**Перечень вариантов заданий для подготовки к экзамену**

**МДК.02.01 Технология разработки программного обеспечения**

[Разработать диаграмму деятельности программного обеспечения по варианту задания. 1](#_Toc202393473)

[Разработать функциональную схему программного обеспечения по варианту задания. 2](#_Toc202393474)

[Разработать диаграмму прецедентов программного обеспечения по варианту задания. 3](#_Toc202393475)

[Разработать диаграмму последовательности программного обеспечения по варианту задания. 4](#_Toc202393476)

[Разработать диаграмму потоков данных программного обеспечения по варианту задания. 5](#_Toc202393477)

[Разработать ER-диаграмму в 3-НФ для программного обеспечения по варианту задания. 7](#_Toc202393478)

[1. Структурное программирование 8](#_Toc202393479)

[2. Модульное программирование 8](#_Toc202393480)

[data.py (хранение данных) 9](#_Toc202393481)

[calculations.py (расчеты) 9](#_Toc202393482)

[report.py (формирование отчета) 9](#_Toc202393483)

[main.py (основная программа) 9](#_Toc202393484)

[3. Объектно-ориентированное программирование 10](#_Toc202393485)

# Разработать диаграмму деятельности программного обеспечения по варианту задания.

Что такое Диаграмма деятельности (или диаграмма активностей)?

— это один из видов диаграмм, используемых в нотации UML (Unified Modeling Language) для моделирования бизнес-процессов и систем. Она описывает последовательность действий и поток управления в процессе, а также взаимодействие между различными участниками (акторами) и системами.

Основные элементы диаграммы деятельности включают:

Действия — представляют собой отдельные шаги или задачи в процессе.

Потоки управления — показывают порядок выполнения действий и переходы между ними.

Решения — точки, где процесс может разветвляться в зависимости от условий.

Начало и конец — обозначают старт и завершение процесса.

Параллельные действия — показывают, что несколько действий могут выполняться одновременно.

Пример диаграммы по нашему варианту задания –

Изображение выглядит как текст, чек, снимок экрана, Параллельный

Содержимое, созданное искусственным интеллектом, может быть неверным.

# Разработать функциональную схему программного обеспечения по варианту задания.

Что такое Функциональная схема?

Функциональная схема — это графическое представление системы в виде блоков, отображающих основные компоненты и их взаимодействие. Она показывает:

Какие модули есть в системе

Какие функции они выполняют

Как между ними передаются данные и управление

Отличается от диаграммы деятельности тем, что тут акцент делается не на процессах, а на структурных элементах и связях между ними.

Пример диаграммы по нашему варианту задания –

Изображение выглядит как текст, линия, снимок экрана, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

Пояснение:

Прямоугольники — модули системы.

Стрелки — направление обмена данными (SQL-запросы, файлы, отчеты).

База данных вынесена отдельно как ключевой компонент.

# Разработать диаграмму прецедентов программного обеспечения по варианту задания.

Что такое диаграмма прецедентов?

Диаграмма прецедентов — это UML-диаграмма, которая показывает:

Акторов (роли пользователей или внешних систем)

Прецеденты (функции/сценарии использования)

Связи между ними

Основная цель: наглядное представление функциональных возможностей системы с точки зрения пользователя.

Пример диаграммы по нашему варианту задания –

Акторы:

Администратор (управляет данными)

Преподаватель (просматривает свою нагрузку)

БД (пассивный актор – хранит данные)

Прецеденты (основные функции):

Управление профилями преподавателей

Распределение учебной нагрузки

Формирование отчетов

Загрузка фотографий

Изображение выглядит как текст, линия, диаграмма, Параллельный

Содержимое, созданное искусственным интеллектом, может быть неверным.

Пояснение элементов:

Акторы (Actors)

Администратор — основной пользователь системы (полный доступ).

Преподаватель — ограниченный доступ (просмотр своих данных).

База данных — пассивный участник (хранение и выдача данных).

Прецеденты (Use Cases)

Группируются внутри прямоугольника системы.

Например: "Добавить преподавателя", "Сформировать отчет".

Связи:

Сплошная стрелка → Актор взаимодействует с прецедентом.

Пунктирная стрелка (..>) → Прецедент использует данные БД.

# Разработать диаграмму последовательности программного обеспечения по варианту задания.

Что такое диаграмма последовательности?

