

# Bilan sur la gestion d'équipe et de projet

Antoine Turkie, Augustin Chenevois, Aurélien Delory, Mihaja  
Razafimahefa, Sarah Ramaharobandro

# Table des matières

<b>Introduction.....</b>	<b>2</b>
<b>Organisation générale du projet.....</b>	<b>2</b>
<b>Réalisation des étapes A, B et C.....</b>	<b>2</b>
<b>Organisation d'un sprint.....</b>	<b>2</b>
<b>Répartition du temps de travail.....</b>	<b>3</b>
<b>Gestion des labels.....</b>	<b>4</b>
<b>Répartition des tâches.....</b>	<b>4</b>
<b>Evolution de la charge de travail.....</b>	<b>5</b>
Sprint 1 (18 au 22 décembre).....	5
Sprint 2 (08 au 15 janvier).....	6
Sprint 3 (15 au 22 janvier).....	8
Sprint 4 (22 au 25 janvier).....	9
<b>Rétrospective.....</b>	<b>9</b>

# Introduction

Pour réaliser notre Projet GL, nous avons décidé d'appliquer la méthode Agile/Scrum et pour cela nous avons fait appel à GitLab. Cela nous permettait par l'intermédiaire d'un tableau de bord intégré au répertoire du projet d'avoir accès à toutes les tâches à faire, en cours et terminées. Ces tâches sont balisées avec différents labels, notamment pour marquer les User Stories.

# Organisation générale du projet

Nous avons réparti le travail en différents sprints correspondant à un ensemble de User Stories, Technical Stories et Fix remontés pour atteindre un objectif :

- Sprint 1 : Afficher “Hello World”.
- Sprint 2 : Réaliser le langage sans-objet et la documentation de l’extension.
- Sprint 3 : Réaliser le langage avec objet et la conception de l’extension.
- Sprint 4 : Réaliser l’extension, les rendus finaux et la soutenance.

A chaque Sprint `x` correspond une branche labellisée `sprintx` sur laquelle nous travaillons afin de toujours avoir la branche `master` fonctionnelle. Une fois toutes les tâches du sprint courant effectué et l’objectif atteint, vérifié à l’aide de nos scripts de tests, on décidait par approbation de tous les membres du groupe de merge la branche `sprintx` avec `master` pour valider nos changements et être prêt pour le rendu.

# Réalisation des étapes A, B et C

Nous avons choisi de réaliser le sprint 1 ensemble afin de se familiariser sur les différentes parties du projet et les 3 grandes étapes. Ensuite, lorsqu'on a commencé le langage sans-objet, nous avons opté pour se séparer en User Story et adopté une approche par incrémentation des fonctionnalités. Pour une tâche User Story assigné, on réalisait les étapes A, B et C de la fonctionnalité afin de pouvoir si le temps venait à nous manquer rendre un compilateur avec ses fonctionnalités développées, testées et opérationnelles.

Pour le langage avec objet nous étions contraints d'adopter la démarche Waterfall : réaliser d'abord les étapes A et B avant l'étape C puisque cette dernière dépendait des résultats obtenus lors des différentes passes de l'étape B. Nous avons néanmoins pu réaliser la totalité des fonctionnalités attendues.

# Organisation d'un sprint

L'organisation d'un sprint était extrêmement minutieuse dans notre groupe afin d'éviter tous les soucis classiques qu'un groupe lambda peut avoir (problèmes de merge, dédoublement de tâches, code faux ajouté, etc....). Voici donc comment notre groupe opérait :

- Création d'une nouvelle issue indiquant le travail que l'on doit effectuer (ex : Réalisation du gencode pour instanceof) et qui s'occupera de cette tâche (approximatif, l'auteur du merge request fera aussi bon foi du réalisateur effectif de la tâche).
- Création de la branche correspondante sur Git partant de la branche du sprint actuel et création de la merge request correspondante.
- Réalisation de la tâche sur cette branche + une série de tests prouvant son bon fonctionnement.
- Lancement du script de tests afin de vérifier que rien n'a été cassé par l'ajout du nouveau code.
- Demande de review de la merge request par une personne qui n'a pas effectué la tâche.
- Si le reviewer valide, merge sur le sprint actuel. Sinon, il remonte des points à fix.
- Lancement du script de tests sur la branche **sprintx** pour valider la non-régression.
- On close l'issue.

