TP1

-

Maintenance Applicative

Delcros Andrea
Khaldi Mohamed

Tables des matières

Introduction

Dans le cadre du module de maintenance applicative, nous avons été amenés à étudier une application existante écrite en langage C intitulée *pixel_tracer*. Cette application propose un environnement de dessin vectoriel en mode texte permettant la création et la manipulation de formes géométriques telles que des points, des lignes ou encore des cercles. Le projet fourni ne disposant que d'une documentation partielle, il a été nécessaire d'effectuer un travail de rétro-conception afin de comprendre son architecture et son fonctionnement interne.

L'objectif principal de ce travail est de produire une documentation technique complète à partir du code source existant. Cette démarche s'inscrit dans un processus de maintenance applicative plus large qui se déroulera en plusieurs phases. La première phase consiste à analyser et documenter l'application afin d'en comprendre les différents modules, leurs interactions ainsi que les structures de données utilisées. Les phases suivantes porteront sur la transformation du modèle vers une approche orientée objet puis sur le refactoring de l'application en langage Java.

Pour mener à bien cette étude, nous avons commencé par explorer la structure du projet et compiler l'application afin d'observer son comportement. Une analyse détaillée du code source a ensuite été réalisée pour identifier les fonctions principales, les structures de données et les dépendances entre les différents fichiers. Cette analyse a permis la réalisation de diagrammes techniques ainsi que la rédaction d'une documentation structurée. Enfin, l'outil Doxygen a été utilisé afin de générer automatiquement une documentation à partir des commentaires intégrés au code source.

1. Prise en main du projet

1.1 Structure du projet

Après extraction de l'archive fournie, nous avons examiné l'organisation générale des fichiers constituant l'application. Le projet est structuré de manière modulaire et repose sur plusieurs fichiers sources en langage C ainsi que leurs fichiers d'en-tête associés. Cette organisation permet de séparer les différentes responsabilités de l'application en modules distincts, facilitant ainsi la compréhension et la maintenance du code.

Le fichier principal du programme constitue le point d'entrée de l'application et assure l'initialisation du système ainsi que la gestion de la boucle principale. D'autres modules sont dédiés à la gestion des commandes utilisateur, à la manipulation des formes géométriques, à la gestion des calques et à l'affichage des éléments à l'écran. Chaque module est accompagné d'un fichier d'en-tête contenant les déclarations des fonctions et des structures utilisées.

L'ensemble du projet est compilé à l'aide d'un Makefile qui automatise la compilation des différents fichiers sources et la génération de l'exécutable final. Cette organisation respecte les conventions classiques des projets écrits en langage C et permet une compilation simple et rapide.

1.2 Compilation et exécution

La compilation du projet a été réalisée à l'aide de la commande « make » fournie dans le répertoire racine. Le Makefile compile l'ensemble des fichiers sources et génère l'exécutable correspondant à l'application. Aucune erreur majeure de compilation n'a été rencontrée lors de cette étape, ce qui indique que le projet est fonctionnel dans son état initial.

Une fois la compilation terminée, l'exécutable a été lancé depuis le terminal. L'application s'exécute en mode texte et propose une interface permettant d'entrer différentes commandes afin de créer et manipuler des formes géométriques. Le programme fonctionne selon un modèle interactif dans lequel l'utilisateur saisit des commandes qui sont ensuite interprétées par le module de gestion des commandes.

1.3 Fonctionnalités observées

Les tests réalisés ont permis d'identifier les principales fonctionnalités de l'application. L'utilisateur peut créer différentes formes géométriques, les organiser au sein de calques et demander leur affichage. Le système de calques permet de gérer la superposition des formes et d'organiser la scène de manière structurée. Les commandes entrées par l'utilisateur sont analysées puis transmises aux modules concernés afin d'effectuer les opérations demandées.

