

### 3) Daily meeting

Tous les matins, les membres parleront pendant une quinzaine de minutes du travail réalisé, des problèmes rencontrés et de la répartition du travail de la journée.

### 4) Avancement organisé

Tous les avancements devront être validés au préalable. Aucun membre ne pourra avancer dans le projet sans avoir prévenu et avoir été validé par au moins le Chef de Projet.

### 5) Gestion des tensions et conflits

Il existe trois possibilités que l'on a établi pour gérer ce genre de situations:

- Les deux parties impliqués essaient de résoudre de manière pacifique et informelle le conflit.
- Si le conflit n'est pas résolu, une personne indépendante au conflit jouera le rôle de médiateur informel.
- Si le conflit n'est toujours pas résolu, l'équipe réalisera une réunion afin de laisser les deux parties se défendre et exposer leurs idées avant que l'équipe finisse par délibérer de manière formelle.
- Si le conflit n'est toujours pas résolu, l'équipe cherchera la médiation d'un professeur.

## III. Rôles

Les rôles suivants ont été attribués en considérant les envies de chacun, mais également leurs traits de personnalité, en particulier pour le Chef de Projet et le Git Master qui s'occupe aussi des relations humaines. Les rôles sont considérés comme définitifs.

### ● Chef de Projet – Nils Depuille

Le Chaff (Chef de Projet) est chargé de suivre l'avancement global du projet. Le Chaff devra suivre individuellement le travail des différents membres pour évaluer les problèmes relevés, les difficultés, les retards ainsi que le bon fonctionnement de l'implémentation. Le Chaff devra avoir aussi une connaissance globale de l'architecture du projet.

- ## DSI & Secrétaire Général – Vianney Vouters

Le SecGen (Secrétaire Général & DSI) devra gérer tous les outils hors code et Git (Trello, Planner, ...) afin qu'ils soient à jour avec l'avancement du projet. Le SecGen devra, entre autre, actualiser les diagrammes de Gantt, gérer le Google Drive de ressources, gérer les ordres du jour des réunions et noter ce qui a été dit pendant les réunions et Daily Meet. Il a ainsi pour rôle d'animer les réunions.

- ## Git Master & Human Ressources – Jorge Luri Vañó

Le Git Master est chargé de résoudre tous les éventuels problèmes de Git afin d'assurer le bon fonctionnement du projet. Il sera aussi chargé de sauvegarder les copies de sécurité du Git dans un espace sécurisé et indépendant. Le Git Master aura aussi un rôle de ressources humaines. Il devra réaliser des indicateurs pour vérifier que tout est en ordre, sortir des statistiques du projet et prévenir le Chaff d'éventuels problèmes. Il sera chargé aussi de la cohésion interne et d'épauler le Chef de Projet.

Le Git Master a le droit de faire des modifications directement dans develop et master pour modifier les paramétrages du Git.

- ## Test Master – Logan Willem

Le Test Master est chargé de la maintenance de la banque de tests du projet. Il sera chargé de vérifier que, pour chaque fonctionnalité, les tests présents sont suffisants. Il devra noter les tests qui ne passent pas pour suivre l'avancement du projet et maintenir le document de gestion des tests.

Le Test Master a le droit de modifier directement develop pour modifier la banque de tests du projet.

- ## Code Master – Virgile Henry

Le Code Master est chargé de vérifier que le code est bien formaté et correct. Il devra vérifier que la documentation est correcte et suffisante, vérifier que le code est cohérent avec l'architecture décidée en amont et sera responsable du debugage du projet.

Le Code Master a le droit de modifier directement develop pour formater les codes et ajouter la documentation.

# IV. Code

## 1) Langue de programmation

Seul l'anglais sera utilisé dans l'ensemble du code.

## 2) IDE

D'un commun accord, l'équipe n'utilisera que l'éditeur de code Visual Studio Code.

## 3) Formatage

Les normes élémentaires de formatage devront être respectées. En particulier, grâce au point IV.2., un simple Ctrl+Shift+I permettra ce formatage adéquat.

## 4) Javadoc

Chaque méthode se doit d'être documentée de sorte à permettre la génération automatique de la documentation avec Javadoc.

## 5) Documentation

Lors de l'écriture de code, les membres doivent considérer qu'il est important de commencer par rédiger une documentation de ce qu'ils comptent faire. Cela facilite une prise de recul sur ce qu'ils font, une meilleure hygiène de code et une facilité d'écriture. Cette ébauche de documentation doit ensuite être complétée pour être claire, concise et lisible.  De plus, les parties plus complexes du code se doivent d'être commentées.

# V. Outils utilisés

- ## Google Drive

  Cloud de stockage des documents relatifs au projet.

- ## Google Docs

  Bureautique et documentation.

- ## Trello

  Outil de gestion de projet.

- ## Discord

  Outil de télétravail et communication du projet.

- ## Messenger

  Moyen de communication informel.

- ## Fork

  Gestion de Git pour le Git Master.

- ## Planner

  Pour établir le diagramme de Gantt.

- ## Visual Studio Code

  Éditeur de code imposé par la partie IV.2.

## Signatures

Les cinq membres approuvent l'intégralité de la charte et s'engagent à respecter la totalité des articles et des pénalités.

Signé à Grenoble le 04/01/2023

Jorge Luri Vañó

Virgile Henry

Logan Willem

Nils Depuille

Vianney Vouters