En réalisant de manière rigoureuse ce déroulement, il est simple d'éviter les problèmes courants et de toujours garder un œil sur chaque partie du projet même si on ne travaille pas forcément dessus par une review.

## Répartition du temps de travail

Nous n'avons pas réalisé de planning poker pour attribuer un poids à nos User Story. En effet, nos tâches étaient assez granuleux pour se suffir à elle-même et être développé sur le tas. Néanmoins, grâce à notre procédure d'association User Story/Tâches et Merge Request, GitLab nous a produit des statistiques de notre temps de travail entre la création du Merge Request et sa finition en faisant le merge dans le sprint courant. Une fonctionnalité donnée prend en moyenne une journée de travail pour être implémentée complètement (étape A, B et C) mais cela peut varier en fonction de la difficulté de la tâche. Très souvent, on appliquait la programmation par pairs pour la conception et l'implémentation. Pour les étapes A et B, la phase d'analyse et de conception consiste à se documenter sur le polycopié fourni pour repérer les règles de grammaires nécessaires ou les conditions contextuelles à vérifier. Cette partie ne prend pas beaucoup de temps. Par contre, l'étape C nous a posé un peu plus de difficulté puisque c'est la partie la moins guidée, on y a donc consacré plus de temps pour la conception. Ensuite viennent le codage et la vérification avec des tests qui prennent la plus grande partie de notre temps. La programmation par paire est particulièrement efficace pour éviter facilement certaines erreurs ou oubli. Enfin, une phase de validation vient compléter le travail fourni, cette phase peut également être laborieuse et prendre aussi du temps puisqu'il faut se plonger dans le code de quelqu'un d'autre mais elle a été vitale pour nous apercevoir de certains bugs.

En ce qui concerne la documentation, nous avons répertorié tous les documents nécessaires

au projet et tous les documents de recherche en lien avec l'extension sur des posts Discord afin d'avoir une base de données et de connaissances accessibles et contribuables par tous.

## Gestion des labels

### Créer une classe vide

#51 · created 1 week ago by Sarah Ramaharobandro · Sprint3 · Doing · Priorité 1 · User story

Pour bien comprendre l'importance et le traitement de chaque tâche de notre tableau de bord courant, nous avons opté pour différents labels :

- Un label **User Story** permettant d'indiquer que la tâche est une fonctionnalité visible par l'utilisateur, à implémenter pour le sprint courant.
- Un label **Technical Story** permettant d'indiquer que la tâche est une fonctionnalité technique visible par nous développeur, à implémenter pour le sprint courant.
- Un label de **Priorité** afin de savoir l'importance de la tâche à effectuer et son effet sur la qualité de notre compilateur. Cela nous permettait de classifier les en gérant les fonctionnalités indispensables (opérations arithmétiques, création de classe, etc...) des fonctionnalités qui avaient une moins grande priorité.
- Un label **Doing** qui permettait de savoir si la tâche était en cours de réalisation par un membre du groupe.
- Un label **Fix** qui indique qu'une fonctionnalité était incomplète ou présentait un bug à corriger.
- Un label **Optim** en lien avec l'extension.

## Répartition des tâches

Open	Doing	Closed
Gestion de READINT et READFLOAT #33	Déclarer et instancier les types int, float, boolean User story #14	Vérifier l'affectation avec la nouvelle algo des opérations arithmétiques Fix #42
Utiliser les structures de contrôle if, else User story #19	Imprimer les variables de type int, float User story #15	Deleguer toutes les instructions d'impression a AbstractPrint Fix #39
	Utiliser la structure de contrôle while User story #26	Add tests for arithmetic operations and Etape A Technical Story #40
		Ajouter des tests #29
		Comparer des entiers et flottants User story #18

### Vue d'ensemble du tableau de bord des issues

Le scrum master ouvre les issues relatives à un nouveau sprint. Après un débrief, les tâches à accomplir sont réparties et au cours du sprint les tâches qui restent à faire sont choisies à partir de ce board.

