

# Compte Rendu DevOO

Hexanome: 4105

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

3 décembre 2015



# Table des matières

<b>1</b>	<b>Capture et Analyse des besoins</b>	<b>5</b>
1.1	Planning prévisionnel du projet . . . . .	5
1.2	Modèle du domaine . . . . .	7
1.3	Glossaire . . . . .	7
1.4	Diagramme de cas d'utilisation . . . . .	7
1.5	Description textuelle structurée des cas d'utilisation . . . . .	7
<b>2</b>	<b>Conception</b>	<b>9</b>
2.1	Liste des événements utilisateur et diagramme États-Transitions	9
2.2	Diagrammes des packages et de classes . . . . .	9
2.3	Document expliquant les choix architecturaux et design patterns utilisés . . . . .	9
2.4	Diagramme des sequence du calcul de la tournée a partir d'une demande de livraison . . . . .	9
<b>3</b>	<b>Implémentation et tests</b>	<b>11</b>
3.1	Code du prototype et des tests unitaires . . . . .	11
3.2	Documentation JavaDoc du code . . . . .	11
3.3	Diagramme de packages et de classes rétro-générés à partir du code	11
<b>4</b>	<b>Bilan</b>	<b>13</b>
4.1	Planning effectif du projet . . . . .	13
4.2	Bilan humain et technique . . . . .	14
4.2.1	Bilan humain . . . . .	14
4.2.2	Bilan technique . . . . .	15



# Chapitre 1

## Capture et Analyse des besoins

### 1.1 Planning prévisionnel du projet

Le planning prévisionnel du projet s'est déroulé en trois étapes. Au début on a identifié les stations nécessaires pour réaliser le projet systématiquement. Cette découpage en missions réalisables est exprimé par la liste suivante :

- Planning prévisionnel du projet (3.0 h)
- Analyse du modèle (3.0h)
- Conception du modèle (2.5 h)
- Analyse des *CUs*, conception des *diagrammes des sequence* (3.5 h)
- Description textuelle des *CUs* (4.0 h)
- Implémentation des classes représentant données des *XMLs* (1.5 h)<sup>1</sup>
- Conception d'IHM (prototype) (8.0 h)
- Conception d'IHM (précise) (8.0 h)
- Spécification '*événements utilisateur*' (4.0 h)
- Conception du diagramme *États-transitions* (6.0h)
- Implémentation du serialiseur / deserialiseur (8.0 h)
- Spécification des Interfaces du package contrôleur (5.0 h)
- Spécification des Interfaces du package vue (3.0 h)
- Spécification des Interfaces du package modele (5.0 h)
- Conception de diagramme des classes du package controleur (6.0 h)
- Conception de diagramme des classes du package vue (3.0 h)
- Conception de diagramme des classes du package modele (6.0 h)
- Implémentation des observateurs qui notifient la vue (4.0 h)
- Réalisation de l'IHM (package vue) (20 h)
- Mock implémentation des classes du package controleur (3.5 h)
- Conception et réalisation des tests unitaires (20 h)
- Conception et réalisation des tests fonctionnels (20 h)
- Implémentation du package contrôleur (30 h)
- Implémentation du package vue (20 h)

---

1. La planification d'une heure et demi peut apparaître insuffisant pour implémenter un modèle. Au moment du conception de l'architecture, le modèle était compris comme une ensemble des Java Beans. La réalisation des algorithmes était prévu dans le package contrôleur.

- Implémentation du package modèle (10 h)
- Rétro-génération des diagrammes classes a partir du code (3.0 h)
- Rédaction du glossaire (2.0 h)

Deuxièmement on a essayé de trouver toutes les dépendances entre les missions prévus. Le but de cette démarche était de pouvoir paralléliser les développements. La graphique suivant montre le déroulement prévisionnel du projet et également qui était le responsable prévu pour chaque étape.

