Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
polytech.univ-tours.fr

Projet de programmation et génie logiciel

2018-2019

Création de fond de planning

Reprise de projet

Tuteur académique
Christophe Lenté

Étudiants
Théo Bertet (DI4)
Alexia Neufond (DI4)

6 janvier 2019

Liste des intervenants

Nom	Email	Qualité
Théo Bertet	theo.bertet@etu.univ-tours.fr	Étudiant DI4
Alexia Neufond	alexia.neufond@etu.univ-tours.fr	Étudiant DI4
Christophe Lenté	christophe.lente@univ-tours.fr	Tuteur académique, Département Informatique

Avertissement

Ce document a été rédigé par Théo Bertet et Alexia Neufond susnommés les auteurs.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Christophe Lenté susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

Les auteurs reconnaissent assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

Les auteurs attestent que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

Les auteurs attestent ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

Les auteurs attestent que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

Les auteurs reconnaissent qu'ils ne peuvent diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

Les auteurs autorisent l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document

Théo Bertet et Alexia Neufond, *Création de fond de planning: Reprise de projet*, Projet de programmation et génie logiciel, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2018-2019.

```
@mastersthesis{
    author={Bertet, Théo and Neufond, Alexia},
    title={Création de fond de planning: Reprise de projet},
    type={Projet de programmation et génie logiciel},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2018-2019}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document `polytech.cls` V2.2.5 (2017/05/02)

Table des matières

Liste des intervenants a

Avertissement b

Pour citer ce document c

Table des matières i

Table des figures iii

1 Introduction 1

1 Contexte du projet 1

2 Objectif du projet 1

2 Environnement de travail 2

1 Langage de programmation 2

2 Outils de gestion de projet 2

3 Analyse des besoins 3

1 Spécifications du logiciel 3

2 Modélisation 5

4 Planification 7

1 Définition des tâches 7

2 Répartition des tâches 9

3 Diagramme de Gantt 9

5 Tests applicatifs **11**
1 Définition des tests 11
2 Résultats des tests 11

6 Gestion des erreurs **13**
1 Changement de la version de Java 13
2 Résolution des erreurs signalées 14
3 Journalisation 16

7 Automate de vérification **18**
1 Pourquoi un automate ? 18
2 Modélisation 18
3 Programmation 20

Conclusion 22

Annexes 24

A Exemple de fiche de test **25**
1 Exemple 1 — Cas nominal 25
2 Exemple 2 — Cas anormal 26

Table des figures

3 Analyse des besoins

1 Diagramme de classe fournie 5
2 Ajout au diagramme de classe 6

4 Planification

1 Tâches prévues sur l'outil en ligne Trello 7
2 Planning initial du projet effectué sur Toms Planner 10
3 Planning adapté du projet effectué sur Toms Planner 10

6 Gestion des erreurs

1 Fichier de configuration de Log4J - Logback.xml 17

7 Automate de vérification

1 Première version de l'automate 19
2 Automate factorisé pour le programme 20

1 Introduction

1 Contexte du projet

Ce projet repose sur la reprise d'un projet de programmation et génie logiciel de l'année 2017-2018. Le programme que nous reprenons permet la génération d'un fond de planning sur la base de fichier Excel pour permettre de faciliter le travail de création des emplois du temps. Cependant, le programme n'est pas totalement fonctionnel. Certains fichiers d'entrée ne permettent pas la génération sans erreur et n'ont pas assez de précision quant à la cause de ces erreurs ; Elles retournent l'erreur "null" avec certains fichiers.

2 Objectif du projet

L'objectif de ce projet est de rendre le programme plus simple d'utilisation en mettant en place une gestion d'erreur plus claire, permettant à l'utilisateur de savoir quel fichier cause l'erreur et où il la cause. Nous devons de plus résoudre des problèmes d'utilisation rencontrés par l'utilisateur sur des fichiers fonctionnels.

2 Environnement de travail

1 Langage de programmation

Le logiciel étant pré-existant, le langage Java nous était imposé. Il était disponible sous sa version 8. Le projet a été réalisé avec le gestionnaire de dépendances **Maven** qui nous était donc aussi imposé.

2 Outils de gestion de projet

Planification

La planification du projet s'est faite avec l'outil en ligne **TomsPlanner** permettant de créer un planning pour le projet accessible en ligne pour tous les membres de l'équipe. La planification sera détaillée plus loin dans le document.

Avancement des tâches

L'avancement des tâches a été visualisé au travers du *Kanban* en ligne **Trello**. Il permet de lister et visualiser toutes les tâches du projets et de connaître leur avancement au travers de trois listes **To do, Doing** et **Done** ainsi qu'au travers de *checklist* permettant de connaître le pourcentage d'avancement d'une tâche particulière.

Travail en parallèle et gestion de version

Ce projet s'effectuant en binôme, il nous a fallu pouvoir interagir sur le code simultanément. Pour cela, nous avons mis en place l'outil de *versionning* et travail en équipe **Git**. Celui-ci nous permettait d'être sur deux postes différents et de mettre aisément en commun nos différents travaux sur le projet. **Git** de par sa licence étudiante nous permettait de centraliser le code dans un projet privé.

3 Analyse des besoins

Le logiciel qui nous a été fourni permet d'analyser des fichiers afin de générer un ou plusieurs fond de planning. Les besoins ajoutés au projet existant sont principalement de l'ordre fonctionnel. Il s'agit ici de résoudre les problèmes liés à la gestion d'erreur.

Nous devons ainsi dans un premier temps analyser des maquettes fournies afin de comprendre ce qui les rend non valides au près de l'application. Dans un deuxième temps, nous devrons mettre en place une gestion d'erreur plus claire et précise pour que l'utilisateur puisse cibler la cause d'une erreur.

1 Spécifications du logiciel

Le programme étant déjà pré-existant et notre binôme n'ayant pas à toucher à la structure fonctionnelle de l'application, en ajoutant des fonctionnalités, les spécifications du logiciel n'ont pas été modifiées. Celles présentées ont donc été rédigées par le binôme ayant précédemment travaillé sur ce projet. Ces spécifications sont issues du document : Hanyuan Peng et Stéphane Deluce *Création de fond de planning*, Projet de programmation et génie logiciel, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2017-2018.

1. **Lire un fichier Excel**
(a) Rôle : Primaire
(b) Spécification :
- Entrée : Fichier Excel
- Sortie : Données lues
- Pré-condition : Fichier Excel au format valide
- Post-condition :
• Cette fonction a pour but récupérer les données utiles à la génération d'un fond de planning, contenue dans un fichier Excel.