L'affichage se fait en mode texte et représente les formes sous forme de caractères dans le terminal. Bien que l'interface soit simple, elle permet de visualiser correctement les formes créées et de comprendre le fonctionnement global de l'application. Quelques limitations ont toutefois été observées, notamment l'absence d'interface graphique avancée et une gestion des erreurs parfois limitée lorsque l'utilisateur saisit des commandes incorrectes.

2. Redocumentation de l'application

L'objectif de cette partie est d'analyser le code source afin de produire une documentation technique claire et structurée. Cette étape de rétroconception permet de comprendre l'architecture interne de l'application, les relations entre les modules ainsi que les principaux mécanismes de fonctionnement. Elle constitue une base essentielle pour les futures phases de transformation vers un modèle orienté objet et de refactoring.

2.1 Diagramme de dépendances

Une première analyse a consisté à étudier les dépendances entre les différents fichiers sources du projet. Pour cela, nous avons examiné les directives d'inclusion présentes dans chaque fichier afin d'identifier les relations entre les modules. Cette analyse permet de comprendre quels fichiers dépendent des autres et de repérer les éléments centraux de l'application.

Le fichier principal dépend des modules de gestion des commandes, des formes et de l'affichage. Le module de gestion des formes interagit avec le module de gestion des calques afin d'ajouter ou de supprimer des éléments dans la scène. Le module d'affichage utilise quant à lui les informations contenues dans les structures de formes et de calques pour produire un rendu en mode texte.

Cette organisation montre que certains fichiers jouent un rôle central dans l'application, notamment ceux liés à la gestion des formes et des commandes. D'autres fichiers ont un rôle plus spécifique et sont uniquement utilisés par un nombre limité de modules. Par exemple nous pouvons voir l'importance du fichier commande dans le projet.

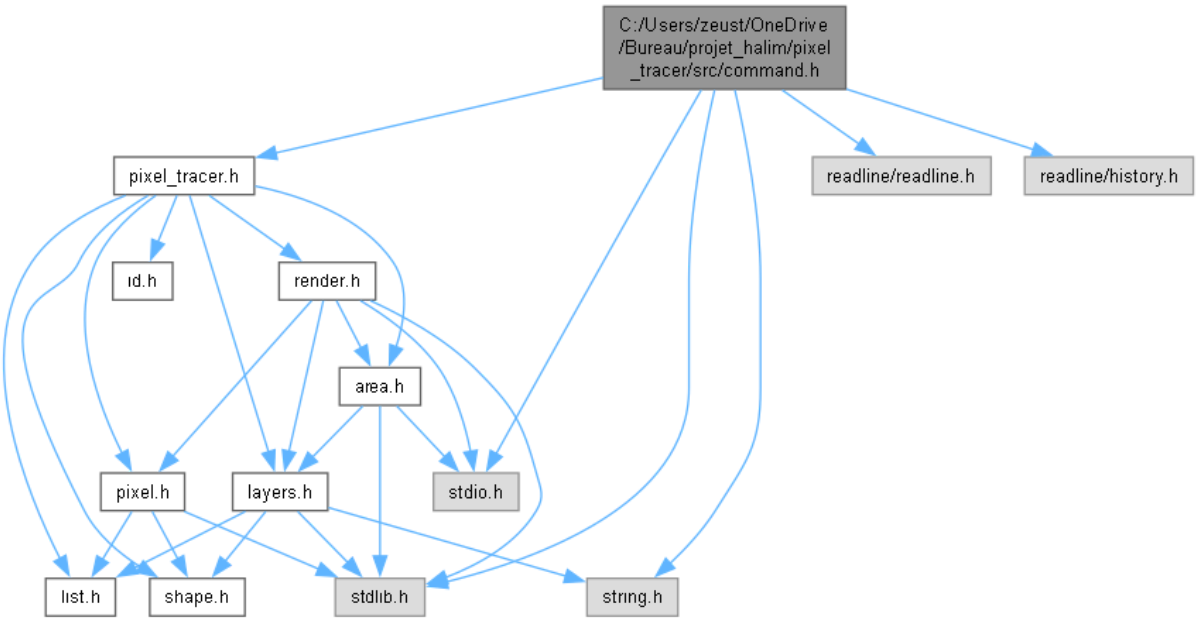

figure 1 : diagramme de dépendance pour commande.h

2.2 Diagramme de composants

Afin d'obtenir une vision plus globale de l'architecture de l'application, les fichiers ont été regroupés en composants fonctionnels. Cette approche permet d'identifier les responsabilités principales de chaque module et les interactions entre eux.

