МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

СЕВЕРО-КАВКАЗСКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ
СРЕДНЕПРОФЕССИОНАЛЬНЫЙ КОЛЛЕДЖ

З. С. Шовкарова

РАЗРАБОТКА КОДА ИНФОРМАЦИОННЫХ СИСТЕМ

ПРАКТИКУМ
для студентов III курса специальности
09.02.07 Информационные системы и программирование

Черкесск
2025

УДК 004
ББК 16.33
Ш78

Рассмотрено на заседании ЦК «Информационные дисциплины».
Протокол № 1 от «02» 09. 2024 г.
Рекомендовано к изданию редакционно-издательским советом СКГА
Протокол № 27 от «07» 11. 2024 г.

Рецензент: Моисеенко Л. А. –председатель ЦК «Информационные дисциплины»

Ш78 **Шовкарова, З. С.** Разработка кода информационных систем: практикум для студентов III курса специальности 09.02.07 Информационные системы и программирование/ З. С. Шовкарова. – Черкесск: БИЦ СКГА, 2025. – 42 с.

В данном практикуме представлены основные принципы и подходы к разработке кода для информационных систем. Она предназначена для студентов III курса специальности 09.02.07 Информационные системы и программирование, которые стремятся улучшить свои навыки в разработке эффективных и надежных программных решений.

УДК 004
ББК 16.33

СОДЕРЖАНИЕ

ПРАКТИЧЕСКАЯ РАБОТА 1. ПОСТРОЕНИЕ ДИАГРАММЫ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ И ДИАГРАММЫ. ПОСЛЕДОВАТЕЛЬНОСТИ И ГЕНЕРАЦИЯ КОДА

Для построения диаграмм вариантов использования и диаграмм последовательности, а также для генерации кода, можно следовать нескольким шагам.

Рассмотрим основные этапы:

1. Построение диаграммы вариантов использования (Use Case Diagram).

Диаграмма вариантов использования помогает описать функциональность системы с точки зрения пользователей (актеров). Она показывает, как пользователи взаимодействуют с системой.

Шаги для построения:

- Определите актеров: Кто будет взаимодействовать с вашей системой? (например, Пользователь, Администратор)
- Определите варианты использования: Какие действия могут выполнять актеры? (например, Вход в систему, Регистрация, Просмотр данных)
- Связи: Свяжите актеров с вариантами использования стрелками.

Пример:

[Пользователь] -- (Регистрация)
[Пользователь] -- (Вход в систему)
[Пользователь] -- (Просмотр данных)
[Администратор] -- (Управление пользователями)

2. Построение диаграммы последовательности (Sequence Diagram).

Диаграмма последовательности показывает, как объекты взаимодействуют друг с другом в определенном сценарии, отображая порядок сообщений.

Шаги для построения:

- Определите объекты: Какие объекты участвуют в сценарии?
- Определите сообщения: Какие сообщения передаются между объектами?
- Постройте временную шкалу: Отобразите порядок сообщений по вертикали.

Пример:

Пользователь -> Система: Вход в систему
Система -> База данных: Проверка учетных данных
База данных -> Система: Успех/Ошибка
Система -> Пользователь: Сообщение о результате

3. Генерация кода.

После создания диаграмм вы можете использовать их для генерации кода. Это можно сделать вручную или с помощью инструментов автоматизации.

Шаги для генерации кода:

1. Определите классы и методы: На основе диаграмм определите, какие классы и методы вам нужны.

2. Создайте структуру проекта: Создайте папки и файлы для вашего проекта.

3. Напишите код: Реализуйте классы и методы, следуя логике, описанной в диаграммах.

Пример кода на Python:

```python
class User:
 def login(self, username, password):
 # Логика входа в систему
 pass
class Database:
 def check_credentials(self, username, password):
 # Проверка учетных данных
 pass
class System:
 def user_login(self, username, password):
 db = Database()
 if db.check_credentials(username, password):
 return "Успех"
 else:
 return "Ошибка"
```

Домашнее задание:

Задание 1: Построение диаграммы вариантов использования

1. Выберите тему: Определите, какую систему вы будете моделировать. Например, это может быть система управления библиотекой, интернет-магазин или система бронирования.

2. Определите актеров: Составьте список всех пользователей, которые будут взаимодействовать с вашей системой. Например:

- Пользователь
- Администратор
- Гость

3. Определите варианты использования: Для каждого актера определите, какие действия они могут выполнять.

Например:

- Пользователь: Регистрация, Вход в систему, Поиск книг, Заказ книги.
- Администратор: Управление пользователями, Добавление книг, Удаление книг.

- Гость: Просмотр каталога книг.

4. Создайте диаграмму: Используйте инструменты для рисования (например, Lucidchart, draw.io или даже ручка и бумага), чтобы визуализировать диаграмму вариантов использования.

Задание 2: Построение диаграммы последовательности

1. Выберите сценарий: Выберите один из вариантов использования, который вы описали в первом задании. Например, "Регистрация пользователя".

2. Определите объекты: Определите, какие объекты будут участвовать в этом сценарии. Например:

- Пользователь
- Система
- База данных

3. Определите сообщения: Опишите, какие сообщения будут передаваться между объектами.

Например:

- Пользователь -> Система: Запрос на регистрацию
- Система -> База данных: Сохранить данные пользователя
- База данных -> Система: Подтверждение сохранения
- Система -> Пользователь: Сообщение об успешной регистрации

4. Создайте диаграмму: Визуализируйте диаграмму последовательности, используя те же инструменты, что и для диаграммы вариантов использования.

Задание 3: Генерация кода

1. Определите классы и методы: На основе диаграмм, которые вы создали, определите, какие классы и методы вам нужны. Например:

- Класс `User ` с методом `register()`
- Класс `Database` с методом `save_user()`
- Класс `System` с методом `user_registration()`

2. Создайте структуру проекта: Определите, как будет организован ваш проект.

Например:

/my_project
/models
user.py
database.py
/controllers
system.py
main.py

3. Напишите код: Реализуйте классы и методы. Пример кода на Python:

python
class User:
def __init__(self, username, password):
self.username = username
self.password = password

```
def register(self):
# Логика регистрации пользователя
pass
class Database:
def save_user(self, user):
# Логика сохранения пользователя в БД
pass
class System:
def user_registration(self, username, password):
user = User(username, password)
db = Database()
db.save_user(user)
return "Регистрация успешна"
```

ПРАКТИЧЕСКАЯ РАБОТА 2. ПОСТРОЕНИЕ ДИАГРАММЫ КООПЕРАЦИИ И ДИАГРАММЫ РАЗВЕРТЫВАНИЯ И ГЕНЕРАЦИЯ КОДА.

Построение диаграмм кооперации и развертывания, а также генерация кода — это важные шаги в процессе проектирования и разработки программного обеспечения. Давайте разберем каждый из этих этапов.

1. Диаграмма кооперации (Collaboration Diagram)

Диаграмма кооперации (или диаграмма взаимодействия) иллюстрирует, как объекты взаимодействуют друг с другом для выполнения определенной задачи. Она показывает связи между объектами и порядок сообщений, которые они обмениваются.

