PROJET DE DEVELOPPEMENT EN INGÉNIERIE LOGICIELLE

– Portfolio –

Authors :

DACONDA Olivier
LOGAN Victoria
WICHOUD Nicolas

Destined to :

Professor CHAPPUIS Bertil
Assistant SANTAMARIA Miguel

Thursday, the 5th of May 2022

Table des matières

INTRODUCTION

Nous allons développer un projet de constructeur de site en utilisant les méthodes de développement AGILE. C'est un processus semi-piloté avec une planification incrémentale adapté aux besoins et contraintes du client. Les activités sont entrelacées plutôt que séparées et les échanges entre activités sont permis. L'automatisation joue un rôle clé dans le processus. Par exemple, les tests unitaires, l'intégration et livraison continue permettent d'avoir plusieurs versions exécutables à tout moment. Le but de ce projet est de se familiariser avec ces méthodes et de développer une méthodologie de travail dans le cadre d'un développement de projet.

1 METHODOLOGIE

1.1 Description de l’équipe

WICHOUD Nicolas :

Nicolas est à l'aise avec les maths et la programmation. En revanche il a parfois de la peine à s'organiser à l'avance et à prendre des initiatives, mais se donne cependant corps et âme pour arriver à ses fins lorsqu’on lui attribue une tâche bien définie.
Il a également de bonnes capacités sociales et n'a pas spécialement de difficultés pour s'intégrer dans un groupe de travail.

LOGAN Victoria :

Victoria est à l'aise avec la rédaction et la programmation. Elle a de très bonnes capacités sociales, comprend facilement les autres dans leurs besoins et a une aisance dans la gestion de conflits ou tout autre situation sujette à la communication.
Aussi, il est facile pour elle de mettre en place des plans d'organisation clairs et pertinents, bien qu'elle peine parfois à les suivre par la suite.

DACONDA Olivier :

Olivier est à l'aise avec l'organisation et la programmation. Souvent appliqué et consciencieux, Olivier saura motiver les troupes afin de se surpasser.
En revanche, il a du mal à gérer la pression face aux échéances et au retard.

1.2 Description du mode de collaboration

Pour chaque tâche, nous allons les distribuer entre nous trois et chacun.e créer des issues et les lier à une branche. Lorsqu’une de ces tâches est terminée, l’auteur.e effectuera une pull-request qui devra être examinée et validée par un pair, ceci afin de la joindre au main – fil directeur de notre projet.

Concernant notre organisation d’équipe, nous allons nous réunir hebdomadairement pour une séance qui consistera à être tenu.e.s au courant de l’avancée des différent.e.s membres de cette équipe et ainsi prévoir la suite du projet.

En ce qui concerne la mise à jour du Portfolio ci-joint, elle ne sera pas liée à une pull-request et la personne en charge de la push sur le main sera Victoria. Ce portfolio sera mis à jour lors de chaque séance, pour y exposer nos réflexions, changements, mises à jour de façons de procéder, ou toute autre information pertinente.

Nous allons utiliser un Kanban basique pour chaque projet (chaque sprint). Pour chaque issue correspondant à une tâche, nous allons créer une branche y étant associée et il revient donc à la personne qui s'occupe de cette issue de déplacer cette tâche de la colonne « To Do » (c.f. section suivante) à la colonne « In progress » puis finalement à la colonne « Done ». Les branches associées aux issues seront nommées selon la convention « fb-nomDeTacheCorrespondant ».

Lorsque le.la membre de l'équipe a terminé une tâche, iel fait un push sur le repo et il revient à un.e autre membre de revoir, d'accepter le cas échéant la Pull Request et de la fusionner avec le main. Ainsi l'auteur.e de cette tâche pourra la mettre dans la colonne « Done ».

1.2.1 Séance hebdomadaire

Chaque semaine, nous nous retrouverons pour une séance qui consistera en la revue de ce qui a été fait, ce qui reste à faire pour le sprint en cours et la redistribution des tâches – courantes et/ou futures.

Les points à aborder sont :

- Examiner les progrès réalisés et comparer au progrès attendus
- Sélectionner les stories à mettre en oeuvre ensemble
- Diviser les stories en tâches
- Répartitir le travail
- [Réflechir sur des problèmes rencontrés par d'autres membres (optionnel)]

Ces séances auront une durée de 10 à 40 minutes maximum : 5 à 10 minutes pour Examiner et Selectionner, 5 à 10 minutes pour Diviser et Repartir et 15 minutes maximum sur la partie Réfléchir, si besoin.

1.2.2 Organisation

Pour chaque sprint, nous créront un projet et y associeront un kanban afin d'étiqueter les issues. Ainsi, une fois les issues créée, elle sera automatiquement mise dans la colonne « To Do ». Lorsqu'un membre commence à travailler dessus, nous les déplaçons manuellement dans la colonne « In progress », pour finalement la déplacer dans la colonne « Done » lorsque la tâche est terminée et que la pull-request associée a été acceptée et intégrée au main.

