Rapport Qualité de programmation

*Gestion d’un emploi du temps*

Adrien Couchot, Olivier Ettlin, Hicham Chouhad, Benjamin Bunner

1/9/2017

Sommaire

I Introduction

1. Présentation du projet

II Organisation du projet

1. Normalisation de noms
2. Gestion des versions
3. Organisation des fichiers
4. Répartition des taches
5. Documentation
6. Tests

III Problèmes rencontrés et solutions apportées

1. Organisation des classes
2. Utilisation de github

IV Conclusion

Introduction

Présentation du projet

Parmi les différents projets proposés, nous avons choisi de travailler sur la gestion d’un emploi du temps.

Une journée est composée de 4 créneaux fixes : 8h-10h, 10h-12h, 14h-16h et 16h-18h. Le but est de placer des cours dans ces créneaux tout en tenant compte des ressources nécessaires pour réaliser ces cours (enseignants, salle, professeur…). On doit pouvoir ajouter, déplacer ou supprimer des cours et des ressources.

On doit aussi pouvoir visualiser l’emploi du temps sur une semaine correspondant à une ou plusieurs ressources avec les détails du cours placé à un moment donné.

Une année est constituée de 52 semaines numérotées de 1 à 52 et chaque semaine aura 7 jours du lundi au dimanche (on ne tient pas compte des mois et des numéros de jours dans le mois). L’affichage est exportable dans plusieurs formats (txt, CSV…).

Le projet est réalisé en C++ avec le système de contrôle de version git. Il contient des commentaires réalisés avec Catch et inclus une documentation doxygen.

Organisation du projet

Normalisation de noms

Afin de mieux nous organiser, nous avons rapidement mis en place une normalisation des noms.

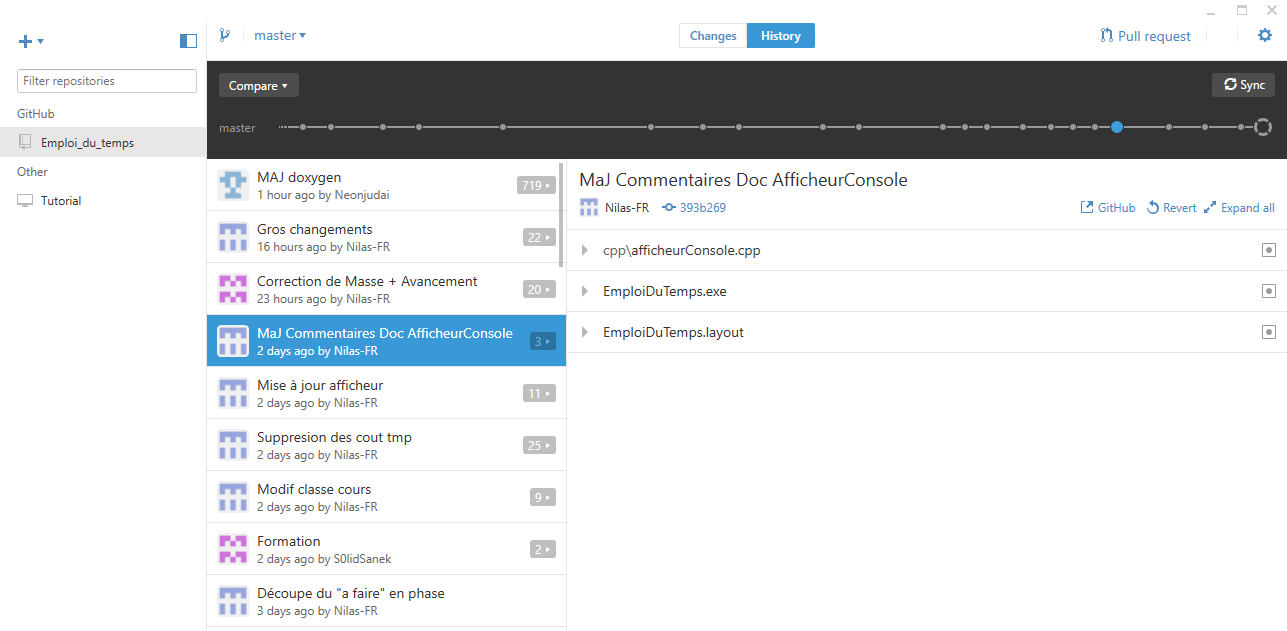
Les noms de fichiers sont écrits sous la forme « bosse de dromadaire » (« afficheurConsole.cpp »).