2. **Parser les données récupérés dans les fichiers Excel**
(a) Rôle : Primaire
(b) Spécification :
- Entrée : Données brutes
- Sortie : Données structurées
- Pré-condition : Les données brutes doivent être valides

- Post-condition :
• Cette fonction a pour but structurer les données bruts.

3. **Stocker les données**
(a) Rôle : Primaire
(b) Spécification :
- Entrée : Données structurées
- Sortie : Objets construis avec les données
- Pré-condition : Les données brutes doivent être valides
- Post-condition :
• Cette fonction sera utile pour stocker les données structurées dans des objets

4. **Générer un objet Planning**
(a) Rôle : Primaire
(b) Spécification :
- Entrée : Objets construis avec les données
- Sortie : Objets planning
- Pré-condition : Tous les objets doivent être présents
- Post-condition :
• Cette fonction a pour un objet fond de planning complet.

5. **Créer un fichier Excel contenant le fond de planning**
(a) Rôle : Primaire
(b) Spécification :
- Entrée : Objets planning
- Sortie : Fichier Excel
- Pré-condition : L'objet doit être valide
- Post-condition :
• Cette fonction a pour but transformer un objet planning en un fichier Excel.

Nous avons dans ces fonctionnalités principalement agit dans les fonctions **lire un fichier Excel** et **parser des données récupérés dans les fichiers Excel** qui sont les principaux points de levés d'erreurs clés pour l'utilisateur. Cependant, la nouvelle gestion d'erreurs a demandé une restructuration générale des levées d'erreurs - Qui seront expliquées plus loin dans ce rapport.

2 Modélisation

Modélisation existante

Le programme a été développé selon le modèle MVC, ce qui nous permet de cibler plus simplement les classes qui soulèvent des erreurs. Il s'agit en effet, des classes du package **controller** qui s'occupent de la lecture des fichiers Excel, du stockage des données, de leur analyse et de la génération des fonds de planning.

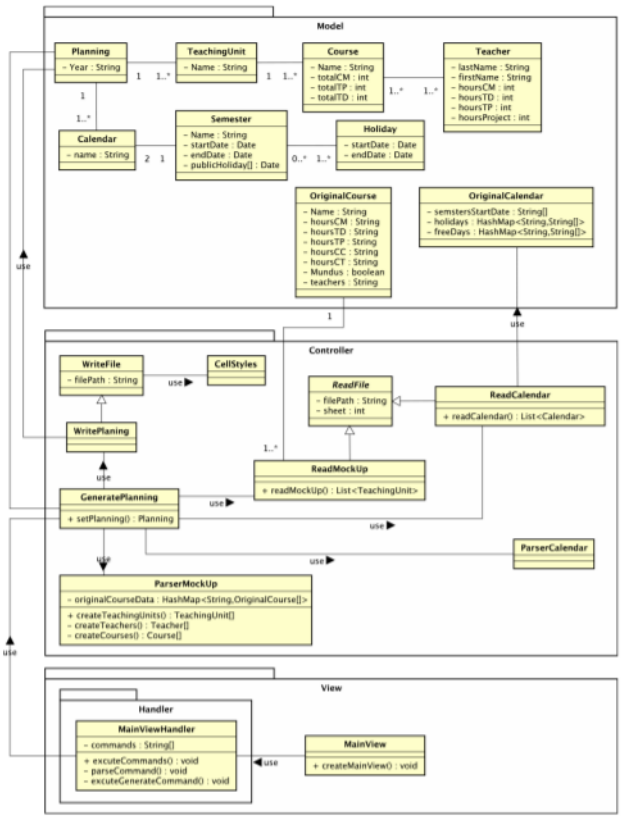

Figure 1 – *Diagramme de classe fournie*

Ajout à la modélisation

Au cours du projet, des modifications on été apportées au projet, notamment avec l'ajout de deux nouvelles classes liées à l'automate créé pour s'assurer de la bonne syntaxe de la chaîne à analyser. On retrouve donc le nouveau diagramme de classe (cf. figure 2) comportant les deux classes ajoutés au modèle.

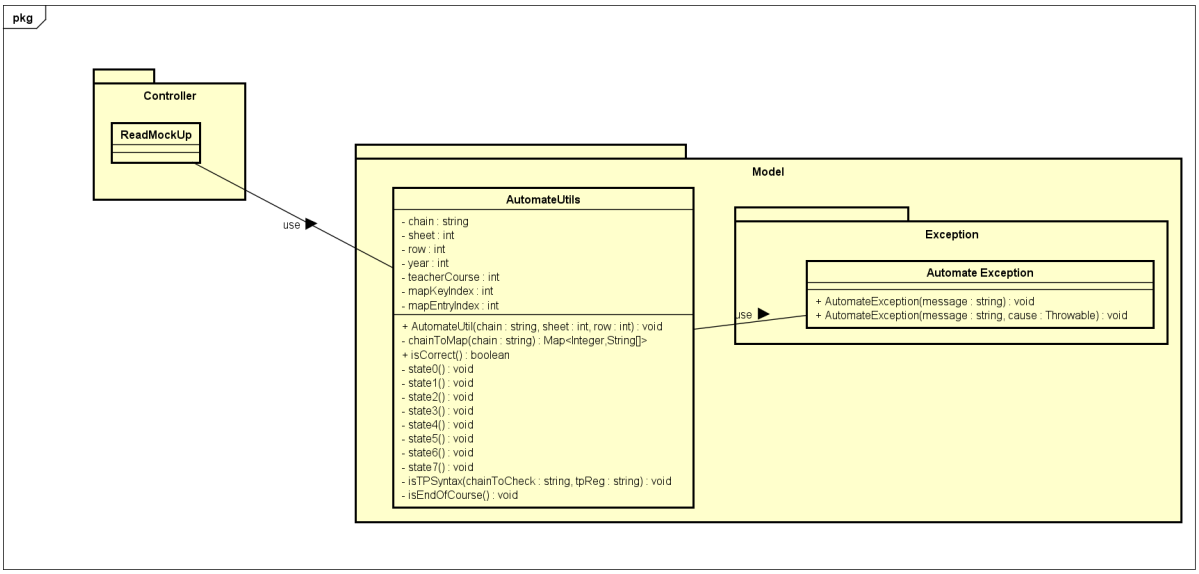

Figure 2 – *Ajout au diagramme de classe*

Définition des nouvelles classes

1. ### Classe AutomateUtil

   Cette classe sert à analyser la chaîne contenant les informations relative au cours.
   Attributs :
   - chain : la chaîne à analyser
   - sheet : le numéro de la feuille d'où provient la chaîne
   - row : le numéro de la ligne de la chaîne dans la feuille
   - year : le numéro de l'année correspondant à la feuille
   - teacherCourse : Map correspondant pour chaque professeur de la chaîne aux arguments mentionnés
   - mapKeyIndex :numéro indiquant pour quel professeur on analyse les informations
   - mapEntryIndex : indique l'argument en cours d'analyse pour le professeur courant

2. ### Classe AutomateException

   Cette classe permet la création d'objets permettant de savoir que l'erreur provient de l'automate. Cette classe contient uniquement deux constructeurs, l'un permettant d'ajouter d'autres erreurs non propre à l'automate.

4 Planification

La planification est une étape essentielle lors de la réalisation d'un projet ; Elle est très importante et permet d'en assurer une structure stable dans l'optique d'optimiser le temps et les moyens employés.

1 Définition des tâches

La première des choses à faire est de définir les tâches. Pour se faire, il était important de faire une liste des différents éléments importants à réaliser, puis de différencier les éléments à modifier des éléments à ajouter. C'est en étudiant le rapport du projet de l'an dernier que nous avons initialement basé nos tâches, puis nous les avons précisé à travers une réunion avec notre tuteur académique, M. Lenté.

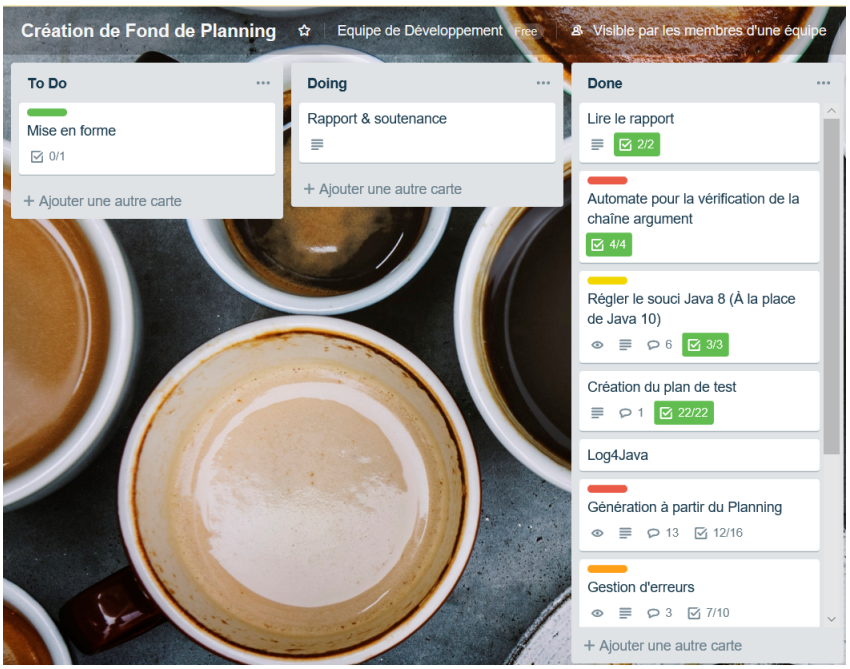

Figure 1 – *Tâches prévues sur l'outil en ligne Trello*

1. **Compatibilité de versions du langage**

   La toute première chose qui nous a posé un problème était liée à la version du langage Java utilisé. Lorsqu'il a été rendu l'année précédente, le projet fonctionnait sous la version 8 de Java.
   En Juillet 2017, Java 9 est paru puis, en Mars 2018, Java 10 a fait son apparition. L'arrivée de Java 10 a provoqué de grands changements dans les applications fonctionnant sous Java car le nouveau JDK a partiellement été revisité. En effet, un grand nombre de bibliothèques initialement incluses dans les anciennes versions de Java ont été retirées pour alléger le JDK et, ainsi, n'en faire qu'un noyau essentiel.
   Dans le cas actuel, quelques fonctions du projet initial n'étaient plus fonctionnelles avec la version 10 de Java (Version actuelle lors de la reprise du Projet) ; L'exécution du logiciel ne fonctionnant plus avec des versions supérieures à Java 8. Il nous semblait alors évident de **mettre à jour le logiciel** afin qu'il soit **fonctionnel avec les dernières versions de Java**.
   Avancées :
   — La tâche est terminée et effective.

2. **Gestion des erreurs**

   Lors de notre première réunion avec notre tuteur académique, M. Lenté, nous avons appris que le logiciel ne fonctionnait pas avec ses maquettes personnalisées. Jusqu'alors, les tests effectués sur le logiciel pour assurer son fonctionnement n'étaient exercés qu'à travers une seule maquette.
   M. Lenté nous a alors montré le comportement du logiciel lorsqu'il lui indiquait sa maquette, et ce dernier n'affichait aucun message d'erreur alors qu'il ne fonctionnait pas ; Seul un message "null" était affiché. Le problème pouvait venir d'un mauvais format de la maquette, tout comme il pouvait venir d'un problème logiciel ; Dans tous les cas, aucune erreur n'était affichée.
   Afin de ne plus avoir de cas similaire, nous avons décidé de retravailler le programme afin qu'il puisse afficher des erreurs à l'utilisateur, lui permettant de réagir en fonction de ce qui ne fonctionne pas. Cette tâche est immédiatement devenu pour nous une **tâche primordiale** pour le bon fonctionnement du logiciel.
   Avancées :
   — La tâche est majoritairement terminée.
   — Certaines fonctionnalités demandent encore de l'attention.

3. **Tests unitaires et fonctionnels**

   La version initiale du projet présentait quelques tests unitaires et fonctionnels, permettant alors de vérifier le fonctionnement du logiciel sur certains points. Cependant, nous avons remarqué que de nombreux cas n'ont pas été pris en compte, telle que la vérification d'une chaîne de caractères correcte (pour définir les heures et professeurs d'un cours, par exemple).