1.2.3 Convention de nommages

- **Constante** en majuscules et séparation de mot par un _
  → VITESSE_LUMIERE
- **Nom de variable** en camelcase
  → plusPetitMultipleCommun
- **Nom de méthode** en snake
  → afficher_personne()
- **Nom des paramètres** en camelcase
  → superficie(nbCote,tailleCote)
- **Release** : notation semver lors de la fin de chaque sprint
  → 1.0.0 < 2.0.0 < 2.1.0 < 2.1.1

1.2.4 Autres conventions

- Langue : anglais
- Autoformatage : formatage par défaut d'intelliJ (indentation, etc.)
- Documentation du code en Javadoc

1.2.5 Principes

Voici les principes clés à adopter pour notre travail :

- Stories
- Code reuse
- Test-first programming
- Conception incrémentale
- Décomposition
- Refactoring
- Test d'intégration et tests systèmes
- Automatisation
- Commit early commit often

1.2.6 Création de fonctionnalité

Pour chaque fonctionnalité nous allons suivre cette procédure.

1. Rédaction d'une histoire pour des utilisateurs spécifiques ;
2. Définition des spécifications ;
3. Séparation du problème en tâches solvables.

Puis pour chaque tâche :

1. Création d'une issue et d'une branche associée ;
2. Création de tests ;
4. Développement ;
5. Pull request ;
6. Intégration par un.e autre membre.

1.2.7 Release

A la fin de chaque sprint, nous avons décidé de créer une release avec du code fonctionnel. La checklist associée sera

1. Terminer tous les issues possibles du Kanban
2. Merge toutes les pull request
3. Créer un tag
4. Ajouter l'exécutable sur Github

Ceci marquera ainsi la fin du sprint en cours.

2 SPRINT I – DEVELOPPEMENT AGILE

2.1 Collaboration

Durant ce sprint, nous avons eu beaucoup de changements quant à la collaboration pour ce projet. Nous avons eu en effet beaucoup de peine premièrement à tenir nos délais, mais également à avoir une stratégie structurée. Nous avons ainsi découvert que nous étions très efficaces en matière de discuter des problèmes, s'entre-aider lorsque l'un.e d'entre nous était bloqué.e et se fixer des objectifs.

Premièrement, le chapitre méthodologie nous a posé problème ; n'ayant pas une grande maîtrise de l'outil GitHub, nous avons eu plusieurs soucis quant à son utilisation – notamment sur la gestion des pull request, Kanban, un problème sur GitHub qui rendait l'accès à la classroom impossible pour Victoria, l'organisation en branches, etc. Il nous a donc fallu du temps et de la réfelxion pour trouver une manière optimale de l'utiliser afin de travailler correctement et efficacement en groupe.

Nous avons aussi constaté avoir pris trop de temps en ce qui concerne les séances hebdomadaires, que nous avons donc décidé durant ce Sprint I de mieux structurer, en établissant un nouveau plan de déroulement :

- Ecoute de chacun, de là où il.elle en est (~5-10mn) ;
- Etablissement des objectifs pour la semaine suivante (~5-10mn) ;
- Si besoin, discussion d'un problème rencontré par l'un.e de l'équipe, pour le résoudre ensemble (~15mn max.).

Ce nouveau schéma de séance nous permet ainsi de prendre en moyenne 15 minutes pour cette séance hebdomadaire, et au maximum 35 minutes ; ce qui nous fait gagner en efficacité sur le projet durant les périodes à disposition.

Ce premier sprint a donc été grandement utile en manière d'organisation de groupe, de structuration et de méthodologie de travail.

2.2 Stories

Nous avons choisi de diviser ce sprint en deux stories distinctes. La première consistait à créer un parseur pour les fichiers contenant les informations du site statique et la seconde consistait à gérer la command line. Le temps prévu pour la première story nous semblait plus conséquent et nous avons donc décidé de la diviser en deux tâches, respectivement le parseur markdown géré par Nicolas et le parseur yaml géré par Victoria, tandis qu'Olivier s'occupait de la ligne de commande. Il s'est avéré que la ligne de commande demandait un plus gros investissement que prévu et nous n'avons donc pas eu le temps d'aller au bout de cette story.

En ce qui concerne la story concernant le parseur, nous avions d'abord comme expliqué pensé séparer son implémentation en deux phases. Après recherches et réflexions, nous avons pensé à chercher du code déjà écrit (code reuse, c.f point suivant) ; nous avons finalement trouvé du code faisant ce qui était souhaité et l'avons ainsi implémenté par la suite sans difficulté.

