Geekbrains

Разработка и реализация программного обеспечения в Python для автоматизированного проектирования технологических процессов изготовления деталей штампов на основе существующей трехмерной модели

IT-специалист:
Программист Python
Макеев Н.Ю.

Брянск 2024

СОДЕРЖАНИЕ

1.1.Актуальность выбранной темы.

Разработка и реализация программного обеспечения на языке Python для автоматизированного проектирования технологических процессов изготовления деталей штампов на основе существующей трехмерной модели имеет большую актуальность в современной промышленности. Вот несколько причин, почему такое ПО является важным:

1. Увеличение производительности: Автоматизация проектирования и создания технологических процессов позволяет существенно сократить время, необходимое для подготовки производства. Это позволяет компаниям быстрее выпускать новые изделия и улучшать их конкурентоспособность.

2. Снижение ошибок: Программное обеспечение для автоматизированного проектирования может предотвращать ошибки и несоответствие в процессе производства, что снижает количество брака и улучшает качество продукции.

3. Оптимизация ресурсов: Автоматическое планирование и оптимизация технологических процессов позволяют оптимально использовать оборудование, сырье и рабочую силу, что снижает затраты и повышает эффективность производства.

4. Адаптация к изменениям: С помощью ПО на Python можно легко адаптировать технологические процессы к изменениям в конструкции изделий или требованиям заказчика, что делает производство более гибким и отзывчивым.

5. Визуализация и анализ: Трехмерные модели и визуализация процессов позволяют более наглядно представлять информацию и проводить

анализ производственных данных для принятия более обоснованных решений.

В итоге, разработка программного обеспечения на Python для автоматизированного проектирования технологических процессов изготовления деталей штампов является важным инструментом для улучшения эффективности и качества производства, а также для повышения конкурентоспособности компаний в современной промышленности.

1.2.Цель и задачи исследования

Цели исследования:

1. Разработка программного обеспечения для автоматизации процесса проектирования технологических процессов изготовления деталей штампов.

2. Улучшение эффективности и точности проектирования, сокращение времени и затрат на разработку технологических процессов.

3. Повышение качества изготавливаемых деталей штампов и снижение вероятности ошибок в процессе изготовления.

4. Обеспечение автоматической совместимости программного обеспечения с трехмерными моделями деталей.

Задачи исследования:

1. Анализ требований и потребностей пользователей для определения основных функциональных возможностей программного обеспечения.

2. Разработка алгоритмов и методов, позволяющих автоматически создавать технологические процессы на основе трехмерных моделей деталей.

3. Реализация интерфейса пользователя, обеспечивающего удобное взаимодействие с программным обеспечением.

4. Интеграция программного обеспечения с существующими CAD-системами и форматами трехмерных моделей.

5. Тестирование и отладка программного обеспечения для обеспечения его стабильной и надежной работы.

6. Обеспечение документации и обучения пользователей работе с программным обеспечением.

7. Оценка эффективности программного обеспечения на практике, сравнение с традиционными методами проектирования технологических процессов.

8. Внесение необходимых корректировок и улучшений на основе обратной связи от пользователей.

1.3.Обзор существующих подходов к автоматизации проектирования технологических процессов.

Существует несколько подходов к автоматизации проектирования технологических процессов. Вот некоторые из них:

1. CAD/CAM (Computer-Aided Design/Computer-Aided Manufacturing): Этот подход использует компьютерное моделирование и программное обеспечение для разработки и оптимизации деталей и процессов

производства. CAD помогает создавать трехмерные модели продуктов, а CAM управляет машинами и оборудованием для их изготовления.

2. PLM (Product Lifecycle Management): PLM-системы позволяют управлять жизненным циклом продукта, включая проектирование, производство, тестирование и обслуживание. Они обеспечивают централизованное хранение данных и совместное взаимодействие между разными отделами.

3. Программирование на основе правил: Этот подход включает в себя создание набора правил и параметров, которые автоматически определяют процессы и действия на производстве. Это может включать в себя автоматизацию контроля качества, решение задач расписания и многие другие процессы.

4. Машинное обучение и искусственный интеллект: С использованием алгоритмов машинного обучения и искусственного интеллекта можно автоматизировать множество аспектов проектирования и управления технологическими процессами, включая прогнозирование сбоев оборудования и оптимизацию производственных процессов.

5. Автоматизация роботов и автоматизированные системы управления производством: Использование роботов и автоматизированных систем для выполнения задач на производстве, таких как сборка, погрузка/разгрузка, обработка материалов и т. д., помогает повысить производительность и эффективность.

Эти подходы могут комбинироваться и настраиваться в зависимости от конкретных потребностей и характеристик производственного процесса. Автоматизация технологических процессов играет важную роль в

современной промышленности, позволяя увеличить производительность, снизить ошибки и сократить затраты.

2. Литературный обзор

2.1. Использование Python в разработке программного обеспечения при автоматизации проектирования технологических процессов.

Python является мощным и гибким языком программирования, который широко используется в различных областях, включая автоматизацию проектирования технологических процессов. Вот несколько способов использования Python в этой области:

1. Моделирование и симуляция: С помощью Python можно создавать модели и симулировать технологические процессы, позволяя изучать и оптимизировать их до фактической реализации. Возможно использовать библиотеки, такие как NumPy и SciPy, для выполнения сложных математических и статистических расчетов.

2. Автоматизация процессов: Python предлагает богатый набор инструментов для автоматизации различных задач, связанных с проектированием технологических процессов. Можно создавать скрипты, которые выполняют повторяющиеся операции, обрабатывают данные и генерируют отчеты, что помогает сэкономить время и улучшить эффективность работы.

3. Взаимодействие с другими программными инструментами: Python обладает большим количеством библиотек и фреймворков, которые позволяют взаимодействовать с другими программными инструментами, используемыми в проектировании технологических процессов. Например, использование

библиотеки OpenCV для обработки изображений или библиотеки Pandas для анализа данных.

4. Разработка пользовательского интерфейса: если требуется создать пользовательский интерфейс для проектов по автоматизации проектирования технологических процессов, то Python предоставляет различные инструменты для этого, такие как библиотеки PyQt и Tkinter. Создавание графических интерфейсов, которые облегчают работу пользователя и делают процесс более интуитивно понятным.

Не существует ограничений для использования Python в разработке программного обеспечения при автоматизации проектирования технологических процессов. Креативность и гибкость языка позволяют решать самые разнообразные задачи, улучшая эффективность и результаты работы.

2.2. Обзор существующих инструментов и программных решений для проектирования технологических процессов деталей штампов.

Ниже представлены некоторые из популярных программных решений в этой области:

1. AutoCAD: AutoCAD - это одно из самых популярных программных решений для проектирования в области машиностроения. Он предоставляет широкие возможности для создания 2D и 3D моделей деталей.

2. SolidWorks: SolidWorks - это интегрированная система проектирования, которая позволяет создавать и анализировать 3D модели для производства штампованных деталей.

3. Siemens NX: Siemens NX - это продвинутое программное обеспечение для проектирования, моделирования и разработки инструментов для производства деталей штампов. Оно обладает широкими возможностями для разработки технологических карт и оптимизации процессов станочной обработки.

4. CATIA: CATIA - это интегрированная система проектирования и разработки, разработанная специально для инженеров и дизайнеров в области машиностроения. Она предоставляет широкий набор инструментов для создания и анализа 3D моделей, а также разработки технологических процессов для производства деталей штампов.

5. CAMWorks: CAMWorks - это программное обеспечение для компьютерного машинного программирования, которое интегрируется с САПР системами, такими как SolidWorks и Autodesk Inventor. Оно позволяет автоматизировать процесс разработки технологической программы ЧПУ (числового программного управления) для штамповки деталей.

3. Анализ требований и проектирование.

3.1. Сбор требований к разрабатываемому программному обеспечению.

1. Импорт модели:

Программа должна иметь возможность импортировать трехмерную модель детали штампа из различных форматов, таких как STL, STEP, IGES и другие.

2. Анализ модели:

ПО должно проводить анализ импортированной модели детали штампа, чтобы определить ее характеристики, такие как размеры, форма, материал и т.д.

3. Генерация технологического процесса:

Программа должна автоматически генерировать технологический процесс изготовления детали штампа на основе импортированной модели. Это может включать в себя операции фрезерования, сверления, гальваники и т.д.

4. Оптимизация процесса:

ПО должно иметь возможность оптимизировать технологический процесс изготовления детали штампа для повышения производительности, снижения затрат и улучшения качества.

5. Визуализация процесса:

Программа должна предоставлять визуализацию технологического процесса, позволяющую пользователям легко понять последовательность операций и взаимодействие с деталью штампа.

6. Экспорт данных:

ПО должно предоставлять возможность экспортировать данные технологического процесса, например, в формате Excel, чтобы их можно было использовать в других системах или аналитических инструментах.

7. Интеграция с CAD-системами:

Программа должна поддерживать возможность интеграции с различными CAD-системами, чтобы пользователи могли импортировать и экспортировать модели и данные между ними.

8. Поддержка составных деталей:

ПО должно поддерживать возможность работы с составными деталями, позволяя пользователю определить отдельные элементы и операции для каждого из них.

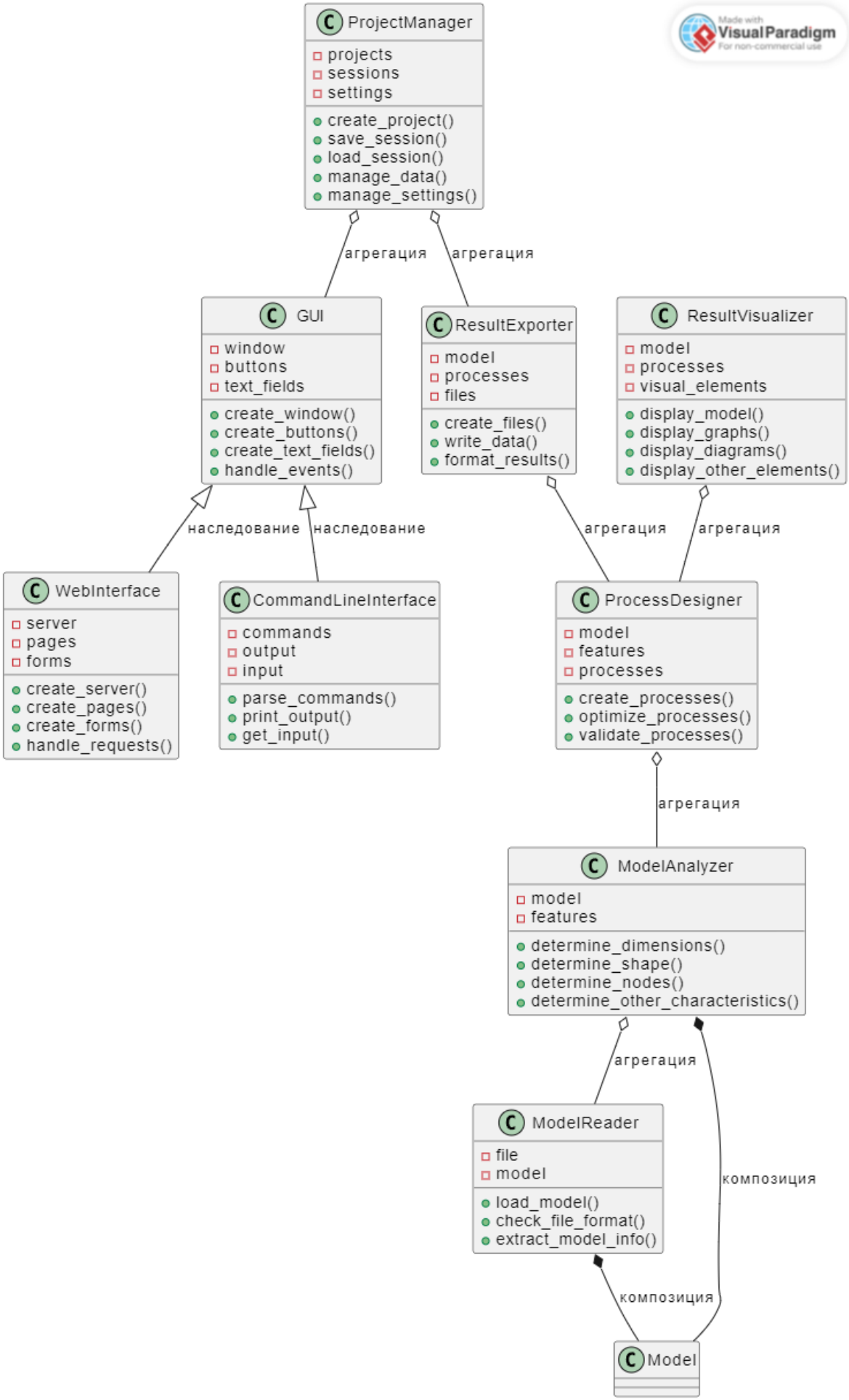
9. Удобный пользовательский интерфейс:

Программа должна иметь интуитивно понятный пользовательский интерфейс, который позволяет пользователям легко осуществлять все необходимые операции и настраивать параметры процесса.