Диаграмма последовательности — это UML-диаграмма, которая показывает, как объекты взаимодействуют друг с другом в определенном сценарии, а также порядок вызовов и сообщений между ними. Она помогает визуализировать динамику системы и последовательность выполнения операций.

Пример диаграммы по нашему варианту задания –

Пользователь (Администратор) — инициирует процесс.

Система — программное обеспечение, обрабатывающее запросы.

База данных — хранит информацию о преподавателях и их нагрузке.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Пояснение элементов:

Акторы и участники:

Администратор — инициатор процесса.

Система — программное обеспечение, обрабатывающее запросы.

База данных — хранит информацию о преподавателях и их нагрузке.

Сообщения:

Стрелки показывают направление сообщений между участниками.

Например, администратор вводит данные, система сохраняет их в базе данных.

Ответы:

Стрелки с обратным направлением (например, db --> system) показывают подтверждение выполнения операций.

# Разработать диаграмму потоков данных программного обеспечения по варианту задания.

Что такое диаграмма потоков?

Диаграмма потоков данных визуализирует:

Процессы обработки данных (кружки)

Хранилища данных (две параллельные линии)

Потоки данных между ними (стрелки с подписями)

Внешние сущности (источники/получатели данных)

Уровни:

Контекстная (0 уровень) – один процесс и его связи с внешними сущностями

Декомпозиция (1+ уровень) – детализация процессов

Контекстная DFD (Уровень 0)

Главный процесс:

Система учета преподавателей

Внешние сущности:

Администратор (вносит/редактирует данные)

БД (хранит структурированные данные)

Файловое хранилище (сохраняет фотографии)

Уровень 1 (Декомпозиция основного процесса)

Ключевые подпроцессы:

1.0 – Управление профилями

2.0 – Распределение нагрузки

3.0 – Генерация отчетов

Пример диаграммы по нашему варианту задания –

0 Уровень –

Изображение выглядит как текст, диаграмма, линия, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

1 Уровень –

Изображение выглядит как текст, диаграмма, линия, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

# Разработать ER-диаграмму в 3-НФ для программного обеспечения по варианту задания.

Что такое ER-диаграмма?

ER-диаграмма — это графическое представление сущностей, их атрибутов и взаимосвязей между ними. Она помогает визуализировать структуру базы данных и организовать данные в соответствии с нормализацией.

Нормализация — это процесс организации данных для уменьшения избыточности и зависимости. 3-Нормальная Форма (3-НФ) требует, чтобы:

Сущности были в 2-НФ.

Все неключевые атрибуты зависели только от первичного ключа.

Пример диаграммы по нашему варианту задания –

Изображение выглядит как текст, Шрифт, диаграмма, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Система учета преподавателей колледжа**

Входные данные:

Личные данные преподавателя (ФИО, дата рождения, адрес, телефон).

Сведения о семестре: предметы, группы, кабинеты.

Фотография (файл в формате JPEG/PNG).

**Функционал:**

Добавление/редактирование профиля: Внесение анкетных данных, загрузка фото.

Управление нагрузкой: Распределение групп и кабинетов на семестр.

Отчеты: Формирование списка преподавателей с их нагрузкой (PDF/Excel).

**Как работает:**

Данные хранятся в SQL-БД (таблицы: Преподаватели, Семестры, Предметы).

Для загрузки фото используется хранение ссылок на файлы.

Фильтрация по группам/предметам через SQL-запросы.

**Пример использования:**

> Добавить преподавателя: Иванов А.С., р. 15.03.1980, тел. +7-XXX-XXX-XX-XX.

> Назначить на семестр: Математика, группы П-21, каб. 305.

> Сформировать отчет: "Преподаватели весеннего семестра 2024".

1. Вариант задания реализуйте в одной из трёх парадигм: структурной, модульной и объектно-ориентированной

**ПП 'Зарплата'**

Бухгалтерия: почасовая оплата, табель, ставки, расчёт зарплаты.

1. Структурное программирование:

Реализуйте задачу в виде линейной программы, используя списки и функции без классов.