Шаги для построения диаграммы кооперации:

- Идентификация объектов: Определите объекты, которые будут участвовать в взаимодействии.
- Определение сообщений: Определите, какие сообщения будут обмениваться между объектами, и в каком порядке.
- Связи между объектами: Нарисуйте связи между объектами, показывая, как они взаимодействуют.
- Нумерация сообщений: Пронумеруйте сообщения для отображения порядка их отправки.

Пример:

Если у вас есть система для онлайн-заказа, объекты могут включать `Клиент`, `Заказ`, `Система оплаты`, и вы можете показать, как они взаимодействуют при создании и обработке заказа.

2. Диаграмма развертывания (Deployment Diagram)

Диаграмма развертывания показывает физическую архитектуру системы, включая узлы (например, серверы, устройства) и компоненты, которые будут развернуты на этих узлах.

Шаги для построения диаграммы развертывания:

- Определение узлов: Определите узлы, на которых будут развернуты компоненты системы (например, серверы, клиенты).
- Определение компонентов: Определите компоненты, которые будут развернуты на каждом узле.
- Связи между узлами: Установите связи между узлами, если они взаимодействуют друг с другом.
- Спецификация ресурсов: Укажите ресурсы, которые необходимы для каждого узла (например, объем памяти, процессор).

Пример:

Для системы онлайн-магазина узлы могут включать `Веб-сервер`, `Сервер базы данных` и `Клиентское устройство`, а компоненты могут включать `Веб-приложение`, `БД` и `Интерфейс пользователя`.

3. Генерация кода

Генерация кода — это процесс преобразования моделей (например, диаграмм) в исполняемый код. Это может быть сделано вручную или с использованием инструментов автоматизации.

Шаги для генерации кода:

- Выбор языка программирования: Определите, на каком языке будет написан код (например, Java, Python, C#).

- Инструменты генерации кода: Используйте инструменты, такие как UML-генераторы, которые могут автоматически создавать код на основе диаграмм.

- Ручная доработка: После генерации кода может потребоваться ручная доработка для реализации специфической логики или обработки исключений.

Пример: Если вы используете UML-диаграммы, вы можете использовать такие инструменты, как Enterprise Architect или Visual Paradigm, которые позволяют генерировать код на основе созданных вами диаграмм.

Домашнее задание

Задание 1: Построение диаграммы кооперации

1. Выберите сценарий: Определите, какой сценарий вы будете моделировать. Например, "Заказ книги".

2. Определите объекты: Составьте список объектов, участвующих в сценарии:

- Пользователь
- Система
- База данных
- Книга

3. Определите связи между объектами: Опишите взаимодействия между объектами:

- Пользователь запрашивает информацию о книге у Системы.
- Система получает данные о книге из Базы данных.
- Система отправляет информацию о книге Пользователю.

4. Создайте диаграмму кооперации: Используйте инструменты для рисования (например, Lucidchart, draw.io) для визуализации. Ваша диаграмма может выглядеть примерно так:

[Пользователь] --> (Запрос информации о книге) --> [Система]

[Система] --> (Запрос данных о книге) --> [База данных]

[База данных] --> (Данные о книге) --> [Система]

[Система] --> (Отправка информации) --> [Пользователь]

Задание 2: Построение диаграммы развертывания

1. Определите компоненты системы: Составьте список компонентов, например:

- Сервер приложений
- База данных
- Клиентское приложение

2. Определите узлы: Опишите узлы для развертывания компонентов:

- Сервер (где будет размещаться сервер приложений)
- Клиент (где будет размещаться клиентское приложение)

3. Определите связи между узлами: Опишите взаимодействия:
- Клиентское приложение взаимодействует с Сервером приложений.
- Сервер приложений взаимодействует с Базой данных.

4. Создайте диаграмму развертывания: Визуализируйте диаграмму. Она может выглядеть так:

[Клиент] --> (HTTP-запрос) --> [Сервер приложений]
[Сервер приложений] --> (SQL-запрос) --> [База данных]

Задание 3: Генерация кода

1. Определите классы и методы: На основе диаграмм, которые вы создали, определите, какие классы и методы вам нужны. Пример:
- Класс `Book` с методами для получения информации.
- Класс `User ` с методами для взаимодействия с системой.
- Класс `Database` для работы с данными.

2. Создайте структуру проекта:

/my_project
/models
book.py
user.py
database.py
/controllers
system.py
main.py

3. Напишите код: Пример кода на Python:

```python
# models/book.py
class Book:
def __init__(self, title, author):
self.title = title
self.author = author

def get_info(self):
return f"{self.title} by {self.author}"
# models/user.py
class User:
def __init__(self, username):
self.username = username
def request_book_info(self, book):
return book.get_info()
# models/database.py
class Database:
def __init__(self):
self.books = []
def add_book(self, book):
```

```
self.books.append(book)
def get_book(self, title):
for book in self.books:
if book.title == title:
return book
return None
# controllers/system.py
from models.book import Book
from models.user import User
from models.database import Database
db = Database()
book1 = Book("1984", "George Orwell")
db.add_book(book1)
user = User("Alice")
requested_book = db.get_book("1984")
if requested_book:
print(user.request_book_info(requested_book))
# main.py
if __name__ == "__main__":
# Запуск системы
pass
```

ПРАКТИЧЕСКАЯ РАБОТА 3. ПОСТРОЕНИЕ ДИАГРАММЫ ДЕЯТЕЛЬНОСТИ, ДИАГРАММЫ СОСТОЯНИЙ И ДИАГРАММЫ КЛАССОВ И ГЕНЕРАЦИЯ КОДА

Построение диаграмм деятельности, диаграмм состояний и диаграмм классов, а также генерация кода — это ключевые этапы в проектировании программного обеспечения. Давайте рассмотрим каждый из этих этапов более подробно.

1. Диаграмма деятельности (Activity Diagram)

Диаграмма деятельности описывает последовательность действий и поток управления в системе. Она полезна для моделирования бизнес-процессов и алгоритмов.

Шаги для построения диаграммы деятельности:

- Определение начальной и конечной точек: Установите начальную и конечную точки процесса.

- Идентификация действий: Определите действия, которые будут выполняться в процессе.

- Определение переходов: Установите переходы между действиями, указывая условия, при которых они происходят.

- Использование параллельных потоков: Если необходимо, используйте параллельные потоки для отображения одновременных действий.

Пример: Для системы онлайн-заказа диаграмма деятельности может включать действия, такие как "Выбор товара", "Добавление в корзину", "Оформление заказа" и "Оплата".

2. Диаграмма состояний (State Diagram)

Диаграмма состояний описывает различные состояния объекта и переходы между этими состояниями в ответ на события. Она полезна для моделирования поведения объектов в системе.

Шаги для построения диаграммы состояний:

- Определение состояний: Определите состояния, в которых может находиться объект.

-Определение переходов: Установите переходы между состояниями, указывая события, которые вызывают эти переходы.

- Определение действий: Укажите действия, которые выполняются при переходах или при входе/выходе из состояний.

Пример: Для заказа в системе онлайн-магазина состояния могут включать "Создан", "Оплачен", "Отправлен" и "Доставлен", а переходы могут быть вызваны событиями, такими как "Оплата подтверждена" или "Заказ отправлен".

3. Диаграмма классов (Class Diagram)

Диаграмма классов описывает структуру системы, показывая классы, их атрибуты, методы и отношения между классами. Она является основой для объектно-ориентированного проектирования.

