Sept 2022

Gestion agile de Projet avec GitLab

IUT Lyon1, Département Informatique

Introduction

Ce court document explique comment gérer son projet en agile avec GitLab : découper la durée du projet en périodes (*sprints*) ; définir les tâches et les découper, leur assigner des responsables et des échéances ; commenter des tâches, les valider ; fusionner des tâches validées avec le cœur de la réalisation. Un tableau de bord (*taskboard* ou *board*) permet de voir et gérer les tâches à faire, en cours, terminées, ainsi que les bugs à corriger.

Nous avons créé un dépôt avec un modèle GitLab (*template*) que vous pourrez réutiliser pour gérer le projet simplement. C'est un projet à importer (lien ci-après) : l'un de vous en serez propriétaire et pourrez donc le modifier comme vous le souhaitez pour l'adapter à votre méthodologie.

Gestion de codes avec GitLab

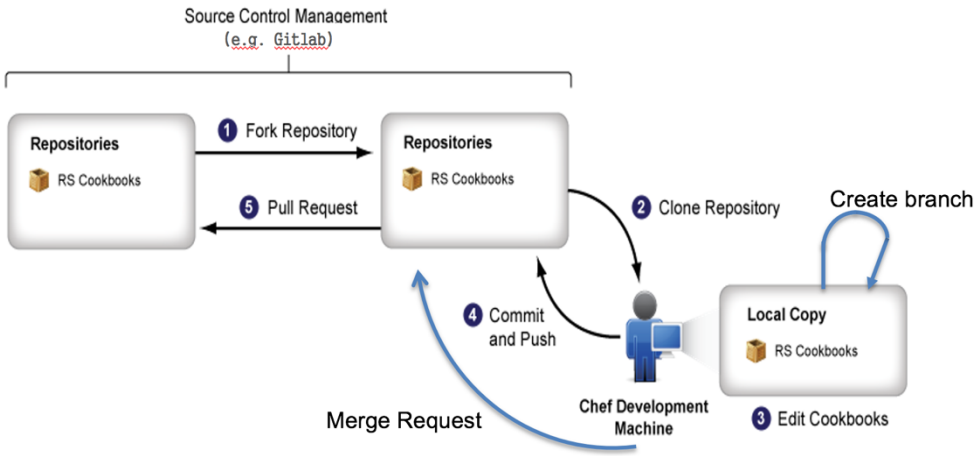
GitLab est un dépôt[1] qui exploite GIT pour gérer les différentes versions du code. D'autres dépôts existent tels que GitHub. Vous pouvez utiliser ces autres dépôts à votre façon mais impérativement en **projet privé** (Paramètres – Visibilité), en nous donnant les **accès** (Information du projet – Membre, mettre le tuteur-SAE en *Reporter*), et avec **les éléments demandés pour le suivi** (précisés ci-après).

Démarche générale : branches, fusion

Les concepteurs utilisent GIT pour gérer leur développement : *fork*[2] du projet dans votre dépôt[3], *clone* du projet en local, travail en local sur une branche spécifique (pas la *Master*), avec des *commits* réguliers pour pouvoir revenir en arrière en cas de problèmes, *push* d'une version stabilisée du code. Une fois éventuellement validée par ses pairs, le concepteur demande à fusionner son travail (sa branche) avec la branche *Master* à l'aide d'une *MR, merge request* (*pull request* dans la terminologie gitHub). La demande de fusion prend alors le statut de *Draft*. Des tests unitaires, fonctionnels ou d'intégration peuvent alors être effectués sur ce nouveau code, des échanges ont lieu sous forme de commentaires. Si tout va bien, le *merge* est validé : un concepteur approuve la proposition de fusion (*Approve Merge*). Sinon, les échanges sur la tâche se font par des commentaires jusqu'à obtenir une version valide.

Sur le dépôt aussi, on peut revenir en arrière dans l'historique des versions du projet, en cas de problème de code.

On recommence ensuite avec une nouvelle tâche du *Reste à Faire (RAF ou Backlog)* : nouvelle branche etc.



[1] Centralisé, ii hébergé par un serveur GitLab de l'Université Lyon 1 : c'est la *Forge Lyon 1*.
[2] Un *fork* est une copie d'un projet vers son propre dépôt, qui garde la trace de l'origine : ainsi si besoin, on pourra soumettre des modifications au propriétaire avec des Pull Request.
[3] Pour le *template* proposé, on fera un import

Dans le modèle on propose 3 branches : **Main** / **Pré-Démonstration** (logiciel en test par ex. avec un jeu de données restreint) / **Revue-finale** (version finale correspondant à l'environnement du serveur Client : accès à la vraie BD, etc.). Vous pourrez créer vos propres branches. Rappel : on ne travaille jamais directement sur la branche principale Master / Main. On développe ses propres branches et on demande une fusion sur la branche Main quand on estime l'avoir terminé.

