Télécom SudParis
2022/2023

Projet Informatique 1ère Année
PRO 3600

Jeu XKube

MOREAU Romain
LANTIGNY Valentin
ROUX Basile
FOISSY Thomas

Enseignant responsable : **SIMATIC Michel**

24 Février 2023

Sommaire

1 Introduction

Ce document explicite les informations à donner dans un rapport de projet. Bien entendu, le plan et le contenu doivent être adaptés pour chaque projet.

2 Cahier des charges

Dans le cadre des projets du module CSC3502, les rôles ne sont pas aussi rigoureusement définis qu'ils peuvent l'être dans des projets réels : l'enseignant responsable est en effet à la fois utilisateur final et maître d'ouvrage (i.e. donneur d'ordre et porteur du besoin fournissant les spécifications fonctionnelles) et l'étudiant chef de projet à la fois maître d'oeuvre (responsable de la conception, du bon déroulement des travaux, de la coordination des divers prestataires et de la qualité technique) et développeur contribuant à la réalisation.

Le cahier des charges précise les attentes du projet en termes de fonctionnalités du logiciel à développer (re-formulation du sujet) et en termes de gestion du projet. Le sujet donné par l'enseignant n'est pas toujours définitif dans ses contours. Aussi après une première étude, il peut être modifié dans ses limites, après accord de l'encadrant.

Le cahier des charges est un document de référence contractuel qui formalise les besoins pour en assurer une compréhension homogène par tous les acteurs et cadrer la mission de ces derniers. Il peut comporter une partie technique énumérant les contraintes techniques à respecter. Il permet de garantir que les livrables fournis par le maître d'oeuvre sont conformes à ce qui a été demandé par le maître d'ouvrage, et d'éviter des modifications de la demande en cours de projet, comme l'introduction de nouvelles fonctionnalités non prévues initialement, par exemple.

3 Développement

3.1 Analyse du problème et spécification fonctionnelle

Cette phase d'analyse a pour objectif de préciser les fonctionnalités du logiciel indépendamment de toute considération purement informatique (module, sous-programme, tableau, pointeur...). La description de cas d'utilisation peut aider à illustrer les fonctionnalités attendues.

Pour une application de taille modeste, la spécification se limite en général à la spécification fonctionnelle qui décrit les fonctionnalités de l'application et les conditions d'utilisation de ces fonctionnalités (opérations à exécuter par l'utilisateur, interactions, règles...), souvent au travers de ``cas d'utilisation''.

Pour un développement plus conséquent, la spécification peut comporter une spécification d'architecture (ou étude technique) pour décrire les moyens techniques à mettre en oeuvre et l'organisation générale (par exemple en couches) de l'application et du système informatique dans lequel elle doit être intégrée.

Indiquer les données manipulées, les liens existants entre ces données, et une représentation ``haut niveau'' des données, indépendamment de toute représentation ``bas niveau'' avec des pointeurs et des tableaux. Par exemple un réseau routier serait modélisé par un graphe ou encore une image monochrome représentée par un arbre quaternaire.

Préciser les caractéristiques de ces données : leur caractère temporaire/persistant, leur caractère statique/dynamique, le volume des données...

Préciser aussi les interfaces du logiciel avec l'utilisateur (affichage graphique ou alphanumérique...). Préparer les maquettes de ces interfaces, ainsi que les **tests de validation** qui permettront d'établir la conformité de l'application aux spécifications.

3.2 Conception préliminaire

Cette première étape de la conception vise à décrire à haut niveau et d'un point de vue "informatique" comment le logiciel est réalisé :

- les structures de données qui seront utilisées dans l'application,
- le stockage de l'information,
- la structuration de l'application en modules,
- toutes les fonctions nécessaires au fonctionnement de l'application (prototypes et description sommaire),
- les tests d'intégration à prévoir.

Cette phase permet de définir de manière descendante la structure du programme en précisant son architecture globale en termes de modules et de sous-programmes (par exemple un module

de gestion de l'interface avec l'utilisateur). Pour chaque sous-programme, on précise son nom, ses paramètres d'entrée et de sortie, ainsi que son rôle.

Ne pas hésiter à faire des schémas : par exemple un schéma contenant la représentation d'un graphe avec des pointeurs.

Ensuite, préparer le plan de **tests d'intégration** en précisant les différents tests qui seront effectués sur le logiciel pour vérifier son bon fonctionnement lors de l'intégration des différents modules.

3.3 Conception détaillée

La phase de conception détaillée a pour objectif de décrire dans le détail **comment vous implantez votre application**, en termes d'unités de programme, de structuration « bas-niveau » des données, et d'algorithmes pour les sous-programmes définis lors de la phase de conception préliminaire. Pensez aussi à préparer les **tests unitaires** (jeux de tests, procédures).

3.4 Codage

Les programmes sources peuvent être commentés comme sur l'exemple donné en annexe de manière à faciliter la génération de la documentation à l'aide de l'outil **doxygen** [1].

Nous recommandons que le code écrit soit sous licence logiciel libre **GNU/GPL**[1].

Ne pas stocker les sources du projet en un seul endroit : prévoir des sauvegardes (un `rm -Rf *` malencontreux, une clé USB qu'on perd ou un petit frère qui casse le disque dur contenant le projet est si vite arrivé...). Dans la pratique, après une sauvegarde, il faut tester que l'on peut relire cette sauvegarde **avant** d'avoir effectivement besoin de les relire...