Шаги для построения диаграммы классов:

-Идентификация классов: Определите классы, которые будут использоваться в системе.

- Определение атрибутов и методов: Укажите атрибуты и методы для каждого класса.

- Установление отношений: Определите отношения между классами (например, ассоциации, наследование, агрегация).

Пример: В системе онлайн-заказа классы могут включать `Клиент`, `Заказ`, `Товар`, с атрибутами, такими как `имя`, `цена`, и методами, такими как `оформить Заказ()`.

4. Генерация кода

Генерация кода — это процесс преобразования моделей (например, диаграмм) в исполняемый код. Это может быть выполнено вручную или с использованием инструментов автоматизации.

Шаги для генерации кода:

-Выбор языка программирования: Определите, на каком языке будет написан код (например, Java, Python, C#).

- Инструменты генерации кода: Используйте инструменты, такие как UML-генераторы, которые могут автоматически создавать код на основе диаграмм.

- Ручная доработка: После генерации кода может потребоваться ручная доработка для реализации специфической логики или обработки исключений.

Пример: Используя UML-диаграммы, вы можете использовать инструменты, такие как Enterprise Architect или Visual Paradigm, которые позволяют генерировать код на основе созданных вами диаграмм.

ПРАКТИЧЕСКАЯ РАБОТА 4. ПОСТРОЕНИЕ ДИАГРАММЫ КОМПОНЕНТОВ И ГЕНЕРАЦИЯ КОДА

Построение диаграммы компонентов и генерация кода — это важные этапы в проектировании программного обеспечения, особенно в архитектуре систем. Давайте рассмотрим, как это можно сделать на практике, включая пример диаграммы компонентов и процесс генерации кода.

1. Построение диаграммы компонентов.

Диаграмма компонентов (Component Diagram) описывает физические компоненты системы и их взаимосвязи. Компоненты могут включать модули, библиотеки, базы данных и другие элементы, которые составляют систему.

Шаги для построения диаграммы компонентов:

1. Идентификация компонентов: Определите основные компоненты системы. Например, для системы онлайн-магазина это могут быть:

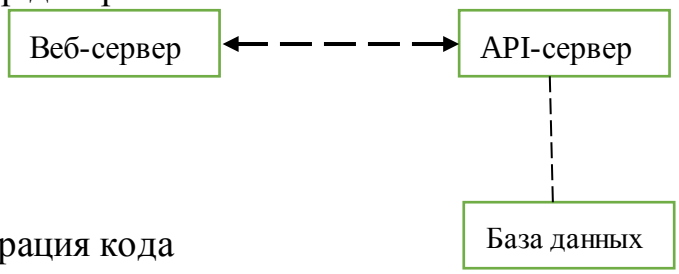
- Веб-сервер
- База данных
- API-сервер
- Клиентское приложение

2. Определение интерфейсов: Для каждого компонента определите интерфейсы, через которые они взаимодействуют друг с другом. Например, API-сервер может иметь интерфейс для доступа к данным о товарах.

3. Определение зависимостей: Установите зависимости между компонентами. Например, веб-сервер зависит от API-сервера, а API-сервер — от базы данных.

4. Создание диаграммы: Используйте инструменты для построения диаграмм, такие как Lucidchart, Draw.io или Microsoft Visio, чтобы визуализировать компоненты и их связи.

Пример диаграммы компонентов



2. Генерация кода

После создания диаграммы компонентов можно перейти к генерации кода. Это может быть сделано вручную или с использованием инструментов автоматизации. Рассмотрим пример генерации кода для компонентов на языке Python.

Шаги для генерации кода:

1. Определите структуру проекта: Создайте структуру каталогов для вашего проекта.

   Например:

2. Создайте код для каждого компонента:
Пример кода для `app.py` (Веб-сервер):

```
from flask import Flask
from api import get_products
app = Flask(__name__)
@app.route('/')
def home():
 return "Welcome to the Online  Store!"
@app.route('/products')
def products():
 return get_products()
if __name__ == '__main__':
 app.run(debug=True)
```

Пример кода для `api.py` (API-сервер):

```
python
from database import get_all_products
def get_products():
 products = get_all_products()
 return {"products": products}
```

```
**Пример кода для `db.py` (База данных)**:
```python
# Пример данных
products = [
 {"id": 1, "name": "Product 1", "price": 100},
 {"id": 2, "name": "Product 2", "price": 150},
]
def get_all_products():
 return products
```

3. Запуск проекта: После написания кода вы можете запустить веб-сервер и протестировать его, используя команду:

```
bash
python web_server/app.py
```
```

ПРАКТИЧЕСКАЯ РАБОТА 5. ПОСТРОЕНИЕ ДИАГРАММ ПОТОКОВ ДАННЫХ И ГЕНЕРАЦИЯ КОДА

Построение диаграмм потоков данных (DFD) и генерация кода — это важные аспекты разработки программного обеспечения, и Microsoft Visio является одним из инструментов, который может помочь в этом процессе. Давайте рассмотрим, как можно использовать Visio для создания DFD и генерации кода, а также сравним его с другими подходами.

Построение диаграмм потоков, данных в Visio

1. Создание DFD:

- Откройте Microsoft Visio и выберите шаблон для диаграммы потоков данных.

- Используйте фигуры, такие как процессы, потоки данных, хранилища данных и внешние сущности, чтобы визуализировать систему.

- Соедините фигуры с помощью стрелок, чтобы показать, как данные перемещаются между процессами и хранилищами.

2. Настройка элементов:

- Каждую фигуру можно настроить, добавляя текст, чтобы описать процессы и данные.

- Используйте цветовые коды или стили, чтобы выделить разные части диаграммы.

3. Экспорт и интеграция:

- Visio позволяет экспортировать диаграммы в различные форматы, такие как PDF или изображения, которые можно использовать в документации.

- Если необходимо, можно интегрировать диаграммы с другими инструментами разработки, используя API или плагины.

Генерация кода

Visio не предоставляет встроенных возможностей для генерации кода, как это делают некоторые специализированные инструменты. Однако, вы можете использовать DFD как основу для проектирования и затем вручную реализовать код на выбранном языке программирования, основываясь на диаграмме.

Сравнение с другими инструментами

1. Специализированные инструменты (например, Lucidchart, Draw.io):

- Эти инструменты часто предлагают более интуитивно понятный интерфейс для создания диаграмм и могут быть более доступными для совместной работы.

- Некоторые из них могут иметь функции для генерации кода или интеграции с системами управления версиями.

2. IDE с поддержкой UML:

- Некоторые среды разработки (например, IntelliJ IDEA, Eclipse) имеют плагины для построения UML-диаграмм и могут генерировать код на основе диаграмм классов.

- Это может быть полезно для быстрого прототипирования и создания архитектуры приложения.

3. Кодогенераторы:

- Существуют инструменты, которые могут генерировать код на основе спецификаций или моделей (например, Enterprise Architect).

- Они могут поддерживать различные языки программирования и фреймворки.

ПРАКТИЧЕСКАЯ РАБОТА 6. ОБОСНОВАНИЕ ВЫБОРА ТЕХНИЧЕСКИХ СРЕДСТВ. СТОИМОСТНАЯ ОЦЕНКА ПРОЕКТА

При выборе технических средств для создания диаграмм потоков данных (DFD) и других визуализаций в рамках проекта, важно учитывать несколько факторов, включая функциональность, удобство использования, стоимость и интеграцию с другими инструментами. Рассмотрим обоснование выбора Microsoft Visio и проведем стоимостную оценку проекта.