Le Board

Le tableau de bord est un outil de management visuel comme Trello (basé sur les colonnes Kanban), qui montrent l'avancement des tâches. Par défaut, un *board* est affiché avec deux colonnes, "Open" et "Closed". *Open* correspond au *Backlog* (ou RAF, reste à faire), *Closed* aux tâches qui ont été validées et sont donc terminées.

Sous GitLab, on le gère par le menu **: Tickets / Tableaux (*Issues/Board*)**. Les autres colonnes prédéfinies du modèle proposé sont : *Sprint* (les tâches sélectionnées pour le sprint en cours), *En cours* (les tâches en cours de développement sur le sprint), *à valider, à corriger*. On définit les entêtes de colonnes à l'aide des *Other Labels* du modèle proposé. On ajoute alors des *issues* dans les colonnes, et on les fait glisser d'une colonne à l'autre en fonction de l'avancement.

On peut créer plusieurs *boards*, par ex. un plus fin pour le *sprint* en cours, mais aussi **filtrer les colonnes** avec différents critères, notamment les jalons (les tâches sans échéance, c'est un problème !), les labels, etc.

Les Issues/tickets

Les *Issues* sont des tickets, c'est-à-dire des tâches à faire. Ça peut être : organiser une réunion avec le Client (label Animation), définir le besoin (Conception), rédiger la documentation, reviewer la tâche X, etc.

Les *issues* sont très puissantes sous GitLab : on peut leur affecter des catégories (*labels*), assigner les tâches à des membres, estimer leur durée, ajouter des fichiers, commenter, recevoir des notifications, etc. C'est un espace idéal pour centraliser les discussions autour d'un projet et favoriser la collaboration.

Des modèles sont proposés dans le *template* pour renseigner les *issues* et les *merge requests*. On a aussi défini un modèle *d'issues* de suivi qui aide à préparer les séances avec le tuteur-SAE.

Les *issues* sont planifiées à une date donnée avec une durée. On peut mentionner la durée estimée *(/estimate* en commentaire) et réelle *(/spent*) et voir ainsi le retard généré sur le projet.

Les labels

Les *labels* sont des éléments qui sont associés à des *issues*, des *merge requests* et qui permettent de les classer, les organiser mais aussi les identifier simplement. On les utilise aussi pour nommer les colonnes du *Board*.

Dans *Project Information / Labels*, vous pouvez gérer vos labels. Certains sont prioritaires pour GitLab (*Prioritized*), comme Back, Front, Feature (fonctionnalité), mais aussi Animation (réunions), Conception, Infrastructure, Documentation, etc. et d'autres sont plus secondaires (*Other labels*), comme déjà évoqués : *sprint, en cours, à corriger, à valider*.

Les milestones (jalons)

Un jalon est une échéance particulière. Pour le suivi de la SAE de s3, on a proposé un certain nombre de *milestones* prédéfinis : un par *sprint*. Les *sprints* sont de 2 semaines sauf à la fin (une semaine). On peut mentionner les dates de début et de *fin au plus tard* du jalon.

On peut attacher des *Issues* au *milestone*. Une *issue* peut être rattachée à un et un seul *jalon*.

À la fin de chaque *sprint*, toutes les *issues* doivent être fermées dans l'objectif de clôturer le *milestone*. Les *issues* non réalisées peuvent être repositionnées sur un autre jalon.

Gestion Agile du projet

Ici on gère le projet en "SCRUM light[4]" : on définit un *backlog* et on procède par *sprint*. Un *sprint* est une durée de développement, la même durée définie pour toute la durée du projet, souvent 2 semaines à temps plein. On avance d'autre part de façon :