L'application peut être décomposée en plusieurs composants majeurs. Un premier composant est chargé de la gestion des commandes utilisateur. Il interprète les instructions saisies dans le terminal et appelle les fonctions appropriées dans les autres modules. Un second composant gère les formes géométriques, leur création, leur modification et leur stockage. Un troisième composant est dédié à la gestion des calques, permettant d'organiser les formes dans la scène. Enfin, un composant d'affichage est responsable du rendu des formes dans le terminal.

Ces différents composants interagissent entre eux afin d'assurer le bon fonctionnement de l'application. Le module de commandes agit comme un point d'entrée qui coordonne les actions réalisées par les autres modules.

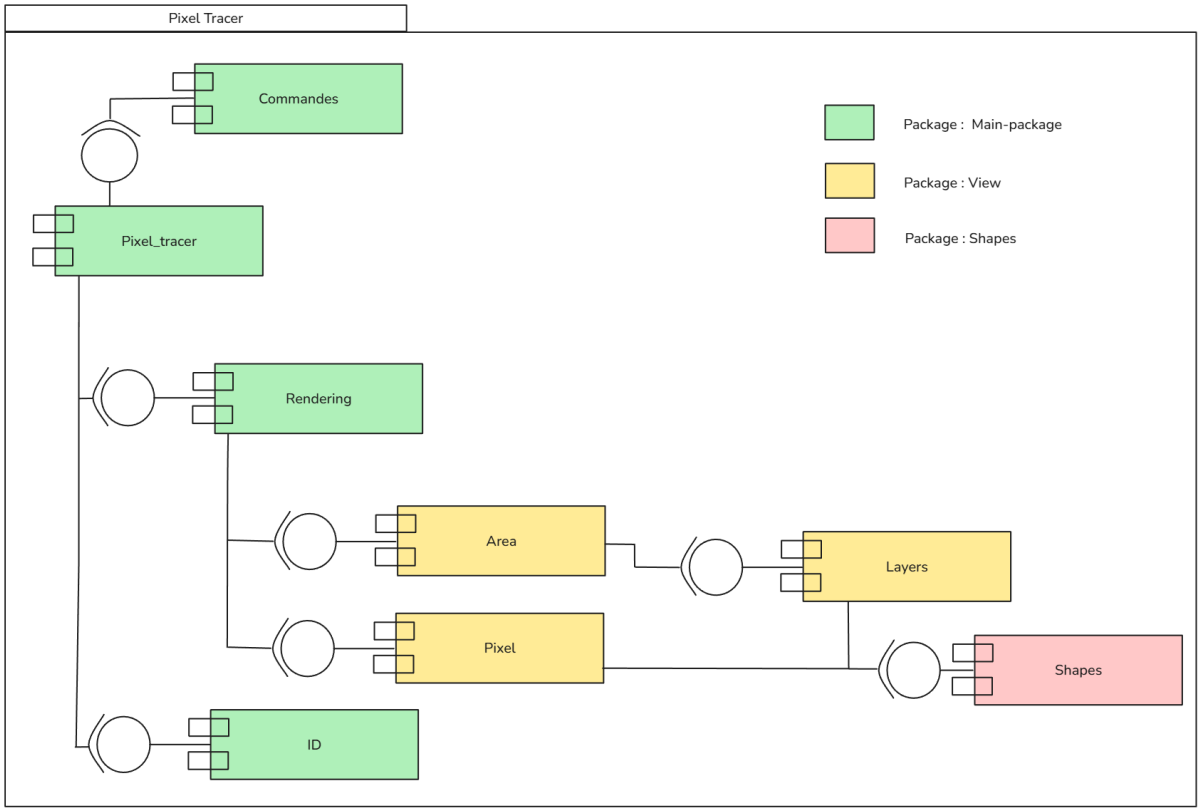

figure 2 diagramme de composant

2.3 Diagramme de classes

Afin de préparer la transition vers une approche orientée objet, nous avons réalisé un diagramme de classes représentant les principales structures de données du projet. Bien que l'application soit écrite en langage C et repose sur des structures de type `struct`, celles-ci peuvent être assimilées à des classes dans une représentation UML.

Chaque structure représente une entité importante du système, comme une forme géométrique, un calque ou encore une commande. Les champs des structures correspondent aux attributs de ces classes, tandis que les fonctions associées aux structures peuvent être vues comme des méthodes.

Ce diagramme permet de visualiser les relations entre les différentes structures, notamment les liens d'inclusion et les références par pointeur. Il met également en évidence l'organisation des données dans l'application et la manière dont les formes sont stockées et manipulées au sein des calques.

Cette représentation constitue une étape importante en vue de la transformation future du projet vers un modèle orienté objet lors des prochains travaux pratiques.