Обоснование выбора Microsoft Visio

1. Функциональность:

- Visio предлагает широкий набор шаблонов и фигур для создания DFD, что упрощает процесс визуализации сложных систем.

- Поддерживает интеграцию с другими продуктами Microsoft, такими как Excel и SharePoint, что может быть полезно для совместной работы и обмена данными.

2. Удобство использования:

- Интуитивно понятный интерфейс и возможность перетаскивания элементов делают Visio доступным для пользователей с разным уровнем подготовки.

- Наличие обучающих материалов и шаблонов помогает быстро освоить инструмент.

3. Совместная работа:

- Visio позволяет нескольким пользователям одновременно работать над одной диаграммой, что улучшает взаимодействие в команде.

- Возможность комментирования и аннотирования диаграмм упрощает процесс обсуждения и внесения изменений.

4. Экспорт и интеграция:

- Диаграммы можно экспортировать в различные форматы (PDF, PNG, SVG), что упрощает их использование в документации и презентациях.

- Поддержка интеграции с другими инструментами разработки и управления проектами.

Стоимостная оценка проекта

При оценке стоимости проекта с использованием Microsoft Visio следует учитывать следующие аспекты:

1. Лицензия на Microsoft Visio:

- Стоимость лицензии зависит от версии (например, Visio Standard или Visio Professional). На момент последнего обновления, стоимость лицензии может варьироваться от $5 до $15 в месяц за подписку на Microsoft 365.

- Для одной лицензии на Visio Professional стоимость может составлять около $250-$300 при однократной покупке.

2. Обучение пользователей:

- Если команда не знакома с Visio, может потребоваться обучение. Это может включать в себя курсы или вебинары, стоимость которых может варьироваться от $100 до $500 в зависимости от продолжительности и формата.

3. Затраты на проектирование:

- Время, необходимое для создания диаграмм, зависит от сложности проекта. Оцените, сколько часов потребуется команде для разработки DFD и других визуализаций.

- Например, если команда из 3-х человек работает 20 часов на создание диаграмм, и средняя почасовая ставка составляет $50, общие затраты на проектирование составят $3,000.

4. Дополнительные расходы:

- Учитывайте возможные расходы на техническую поддержку, обновления программного обеспечения и интеграцию с другими системами.

ПРАКТИЧЕСКАЯ РАБОТА 7. ПОСТРОЕНИЕ И ОБОСНОВАНИЕ МОДЕЛИ ПРОЕКТА. УСТАНОВКА И НАСТРОЙКА СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ С РАЗГРАНИЧЕНИЕМ РОЛЕЙ

1. Построение и обоснование модели проекта

Модель проекта — это структурированное представление всех аспектов проекта, включая цели, задачи, ресурсы, сроки и риски. Построение модели проекта помогает команде лучше понять, что нужно сделать, и как организовать работу.

Основные элементы модели проекта:

- Цели проекта: Четко определите, что вы хотите достичь. Это могут быть как краткосрочные, так и долгосрочные цели.

- Задачи: Определите конкретные задачи, которые необходимо выполнить для достижения целей. Разделите их на подзадачи для лучшей управляемости.

- Ресурсы: Укажите ресурсы, необходимые для выполнения проекта, включая людей, технологии, финансы и время.

- Сроки: Установите временные рамки для каждой задачи и общего проекта.

- Риски: Идентифицируйте потенциальные риски и разработайте стратегии их минимизации.

Обоснование модели:

- Анализ потребностей: Определите потребности заинтересованных сторон и обоснуйте, почему проект важен.

- Оценка затрат и выгод: Проанализируйте, какие выгоды принесет проект по сравнению с его затратами.

- Альтернативные решения: Рассмотрите альтернативные подходы и обоснуйте выбор выбранной модели.

2. Установка и настройка системы контроля версий с разграничением ролей

Система контроля версий (СКВ) — это инструмент, который помогает командам отслеживать изменения в коде и документации, а также управлять совместной работой над проектом. Одной из самых популярных СКВ является Git.

Шаги по установке и настройке Git:

1. Установка Git:

- Скачайте и установите Git с официального сайта git-scm.com.

- Следуйте инструкциям по установке для вашей операционной системы (Windows, macOS, Linux).

2. Настройка Git:

- Откройте терминал (или командную строку) и выполните следующие команды для настройки имени пользователя и электронной почты:

```bash
git config --global user.name "Ваше Имя"
git config --global user.email "ваш_email@example.com"
```

3. Создание репозитория:

- Перейдите в папку вашего проекта и выполните команду:

```bash
git init
```

- Это создаст новый репозиторий Git в текущей папке.

4. Создание и настройка удаленного репозитория:

- Создайте новый репозиторий на платформе (например, GitHub, GitLab, Bitbucket).

- Свяжите локальный репозиторий с удаленным:

```bash
git remote add origin https://github.com/ваш_пользователь/ваш_репозиторий.git
```

5. Разграничение ролей:

- Определите роли участников команды (например, разработчик, тестировщик, менеджер проекта).

- Настройте разрешения в системе управления проектами (например, в GitHub можно установить разные уровни доступа к репозиторию: администратор, разработчик, читатель).

- Используйте ветвление для разграничения работы: создавайте отдельные ветки для каждой задачи или функционала (например, `feature/имя_функции`).

6. Работа с ветками:

- Создавайте новые ветки для разработки новых функций:

```bash
git checkout -b feature/имя_функции
```

- После завершения работы с функцией, объедините изменения в основную ветку (обычно `main` или `master`):

```bash
git checkout main
git merge feature/имя_функции
```

ПРАКТИЧЕСКАЯ РАБОТА 8. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ. РАЗРАБОТКА ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Проектирование и разработка графического интерфейса пользователя (GUI) — это важный этап в разработке программного обеспечения, который позволяет создать удобный и интуитивно понятный интерфейс для пользователей. Microsoft Visio может быть полезным инструментом для визуализации и проектирования интерфейсов. Давайте рассмотрим, как использовать Visio для разработки графического интерфейса пользователя.

Этапы проектирования графического интерфейса с помощью MS Visio

1. Определение требований:

- Прежде чем начать проектирование, важно собрать требования от пользователей и заинтересованных сторон. Это поможет понять, какие функции и элементы интерфейса необходимы.

2. Создание структуры интерфейса:

- Используйте Visio для создания схемы структуры интерфейса. Это может включать в себя создание иерархии страниц (например, главная страница, страницы с продуктами, страницы контактов и т. д.).

- Определите ключевые элементы навигации, такие как меню, кнопки и ссылки.

3. Разработка макетов:

- Создайте макеты (wireframes) для различных экранов приложения. В Visio есть множество шаблонов и фигур, которые можно использовать для создания простых и понятных макетов.

- Определите расположение элементов управления (кнопки, поля ввода, изображения) и их размеры.

4. Добавление деталей:

- После создания базовых макетов добавьте детали, такие как текстовые метки, иконки и другие визуальные элементы. Это поможет лучше представить, как будет выглядеть конечный продукт.

