Génie logiciel pour (l'entreprise) la modélisation
II: Normes de programmation (style)
Documentation automatique (pseudo codes, doxygen)

Jean-Baptiste Apoung Kamga

Cours M2 IM
Université Paris-sud XI

Plan

Outline

Normes de Programmation

Brian KERNIGHAN & Rob PIKE, 1999

"Programms are read not only by computers but also by programmers"

Les normes de programmation sont encouragées et soutenues par et pour les points

- 3 et 4 de pratiques de *développement* dans l'XP
- 2 et 3 de pratiques de *développeur* dans l'XP

Une norme de programmation est un peu comme une classification des figures de style d'un langage de programmation. Ces figures de style vont permettre aux développeurs d'enrichir leur code avec des informations qui ne se trouvent pas dans le code lui-même. A la relecture, les développeurs partageant cette norme sauront reconnaître ces informations et améliorer leur compréhension du code.

L'essentiel dans une équipe est de trouver une norme qui convienne à tous et que tous les membres de l'équipe la respectent.

Outline

Considérations Générales de définition de Normes

1. Se mettre d'accord sur les noms de variables (convention de nommage).
2. Se mettre d'accord sur la structuration des fichiers (formattage).

Noms de Variables 1/3

Liste de contrôle

Considération générales sur les noms

- Le nom décrit-il de manière complète et précise ce que représente la variable ?
- fait-il référence au problème réel plutôt qu'à la solution en langage de programmation ?
- Le nom est-il assez long pour vous ne soyez pas obligé de deviner sa signification ?
- les quantificateurs de valeurs calculées, s'il y en a, sont-ils placés à la fin du nom ?
- Le nom contient-il *Count* ou *Index* à la place de *Num* ?

Des noms adaptés aux données qu'ils représentent

- Les indices de boucles ont-ils des noms significatifs ?
- Toutes les variables "temporaires" ont-elles été renommées au profit de quelque chose de plus significatif ?
- Les variables booléennes ont-elles des noms qui indiquent clairement leur signification lorsqu'elles sont *true* ?
- Les noms des types énumérés comprennent-ils un préfixe ou un suffixe qui indique la catégorie (*Color_ pour Color_Red*) ?
- Les constantes nommées ont-elles des noms qui désignent les entités abstraites qu'elles représentent plutôt que le nombre qu'elles remplacent (*CYCLES_NEEDED = 6 au lieu de FIVE = 6*) ?

Noms de Variables 2/3

Les conventions de noms

- La convention fait-elle la distinction entre les données locales, de classe et globales ?
- La convention distinguent-elle les noms de types, les constantes nommées, les types énumérés et les variables ?
- Identifie-t-elle les paramètres en entrée seulement des sous-programmes dans les langages qui ne les contrôlent pas ?
- Est-elle aussi compatible que possible avec les conventions standards du langage ?
- Les noms sont-ils mis en forme pour une lisibilité maximale ?

Les noms courts

- Le code utilise-t-il des noms assez longs (sauf si les noms courts sont obligatoires) ?
- Le code évite-t-il les abréviations qui ne font que gagner un caractère ?
- Les mots sont-ils abrégés de manière cohérente ?
- Les noms sont-ils prononçables ?
- Avez-vous utilisé des noms qui pourraient être mal lus ou mal prononcés ?
- Les noms courts sont-ils documentés dans les tables de correspondances ?

Noms de Variables 3/3

Problèmes de noms courants : avez-vous évité...

- ...les noms trompeurs ?
- ...les noms dont les significations sont semblables ?
- ...les noms qui ne diffèrent que par un ou deux caractères ?
- ...les noms contenant des chiffres ?
- ...les noms mal orthographiés pour être plus courts ?
- ...les noms très souvent mal orthographiés ?
- ...les noms en conflit avec les noms des sous-programmes de la bibliothèque standard ou des noms de variables prédéfinis ?