   Il nous semblait alors important de **mettre à jour** ces tests sur **les points qu'ils ne couvraient pas déjà**, mais également sur ceux relatifs aux **ajouts que nous avons effectués**.
   Avancées :
   — La tâche est terminée ; Tous les tests n'ont cependant pas été effectués avec JUnit.

4. **Création d'un automate**

   La maquette a donner en argument au logiciel demande, pour chaque cours de chaque semestre de chaque année, une chaîne de caractères vérifiant une structure très précise indiquant le, ou les, professeur concerné, ainsi que les heures de CM, TD et TP allouées. Certaines options spéciales sont également possibles.
   Cependant, le programme initial ne prévoit aucune vérification de ces chaînes de caractères. Peu importe ce qui est rentré dans les cases de la maquette, il s'exécute et s'adapte

en fonction de ce qu'il comprend ; Généralement, il remplit les informations de manière incorrecte, ce qui ne permet pas d'utiliser le fond de planning.
Notre tuteur académique, M. Lenté, nous a conseillé de **concevoir un automate** afin de **prévoir les différents cas d'utilisation possibles** et de **refuser ceux impossibles**. C'est ce que nous avons fait.
Avancées :
— La tâche est terminée.

5. **Journalisation de l'utilisation**

   En écho à la gestion d'erreurs du logiciel, il existait initialement deux types de comportements lors de la rencontre d'une erreur : L'affichage d'un message "Null" et l'affichage entier de toute l'erreur, avec la Stacktrace.
   Afin d'éviter à un utilisateur d'être confronté à une dizaine de lignes relatant les raisons de l'erreur de manière purement technique, nous avons pris la décision d'utiliser une **bibliothèque Java permettant d'ajouter une journalisation**. Les erreurs d'utilisation seront affichées devant l'utilisateur et les erreurs dues au logiciel seront vaguement annoncées devant l'utilisateur, mais précisées dans des fichiers d'historiques. De cette manière, l'utilisateur comprendra si l'erreur vient de lui ou si elle vient du logiciel. Dans le second cas, la journalisation permettra aux développeurs de régler le souci de manière plus aisée.
   Avancées :
   — La tâche est terminée.

6. **Mise en forme**

   Lors d'un succès de génération, le tableur généré présente encore des erreurs mais également des manques que notre tuteur académique, M. Lenté, souhaiterait voir comblés.
   Une refonte de quelques cases et la correction d'erreurs de génération a donc été à prévoir.
   Avancées :
   — La tâche n'a pas été commencée ; Elle est à prévoir à l'avenir.

2 Répartition des tâches

La seconde chose à faire lors de la planification du projet était de répartir les tâches entre les membres du binôme. Ainsi, nous avions décidés :

1. **Alexia Neufond**
   (a) Tests unitaires et fonctionnels.
   (b) Création d'un automate.
2. **Théo Bertet**
   (a) Gestion des erreurs.
   (b) Journalisation de l'utilisation.
3. **Binôme entier**
   (a) Compatibilité de versions du langage.

3 Diagramme de Gantt

Avant de débuter pleinement les différentes tâches à réaliser, nous avions décidé de planifier notre travail afin de prévoir le temps à allouer à chaque étape du projet. Lors de notre premier jet, toutes les tâches n'avaient pas encore été formulées, c'est pourquoi nous avons pris la

décision de refaire un diagramme de Gantt vers la fin du projet, afin de montrer de manière plus représentation la répartition temporelle de notre travail.

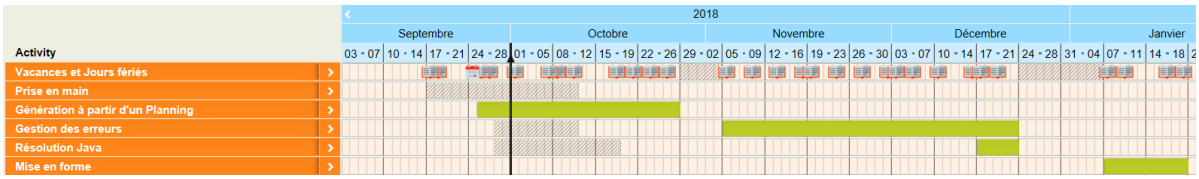

Figure 2 – *Planning initial du projet effectué sur Toms Planner*

Nous avons donc créés un nouveau planning davantage similaire aux vraies périodes que nous avons alloué aux tâches du projet, jusqu'à la fin de la pause pédagogique de fin d'année.

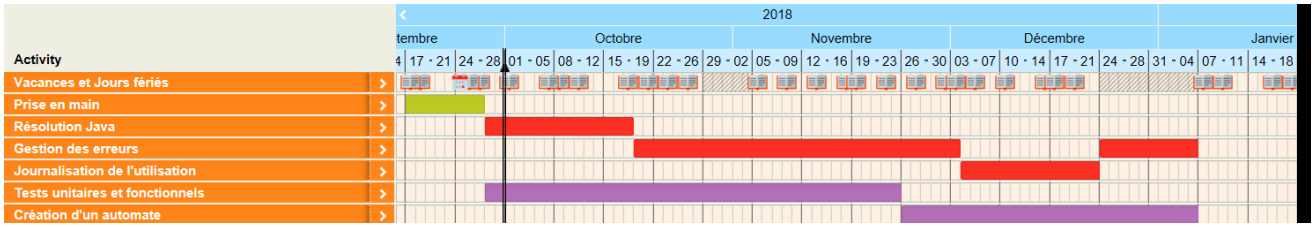

Figure 3 – *Planning adapté du projet effectué sur Toms Planner*

On note les couleurs représentant :

1. **Vert :** Alexia Neufond et Théo Bertet
2. **Rouge :** Théo Bertet
3. **Violet :** Alexia Neufond

5 Tests applicatifs

1 Définition des tests

Le programme comportant quelques erreurs lors de l'utilisation des fichiers fournis par notre encadrant, il nous a semblé nécessaire de mettre en place des tests fonctionnels. Ceux-ci on pour but de lister les erreurs non gérées par le programme et de trouver une possible cause commune à certaines d'entre elle.

Il a été décidé de faire 26 tests, dont un représentant le fonctionnement nominal, les autres représentant un fonctionnement anormal. Ceux-ci ont été créés après avoir chercher une liste la plus exhaustive possible de cas pouvant provoquer une erreur devant être gérée par l'application. On prend alors en compte les oublis de virgules dans la chaîne analysée, que des dates ont été changées dans le fichier Calendrier mais que la date de début est après la date de fin, etc.

Par la suite, il a été nécessaire de voir les cas pouvant être regroupé dans un test. Il s'agit alors de savoir ce qu'il se passe en cas de l'absence d'une feuille dans la maquette ou d'un indexage différents de celui demandé.

Les tests ont été décrits au travers de fiche de tests, permettant de savoir comment sera exécuté le test, avec quels arguments de lancement et le résultat attendu (cf. Section 1 (Annexe A) et Section 2 (Annexe A))

2 Résultats des tests

Tests liés au lancement du programme

Le programme étant lancé via l'invite de commande, ou un fichier de commande, il était nécessaire de s'assurer que l'application sache s'il lui manque un argument ou si deux ont été inversés. Ce qui après plusieurs tests, visant à inverser des arguments où en oublier, fut confirmé le programme indiquant une erreur et invitant à se référer à l'aide de l'application.

Tests liés à la chaîne à analyser

La syntaxe de cette chaîne analysée par le programme est très stricte, il fallait donc être sûr que l'application vérifie sa syntaxe pour veiller au bon fonctionnement et une génération de fichiers cohérents avec les données.

Après exécution d'une série de tests regroupant les différentes erreurs pouvant être commise sur la chaîne (oublie de virgule, point-virgule, ou mention "+Mundus" pour des années supérieur à la 3A), il a été noté qu'aucune vérification n'était effectuée sur cette chaîne.

Ainsi, les différentes erreurs pouvaient mener à des incohérences lors de la génération de fichiers tel qu'un nom de professeur comprenant les horaires de CM, par exemple. Il semblait ainsi important de mettre en place cette vérification (cf. Chapitre 7) cette chaîne étant l'élément principal pour la génération du fond de planning. Elle ne peut donc être erronée.

Tests liés à la lecture des fichiers

Lors de la lecture des fichiers, le numéro de la feuille est importante pour l'application. En effet, nous avons pu constater que le programmer se référait au numéro de la feuille pour distinguer les différentes informations à aller chercher (exemple : feuille d'indice 1 et 2 pour les Unités d'Enseignement de 3A).

