бюджетное профессиональное образовательное учреждение Вологодской области
«Череповецкий лесомеханический техникум им. В.П. Чкалова»

Специальность **09.02.07** «Информационные системы и программирование»

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ
ПП по ПМ.03 РЕВЬЮИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ

Выполнил студент 3 курса группы ИС-____
_____
подпись _____
место практики _____
наименование юридического лица, ФИО ИП

Период прохождения:
с «___» _____ 2024 г.
по «___» _____ 2024 г.

Руководитель практики от
предприятия
должность_____
_____
подпись_____

Руководитель практики от
техникума: Материкова А.А.
_____
_
Оценка:_____
_
«___» _____2024 года

МП

г. Череповец
2024

«Содержание»

Введение:

1. Общая характеристика предприятия:

ООО "Малленом Системс" — российская компания, специализирующаяся на разработке и внедрении систем машинного зрения и видеоаналитики для промышленных и транспортных предприятий. Созданная в 2011 году на основе команды из Санкт-Петербургского политехнического университета, компания развивает решения на базе технологий искусственного интеллекта и машинного обучения, в том числе для анализа изображений и обработки данных в реальном времени. Основные направления деятельности компании включают разработку систем контроля качества продукции, отслеживания и идентификации товаров, а также решения для промышленного и транспортного секторов. К примеру, для железнодорожной отрасли компания предлагает систему распознавания номеров вагонов, что способствует автоматизации учёта и контроля на станцияхКомпания активно разрабатывает и адаптирует программные и аппаратные решения для различных отраслей, включая металлургию, фармацевтику и логистику. Она создала и внедрила систему ВИСКОНТ.Фарма, которая выполняет задачи сериализации и агрегации лекарственных средств для отслеживания их оборота. Эта система интегрируется с ERP-платформами и позволяет отслеживать продукцию по всему логистическому циклу, что особенно актуально в фармацевтической отрасли. По структуре "Малленом Системс" включает отделы научных исследований и разработки, где трудится команда из более чем 80 специалистов, включая доктора и кандидатов наук.

1.1.Организационная структура предприятия:

Организационная структура компании включает исследовательские и проектные отделы, которые работают над задачами в сфере машинного зрения и аналитики. В штате компании более 80 сотрудников, включая экспертов с учеными степенями, что позволяет ей успешно решать комплексные научно-технические задачи. Внутренняя структура также

обеспечивает гибкость в управлении проектами для удовлетворения специфических потребностей клиентов в различных отраслях

1.2. Внутренний распорядок работы предприятия, охрана труда на предприятии:

У предприятия есть 2 офиса

Ул. Металлургов 21Б

Офис работает:

Пн-Пт. с 8:00 до 20:00

Сб-Вс. с 10:00 до 17:00

Ул. Ленина 110Б

Офис работает:

Пн-Пт. с 8:00 до 18:00

Рабочий день

Полная ставка

Продолжительность рабочего времени составляет 40 часов в неделю

Два выходных дня –суббота и воскресенье

Неполная ставка

Продолжительность рабочего времени определяется долей ставки:

0,25 –10 часов в неделю

0,3 –12 часов в неделю

0,5 –20часов в неделю

Выходные такие же –суббота и воскресенье

В компании есть общепринятый режим работы для большинства сотрудников, работающих на полную ставку:

рабочее время с 9.00 до 18.00

обед с 13.00 до 14.00

Два технологических перерыва по 20 минут в течение дня

Режим работы может быть установлен для работника индивидуально, по согласованию с руководителем, но при условии отработки нормы рабочего времени за неделю.

1.3. Должностные инструкции ИТ-специалистов предприятия:

Основные должности в компании:

Инженер-программист

разработка приложений под ОС Windows;
интеграция с алгоритмами машинного обучения;
программирование UI;
реализация алгоритмов машинного зрения;
доработка существующих проектов;
оптимизация и рефакторинг.

Специалист по машинному обучению

дообучение / улучшение существующих нейросетей, используемых в production;
создание и обучение нейросетей;
анализ современных моделей на применимость их бизнес-задачам компании;
визуализация данных;
работа с датасетами.