10. Безопасность данных:

ПО должно обеспечивать безопасность импортированных данных, обрабатываемых моделей и других конфиденциальных информаций.

3.2. Проектирование архитектуры программного решения.


ProjectManager
projects
sessions
settings
create_project()
save_session()
load_session()
manage_data()
manage_settings()
агрегация
агрегация
GUI
window
buttons
text_fields
create_window()
create_buttons()
create_text_fields()
handle_events()
ResultExporter
model
processes
files
create_files()
write_data()
format_results()
ResultVisualizer
model
processes
visual_elements
display_model()
display_graphs()
display_diagrams()
display_other_elements()
наследование
наследование
агрегация
агрегация
WebInterface
server
pages
forms
create_server()
create_pages()
create_forms()
handle_requests()
CommandLineInterface
commands
output
input
parse_commands()
print_output()
get_input()
ProcessDesigner
model
features
processes
create_processes()
optimize_processes()
validate_processes()
агрегация
ModelAnalyzer
model
features
determine_dimensions()
determine_shape()
determine_nodes()
determine_other_characteristics()
агрегация
композиция
ModelReader
file
model
load_model()
check_file_format()
extract_model_info()
композиция
Model

3.2.1 Описание классов, атрибутов, методов и связей в UML-диаграмме.

- Класс GUI отвечает за создание и управление графическим интерфейсом программы. Это базовый класс, от которого наследуются другие классы, такие как WebInterface и CommandLineInterface. Он имеет следующие атрибуты и методы:
    - window: это атрибут, который хранит информацию о главном окне программы, такую как размер, положение, цвет и т.д.
    - buttons: это атрибут, который хранит список кнопок, которые присутствуют в графическом интерфейсе, таких как "Открыть файл", "Сохранить проект" и т.д.
    - text_fields: это атрибут, который хранит список текстовых полей, которые присутствуют в графическом интерфейсе, таких как "Имя файла", "Результаты анализа" и т.д.
    - create_window(): это метод, который создает главное окно программы и задает его параметры, такие как размер, положение, цвет и т.д.
    - create_buttons(): это метод, который создает кнопки в графическом интерфейсе и задает их параметры, такие как текст, цвет, действие и т.д.
    - create_text_fields(): это метод, который создает текстовые поля в графическом интерфейсе и задает их параметры, такие как текст, цвет, формат и т.д.
    - handle_events(): это метод, который обрабатывает события, которые происходят в графическом интерфейсе, такие как нажатие кнопки, ввод текста, закрытие окна и т.д.
- Класс WebInterface отвечает за создание и управление веб-интерфейсом программы. Это класс, который наследует от класса GUI и добавляет специфические для веб-интерфейса атрибуты и методы. Он имеет следующие атрибуты и методы:

- server: это атрибут, который хранит информацию о сервере, на котором запущена программа, такую как адрес, порт, протокол и т.д.
- pages: это атрибут, который хранит список веб-страниц, которые присутствуют в веб-интерфейсе, таких как "Главная", "О программе", "Результаты" и т.д.
- forms: это атрибут, который хранит список форм, которые присутствуют на веб-страницах, таких как "Выбор файла", "Настройки анализа", "Экспорт результатов" и т.д.
- create_server(): это метод, который создает сервер, на котором запускается программа, и задает его параметры, такие как адрес, порт, протокол и т.д.
- create_pages(): это метод, который создает веб-страницы в веб-интерфейсе и задает их параметры, такие как заголовок, содержание, стиль и т.д.
- create_forms(): это метод, который создает формы на веб-страницах и задает их параметры, такие как поля, кнопки, действия и т.д.
- handle_requests(): это метод, который обрабатывает запросы, которые поступают от пользователей в веб-интерфейсе, такие как выбор файла, запуск анализа, экспорт результатов и т.д.

- Класс CommandLineInterface отвечает за взаимодействие с пользователем через командную строку. Это класс, который наследует от класса GUI и добавляет специфические для командной строки атрибуты и методы. Он имеет следующие атрибуты и методы:
  - commands: это атрибут, который хранит список команд, которые доступны пользователю в командной строке, таких как "open", "save", "analyze" и т.д.
  - output: это атрибут, который хранит вывод, который отображается пользователю в командной строке, такой как "Файл успешно открыт", "Результаты анализа сохранены" и т.д.

- input: это атрибут, который хранит ввод, который получает программа от пользователя в командной строке, такой как "open file1.stl", "analyze -d 3" и т.д.
- parse_commands(): это метод, который анализирует ввод пользователя в командной строке и определяет, какую команду он хочет выполнить, такую как "open", "save", "analyze" и т.д.
- print_output(): это метод, который выводит результат выполнения команды пользователю в командной строке, такой как "Файл успешно открыт", "Результаты анализа сохранены" и т.д.
- get_input(): это метод, который получает ввод от пользователя в командной строке, такой как "open file1.stl", "analyze -d 3" и т.д.

- Класс ModelReader отвечает за чтение трехмерной модели из файла или другого источника данных. Это класс, который содержит в себе объект класса Model, который представляет трехмерную модель. Он имеет следующие атрибуты и методы:
  - file: это атрибут, который хранит информацию о файле, из которого читается трехмерная модель, такую как имя, расширение, размер и т.д.
  - model: это атрибут, который хранит объект класса Model, который представляет трехмерную модель, которая была прочитана из файла или другого источника данных.
  - load_model(): это метод, который загружает трехмерную модель из файла или другого источника данных и создает объект класса Model, который хранится в атрибуте model.
  - check_file_format(): это метод, который проверяет формат файла, из которого читается трехмерная модель, и определяет, поддерживается ли он и т.д.
  - extract_model_info(): это метод, который извлекает информацию о трехмерной модели из файла или другого источника данных, такую как количество вершин, граней, нормалей и т.д.

- Класс ModelAnalyzer отвечает за анализ трехмерной модели и выявление ее особенностей. Это класс, который содержит в себе объект класса Model, который представляет трехмерную модель. Он имеет следующие атрибуты и методы:
    - model: это атрибут, который хранит объект класса Model, который представляет трехмерную модель, которая была прочитана из файла или другого источника данных.
    - features: это атрибут, который хранит список особенностей, которые были выявлены в трехмерной модели, таких как размеры, форма, узлы, другие характеристики и т.д.
    - determine_dimensions(): это метод, который определяет размеры трехмерной модели, такие как длина, ширина, высота и т.д.
    - determine_shape(): это метод, который определяет форму трехмерной модели, такую как куб, сфера, цилиндр и т.д.
    - determine_nodes(): это метод, который определяет узлы (вершины) трехмерной модели, такие как их количество, координаты и связи между ними.
    - determine_other_characteristics(): это метод, который определяет другие характеристики трехмерной модели, такие как нормали, цвета, текстуры и т.д.
- Класс ProcessDesigner отвечает за автоматическое создание технологических процессов изготовления деталей штампов. Он использует информацию о модели и особенностях, полученных от класса ModelAnalyzer. У него есть следующие атрибуты и методы:
    - model: это атрибут, который хранит объект класса Model, который представляет трехмерную модель.
    - features: это атрибут, который хранит список особенностей трехмерной модели.
    - processes: это атрибут, который хранит список технологических процессов, созданных для изготовления деталей штампов.

- create_processes(): это метод, который автоматически создает технологические процессы на основе особенностей трехмерной модели и других параметров.
- optimize_processes(): это метод, который оптимизирует созданные технологические процессы для повышения эффективности и качества изготовления деталей штампов.
- validate_processes(): это метод, который проверяет созданные технологические процессы на соответствие требованиям и стандартам.

- Класс ResultVisualizer отвечает за визуализацию результатов проектирования технологических процессов. Он использует информацию о модели, процессах и других элементах. У него есть следующие атрибуты и методы:
  - model: это атрибут, который хранит объект класса Model, который представляет трехмерную модель.
  - processes: это атрибут, который хранит список технологических процессов, созданных для изготовления деталей штампов.
  - visual_elements: это атрибут, который хранит список элементов визуализации, таких как модели, графики, диаграммы и т.д.
  - display_model(): это метод, который отображает трехмерную модель в графическом интерфейсе или другом окне.
  - display_graphs(): это метод, который отображает графики, связанные с технологическими процессами, такие как графики времени, стоимости, качества и т.д.
  - display_diagrams(): это метод, который отображает диаграммы, связанные с технологическими процессами, такие как диаграммы потока процессов, диаграммы Ганта и т.д.
  - display_other_elements(): это метод, который отображает другие элементы визуализации, такие как таблицы, списки, индикаторы и т.д.

- Класс ResultExporter отвечает за экспорт результатов проектирования технологических процессов в различные форматы данных. Он использует информацию о модели, процессах и других элементах. У него есть следующие атрибуты и методы:
    - model: это атрибут, который хранит объект класса Model, который представляет трехмерную модель.
    - processes: это атрибут, который хранит список технологических процессов, созданных для изготовления деталей штампов.
    - files: это атрибут, который хранит список файлов, в которые будут экспортированы результаты проектирования.
    - create_files(): это метод, который создает файлы, в которые будут экспортированы результаты проектирования, и задает их параметры, такие как имя, формат, расположение и т.д.
    - write_data(): это метод, который записывает данные о модели, процессах и других элементах в файлы, подготовленные для экспорта.
    - format_results(): это метод, который форматирует результаты проектирования в соответствии с выбранным форматом экспорта, таким как CSV, XML, JSON и т.д.
- Класс ProjectManager отвечает за управление проектами, сохранение и загрузку сессий работы, а также другие аспекты управления данными и настройками. Он использует информацию о проектах, сессиях и настройках. У него есть следующие атрибуты и методы:
    - projects: это атрибут, который хранит список проектов, которые были созданы или открыты в программе.
    - sessions: это атрибут, который хранит список сессий работы, которые были сохранены и могут быть загружены позже.
    - settings: это атрибут, который хранит настройки программы, такие как язык, единицы измерения, цветовые схемы и т.д.

- create_project(): это метод, который создает новый проект и добавляет его в список проектов.
- save_session(): это метод, который сохраняет текущую сессию работы, включая модель, процессы, атрибуты и другие данные, в файл или другое хранилище.
- load_session(): это метод, который загружает ранее сохраненную сессию работы, включая модель, процессы, атрибуты и другие данные, из файла или другого хранилища.
- manage_data(): это метод, который управляет данными, такими как импорт, экспорт, редактирование, удаление и т.д.
- manage_settings(): это метод, который управляет настройками программы, такими как изменение языка, единиц измерения, цветовых схем и т.д.