Ainsi, il nous fallait nous assurer que le programme ne puisse aller chercher une feuille erronée. Cependant, les tests ont montrés que l'application en l'état nécessite que toutes les feuilles soient dans le bon ordre. Dans le cas contraire, il peut générer des données incohérentes ou causer une erreur car il ne trouve pas la feuille du fichier Excel qu'il souhaite lire.

De plus, lors de la lecture du calendrier, si deux dates ne sont pas cohérentes (date de début après la date de fin), le fond de planning sera généré sans prendre en compte le semestre concerné. Aucune erreur ne sera donc soulevée dans ce cas.

Résultat global

Ainsi, dans son ensemble, l'application comporte plusieurs failles dans son exécution par bon nombre de cas qui n'ont pas pu être pris en compte par le binôme nous ayant précédé sur ce projet. Il y a donc plusieurs erreurs à résoudre afin que le programme soit totalement fonctionnel avec en priorité l'analyse de la chaîne fournissant les informations.

Une fois ces erreurs résolus, les tests pourront donc être validés. En effet, 11 des 26 tests réalisés non pu être validés, en prenant en compte que 4 d'entre eux peuvent être résolue en analysant la chaîne fournissant les informations.

6 Gestion des erreurs

La gestion des erreurs est une partie très importante du projet. Son but principal est de permettre à l'utilisateur de comprendre pourquoi son logiciel ne fonctionne pas. On différencie deux types d'erreurs :

— Les **erreurs d'utilisation** : Ce sont des erreurs commises par l'utilisateur. Elles peuvent concerner les fichiers utilisés pour la génération, ou la chaîne d'information envoyée à l'exécutable.
— Les **erreurs logicielles** : Ces erreurs sont propres à la conception du logiciel. Dans ce cas, il est nécessaire de le préciser à l'utilisateur. Cependant, il lui est inutile d'observer la StackTrace ; L'utilisateur n'étant ici pas le développeur.

Dans cette partie du rapport, nous traiterons de trois points importants :

1. Les **changements de la version de Java** ; Ce qui a changé et pourquoi.
2. La **résolution des erreurs signalées** ; Faire en sorte que la console de l'utilisateur ne soit pas poluée d'informations inutiles.
3. La **Journalisation** ; Journaliser l'information afin de pouvoir aider les interventions futures en cas de problèmes.

1 Changement de la version de Java

L'une des principales modifications à faire concernait le changement de la version de Java. Comme évoqué plus tôt dans le rapport, la version initiale du projet était Java 8. Cependant, pour des raisons de compatibilités actuelle et future, nous avons été obligés de mettre à jour Java à sa version 1.

Pour se faire, il fallait initialement avoir Java 10 sur la machine de production. Les versions de Java sont disponibles sur le site officiel de Java : Java.com. Ensuite, il fallait mettre à jour les dépendances Maven.

— Maven est un gestionnaire de dépendance permettant de rajouter des classes étrangères au projet de manière dynamique et facile. Il permet également de définir la version de Java utilisée.

Également évoqué plus tôt, la version 10 de Java retire un grand nombre de fonctionnalités présentes dans sa version 8. Elles sont cependant toujours accessibles, mais impliquant des importations. On utilise une nouvelle fois Maven, pour les dépendances suivantes :

— Jacorb
— Orb
— Glassfish-corba-orb
— Psm
— Piccolo
— Poi
— Poi-ooxml
— Poi-ooxml-schemas

Pour rajouter une dépendance, il faut respecter le schéma suivant :

<...>

<dependencies>

<dependency>

<groupId> ... </groupId>

<artifactId> ... </artifactId>

<version> ... </version>

</dependency>

</dependencies>

<...>

Les blocs "dependency" sont disponibles sur internet ; On ne peut pas les inventer. Lorsqu'ils sont corrects, ils sont comme des URL ; Maven se connecte à internet pour rajouter au projet les bibliothèques qui lui manquent.

Une fois chaque dépendance mise à jour ou rajoutée, il ne faut pas oublier de compiler le logiciel avec la version 10 de Java. (Selon l'IDE, la configuration se fait différement.) Enfin, de la même manière, l'exécution du logiciel doit se faire **depuis la version 10 de Java** ou **supérieure**.

2 Résolution des erreurs signalées

La deuxième partie de la gestion des erreurs concernait l'affichage des erreurs dans la console, en fonction des types d'erreurs. Initialement, le projet ne possédait qu'un système très basique de gestion d'erreurs.

En programmation orientée objet, la gestion d'erreur doit être intelligente et très précise. Certaines erreurs ont pour nécessiter d'arrêter le programme tout entier ; On imagine mal un programme lire un fichier s'il n'a pas réussi à l'ouvrir. D'autres erreurs peuvent être ignorées, mais constituent un grand risque pour la pérennité du programme. Chaque erreur ignorée doit faire preuve d'une extrême rigueur.

On parlera ici également de niveau d'erreur ; Certaines sont à haut niveau, et d'autres à bas niveau. La hauteur d'une erreur se mesure par la profondeur à laquelle elle peut survenir. Une erreur de haut niveau surviendra sur les couches principales de l'application ; Celles écrites par l'équipe de développement et qui appellent toutes les autres. Plus une erreur est basse, plus elle est profonde dans les couches de fonctions ; Elle peut être un calcul, une vérification, une ouverture de fichier.

Dans le cas initial, toutes les erreurs - hautes comme basses - levaient des erreurs à leur niveau. Pour qu'un logiciel cesse de fonctionner à la levée d'une erreur, il est important que rien ne soit exécutée après la levée d'erreur.

— NB : En programmation, on parle de *levée d'erreur* lorsqu'on empêche l'exécution d'un certain nombre de lignes de code lorsqu'une erreur survient.

Dans le cas d'une erreur générale, ladite erreur est sensée être remontée au plus haut niveau du code ; C'est à dire à la fonction principale. Pour les erreurs non fatales, elles peuvent être remontées à tout niveau - selon la volonté du développeur.

Dans le cas actuel, la première erreur à avoir été réglée est celle concernant la lecture d'une maquette :

- Lorsque le logiciel lit une maquette, il se soucie de l'existence du document. S'il n'existe pas, il lève une erreur mais continue son traitement. Comme le document n'existe pas, il lève de nouvelles erreurs qui cessent l'analyse, puis le traitement. Au final, ce sont trois erreurs relevées alors qu'une seule suffirait.
- Si la maquette existe, il va chercher à lire les pages qui concernent la demande de l'utilisateur. Si une des pages est manquante, le logiciel va lever l'erreur mais continuer son traitement pour final s'arrêter après trois erreurs relevées. (Démarche identique que précédemment.)
- Si la page existe, il va analyser chaque cellule de la page du tableur. Cependant, si une cellule censée être remplie est vide, il peut lever une erreur - Il se contentera parfois de ne rien lire. Ici, et en fonction de l'erreur, le programme s'arrêtera ou continuera son traitement avant de lever d'autres erreurs et de s'arrêter.