Инженер

проработка и согласование технических заданий по проектам;
подбор оборудования и комплектующих, разработка спецификаций;
подготовка оборудования к инсталляции;

выполнение проектно-изыскательских работ;
выполнение пусконаладочных работ на объектах внедрения (служебные командировки);
обучение операционного персонала Заказчика;
техническая поддержка клиентов;
разработка технической документации

Специалист по тестированию ПО

ручное тестирование;
составление тестовых сценариев;
поддержка и расширение документации по продуктам проекта;
документирование и верификация дефектов, контроль исправления выявленных ошибок разработчиком;
взаимодействие с командой разработки и технической поддержки;
тестирование продуктов проекта;
актуализация документации по продуктам проекта.

Менеджер по продажам

Обработка входящих запросов от клиентов.
Ведение коммерческих переговоров с клиентами, консультирование о продуктах Малленом Системс для транспортной отрасли.
Подготовка ТКП (совместно с техническими специалистами), согласование конфигурации продукции под каждую задачу, подбор оборудования под проект.
Заключение договоров (совместно с юристом) и их сопровождение.
Контроль работы по отгрузке и доставке товаров покупателям по заключенным договорам, подготовка товара к отправке.
Контроль оплаты договоров клиентами.
Участие в торгах на поставку продукции Компании на торговых площадках
Ведение информационных баз клиентов и партнеров, документооборота.

2. Ревьюирование программных продуктов:

2.1 Ревьюирование программного кода в соответствии с технической документацией:

1. Диаграмма компонентов

Диаграмма компонентов показывает основные модули системы и их зависимости.

Описание компонентов:

- main.py: Главный модуль, запускает приложение.
- main_window.py: Определяет пользовательский интерфейс (GUI).
- image_processing.py: Предоставляет функции обработки изображений.
- file_utils.py: Реализует утилиты для работы с файлами.
- PyQt5: Внешняя библиотека для создания GUI.
- Pillow (PIL): Внешняя библиотека для обработки изображений.

Связи компонентов:

- main.py зависит от main_window.py.
- main_window.py зависит от image_processing.py и file_utils.py.
- image_processing.py использует Pillow.
- main_window.py зависит от PyQt5.

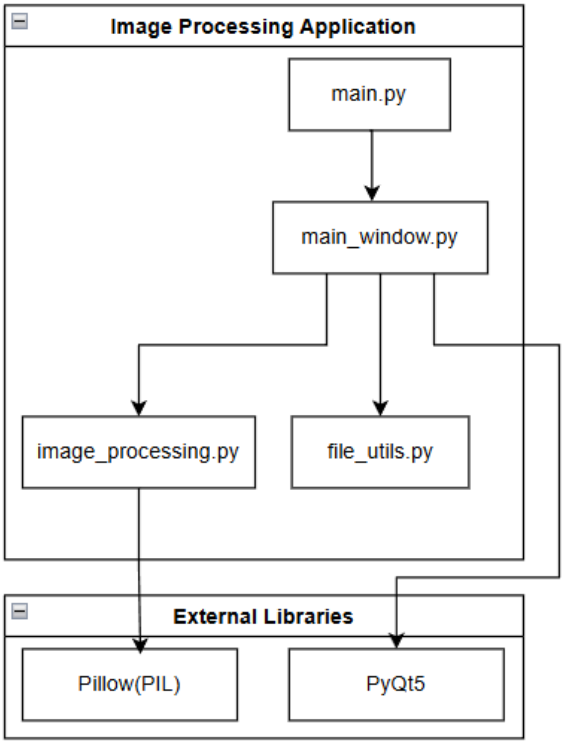

Рисунок 1: Диаграмма компонентов

2. Диаграмма сценариев использования

Показывает основные взаимодействия пользователя с системой.

Основные сценарии:

1. Выбрать изображение:
    - Пользователь вводит путь или выбирает изображение через диалоговое окно.
    - Программа отображает изображение.
2. Конвертация в градации серого:
    - Пользователь нажимает кнопку "Convert to Grayscale".
    - Программа вызывает функцию обработки изображения и отображает результат.
3. Перемещение изображения:
    - Пользователь нажимает кнопку "Move Image".
    - Программа перемещает файл в указанную папку.

