LISV Laboratoire d'ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

TP1 - Maintenance Applicative
Analyse préliminaire et Rétroconception

Halim Djerroud

Révision 0.2

Table des matières

1 Introduction

1.1 Contexte du projet

Le projet `pixel_tracer` est un logiciel de dessin vectoriel écrit en langage C. Il permet de créer et manipuler des formes géométriques (points, lignes, cercles, polygones, courbes de Bézier) dans un environnement de dessin en mode texte.

1.2 Objectifs pédagogiques

L'objectif de ce TP est de vous initier aux techniques de **rétroconception** (reverse engineering) et de **redocumentation** d'une application existante. Ces compétences sont essentielles dans le cadre de la maintenance applicative, qui représente une part importante du travail d'un développeur professionnel.

Ce travail s'inscrit dans le cadre d'une démarche de maintenance applicative en trois phases :

1. **Rétroconception** : Analyse et documentation du code existant (ce TP)
2. **Récupération du modèle de conception** : Transformation vers l'orienté objet (TP2)
3. **Refactoring** : Réécriture en Java avec amélioration du code (TP3)

1.3 Compétences visées

À l'issue de ce TP, vous serez capable de :

— Analyser une base de code existante en C
— Identifier les dépendances entre modules
— Documenter du code de manière professionnelle
— Utiliser des outils de documentation automatique (Doxygen)
— Produire des diagrammes techniques (UML, flux, architecture)
— Identifier les algorithmes métiers d'une application

2 Partie 1 : Découverte et prise en main du projet

2.1 Étape 1.1 : Téléchargement et installation

1. Rendez-vous à l'adresse suivante :
   https://perso.halim.info/iut_25_26/info/maintenance/
2. Téléchargez l'archive du projet `pixel_tracer.tar.gz`
3. Créez un répertoire de travail dédié à ce projet :

```
mkdir ~/maintenance_app
cd ~/maintenance_app
```

4. Décompressez l'archive :

```
tar -xzvf pixel_tracer.tar.gz
cd pixel_tracer
```

2.2 Étape 1.2 : Exploration de la structure du projet

Examinez attentivement la structure du projet et répondez aux questions suivantes dans votre rapport :

1. **Organisation des fichiers** :
   - Combien de fichiers `.c` et `.h` sont présents ?
   - Quelle est la convention de nommage utilisée ?
   - Y a-t-il des sous-répertoires ? Si oui, lesquels et pourquoi ?
2. **Fichiers de configuration** :
   - Identifiez le `Makefile`. Quelles sont les cibles principales ?
   - Y a-t-il un fichier `README` ou une documentation ?
   - Existe-t-il des fichiers de test ?
3. **Dépendances externes** :
   - Quelles bibliothèques externes sont utilisées ?
   - Comment sont-elles incluses dans le projet ?

Conseil : Créez un tableau récapitulatif dans votre rapport listant tous les fichiers sources avec leur rôle présumé.

2.3 Étape 1.3 : Compilation et tests

1. Compilez le projet :

```
make clean
make
```

2. Si la compilation échoue, documentez les erreurs et les solutions apportées.
3. Testez l'exécutable généré :

```
./pixel_tracer
```

4. Explorez les fonctionnalités du programme :
   - Essayez de créer différentes formes (points, lignes, cercles, polygones)
   - Testez les commandes disponibles
   - Manipulez les calques
   - Observez l'affichage en mode texte
5. Réalisez des captures d'écran des différentes fonctionnalités pour votre rapport.

Livrable : Une section dans votre rapport décrivant :

- Les fonctionnalités principales de l'application
- Le mode d'interaction avec l'utilisateur
- Les limites ou bugs éventuels observés

3 Partie 2 : La redocumentation

3.1 Contexte

La redocumentation consiste à analyser le code source existant pour produire une documentation claire, structurée et professionnelle. Cette étape est cruciale lorsqu'on reprend un projet existant, surtout s'il est peu ou mal documenté.