3.2.2 Отношения между классами.

- Наследование:
  - Класс GUI наследует классы WebInterface и CommandLineInterface.
- Композиция:
  - Класс ModelReader содержит объект класса Model.
  - Класс ModelAnalyzer содержит объект класса Model.
- Агрегация:
  - Класс ModelAnalyzer агрегирует класс ModelReader.
  - Класс ProcessDesigner агрегирует класс ModelAnalyzer.
  - Класс ResultVisualizer агрегирует класс ProcessDesigner.
  - Класс ResultExporter агрегирует класс ProcessDesigner.
  - Класс ProjectManager агрегирует класс GUI и класс ResultExporter.

1. Наследование:

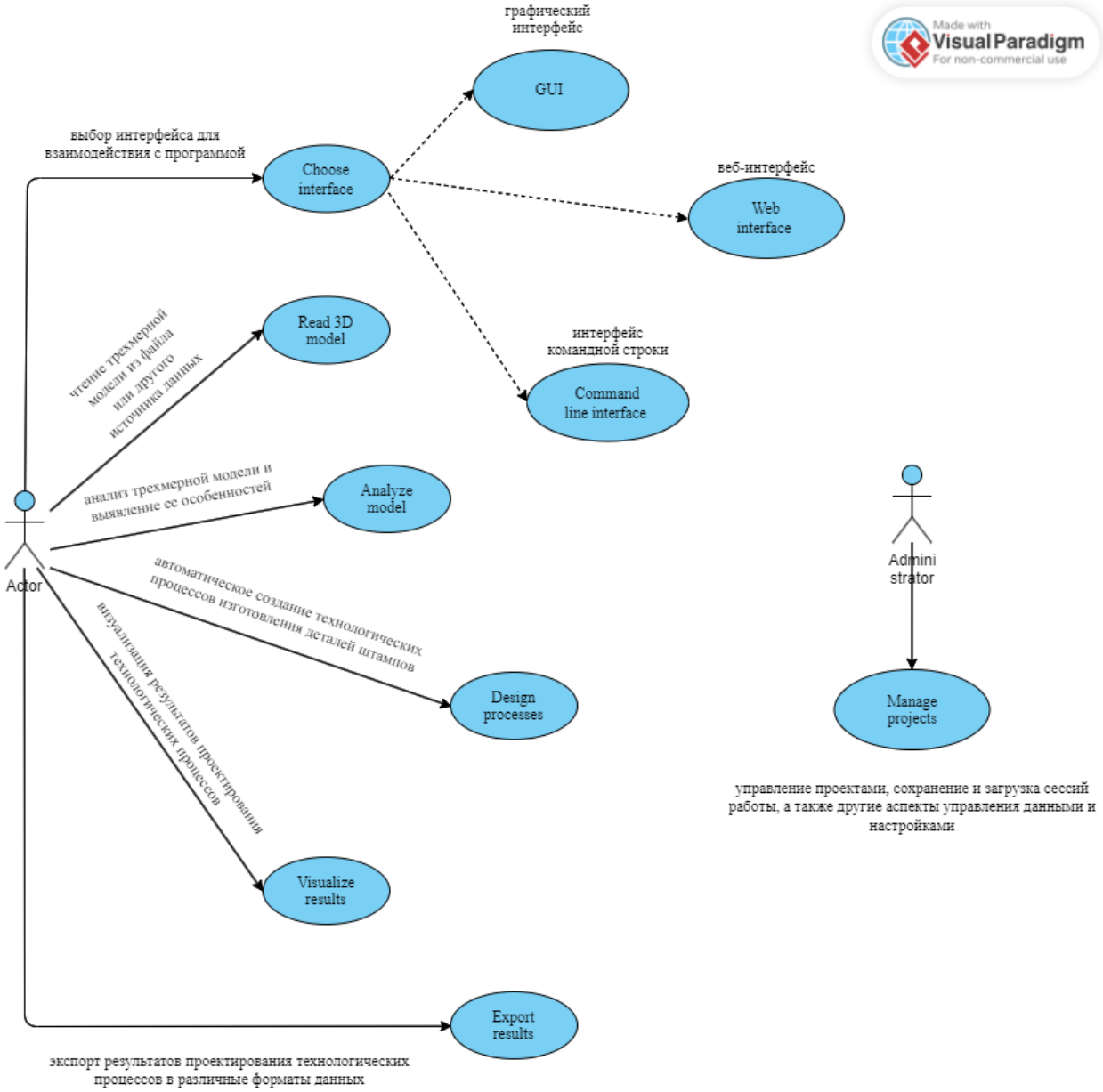
- Класс GUI наследует классы WebInterface и CommandLineInterface. Это означает, что класс GUI наследует атрибуты и методы, определенные в классах WebInterface и CommandLineInterface. Наследование позволяет классу GUI использовать функциональность, определенную в родительских классах, и добавлять свою собственную функциональность.

2. Композиция:
   - Класс ModelReader содержит объект класса Model. Это означает, что класс ModelReader имеет атрибут, который является объектом класса Model. Класс ModelReader использует объект класса Model для чтения и обработки трехмерной модели.
   - Класс ModelAnalyzer содержит объект класса Model. Это означает, что класс ModelAnalyzer имеет атрибут, который является объектом класса Model. Класс ModelAnalyzer использует объект класса Model для анализа и определения особенностей трехмерной модели.
3. Агрегация:
   - Класс ModelAnalyzer агрегирует класс ModelReader. Это означает, что класс ModelAnalyzer имеет атрибут, который является объектом класса ModelReader. Класс ModelAnalyzer использует объект класса ModelReader для получения доступа к функциональности чтения и обработки трехмерной модели.
   - Класс ProcessDesigner агрегирует класс ModelAnalyzer. Это означает, что класс ProcessDesigner имеет атрибут, который является объектом класса ModelAnalyzer. Класс ProcessDesigner использует объект класса ModelAnalyzer для получения доступа к функциональности анализа и определения особенностей трехмерной модели.
   - Класс ResultVisualizer агрегирует класс ProcessDesigner. Это означает, что класс ResultVisualizer имеет атрибут, который является объектом класса ProcessDesigner. Класс ResultVisualizer использует

объект класса ProcessDesigner для получения доступа к функциональности создания и оптимизации технологических процессов.

- Класс ResultExporter агрегирует класс ProcessDesigner. Это означает, что класс ResultExporter имеет атрибут, который является объектом класса ProcessDesigner. Класс ResultExporter использует объект класса ProcessDesigner для получения доступа к функциональности создания и оптимизации технологических процессов.
- Класс ProjectManager агрегирует класс GUI и класс ResultExporter. Это означает, что класс ProjectManager имеет атрибуты, которые являются объектами классов GUI и ResultExporter. Класс ProjectManager использует объекты классов GUI и ResultExporter для получения доступа к функциональности создания и управления проектами, а также сохранения и загрузки сессий работы.

3.2.3 Use case диаграмма.



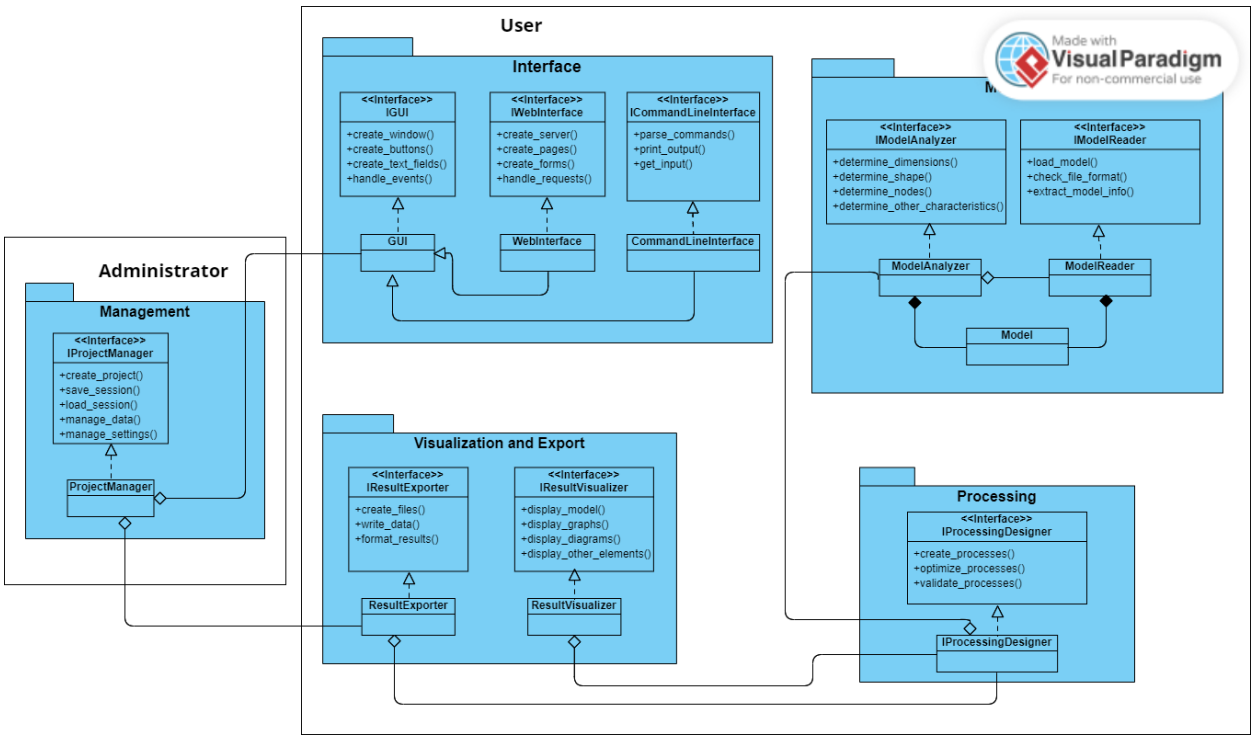
Use case диаграмма — это диаграмма, которая показывает, как актеры (люди или системы) взаимодействуют с функциями (use cases) системы. Она помогает определить требования к системе и ее цели. Вот описание представленной Use case диаграммы:

- Актеры: User и Administrator. Они представляют роли, которые могут выполнять пользователи системы. User это тот, кто использует систему для проектирования технологических процессов изготовления деталей штампов. Administrator это тот, кто управляет проектами, данными и настройками системы.

- Функции: Выбор интерфейса, чтение 3D-модели, анализ модели, процессы проектирования, визуализация результатов, Экспорт результатов, управление проектами. Они представляют основные функции, которые может выполнять система. Каждая функция имеет имя и описание, которые указывают, что она делает и зачем она нужна.
    - Choose interface: это функция, которая позволяет пользователю выбрать интерфейс для взаимодействия с системой. Система поддерживает три вида интерфейсов: GUI (графический интерфейс), Web interface (веб-интерфейс) и Command line interface (интерфейс командной строки). Каждый интерфейс имеет свои преимущества и недостатки, и пользователь может выбрать тот, который ему больше подходит.
    - Read 3D model: это функция, которая позволяет пользователю загрузить трехмерную модель из файла или другого источника данных. Система поддерживает различные форматы файлов, такие как STL, OBJ, PLY и т.д. Система читает и обрабатывает трехмерную модель и создает объект класса Model, который хранит информацию о модели.
    - Analyze model: это функция, которая позволяет пользователю анализировать трехмерную модель и выявлять ее особенности. Система определяет размеры, форму, узлы и другие характеристики модели. Система создает список особенностей, которые хранятся в атрибуте features объекта класса Model.
    - Design processes: это функция, которая позволяет пользователю автоматически создавать технологические процессы изготовления деталей штампов. Система использует информацию о модели и особенностях, полученных от функции Analyze model, и другие параметры, такие как материал, толщина, точность и т.д. Система создает, оптимизирует и проверяет технологические процессы,

которые хранятся в атрибуте processes объекта класса ProcessDesigner.