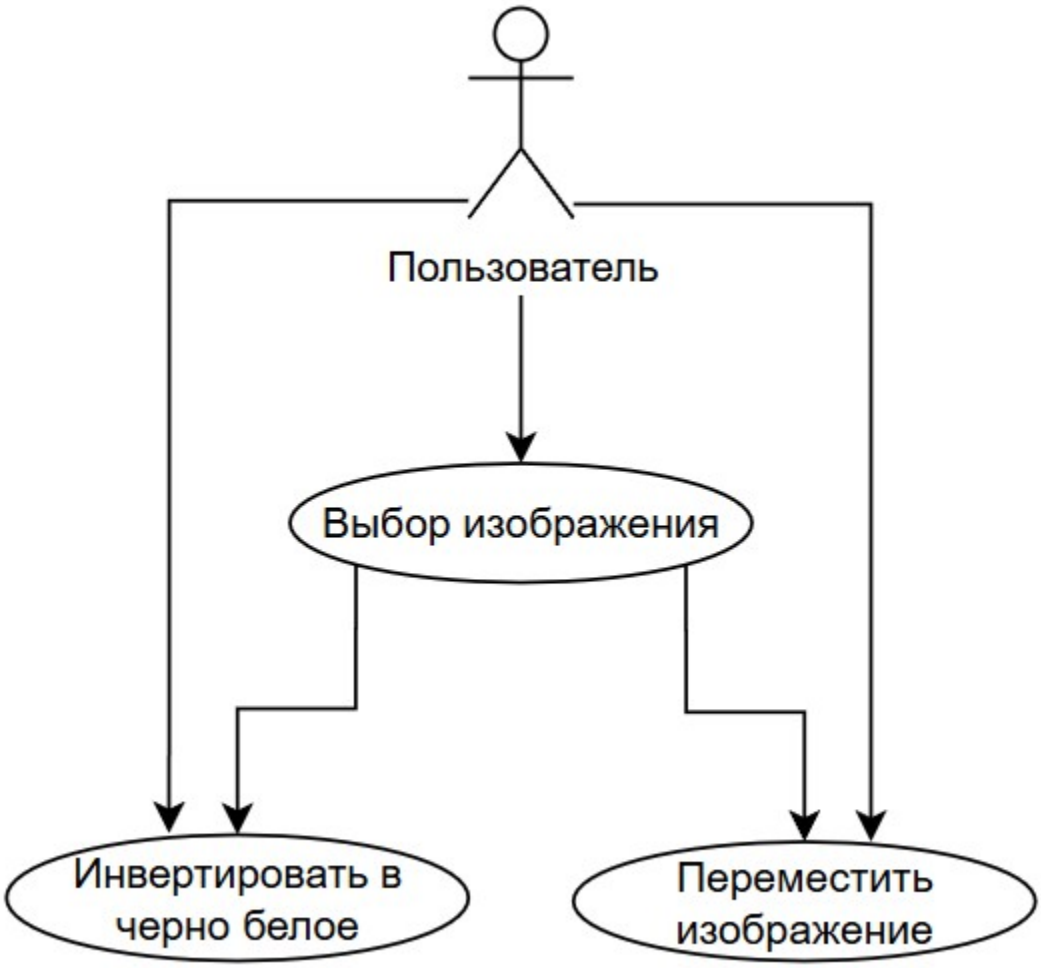

Рисунок 2: Диаграмма сценариев использования

3. Диаграмма последовательности

Отражает последовательность вызовов для сценария "Конвертация в градации серого":

1. Пользователь нажимает кнопку "Convert to Grayscale".
2. MainWindow.handle_grayscale():
    - Проверяет корректность пути.
    - Вызывает convert_to_grayscale(image_path).
3. convert_to_grayscale():
    - Загружает изображение.
    - Конвертирует в градации серого.

- Сохраняет новое изображение.