Exemple de convention en C++ et Java

- NomClasse
- NomType
- TypesEnumeres
- variableLocale
- parametreDeSousProgramme
- NomDeSousProgramme()
- m_VariableDeClasse
- g_VariableGlobale
- s_VariableStatique
- CONSTANTE, MACRO
- Base_TypeEnumere

Exemple de convention de nom en C

- NomType
- NomDeSousProgrammeGlobal()
- f_NomDeSousProgrammeFichier()
- Variablelocale
- ParametreDeSousProgramme
- f_VariableStatiqueFichier
- G_GLOBALE_VariableGlobale
- CONSTANTE_LOCALE
- G_GLOBALE_CONSTANTE
- MACROLOCALE()
- G_GLOBALE_MACRO()

Considératin sur le Formattage 1/3

Extension des fichiers

choisir entre (.hh /.cc) , **(.h /.cc)**, (.h /.cpp) et **(.hpp/.cpp)**

Fichiers en-têtes (include)

```
#ifndef MONFICHIER_HPP
#define MONFICHIER_HPP
....
#endif //_MONFICHIER_HPP
```

Acolades et commentaires de blocs

```
{
//Block1 (commentaires significatifs)
 ... du code
  {
    // Bloc2 (commentaires significatifs)
        ... du code
  }// Fin Bloc2
}//Fin Bloc 1
```

Considératin sur le Formattage 2/3

Fichier template de classe

Fichier classe XX.hpp

```
/**Commentaire doxygen*/
#ifndef XX_hpp
#define XX_hpp
//INCLUDES SYTEMES
//INCLUDES PROJETCS
//INCLUDES LOCAUX
//FORWARD REFERENCES
class XX
{
public:
//CYCLE DE VIE
/**Constructeur par défaut.*/
XX(void);
/**Constructor par copie.*/
XX(const XX& from);
/** Destructeur.*/
~XX(void);
// OPERATEURS
/**D'opérator d'affectation.*/
XX& operator=(const XX& from);
//OPERATIONS
//ACCES
//QUESTIONNEMENT (eg Is..,Has..)
protected:
private:
//VARIABLES
//FONCTIONS PRIVEES
};
//INLINE METHODS
//EXTERNAL REFERENCES
#endif // _XX_h_
```

Fichier classe XX.cpp

```
#include "XX.h" // class implemented
//////////// PUBLIC /////////////////
//======== CYCLE DE VIE ================
XX::XX()
{
}// XX
XX::XX(const XX&)
{

}// XX
XX::~XX()
{
}// ~XX

//=========== OPERATEURS ===============
XX&
XX::operator=(const XX& rXX);
{
  if(this != &rXX)
   {
    ....
   }
 return *this;
}// =
//============OPERATIONS===============
//============ACCES====================
//============QUESTIONNEMENT============
//////////PROTECTED//////////////////////
////////// PRIVATE//////////////////////
```

Considération sur le Formattage 3/3

Autres détails

Voir TP.

Consulter les conseils sur les bonnes pratiques en c++

- http://www.abxsoft.com/ccs/ccs-std.html
- http://www.possibility.com/Cpp/CppCodingStandard.html
- http://www.doc.ic.ac.uk/lab/cplus/c++.rules/

Outline

Processus de Programmation Pseudo Code (PPP)

Bien fondé

- Un moyen de produire sans paresse, une documentation du code
- Une méthode de création efficace de classes et de sous-programmes

S'oppose aux méthodes alternatives suivantes

- **Développement "tests d'abord" :** tests sont écrits avant le code
- **Réaménagement "refactoring" :** amélioration du code par une série de transformation préservant la sémantique
- **Conception par contract :** chaque sous-programmes conçus du point de vue de ses préconditions et postconditions.
- **Bricolage :** bricoler jusqu'à obtenir un code qui fonctionne.

