Laboratoire 4
Introduction aux outils de documentation automatique et de versionnement de code

ELE4205 - Département de génie électrique
Polytechnique Montréal

14 septembre 2018

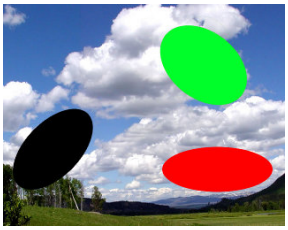
Table des matières

1 Introduction

L'objectif de ce laboratoire est de vous familiariser avec l'outil de documentation automatique Doxygen et avec GIT, l'outil de versionnement de code que vous avez brièvement utilisé lors des laboratoires précédents. Vous aurez à créer une branche d'un logiciel auquel vous appliquerez des *patchs* dont le code vous est fourni.

2 Mise en contexte

Un photographe prend des photographies d'objet de forme elliptique dans la nature. Ce dernier aimerait analyser les caractéristiques des ellipses telles que la longueur de l'axe principal, l'orientation des axes majeur et mineur. Cependant, ceux-ci n'ont pas la même orientation, et il donc difficile de les analyser.



Le photographe prend au plus une ellipse par photo. Le programme que vous allez utilisé pour ce laboratoire génère des ellipses aléatoirement, écrit sur le disque les images des ellipses générées, les réaligne pour que l'axe principal soit à l'horizontal et les sauvegarde par la suite. Le logiciel utilise OpenCV pour la détection des ellipses.

3 Introduction au système de versionnement de code

Dans vos cours de programmation C/C++, vous deviez faire des programmes en équipe de deux pour chacun des laboratoires. Vous utilisiez alors des courriels avec pièces jointes ou un système de fichiers sur le *cloud* tel que Google Drive, Dropbox ou autre lorsque vous vouliez travailler chacun sur une partie du logiciel. Vous avez sûrement eu des conflits de fichiers lorsque vous travailliez les deux sur le même fichier.

Dans le monde professionnel, ces systèmes, bien qu'accessibles au grand public, sont très peu utilisés dans le cadre d'une équipe de développement. Un système de versionnement de code est utilisé.

Qu'est-ce-qu'un système de versionnement de code ? C'est un système de suivi de l'historique des différentes versions du code sous la forme d'un arbre avec des branches qui contiennent différentes versions du système de fichier d'un projet. Chacune des branches peut posséder plusieurs noeuds ordonnancés de façon chronologique. Certaines branches peuvent également être le commencement d'un nouveau projet ou d'une nouvelle version. Pas exemple deux versions majeures d'un logiciel sont disponibles : la version 2.6.4 et la version 3.0 qui vient juste d'être lancée. Vous allez recevoir de *bug reports* pour les deux branches qui vont évoluer en parallèle tant que la version 2.6.x sera supportée et on peut voir dans l'historique à quel moment un nouveau bogue a été réglé (ou introduit). Ce genre de système est pratique car aucune donnée ne peut être perdue ou corrompue. Il incorpore également un mécanisme de *merge* de fichiers, c'est-à-dire que des versions du même fichier peuvent générer un conflit et un fichier avec les différences entre les deux versions sera produit pour aider le développeur à fusionner les deux versions cela est requis.

Plusieurs systèmes de versionnement de code sont utilisés dans le monde *open-source* et professionnel dont

- Git : Développé initialement par Linus Torvald puis par les développeurs de Linux pour remplacer BitKeeper, il est maintenant intégré dans la plupart des distributions. Il est de plus en plus utilisé à cause de sa flexibilité et de sa fiabilité.
- SVN : Développé par la fondation Apache, ce système est très utilisé à cause de sa simplicité d'utilisation.
- Hg : Développé en même temps que GIT par des utilisateurs de l'ancien système de versionnement payant BitKepper utilisé pour le suivi du noyau de Linux.

Ils offrent une compatibilité complète avec Microsoft Windows, Mac OS X et Linux. Dans le cadre de ce laboratoire, nous allons utiliser Git pour gérer un projet de développement.