3.2 Étape 2.1 : Lecture de la documentation existante

1. Lisez attentivement la documentation existante du projet disponible à : https://perso.halim.info/iut/info/mantenance/Projet_de_programmation_C_maintenace_app_IUT__pixel_tracer_-1.pdf
2. Identifiez dans votre rapport :
   — Les informations manquantes ou incomplètes
   — Les sections obsolètes ou incorrectes
   — Les points à clarifier ou à approfondir
3. Prenez des notes sur :
   — Les structures de données principales
   — Le vocabulaire métier utilisé
   — Les conventions de codage

3.3 Étape 2.2 : Diagramme de dépendances

3.3.1 Objectif

Identifier et visualiser les relations d'inclusion entre les fichiers sources du projet.

3.3.2 Méthodologie

1. Pour chaque fichier `.c` :
   — Listez tous les `#include` présents
   — Distinguez les includes système (entre `<>`) des includes locaux (entre `""`)
2. Créez une matrice de dépendances ou un graphe orienté montrant :
   — Les fichiers sources (`.c`) en tant que nœuds principaux
   — Les fichiers d'en-têtes (`.h`) en tant que nœuds secondaires
   — Les flèches indiquant les dépendances
3. Identifiez :
   — Les fichiers centraux (très inclus)
   — Les fichiers périphériques (peu de dépendances)
   — Les éventuelles dépendances circulaires (à éviter)

3.3.3 Outils recommandés

— Draw.io (https://app.diagrams.net/)
— PlantUML
— Graphviz
— Lucidchart

Exemple de représentation :

```
main.c --> shape.h
main.c --> layer.h
main.c --> command.h
shape.c --> shape.h
shape.c --> geometry.h
```

3.4 Étape 2.3 : Diagramme de composants

3.4.1 Objectif

Identifier les modules fonctionnels principaux et leurs interactions.

3.4.2 Méthodologie

1. Regroupez les fichiers par domaine fonctionnel. Par exemple :
   - Module **Formes géométriques** : gestion des shapes
   - Module **Calques** : gestion des layers
   - Module **Affichage** : rendering en mode texte
   - Module **Commandes** : interface utilisateur
   - Module **Algorithmes** : calculs géométriques
2. Pour chaque composant, identifiez :
   - Son **nom**
   - Sa **responsabilité principale**
   - Ses **interfaces** (fonctions publiques)
   - Ses **dépendances** vers d'autres composants
3. Dessinez un diagramme de composants UML montrant :
   - Les composants (rectangles)
   - Les interfaces fournies et requises
   - Les relations entre composants

Livrable : Un diagramme de composants UML avec une légende explicative.

3.5 Étape 2.4 : Diagramme de fonctionnement

3.5.1 Objectif

Comprendre et illustrer le flux d'exécution de l'application.

3.5.2 Approches possibles

Choisissez l'une ou plusieurs des approches suivantes :

1. **Diagramme de flux de données** :
   - Montrez comment les données circulent dans l'application
   - De l'entrée utilisateur à l'affichage final
2. **Diagramme d'activité UML** :
   - Illustrez le cycle de vie d'une commande utilisateur
   - Montrez les branchements conditionnels
   - Indiquez les boucles principales
3. **Diagramme de séquence** :
   - Pour un scénario d'utilisation typique
   - Exemple : "Créer un cercle sur un nouveau calque"

3.5.3 Scénarios à documenter

Créez au minimum un diagramme pour chacun des scénarios suivants :

1. Démarrage de l'application
2. Création d'une forme géométrique simple (ligne ou cercle)
3. Ajout d'une forme sur un calque
4. Affichage de la scène complète

3.6 Étape 2.5 : Documentation technique des fonctions

3.6.1 Identification des fonctions principales

Analysez le code et identifiez au minimum 10 fonctions principales. Critères de sélection :

— Fonctions exposées dans les fichiers `.h` (API publique)
— Fonctions appelées fréquemment
— Fonctions complexes ou critiques
— Fonctions métiers essentielles