Etapes de construction avec PPP 1/2

D'une classe

- **création d'une construction générale** : ses rôles spécifiques, les "secrets" qu'elle doit masquer, quelle abstraction elle représentera. Dérivera-t-elle d'une autre classe ? Quelles classes pourront en être dérivées ? Quelles sont les méthodes publiques principales et quelles seront les données membres ?
- **construction des sous-programmes de la classe** : construire les sous-programmes identifiées ci-dessus l'un après l'autre
- **révision et tests de la classe** : tester chaque sous-programme puis tester au niveau global ce que les sous-programmes n'ont pas vérifié.

Etapes de construction avec PPP 2/2

D'un sous-programme

- **Conception et vérification** : Vérifiez les conditions préalables, définissez le problème qui doit être résolu par le sous-programme, nommer le sous-programme, choisissez la méthode de test du sous-programme, rechercher les fonctionnalités disponibles dans les bibliothèques standards, pensez au traitement d'erreur, réfléchissez à l'efficacité, recherchez les algorithmes et les types de données, écrivez le **pseudocode**, réfléchissez aux données, vérifiez le pseudocode, essayez plusieurs idées en pseudocode et conservez la meilleure.
- **Codage** : écrivez la déclaration du sous-programme, transformez le pseudocode en commentaire de haut niveau, écrivez le code en dessous de chaque commentaire, vérifier si le code doit être mieux élaboré.
- **Vérification du codage** : vérifiez mentalement les erreurs de fonctionnement, compilez le sous-programme, exécutez le code pas à pas dans un débogueur, testez le code, corrigez les erreurs.
- **Le netoyage des vestiges inutiles** :
- **Reprise** :recommencez si insuffisance ou insatisfaction.

Points clés sur la PPP

- La construction de classes et celles des sous-programmes est un processus itératif. Les connaissances acquises au cours de la construction de certains sous-programmes s'avèrent souvent utiles à une meilleure conception de la classe.
- L'écriture de pseudocode de bonne qualité impose l'utilisation d'un langage naturel compréhensible, en évitant des fonctionnalités propres à un langage de programmation particulier et en écrivant au niveau des intentions (c'est-à-dire en décrivant ce que fait la conception plutôt que la manière dont elle le fait).
- Le processus de Programmation en Pseudocode est un outil de conception utile qui rend le codage facile. Le pseudocode se transforme en commentaires, ce qui garantit la justesse et l'efficacité de ceux-ci.
- Ne vous arrêtez pas à la première conception qui vous vient à l'esprit. Développez plusieurs approches en pseudocode et choisissez la meilleure avant de commencer à coder.
- Vérifiez votre travail à la fin de chaque étape et encouragez les autres à faire de même. Vous pourrez ainsi déceler des erreurs au niveau où elles coûtent le moins cher, c'est-à-dire au moment où l'effort investi est minimal.

Outline

Introduction

C'est quoi Doxygen ?

- Système de documentation automatique de code (en C, C++, Java, etc.)
- Production de documentation aux formats : HTML, RTF, Latex, etc.

Pourquoi Doxygen ?

Outils très utile

- au développeur : maintenance et compréhension des larges projets,
- à utilisateur : génération de documentation (exemples, etc.)

Qui sont les utilisateurs de Doxygen ?

- Dimitri van Heesch
- Beaucoup d'autres développeurs

Concepts

Pour une utilisation de Doxygen il faut pour chaque projet :

1. Créer un fichier de configuration
2. Documenter le code selon certains critères

Concepts 1/3 : Configuration

Fichier Configuration

Le fichier de configuration contient les informations sur

- Ce qui va être commenté
- Comment ces commentaires seront orchestrés

On reviendra sur la génération du fichier de configuration.