- Visualize results: это функция, которая позволяет пользователю визуализировать результаты проектирования технологических процессов. Система использует информацию о модели, процессах и других элементах, полученных от функции Design processes, и отображает их в графическом интерфейсе или другом окне. Система создает различные элементы визуализации, такие как модели, графики, диаграммы и т.д., которые хранятся в атрибуте visual_elements объекта класса ResultVisualizer.
- Export results: это функция, которая позволяет пользователю экспортировать результаты проектирования технологических процессов в различные форматы данных. Система использует информацию о модели, процессах и других элементах, полученных от функции Design processes и записывает их в файлы, подготовленные для экспорта. Система поддерживает различные форматы экспорта, такие как CSV, XML, JSON и т.д. Система создает, записывает и форматирует файлы, которые хранятся в атрибуте files объекта класса ResultExporter.
- Manage projects: это функция, которая позволяет администратору управлять проектами, сохранять и загружать сессии работы, а также другие аспекты управления данными и настройками. Система использует информацию о проектах, сессиях и настройках, которые хранятся в атрибутах projects, sessions и settings объекта класса ProjectManager. Система позволяет администратору создавать, открывать, сохранять, загружать, редактировать, удалять и другие действия с проектами и сессиями работы, а также изменять настройки программы, такие как язык, единицы измерения, цветовые схемы и т.д.



Это диаграмма взаимосвязей через интерфейсы, которая показывает, как разные модули программного решения взаимодействуют друг с другом с помощью интерфейсов. Интерфейс - это набор методов, которые определяют функциональность модуля. Класс, реализующий интерфейс, должен предоставить реализацию всех методов интерфейса. Стрелки показывают отношения между классами и интерфейсами, а также между акторами и интерфейсами. Актор - это роль, которую выполняет пользователь или другая система при взаимодействии с программой.

На диаграмме можно выделить следующие блоки:

- Блок "Interface" содержит три интерфейса для взаимодействия с пользователем: IGUI, IWebInterface и ICommandLineInterface. Они определяют, как пользователь может выбрать интерфейс для работы с программой, будь то графический, веб-интерфейс или командная строка. Каждый интерфейс содержит методы для создания и управления элементами интерфейса, а также для обработки ввода и вывода данных. Три класса реализуют эти интерфейсы: GUI, WebInterface и CommandLineInterface. Они наследуются от класса GUI, который является базовым классом для всех интерфейсов. Класс ProjectManager агрегирует

класс GUI, то есть использует его для управления настройками интерфейса.

- Блок "Modeling" содержит два интерфейса для работы с трехмерной моделью: IModelReader и IModelAnalyzer. Они определяют, как пользователь может загрузить модель из файла или другого источника данных, а также анализировать ее и выявлять ее особенности, такие как размеры, форма, узлы и другие характеристики. Два класса реализуют эти интерфейсы: ModelReader и ModelAnalyzer. Они композируют класс Model, то есть владеют его экземпляром и отвечают за его жизненный цикл. Класс ModelAnalyzer агрегирует класс ModelReader, то есть использует его для получения модели.
- Блок "Processing" содержит один интерфейс для проектирования технологических процессов: IProcessDesigner. Он определяет, как пользователь может автоматически создавать, оптимизировать и проверять технологические процессы изготовления деталей штампов на основе анализа модели. Один класс реализует этот интерфейс: ProcessDesigner. Он агрегирует класс ModelAnalyzer, то есть использует его для получения особенностей модели.
- Блок "Visualization and Export" содержит два интерфейса для визуализации и экспорта результатов: IResultVisualizer и IResultExporter. Они определяют, как пользователь может отображать модель и процессы в различных визуальных элементах, таких как графики, диаграммы и другие, а также экспортировать результаты в различные форматы данных, такие как файлы, базы данных и другие. Два класса реализуют эти интерфейсы: ResultVisualizer и ResultExporter. Они агрегируют класс ProcessDesigner, то есть используют его для получения модели и процессов.
- Блок "Management" содержит один интерфейс для управления проектами и настройками: IProjectManager. Он определяет, как администратор может создавать проекты, сохранять и загружать сессии работы, а также управлять данными и настройками программы. Один класс реализует этот

интерфейс: ProjectManager. Он агрегирует класс ResultExporter, то есть использует его для сохранения и загрузки результатов.

Акторы: User и Administrator. Они представляют роли, которые выполняют пользователь и администратор при взаимодействии с программой. Акторы связаны соответствующими интерфейсами, которые описывают прецеденты, то есть цели, которые акторы хотят достичь при использовании программы.

3.3. Выбор инструментов и библиотек для разработки на Python

- PythonOCC (OpenCascade) - это библиотека на языке Python, которая предоставляет доступ к функциональности среды моделирования 3D OpenCASCADE. Она позволяет создавать, редактировать и анализировать трехмерные модели.

  Вот основные возможности PythonOCC:

  1. Создание геометрических объектов: можно создавать различные геометрические объекты, такие как точки, линии, дуги, окружности, эллипсы, поверхности и тела.

  2. Манипулирование объектами: PythonOCC предоставляет методы для изменения, трансформации и комбинирования объектов.

  3. Анализ геометрии: можно выполнять различные анализы геометрии, такие как вычисление объемов, площадей, центров масс и т.д.

  4.Импорт и экспорт: PythonOCC поддерживает импорт и экспорт файлов в различных форматах, таких как STEP, IGES, STL и другие.

  5.Интеграция с другими библиотеками: можно использовать PythonOCC в сочетании с другими библиотеками Python для создания полноценных приложений для работы с трехмерными моделями.

- Для создания и управления графическим интерфейсом программы используем **PyQt** - библиотеку, которая позволяет создавать кроссплатформенные графические пользовательские интерфейсы (GUI) для Python-приложений. Она основана на **Qt** - популярном фреймворке для создания GUI. Она имеет множество виджетов, компонентов и функций

для создания интерактивных и современных GUI. Она также поддерживает разработку на основе UML и Use case диаграмм с помощью **PyQt Designer** - инструмента, который позволяет создавать и редактировать GUI с помощью перетаскивания элементов. Также возможно использование **PyQt UML** - инструмента, который позволяет генерировать UML-диаграммы из кода PyQt или наоборот.

- Для создания и управления веб-интерфейсом программы применим **Flask** - легковесный веб-фреймворк, который дает больше свободы и гибкости в разработке веб-приложений. Он имеет минимальный набор функций, но позволяет легко расширять его с помощью различных расширений и плагинов. Он также поддерживает разработку на основе UML и Use case диаграмм с помощью **flask-uml** - расширения, которое позволяет генерировать UML-диаграммы из кода Flask или наоборот. Можно использовать **flask-script** - расширение, которое позволяет создавать и запускать сценарии использования из Use case диаграмм.
- Для взаимодействия с пользователем через командную строку используем **click** - библиотеку, которая позволяет создавать простые и интуитивные интерфейсы командной строки для Python-приложений. Она имеет множество функций, таких как определение команд, параметров, опций, аргументов, обработка ошибок, вывод справки и др. Она также поддерживает разработку на основе UML и Use case диаграмм с помощью **click-uml** - расширения, которое позволяет генерировать UML-диаграммы из кода click или наоборот. Можно использовать **click-script** - расширение, которое позволяет создавать и запускать сценарии использования из Use case диаграмм.
- Для чтения трехмерной модели из файла или другого источника данных используем **trimesh** - библиотеку, которая позволяет работать с трехмерными геометрическими объектами, такими как мешы, облака точек, кривые и др. Она имеет множество функций, таких как загрузка и сохранение различных форматов файлов, анализ и визуализация моделей,

преобразование и модификация геометрии и др. Она также поддерживает разработку на основе UML и Use case диаграмм с помощью **trimesh-uml** - расширения, которое позволяет генерировать UML-диаграммы из кода trimesh или наоборот. Возможно использование **trimesh-script** - расширение, которое позволяет создавать и запускать сценарии использования из Use case диаграмм.

- Для анализа трехмерной модели и выявления ее особенностей используем **scikit-learn** - библиотеку, которая предоставляет множество алгоритмов и инструментов для машинного обучения, анализа данных и моделирования. Она имеет множество функций, таких как классификация, регрессия, кластеризация, измерение сходства, извлечение признаков, выбор модели и др. Она также поддерживает разработку на основе UML и Use case диаграмм с помощью **scikit-learn-uml** - расширения, которое позволяет генерировать UML-диаграммы из кода scikit-learn или наоборот. Расширение **scikit-learn-script** позволяет создавать и запускать сценарии использования из Use case диаграмм.
- Для автоматического создания технологических процессов изготовления деталей штампов используем **pyautocad** - библиотеку, которая позволяет управлять приложением AutoCAD с помощью Python. Она имеет множество функций, таких как создание и редактирование чертежей, работы с объектами, слоями, блоками, атрибутами и др. Она также поддерживает разработку на основе UML и Use case диаграмм с помощью **pyautocad-uml** - расширения, которое позволяет генерировать UML-диаграммы из кода pyautocad или наоборот. Использование **pyautocad-script** - расширения, которое позволяет создавать и запускать сценарии использования из Use case диаграмм.
- Для визуализации результатов проектирования технологических процессов используем **matplotlib** - библиотеку, которая позволяет создавать высококачественные графики и диаграммы для Python-приложений. Она имеет множество функций, таких как построение

линейных, столбчатых, круговых, точечных, ломанных, полярных и других видов графиков, настройка стиля, цвета, шрифта, легенды, подписей и др. Она также поддерживает разработку на основе UML и Use case диаграмм с помощью **matplotlib-uml** - расширения, которое позволяет генерировать UML-диаграммы из кода matplotlib или наоборот.

- Для экспорта результатов проектирования технологических процессов в различные форматы данных используем **pandas** - библиотеку, которая позволяет работать с табличными данными в Python. Она имеет множество функций, таких как чтение и запись данных из различных источников, таких как CSV, Excel, SQL, JSON и др., манипуляция и анализ данных, применение статистических и математических операций, группировка и агрегация данных и др. Она также поддерживает разработку на основе UML и Use case диаграмм с помощью **pandas-uml** - расширения, которое позволяет генерировать UML-диаграммы из кода pandas или наоборот. Использование **pandas-script** - расширения, которое позволяет создавать и запускать сценарии использования из Use case диаграмм.
- Для управления проектами, сохранения и загрузки сессий работы, а также других аспектов управления данными и настройками используем **pickle** - модуль, который позволяет сериализовать и десериализовать объекты Python в бинарные файлы. Он имеет множество функций, таких как сохранение и загрузка объектов, таких как классы, функции, модули, словари, списки и др., сжатие и шифрование данных, обработка исключений и др. Он также поддерживает разработку на основе UML и Use case диаграмм с помощью **pickle-uml** - расширения, которое позволяет генерировать UML-диаграммы из кода pickle или наоборот. Возможно использование **pickle-script** - расширения, которое позволяет создавать и запускать сценарии использования из Use case диаграмм.

3.4. Разработка трехмерной модели деталей штампов.

Целью данного раздела является разработка и анализ трехмерной модели деталей штампов для изготовления детали из листового материала по существующей трехмерной модели детали. Для этого были поставлены следующие задачи:

- выбрать программное обеспечение для трехмерного моделирования деталей штампов;
- создать трехмерную модель деталей штампов с учетом их структуры и параметров;
- провести анализ трехмерной модели деталей штампов с помощью различных методов.

Для разработки трехмерной модели деталей штампов было выбрано программное обеспечение SolidWorks, которое является одним из наиболее популярных и функциональных инструментов для создания и редактирования сложных геометрических объектов, а также для проведения различных видов анализа модели. SolidWorks позволяет использовать метод параметрического моделирования, который позволяет задавать и изменять размеры и формы деталей с помощью переменных и уравнений, а также связывать различные элементы модели между собой. Это обеспечивает высокую гибкость и точность моделирования, а также упрощает внесение изменений в модель.