Les noms des méthodes, des variables et des objets sont aussi sous la forme « bosse de dromadaire » (« cours listeCours::coursNumero(int i) const »).

Les constantes sont écrites en majuscules avec des « underscore » séparant les noms composés (« const int NUM\_SEMAINE=2%52; »).

Gestion des versions

Afin de gérer la différente version du projet, nous avons utilisé le gestionnaire de version github. Nous avons tout d’abord créé un dossier partagé sur <https://github.com> (<https://github.com/Neonjudai/Emploi_du_temps>) que nous avons ensuite cloné sur nos machines respectifs en utilisant Github Desktop. Avant chaque modification du projet, une synchronisation était effectuée afin de récupérer la dernière version du projet. Après la modification, un « commit » accompagné d’un titre et d’une description des modification était lancé afin de permettre aux autres membres du projet d’avoir accès à la dernière version du projet. Le fait de revenir à une version précédente du projet est possible mais ne nous a pas été nécessaire.



Organisation des fichiers

Nous avons défini l’organisation des fichiers comme étant la suivante :

\_ les projets sont localisés dans le fichier Emploi\_du\_temps

\_ les fichiers .cpp autres que ceux liés aux tests sont localisés dans le fichier Emploi\_du\_temps\cpp

\_ les fichiers .h sont localisés dans le fichier Emploi\_du\_temps\h

\_ le fichier Emploi\_du\_temps\LancementTests.bat permet de lancer directement les tests liés au projet.

\_ le fichier Emploi\_du\_temps\Documentation.bat permet de visualiser la documentation liée au projet dans le navigateur par défaut.

\_ les fichiers .cpp liés aux tests sont localisés dans le fichier Emploi\_du\_temps\test

\_ les fichiers de documentation sont localisés dans le dossier Emploi\_du\_temps\html

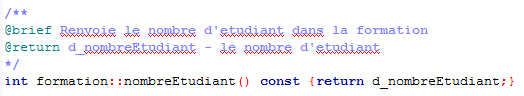
\_ le dossier Emploi\_du\_temps\latex contient des versions pdf, tex et md5 des diagrammes des différentes classes du projet.

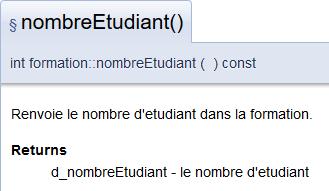
Répartition des Tâches

La répartition des tâches est détaillée dans « Répartition Taches.xlsx ».

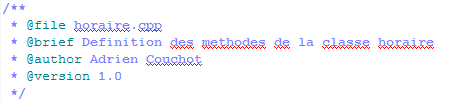
Documentation

Une documentation liée au projet est présente. Sa visualisation peut être effectuée via le fichier Emploi\_du\_temps\Documentation.bat. Il s’agit d’une documentation doxygen avec des commentaires de type C/javadoc :

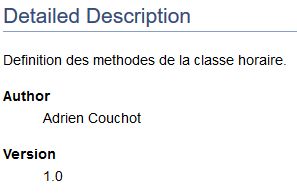




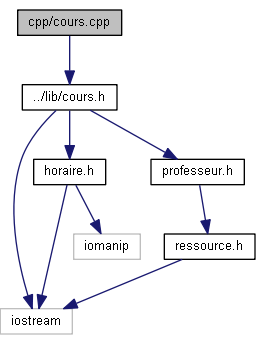
Chaque fichier contient un entête semblable au suivant :



Cet entête donne le résultat suivant dans la documentation :



Des diagrammes générés par le package GraphViz représentant les différentes classes sont présents dans la documentation ainsi que dans le dossier Emploi\_du\_temps\latex sous différents formats.



Tests

Afin de séparer les tests du projet principal, un projet TestEmploiDuTemps a été créé. Les tests unitaires sont entièrement gérés par ce projet. Pour générer ces tests, la bibliothèque catch a été utilisée. On peut lancer ces tests directement avec le fichier Emploi\_du\_temps\LancementTests.bat.