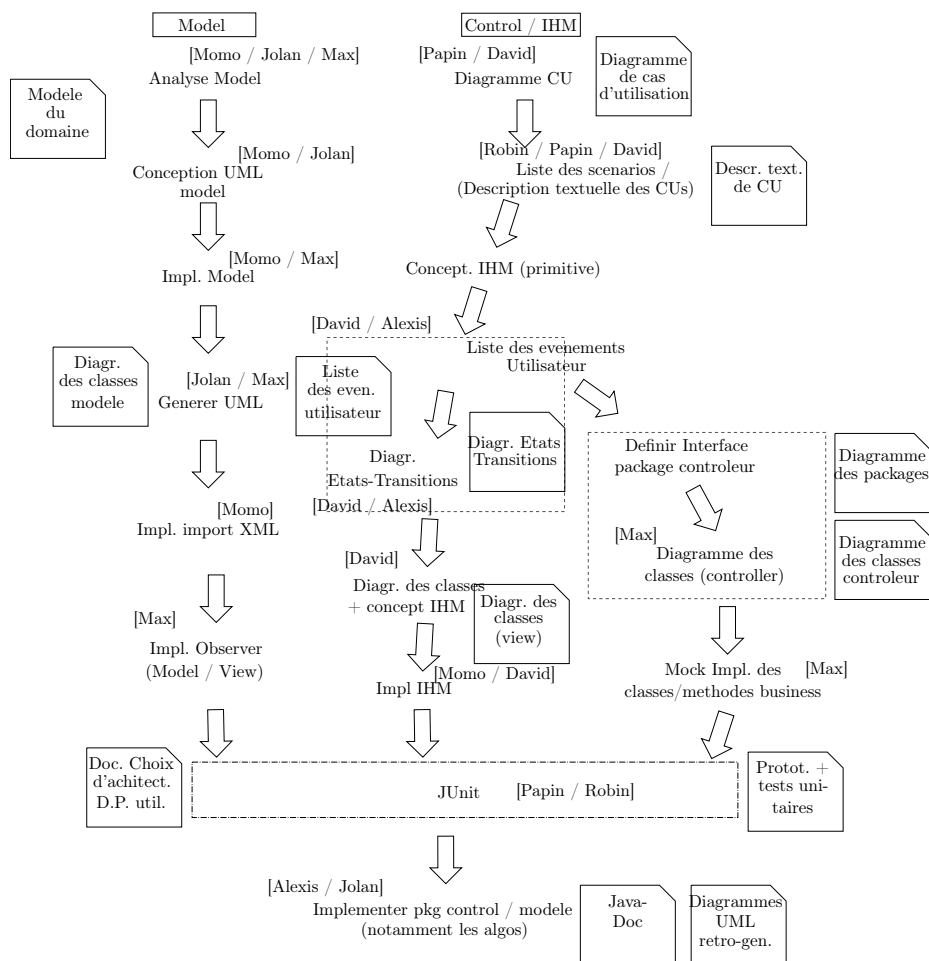


FIGURE 1.1 – Dépendances et déroulement prévisionnel du projet

Dans la troisième et dernière étape, on a essayé d'estimer le temps nécessaire pour réaliser ces tâches en utilisant redmine.

<input type="checkbox"/>	19	Anomalie	Nouveau	Diagrammes Etc- Transitions	Alexis Andra	11/15/2015 11:44 AM	max allemand		6.00	0.00		11/12/2015	11/12
<input type="checkbox"/>	18	Evolution	Résolu	Liste des evenements Utilisateurs	Alexis Andra	11/16/2015 04:18 AM	max allemand	11/13/2015	4.00	0.00		11/12/2015	11/12
<input type="checkbox"/>	17	Anomalie	Résolu	Conception UML (primitive)	max allemand	11/15/2015 09:17 AM	max allemand	11/13/2015	8.00	4.00		11/12/2015	11/12
<input type="checkbox"/>	16	Evolution	Nouveau	Implementer Observer	jolan cornevin	11/12/2015 12:41 PM	max allemand	11/15/2015	4.00	0.00		11/14/2015	11/12
<input type="checkbox"/>	15	Anomalie	Nouveau	Implt import XML	Mohamed El Mouctar Hadjara	11/16/2015 02:00 PM	max allemand	11/14/2015	8.00	4.00		11/13/2015	11/12
<input type="checkbox"/>	14	Anomalie	En cours	Conception UML model	jolan cornevin	11/12/2015 12:27 PM	max allemand	11/13/2015		0.00		11/12/2015	11/12
<input type="checkbox"/>	9	Assistance	Nouveau	Planning previsionnel Redmine	max allemand	11/12/2015 04:58 AM	Redmine Admin	11/12/2015	1.00	0.00		11/12/2015	11/12
<input type="checkbox"/>	8	Anomalie	Nouveau	Implt. Model	jolan cornevin	11/12/2015 12:23 PM	Redmine Admin	11/13/2015	1.50	0.00		11/12/2015	11/12
<input type="checkbox"/>	7	Evolution	En cours	Generer UML (model)	Mohamed El Mouctar Hadjara	11/12/2015 12:24 PM	Redmine Admin	11/12/2015	3.00	0.00		11/12/2015	11/12
<input type="checkbox"/>	6	Evolution	En cours	Planning previsionnel du projet	max allemand	11/12/2015 12:22 PM	Redmine Admin	11/12/2015	2.00	0.00		11/12/2015	11/12
<input type="checkbox"/>	5	Evolution	En cours	Glossaire	Alexis Papin	11/15/2015 03:05 PM	Redmine Admin	11/12/2015	2.00	2.00		11/12/2015	11/12
<input type="checkbox"/>	4	Evolution	En cours	Liste des Scenarios	Alexis Andra	11/15/2015 11:43 AM	Redmine Admin	11/12/2015	4.00	0.00		11/12/2015	11/12