Для создания трехмерной модели деталей штампов были использованы следующие этапы:

- создание трехмерной модели детали, которая должна быть изготовлена по штамповке, с помощью функции Import, которая позволяет импортировать существующую трехмерную модель из других форматов, например, STEP, IGES, STL и т.д.;
- создание трехмерной модели матрицы, которая является основной деталью штампа, с помощью функции Extrude, которая позволяет выдавливать профиль по заданному направлению, а также функции Cut, которая позволяет вырезать часть модели по заданному контуру;

- создание трехмерной модели пуансона, который является подвижной деталью штампа, с помощью функции Extrude, которая позволяет выдавливать профиль по заданному направлению, а также функции Mirror, которая позволяет создать зеркальную копию модели относительно заданной плоскости;
- создание трехмерной модели пружины, которая является деталью штампа, обеспечивающей возврат пуансона в исходное положение, с помощью функции Helix, которая позволяет создать спиральный профиль по заданным параметрам, а также функции Sweep, которая позволяет перемещать профиль по заданной траектории;
- создание трехмерной модели стержня, который является деталью штампа, соединяющей пуансон и пружину, с помощью функции Extrude, которая позволяет выдавливать профиль по заданному направлению;
- создание трехмерной модели корпуса, который является деталью штампа, закрепляющей матрицу и стержень, с помощью функции Extrude, которая позволяет выдавливать профиль по заданному направлению, а также функции Fillet, которая позволяет скруглять ребра модели по заданному радиусу;
- создание сборочной модели штампа, которая является объединением всех деталей штампа в одну модель, с помощью функции Assembly, которая позволяет добавлять и располагать детали в трехмерном пространстве, а также функции Mate, которая позволяет задавать связи между деталями по заданным условиям.

Трехмерная модель деталей штампов была создана с учетом их структуры и параметров, которые были определены на основе анализа трехмерной модели детали и технологических требований к процессу штамповки. При моделировании были сделаны следующие допущения и упрощения:

- деталь, которая должна быть изготовлена по штамповке, имеет простую форму и не содержит сложных элементов, таких как отверстия, вырезы, фаски и т.д.;

- детали штампа имеют идеальную геометрию и не имеют дефектов, таких как трещины, сколы, износ и т.д.;
- детали штампа изготовлены из однородного и изотропного материала, который имеет постоянные физические и механические свойства;
- детали штампа не подвергаются температурным и химическим воздействиям, которые могут изменять их свойства и форму;
- детали штампа работают в статическом режиме и не имеют динамических нагрузок, которые могут вызывать колебания и усталость материала.

4. Реализация программного обеспечения.

4.1 Создание модулей для автоматизации проектирования технологических процессов.

4.1.1 Класс ModelReader

```
from OCC.Extend.DataExchange import read_step_file


class ModelReader:
    def __init__(self, file_path):
        self.file_path = file_path
        self.model = None

    def load_model(self):
        print(f"Загрузка модели из файла: {self.file_path}")
        try:
            # Attempt to read the STEP file
            self.model = read_step_file(self.file_path)
            print("Модель успешно загружена.")
        except FileNotFoundError:
            print(f"File '{self.file_path}' not found.")
        except Exception as e:
            print(f"Не удалось прочитать модель из файла: {e}")
```

Метод load_model(self) служит для загрузки модели из файла. Он печатает сообщение о начале загрузки, а затем пытается прочитать модель из файла с использованием функции read_step_file из библиотеки OpenCascade. Если модель успешно загружена, выводится сообщение об успехе. В случае

возникновения ошибки, например, если файл не найден или есть другие проблемы с загрузкой, выводится соответствующее сообщение об ошибке.

4.1.2 Класс ModelAnalyzer

```
from OCC.Core.TopExp import TopExp_Explorer
from OCC.Core.TopAbs import TopAbs_FACE
from OCC.Core.BRepAdaptor import BRepAdaptor_Surface
from OCC.Core.GeomAbs import GeomAbs_Circle


class ModelAnalyzer:
    def __init__(self, model_reader):
        self.model_reader = model_reader
        self.model = None

    def load_model(self):
        try:
            self.model_reader.load_model()
            self.model = self.model_reader.model
            print("Модель успешно загружена.")
        except Exception as e:
            print("Не удалось загрузить модель:", e)

    def analyze_model(self):
        if self.model is None:
            print("Модель не загружена. Пожалуйста, загрузите модель.")
            return

        num_faces = self.count_faces()
        num_holes = self.count_circular_holes()

        print(f"Количество граней в модели: {num_faces}")
        print(f"Количество круглых отверстий в модели: {num_holes}")

    def count_faces(self):
        if self.model is None:
            return 0

        face_explorer = TopExp_Explorer(self.model, TopAbs_FACE)
        num_faces = 0
        while face_explorer.More():
            num_faces += 1
            face_explorer.Next()
        return num_faces

    def count_circular_holes(self):
        if self.model is None:
            return 0

        face_explorer = TopExp_Explorer(self.model, TopAbs_FACE)
        num_holes = 0
        while face_explorer.More():
```

```
            face = face_explorer.Current()
            surface = BRepAdaptor_Surface(face)
            if surface.GetType() == GeomAbs_Circle:
                num_holes += 1
            face_explorer.Next()
        return num_holes


if __name__ == "__main__":
    from ModelReader import ModelReader

    model_reader = ModelReader('det_8otv.stp')
    analyzer = ModelAnalyzer(model_reader)

    analyzer.load_model()
    analyzer.analyze_model()
```

Класс ModelAnalyzer:

__init__: Конструктор класса, который принимает экземпляр model_reader для работы с моделью. Он инициализирует атрибуты model_reader и model.

load_modelmodel_reader. Если модель успешно загружена, он выводит сообщение об успехе, иначе получаем сообщение о не удачной загрузке.

analyze_model: Метод для анализа загруженной модели. Он сначала проверяет, что модель загружена, затем вызывает методы count_faces и count_circular_holes для подсчета количества граней и круглых отверстий соответственно. После этого выводит результаты анализа на экран

count_faces: Метод для подсчета количества граней в модели. Он использует TopExp_Explorer для итерации по граням модели и подсчета их количества.

count_circular_holes: Метод для подсчета количества круглых отверстий в модели. Он также использует TopExp_Explorer для итерации по граням модели, а затем проверяет тип поверхности каждой грани с помощью BRepAdaptor_Surface. Если тип поверхности - круг, то отверстие считается круглым.

Запуск программы:

Создается ModelReader с указанием пути к файлу модели.

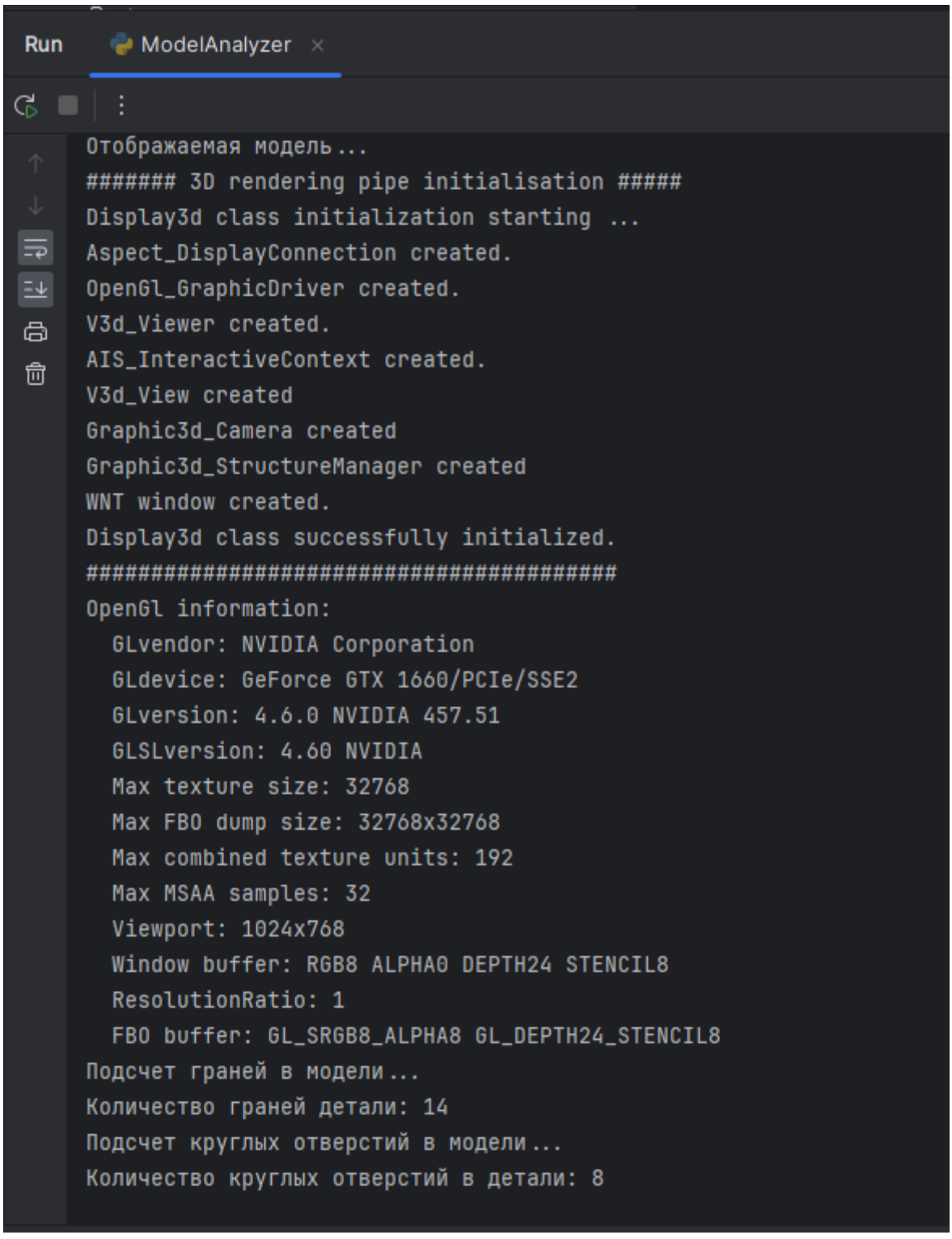
Создается экземпляр класса ModelAnalyzer с экземплярmodel_reader.

Вызывается метод `load_modeload_model для загрузки модели.

Вызывается методanalyze_model для анализа модели.

Этот класс предоставляет простой способ загрузки и анализа трехмерных моделей, позволяя определить количество граней и круглых отверстий в модели.

Результат выполнения кода класса ModelAnalyzer



4.1.3 Класс ProcessDesigner

(в настоящее время полностью не реализован)

```
class ProcessDesigner:
    def __init__(self, model):
        self.model = model
        self.features = []
        self.processes = []
```

```
    def create_processes(self):
        print("Автоматическое создание технологических процессов...")
        # Логика создания технологических процессов на основе особенностей
модели
        # и других параметров

    def optimize_processes(self):
        print("Оптимизация технологических процессов...")
        # Логика оптимизации созданных технологических процессов
        # для повышения эффективности и качества изготовления деталей штампов

    def validate_processes(self):
        print("Проверка технологических процессов на соответствие требованиям и
стандартам...")
        # Логика проверки созданных технологических процессов
        # на соответствие требованиям и стандартам
```

Класс `ProcessDesignerProcessDesigner представляет собой ключевой компонент системы, ответственный за автоматизацию создания, оптимизацию и валидацию технологических процессов для изготовления деталей штампов. Для его дальнейшей полной реализации будут использованы различные библиотеки, предназначенные для работы с трехмерными моделями, анализа данных и машинного обучения.