3.1 *Fork* du répertoire GIT du laboratoire 4 sur BitBucket

Pour utiliser un répertoire GIT, il faut un serveur qui peut l'héberger. De nombreux sites offrent l'hébergement de projets Open-Source, c'est-à-dire que les sources doivent être disponibles de tout le monde. En voici quelques-uns :

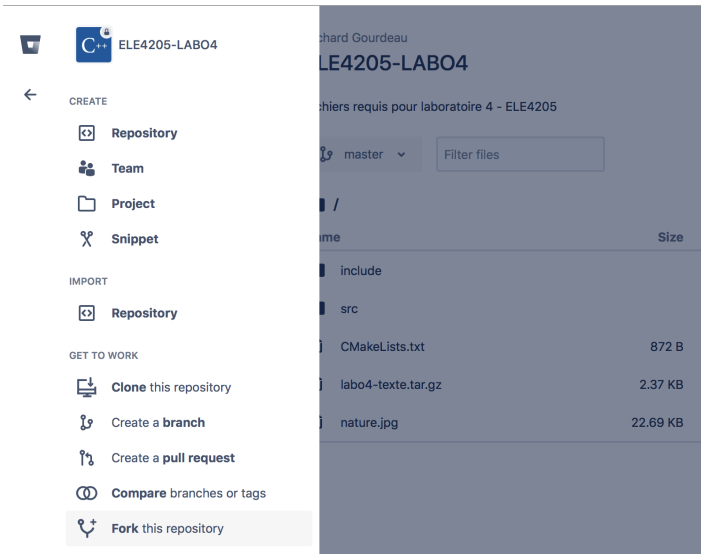
- Github
- Sourceforge
- Codeplex

— etc.

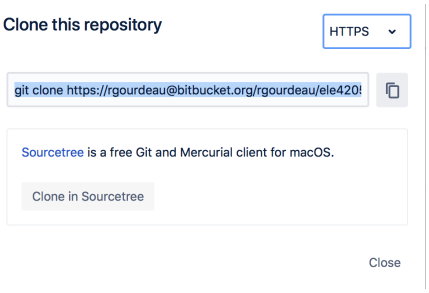
Cependant, lorsque que l'on ne veut pas rendre public nos sources, la plupart des sites webs d'hébergement offrent des services avec différents forfaits. Bitbucket en fait partie, mais tout compte avec une adresse courriel universitaire (votre compte de Polytechnique) peut héberger ses sources gratuitement avec un nombre illimité de collaborateurs. Nous allons en profiter pour y héberger notre projet.

Puisque nous travaillons déjà dans un répertoire GIT, nous n'avons qu'à *forker* le projet existant sur BitBucket. Un *fork* d'un projet est une copie du projet original, laquelle contient des modifications au code qui peuvent être approuvées plus tard par les développeurs officiels. Ce mécanisme est utilisé très souvent dans les projets *Open-Source*. Par exemple, le moteur de jeu Unity 3D publie les sources des différents modules et des collaborateurs non-officiels peuvent créer des *forks* pour améliorer certaines fonctionnalités pour les soumettre puis les faire approuver par les développeurs. Il serait très dangereux de donner les droits d'administrateur à n'importe quel développeur dans ce cas-ci.

Allez sur Bitbucket et créez un fork du projet du laboratoire 4 (`ELE4205-LAB04`). Vous devez d'abord allez dans le dépôt `ELE4205-LAB04`, puis cliquez sur le **+** à gauche et cliquez sur `Fork this repository`.



Entrez comme nom `ELE4205-LAB04-<votre numéro d'équipe>` et cliquez sur `Fork Repository`. Vous serez alors sur la page principale de votre *fork* du projet du laboratoire 4. En haut à droit se situe le bouton `Clone` pour télécharger le répertoire GIT.



Clonez le répertoire GIT avec l'adresse dans la figure (votre adresse!!) :

```
git clone https://rgourdeau@bitbucket.org/rgourdeau/ele4205-labo4-test.git
```

Vous savez maintenant utiliser GIT pour créer un *fork* d'un projet. Nous reviendrons sur des fonctionnalités plus avancés de GIT plus loin dans le laboratoire, mais avant de travailler avec Doxygen, nous « simuler » une édition des fichiers source en décompressant des nouvelles versions du programme (dans le répertoire `ele4205-labo4`) :

```
% tar -xvf labo4-texte.tar.gz
% git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

modified:   include/ImageManager.h
modified:   src/ImageManager.cpp
modified:   src/main.cpp

Untracked files:
  (use "git add <file>..." to include in what will be committed)

include/ImageManager.h~
src/ImageManager.cpp~
src/main.cpp~

no changes added to commit (use "git add" and/or "git commit -a")
```

Il y a des versions modifiées et des nouveaux fichiers (des *backups* `*~`). Comme on ne veut pas suivre les *backups*, nous allons modifier notre `.gitignore` :

```
% echo "*~" >> .gitignore
% git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

modified:   include/ImageManager.h
modified:   src/ImageManager.cpp
modified:   src/main.cpp

Untracked files:
  (use "git add <file>..." to include in what will be committed)

.gitignore

no changes added to commit (use "git add" and/or "git commit -a")
```

Avant de faire un `commit` sur ces modifications, nous allons documenter nos sources.

