Compte Rendu DevOO

Hexanome: 4105

Alexis Andra
Jolan Cornevin
Mohamed Haidara
Alexis Papin
Robin Royer
Maximilian Schiedermeier
Davis Wobrock

3 décembre 2015

Table des matières

Chapitre 1

Capture et Analyse des besoins

1.1 Planning prévisionnel du projet

1.2 Modèle du domaine

1.3 Glossaire

1.4 Diagramme de cas d'utilisation

1.5 Description textuelle structurée des cas d'utilisation

Chapitre 2

Conception

2.1 Liste des événements utilisateur et diagramme États-Transitions

2.2 Diagrammes des packages et de classes

2.3 Document expliquant les choix architecturaux et design patterns utilisés

2.4 Diagramme des sequence du calcul de la tournée a partir d'une demande de livraison

Chapitre 3

Implémentation et tests

3.1 Code du prototype et des tests unitaires

3.2 Documentation JavaDoc du code

3.3 Diagramme de packages et de classes rétro-générés à partir du code

Chapitre 4

Bilan

4.1 Planning effectif du projet

4.2 Bilan humain et technique

4.2.1 Bilan humain

Coordonner le projet et notamment distribuer les taches dans l'équipe était un devoir délicat. D'un cote un but était la parallélisation les fils de développement le plus possible. D'autre part il y avait beaucoup des dépendances entre les étapes prévus. C'est pour ça qu'au début du projet la priorité était d'identifier une procédure raisonnable qui permettait d'identifier les stations critiques. Comme je n'étais jamais avant en charge d'une équipe assez nombreuse et puissant je dois admettre que d'abord j'avais sous-estimé ce défi.
Le résultat de cet effort nous a bien permit de profiter au maximum des ressources humaines. Néanmoins il était important de tenir tout l' équipe en courant en ce que regarde les développements qui se sont déroule dans les cotes divers du projet en parallèle. En particulier la convention de bien commenter son code en combinaison avec une communication fréquente nous ont permis de maitriser cette mission.
Personnellement je le trouvait éprouvant de trouver une raisonnable granularité pour la distribution des taches ouverts dans l'équipe. Si une charge est trop petit, on risque de perdre du temps en expliquant le contexte. Aurait elle pu réalisé plus rapide directement par la personne qui s'en a déjà occupe ? Sinon, quand les taches sont trop complexe on risque qu'un développeur sera bloqué et exclu des événements qui se passent concurremment. Comme la section avant a relevé, la nombre des heures passes sur le project par développeur n'est pas parfaitement équilibré. Cependant on ne doit pas oublier que les chiffres ne correspondent pas toujours précisément au travail réalisé. Notamment le temps passe avec planification, communication et réflexion n'est pas facile à mesurer. Pour ça il me semble plus avéré d'évaluer l'équipe comme un entité indivisible. Parlant de l'équipe, on peut résumer que on était enchanté de pouvoir observer la réalisation de nos conceptions. Même s'il y avait parfois des défis qu'on n'avait pas prévu, l'équipe était toujours motivé de discuter et trouver la meilleure solution possible.

4.2.2 Bilan technique

Pour réaliser ce projet nous avons profite d'une grande variété des techniques. Au but de permettre à chaque développeur de travailler avec son IDE préféré, nous avons décidé de ne pas garder les fichiers de gestion d'IDE dans le répertoire git. Grâce a çà, nous avons réussi à travailler avec des IDEs *NetBeans*, *Eclipse* et *IntelliJ* en même temps. Concernant les fichiers de conception nous avons également utilisé plusieurs outils :

- La conception des diagrammes de classes était réalisé avec le logiciel *UMLet*[1]. Les fichiers utilisés par ce logiciel étaient également partage dans le répertoire git.
- Pour tous les autres diagrammes on a profite des outils en ligne, particulièrement *Draw.io*, disponible sur *Google Drive*.
- Pour automatiquement charger les librairies dont on avait besoin, on a intégré *Apache Maven*. On pourrait argumenter que l'intégration de Maven n'était pas forcement nécessaire, car il y avait seulement quatre dépendances. Néanmoins son intégration était très confortable pour l'équipe parce qu'on a jamais perdu du temps en ressoudant les dépendances soulevé par un collègue.
- Les tests fonctionnels et unitaires étaient réalises en utilisant *JUnit*. Même si les tests ont contribue beaucoup à trouver et résoudre des erreurs, on aurait encore pu intensifier leur intégration. On a suivi le conseil de jamais laisser un développeur tester son propre code. Quoique ce soit en général une bonne pratique, on avait parfois pas assez de communication entre un auteur d'un classe et le testeur. Ça peut causer que les tests écrit ne peuvent pas passer, car ils n'utilisent pas les interfaces la manière prévu.

Finalement on peut résumer que même si il s'agitait d'un projet complexe et qu'on avait pas beaucoup du temps pour le réaliser, nous avons choisi les bonnes outils. Celles ci nous avons permis d'avancer rapidement et créer un produit fiable.

1. http ://www.umlet.com/