En d'autres termes, la gestion d'erreur n'avait pas été très travaillées et provoquait une grande cause des incompréhensions d'erreurs d'utilisation. Pour régler le problème, nous avons pris la décision **d'uniformiser la gestion d'erreur** en faisant en sorte que chaque erreur soit **relevée au plus haut niveau** afin **d'arrêter le programme en cas d'erreur**. Certains cas nécessiterait peut-être de pouvoir être ignorés, mais nous avons préféré ne pas passer trop de temps là-dessus pour nous concentrer sur autre chose.

Ainsi, chaque fonction employant une autre fonction nécessitant une levée d'erreur nécessite désormais une levée d'erreur au niveau supérieur. Prenons un exemple :

- La fonction **MainViewHandler.parseCommand()** : Elle est très utile à la compréhension de la volonté de l'utilisateur. Elle permet de séparer les différents éléments de la chaîne d'arguments donnée au logiciel. (Années à traiter, nom du fichier maquette, années.)
  Elle utilise des fonctions nécessitant la gestion de quatre types d'erreurs différentes : *IllegalArgumentException* ; *NullPointerException* ; *IOException* ; *ParseException*.
  De ce fait, au lieu d'entourer ces fonctions du traditionnel *try ... catch ...* permettant d'effectuer les instructions présentes dans le *catch* si une erreur est levée dans le *try*, nous demanderons aux fonctions appelant la fonction **parseCommand()** de se charger de cette gestion d'erreurs ; Qu'ils utilisent un *try ... catch ...* ou la même méthode. L'intérêt d'une telle manipulation est de faire remonter l'erreur aux niveaux supérieurs.
  Pour se faire, on précise lors de la déclaration de la fonction la présence de possibles erreurs de la manière suivante :
  - private void parseCommand() throws IllegalArgumentException, NullPointerException, IOException, ParseException ...

  De cette manière, chaque fonction utilisant **parseCommand()** sera obligée de prendre en compte les quatre types d'erreurs impliqués.
- La fonction **MainViewHandler.executeCommand()** appelle la fonction **MainViewHandler.parseCommand()** ; Elle nécessite donc désormais de prévoir la gestion d'erreur. De la même manière - et parce que nous ne sommes pas au niveau le plus haut du code -, nous rajouterons la mention *throws ...*.
- La fonction **MainView.createMainView()** appelle la fonction **MainViewHandler.executeCommand()**. De la même manière, il faut rajouter la mention *throws ...*.
- La fonction **App.main()** appelle la fonction **MainView.createMainView()**, ce qui implique à son tour de prendre en compte la présence d'erreurs potentielles. Cependant, nous sommes ici au niveau le plus haut du code ; Il est donc nécessaire cette fois-ci d'utili-

ser le conventionnel *try ... catch ...*. On entoure tout le code ne devant pas s'effectuer en cas d'erreur entre le *try* et le *catch* puis, au-delà du *catch*, on affiche le message de l'erreur levée. De cette manière, l'utilisateur ne verra pas trois erreurs relatant de la même initiale, mais bien une seule.

De cette manière, nous ne détaillerons pas toutes les levées d'erreurs modifiées. Cependant, chaque fonction a été analysée afin que seule la fonction principale - de la couche la plus haute - ne lève une erreur pour arrêter le programme. Nous n'avons pas souhaités nous attarder sur les erreurs ne nécessitant pas d'arrêt global du logiciel.

Lorsque nous recherchions les erreurs, nous sommes tombés sur deux types d'erreurs pouvant être ignorées ; Il s'agit de **ReadFile.readNumericCell()** et de **ReadFile.readCell()**. Dans la première, il s'agit de la présence de texte dans une cellule d'informations inutile au traitement de l'information ; Nous avons pris la décision d'ignorer cette erreur mais d'afficher dans la journalisation la présence de l'erreur. La seconde fonction concerne une erreur de format de cellule.

— NB : Nous avons passé beaucoup de temps sur l'origine des erreurs de format de cellule, sans succès. Après rendez-vous avec notre tuteur académique, M. Lenté, nous en sommes venus aux hypothèses que les erreurs étaient liées à une erreur de conversions des fichiers Excel MAC à Excel Windows, ou de fichiers OpenOffice/LibreOffice à Excel.
La bibliothèque utilisée présentait alors des erreurs incompréhensibles (Le logiciel ne réagissait pas toujours de ma même manière face à différentes cellules pourtant vides).

Pour ce cas, nous avons décidé également d'ignorer l'erreur - qui ne provoque pas d'autres erreurs pour le traitement de l'information - tout en affichant dans la journalisation sa présence. Attention toutefois à ce que cela ne pose pas de problèmes à l'avenir.

3 Journalisation

Comme évoqué à plusieurs reprises plus tôt dans le rapport, nous avons décidé de mettre en place un système de journalisation permettant à l'utilisateur et au développeur de se renseigner sur le fonctionnement du logiciel.

Dans le cas d'une levée d'erreur, il est inutile que l'utilisateur soit confronté à une StackTrace.

— NB : Une StackTrace est la trace indiquant chaque les lignes de code concernées par l'erreur ; Dans le cas actuel, toutes les StackTrace partent du fichier principal pour remonter jusqu'à la ligne écrite posant un problème.

Ainsi, un simple message ressemblant à "*Le fichier Maquette est introuvable. Vérifiez qu'il soit bien présent.*" ou à "*La chaîne d'informations de la cellule A30 de la page 2 est incorrecte.*" sera bien plus intéressant. Les informations plus complexes - et complètes - seront quant à elles présentes dans des fichiers dits *de logs*. Ils respectent le format **INFO/WARN/DANGER HH :mm :ss.SSS NomDeLaFonction : MessageDErreur**.

Pour mettre en place un tel système de journalisation, nous avons décidé de mettre en place la librairie *Log4J* permettant de mettre en place aisément un système de journalisation.

La première chose est de l'inclure depuis Maven. Une fois fait, il faut rajouter un dossier "*resources*" qui aura pour but de contenir le fichier de configuration de l'écriture des fichiers de Logs. Il doit se nommer **logback.xml** et contenir les lignes suivantes :

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <Pattern>%d{HH:mm:ss.SSS} [%-5level] [%thread] [%logger{0}] %msg%n</Pattern>
        </encoder>
    </appender>

    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>log/LOG_Planning_%d{yyyyMMdd}_%d{HHmmss}.log</fileNamePattern>

            <!-- <maxHistory>7</maxHistory> : To keep 7 days' worth of history -->
        </rollingPolicy>

        <encoder>
            <Pattern>[ %-5level] %d{HH:mm:ss.SSS} [%thread:%logger{0}] : %msg%n</Pattern>
        </encoder>
    </appender>

    <root level="debug">
        <appender-ref ref="FILE" />
        <!--<appender-ref ref="STDOUT" />-->
    </root>
</configuration>
```