4 Introduction à l'utilisation de Doxygen

La première partie du laboratoire va être consacrée à la documentation de l'application.

Doxygen est largement utilisé pour la documentation des projets *Open-Source* et commerciaux. Ce système peut générer, par exemple, des pages HTML pour la documentation en ligne, etc, en interprétant les commentaires insérés dans le code. Dans le laboratoire 1, par exemple, le code a été commenté pour que Doxygen puisse les interpréter.

4.1 Configuration de Doxygen manuellement

Récupérez le code du premier laboratoire. Faites cette commande :

```
% doxygen -g
```

Ceci génère un fichier de configuration de Doxygen. Éditez maintenant le fichier de configuration :

```
% gedit Doxyfile &
```

Spécifiez ces lignes :

```
OUTPUT_DIRECTORY       = doc
INPUT                  = include src
```

La première ligne indique l'endroit où les pages vont être générées et la deuxième ligne indique les répertoires/fichiers à inclure. Doxygen va se charger de lire automatiquement les fichiers `.c`, `.cpp` et `.h` dans ces répertoires.

Sauvegardez le fichier de configuration et lancez Doxygen :

```
% doxygen Doxyfile
```

Vous devriez apercevoir des pages en LaTeX et HTML générées dans le répertoire `doc`. Ouvrer le fichier `index.html` situé dans `doc/html` et naviguer au travers de la documentation.

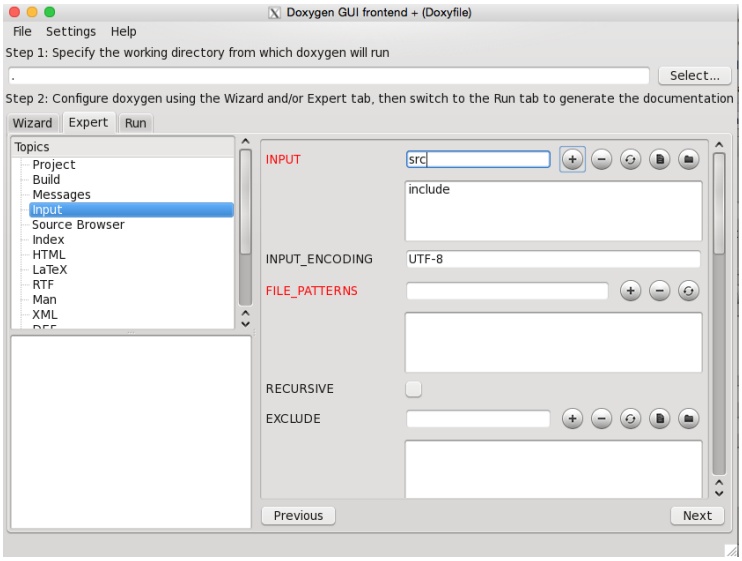
4.2 Configuration de Doxygen avec Doxywizard

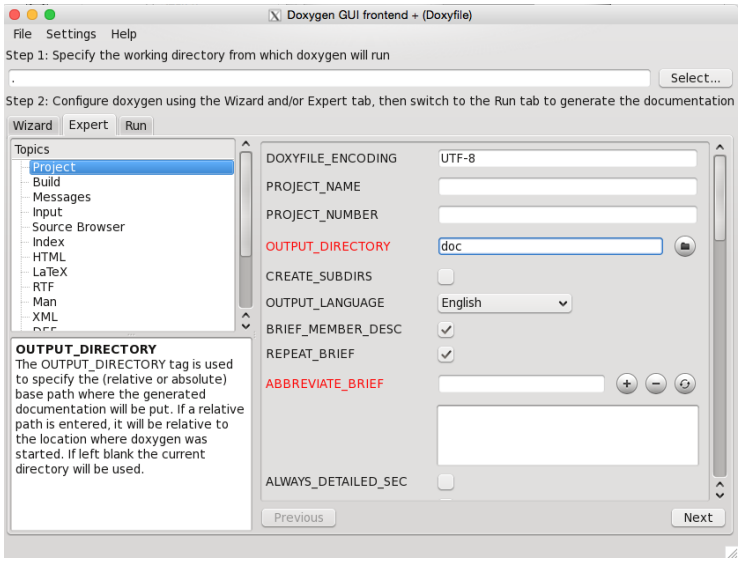
Il existe une interface graphique pour Doxygen : Doxywizard, qui permet une meilleure ergonomie pour la configuration de Doxygen.