# Данные сотрудников

employees = [

{"id": 1, "name": "Иванов И.И.", "hourly\_rate": 500},

{"id": 2, "name": "Петров П.П.", "hourly\_rate": 450},

{"id": 3, "name": "Сидоров С.С.", "hourly\_rate": 550}

]

# Данные табеля учета времени (id сотрудника, часы работы)

timesheet = [

(1, 160),

(2, 145),

(3, 180)

]

# Функция расчета зарплаты

def calculate\_salary(employee\_id, hours\_worked):

employee = next(emp for emp in employees if emp["id"] == employee\_id)

return hours\_worked \* employee["hourly\_rate"]

# Функция формирования отчета

def generate\_report():

print("Отчет по зарплатам")

print("==================")

total\_payment = 0

for entry in timesheet:

employee\_id, hours = entry

employee = next(emp for emp in employees if emp["id"] == employee\_id)

salary = calculate\_salary(employee\_id, hours)

total\_payment += salary

print(f"{employee['name']}: {hours} часов × {employee['hourly\_rate']} руб./час = {salary} руб.")

print("==================")

print(f"Итого к выплате: {total\_payment} руб.")

# Основная программа

if \_\_name\_\_ == "\_\_main\_\_":

generate\_report()

1. Модульное программирование:

Разделите функциональность на модули (например: обработка данных, расчёты, отчёт).

Разделим программу на несколько модулей:

### data.py (хранение данных)

employees = [

{"id": 1, "name": "Иванов И.И.", "hourly\_rate": 500},

{"id": 2, "name": "Петров П.П.", "hourly\_rate": 450},

{"id": 3, "name": "Сидоров С.С.", "hourly\_rate": 550}

]

timesheet = [

(1, 160),

(2, 145),

(3, 180)

]

### calculations.py (расчеты)

def calculate\_salary(employee\_id, hours\_worked, employees):

employee = next(emp for emp in employees if emp["id"] == employee\_id)

return hours\_worked \* employee["hourly\_rate"]

def calculate\_total\_payment(timesheet, employees):

total = 0

for entry in timesheet:

employee\_id, hours = entry

total += calculate\_salary(employee\_id, hours, employees)

return total

### report.py (формирование отчета)

def generate\_report(employees, timesheet):

from calculations import calculate\_salary, calculate\_total\_payment

print("Отчет по зарплатам")

print("==================")

for entry in timesheet:

employee\_id, hours = entry

employee = next(emp for emp in employees if emp["id"] == employee\_id)

salary = calculate\_salary(employee\_id, hours, employees)

print(f"{employee['name']}: {hours} часов × {employee['hourly\_rate']} руб./час = {salary} руб.")

print("==================")

total = calculate\_total\_payment(timesheet, employees)

print(f"Итого к выплате: {total} руб.")

### main.py (основная программа)

from data import employees, timesheet

from report import generate\_report

if \_\_name\_\_ == "\_\_main\_\_":

generate\_report(employees, timesheet)

1. Объектно-ориентированное программирование:

Создайте соответствующие классы (например: Сотрудник, Ставка, Нагрузка) и реализуйте методы работы с данными.

class Employee:

def \_\_init\_\_(self, id, name, hourly\_rate):

self.id = id

self.name = name

self.hourly\_rate = hourly\_rate

def calculate\_salary(self, hours):

return self.hourly\_rate \* hours

class TimesheetEntry:

def \_\_init\_\_(self, employee, hours):

self.employee = employee

self.hours = hours

def get\_salary(self):

return self.employee.calculate\_salary(self.hours)

class Payroll:

def \_\_init\_\_(self):

self.employees = []

self.timesheet = []

def add\_employee(self, employee):

self.employees.append(employee)

def add\_timesheet\_entry(self, employee\_id, hours):

employee = next(emp for emp in self.employees if emp.id == employee\_id)

self.timesheet.append(TimesheetEntry(employee, hours))

def generate\_report(self):

print("Отчет по зарплатам")

print("==================")

total\_payment = 0

for entry in self.timesheet:

salary = entry.get\_salary()

total\_payment += salary

print(f"{entry.employee.name}: {entry.hours} часов × {entry.employee.hourly\_rate} руб./час = {salary} руб.")

print("==================")

print(f"Итого к выплате: {total\_payment} руб.")

# Основная программа

if \_\_name\_\_ == "\_\_main\_\_":

payroll = Payroll()

# Добавляем сотрудников

payroll.add\_employee(Employee(1, "Иванов И.И.", 500))

payroll.add\_employee(Employee(2, "Петров П.П.", 450))

payroll.add\_employee(Employee(3, "Сидоров С.С.", 550))

# Добавляем записи табеля

payroll.add\_timesheet\_entry(1, 160)

payroll.add\_timesheet\_entry(2, 145)

payroll.add\_timesheet\_entry(3, 180)

# Формируем отчет

payroll.generate\_report()