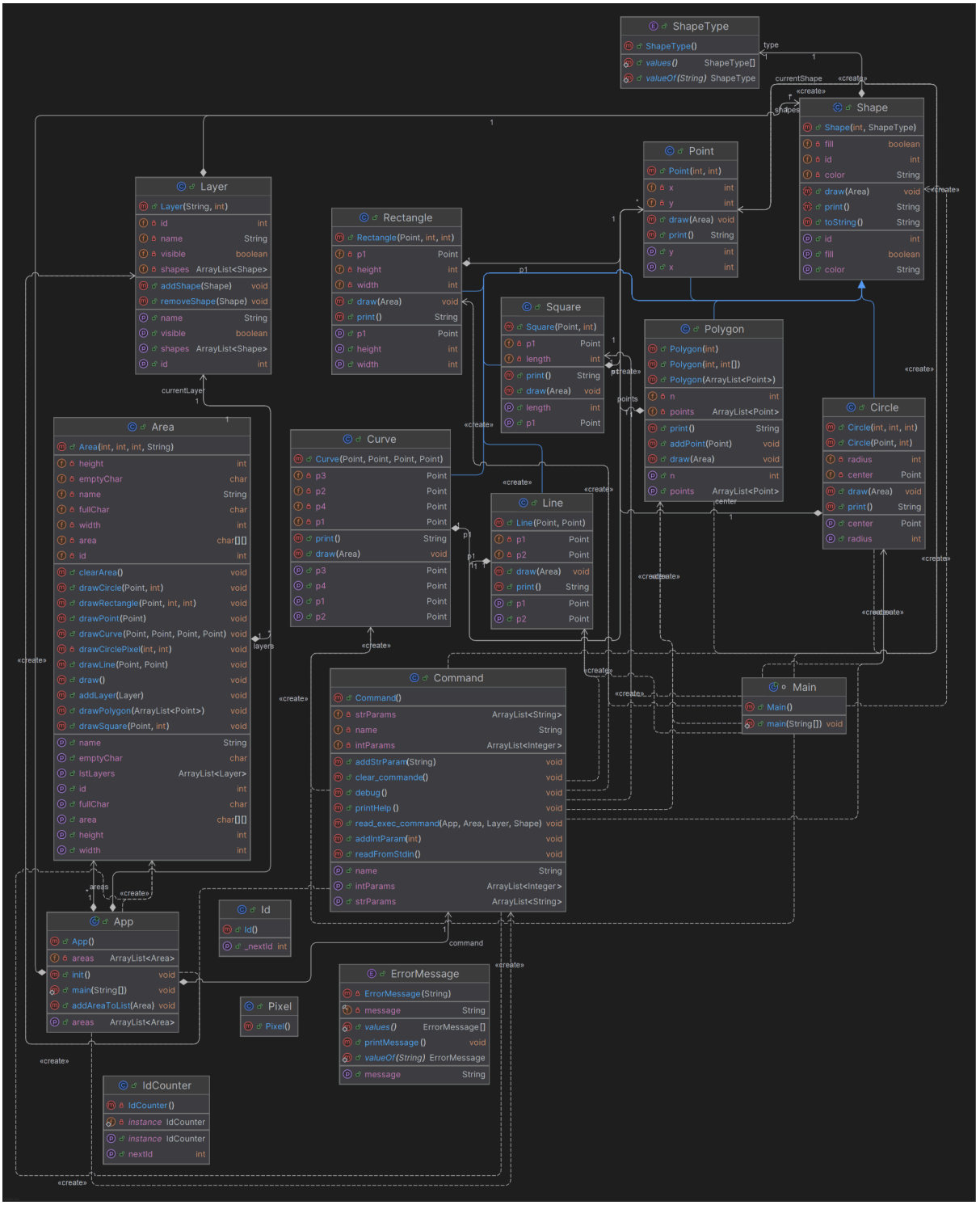

figure 3 diagramme de classes

2.4 Documentation des fonctions principales

L'analyse du code source a permis d'identifier un ensemble de fonctions essentielles au fonctionnement de l'application. Ces fonctions constituent l'interface principale de manipulation des formes géométriques et des structures internes du programme. Elles sont principalement définies dans le module shape.c et assurent la création et la destruction des différentes formes manipulées par l'utilisateur.

La fonction create_point permet de créer dynamiquement un point en mémoire à partir de coordonnées fournies en paramètre. Elle alloue la mémoire nécessaire pour une structure de type Point, initialise les coordonnées puis retourne un pointeur vers la structure créée. Cette fonction constitue une base pour la création de nombreuses autres formes, celles-ci reposant souvent sur des points.

La fonction delete_point est chargée de libérer la mémoire associée à un point précédemment créé. Elle prend en paramètre un pointeur vers une structure Point et libère l'espace mémoire correspondant. Cette fonction doit être appelée afin d'éviter toute fuite mémoire.

La fonction create_line permet de créer une ligne entre deux points. Elle reçoit en paramètre deux pointeurs vers des structures Point représentant les extrémités de la ligne. La fonction alloue ensuite une structure Line, initialise les pointeurs vers les points fournis et retourne la ligne créée.