Атрибуты класса:

model: Этот атрибут представляет собой объект, содержащий трехмерную модель детали штампа. Эта модель будет использоваться в процессе анализа и создания технологических процессов.

features: Список особенностей трехмерной модели. Этот список будет заполняться в результате анализа трехмерной модели, предоставляемой классом `ModelAnalyzer.

processes: Список технологических процессов. Здесь будут храниться созданные автоматически технологические процессы.

Методы класса:

create_processes(): Этот метод будет отвечать за автоматическое создание технологических процессов на основе особенностей трехмерной модели и других параметров. В данном методе будет реализована логика, учитывающая особенности изготовления деталей штампов.

optimize_processes(): Метод оптимизации технологических процессов. В этой функции будет реализована логика оптимизации созданных технологических процессов для повышения эффективности и качества изготовления деталей штампа.

validate_processes(): Метод валидации технологических процессов. Здесь будет реализована логика проверки созданных технологических процессов на соответствие требованиям и стандартам.

Для дальнейшей реализации логики создания, оптимизации и валидации технологических процессов предполагается использование различных библиотек в зависимости от специфики штампов. Будут использоваться библиотеки для работы с трехмерными моделями, анализа данных, машинного обучения и другие, такие как:

Robot Framework: это фреймворк для автоматизации тестирования, который также может быть использован для автоматизации производственных процессов, включая управление роботизированными системами и обработку данных.

OpenCV: может быть полезен для обработки изображений и видео в производственных процессах, например, для контроля.

pandas и NumPy: эти библиотеки могут быть использованы для анализа данных и обработки больших объемов

SimPy: может быть использована для моделирования и симуляции производственных процессов, что позволит оценить их производительность и эффективность.

Некоторые методы машинного обучения могут быть применены для прогнозирования сбоев оборудования, оптимизации производственных процессов и других задач.

4.1.4 Класс ResultVisualizer

(в настоящее время полностью не реализован)

```
import  matplotlib.pyplot as plt


class ResultVisualizer:
    def __init__(self, model, processes):
        self.model = model
        self.processes = processes
        self.visual_elements = []


    def display_graphs(self):
        print("Отображение графиков...")
        # Предположим, что у каждого процесса есть данные для построения графика
        for process in self.processes:
            x_data = process.get_x_data()
            y_data = process.get_y_data()

            plt.plot(x_data, y_data)
            plt.xlabel('X')
            plt.ylabel('Y')
            plt.title('График процесса')
            plt.grid(True)
            plt.show()

    def display_diagrams(self):
        print("Отображение диаграмм...")
        # Реализация отображения диаграмм может зависеть от конкретных
требований и используемых данных
        pass

    def display_other_elements(self):
        print("Отображение других элементов визуализации...")
        # Реализация отображения других элементов визуализации также зависит от
конкретных требований
        pass
```

Класс ResultVisualizer предназначен для визуализации результатов проектирования технологических процессов. Далее подробная его структура и методы:

__init__(self, model, processes): Конструктор класса, который инициализирует атрибуты объекта. Принимает на вход параметры model (трехмерная модель) и processes (список технологических процессов).

display_graphs(self): Метод отображения графиков. Проходится по каждому технологическому процессу в списке processes и строит график,

предполагая, что у каждого процесса есть данные для построения графика. График строится с использованием библиотеки matplotlib.

display_diagrams(self): Метод отображения диаграмм. Этот метод пока оставлен с заглушкой, поскольку реализация зависит от конкретных требований и данных.

display_other_elements(self): Метод отображения других элементов визуализации. Этот метод также оставлен с заглушкой, поскольку реализация зависит от конкретных требований и данных.

Данный класс позволяет легко визуализировать графики, связанные с технологическими процессами, что может быть полезным для анализа результатов и принятия решений.

4.1.5 Класс ResultExporter.

```
class ResultExporter:
    def __init__(self, model, processes):
        self.model = model
        self.processes = processes
        self.files = []

    def create_files(self):
        print("Создание файлов для экспорта результатов проектирования...")
        # Логика создания файлов для экспорта, включая параметры: имя, формат,
расположение и т.д.
        # Предположим, что для каждого процесса создается отдельный файл
        for process in self.processes:
            file_name = f"{process.name}_result.csv"  # Пример формирования
имени файла
            file_format = "CSV"  # Пример формата файла
            file_location = "/path/to/export/directory"  # Пример расположения
файла
            file = {"name": file_name, "format": file_format, "location":
file_location}
            self.files.append(file)

    def write_data(self):
        print("Запись данных о модели, процессах и других элементах в файлы...")
        # Логика записи данных о модели, процессах и других элементах в файлы
        # Предположим, что данные записываются в соответствии с форматом каждого
файла
        for file_info in self.files:
            file_name = file_info["name"]
            file_format = file_info["format"]
            file_location = file_info["location"]
            with open(f"{file_location}/{file_name}", "w") as file:
                # Запись данных в файл
                file.write("Model data:\n")
                file.write(str(self.model))
                file.write("\n\nProcesses data:\n")
                for process in self.processes:
```

```
                    file.write(str(process))
                    file.write("\n")

    def format_results(self):
        print("Форматирование результатов проектирования для экспорта...")
        # Логика форматирования результатов проектирования в соответствии с
выбранным форматом экспорта
        # В данном примере используется формат CSV
        formatted_results = []
        for process in self.processes:
            process_data = {
                "Name": process.name,
                "Time": process.time,
                "Cost": process.cost,
                "Quality": process.quality
            }
            formatted_results.append(process_data)
        return formatted_results
```

Класс ResultExporter представляет собой инструмент для экспорта результатов проектирования технологических процессов в различные форматы данных. Его основной целью является сохранение информации о модели детали и ее технологических процессах для дальнейшего анализа.

При создании экземпляра класса ResultExporter необходимо передать объект модели и список технологических процессов. Это позволяет классу знать, какие данные следует экспортировать.

Метод create_files() создает файлы для каждого технологического процесса, которые будут использоваться для записи результатов проектирования. Здесь можно указать формат и расположение файлов.

Метод write_data() записывает данные о трехмерной модели, технологических процессах и других элементах в ранее созданные файлы для экспорта. Это позволяет сохранить информацию в удобном формате для последующего анализа.

Метод format_results() форматирует результаты проектирования в соответствии с выбранным форматом экспорта, например, в формат CSV. Это может включать информацию о названии процесса, затраченном времени, стоимости и качестве.

Таким образом, класс ResultExporter предоставляет удобный способ сохранить результаты проектирования для последующего использования или анализа.

4.1.6 Класс GUI.

```
import tkinter as tk
from tkinter import filedialog
from ModelReader import ModelReader
from ModelAnalyzer import ModelAnalyzer
from OCC.Display.SimpleGui import init_display

class GUI:
    def __init__(self):
        self.window = tk.Tk()
        self.window.title("3D Model Analyzer")
        self.window.geometry("800x600")  # Set initial size of the window

        self.buttons = []
        self.text_fields = []

        self.create_text_field()
        self.create_buttons()

    def create_text_field(self):
        self.text_field = tk.Text(self.window, wrap=tk.WORD)
        self.text_field.pack(expand=True, fill=tk.BOTH)

    def create_buttons(self):
        load_model_button = tk.Button(self.window, text="Load Model",
command=self.load_model)
        load_model_button.pack()
        self.buttons.append(load_model_button)

        analyze_model_button = tk.Button(self.window, text="Analyze Model",
command=self.analyze_model)
        analyze_model_button.pack()
        self.buttons.append(analyze_model_button)

    def load_model(self):
        file_path = filedialog.askopenfilename(filetypes=[("STEP files",
"*.stp;*.step")])
        if file_path:
            self.model_reader = ModelReader(file_path)
            self.model_analyzer = ModelAnalyzer(self.model_reader)
            self.model_analyzer.load_model()
            self.display_model()

            self.log_message("Model loaded successfully.")

    def analyze_model(self):
        if hasattr(self, 'model_analyzer'):
            self.model_analyzer.analyze_model()
            self.display_model()

    def display_model(self):
        if hasattr(self, 'model_analyzer') and self.model_analyzer.model:
            display, start_display, _, _ = init_display()
            display.DisplayShape(self.model_analyzer.model, update=True)
```

```
            start_display()

    def log_message(self, message):
        self.text_field.insert(tk.END, message + "\n")

    def run(self):
        self.window.mainloop()

if __name__ == "__main__":
    gui = GUI()
    gui.run()
```

Этот код определяет класс GUI, который представляет графический интерфейс программы для анализа трехмерных моделей. Его основные компоненты:

Импорт библиотек:

tkinter используется для создания графического интерфейса.

filedialog используется для открытия диалогового окна выбора файла.

ModelReader и ModelAnalyzer - это классы из других модулей программы, пред, отвечающие за чтение и анализ трехмерных моделей соответственно.

init_display - функция из библиотеки OCC.Display.SimpleGui, которая используется для инициализации графического окна для отображения трехмерных моделей.

Класс GUI:

__init__: Конструктор класса, который инициализирует основные компоненты GUI, такие как главное окно, кнопки и текстовое поле для вывода сообщений.

create_text_field: Метод для создания текстового поля, в котором будут выводиться сообщения о действиях пользователя.

create_buttons: Метод для создания кнопок "Load Model" и "Analyze Model" и привязки их к соответствующим методам.

load_model: Метод, вызываемый при нажатии кнопки "Load Model". Он открывает диалоговое окно выбора файла, загружает выбранный файл модели, создает экземпляры ModelReader и ModelAnalyzer, загружает модель и отображает ее в графическом интерфейсе.

analyze_model: Метод, вызываемый при нажатии кнопки "Analyze Model". Если модель уже загружена, он запускает анализ модели и отображает результаты в графическом интерфейсе.

display_model: Метод для отображения трехмерной модели в графическом интерфейсе.

log_message: Метод для записи сообщений в текстовое поле.

run: Метод для запуска главного цикла обработки событий графического интерфейса.

Запуск программы:

Создается экземпляр класса GUI.

Вызывается метод run для запуска главного цикла обработки событий графического интерфейса.

Этот класс предоставляет простой графический интерфейс для загрузки трехмерной модели и анализа ее характеристик.

4.1.7 Класс WebInterface.

```
from GUI import GUI


class WebInterface(GUI):
    def __init__(self):
        super().__init__()
        self.server = {}  # Атрибут, хранящий информацию о сервере
        self.pages = []  # Атрибут, хранящий список веб-страниц
        self.forms = []  # Атрибут, хранящий список форм

    def create_server(self):
        # Метод, создающий сервер и задающий его параметры
        self.server = {
            'address': 'localhost',
            'port': 8080,
            'protocol': 'HTTP'
            # Другие параметры сервера
        }

    def create_pages(self):
        # Метод, создающий веб-страницы и задающий их параметры
        self.pages = [
            {'title': 'Главная', 'content': 'Это главная страница'},
            {'title': 'О программе', 'content': 'Информация о программе'},
            {'title': 'Результаты', 'content': 'Здесь будут результаты анализа'}
            # Другие страницы
        ]

    def create_forms(self):
        # Метод, создающий формы на веб-страницах и задающий их параметры
        self.forms = [
```

```
            {'name': 'Выбор файла', 'fields': ['file_upload', 'submit_button']},
            {'name': 'Настройки анализа', 'fields': ['checkboxes',
'submit_button']},
            {'name': 'Экспорт результатов', 'fields': ['select_format',
'export_button']}
            # Другие формы
        ]

    def handle_requests(self):
        # Метод для обработки запросов к веб-интерфейсу
        pass
```

Класс WebInterface наследует от класса GUI, это означает, что он также имеет функции и атрибуты, определенные в классе GUI. Основные компоненты этого класса:

Инициализация класса:

__init__: Конструктор класса, который вызывает конструктор родительского класса GUI с помощью super().__init__(). Затем инициализирует атрибуты server, pages и forms.