## Evolution de la charge de travail

### Sprint 1 (18 au 22 décembre)

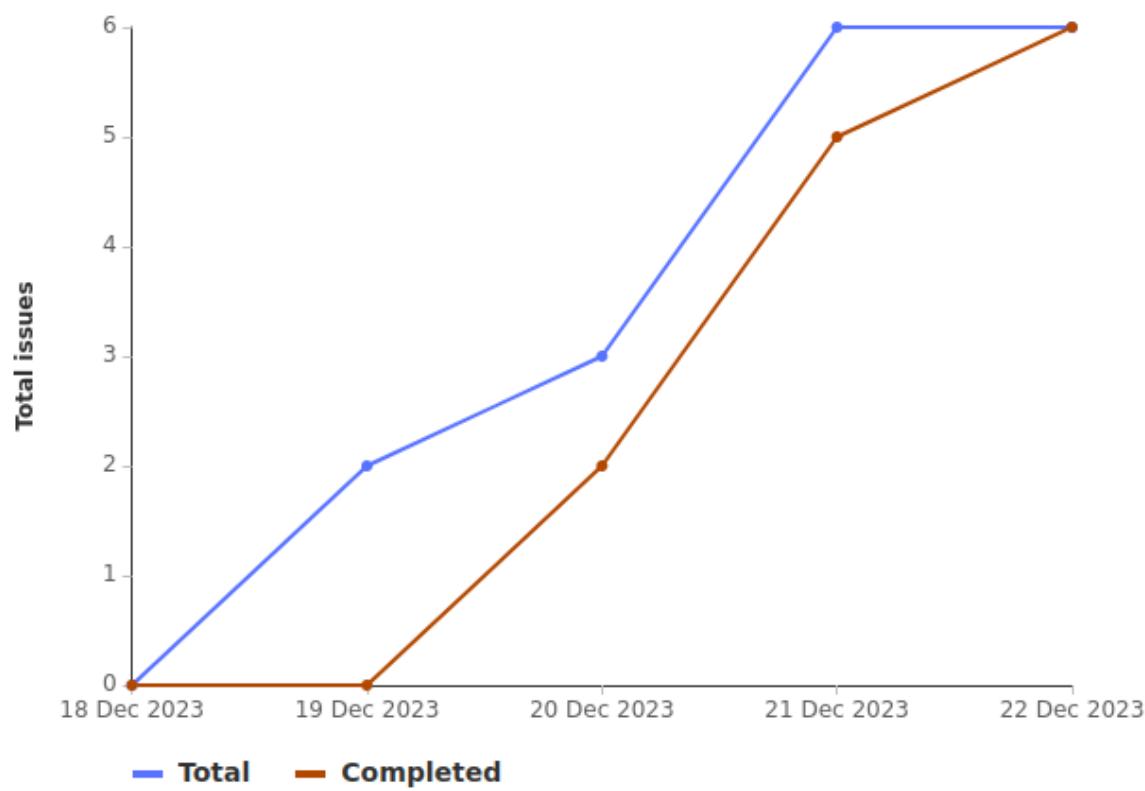
Il s'agit surtout d'un sprint où l'objectif principal était de comprendre le sujet en globalité et de se documenter. A l'issue de cette première semaine, la fonctionnalité `print("Hello world")` a été divisée en plusieurs issues dont l'analyse lexicale, syntaxique, contextuelle et la génération de code. La réflexion s'est surtout faite ensemble durant ce sprint.