Figure 1 – *Fichier de configuration de Log4J - Logback.xml*

On remarque deux types de balises : Les balises **appender** et les balises **root**. Les premières permettent de définir un paramètre d'affichage de la journalisation pour un type précis, tandis que les secondes permettent de définir quels types sont à mettre en place.

Nous avons grossièrement survolé la bibliothèque, ne nécessitant pas d'entrer dans trop de détails. Pour *appender FILE*, les deux lignes importantes à retenir concernent le nom du fichier de log - *fileNamePattern* - et le format des lignes de log - *pattern*. Le format du document est strict et il faut le respecter.

Actuellement, les documents de logs seront stockés dans le fichier **log** qui sera automatiquement généré dans le fichier racine.

7 Automate de vérification

1 Pourquoi un automate ?

Après réalisation des tests fonctionnels, il nous est apparu que la plupart de ceux qui trouvait des erreurs ou des fichiers incohérents étaient dû à une non vérification de la chaîne contenant ces informations Section 2 (Chapitre 5).

Il nous a donc paru important de résoudre ce problème en priorité afin de s'assurer d'avoir au final un fichier cohérent. Ainsi, nous avons donc créé un automate pour l'application, en suivant le processus suivant :

1. Analyser les différentes possibilités de syntaxes
2. Trouver un pattern
3. Modéliser l'automate correspondant au pattern
4. Factoriser l'automate selon le langage de programmation
5. Développer l'automate
6. Intégrer l'automate à l'application.

2 Modélisation

Chaîne à analyser

La chaîne à analyser possède un pattern stricte qu'il faut totalement respecter pour que le programme fonctionne totalement. En effet, bien que le programme ne soulevait avant mise en place de l'automate aucune erreur si l'on oubliait une virgule le document final perdait quant à lui en cohérence. Il était donc nécessaire de prendre en compte tous les cas possibles d'écriture de la chaîne afin de la généraliser au maximum.

La chaîne doit donc posséder la structure suivante, d'après la documentation qui nous a été fournie :

```
<nomEnseignant>, <x>hCM, <y>hTD x<nbGroupes>gr (+ Mundus)>,<z>hTP x<nbGroupes> (+ Mundus)
```

Il faut de plus prendre en compte que cette structure peut-être répétée après ajout d'un ;.