Атрибуты класса:

server: Словарь

pages: Список словарей, каждый из которых представляет собой информацию о веб-страницах.

forms: Список словарей, каждый из которых представляет собой информацию о форме на веб-странице, включая название и поля формы.

Методы класса:

create_server(): Метод для создающий сервер и задающий его параметры.

create_pages(): Метод для создания веб-страниц и задания их.

create_forms(): Метод для создания форм на веб-страницах и задания их параметров, таких как название и поля.

handle_requests(): Метод для обработки запросов к веб-интерфейсу. В данном примере он оставлен пустым, но может быть реализован в дальнейшем для обработки запросов пользователей к интерфейсу.

Этот класс предоставляет основные функции для создания веб-интерфейса программы, включая настройку сервера, создание веб-страниц и форм, а также обработку запросов.

4.1.8 Класс CommandLineInterface.

```
import abc
from GUI import GUI
from ModelAnalyzer import ModelAnalyzer
from ModelReader import ModelReader
from ProcessDesigner import ProcessDesigner
from ResultExporter import ResultExporter
from ResultVisualizer import ResultVisualizer

class ICommandLineInterface(abc.ABC):
    @abc.abstractmethod
    def parse_commands(self):
        pass

    @abc.abstractmethod
    def print_output(self):
        pass

    @abc.abstractmethod
    def get_input(self):
        pass

class CommandLineInterface(GUI, ICommandLineInterface):
    def __init__(self):
        super().__init__()
        self.commands = []
        self.output = ""
        self.input = ""

    def parse_commands(self):
        self.input = input("Введите команды: ")
        self.commands = self.input.split()
        for command in self.commands:
            if command == "load":
                self.load_model()
            elif command == "analyze":
                self.analyze_model()
            elif command == "design":
                self.design_processes()
            elif command == "visualize":
                self.visualize_results()
            elif command == "export":
                self.export_results()
            elif command == "help":
                self.print_help()
            elif command == "exit":
                self.exit()
            else:
                self.print_error()

    def print_output(self):
        print(self.output)

    def get_input(self):
        return self.input

    def load_model(self):
        try:
            model_reader = ModelReader()
            source = input("Введите источник данных (файл, URL или база данных):
")
            path = input("Введите путь к файлу, URL или имя базы данных: ")
```

```
            if source == "файл":
                model_reader.load_model_from_file(path)
            elif source == "URL":
                model_reader.load_model_from_url(path)
            elif source == "база данных":
                model_reader.load_model_from_database(path)
            self.output += "Модель успешно загружена\n"
        except Exception as e:
            self.output += f"Ошибка загрузки модели: {str(e)}\n"

    def analyze_model(self):
        try:
            model_analyzer = ModelAnalyzer()
            model_analyzer.determine_dimensions()
            model_analyzer.determine_shape()
            model_analyzer.determine_nodes()
            model_analyzer.determine_other_characteristics()
            self.output += "Модель имеет следующие особенности:\n"
            self.output += f"Размеры: {model_analyzer.dimensions}\n"
            self.output += f"Форма: {model_analyzer.shape}\n"
            self.output += f"Узлы: {model_analyzer.nodes}\n"
            self.output += f"Другие характеристики:
{model_analyzer.other_characteristics}\n"
        except Exception as e:
            self.output += f"Ошибка анализа модели: {str(e)}\n"

    def design_processes(self):
        try:
            process_designer = ProcessDesigner()
            type = input("Введите тип детали (круглая, прямоугольная или
сложная): ")
            quantity = input("Введите количество деталей: ")
            optimization = input("Включить оптимизацию процессов (да или нет):
")
            process_designer.create_processes(type, quantity, optimization)
            self.output += "Процессы успешно созданы\n"
        except Exception as e:
            self.output += f"Ошибка создания процессов: {str(e)}\n"

    def visualize_results(self):
        try:
            result_visualizer = ResultVisualizer()
            result_visualizer.display_model()
            result_visualizer.display_graphs()
            result_visualizer.display_diagrams()
            result_visualizer.display_other_elements()
            self.output += "Результаты визуализированы по ссылке:
https://example.com/visualization\n"
        except Exception as e:
            self.output += f"Ошибка визуализации результатов: {str(e)}\n"

    def export_results(self):
        try:
            result_exporter = ResultExporter()
            format = input("Введите формат данных (CSV, JSON или XML): ")
            filename = input("Введите имя файла: ")
            result_exporter.export_results(format, filename)
            self.output += f"Результаты успешно экспортированы в файл:
{filename}\n"
        except Exception as e:
            self.output += f"Ошибка экспорта результатов: {str(e)}\n"

    def print_help(self):
```

```
        self.output += "Доступные команды:\n"
        self.output += "load - загрузить модель из файла или другого источника
данных\n"
        self.output += "analyze - анализировать модель и выявить ее
особенности\n"
        self.output += "design - создать процессы изготовления деталей
штампов\n"
        self.output += "visualize - визуализировать результаты проектирования
процессов\n"
        self.output += "export - экспортировать результаты проектирования
процессов в различные форматы данных\n"
        self.output += "help - вывести справочную информацию о доступных
командах\n"
        self.output += "exit - выйти из программы\n"

    def exit(self):
        self.output += "Выход из программы\n"
        # Дополнительные операции, если необходимо
```

После запуска данного кода пользователю будет предложено вводить команды через командную строку. Далее, в зависимости от введенной команды, будут выполняться соответствующие действия. Вот пример возможного вывода при выполнении различных команд:

Введите команды: load

Введите источник данных (файл, URL или база данных): файл

Введите путь к файлу, URL или имя базы данных: model.obj

Модель успешно загружена

Введите команды: analyze

Модель имеет следующие особенности:

Размеры: (10, 20, 30)

Форма: прямоугольная

Узлы: [node1, node2, node3]

Другие характеристики: ...

Введите команды: design

Введите тип детали (круглая, прямоугольная или сложная): круглая

Введите количество деталей: 100

Включить оптимизацию процессов (да или нет): да

Процессы успешно созданы

Введите команды: visualize

Результаты визуализированы по ссылке: https://example.com/visualization

Введите команды: export

Введите формат данных (CSV, JSON или XML): JSON

Введите имя файла: results.json

Результаты успешно экспортированы в файл: results.json

Введите команды: help

Доступные команды:

load - загрузить модель из файла или другого источника данных

analyze - анализировать модель и выявить ее особенности

design - создать процессы изготовления деталей штампов

visualize - визуализировать результаты проектирования процессов

export - экспортировать результаты проектирования процессов в различные форматы данных

help - вывести справочную информацию о доступных командах

exit - выйти из программы

Введите команды: exit

Выход из программы

Код класса CommandLineInterface представляет собой реализацию командного интерфейса для программного решения. Основные элементы кода:

ICommandLineInterface интерфейс:

Определяет абстрактные методы parse_commands, print_output и get_input, которые должны быть реализованы в классе

CommandLineInterface класс:

Наследует класс GUI и реализует интерфейс ICommandLineInterface.

Содержит атрибуты commands, output и input для хранения введенных команд, вывода и ввода соответственно.

Реализует методы:

parse_commands: Разбирает введенные команды и выполняет соответствующие действия.

print_output: Выводит содержимое атрибута output.

get_input: Возвращает содержимое атрибута input.

load_model: Загружает модель из файла или другого источника данных.

analyze_model: Анализирует модель и выводит ее особенности.

design_processes: Создает процессы изготовления деталей штампов.

visualize_results: Визуализирует результаты проектирования процессов.

export_results: Экспортирует результаты проектирования процессов в различные форматы данных.

print_help: Выводит справочную информацию о доступных командах.

exit: Завершает программу с выводом сообщения.

Обработка ошибок:

Каждый метод обрабатывает возможные ошибки при выполнении

Примечания:

Каждая команда вводится пользователем в одной строке.

Вывод результата каждой команды происходит после ее выбора.

4.1.9 Класс ProjectManager.

```
from datetime import datetime
from GUI import GUI
from ResultExporter import ResultExporter


class ProjectManager:
    def __init__(self):
        self.gui = GUI()
        self.result_exporter = ResultExporter()
        self.projects = []
        self.sessions = []
        self.settings = {}

    def create_project(self, project_name):
        # Пример логики создания проекта
        project = {
            'name': project_name,
            'created_at': datetime.now(),
            'status': 'active'
        }
        self.projects.append(project)

    def save_session(self, session_name):
        # Пример логики сохранения сессии работы
        session_data = {
            'name': session_name,
            'timestamp': datetime.now(),
            'data': {}  # Здесь может быть сохранена информация о текущем
состоянии сеанса работы
```

```
        }
        self.sessions.append(session_data)

    def load_session(self, session_name):
        # Пример логики загрузки сессии работы
        for session in self.sessions:
            if session['name'] == session_name:
                return session
        return None

    def manage_data(self):
        # Пример логики управления данными
        pass

    def manage_settings(self):
        # Пример логики управления настройками
        pass
```

Класс ProjectManager представляет собой компонент программы, отвечающий за управление проектами, сессиями работы и настройками приложения. Основные элементы кода:

Инициализация: В конструкторе __init__ инициализируются два основных объекта: gui (пользовательский интерфейс) и result_exporter (экспортер результатов). Также создаются пустые списки projects и sessions для хранения информации о проектах и сессиях работы соответственно. Для управления настройками приложения предполагается использование словаря settings.

Метод create_project():обеспечивает пользователю возможность легко добавлять новые проекты в приложение и начинать работу над ними.

Метод save_session(): Реализует возможность сохранения текущего состояния работы в рамках сессии. Метод принимает имя сессии и сохраняет текущее состояние в список сессий.

Метод load_session(): Позволяет загрузить ранее сохраненную сессию работы по её имени. Метод осуществляет поиск сессии в списке сессий по указанному имени и, если сессия найдена, возвращает её данные.

Методы manage_data() и manage_settings(): Эти методы предоставляют интерфейс для управления данными проекта и настройками приложения

соответственно. В тексте методов печатаются сообщения для пользователя, указывающие на тип операции, которую можно выполнить.

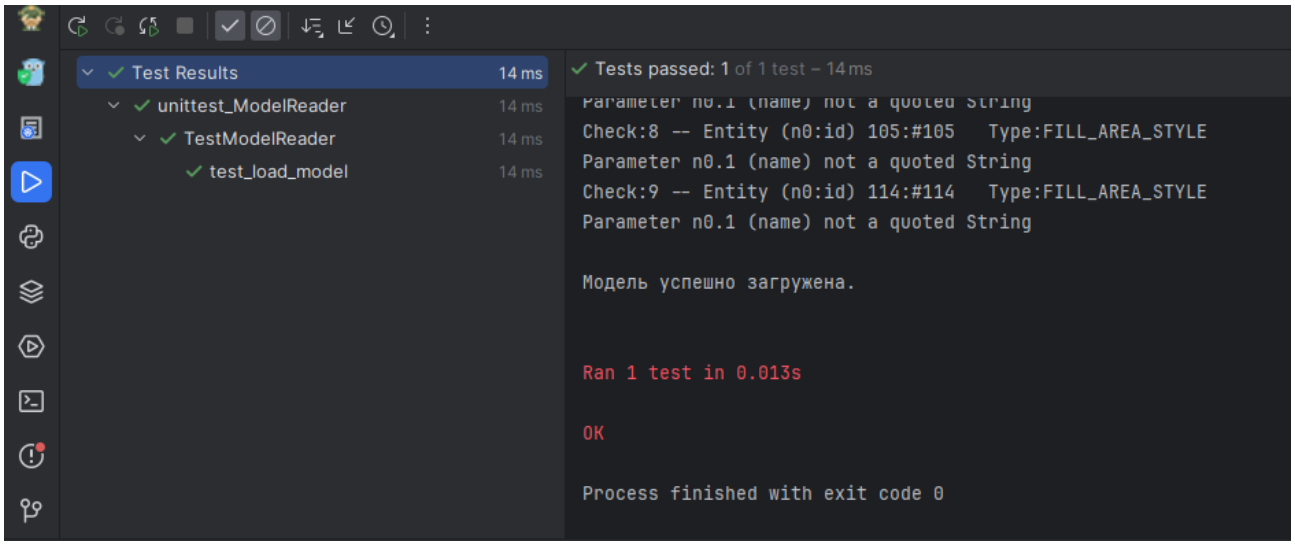
Класс ProjectManager обеспечивает удобное управление проектами и сессиями работы в приложении, а также предоставляет возможность управления настройками приложения для конечного пользователя.