Voici comment la charge de travail a évolué durant ce sprint :

### Burndown chart



### Burnup chart

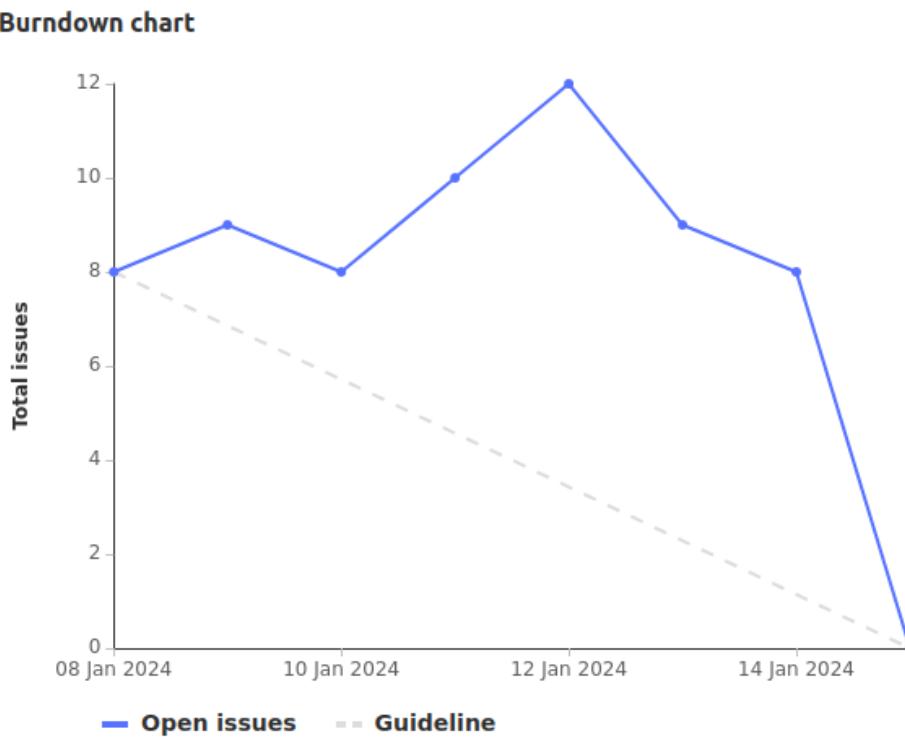


## **Sprint 1 : Implémentation de “Hello world”**

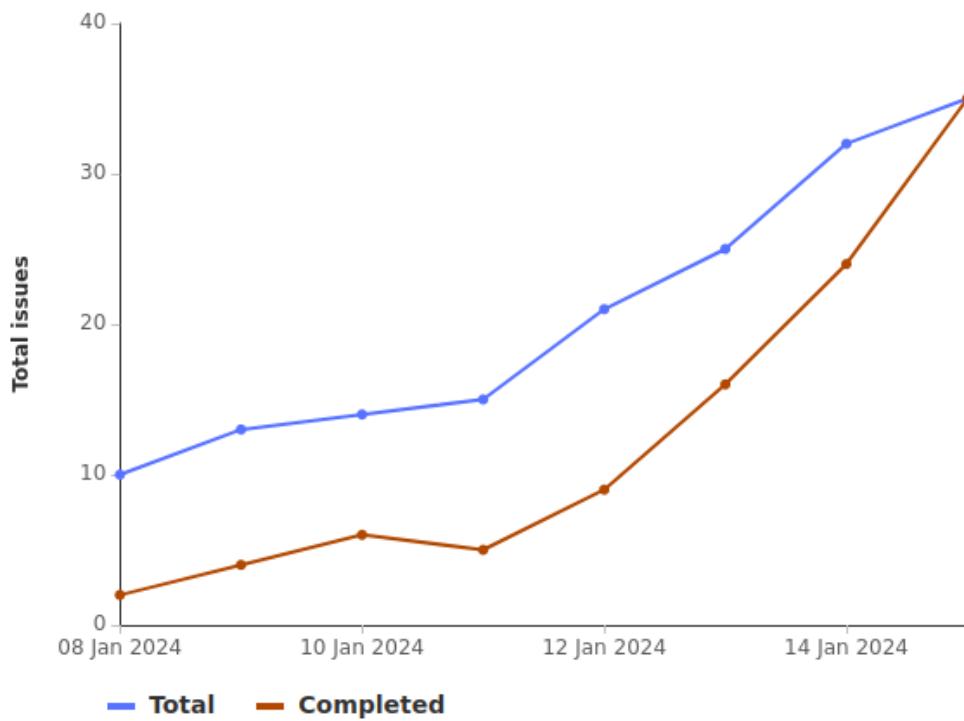
### **Sprint 2 (08 au 15 janvier)**

À partir de ce sprint, notre organisation en User Story et la procédure de validation de nos merge request nous a été d'une grande aide pour aller au bout de toutes les fonctionnalités à implémenter et de toutes les notions importantes à assimiler (étape B et C). Nous avons également développé durant ce sprint nos scripts d'automatisation des tests, intégrés au cycle de développement de Maven. Ainsi, il nous suffisait désormais de lancer `mvn test` pour vérifier la non-régression des fonctionnalités que l'on venait de développer.