3.6.2 Format de documentation

Pour chaque fonction, rédigez une fiche technique complète comprenant :

1. **Signature** :

```
type_retour nom_fonction(type param1, type param2, ...);
```

2. **Description** :
   — Rôle de la fonction en 2-3 phrases
   — Contexte d'utilisation
3. **Paramètres** :
   — Nom du paramètre
   — Type
   — Rôle (que représente ce paramètre ?)
   — Contraintes éventuelles (valeurs acceptées, préconditions)
4. **Valeur de retour** :
   — Type de retour
   — Signification de la valeur retournée
   — Codes d'erreur éventuels
5. **Effets de bord** :
   — Modification de variables globales ?
   — Allocation/libération mémoire ?
   — Interactions avec des ressources externes ?
6. **Exemple d'utilisation** :

```
// Exemple de code montrant l'usage typique
```

3.6.3 Utilisation de Doxygen

Doxygen est un outil de génération automatique de documentation à partir de commentaires structurés dans le code source.

Installation :

```
sudo apt-get install doxygen graphviz
```

Format des commentaires Doxygen :

```
/**
 * @brief Crée un nouveau point avec les coordonnées spécifiées
 *
 * Cette fonction alloue dynamiquement la mémoire nécessaire
 * pour créer un nouveau point et initialise ses coordonnées.
 *
 * @param x Coordonnée x du point (en pixels)
 * @param y Coordonnée y du point (en pixels)
 * @return Pointeur vers le point créé, NULL en cas d'échec
 *
 * @warning L'appelant est responsable de libérer la mémoire
 * @see delete_point()
 */
Point* create_point(int x, int y);
```

Génération de la documentation :

1. Créez un fichier de configuration :

   ```
   doxygen -g Doxyfile
   ```

2. Modifiez le `Doxyfile` (paramètres importants) :

   ```
   PROJECT_NAME = "Pixel Tracer"
   INPUT = ./src
   RECURSIVE = YES
   EXTRACT_ALL = YES
   GENERATE_HTML = YES
   GENERATE_LATEX = NO
   ```

3. Générez la documentation :

   ```
   doxygen Doxyfile
   ```

4. Consultez la documentation générée dans `html/index.html`

Livrable :

— Fiches techniques de 10 fonctions principales

— Documentation HTML générée par Doxygen

3.7 Étape 2.6 : Analyse des structures de données

3.7.1 Objectif

Identifier et documenter les structures de données principales utilisées dans l'application.

3.7.2 Structures à documenter

Pour chaque structure `struct` trouvée dans le code :

1. **Nom et définition** :

```
typedef struct {
    // Définition complète
} nom_structure;
```

2. **Rôle** : Que représente cette structure ? (entité métier, conteneur, etc.)
3. **Champs** : Pour chaque champ :
   — Nom
   — Type
   — Signification
   — Valeurs possibles/contraintes
4. **Relations** :
   — Cette structure contient-elle d'autres structures ?
   — Y a-t-il des pointeurs vers d'autres structures ?
   — Représentez ces relations sous forme de diagramme
5. **Utilisation** :
   — Où et comment cette structure est-elle instanciée ?
   — Cycle de vie (création, utilisation, destruction)

3.7.3 Diagramme de classes (préparation pour l'OO)

Bien que le code soit en C, créez un diagramme de classes UML montrant les structures comme des classes :

— Les `struct` deviennent des classes
— Les champs deviennent des attributs
— Les fonctions opérant sur une structure deviennent des méthodes

Exemple :

```
+------------------+
|     Point        |
+------------------+
| - x: int         |
| - y: int         |
+------------------+
| + create()       |
| + move()         |
| + display()      |
+------------------+
```

3.8 Étape 2.7 : Identification des algorithmes métiers

3.8.1 Objectif

Repérer et documenter les algorithmes métiers spécifiques à l'application de dessin vectoriel.

