

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

| | | |
|----------|---|-----------|
| 1 | Capture et Analyse des besoins | 5 |
| 1.1 | Planning prévisionnel du projet | 5 |
| 1.2 | Modèle du domaine | 5 |
| 1.3 | Glossaire | 5 |
| 1.4 | Diagramme de cas d'utilisation | 5 |
| 1.5 | Description textuelle structurée des cas d'utilisation | 5 |
| 2 | Conception | 7 |
| 2.1 | Liste des événements utilisateur et diagramme États-Transitions | 7 |
| 2.2 | Diagrammes des packages et de classes | 7 |
| 2.3 | Document expliquant les choix architecturaux et design patterns utilisés | 7 |
| 2.4 | Diagramme des sequence du calcul de la tournée a partir d'une demande de livraison | 7 |
| 3 | Implémentation et tests | 9 |
| 3.1 | Code du prototype et des tests unitaires | 9 |
| 3.2 | Documentation JavaDoc du code | 9 |
| 3.3 | Diagramme de packages et de classes rétro-générés à partir du code | 9 |
| 4 | Bilan | 11 |
| 4.1 | Planning effectif du projet | 11 |
| 4.2 | Bilan humain et technique | 11 |
| 4.2.1 | Bilan humain | 11 |
| 4.2.2 | Bilan technique | 12 |

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 tâches dans l'équipe était un devoir délicat. D'un côté un but était de paralléliser les fils de développement le plus que possible. D'autre part il y avait beaucoup des dépendances entre les étapes prévus. C'est pour ça qu'au début de projet la priorité était de 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 permis 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 déroulés dans les cotes divers du projet. En particulier la convention de bien commenter son code en combinaison avec une communication fréquente nous ont permis de maîtriser cette mission. Personnellement je le trouvais éprouvant de trouver une raisonnable granularité pour la gestion des tâches ouverts dans l'équipe. Si une charge est trop petit, on risque de perdre du temps en expliquant son contexte. Peut-être elle aurait pu être réalisé plus rapide directement par la personne qui s'en a déjà occupé. Sinon, quand les tâches sont trop complexes on risque qu'un développeur sera bloqué pour trop long et sera exclu des événements qui se passent concurremment. Comme la section avant a relevé, la nombre des heures passées sur le projet 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 motivée de discuter et trouver la meilleure solution ensemble.

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 ça, nous avons réussi à travailler avec des IDEs *NetBeans*, *Eclipse* et *IntelliJ* en même temps. Concernant les fichiers de conception nous avons également utilise 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 a besoin, on a intégré maven. On pourrait argumenter que l'integration de maven n'était pas forcément nécessaire, car il y avait seulement quatre dependances. Néanmoins son intégration était très confortable pour l'équipe parce qu'on na pas 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 idée, on avait parfois pas assez de communication entre l'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.

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