4. MainWindow.display_image():
    - Отображает преобразованное изображение.
5. Выводит сообщение об успешной операции.

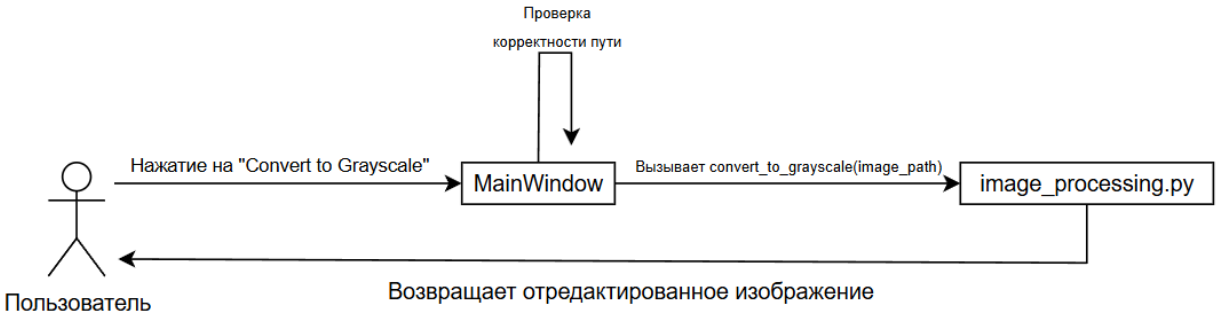

Рисунок 3: Диаграмма последовательности

4. Диаграмма деятельности

Показывает общий процесс для сценария "Перемещение изображения":

1. Начало.
2. Пользователь нажимает "Move Image".
3. Открывается диалог выбора директории.
4. Программа проверяет:
    - Выбран файл.
    - Директория указана.
5. Если условия выполнены:
    - Перемещает файл.
    - Выводит сообщение об успешной операции.
6. Если условия не выполнены:
    - Выводит предупреждение.
7. Конец.