Cette chaîne a donc été analysée et les points suivants ont été notés et permettent une modélisation plus aisée de l'automate :

1. On peut renseigner uniquement le nom d'un professeur.
2. On peut renseigner uniquement le nom du professeur suivit par la mention **,x<nbGroupes>gr**.
3. La mention **Mundus** ne concerne que les 3ème années.
4. Les CM, TD et TP sont toujours ordonnés dans cet ordre, même si un des trois est manquant.

Première modélisation

La première modélisation de l'automate ne prend pas en compte le langage de programmation utilisé. Il s'agissait ici de mettre en forme l'automate selon une analyse caractère par caractère (à l'exception de certains mots-clés). On obtient ainsi l'automate de la figure1 qui possède 24 états.

En rouge sont représentés les éléments qui doivent être présents uniquement dans le cas d'un cours de troisième année et en noir les éléments communs à toutes les années.

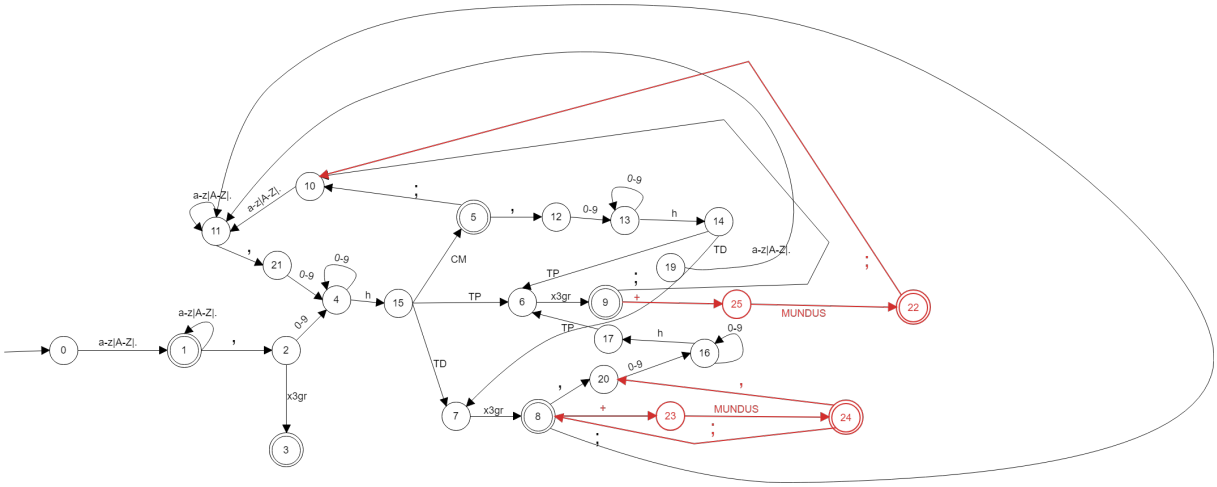

Figure 1 – *Première version de l'automate*

3 Programmation

Adaptation au langage de programmation

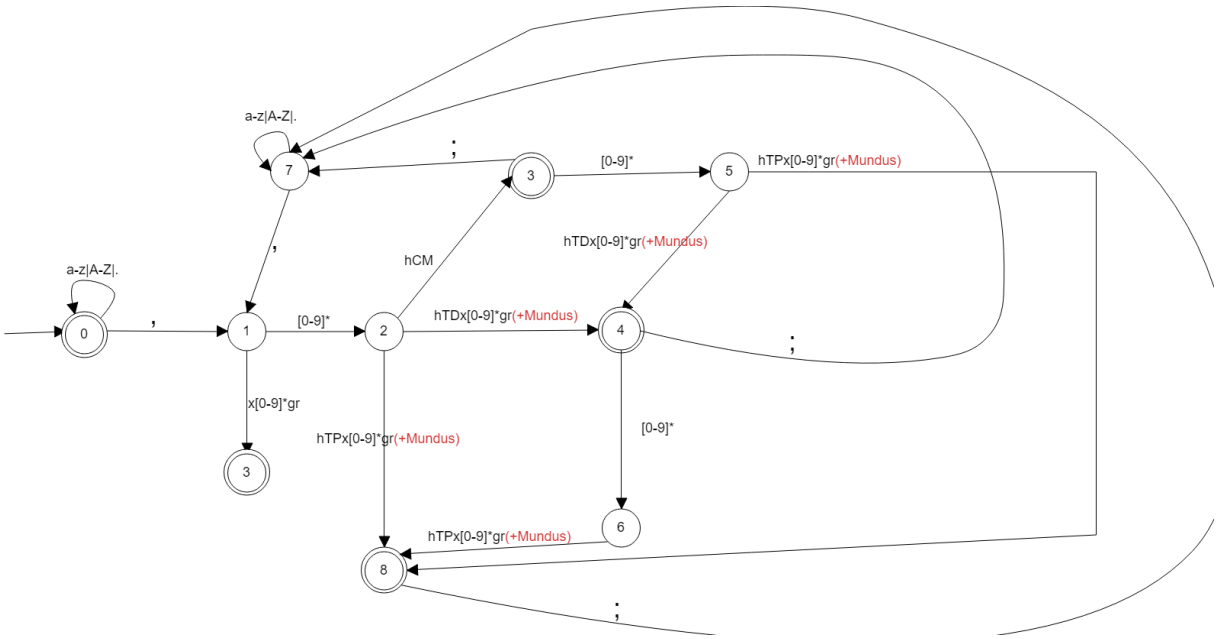

Figure 2 – *Automate factorisé pour le programme*

Le programme s'effectuant en Java, il est possible par l'utilisation du **Regex** de coder l'automate plus simplement que montré ci-dessus. Ainsi, il a été possible de factoriser l'automate figure 1 en une version plus simple ne contenant alors plus que 8 états, comme le montre le schéma figure 2.

Développement

Il fut ainsi plus simple de modéliser ce nouvel automate. Chaque état représentant une fonction dans le programme, on a ainsi moins de fonction à développer et celles-ci sont simplifiées par l'utilisation du regex. Le risque d'erreur d'implémentation était alors réduit.

Nous avons de plus conclu qu'il serait plus judicieux de développer cette classe en dehors du programme principale. En effet, il nous fallait étudier l'emplacement dans le code existant le plus précis pour appeler la fonction de vérification en ayant tous les paramètres nécessaires à son fonctionnement.

En l'intégrant après s'être assuré de son bon fonctionnement, on pouvait de plus assurer la stabilité du programme préexistant sans avoir besoin de chercher une version antérieur sur notre outil de versionning.

Test Unitaire

Cette nouvelle classe nécessitait de tester son bon fonctionnement avant d'être intégré au programme principal. Il s'agit en effet d'une classe qui si elle détecte de fausse erreur peut bloquer le programme. Plusieurs tests ont donc été créés correspondant aux différentes fonctions publiques présentes dans le programme.

Intégration

Après validation de ces différents tests en prenant des cas où la chaîne était correcte et d'autre où elle était incorrecte nous avons ainsi pu intégrer la classe au programme principale et lancer les tests unitaires préétablis par le binôme précédent afin de valider que cette classe ne compromettait pas le fonctionnement de l'application.

L'automate est ainsi utilisé lors de la lecture du fichier de la maquette dans la classe **ReadMockUp**. En l'appelant, lors de la lecture des Unités d'Enseignement, on s'assure alors que le programme n'ira pas plus loin que la lecture en cas d'erreur dans la chaîne à analyser.