Allons dans le répertoire du laboratoire 4 (votre *fork*), exécutez le programme :

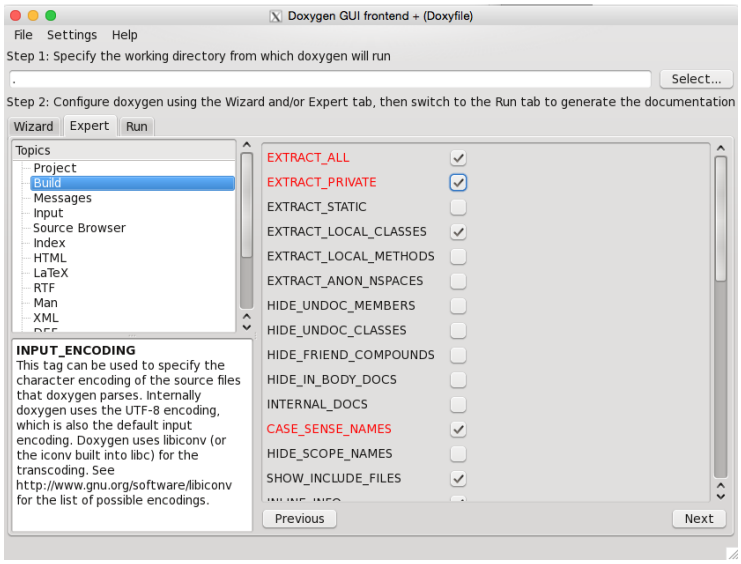
```
% doxygen -g
% doxywizard Doxyfile
```

Le programme démarre dans le mode Wizard. Passons en mode Expert. Ajoutons nos deux répertoires pour les sources (`include` et `src`). Le mode Expert offre plus de possibilités. Ajoutons le répertoire de sortie `doc`.





Allez dans l'onglet `Expert->Build`. Cochez Extract All et Extract Private. La première option liste toutes les fonctions/variables même si elles ne sont pas documentées et la deuxième liste les attributs privés.



Nous allons également utiliser des images pour la documentation. Dans l'onglet `Expert->Input`, ajoutez le répertoire racine du projet `.` qui contient l'image `nature.jpg` (variable `IMAGE_PATH`). Puis dans `Expert->Source Browser`, cochez `Source Browser` afin d'avoir les sources disponibles dans la documentation.

Vous pouvez maintenant aller dans l'onglet `Run` puis cliquer sur `Run Doxygen`. Les pages devraient également avoir été générées. Allez les parcourir avec votre fureteur.

Sauvez la configuration et quittez Doxywizard.

5 Documentation du code de l'application du laboratoire 4

Maintenant que vous savez comment exécuter Doxygen, nous allons documenter notre application.

5.1 Documentation de `ImageManager.h`

Nous allons utiliser KWrite pour commenter le code.

```
% kwrite &
```

`ImageManager` est une classe qui permet la gestion des fichiers image. Voici la description des fonctions de la classe :

— `ImageManager()` : Constructeur de la classe ImageManager. Ne fait rien d'utile.

— `ImageManager()` : Destructeur de la classe ImageManager. Libère automatiquement les données allouées par les conteneurs de la STL.

— `void loadData(std::string filename)` : Permet de lire un fichier texte qui contient la liste des images à lire et les enregistre en mémoire.

— `void generateEllipses(int num)` : Permet de générer num ellipses. Les ellipses sont générées aléatoirement (longueur des axes et angle). Elles sont également enregistrées sur le disque et en mémoire sous format `ellipse*.tif`.