Voici comment la charge de travail a évolué durant ce sprint :



## Burnup chart



### Sprint 2 : Implémentation du langage sans objet

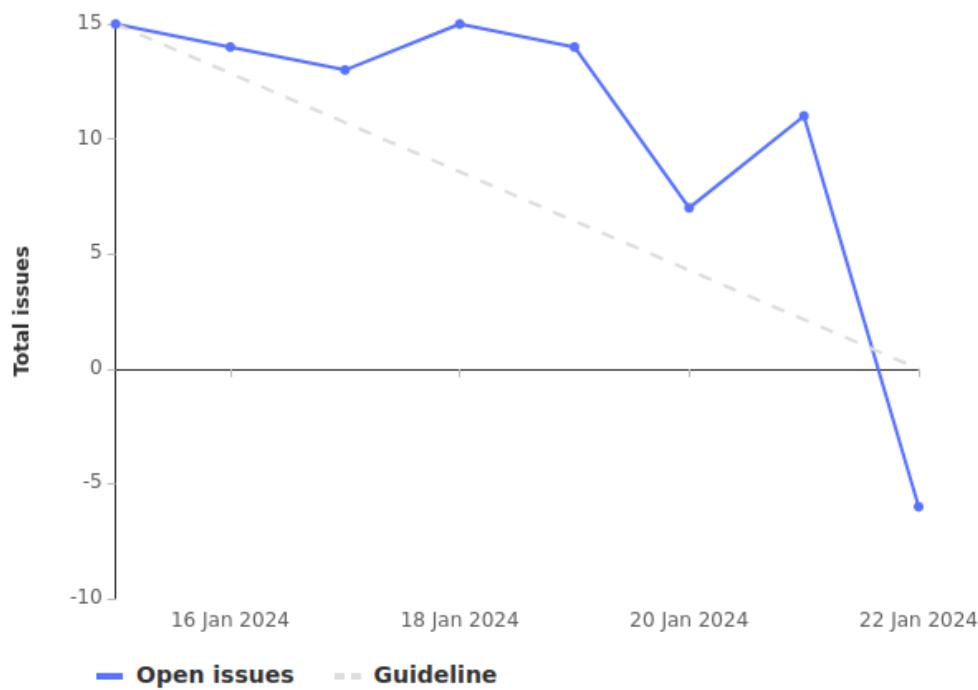
On peut constater qu'on a eu un pic de retard vers la fin de la semaine. En effet, il y a eu une assimilation plus longue de l'étape B/C pour certaines fonctionnalités (décoration de l'arbre pour les déclarations de variable, génération de code des opérations arithmétiques,...). Cependant, en prenant avantage du pair programming et en faisant une réévaluation de la répartition du travail en fonction de nos compétences, on a pu combler ce retard pour la fin du sprint.