Conclusion

Il est compliqué de concevoir un logiciel ; Il est important de bien cerner les objectifs en fonction des attentes du client, mais il est également important de faire attention à produire un rendu de qualité. Reprendre la conception d'un logiciel est une mission toute aussi compliquée ; La difficulté dépend des travaux précédents.

Beaucoup de choses influent sur la facilité de reprise d'un document ; Documentation, technologies mises en place, méthodologie, approches adoptés... C'est pourquoi le projet que relate ce rapport a été très intéressant pour nous.

Dans un premier temps, comprendre l'approche d'un projet par d'autres personnes a été une réelle expérience d'adaptation. Il nous a fallu prendre en main et nous approprier le code déjà existant, afin de comprendre comme le logiciel fonctionnait et comment les erreurs pouvaient survenir. Également, afin de pouvoir apporter de nouveaux ajouts à l'application, il était nécessaire de bien comprendre les différentes étapes de fonctionnement du logiciel.

Dans un second temps, nous avons pu avoir un regard extérieur sur le travail d'un autre groupe (Bien que de l'an passé). Ainsi, nous avons pu cibler les problèmes majeurs et ainsi nous retracer les raisonnements qui ont été adoptés lors de la réalisation initiale. Que ce soit pour la gestion des erreurs ou la mise en place d'un format de chaîne très strict, nous avons appris des erreurs de nos camarades pour réaliser que tout doit être fait avec préparation et étude des risques.

Enfin, pouvoir modifier les parties importantes du logiciel et lui permettre de fonctionner selon les attentes de notre tuteur académique nous a permis de comprendre les enjeux et les attentes vis-à-vis du projet ; C'était un cas concret.

D'un point de vue plus technique, ce projet nous a permis de mieux réaliser l'importance de la gestion d'erreurs. Il nous également permis de mettre en place des systèmes que nous n'avions vu que théoriquement, comme un automate ou un système de journalisation, et ce parfois à travers des technologies d'aujourd'hui. Enfin, le lien avec la lecture et l'écriture de fichiers Excel nous a également permis de nous montrer à quoi un tel logiciel pouvait ressembler, mais également de mettre en lumière les difficultés parfois ambigües que cela peut provoquer.

D'un point de vue plus personnel, le projet nous a apporté une approche plus pratique et concrète de la gestion de projet. De la planification à la concrétisation, il nous a été très

intéressant d'être autonomes sur cette mission, tout en gardant un aspect encadré par le tuteur académique - Présent pour les réunions lorsque nous avions besoin d'avis ou lorsque nous avions des question.

Pour finir, ce projet a été très bénéfique pour nous sur de nombreux points. Nous sommes conscient ne pas avoir achevé le projet, mais nous pensons l'avoir rendu plus opérationnel et quelque peu utilisable. Il reste une grande part du travail à effectuer sur le plan génération ; Certaines vérifications à faire, mais également des modifications au niveau du format attendu.

Annexes

A Exemple de fiche de test

1 Exemple 1 — Cas nominal

Projet : Fond de planning

Feuille : Général - 1

Auteur : NEUFOND Alexia

Test référence : FdPlNom1G

Date : 25 octobre 2018

Identification du composant : Composant de génération du planning

Description du test :

Ici, on va tester le fichier Maquette, afin de savoir si les différents cas d'utilisation et de notation pour les professeurs sont fonctionnels, ainsi que le fichier Calendrier. Les deux devant fonctionner conjointement.

Arguments :

- Promotion : di4
- Année : 2018-2019
- Maquette : Maquette.xlsx
- Calendrier : Calendar.xlsx

Contenu du fichier Maquette (structure des lignes «argument») :

- <nomEnseignant>
- <nomEnseignant> ,<x>hCM, <y>hTD x<nbGroup>gr>,<z> hTP x<nbGroupes> (+ Mundus)
- <nomEnseignant> ,<x>hCM, <y>hTD x<nbGroup>gr (+ Mundus)>,<z> hTP x<nbGroupes> ; <nomEnseignant> <y'>hTD x<nbGroup>gr (+ Mundus)>,<z'> hTP x<nbGroupes>

Contenu du fichier Calendrier : Toutes les dates sont au format demandé sans problème d'ordre. Toutes les titres de colonnes et/ou lignes sont conformes aux instructions fournies.

Scénario :

1. Lancement de l'application avec les arguments fournis
2. Le programme analyse les fichiers passés en paramètre.

3. Le programme génère le fond de planning

Aucune erreur n'est générée. Le fond de planning est conforme et contient toutes les informations nécessaires à son utilisation.

Résultat obtenu :

Lundi 19 novembre : Aucune erreur n'est générée. Le fond de planning est conforme.

Test défectueux n° :

2 Exemple 2 — Cas anormal

On suppose un oubli de virgule après le nom du professeur dans la chaîne argument.

Projet : Fond de planning

Feuille : Général - 2

Auteur : NEUFOND Alexia

Test référence : FdPlAnorm1G

Date : 25 octobre 2018

Identification du composant : Composant de lecture de la maquette

Description du test :

Ici, on va tester le fichier Maquette principalement afin de vérifier le comportement du composant de lecture de la maquette en cas d'oubli d'une virgule après le nom du professeur.

Arguments :

- Promotion : di4
- Année : 2018-2019
- Maquette : Maquette.xlsx
- Calendrie : Calendar.xlsx

Contenu du fichier Maquette (structure des lignes «argument») : <nomEnseignant> ,<x>hCM, <y>hTD x<nbGroup>gr>,<z> hTP x<nbGroupes>

Contenu du fichier Calendrier : Le fichier sera conforme afin de veiller à ce qu'aucune erreur ne soit produite par ce fichier.

Scénario :

1. Lancement de l'application avec les arguments fournis
2. Le programme analyse les fichiers passés en paramètre.
3. Le programme génère le fond de planning

Résultat attendu :

Une erreur est générée à la lecture ou l'analyse de la ligne concernée. L'utilisateur est prévenu de l'erreur avec un retour de la ligne concernée par l'erreur.

Résultat obtenu :

Lundi 19 novembre : Le fichier a été généré et aucune erreur n'a été lancé. Les données ne sont pas conformes est les heures de CM n'ont pas correctement été renseigné. En effet, pour la chaîne argument : **J.C. Billaut 16hCM, 8hTDx3gr, 8hTP x3gr** le nom du professeur est renseigné comme étant "**J.C. Billaut 16hCM**" et ainsi les heures de CM ne sont pas renseignées dans le fichier final.

Lundi 20 décembre : Le programme génère une erreur indiquant qu'il manque une virgule ainsi que le numéro de la feuille et de la ligne produisant l'erreur.

Test défectueux n° : 1

Création de fond de planning

Reprise de projet

Résumé

Il s'agit de la reprise du projet d'une application de fond de planning. Le but est ici d'apporter un correctif à des bugs de l'application et mettre en place une gestion d'erreur plus claire et efficace.

Mots-clés

java, automate, fond de planing, gestion d'erreur, reprise de projet

Abstract

This project is the continuity of a previous application which generate a planning background. The aim is here to correct some bugs in the application and do a more efficient and clear error management.

Keywords

java, automate, planning background, error management, project recovery

Tuteur académique
Christophe Lenté

Étudiants
Théo Bertet (DI4)
Alexia Neufond (DI4)