- Используйте цветовую палитру и шрифты, которые соответствуют стилю вашего бренда.

5. Создание интерактивных прототипов:

- Visio позволяет создавать интерактивные прототипы, связывая различные страницы и элементы. Это поможет пользователям лучше понять, как будет работать интерфейс.

- Используйте гиперссылки для навигации между различными частями макета.

6. Обсуждение и обратная связь:

- Поделитесь созданными макетами с командой и заинтересованными сторонами для получения обратной связи. Visio позволяет легко вносить изменения на основе комментариев и предложений.

- Организуйте сессии обсуждения, чтобы понять, что нравится пользователям, а что требует доработки.

7. Финализация дизайна:

- После внесения всех правок и получения одобрения, финализируйте дизайн интерфейса. Подготовьте документацию, которая будет включать в себя описание каждого элемента интерфейса и его функциональности.

8. Передача в разработку:

- Подготовьте все необходимые материалы для передачи команде разработчиков. Это может включать в себя макеты, спецификации и рекомендации по дизайну.

Преимущества использования MS Visio для проектирования интерфейсов

- Удобство использования: Visio предлагает интуитивно понятный интерфейс и множество шаблонов, что упрощает процесс проектирования.

- Визуализация: Возможность создания наглядных схем и макетов помогает лучше понять структуру и функциональность интерфейса.

- Совместная работа: Visio позволяет нескольким пользователям работать над проектом одновременно, что улучшает взаимодействие в команде.

- Интеграция: Легкая интеграция с другими продуктами Microsoft позволяет использовать данные из Excel или других источников для улучшения дизайна.

ПРАКТИЧЕСКАЯ РАБОТА 9. РЕАЛИЗАЦИЯ АЛГОРИТМОВ ОБРАБОТКИ ЧИСЛОВЫХ ДАННЫХ. РЕАЛИЗАЦИЯ АЛГОРИТМОВ ПОИСКА. ОТЛАДКА ПРИЛОЖЕНИЯ

Реализация алгоритмов обработки числовых данных и поиска в приложении, таком как Microsoft Visio, может быть интересной задачей, особенно если вы хотите интегрировать визуальные элементы с функциональностью обработки данных. Давайте рассмотрим, как это можно сделать.

1. Реализация алгоритмов обработки числовых данных

– Обработка числовых данных может включать в себя такие операции, как:

– Сортировка: Упорядочивание чисел по возрастанию или убыванию.

– Агрегация: Подсчет суммы, среднего, максимума или минимума.

– Фильтрация: Выбор чисел, которые соответствуют определенным критериям.

Пример алгоритма сортировки (псевдокод):

```plaintext
функция сортировка(числа):
 для i от 0 до длина(числа) - 1:
 для j от 0 до длина(числа) - i - 1:
 если числа[j] > числа[j + 1]:
 поменять числа[j] и числа[j + 1]
 вернуть числа
```

2. Реализация алгоритмов поиска

Алгоритмы поиска могут включать в себя:

– Линейный поиск: Поиск элемента в массиве по одному элементу за раз.

– Бинарный поиск: Поиск элемента в отсортированном массиве, деля массив пополам.

Пример алгоритма бинарного поиска (псевдокод):

```plaintext
функция бинарный_поиск(числа, целевое_число):
низ = 0
высок = длина(числа) - 1
пока низ <= высок:
середина = (низ + высок) // 2
если числа[середина] == целевое_число:
вернуть середина
иначе если числа[середина] < целевое_число:
низ = середина + 1
иначе:
```

высок = середина - 1
вернуть -1 // элемент не найден

3. Отладка приложения в MS Visio

Отладка в MS Visio может включать в себя несколько шагов, особенно если вы разрабатываете макеты или прототипы с использованием формул и автоматизации:

- Проверка логики: Убедитесь, что алгоритмы, которые вы реализовали, работают правильно на тестовых данных.
- Использование встроенных инструментов: Visio имеет инструменты для проверки диаграмм и макетов на наличие ошибок.
- Тестирование интерактивных элементов: Если вы создали интерактивные прототипы, убедитесь, что все ссылки и действия работают правильно.
- Обратная связь от пользователей: Попросите коллег протестировать ваш макет и дать обратную связь.

ПРАКТИЧЕСКАЯ РАБОТА 10. РЕАЛИЗАЦИЯ ОБРАБОТКИ ТАБЛИЧНЫХ ДАННЫХ. ОТЛАДКА ПРИЛОЖЕНИЯ

Реализация обработки табличных данных и отладка приложения в Microsoft Visio — это важные аспекты, которые могут помочь вам создать эффективные и функциональные решения. Давайте рассмотрим, как это можно сделать.

1. Реализация обработки табличных данных

Обработка табличных данных включает в себя различные операции, такие как:

- Импорт данных: Загрузка данных из внешних источников (например, Excel).

- Обработка данных: Выполнение операций, таких как фильтрация, сортировка и агрегация.

- Визуализация данных: Представление данных в виде диаграмм или графиков.

Пример обработки табличных данных

Импорт данных из Excel в Visio:

1. Откройте Microsoft Visio и создайте новый документ.

2. Перейдите в меню "Данные" и выберите "Импорт данных".

3. Выберите "Из Excel" и следуйте инструкциям мастера для выбора файла и диапазона данных.

Обработка данных:

- Фильтрация: Вы можете использовать функции Visio для фильтрации данных. Например, вы можете создать диаграмму, которая отображает только определенные записи на основе заданного критерия.

- Сортировка: В Visio можно отсортировать данные, используя встроенные инструменты. Выберите таблицу и используйте опции сортировки в меню.

Агрегация данных:

- Вы можете использовать формулы для вычисления сумм, средних значений и других статистических показателей. Например, создайте поле, которое будет показывать сумму значений в определенном столбце.

2. Отладка приложения в MS Visio

Отладка в MS Visio включает в себя проверку правильности работы диаграмм, макетов и интерактивных элементов. Вот несколько шагов, которые помогут вам в этом процессе:

- Проверка логики: Убедитесь, что все формулы и связи между данными работают корректно. Проверьте, что данные отображаются правильно и соответствуют ожиданиям.

- Использование инструментов проверки: Visio предлагает инструменты для проверки диаграмм на наличие ошибок. Вы можете использовать эти инструменты для выявления проблем в структуре диаграммы или логике отображения данных.

- Тестирование интерактивных элементов: Если вы создали интерактивные элементы (например, кнопки или ссылки), протестируйте их, чтобы убедиться, что они работают правильно и ведут к ожидаемым результатам.

- Обратная связь от пользователей: Попросите коллег протестировать ваш проект и предоставить обратную связь. Это поможет выявить возможные проблемы и улучшить общий пользовательский опыт.

- Документация: Ведите документацию по всем изменениям и исправлениям, которые вы вносите в проект. Это поможет вам отслеживать прогресс и улучшать проект в будущем.

ПРАКТИЧЕСКАЯ РАБОТА 11. РАЗРАБОТКА И ОТЛАДКА ГЕНЕРАТОРА СЛУЧАЙНЫХ СИМВОЛОВ

Создание генератора случайных символов на Python — это интересная задача! Давайте разработаем простой генератор, который будет создавать случайные строки заданной длины, используя буквы, цифры и специальные символы. Мы будем использовать встроенный модуль `random` и `string`.