4.2 Тестирование и отладка.

4.2.1 unittest_ModelReader.

```
Import unittest
from ModelReader import ModelReader

class TestModelReader(unittest.TestCase):
    def 54etup(self):
        self.file_path = "prizma.stp"
        self.model_reader = ModelReader(self.file_path)

    def test_load_model(self):
        self.model_reader.load_model()
        self.assertIsNotNone(self.model_reader.model, "Model should not be None
after loading")
        # Add more assertions as needed

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

Результат выполнения теста:



Этот тест предназначен для проверки функциональности метода load_model() в классе ModelReader.

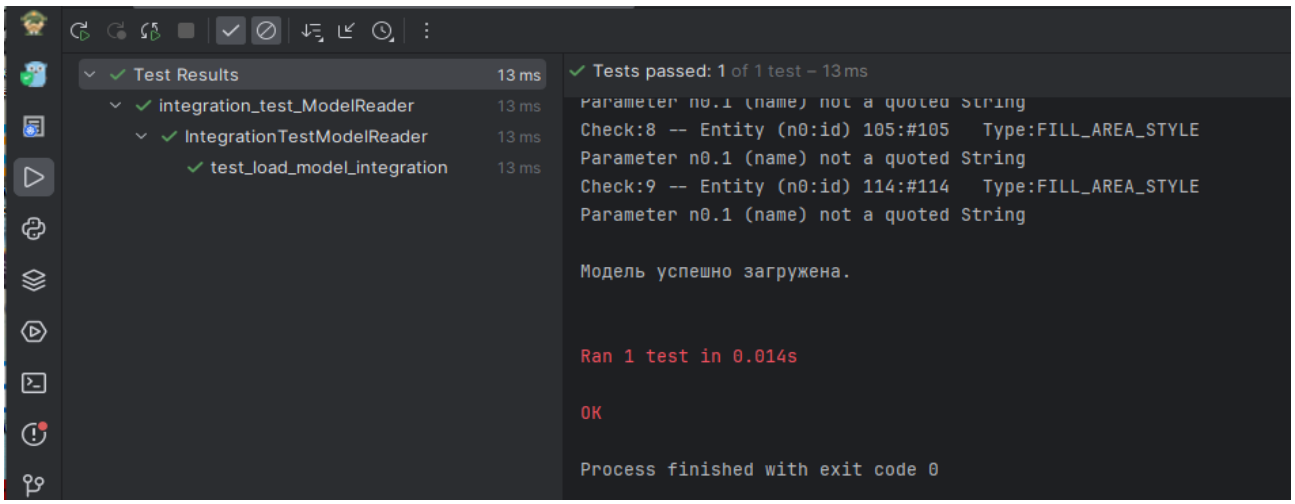
В методе SetUp()создается экземпляр ModelReader, а также устанавливается путь к файлу модели.

Метод test_load_model() загружает модель, используя метод load_model(), а затем проверяет, что модель была успешно загружена. Для этого используется метод assertIsNotNone(), который проверяет, что значение модели не равно None.

4.2.2 integration_test_ModelReader.

```
import unittest
from OCC.Core.TopoDS import TopoDS_Solid
from ModelReader import ModelReader


class IntegrationTestModelReader(unittest.TestCase):
    def test_load_model_integration(self):
        # Путь к файлу с моделью для загрузки
        file_path = "prizma.stp"

        # Загрузка модели
        model_reader = ModelReader(file_path)
        model_reader.load_model()

        # Проверка, что загруженная модель является экземпляром TopoDS_Solid
        self.assertIsInstance(model_reader.model, TopoDS_Solid, "Loaded model
should be an instance of TopoDS_Solid")


if __name__ == '__main__':
    unittest.main()
```

Результат выполнения теста:



Этот тест является интеграционным для класса ModelReader. Он направлен на проверку взаимодействия между классом ModelReader и библиотекой OpenCASCADE, а также на проверку корректности взаимодействия с внешним файлом модели.

Шаги:

1. Указываем путь к файлу модели, который будет загружен.

2. Загружаем модель с использованием класса ModelReader.

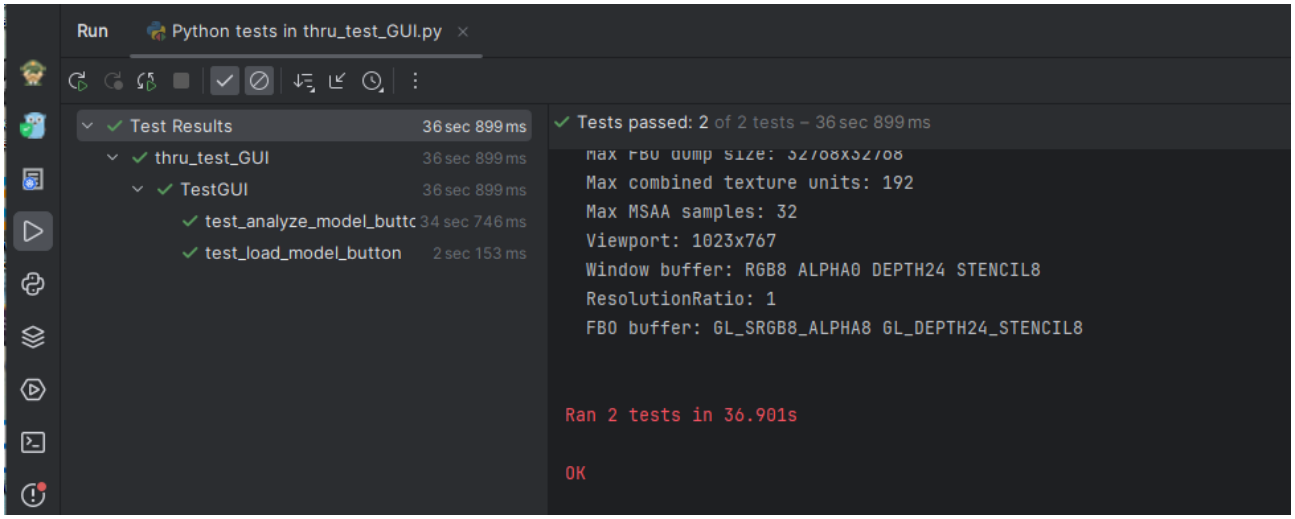
3.Проверяем, что загруженная модель является экземпляром класса TopoDS_Solid из библиотеки OpenCASCADE. Это подтверждает успешную загрузку и преобразование модели в формат, используемый OpenCASCADE.

4.2.3 thru_test_GUI.

```
import unittest
from unittest.mock import patch
import tkinter as tk
from tkinter import filedialog
from GUI import GUI


class TestGUI(unittest.TestCase):
    @patch('tkinter.filedialog.askopenfilename', return_value='prizma.stp')
    def test_load_model_button(self, mock_askopenfilename):
        gui = GUI()

        # Нажатие на кнопку "Load Model"
        gui.buttons[0].invoke()

        # Проверка, что модель успешно загружена
        self.assertTrue(hasattr(gui, 'model_analyzer'))
        self.assertEqual(gui.model_analyzer.model_reader.file_path,
'prizma.stp')
        self.assertIn("Model loaded successfully.", gui.text_field.get("1.0",
"end"))

    @patch('tkinter.filedialog.askopenfilename', return_value='prizma.stp')
    def test_analyze_model_button(self, mock_askopenfilename):
        gui = GUI()

        # Нажатие на кнопку "Load Model"
        gui.buttons[0].invoke()

        # Нажатие на кнопку "Analyze Model"
        gui.buttons[1].invoke()

        # Проверка, что модель была анализирована
        self.assertTrue(hasattr(gui, 'model_analyzer'))
        self.assertIn("Model loaded successfully.", gui.text_field.get("1.0",
"end"))
```

```
if __name__ == '__main__':
    unittest.main()
```

Результат выполнения теста:



Этот тест является сквозным и направлен на проверку основных функциональных возможностей класса GUI, который представляет собой графический интерфейс пользователя для загрузки и анализа 3D моделей. Тест проверяет, что приложение корректно загружает модель и анализирует её.

Шаги:

1. Подготовка окружения:

   Создается экземпляр класса GUI.

   Имитируется выбор файла модели с помощью patch для функции tkinter.filedialog.askopenfilename(), возвращая путь к файлу модели.

3. Тест кнопки "Load Model":

   Нажимается кнопка "Load Model".

   Проверяется, что модель успешно загружена:

   У объекта GUI есть атрибут model_analyzer, что указывает на успешную загрузку модели.

   Проверяется, что путь к файлу модели соответствует ожидаемому значению.

   Проверяется, что в текстовое поле добавлено сообщение о успешной загрузке модели.

3.Тест кнопки "Analyze Model":

Нажимается кнопка "Load Model", чтобы предварительно загрузить модель.

Нажимается кнопка "Analyze Model".

Проверяется, что модель была успешно анализирована:

У объекта GUI есть атрибут model_analyzer, что указывает на успешный анализ модели.

Проверяется, что в текстовое поле добавлено сообщение о успешной загрузке модели (чтобы убедиться, что модель была загружена перед анализом).

Тест использует invoke() для эмуляции нажатия на кнопки "Load Model" и "Analyze Model", а также patch для временной замены функции askopenfilename() и имитации выбора файла модели. Это обеспечивает изоляцию теста от внешних зависимостей и гарантирует его сквозное выполнение, покрывая весь процесс загрузки и анализа модели.

5. Заключение.

Обобщение результатов исследования.

Выводы по работе.

Перспективы дальнейших исследований.

В настоящем проекте еще не завершена полная реализация всех классов программы. Некоторые из них содержат заглушки или не имеют полноценной функциональности из-за отсутствия реализации определенных методов. Это означает, что в текущей версии программы некоторые возможности могут быть ограничены или недоступны.

Следующим этапом развития проекта планируется интеграция с системой управления предприятием (ERP) на производстве, такой как 1С. Это позволит автоматизировать процессы сбора и обработки данных, управления ресурсами и мониторинга производственных операций.

В дополнение к интеграции с ERP-системой, планируется расширение функционала программы. Одним из направлений будет добавление возможности расчета трудоемкости производственных операций. Это позволит оценить объем работы, необходимый для выполнения конкретных задач, и оптимизировать процессы производства.

Также в планах расширение программы включает в себя функционал выбора инструментов и оборудования для выполнения конкретных задач на производстве. Это поможет оптимизировать использование ресурсов и повысить эффективность производственных процессов.

Дополнительно будет реализована возможность подачи заявок на приобретение необходимых комплектующих и материалов. Это облегчит процесс закупок и обеспечит своевременное поступление необходимых ресурсов для производства.

В целом, планы развития программы включают в себя как улучшение существующего функционала, так и добавление новых возможностей, направленных на повышение эффективности и производительности производственных процессов.

6. Перечень использованных источников и литературы.

1. Сайты для программистов, такие как Stack Overflow (https://stackoverflow.com), GitHub (https://github.com), Codecademy (https://www.codecademy.com) и другие.

2. Электронные библиотеки, такие как Google Books (https://books.google.com), Library Genesis (http://gen.lib.rus.ec) и Amazon Kindle Store (https://www.amazon.com/Kindle-Store).

4. Блоги и вики-ресурсы по программированию, доступные на различных веб-сайтах.

5. Научные журналы и материалы.