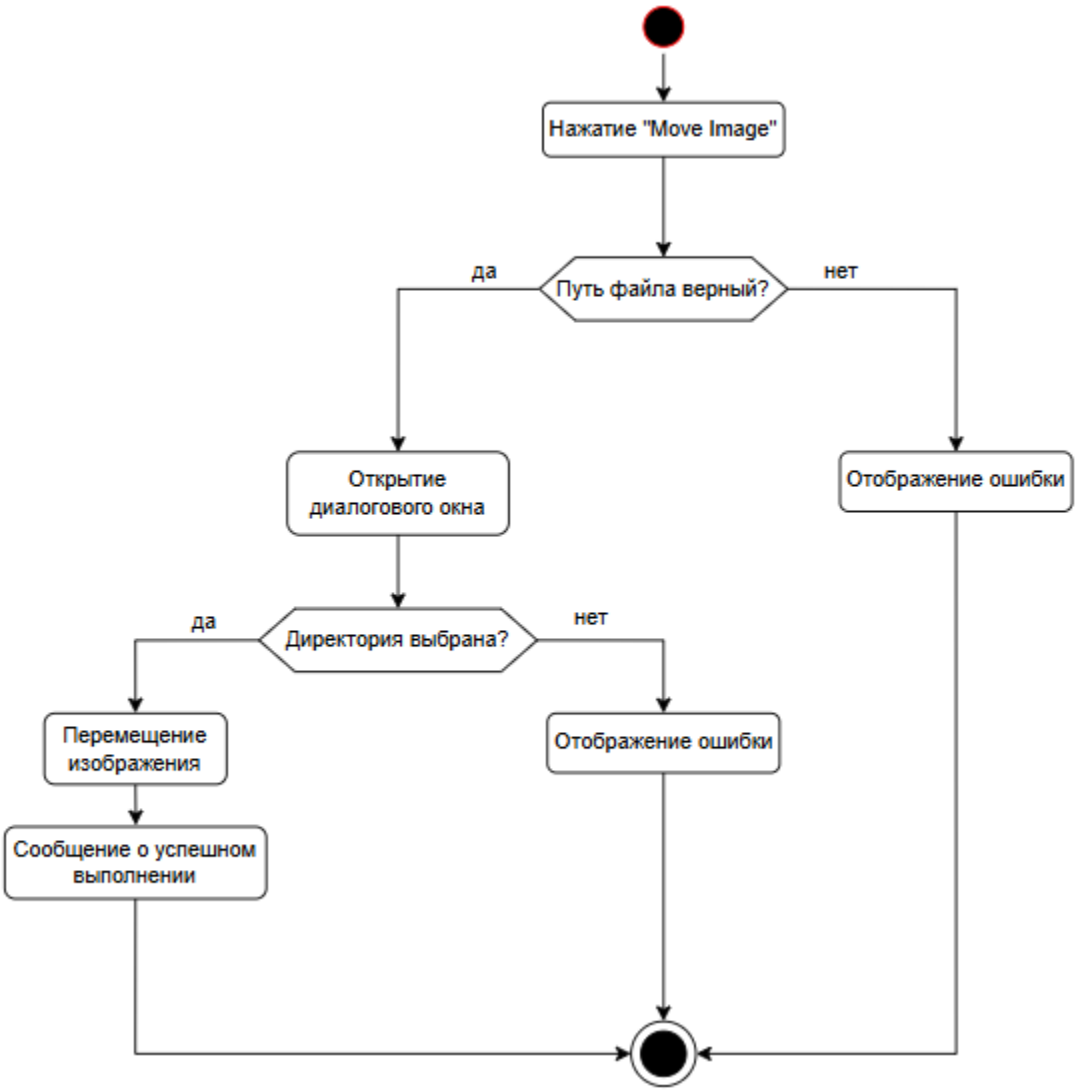

Рисунок 4: Диаграмма деятельности

2.2. Измерение характеристик компонент программного продукта:

Скоростные показатели программы:

GUI Initialization Time: 0.0061 seconds

Image Processing Time: 0.0032 seconds

File Moving Time: 0.0003 seconds

Total Time: 0.0097 seconds

Размеры программы:

file_utils.py — 195 байт

image_processing.py — 252 байта

main.py — 252 байта

main_window.py — 3,36 КБ

2.3. Исследование созданного программного кода с использованием специализированных программных средств:

1. Статический анализ кода
Цель: Выявление ошибок, нарушение стандартов кодирования и возможных улучшений.

Инструменты:

- pylint: Проверка качества кода.
- flake8: Анализ стиля и синтаксиса.
- mypy: Проверка аннотаций типов.

2. Динамический анализ

Цель: Изучение производительности, использования памяти и обработки исключений.

Инструменты:

- memory_profiler: Анализ памяти.
- timeit: Измерение времени выполнения функций.
- pytest: Тестирование функциональности.

3. Визуализация структуры вызовов

Цель: Построить граф вызовов для анализа взаимодействия между модулями.

Инструмент:

- pycallgraph: Построение графов вызовов.

2.4. Сравнительный анализ программных продуктов и средств разработки:

1. PyQt5 (GUI - main_window.py)

PyQt5 — это мощный фреймворк для создания графических интерфейсов.

Возможности:

- Широкий функционал: Поддерживает создание сложных интерфейсов с множеством виджетов (кнопки, текстовые поля, окна диалогов и т.д.).
- Кроссплатформенность: Работает на Windows, macOS и Linux.
- Поддержка стилей: Можно настраивать внешний вид интерфейса через CSS или встроенные темы.
- Интеграция с Python: Позволяет легко связывать интерфейс с логикой программы.

Недостатки:

- Сложность: Более громоздкий синтаксис по сравнению с другими GUI-библиотеками, такими как Tkinter.
- Зависимость от Qt: Большой объем библиотеки увеличивает размер программы.
- Лицензия: Для коммерческого использования требуется приобрести лицензию.

Альтернативы:

1. Tkinter:
    - Простая встроенная библиотека Python.
    - Подходит для небольших проектов.
    - Менее мощная по сравнению с PyQt5.
2. Kivy:
    - Подходит для кроссплатформенной разработки, включая мобильные устройства.
    - Имеет интуитивный дизайн, но меньше возможностей для сложных интерфейсов.
3. PySide:

- Аналог PyQt5 с лицензией LGPL (подходит для коммерческого использования).