3.8.2 Algorithmes à identifier

Recherchez et documentez les algorithmes suivants (s'ils existent) :

1. **Algorithme de tracé de ligne** :
   - Est-ce l'algorithme de Bresenham ?
   - Comment sont calculés les pixels intermédiaires ?
2. **Algorithme de tracé de cercle** :
   - Algorithme utilisé (Bresenham circulaire, point milieu) ?
   - Gestion de la symétrie ?
3. **Courbes de Bézier** :
   - Ordre des courbes (quadratique, cubique) ?
   - Méthode de calcul des points (De Casteljau, polynomiale) ?
4. **Remplissage de polygones** :
   - Algorithme de scan-line ?
   - Détection point dans polygone ?
5. **Gestion des calques** :
   - Ordre de superposition
   - Algorithme de fusion/compositing
6. **Transformations géométriques** :
   - Rotation, translation, mise à l'échelle
   - Utilisation de matrices ?

3.8.3 Format de documentation des algorithmes

Pour chaque algorithme identifié :

1. **Nom et référence** :
   - Nom de l'algorithme
   - Référence (article, livre, Wikipedia)
2. **Principe** :
   - Explication en français du fonctionnement
   - Schéma ou pseudo-code si nécessaire
3. **Complexité** :
   - Complexité temporelle (notation Big O)
   - Complexité spatiale
4. **Implémentation dans le projet** :
   - Fichier et fonction concernés
   - Particularités de l'implémentation
   - Points d'amélioration éventuels
5. **Tests** :
   - Comment peut-on vérifier le bon fonctionnement ?
   - Cas limites à tester

Important : Ces algorithmes devront être traduits ligne par ligne en Java lors du TP3 (phase de refactoring).

4 Livrables attendus

4.1 Format du rapport

Le rapport devra être rédigé en LaTeX ou en Markdown et converti en PDF. Structure recommandée :

1. **Page de garde** :
   - Titre du TP
   - Vos noms et prénoms
   - Date
   - Groupe
2. **Sommaire**
3. **Introduction** :
   - Contexte
   - Objectifs
   - Méthodologie adoptée
4. **Partie 1 : Prise en main du projet**
   - Structure du projet
   - Compilation et tests
   - Fonctionnalités observées
5. **Partie 2 : Redocumentation**
   - Diagramme de dépendances
   - Diagramme de composants
   - Diagrammes de fonctionnement
   - Documentation des fonctions principales
   - Analyse des structures de données
   - Identification des algorithmes métiers
6. **Conclusion** :
   - Synthèse du travail réalisé
   - Difficultés rencontrées
   - Préparation pour les TP suivants
7. **Annexes** :
   - Code source annoté
   - Documentation Doxygen (lien ou extrait)
   - Captures d'écran supplémentaires

4.2 Contenu minimal attendu

- **1 diagramme de dépendances** entre fichiers sources (clair et lisible)
- **1 diagramme de composants** UML identifiant les modules principaux
- **2-3 diagrammes de fonctionnement** (flux, activité ou séquence)
- **Documentation de 10 fonctions principales** (format professionnel)
- **Analyse de 5-8 structures de données** principales
- **Identification de 3-5 algorithmes métiers** avec documentation
- **Documentation Doxygen générée** (HTML, fournie en archive séparée)

4.3 Modalités de rendu

— **Date limite** : [À préciser par l'enseignant]
— **Format** :
  — Rapport PDF nommé : `TP1_NOM_Prenom.pdf`
  — Archive contenant : rapport, diagrammes sources, documentation Doxygen
  — Nommage de l'archive : `TP1_NOM_Prenom.zip`
— **Plateforme de dépôt** : [Moodle, email, etc.]

