|  |
| --- |
| Les patriotes |
| Méthodologies Agiles et Ingénierie Logicielle |
| Classe fétiches : Baby-sitter et Enfant  BIT – Magali  GUILLAUME – Josselin  MARY – Laurent  QUARCIA - Anthony |

|  |
| --- |
|  |

# Introduction

Sommaire

[Introduction 1](#_Toc413177443)

[Synthèse 3](#_Toc413177444)

[1) Rôles et contributions 3](#_Toc413177445)

[2) Hypothèses et contexte de travail 3](#_Toc413177446)

[3) Originalité de la solution, valeur ajoutée perçue 4](#_Toc413177447)

[4) Joies et peines 4](#_Toc413177448)

[Outils et méthodes utilisés 5](#_Toc413177449)

[Github 5](#_Toc413177450)

[Acunote 5](#_Toc413177451)

[Plugin Eclemma 5](#_Toc413177452)

[Pair Programming 5](#_Toc413177453)

[Test Driven Development 5](#_Toc413177454)

[Les design patterns 6](#_Toc413177455)

[Diagramme de cas d’utilisation 7](#_Toc413177456)

[Diagramme d’état : 8](#_Toc413177457)

[Scénario d’exécution :   9](#_Toc413177458)

[Conclusion 10](#_Toc413177459)

# Synthèse

## Rôles et contributions

Notre groupe est composé en deux équipes : Magali BIT et Laurent MARY sont en charges des aspects techniques de la solution (Développement, tests unitaires), tandis qu’Anthony QUARCIA et Josselin GUILLAUME représentent la partie métier, utilisateur.

Anthony et Josselin ont donc imaginé un scénario mélangeant les deux classes fétiches (Dresseur et Enfant) utilisées. Il en a résulté la gestion des baby-sitters. Ils ont donc ensuite formulé des tests fonctionnels et les ont proposés comme GuideLine à « l’équipe de développement » afin de respecter le « Test Driven Development ». L’écriture des tests et du code a été fait en respectant la politique du « Pair Programming». Il y a eu une évolution des besoins tout au long du projet grâce aux pratiques AGILE.

De plus, « l’équipe métier » a dépeint la solution à travers des diagrammes présents dans la suite de ce rapport afin de faciliter la compréhension des attentes à chaque du projet.

Suite à l’écriture des tests fonctionnels par les représentants des utilisateurs, Magali BIT et Laurent MARY ont développé le squelette du code, le corps des méthodes créées ainsi que les design patterns requis. De plus, ils ont testé unitairement toutes les méthodes afin de s’assurer de la cohésion de chacune d’entre elles. Cela a été fait en « Pair Programming » ce qui a permis d’aller plus vite dans la compréhension des besoins pour les développeurs et la compréhension de la solution pour les métiers.

## Hypothèses et contexte de travail

Concernant le contexte de travail, nous avons décidé de diviser le travail en plusieurs sprints. Tout d’abord le l’ « Acte 1 » et l’ « Acte 2 » dont les périodes temporelles communes. L’acte 1 a pour objectif de définir les « User Stories », le délai 3 jours a été respecté. L’acte 2 quant à lui, est réservé à la mise en place technique de la solution. Son délai de 4 jours a également été respecté même si certains outils ont été ajoutés par la suite. L’acte 3 concerne le premier jet d’écriture des tests fonctionnels. Il a donc concerné une partie des métiers (ayant des compétences en programmation) et des développeurs (respect du « Pair Programming »). Cet acte 3 représente, comme prévue, le mieux l’aspect Agile de notre travail car il contient l’écriture des tests fonctionnels et le code de la solution respectant les pratiques Agiles, mais également l’expression d’une erreur fonctionnelle par les développeurs puis corrigée par l’équipe métier au sein d’un fichier partagée remplaçant Google Code. L’acte 4, la suite et fin des tests fonctionnels, a donc été prévue à la suite.

## Originalité de la solution, valeur ajoutée perçue

Notre solution permet d’affecter des baby-sitters à des enfants devant être gardés. Notre algorithme d’optimisation permet de proposer aux parents une liste de baby-sitters « optimale ». Les baby-sitters proposées seront celles :

* situé proche de l’habitation des parents
* ayant les meilleurs ranking selon les autres utilisateurs de la solution
* proposant les tarifs les plus attractifs

Nous sommes donc en mesure de garantir un service optimale et cela pour un coût minimum. Cette solution aurait donc toute sa place dans un site Web de mise en relation Parents/Baby-sitters.

## Joies et peines

N’ayant jamais pratiqué les méthodes Agiles, ce projet nous a permis de réaliser à quel point le « Pair Programming » est bénéfique et permet à la solution d’être efficace et rapide. Nous avons donc pris du plaisir à l’appliquer. Il fut cependant compliqué, parfois impossible, de mettre en accord tous les outils. C’est pourquoi nous avons utilisé Eclemma à la place de Cobertura (problème à l’installation). Par ailleurs, nous n’avons pas utilisé Google Code car nous n’en n’avons pas trouvé l’intérêt par rapport à un fichier partagé sur GitHub.

Il est important d’insister sur l’aspect « ludiques » des pratiques agiles employées. Nous sommes en effet bien loin du classique et peux glorifiant du cycle en V. Avec la mise en place de cycles itératifs très courts, il est possible de voir le résultat de notre travail et cela est bien plus motivant que de devoir attendre la fin du cycle en V.