Пример кода генератора случайных символов

```python
import random
import string
def generate_random_string(length=10, use_digits=True, use_special_chars=True):
    # Определяем набор символов для генерации
    characters = string.ascii_letters  # Буквы (верхний и нижний регистры)
    if use_digits:
        characters += string.digits  # Добавляем цифры
    if use_special_chars:
        characters += string.punctuation  # Добавляем специальные символы
    Генерируем случайную строку
    random_string = ''.join(random.choice(characters) for _ in range(length))
    return random_string
    Пример использования
if __name__ == "__main__":
    length = int(input("Введите длину случайной строки: "))
    random_string = generate_random_string(length)
    print(f"Случайная строка: {random_string}")
```

Объяснение кода:

1. Импорт библиотек: Мы импортируем `random` для генерации случайных значений и `string` для доступа к наборам символов (буквы, цифры и специальные символы).

2. Функция `generate_random_string`:

- Принимает параметры:
- `length`: длина генерируемой строки (по умолчанию 10).
- `use_digits`: флаг для использования цифр (по умолчанию True).
- `use_special_chars`: флаг для использования специальных символов (по умолчанию True).
- Создает строку `characters`, которая содержит все возможные символы для генерации.
- Генерирует случайную строку заданной длины с помощью `random.choice()`.

3. Основной блок:

- Запрашивает у пользователя длину строки и вызывает функцию генерации.
- Выводит сгенерированную строку на экран.

Запуск программы

Сохраните код в файл, например, `random_string_generator.py`, и выполните его с помощью Python. Вы сможете ввести желаемую длину строки, и программа сгенерирует случайную строку.

ПРАКТИЧЕСКАЯ РАБОТА 12. РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ МОДЕЛИРОВАНИЯ ПРОЦЕССОВ И ЯВЛЕНИЙ. ОТЛАДКА ПРИЛОЖЕНИЯ

Моделирование процессов и явлений — это важная задача в различных областях, таких как физика, экономика, экология и многие другие. Создание приложений для моделирования позволяет визуализировать, анализировать и предсказывать поведение систем в различных условиях. В этом проекте мы рассмотрим, как разработать простое приложение для моделирования, а также как его отладить.

1. Определение задачи

Для начала определим, что именно мы хотим смоделировать. Например, давайте создадим приложение, которое будет моделировать движение тела под действием силы тяжести, используя закон свободного падения.

2. Проектирование приложения

Приложение будет включать следующие компоненты:

- Ввод параметров (начальная скорость, высота).
- Вычисление времени падения и конечной скорости.
- Визуализация результатов.

3. Реализация приложения

Вот пример простого приложения на Python, которое моделирует свободное падение:

```
python
import math
def calculate_free_fall(height, initial_velocity=0):
 g = 9.81 # Ускорение свободного падения в м/с²
 # Время падения
 time = math.sqrt((2 * height) / g)
 # Конечная скорость
 final_velocity = initial_velocity + g * time
 return time, final_velocity
def main():
 print("Моделирование свободного падения")
 height = float(input("Введите высоту (в метрах): "))
 initial_velocity = float(input("Введите начальную скорость (в м/с): "))
 time, final_velocity = calculate_free_fall(height, initial_velocity)
 print(f"Время падения: {time:.2f} секунд")
 print(f"Конечная скорость: {final_velocity:.2f} м/с")
if __name__ == "__main__":
 main()
```

4. Объяснение кода

1. Импорт библиотеки: Используем библиотеку `math` для математических расчетов.

2. Функция `calculate_free_fall`:

- Принимает параметры высоты и начальной скорости.

- Вычисляет время падения и конечную скорость с использованием формул физики.

3. Функция `main`:

- Запрашивает у пользователя входные данные.
- Вызывает функцию для вычисления и выводит результаты.

5. Отладка приложения

Отладка — это важный этап разработки, который помогает выявить и исправить ошибки. Вот несколько шагов, которые можно предпринять для отладки:

1. Тестирование функций: Проверьте, правильно ли работает функция `calculate_free_fall` с известными значениями. Например, если высота равна 0, время падения должно быть 0, а конечная скорость — равна начальной.

2. Вывод отладочной информации: Добавьте вывод промежуточных значений в консоль для проверки правильности расчетов.

```python
print(f"g: {g}, time: {time}, final_velocity: {final_velocity}")
```

3. Обработка ошибок**: Добавьте обработку исключений для ввода данных.

```python
try:
height = float(input("Введите высоту (в метрах): "))
except ValueError:
print("Ошибка: Введите числовое значение для высоты.")
return
```

4. Проверка на крайние случаи: Убедитесь, что программа корректно обрабатывает крайние случаи, такие как нулевая высота или отрицательные значения.

ПРАКТИЧЕСКАЯ РАБОТА 13. ИНТЕГРАЦИЯ МОДУЛЯ В ИНФОРМАЦИОННУЮ СИСТЕМУ

Интеграция модулей в информационные системы — это ключевая задача в разработке программного обеспечения, позволяющая объединить различные компоненты для создания единого, функционального приложения. В этом проекте мы рассмотрим, как интегрировать модуль в существующую информационную систему, а также основные шаги и методы, которые помогут сделать этот процесс более эффективным.

1. Определение задачи

Перед интеграцией модуля необходимо четко определить его функциональность и требования. Например, пусть это будет модуль для обработки платежей в системе электронной коммерции.

2. Проектирование модуля

На этапе проектирования следует учесть следующие аспекты:

- Функциональные требования: Что должен делать модуль? Какие операции он должен поддерживать (например, создание платежа, возврат, обработка ошибок)?

- Интерфейсы: Как модуль будет взаимодействовать с другими частями системы? Это может быть API, библиотека или интерфейс командной строки.

- Безопасность: Как будет обеспечиваться безопасность данных при интеграции (например, шифрование, аутентификация)?

3. Реализация модуля

Пример простого модуля для обработки платежей на Python:

```
python
class PaymentProcessor:
def __init__(self, api_key):
self.api_key = api_key
def create_payment(self, amount, currency):
 # Логика для создания платежа
 print(f"Создание платежа на сумму {amount} {currency}")
 Здесь должна быть интеграция с платежным API
 return {"status": "success", "amount": amount, "currency": currency}
 def refund_payment(self, payment_id):
 # Логика для возврата платежа
 print(f"Возврат платежа с ID {payment_id}")
 return {"status": "success", "payment_id": payment_id}
 Пример использования
if __name__ == "__main__":
 processor = PaymentProcessor(api_key="YOUR_API_KEY")
 payment = processor.create_payment(100, "USD")
 print(payment)
```

4. Интеграция модуля в систему

1. Подключение модуля: Импортируйте модуль в основное приложение. Убедитесь, что все зависимости установлены.

```python
from payment_processor import PaymentProcessor
```

2. Настройка конфигурации: Проверьте, что все необходимые параметры (например, API-ключи) переданы в модуль.

3. Тестирование интеграции: После интеграции проведите тестирование, чтобы убедиться, что модуль работает корректно в рамках всей системы. Используйте как юнит-тесты, так и интеграционные тесты.

4. Документация: Обновите документацию системы, чтобы отразить изменения, связанные с интеграцией нового модуля. Это важно для будущих разработчиков и пользователей.