### Sprint 3 (15 au 22 janvier)

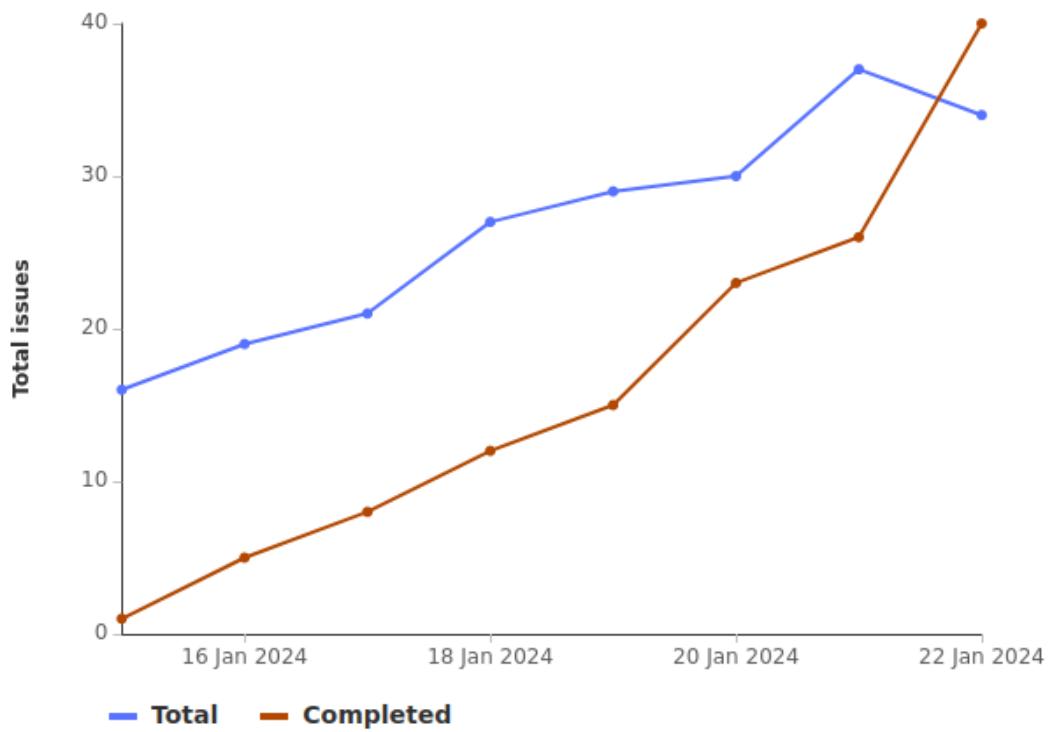
La dernière étape du projet consiste à implémenter le langage avec objet. Étant maintenant habitué à manipuler les outils de git (user story, merge request,...), on a pu travailler plus rapidement pour ce sprint. Cependant l'interdépendance des fonctionnalités à implémenter nous a contraint à avoir une approche moins incrémentale sur le travail à fournir. En effet l'intégralité des étapes A et B a été implémentée avant qu'on puisse se pencher sur l'étape C.

Voici comment la charge de travail a évolué durant ce sprint :