La fonction delete_line assure la suppression d'une ligne. Elle libère d'abord les points constituant la ligne puis libère la mémoire associée à la structure Line. Cette fonction a donc pour effet de détruire l'ensemble des éléments composant la ligne.

La fonction create_rectangle permet de créer un rectangle à partir d'un point d'origine, d'une largeur et d'une hauteur. Elle alloue une structure Rectangle, initialise ses champs puis retourne le pointeur vers la structure nouvellement créée. Le point fourni en paramètre correspond généralement au coin supérieur gauche du rectangle.
La fonction delete_rectangle libère la mémoire associée à un rectangle. Elle supprime d'abord le point d'origine puis libère la structure du rectangle. Cette fonction est essentielle pour la gestion correcte de la mémoire.

La fonction create_cercle permet de créer un cercle à partir d'un point central et d'un rayon. Elle alloue dynamiquement une structure Cercle, initialise ses champs et retourne un pointeur vers cette structure. Cette fonction est utilisée lors de la création de formes circulaires dans l'application.

La fonction delete_cercle supprime un cercle en libérant la mémoire associée au point central puis celle de la structure du cercle. Elle doit être appelée lorsque la forme n'est plus utilisée.

La fonction create_polygon permet de créer un polygone vide contenant un nombre défini de sommets. Elle alloue une structure Polygon ainsi qu'un tableau de pointeurs vers des points.

Les positions des sommets sont initialisées à NULL afin de pouvoir être définies ultérieurement.