5. Обработка ошибок и отладка

- Логирование: Включите логирование в модуле, чтобы отслеживать ошибки и важные события.

- Обработка исключений: Добавьте обработку исключений, чтобы модуль мог корректно реагировать на ошибки, возникающие в процессе работы.

```python
try:
 payment = processor.create_payment(100, "USD")
except Exception as e:
 print(f"Ошибка при создании платежа: {e}")
```

ПРАКТИЧЕСКАЯ РАБОТА 14. ПРОГРАММИРОВАНИЕ ОБМЕНА СООБЩЕНИЯМИ МЕЖДУ МОДУЛЯМИ

Обмен сообщениями между модулями является важной частью разработки программного обеспечения, особенно в многомодульных системах, где различные компоненты должны взаимодействовать друг с другом. Эффективное программирование обмена сообщениями позволяет обеспечить гибкость, масштабируемость и надежность системы. В этом проекте мы рассмотрим основные подходы и технологии для реализации обмена сообщениями между модулями.

1. Определение задачи

Прежде чем приступить к программированию, необходимо определить, какие модули будут взаимодействовать и какие сообщения они будут обмениваться. Например, в системе электронной коммерции могут взаимодействовать модули обработки заказов, управления запасами и уведомлений.

2. Подходы к обмену сообщениями

Существует несколько подходов к организации обмена сообщениями:

- Промежуточные очереди (Message Queues): Использование систем, таких как RabbitMQ или Apache Kafka, для асинхронной передачи сообщений между модулями.

- HTTP API: Использование RESTful или GraphQL API для синхронного обмена данными между модулями.

-Событийная архитектура: Реализация событий, на которые модули могут подписываться и реагировать на изменения состояния системы.

3. Пример реализации с использованием очередей сообщений

В этом примере мы будем использовать библиотеку `RabbitMQ` для реализации обмена сообщениями между модулями.

Установка RabbitMQ

Убедитесь, что RabbitMQ установлен и запущен на вашем компьютере или сервере. Вы можете установить библиотеку `pika` для работы с RabbitMQ:

```bash
pip install pika
```

Модуль отправки сообщений (Producer)

```python
import pika
def send_message(queue_name, message):
 connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
 channel = connection.channel()
 channel.queue_declare(queue=queue_name)  # Создание очереди, если она не существует
 channel.basic_publish(exchange='', routing_key=queue_name, body=message)
```

```
    print(f"Отправлено сообщение: {message}")
    connection.close()
if __name__ == "__main__":
    send_message('order_queue', 'Новый заказ: 12345')
```
Модуль получения сообщений (Consumer)
```python
import pika
def callback(ch, method, properties, body):
    print(f"Получено сообщение: {body.decode()}")
def receive_messages(queue_name):
    connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
    channel = connection.channel()
    channel.queue_declare(queue=queue_name)  # Убедитесь, что очередь существует
    channel.basic_consume(queue=queue_name, on_message_callback=callback, auto_ack=True)
    print('Ожидание сообщений. Для выхода нажмите CTRL+C')
    channel.start_consuming()
if __name__ == "__main__":
    receive_messages('order_queue')
```

4. Тестирование обмена сообщениями

1. Запустите Consumer: В одном терминале запустите модуль получения сообщений.
2. Отправьте сообщение: В другом терминале запустите модуль отправки сообщений.

Вы должны увидеть, как Consumer получает отправленное сообщение.

5. Обработка ошибок

Важно обрабатывать возможные ошибки, такие как проблемы с подключением к RabbitMQ или ошибки при отправке/получении сообщений. Используйте блоки `try-except` для обработки исключений.

```python
try:
    send_message('order_queue', 'Новый заказ: 12345')
except Exception as e:
    print(f"Ошибка при отправке сообщения: {e}")
```

ПРАКТИЧЕСКАЯ РАБОТА 15. ОРГАНИЗАЦИЯ ФАЙЛОВОГО ВВОДА-ВЫВОДА ДАННЫХ

Организация файлового ввода-вывода (I/O) данных — это важный аспект разработки программного обеспечения, позволяющий приложениям взаимодействовать с файловой системой. В этом проекте мы рассмотрим основные методы работы с файлами на Python, включая чтение, запись, и обработку ошибок.

1. Основы работы с файлами

Файлы могут быть открыты в различных режимах, в зависимости от того, что вы хотите с ними сделать:

- Чтение: `r` — открывает файл только для чтения.
- Запись: `w` — открывает файл для записи, очищая его содержимое. Если файл не существует, он будет создан.
-Добавление: `a` — открывает файл для добавления данных в конец. Если файл не существует, он будет создан.
- Чтение и запись: `r+` — открывает файл для чтения и записи.

2. Чтение из файла

Пример чтения данных из текстового файла:

```python
def read_file(file_path):
 try:
 with open(file_path, 'r') as file:
 content = file.read() # Чтение всего содержимого файла
 print(content)
 except FileNotFoundError:
 print(f"Файл {file_path} не найден.")
 except IOError:
 print("Произошла ошибка при чтении файла.")
if __name__ == "__main__":
 read_file('example.txt')
```

3. Запись в файл

Пример записи данных в текстовый файл:

```python
def write_file(file_path, data):
 try:
 with open(file_path, 'w') as file:
 file.write(data) # Запись данных в файл
 print(f"Данные записаны в файл {file_path}.")
 except IOError:
 print("Произошла ошибка при записи в файл.")
if __name__ == "__main__":
 write_file('example.txt', 'Привет, мир!')
```

4. Добавление данных в файл
Пример добавления данных в уже существующий файл:
```python
def append_to_file(file_path, data):
 try:
 with open(file_path, 'a') as file:
 file.write(data + '\n') # Добавление данных в конец файла
 print(f"Данные добавлены в файл {file_path}.")
 except IOError:
 print("Произошла ошибка при добавлении данных в файл.")
if __name__ == "__main__":
 append_to_file('example.txt', 'Это новая строка.')
```
5. Чтение файла построчно
Если файл большой, вы можете читать его построчно:
```python
def read_file_line_by_line(file_path):
 try:
 with open(file_path, 'r') as file:
 for line in file:
 print(line.strip()) # Удаление лишних пробелов и символов новой строки
 except FileNotFoundError:
 print(f"Файл {file_path} не найден.")
 except IOError:
 print("Произошла ошибка при чтении файла.")

if __name__ == "__main__":
 read_file_line_by_line('example.txt')
```
6. Обработка ошибок
Важно обрабатывать ошибки, которые могут возникнуть при работе с файлами. Используйте блоки `try-except`, чтобы перехватывать исключения, такие как `FileNotFoundError` и `IOError`.

ПРАКТИЧЕСКАЯ РАБОТА 16. РАЗРАБОТКА МОДУЛЕЙ ЭКСПЕРТНОЙ СИСТЕМЫ

Экспертные системы — это программы, которые используют знания и правила для решения сложных задач, обычно в специфической области. Они имитируют процесс принятия решений человека-эксперта. В этом проекте мы рассмотрим основные компоненты и модули, необходимые для разработки экспертной системы на Python.

1. Архитектура экспертной системы

Экспертные системы обычно состоят из следующих компонентов:

- База знаний: Хранит факты и правила, используемые для принятия решений.

- Механизм вывода: Обрабатывает информацию из базы знаний и делает выводы.