Enfin, le développement piloté par les tests revêt également un caractère ludique et il est toujours plaisant de faire passer notre résultat de test du rouge au vert.

# Outils et méthodes utilisés

## Github

GitHub est un service web d'hébergement et de gestion de développement de logiciels, nous l’avons donc utilisé afin de partager le code de notre projet mais également quelques annexes (diagrammes, impressions d’écrans, …).

## Acunote

Acunote est un outil utilisable via une interface Web afin de lister les tâches du projet ainsi que leurs avancements mais également dans le but de travailler en méthodologie Agile. Cet outil nous a ainsi permis de fournir une visibilité sur le projet à tous les membres qui le compose.

## Plugin Eclemma

EclEmma est un outil libre d'assurance de code Java pour l’IDE Eclipse, c’est un plugin qui est combiné à JUnit. L'outil espionne le code compilé et effectue un marquage de celui-ci selon un code couleur.

Les couleurs utilisés par défaut sont :

* La couleur verte pour les lignes totalement couverte,
* La couleur jaune pour afficher le code que les tests unitaires couvrent partiellement.
* la couleur rouge est utilisée pour afficher le code que les tests unitaires ne couvrent pas

Ce plugin nous a donc permis de de vérifier d'un coup d'œil les bouts de code qui sont pris en charge ou non.

## Pair Programming

Pair Programming (en français « Programmation en binôme ») est une méthode de travail où deux développeurs sont amenés à travailler sur un même poste de travail.

Cette technique nous a permis de programmer avec l’objectif d’obtenir un code de qualité et d’augmenter notre efficacité.

## Test Driven Development

Test Driven Development (appelé en français « Développement piloté par les tests ») est une technique de développement de logiciel qui préconise d'écrire les [tests fonctionnels](http://fr.wikipedia.org/wiki/Test_unitaire) avant d'écrire le code source de l’application.

La méthode à adapter pour utiliser cette technique est la suivante :

* écrire un premier test
* vérifier que le test codé échoue afin de vérifier sa validité
* écrire juste le code suffisant pour passer le test
* vérifier que le test est désormais valide
* refactoriser le code, c'est-à-dire l'améliorer tout en gardant les mêmes fonctionnalités

Cette technique nous a permis d'obtenir un code plus cohérent et de clarifier des détails sur le fonctionnement de l’application.

## Les design patterns

Nous nous sommes rendu compte lors de nos itérations agiles que l’utilisation de design pattern s’avérait nécessaire.

En effet, dans notre projet, nous avons à gérer :

* des listes de BabySitters, Parent et Enfant
* des statuts pour nos objets Enfant

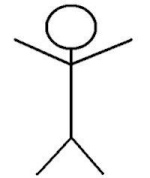
Les design patterns correspondant à une solution éprouvée à un problème précis.

Il s’avère que l’application des design patterns Singleton, Iterator et State apportait une solution à l’ensemble de nos problèmes.

En effet, il nous semblait nécessaire de gérer les différentes listes d’objets manipulés dans l’application : c’est pourquoi, nous avons pensé à utiliser les patterns Singleton et Iterator :

* **Singleton** nous permet d’avoir des classes qui permettent de gérer des listes de d’objets (par exemple : *GestionParent* possède une liste de *Parent*), il était donc nécessaire de s’assurer qu’il y a qu’une seule liste de *Parent* gérée au sein de l’application. C’est pour cela qu’on a appliqué le pattern Singleton pour les *GestionParent*, *GestionEnfant*, *GestionBS* qui gèrent respectivement les listes *Parent*, *Enfant* et *BabySitter* et s’assurer qu’elles soient uniques
* Comme nous manipulons de nombreuses listes au sein de l’application, l’utilisation du pattern **Iterator** s’est avérée vite nécessaire. En effet, il est probable lors de futures évolutions de l’application qu’on éprouve le besoin de gérer la manière dont on parcourt les listes en vue de la croissance de l’application et de la gestion des données

# Diagramme de cas d’utilisation



Utilisateur

L’utilisateur a le choix entre deux systèmes de gestion :

* La gestion des parents
* La gestion des baby-sitters

Le parent pourra créer un nouvel enfant qui n’était pas enregistré. Il pourra également emmener son enfant chez la baby-sitter et le récupérer.

Concernant les baby-sitters, ils leur seront possibles de nourrir un enfant ou de le mettre au lit.

# Diagramme d’état :

EnfantChezParents

EnfantAuLit

EnfantAMangé

EnfantChezBS

Notre programme implique de nombreux changements d’états pour l’enfant. Il s’agit d’une fonctionnalité essentielle de notre programme. En effet, les passages entre les différents états sont fréquents au cours de l’exécution de notre programme. Certaines règles ont donc été mises en place par l’équipe métier afin de définir les changements d’états possibles et impossibles. Tout ceci a permis de faciliter la tâche des développeurs.

On constate donc qu’un enfant ayant le statut « EnfantChezParents » ne peut obtenir QUE le statut « EnfantChezBS ».

Par ailleurs, on constate qu’un parent peut récupérer un enfant dès que celui-ci a le statut « EnfantChezBs » ou « EnfantAMangé » ou « EnfantAuLit ».

# Scénario d’exécution :

Ecran d'accueil

Ecran d'accueil

Gestion parent

Gestion baby-sitter

Créer parent

Choisir parent existant

Choix de gestion

Choix de gestion

Choix de gestion

Créer enfant

Choisir enfant existant

Choisir baby-sitter

Emmener Enfant

Récupérer Enfant

# Conclusion