2. Pillow (PIL) (Обработка изображений - image_processing.py)

Pillow — это мощная библиотека для работы с изображениями.

Возможности:

- Широкий спектр инструментов: Поддержка различных форматов изображений, возможность изменения размеров, применения фильтров, преобразования в градации серого и т.д.
- Простота использования: Легко интегрируется в проекты.
- Кроссплатформенность: Работает на всех популярных ОС.

Недостатки:

- Ограничения в обработке: Не подходит для высокопроизводительных задач по сравнению с библиотеками на C++ (например, OpenCV).
- Поддержка форматов: Хотя поддерживает многие форматы, для специализированных задач (например, DICOM) требуются дополнительные библиотеки.

Альтернативы:

1. OpenCV:
    - Более производительная библиотека для обработки изображений и компьютерного зрения.
    - Подходит для сложных задач, таких как распознавание лиц или обработки видео.
2. ImageIO:
    - Легковесная библиотека для чтения и записи изображений.
    - Подходит для базовой обработки.

3. Стандартная библиотека Python (Работа с файлами - file_utils.py)

Python предлагает мощные встроенные модули для работы с файлами и каталогами.

Возможности:

- Модули os и shutil: Позволяют работать с файловой системой, копировать, перемещать, удалять файлы и директории.
- Простота интеграции: Не требуют установки сторонних библиотек.
- Поддержка всех ОС.

Недостатки:

- Отсутствие высокоуровневых функций: Нет инструментов для работы с облачными хранилищами или специальными файловыми форматами.
- Неинтуитивный интерфейс: Работа с путями может быть сложной без дополнительных библиотек, таких как pathlib.

Альтернативы:

1. pathlib (Python 3.4+):
    - Современный способ работы с путями файловой системы.
    - Интуитивно понятный и удобный синтаксис.
2. watchdog:
    - Подходит для мониторинга изменений в файловой системе в реальном времени.
3. HDFS/S3 SDKs:
    - Для работы с распределенными хранилищами данных (например, Hadoop или Amazon S3).

3. Выполныемые задания:

main.py

```python
import sys
from PyQt5.QtWidgets import QApplication
from main_window import MainWindow

def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```

Рисунок 5: Код модуля main.py

main_window.py

```python
from PyQt5.QtWidgets import QWidget, QLabel, QPushButton, QFileDialog, QVBoxLayout, QHBoxLayout, QLineEdit, QMessageBox
from PyQt5.QtGui import QPixmap
from PyQt5.QtCore import Qt
from image_processing import convert_to_grayscale
from file_utils import move_file
import os

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Image Processor")
        self.setFixedSize(1200, 900)

        main_layout = QHBoxLayout()

        left_layout = QVBoxLayout()
        left_layout.setContentsMargins(20, 20, 20, 20)
        left_layout.setSpacing(15)

        self.path_input = QLineEdit()
        self.path_input.setPlaceholderText("Enter image path here")
        self.path_input.setFixedWidth(300)
        left_layout.addWidget(self.path_input)

        self.browse_button = QPushButton("Browse")
        self.browse_button.setFixedWidth(300)
        self.browse_button.clicked.connect(self.browse_image)
        left_layout.addWidget(self.browse_button)

        self.process_button = QPushButton("Convert to Grayscale")
        self.process_button.setFixedWidth(300)
        self.process_button.clicked.connect(self.handle_grayscale)
        left_layout.addWidget(self.process_button)

        self.move_button = QPushButton("Move Image")
        self.move_button.setFixedWidth(300)
        self.move_button.clicked.connect(self.handle_move)
        left_layout.addWidget(self.move_button)

        left_layout.addStretch()

        self.image_label = QLabel()
        self.image_label.setFixedSize(800, 800)
        self.image_label.setAlignment(Qt.AlignCenter)

        main_layout.addLayout(left_layout)
        main_layout.addWidget(self.image_label)

```

Рисунок 6: Код модуля main_window.py

image_processing.py:

```
        self.setLayout(main_layout)

    def browse_image(self):
        file_path, _ = QFileDialog.getOpenFileName(self, "Open Image", "", "Images (*.png *.xpm *.jpg)")
        if file_path:
            self.path_input.setText(file_path)
            self.display_image(file_path)

    def display_image(self, image_path):
        pixmap = QPixmap(image_path)
        self.image_label.setPixmap(pixmap.scaled(self.image_label.size(), Qt.KeepAspectRatio))

    def handle_grayscale(self):
        image_path = self.path_input.text()
        if not image_path or not os.path.exists(image_path):
            QMessageBox.warning(self, "Warning", "Please select a valid image file.")
            return
        try:
            grayscale_path = convert_to_grayscale(image_path)
            self.display_image(grayscale_path)
            QMessageBox.information(self, "Success", f"Grayscale image saved as {grayscale_path}")
        except Exception as e:
            QMessageBox.critical(self, "Error", f"Failed to convert image: {e}")

    def handle_move(self):
        image_path = self.path_input.text()
        if not image_path or not os.path.exists(image_path):
            QMessageBox.warning(self, "Warning", "Please select a valid image file.")
            return
        new_dir = QFileDialog.getExistingDirectory(self, "Select Destination Folder")
        if new_dir:
            try:
                new_path = move_file(image_path, new_dir)
                QMessageBox.information(self, "Success", f"Image moved to {new_path}")
            except Exception as e:
                QMessageBox.critical(self, "Error", f"Could not move image: {e}")
```

Рисунок 7: Код модуля image_processing.py

file_utils.py:

```
import os


def move_file(source_path, destination_dir):
    new_path = os.path.join(destination_dir, os.path.basename(source_path))
    os.rename(source_path, new_path)
    return new_path
```

Рисунок 8: Код модуля file_utils.py

Заключение

Практика в ООО "Малленом Системс" позволила мне улучшить навыки анализа и разработки программного обеспечения,

а также получить опыт работы с современными инструментами и библиотеками Python. Работа над проектом помогла

мне лучше понять процессы проектирования, тестирования и оптимизации программных продуктов.

В ходе производственной практики мной был проведен всесторонний анализ программных продуктов и их компонентов.

Работа включала ревьюирование кода, измерение характеристик производительности,

использование инструментов анализа и сравнительный обзор применяемых технологий.

Список литературы:

UML - https://practicum.yandex.ru/blog/uml-diagrammy/

Пример измерения скорости используя Time -

https://www.geeksforgeeks.org/how-to-check-the-execution-time-of-python

Диаграммы - https://app.diagrams.net/

Работа с модулями Python - https://metanit.com/python/tutorial/2.10.php

Работа с библиотеками https://metanit.com/sharp/tutorial/3.46.php

Приложения

Вид программы:

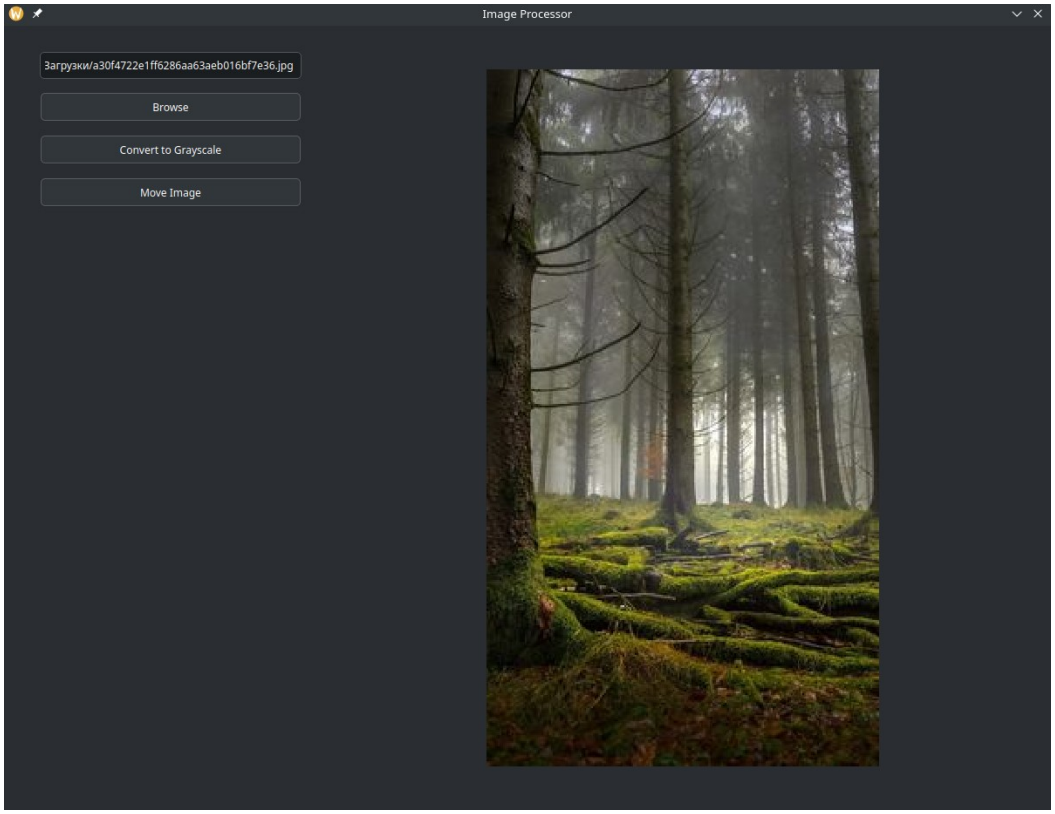

Рисунок 9: Внешний вид программы

Код тестировщика:

```python
import time
from PyQt5.QtWidgets import QApplication
from main_window import MainWindow
from image_processing import convert_to_grayscale
from file_utils import move_file
import os
import shutil

# Тестовые данные
TEST_IMAGE_PATH = "test_image.jpg"
TEST_DIR = "test_dir"
GRAYSCALE_IMAGE_PATH = "test_image_grayscale.png"

# Создание тестовых файлов и директорий
def setup_environment():
    if not os.path.exists(TEST_IMAGE_PATH):
        # Создаем пустое изображение, если его нет
        from PIL import Image
        img = Image.new('RGB', (100, 100), color='white')
        img.save(TEST_IMAGE_PATH)

    if not os.path.exists(TEST_DIR):
        os.mkdir(TEST_DIR)

# Удаление тестовой среды
def cleanup_environment():
    if os.path.exists(TEST_IMAGE_PATH):
        os.remove(TEST_IMAGE_PATH)
    if os.path.exists(GRAYSCALE_IMAGE_PATH):
        os.remove(GRAYSCALE_IMAGE_PATH)
    if os.path.exists(TEST_DIR):
        shutil.rmtree(TEST_DIR)

```

Рисунок 10: Код тестировщика

```python
# Замеры выполнения
def benchmark():
    setup_environment()
    try:
        # 1. Инициализация QApplication
        app = QApplication([])

        # 2. Измеряем время создания окна
        start_time = time.time()
        window = MainWindow()
        gui_time = time.time() - start_time

        # 3. Измеряем время конвертации в градации серого
        start_time = time.time()
        convert_to_grayscale(TEST_IMAGE_PATH)
        processing_time = time.time() - start_time

        # 4. Измеряем время перемещения файла
        start_time = time.time()
        move_file(TEST_IMAGE_PATH, TEST_DIR)
        move_time = time.time() - start_time

        # 5. Суммарное время
        total_time = gui_time + processing_time + move_time

        # Вывод результатов
        print(f"GUI Initialization Time: {gui_time:.4f} seconds")
        print(f"Image Processing Time: {processing_time:.4f} seconds")
        print(f"File Moving Time: {move_time:.4f} seconds")
        print(f"Total Time: {total_time:.4f} seconds")

    finally:
        cleanup_environment()

# Запуск замеров
if __name__ == "__main__":
    benchmark()

```

Рисунок 11: Код тестировщика