(1-19/19)

FIGURE 1.2 – Capture d'écran de la gestion du projet avec redmine

## 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

Le planning prévisionnel nous a aide beaucoup à réaliser ce projet dans une manière bien pensée. La connaissance des dépendances entre les modules nous permettait d'éviter que les développeur étaient bloqués et également de réaliser les tâches dans la bonne ordre.

Developpeur	Tache		Total
Alexis A.	Conception CUs	4h	20h
	Diagramme état-transition	4h	
	Diagramme séquence calcul tournée	4h	
	Développement calcul tournée	8h	
Jolan	Implementation JUnit	4h	48h
	Implementation du Dijkstra	25h	
	Développement modèle	10	
	Conception diagramme modèle	2h	
	Conception diagramme séquence	3h	
	Correctinos des bugs	4h	
Mohamed	Conception diagramme modèle	2h	57h
	Développement Vue	29h	
	Développement Contrôleur	16h	
	Documentation	4h	
	Correction des bugs	5h	
	Redaction Design Patterns	1h	
Alexis P.	Conception CUs	4h	20h
	Diagramme état-transition	4h	
	Diagramme séquence calcul tournée	4h	
	Développement calcul tournée	8h	
Robin	Conception IHM	6h	14h
	JUnit modèle	8h	
Max	Planning previsionel	4h	71h
	Conception architecture et interfaces	9h	
	Development Controleur	18h	

	Development Modele	26h	
	Gestion des taches	3h	
	Gestion et réalisation des livrables	8h	
	Development JUnit Modele	3h	
David	Conception CUs	4h	52h
	Diagramme Sequence	3h	
	Développement vue	20h	
	Documentation	8h	
	JUnit Controleur	5h	
	Correction des bugs	10h	
	Redaction rapport	2h	

Comme le relevé en haut montre, le planning prévisionnel était suffisamment précis pour ce que concerne la planification et conception. Par contre on n'a pas réussi à réaliser les implémentations dans le temps prévu. Il y a plusieurs facteurs qui ont contribué à ce retardement. Parmi eux voici les plus importants :

- On était demandé de profondément changer l'architecture dans un moment où la plupart des classes étaient déjà implémentées.
- Même si pas demandé, on était motivé de trouver la meilleure solution possible. Pour ça on a parfois implémenté plusieurs alternatives. L'implémentation d'annulation des commandes est une bonne exemple.
- Les interfaces et classes fournies qui étaient menées à nous simplifier le calcul du chemin optimal, n'étaient pas très bien documentées et il n'était pas évident comment il fallait les utiliser pour en profiter dans le cadre 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 la parallélisation des fils de développement le plus possible. D'autre part il y avait beaucoup de dépendances entre les étapes prévues. 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 puissante 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 toute l'équipe en courant en ce qui regarde les développements qui se sont déroulés dans les délais divers du projet en parallèle. 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 distribution des tâches ouverts dans l'équipe. Si une tâche est trop petite, on risque de perdre du temps en expliquant le contexte. Aurait-elle pu être réalisée plus rapidement directement par la personne qui s'en est déjà occupée ? Sinon,

quand les tâches sont trop complexes on risque qu'un développeur sera bloqué et exclu des événements qui se passent concurremment. Comme la section avant a relevé, le 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 une 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 possible.

### 4.2.2 Bilan technique

Pour réaliser ce projet nous avons profité 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 à ça, 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 réalisée avec le logiciel *UMLet*<sup>1</sup>. Les fichiers utilisés par ce logiciel étaient également partagés dans le répertoire git.
- Pour tous les autres diagrammes on a profité des outils en ligne, particulièrement *Draw.io*, disponible sur *Google Drive*.
- Pour automatiquement charger les bibliothèques dont on avait besoin, on a intégré *Apache Maven*. On pourrait argumenter que l'intégration de Maven n'était pas forcément 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 soulevées par un collègue.
- Les tests fonctionnels et unitaires étaient réalisés en utilisant *JUnit*. Même si les tests ont contribué 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'une classe et le testeur. Ça peut causer que les tests écrits ne peuvent pas passer, car ils n'utilisent pas les interfaces la manière prévue.

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

---

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