- Интерфейс пользователя: Позволяет пользователю взаимодействовать с системой.

2. База знаний

База знаний может быть реализована с использованием простых структур данных, таких как списки и словари. Например, можно создать базу фактов и правил в виде словаря:

```python
knowledge_base = {
 'правила': {
 'если температура высокая и влажность высокая, то вероятность дождя высокая': True,
 'если температура низкая, то вероятность заморозков высокая': True
 },
 'факты': {
 'температура': 30,
 'влажность': 80
 }
}
```

3. Механизм вывода

Механизм вывода отвечает за интерпретацию правил и фактов. Он может использовать простую логику для принятия решений. Пример реализации механизма вывода:

```python
def infer(knowledge_base):
 for rule in knowledge_base['правила']:
 conditions = rule.split(' и ')
 if all(knowledge_base['факты'].get(cond.split()[1], 0) > int(cond.split()[-1]) for cond in conditions):
```

```
 print(f"Вывод: {rule} - Верно")
 else:
 print(f"Вывод: {rule} - Неверно")
if __name__ == "__main__":
 infer(knowledge_base)
```

4. Интерфейс пользователя

Интерфейс пользователя может быть реализован в виде консольного приложения или графического интерфейса. Для простоты мы рассмотрим консольный интерфейс:

```python
def user_interface():
 print("Экспертная система по прогнозированию погоды")
 temperature = int(input("Введите температуру: "))
 humidity = int(input("Введите влажность: "))
 knowledge_base['факты']['температура'] = temperature
 knowledge_base['факты']['влажность'] = humidity
 infer(knowledge_base)
if __name__ == "__main__":
 user_interface()
```

5. Расширение базы знаний

Одним из преимуществ экспертных систем является возможность расширения базы знаний. Вы можете добавлять новые правила и факты, чтобы улучшить точность системы. Например, добавление новых правил может выглядеть следующим образом:

```python
knowledge_base['правила']['если ветер сильный, то вероятность дождя высокая'] = True
```

6. Обработка ошибок

Важно обрабатывать возможные ошибки, такие как некорректный ввод данных пользователем. Используйте блоки `try-except` для обработки исключений:

```python
try:
 temperature = int(input("Введите температуру: "))
 humidity = int(input("Введите влажность: "))
except ValueError:
 print("Пожалуйста, введите числовые значения.")
```

ПРАКТИЧЕСКАЯ РАБОТА 17. СОЗДАНИЕ СЕТЕВОГО СЕРВЕРА И СЕТЕВОГО КЛИЕНТА

Создание сетевого сервера и клиента — это важный аспект разработки приложений, которые требуют обмена данными по сети. В этом проекте мы рассмотрим, как создать простой сервер и клиента на Python с использованием библиотеки `socket`. Это позволит вам понять основы сетевого программирования.

1. Создание сетевого сервера

Сетевой сервер слушает входящие соединения от клиентов и обрабатывает их. Вот пример простого сервера, который принимает сообщения от клиентов и отправляет ответ.

```
python
import socket
def start_server(host='127.0.0.1', port=65432):
 # Создание сокета
 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
 server_socket.bind((host, port))
 server_socket.listen() # Ожидание подключения клиента
 print(f"Сервер запущен на {host}:{port}")
 while True:
 conn, addr = server_socket.accept() # Принять соединение
 with conn:
 print(f"Подключено к {addr}")
 data = conn.recv(1024) # Получить данные от клиента
 if not data:
 break
 print(f"Получено: {data.decode()}")
 conn.sendall(b'Сообщение получено!') # Отправить ответ клиенту
if __name__ == "__main__":
 start_server()
```

2. Создание сетевого клиента

Сетевой клиент подключается к серверу и отправляет сообщения. Вот пример простого клиента:

```
`python
import socket
def start_client(host='127.0.0.1', port=65432):
 # Создание сокета
 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
 client_socket.connect((host, port)) # Подключение к серверу
 message = input("Введите сообщение для отправки на сервер: ")
```

```
client_socket.sendall(message.encode()) # Отправить сообщение
data = client_socket.recv(1024) # Получить ответ от сервера
print(f"Ответ от сервера: {data.decode()}")
if __name__ == "__main__":
start_client()
```

3. Запуск сервера и клиента

1. Сначала запустите сервер. Он будет ожидать подключения клиентов.

2. Затем запустите клиента. Введите сообщение, и клиент отправит его на сервер.

3. Сервер получит сообщение и отправит ответ обратно клиенту.

4. Обработка ошибок

Важно обрабатывать возможные ошибки, такие как проблемы с сетью или неправильные адреса. Используйте блоки `try-except` для обработки исключений:

```
python
try:
client_socket.connect((host, port))
except ConnectionRefusedError:
print("Не удалось подключиться к серверу. Убедитесь, что сервер запущен.")
```

5. Расширение функциональности

Вы можете расширить функциональность сервера и клиента, добавив:

- Поддержку нескольких клиентов (используя потоки или асинхронное программирование).
- Протокол обмена сообщениями (например, JSON или XML).
- Логику обработки различных команд от клиентов.

СПИСОК ЛИТЕРАТУРЫ

1. Шпаковский В.О. PR-дизайн и PR-продвижение [Электронный ресурс] : учебное пособие / В.О. Шпаковский, Е.С. Егорова. — Электрон. текстовые данные. — М. : Инфра-Инженерия, 2018. — 452 c. — 978-5-9729-0217-0. — Режим доступа: http://www.iprbookshop.ru/78249.htmlКелим, Ю.М. Вычислительная техника [Текст]: учебник для студ. учреждений сред. проф. образования / Ю.М.Келим.- М.: Академия, 2017.- 368с.

2. Пигулевский В.О. Дизайн визуальных коммуникаций [Электронный ресурс] : учебное пособие / В.О. Пигулевский, А.Ф. Стефаненко. — Электрон. текстовые данные. — Саратов: Вузовское образование, 2018. — 303 c. — 978-5-4487-0267-9. — Режим доступа: http://www.iprbookshop.ru/75951.html

3. Единое окно доступа к образовательным ресурсам. Милованов И.В. Основы разработки программного обеспечения вычислительных систем: учебное пособие / И.В. Милованов, В.И. Лоскутов. - Тамбов: Изд-во ГОУ ВПО ТГТУ, 2011. - 88 с.

4. Извозчикова В.В. Эксплуатация и диагностирование технических и программных средств информационных систем [Электронный ресурс] : учебное пособие / В.В. Извозчикова. — Электрон. текстовые данные. — Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2017. — 137 c. — 978-5-7410-1746-3. — Режим доступа: http://www.iprbookshop.ru/71353.html

ШОВКАРОВА Зарина Сейтбиевна

РАЗРАБОТКА КОДА ИНФОРМАЦИОННЫХ СИСТЕМ

ПРАКТИКУМ
для студентов III курса специальности
09.02.07 Информационные системы и программирование

Корректор Джукаев У.М.
Редактор Джукаев У.М

Сдано в набор 23.04.2025 г.
Формат 60х84/16
Бумага офсетная.
Печать офсетная.
Усл. печ. л. 2,44
Заказ № 5083
Тираж 100 экз.

Оригинал-макет подготовлен
в Библиотечно-издательском центре СКГА
369000, г. Черкесск, ул. Ставропольская,