### Burndown chart



### Burnup chart



### Sprint 3 : Implémentation du langage avec objet

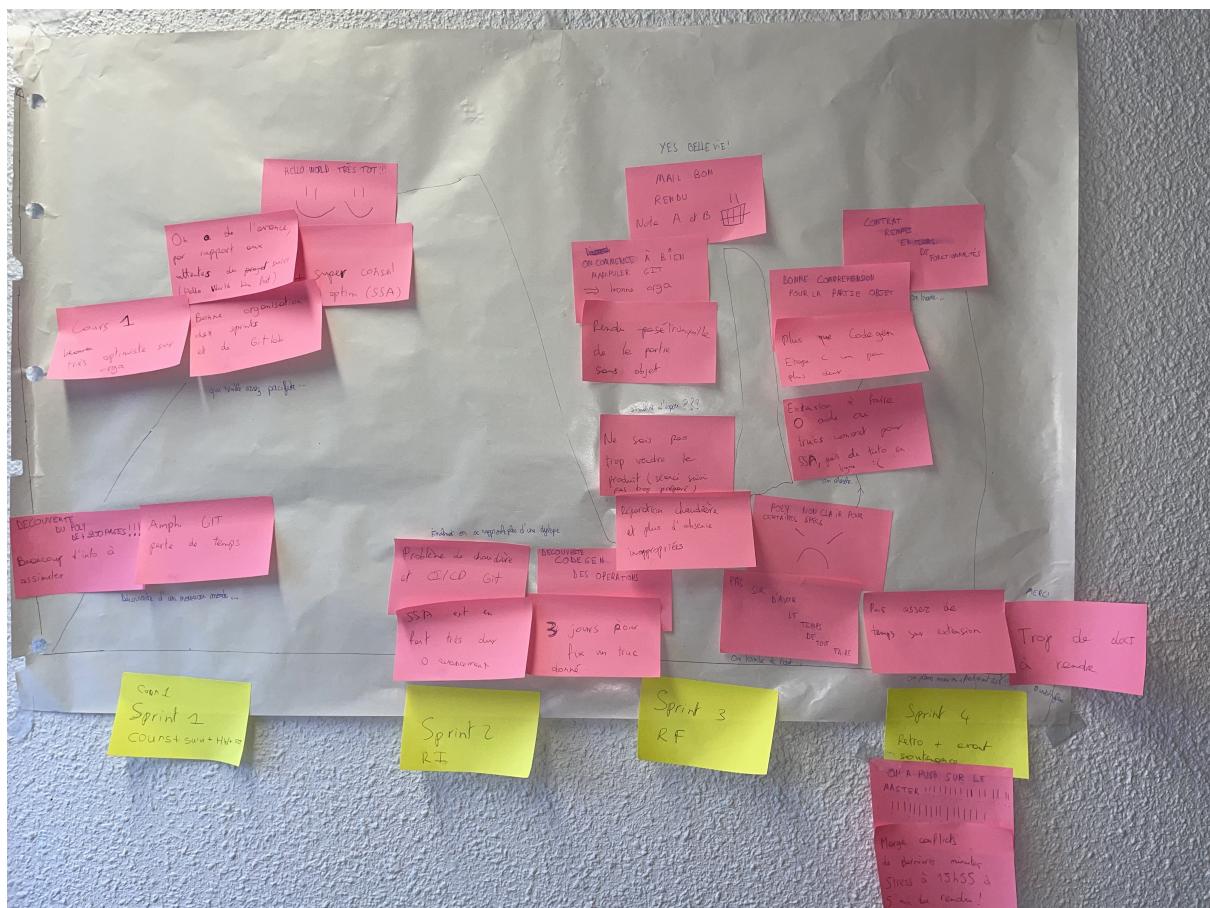
Certaines issues ont été rassemblées à la fin, ce qui a produit cette légère incohérence vers la fin du sprint.