— `void calculate()` : Calcule l'orientation des ellipses enregistrées en mémoire et les enregistre sur le disque lorsqu'elles sont alignées dans le format `*_alligned.png`.

— `void saveImages()` : Enregistre sur le disque les images dans la mémoire avec leur nom original.

— `int getNumImages() const` : Retourne le nombre d'images enregistrées en mémoire.

Voici la description des attributs privées de la classe :

— `std::vector<std::string> _files` : la liste des fichiers lus par la fonction `loadData`.

— `std::map<std::string,cv::Mat> _images` : les images enregistrées en mémoire avec leur nom respectif.

— `int calculateAngle(std::map<std::string,cv::Mat>::iterator it)` : Fonction qui retourne l'angle d'une image enregistrée en mémoire.

— `void rotateImage(cv::Mat & image, int angle)` : Effectue rotation une sur l'image d'un certain angle en degrés. Note : la grandeur de l'image reste inchangée.

Voici finalement la description des constantes globales contenues dans ce fichier :

— `const int MAX_IMAGE_SIZE = 2048` : Définit la grandeur maximale des images (2048x2048).

— `const int MIN_IMAGE_SIZE = 320` ; Définit la grandeur minimale des images (320x320).

— `const int MIN_ECLIPSE_AXIS = 160` ; Définit la grandeur minimal de l'axe d'une ellipse (160 pixels).

Vous utiliserez ces définitions de fonctions lorsque vous les documenterez.

Vous pouvez lire le tutoriel de Doxygen de notes de cours ou bien cette feuille de référence rapide qui est très suffisante.

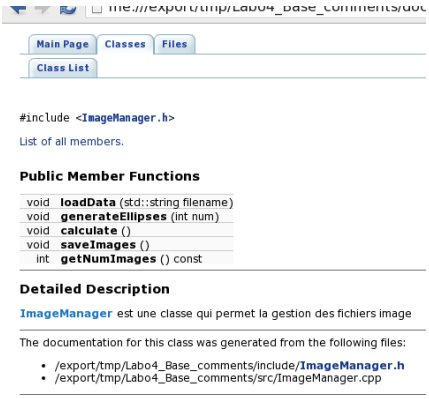
5.2 Documentation de la classe `ImageManager`

Ajoutez maintenant ce bout de commentaire dans le fichier `ImageManager.h` avant la déclaration de la classe :

```
/**
\class ImageManager
ImageManager est une classe qui permet la gestion des fichiers image

**/
```

Compilez et regardez le résultat (`index.html`) : Doxygen s'est chargé de générer la page de la classe `ImageManager` ! Les commentaires qui commencent par `/**` sont interprétés par `Doxygen`.



5.3 Documentation de la fonction `generateEllipses`

Ajoutez ce commentaire avec la déclaration de la fonction `generateEllipses` :

```
/**
        \fn generateEllipses(int num)
        \brief Permet de générer aléatoirement des ellipses
        \param num
        Le nombre d'ellipses à générer.
        **/
```

Vous aurez alors la fonction `generateEllipses` documentée.

5.4 Documentation de l'attribut privé _images

Il n'est pas nécessaire pour le développeur qui utilisera la classe de connaître la fonctionnalité de chacun des attributs privés. Toutefois, il est quand même préférable de les documenter. Premièrement, pour aider les développeurs qui pourraient participer au développement de la classe, et deuxièmement, pour garder une trace de cette variable.

La documentation d'une variable est assez simple. Insérez ce commentaire :

```
/**
        \var _images
        \brief Les images enregistrées en mémoire avec leur nom respectif
        **/
```

Vous aurez alors l'attribut privé `_images` documenté.

5.5 Documentation de la page principale de la documentation

La page principale est vide et n'est pas très invitante jusqu'à maintenant. Nous allons donc mettre le contenu de la mise en contexte sur la page principale. Commentez `main.c` :

```
/**
\mainpage
Un photographe prend des photographies d'ellipses dans la nature.
Ce dernier aimerait analyser les caractéristiques de celles-ci
telles que longueur de l'axe principal, axe majeur, mineur. Cependant,
ceux-ci n'ont pas la même orientation, et il donc difficile de
les analyser.

\image html nature.jpg

Le photographe prend au plus une ellipse par photo. Le programme que
 vous allez utilisé pour ce laboratoire génère des ellipses aléatoirement,
écrit sur le disque les images des ellipses générées, les réaligne pour
que l'axe principal soit à l'horizontal et les sauvegarde par la suite.
Le logiciel utilise OpenCV pour la détection des ellipses.
```

```
**/
int main(int argc, char** argv)
{
```

Vous devriez avoir cette page de couverture générée.