2.3 Code reuse

Pour le parseur, nous avons trouvé - comme mentionné ci-dessus - sur internet une librairie qui effectue automatiquement le parsing d'un fichier composé de yaml et de markdown. Alors que cette tâche nous semblait plutôt ardue, la découverte de cette librairie nous a largement simplifié la vie puisqu'il nous suffisait de rajouter des tests pour vérifier que tout fonctionne correctement. Nous sommes donc plutôt satisfait de notre utilisation de cette pratique agile.

2.4 Test-first programming

Nous avons essayé d'utiliser un maximum de tests unitaires afin de tester notre parseur. En effet, celui-ci devait réussir à lire des titres, des listes à puces ou encore des images en markdown ainsi que des méta-données en yaml. Nous avons testé chacun de ces aspects individuellement et nous sommes très contents du résultat et de la mise en œuvre de cette pratique puisque tous les tests ont passé. En revanche, nous n'avons pas eu l'occasion de mettre en place des tests unitaires pour la ligne de commande en raison des difficultés rencontrées lors de son implémentation. Premièrement, la prise en main de picoCLI a pris plus de temps qu'anticipé. Ensuite, la rédaction des tests était orientée vers des tests d'intégration ce qui a entravé le développement car ils étaient complexes. Cependant, comme nous avons changé plusieurs fois la structure du parseur, la rédaction des tests à pris du temps de développement. De plus, pour la même raison les tests finaux du parseurs n'ont pas été implémentés.

2.5 Commit early, commit often

Nous avons essayé d'effectuer des commits le plus régulièrement possible avec des messages explicites. N'ayant pas jugé cela utile, nous n'avons pas mis en place de convention de nommage stricte pour les messages de commit, il nous semblait évident que ceux-ci se doivent d'être clairs et concis.

Nous avons une marge d'amélioration vis-à-vis de cet aspect. En effet, comme mentionné ci-dessus, nous avons pris plus de temps que prévu sur ce sprint et avons été bloqué.e.s à plusieurs reprises ; ce qui a rendu la fréquence des commits plus faible.

Notamment, nous avons rencontré des soucis avec la story command line ; lors de ce sprint Olivier a eu de la difficulté à réaliser la feature version et est resté bloqué dessus. Étant donné la répartition des tâches, le projet était en attente sur cette résolution, pour laquelle aucun commit n'a été réalisé.

3 SPRINT II – DÉVELOPPEMENT AGILE

3.1 Conception incrémentale et décomposition

La conception incrémentale et la décomposition ont été difficiles à mettre en place pour ce second sprint car nous avions pris trop de retard lors du premier sprint. Nous avons donc décidé de récupérer directement le code fourni sur le github “statique” afin de repartir sur de bonnes bases pour le troisième sprint.

3.2 Stories

Les stories étaient déjà préparées grâce à la consigne mais en raison du retard, nous avons préféré investir le temps imparti dans l'organisation de la méthodologie. Et nous avons compté sur la correction du sprint 2 afin de récupérer une base de code saine.

3.3 Refactoring

Nous avons décidé d'abandonner notre version du projet afin de repartir sur une base nouvelle et saine en utilisant la correction mise à disposition sur github. Nous devons refactoriser la commande build avec une nouvelle classe car celle-ci n'est pas implémentée de manière optimale dans le code fourni et respecte mal la séparation des concepts. L'objectif est de factoriser cette commande durant le prochain sprint pour s'approprier le nouveau code.

3.4 Test d'intégration et tests systèmes

Nous n'avons pas eu le temps d'écrire des tests d'intégration et des tests systèmes car nous n'avions tout simplement pas de code à tester en raison des problèmes rencontrés lors du premier sprint. Cependant, de tels tests étaient déjà présents dans la correction fournie sur github et nous les avons donc récupérés tels quels pour notre projet.

3.5 Automatisation

Nous n'avons pas eu l'occasion d'appréhender d'action particulièrement répétitive lors de ce sprint. Cependant, nous avons vu après le sprint que d'autres groupes avaient automatisé leur Kanban ou le release. Vu notre avancement, nous ne voulons pas perdre de temps avec l'automatisation du release mais songeons à automatiser le kanban pour le prochain sprint.

3.6 Commit early, commit often

En raison d'une grande difficulté à faire fonctionner les outils notamment lors du premier sprint et de la mise en place de la méthodologie – comme mentionné dans la section 2.5 du Sprint I – nous n'avons malheureusement pas eu l'occasion d'effectuer de nombreux commits. Cela représente pour nous un gros échec au vu de l'importance de commits réguliers ; nous avons cependant à ce stade établi un plan plus intensif pour la semaine à venir, ceci afin d'être entièrement à jour sur le projet et commencer le Sprint III sur un bon pied.

Aussi, nous avons prévu d'établir un plan d'objectifs plus précis quant aux échéances de ces derniers pour les prochains Sprint et de nous retrouver hebdomadairement si besoin afin d'être à jour et de pouvoir discuter des problèmes rencontrés si besoin.