5 Critères d'évaluation

Critère	Points	Détails
Prise en main	2	Compréhension du projet, tests fonctionnels
Diag. dépendances	2	Exhaustivité, clarté, pertinence
Diag. composants	3	Identification correcte des modules, notation UML
Diag. fonctionnement	3	Clarté, pertinence des scénarios choisis
Doc. fonctions	4	Qualité, exhaustivité (10 fonctions minimum)
Structures données	2	Analyse complète, diagramme de classes
Algorithmes métiers	3	Identification, explication, documentation
Qualité rédactionnelle	2	Orthographe, clarté, structure du rapport
Doxygen	2	Configuration correcte, génération réussie
Présentation	2	Mise en page, professionnalisme
Total	**25**	

TABLE 1 – Grille d'évaluation détaillée

6 Conseils méthodologiques

6.1 Organisation du travail

1. **Commencez tôt** : Ce TP demande du temps d'analyse et de réflexion
2. **Travaillez de manière itérative** : Faites plusieurs passes sur le code
3. **Prenez des notes** : Documentez vos observations au fur et à mesure
4. **Utilisez des outils** : Doxygen, draw.io, PlantUML...

6.2 Méthodologie d'analyse du code

1. **Vue d'ensemble d'abord** :
   — Parcourez rapidement tous les fichiers
   — Identifiez les modules principaux
   — Comprenez l'architecture générale
2. **Analyse détaillée ensuite** :
   — Étudiez chaque fonction importante

— Tracez les flux d'exécution
— Annotez le code avec des commentaires

3. **Synthèse et documentation** :
   — Créez les diagrammes
   — Rédigez la documentation
   — Vérifiez la cohérence

6.3 Outils recommandés

Outil	Usage
Doxygen	Génération de documentation
Draw.io / Lucidchart	Diagrammes généraux
PlantUML	Diagrammes UML
VS Code	Éditeur de code avec extensions
Graphviz	Visualisation de graphes
GitKraken / SourceTree	Visualisation dépendances (optionnel)

TABLE 2 – Outils utiles pour le TP

6.4 Pièges à éviter

— Ne pas se contenter de recopier la documentation existante
— Éviter les diagrammes trop complexes ou illisibles
— Ne pas négliger la qualité rédactionnelle
— Ne pas tout documenter : concentrez-vous sur l'essentiel
— Éviter le code dans le rapport : privilégiez les annexes

6.5 Questions fréquentes

Q : Dois-je documenter toutes les fonctions ?
R : Non, concentrez-vous sur les fonctions principales (minimum 10). Les fonctions auxiliaires ou triviales peuvent être omises.

Q : Quel niveau de détail pour les algorithmes ?
R : Suffisant pour qu'un développeur puisse comprendre le principe et éventuellement le réimplémenter. Incluez le pseudo-code si nécessaire.

Q : Puis-je travailler en binôme ?
R : [À préciser par l'enseignant]

Q : Les diagrammes doivent-ils être créés avec un outil spécifique ?
R : Non, mais ils doivent être propres et professionnels. Draw.io ou PlantUML sont recommandés.

Q : Comment gérer le code illisible ou mal structuré ?
R : Faites de votre mieux pour le comprendre et le documenter. Mentionnez les difficultés rencontrées dans votre rapport.

7 Ressources complémentaires

7.1 Documentation

— Documentation officielle Doxygen : https://www.doxygen.nl/manual/

— UML : https://www.uml-diagrams.org/
— PlantUML : https://plantuml.com/fr/
— Algorithmes graphiques : https://en.wikipedia.org/wiki/Bresenham's_line_algorithm

7.2 Tutoriels vidéo

— Introduction à Doxygen
— Création de diagrammes UML
— Analyse de code C

7.3 Livres de référence

— *Working Effectively with Legacy Code* - Michael Feathers
— *Code Complete* - Steve McConnell
— *Clean Code* - Robert C. Martin

Conclusion

Ce TP constitue la première étape essentielle de votre projet de maintenance applicative. Une analyse et une documentation rigoureuses faciliteront grandement les phases suivantes (récupération du modèle OO et refactoring en Java).

Prenez le temps de bien comprendre le code existant avant de passer à la suite. La qualité de votre travail de rétroconception conditionnera la réussite de l'ensemble du projet.

Bon courage et bon travail !