Les liés à ce projet ont la forme suivante :

TEST\_CASE ("titre\_du\_test","nom\_de\_la\_classe")

{

SECTION("Point\_à\_tester")

{

REQUIRE(…);

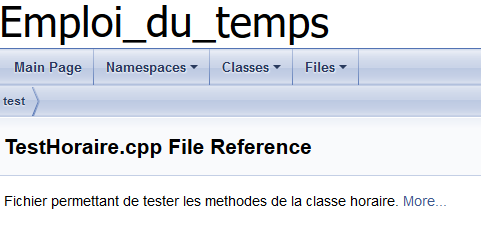
REQUIRE(…);

}

}

/\*\*< Commentaire\_de\_documentation \*/

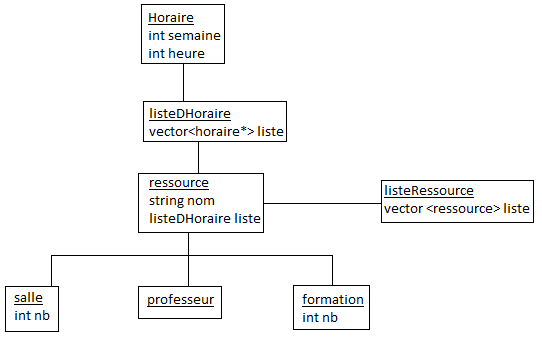
Les tests bénéficient eux aussi d’une documentation :



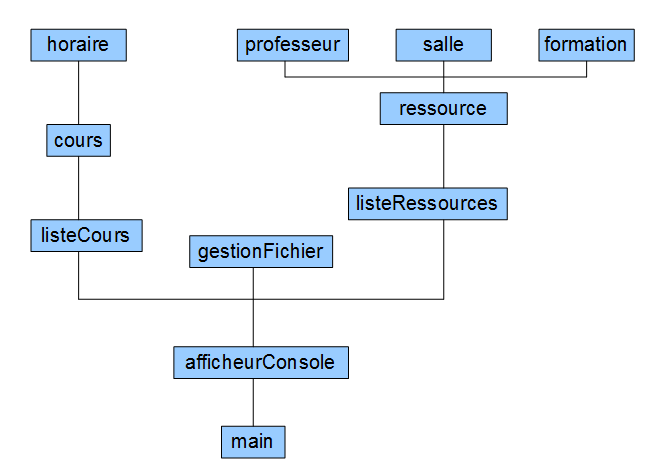
Problèmes rencontrés et solutions apportées

Organisation des classes

Notre première organisation présentait des problèmes d’interdépendance :



Nous avons résolu ce problème en mettant au point une nouvelle organisation des classes :

**

Utilisation de github

L’utilisation de github a présenté certaines difficultés au début du projet, cependant nous avons rapidement réussi à prendre en main ce gestionnaire de version.

Conclusion

# Adrien (Chef du projet)

Ce fut pour moi une nouvelle expérience, c’est la première fois que nous avons eu 3 gros projets à gérer en même temps, la répartition des tâches et le temps pour chacune en ont souffert. Malgré cela, la cohésion du groupe nous as permis de réussir à terminer le projet et à implémenter plusieurs fonctionnalités supplémentaires. Nous aurions voulu pouvoir ajouter des fonctionnalités tel que modifier des ressources, une interface graphique et pouvoir écrire des codes plus propres, notamment sur les copier/coller inutiles.

Faire du code propre, au début c’était facile, mais avec la fin du temps, c’est devenu très difficile à gérer.

# Hicham

Pour ce projet nous devions modifier nos habitudes sur la manière dont nous codons. Un code propre est au final beaucoup plus facile à relire pour notre code, et à lire pour le code des autres membres du groupe. J’ai pu ainsi passer moins de temps sur la compréhension du code. Les tests se sont révélés tout aussi utile. Des tests des plus basiques sur des fonctions m’ont permis d’identifier rapidement des maladresses qui auraient peut-être empêcher le programme de fonctionner correctement. J’utiliserais très probablement ces méthodes pour coder proprement dans mes futurs projets.