3.5 Tests unitaires

Les Tests unitaires consistent à s'assurer, au moyen de l'exécution d'un programme, que le comportement d'une fonction ou d'une procédure est conforme à des données préétablies, en comparant les résultats obtenus aux résultats attendus.

Remarque : Usez et abusez des outils `ddd` et `valgrind` pour vos tests.

3.6 Tests d'intégration

Après validation des différentes fonctions, procédez à l'intégration des différents modules et effectuez les tests d'intégration prévus.

0 www.gnu.org/licences/lgpl.html

3.7 Tests de validation

Il faut enfin s'assurer que le logiciel produit est bien conforme aux attentes, donc procéder aux tests de validation pour vérifier sa conformité aux spécifications.

4 Manuel utilisateur

Cette notice a pour objectif de permettre une utilisation rapide, correcte et aisée de votre logiciel. Elle doit préciser les ressources matérielles et système nécessaires pour son installation et son exploitation.

Le manuel comprend :

5 une partie qui explique comment installer et éventuellement administrer votre logiciel : indiquez au minimum les renseignements donnant accès au programme exécutable, aux fichiers sources, aux fichiers de données éventuels. Expliquez aussi la procédure de génération (makefile) du programme exécutable à partir des fichiers sources.

6 un mode d'emploi qui explique comment utiliser le logiciel : ce mode d'emploi gagne à être illustré par un schéma, des écrans de menu présentés à l'utilisateur, et des écrans de résultats obtenus. Mentionner également les contraintes à respecter par l'utilisateur, et signalez les principales causes d'erreurs possibles. Si votre application comporte une interface écran conviviale, avec des aides en ligne possibles, cette notice utilisateur en est simplifiée d'autant !

7 Conclusion

Cette dernière section est **obligatoire** (comme les autres d'ailleurs!). Elle doit contenir une synthèse de ce qui a été rapporté, et indiquer des améliorations souhaitables et/ou des perspectives d'évolution pour le projet réalisé. Elle peut également comporter les remarques personnelles de l'équipe quant à l'intérêt du travail réalisé (aussi bien en positif qu'en négatif!).

8 Bibliographie

[1] Doxygen, « Générateur de documentation html », **www.doxygen.org**

[2]Auteur, Titre. Éditeur, Année d'édition …

[3] Auteur, Titre, URL …

9 Annexes

9.1 Annexe 1 : Gestion de projet

9.1.1 Plan de charge

Exemple d'un plan de charges

9.1.2 Planning prévisionnel

Exemple d'un planning prévisionnel

9.1.3 Suivi d'activités

Exemple d'un suivi d'activités

9.2 Annexe 2 : Code source

```
/**
  @file allocDynamiqueTab.c
  Ce programme compare une fonction qui renvoie un tableau alloué avec malloc et
  une fonction qui renvoie un tableau alloué dynamiquement sur la pile, afin
  de montrer les problèmes liés à cette dernière allocation.

  @warning La compilation de ce fichier avec l'option -Wall génère un warning signalant
  "function returns address of local variable". Ne pas corriger ce warning dans la mesure
  où l'objectif de ce programme est de montrer ce qui se passe quand on retourne l'adresse
  d'une variable locale !
*/

#include <stdlib.h>
#include <stdio.h>

/**
   Affichage à l'écran du tableau d'entiers de nom @a nomT, pointé par @a t et
   contenant @a taille éléments
   @param nomT (donnée) Nom du tableau à afficher
   @param t (donnée) Pointeur sur le tableau à afficher
   @param taille (donnée) Nombre d'éléments dans ce tableau
*/
void afficherTab(char *nomT, int t[], int taille) {
       /. . . .
}

/**
   Fonction renvoyant un tableau pouvant contenir @a taille entiers et initialisé
   à 0, 1, 2...@a taille - 1
   @param taille (donnée) Taille (en nombre d'entiers) du tableau créé
   @return Pointeur sur tableau créé
   @warning Si le tableau ne peut pas être alloué, cette fonction arrête le programme
   en renvoyant EXIT_FAILURE
```

```
*/
int *creerTab(int taille){
       /. . . .
}

/**
   Fonction renvoyant un tableau pouvant contenir @a taille entiers et initialisé
   à 0, 1, 2...@a taille - 1
   NB : la méthode employée pour créer ce tableau est erronée et entraînera
        des erreurs d'exécution (sans arrêt du programme) dans la suite de l'exécution.
        La compilation de ce fichier avec l'option -Wall détecte ce problème en générant
        un warning signalant "function returns address of local variable". Ne pas corriger
        ce warning dans la mesure où l'objectif de ce programme est de montrer ce qui se
        passe quand on retourne l'adresse d'une variable locale !
   @param taille (donnée) Taille (en nombre d'entiers) du tableau créé
   @return Pointeur sur tableau créé
   @warning Si le tableau ne peut pas être alloué, cette fonction arrête le programme en
   renvoyant EXIT_FAILURE
*/
int *creerTabErrone(int taille){
       /. . . .
}

/**
    Fonction principale
    @return EXIT_SUCCESS (respectivement EXIT_FAILURE) si le programme s'est exécuté
    correctment (respectivement s'il a eu des soucis)
*/
int main() {
       /. . . .
}
```

9.3 Annexe 3 : ….