- **incrémentale** : à chaque fin de sprint, on livre au Client un prototype de logiciel (un incrément) de plus en plus évolués par ex. : les IHMs non fonctionnelles, puis une IHM de connexion opérationnelle et les IHM modifiées, etc.); avec les retours du Client, on est certain de la bonne compréhension des besoins, et on peut réajuster sans que cela ait trop d'impact ; ça signifie pour le projet qu'à chaque fin de sprint, on aura une démonstration d'une ou plusieurs fonctionnalités.
- **itérative** : à chaque sprint, les mêmes phases sont répétées (itérées) :
  - conception (découpage en sous-tâches, estimation des durées, affectation[5]),
  - implémentation,
  - tests en local,
  - push (envoyer le code sur GitLab), MR, fusion (cf démarche générale)
  - Revue de Projet avec le Client (identification des dysfonctionnements ou des erreurs de compréhension)
  - Rétrospective : l'équipe fait le point sur ce qui a marché ou pas et propose une action d'amélioration (ex. "Planifier une réunion de l'équipe avant la séance avec le tuteur")
  - Affinage du *backlog* si nécessaire (nouvelles tâches, nouvelle hiérarchisation du *backlog*).

Premier Sprint

Ce *sprint* de lancement du projet sera découpé en 4 étapes :

(1) **Capture des besoins**, définition des *features* (fonctionnalités) avec leur importance pour le Client (un nb de **points de valeur** affecté au projet global, que l'on ventile sur les *features* : les plus importantes ont plus de points), et premier découpage en tâches (*issues*). Voir en Annexe pour la différence entre *Feature* et *Issue*. Ces éléments constituent le *backlog* ou *Reste à Faire*. En cours d'avancement, de nouvelles *features / issues* peuvent apparaître : on les ajoute au *backlog* et on les retravaillera en phase d'affinage du *backlog*. Un exemple de *backlog* est donné en Annexe.

[4] SCRUM signifie mêlée; il n'y a pas toutes les cérémonies SCRUM dans notre modèle, notamment le Daily Meeting)

[5] Cette phase de Conception est appelée *Planning Game* en SCRUM

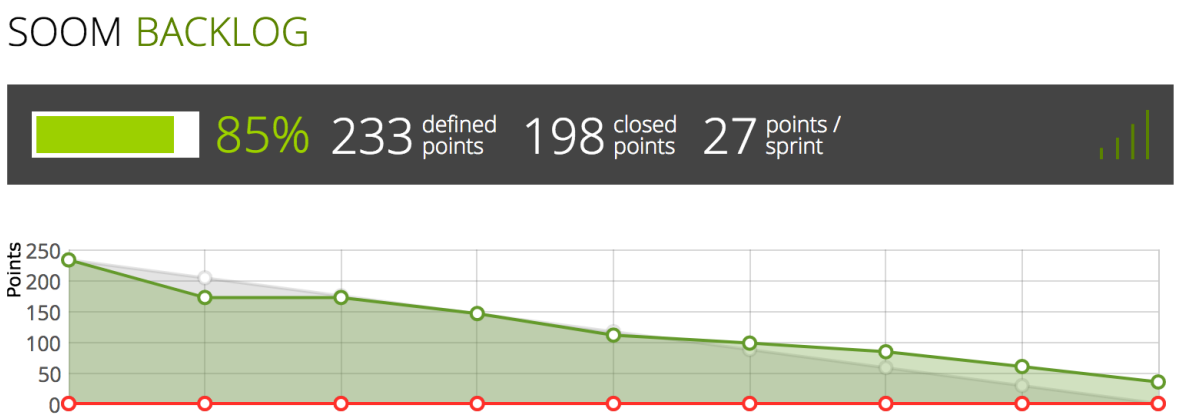
(2) **Estimation et Priorisation du backlog** : l'équipe estime la difficulté de réalisation des *features* à l'aide de **points d'effort** définis d'après la suite de Fibonnacci[6]. On hiérarchise ensuite le *backlog* en fonction du **ROI[7] : importance x difficulté de réalisation**. Les tâches les plus importantes pour le Client et les plus simples à faire seront développées en premier. L'objectif est de MONTRER au Client ce qui l'intéresse le plus rapidement, et d'avoir son feedback : on évite ainsi des erreurs de cadrage.

(3) **Planning des Sprints** : on ventile les *features* sur les *sprints* en respectant l'ordre précédent.

(4) **Définition des rôles** : développeurs, Dev Designer, Dev Ops, Scrum Master (SM, celui qui anime les réunions Scrum : Revue, Rétrospective), le représentant du Client (MOA Maître d'Ouvrage, ou PO *Product Owner*)[8]. Ces rôles sont mentionnés dans le profil des membres de l'équipe sur GitLab.

Pour ceux qui veulent : le Burndown Chart de SCRUM

C'est un outil SCRUM visualisant l'avancement du projet : on voit si on est en retard ou pas par rapport à la fin prévue du projet. En abscisse : les sprints. En ordonnée : le nombre de **points d'effort**[9] du projet. A chaque fin de sprint on mentionne combien de points d'effort ont été réalisés d'après les issues terminées et validées par le Client lors de la Revue de Sprint.