La fonction delete_polygon assure la suppression complète d'un polygone. Elle parcourt l'ensemble des sommets et libère chaque point, puis libère le tableau de points et la structure du polygone elle-même. Cette fonction garantit une libération correcte de toute la mémoire utilisée par la structure.

Enfin, la fonction create_curve permet de créer une courbe de Bézier définie par quatre points de contrôle. Elle alloue une structure Curve, initialise les pointeurs vers les points fournis et retourne la courbe créée. La fonction delete_curve libère quant à elle la mémoire associée à ces quatre points ainsi que celle de la structure de courbe.

L'ensemble de ces fonctions constitue le cœur de la manipulation des formes dans l'application. Elles illustrent une gestion manuelle de la mémoire typique du langage C et devront être adaptées lors de la transformation du projet vers un modèle orienté objet en langage Java.

2.5 Analyse des structures de données

L'étude du code source a permis d'identifier plusieurs structures de données fondamentales au fonctionnement de l'application. Ces structures représentent les entités principales manipulées par le programme et organisent les informations nécessaires au rendu des formes et à la gestion de la scène.

La structure pixel représente l'unité de base du rendu graphique. Elle contient les coordonnées du pixel ainsi qu'une information de couleur. Cette structure est utilisée lors du processus d'affichage pour représenter les éléments dessinés dans la zone de rendu.
La structure Area représente une zone de dessin complète. Elle contient un identifiant, un nom, des dimensions ainsi qu'une liste de calques. Elle stocke également les caractères utilisés pour représenter les pixels vides et remplis. Cette structure constitue le conteneur principal dans lequel sont organisées les formes affichées à l'écran.

La structure pixel_tracer représente l'état global de l'application. Elle contient la liste des zones de dessin disponibles, la zone actuellement sélectionnée, le calque actif et la forme en cours de manipulation. Elle permet de centraliser toutes les informations nécessaires au fonctionnement du programme.

Les structures de formes géométriques telles que Point, Line, Rectangle, Cercle, Polygon et Curve représentent les différentes entités graphiques manipulées par l'utilisateur. Chacune de ces structures contient les informations nécessaires à la définition de la forme, comme les coordonnées des points, les dimensions ou les rayons.

Ces structures sont souvent reliées entre elles par des pointeurs. Par exemple, une ligne est définie par deux points, tandis qu'un polygone contient un tableau de pointeurs vers plusieurs points. Cette organisation permet de construire des structures complexes tout en conservant une certaine modularité.

Le cycle de vie de ces structures suit généralement le même schéma : création dynamique en mémoire, utilisation dans la scène puis suppression lorsque l'objet n'est plus nécessaire. Une gestion rigoureuse de la mémoire est donc indispensable afin d'éviter les fuites mémoire et les erreurs d'accès.

Conclusion

Ce travail de rétro-conception a permis d'analyser en profondeur le fonctionnement de l'application Pixel Tracer. L'étude de la structure du projet, des fonctions principales et des structures de données a permis de mieux comprendre l'architecture générale du programme ainsi que les mécanismes utilisés pour manipuler et afficher les formes géométriques.

La réalisation des diagrammes techniques et de la documentation a permis de mettre en évidence les interactions entre les différents modules et de clarifier le rôle de chaque composant. Cette phase de redocumentation constitue une étape essentielle dans un processus de maintenance applicative, en particulier lorsqu'il s'agit de reprendre un projet existant dont la documentation est incomplète.

Certaines difficultés ont été rencontrées lors de l'analyse du code, notamment en raison de l'absence de commentaires détaillés dans certaines parties du projet et de la gestion manuelle de la mémoire. Toutefois, une lecture progressive du code et la réalisation de schémas ont permis de surmonter ces obstacles.

Ce travail constitue une base solide pour les prochaines étapes du projet, qui consisterait à transformer le modèle vers une approche orientée objet puis à réécrire l'application en langage Java. La compréhension acquise lors de cette phase facilitera grandement le refactoring et l'amélioration de l'architecture du programme.