La page de couverture est maintenant beaucoup plus accueillante.

5.6 Exercice : Documentez `ImageManager.h` en entier

Maintenant que vous savez les bases de Doxygen, documentez le restant du fichier `ImageManager.h`, i.e. les constantes globales au début, les attributs/méthodes privées, publics, etc.

6 Ajout de nos modifications avec GIT

Maintenant que nous avons fait nos modification, nous allons utiliser GIT pour faire un `commit` de nos nouveaux fichiers puis un `push` vers notre serveur (voir les notes de cours pour l'utilisation de GIT).

La commande

```
% git status
```

nous liste les changements apportés depuis notre dernière mise à jour. On remarque qu'il y a nos fichiers modifiés et de nouveaux fichiers Doxygen et le répertoire `/doc`. Comme le répertoire `/doc` ne contient que des fichiers générés par Doxygen, il ne faut pas les mettre sous contrôle de version. Nous allons donc exclure ce répertoire :

```
% echo "doc/" >> .gitignore
```

Maintenant, la commande

```
% git status
```

ne liste plus le répertoire généré par Doxygen. On peut placer dans `.gitignore` des répértoires, des noms de fichiers, des *wildcards*. Par exemple, on exclut les fichiers objets résultant d'une compilation :

```
% echo "*.o" >> .gitignore
```

On peut ajouter nos nouveaux fichiers avec

```
% git add .gitignore Doxyfile
```

Mais vant de faire un `commit` et un `push`, il faut vérifier nos informations personnelles :

```
$ git config --global --list
```

permet d'en afficher le contenu. Donc la première chose à faire est de configurer l'identité de l'utilisateur.

```
$ git config --global user.name "Prenom Nom"
$ git config --global user.email prenom.nom@polymtl.ca
```

Il existe une interface graphique pour GIT, SmartGit. Vous pouvez l'utiliser avec la commande :

```
% smartgit &
```

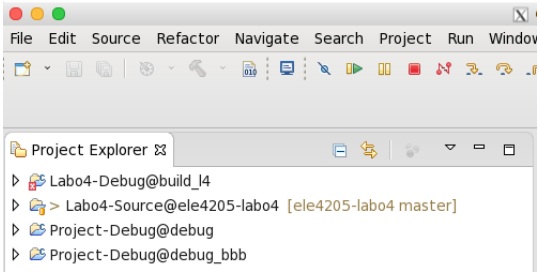
Vous pouvez ajouter votre répertoire du laboratoire avec `Repository->Add or Create`. Vous pouvez maintenant gérer votre dépôt (copies locale et distante) avec SmartGit (dans le laboratoire, ignorez les messages de mises à jour, faites *Cancel*).

7 Projets Eclipse

Utilisons maintenant ce projet sous Eclipse.

```
% cd ..
% mkdir build_l4
% cd build_l4
% cmake -G"Eclipse CDT4 - Unix Makefiles" -D CMAKE_BUILD_TYPE=Debug \
 -DCMAKE_ECLIPSE_GENERATE_SOURCE_PROJECT=TRUE ../ele4205-labo4
```

Nous avons maintenant deux projets un Debug et un Source pour l'édition et le contrôle de version. Allons les charger dans Eclipse avec `File->Import` (les deux répertoires : `build_l4` et `ele4205-labo4`) pour obtenir



Le projet `Labo4-Debug` a un X rouge indiquant un(des) problème(s), dans les faits, les *out of source* causent problème pour l'indexage et le contrôle de révision. L'option `-DCMAKE_ECLIPSE_GENERATE_SOURCE_PROJECT=TRUE` permet de générer un deuxième projet dans le répertoire source qui lui est complèment fonctionnel pour l'édition et les révisions. Il ne faut pas mettre ces projets sous contrôle de GIT, il faudra ajouter `.project` au `.gitignore`. On remarque l'icône de contrôle de version et le nom du dépôt pour `Labo4-Source`. Eclipse avec son *plugin* pour GIT peut aussi être utilisé comme client pour certaines opérations de GIT.