La droite en gris est une droite théorique montrant une progression linéaire avec une fin à la date prévue. Ici on voit qu'au sprint 1, l'équipe a pris de l'avance (elle a réalisé plus de tâches que l'avancement linéaire le préconisait) mais que sur les sprints 6,7 et 8, elle a accumulé du retard.

Le "27 points / sprint" s'appelle la **vélocité de l'équipe** en SCRUM. Dans une équipe stable, quelque soit le projet, une même équipe a généralement la même vélocité (à durée de sprint identique et avec des technologies connues), ce qui permet de correctement planifier l'avancement.

On peut construire ce *Burndown chart* sous Excel.

---

[6] Par exemple, si c'est immédiat : effort de 1 ou 2 ; facile = 3 ; assez facile = 5; moyen = 8; difficile = effort supérieur à 13

[7] ROI : *Return On Investment*

[8] Pour les sujets proposés pour les SAÉ, le représentant du Client/PO sera l'enseignant tuteur

[9] Ne pas confondre Points d'effort et Points de Valeur !

Installer son projet GitLab avec le *template*

Il faut télécharger l'export du dépôt pour l'importer dans votre projet SAE avec tous les éléments proposés (labels, issues, merge requests, board, branches).

1. Télécharger l'export du dépôt : https://forge.univ-lyon1.fr/V.Deslandres/templategp-agile-sae-s3/-/blob/main/.ressources/export.tar.gz
2. Importer l'export dans GitLab : https://docs.gitlab.com/ee/user/project/settings/import_export.html#import-a-project-and-its-data

Les rôles dans GitLab

Il en existe 5 *: Guest, Reporter, Developer, Maintener, Owner.*

Tous les membres sauf les *Guest* peuvent créer modifier des *issues*, des *listes*. Mais seul *l'Owner* peut supprimer une issue.

Le processus de *merge request* dans GitLab

https://www.youtube.com/watch?v=InKNIvky2KE de 1:10 à 2:35

Illustration des fusions *fast-forward* de branches avec GIT:

https://www.youtube.com/watch?v=ZJuUz5jWb44 de 7:10 à 10:57

Annexe : exemple de backlog

Pour une application "Panier de courses en ligne", voici les **User Stories** telles que définies par le PO avec leurs **points de valeur**. Ces stories sont ensuite déclinées en tâches par l'équipe de DEV, qui estime la difficulté de réalisation de chacune (**effort**). Par exemple, l'US 1 comprend les tâches :

- Réaliser l'IHM de création de compte – Effort : 3
- Traiter une adresse email avec un envoi de code – Effort : 5
- Modifier le statut du compte ('EnAttenteCreation', 'Actif') – Effort : 1

Feature : Gestion de Compte – Valeur : 85 – Effort : 69

Issues (US=*User Stories* en SCRUM)

US 1 : en tant que client je peux m'inscrire – Valeur : 20 ; Effort : 9 (3+5+1)
US 2 : en tant que client je peux me connecter – Valeur : 20 ; Effort : 12 (3+3+3+3)
US 3 : en tant que client je peux modifier mes informations – Valeur : 10 ; Effort : 13
US 4 : en tant qu'administrateur, je peux supprimer un compte Client et toutes ses informations – Valeur : 05 ; Effort : 20
US 5 : en tant qu'administrateur, je peux *black-lister* un compte Client – Valeur : 30 ; Effort : 15

Feature : Gestion de la liste de courses – Valeur 120

Issues

US 6 : en tant que client je peux visualiser ma liste de produit
US 7 : en tant que client je peux ajouter un produit à la liste
US 8 : en tant que client je peux supprimer un produit de la liste
US 9 : en tant que client je peux enregistrer une liste de course
US 10 : en tant que client je peux supprimer une liste de course
US 11 : en tant que client je peux accéder aux menus de saison
US 12 : en tant que client je peux ajouter les produits d'un menu à une liste de course

Le backlog comprend aussi des **tâches techniques.**

Exemple de *Stories* techniques :

TS 1 : configurer GitLab du projet
TS 2 : installer l'IDE X

Le backlog comprend enfin des **tâches de debug** :

DS 1 : corriger pb d'alignement colonnes page Machin

DS 2 : faire un test unitaire pour comprendre le mauvais calcul de la fonction foo()