## Sprint 4 (22 au 25 janvier)

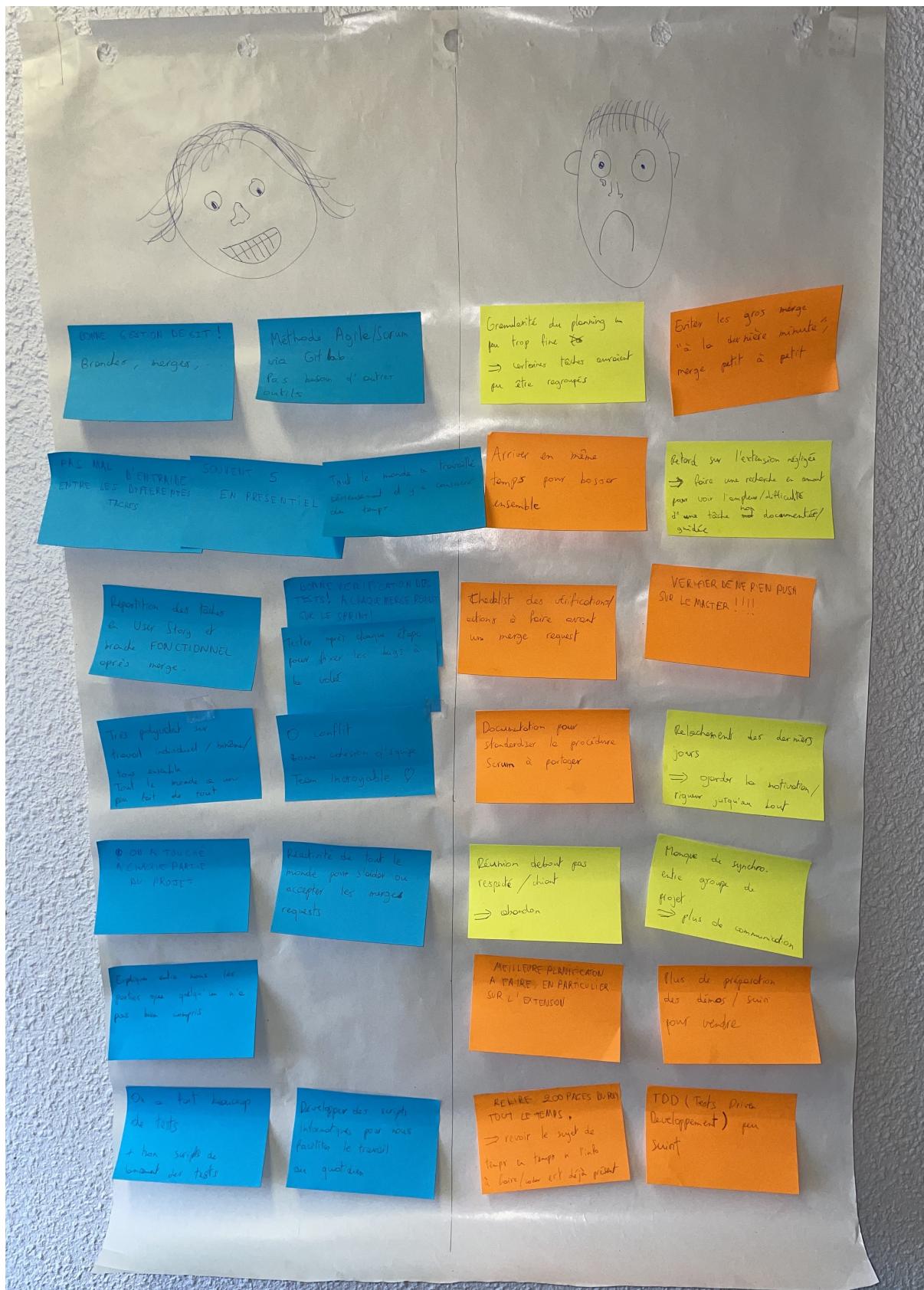
Le sprint 4 correspond à la fin de l'extension, à l'écriture des documents à rendre et à la préparation de la soutenance. Comme les deux dernières parties sont des tâches bureautiques, nous n'avons pas utilisé le burndown ou le burnup chart de GitLab pour surveiller notre progression.

## Rétrospective

Malgré la bonne réalisation de nos sprints, nous sommes conscients des dérives et erreurs passées qui ont pénalisé le bon déroulement et le respect de nos procédures. Dans une processus d'amélioration continue dans le cadre d'une rétrospective Agile/Scrum, nous avons relevé notre ressenti sur ce projet et les actions que l'on juge importantes afin que nos futurs projets se déroule encore mieux :



**Rétrospective collective (en abscisse les moments clés du projet, en ordonnées l'axe (content/pas content))**



**Rétrospective de l'équipe (en bleu les bons points à garder, en rouge les mauvais points à améliorer, en jaune les réactions émotionnelles)**