Les grandes lignes du fichier

```
PROJECT_NAME           ="Votre projet"
#//Le nom du projet
OUTPUT_DIRECTORY       ="/home/username/public_html/doc/"
#//Répertoire où Doxygène mettre la documentation
INPUT                  = "src/"
#//Lieu où doxygen cherchera les sources à commenter
RECURSIVE              = YES
#//Autorise Doxygen à explorer les sous-répertoire
GENERATE_HTML          = YES
#//Dit à Doxygen de générer du html
HTML_OUTPUT            = html
#//dit à doxygen où mettre les fichiers html.
#//ce répertoire est relatif à  OUTPUT_DIRECTORY
HTML_FILE_EXTENSION    = .html
#//Dit quelle sera l'extension des fichiers html (.htm ou html)
Une entête de page et un bas de page des fichier html peuvent être fournies
avec les options HTML_HERADER et HTML_FOOTER
#// Il y a des options similaire pour les formats RTF, LATEX, Man Pages
```

Concepts 2/3 : Documentation Doxygen du code

Doxygen génère les documentations à partir de deux formes de commentaires :

- de type /** */ et de type ///

commentaires avant l'entité

```
/**
 * La documentation de la classe ici
 */
class UneClasse{
/// documentation de variable
int m_Var;

/** Documentation de Methode1 */
void MethodOne();

///Documentation de la Methode2
void MethodTow();
};
```

commentaires après l'entité

```
/**
 * La documentation de la classe ici
 */
class UneClasse{

int m_Var;///<Documentation de variable

void
MethodOne();/**<Documentation de Methode1*/

void
MethodTwo();///<Documentation de Methode2
};
```

Il existe une variété de commandes pouvant apparaître entre ces blocs de commandes spéciales. Ces commandes sont précédées de \ ou @. Les plus répandues sont : **param, return, author, date, file**

commentaires d'une méthode

```
/**
* Prendre un entier et l'élever au carré
*\param a L'entier a élever au carré
*\return Le carré de l'entier
*/
int square(int a);
```

commentaires d'un fichier

```
/**
*\file UnFichier.hpp
*\author Some Guy
*\date 11-01-11
*/
```

Il y a bien d'autres commandes utiles et des options de formattage (consuster la documentation)

Concepts 3/3 : Exécuter Doxygen

Génération du fichier de configuration

Pour générer la configuration, il suffit d'utiliser l'option **-g** .

Génération du fichier de configuration

```
doxygen  -g  Doxyfile.cfg
```

Génération de documentation

Il suffit d'utiliser **doxygen** suivie du fichier de configuration

Génération de documentation

```
doxygen Doxyfile.cfg
```

D'autres références

La présentation ci-dessus est conçue bien succincte, une référence détaillée est accessible sur la page web de Doxygen :
http://www.stack.nl/~dimitri/doxygen/

Configuration et compilation d'un code

- Exemple de code DefensiveProgramming (voir TP1)
- Récupération, en ligne de commande, de code sur la toile puis configuration, compilation et installation (exemple avec superLU voir ci-dessous)
- Portage de code sur les outils graphiques (Anjuta, qtcreator, kdevelop)

Récupération auto de superLU

```
#! /bin/sh
CURRENT_DIR=‘pwd‘
# getting the tar sofware
wget http://crd.lbl.gov/~xiaoye/SuperLU/superlu_4.1.tar↩
      .gz
# untar
tar -zxvf superlu_4.1.tar.gz
# move in the directory
cd SuperLU_4.1
# copy the real make.inc
/bin/cp /pathTo/my_superLU_make.inc make.inc
# perform specifics changes  on make.inc file
sed -e '/^REPP/  c \ REPP='${CURRENT_DIR}'' \
    -e 's/4.0/4.1/' \
    < make.inc > make2.inc

/bin/mv make2.inc make.inc
#compile
make -j8
# build examples
make testing
# build doc
doxygen DoxyConfig
```

Récupération auto de superLU

```
# create the include lib and doc directory
cd ..

if  ! test -d ./include
then
    mkdir include
fi

if ! test -d lib
then
    mkdir lib
fi

if ! test -d doc
then
    mkdir doc
fi

cp SuperLU_4.1/SRC/*.h ./include
cp SuperLU_4.1/lib/* ./lib
cp -rp SuperLU_4.1/DOC/* ./doc
#Fin du fichier
```

Portage sur Anjuta, qtcreator

Anjuta

Voir TP

qtcreator

voir TP