МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Астраханский государственный университет имени В.Н. Татищева»
(Астраханский государственный университет им. В.Н. Татищева)

Филиал АГУ им. В.Н. Татищева в г. Знаменске Астраханской области

СОГЛАСОВАНО
Руководитель ОПОП
Бориско С.Н.
«13» ноября 2025 г.

УТВЕРЖДАЮ
Председатель ЦК (МО)
Фисенко Т.Ю.
протокол заседания ЦК (МО) №3
от «13» ноября 2025 г.

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ
по учебной дисциплине

Основы алгоритмизации и программирования

Составитель	Бориско С.Н., к.т.н., доцент, завкафедрой ЗнМИ; Мустафаев Н.Г., к.т.н., доцент кафедры ЗнМИ; Тимошкин А.А., к.т.н., доцент кафедры ЗнМИ; Устинов А.С., к.т.н., доцент кафедры ЗнМИ; Каштанов Д.Ю., ассистент кафедры ЗнМИ
Согласовано с работодателями	Литвинов С.П., к.т.н., заместитель командира войсковой части 15644 по научно-исследовательской и испытательной работе; Кирьянов М.Н., ведущий инженер ПАО «Ростелеком»
Наименование специальности	09.02.12 Техническая эксплуатация и сопровождение информационных систем
Квалификация выпускника	Специалист по технической эксплуатации и сопровождению информационных систем
Форма обучения	очная
Год приема (курс)	2026 (2 курс)

Знаменск, 2025 г.

СОДЕРЖАНИЕ

1. Общие положения

Фонд оценочных средств (далее – ФОС) предназначен для контроля и оценки результатов освоения обучающимися учебной дисциплины «Основы алгоритмизации и программирования».

ФОС включают контрольные материалы для проведения текущего контроля успеваемости и промежуточной аттестации обучающихся, разработанные в соответствии с требованиями ФГОС СПО и содержанием рабочей программы учебной дисциплины.

2. Результаты освоения учебной дисциплины, подлежащие проверке

Код компетенции	Планируемые результаты освоения учебной дисциплины		
	Практический опыт	Умения	Знания
ОК 2	-способен применять теоретические знания на практике при работе с различными операционными системами; -умеет анализировать и решать задачи системного администрирования; - готов к освоению новых технологий в области операционных систем и сред.	-распознавать задачу и/или проблему в профессиональном и/или социальном контексте, анализировать и выделять её составные части	-актуальный профессиональный и социальный контекст, в котором приходится работать и жить

3. Распределение оценивания результатов обучения по видам контроля

Наименование элемента практического опыта, умений или знаний	Наименование оценочного средства текущего контроля и промежуточной аттестации	
	Текущий контроль	Промежуточная аттестация
ПО.1. способен применять теоретические знания на практике при работе с различными операционными системами ПО.2. умеет анализировать и решать задачи системного администрирования; ПО.3 готов к освоению новых технологий в области операционных систем и сред	Компьютерное тестирование на знание терминологии по теме. Контрольные задания, решение задач по теме.	Вопросы к зачету
У1 распознавать задачу и/или проблему в профессиональном и/или социальном контексте, анализировать и выделять её составные части		

31 актуальный профессиональный и социальный контекст, в котором приходится работать и жить		

4. Контрольные задания для оценки результатов освоения учебной дисциплины

4.1. Контрольные задания для текущего контроля

Раздел 1. Технологии программирования

Тестовые задания

1. (ОВ) Основная цель модульного программирования — это:
a) Создание программ с использованием графического интерфейса
b) Разделение программы на небольшие, независимые и легко управляемые части (модули)
c) Написание всего кода в одной главной функции
d) Оптимизация программы для максимальной скорости выполнения

2. (МВ) Какие из перечисленных преимуществ обеспечивает модульный подход?
a) Упрощение тестирования (можно тестировать модули по отдельности)
b) Повторное использование кода
c) Упрощение отладки и внесения изменений
d) Гарантированное отсутствие ошибок в программе

3. (СО) Высокоуровневый план или структура программы, показывающая, как модули взаимодействуют друг с другом, называется _____ программы.
Ответ: _____

4. (ОВ) Хороший модуль должен обладать:
a) Сильной связностью (high cohesion) и слабой связанностью (low coupling)
b) Слабой связностью и сильной связанностью
c) Максимальным количеством зависимостей от других модулей
d) Возможностью выполнять множество разнородных задач

5. (СО) Принцип, согласно которому детали реализации модуля скрыты от остальной части программы, и доступ к его функциональности осуществляется только через четко определенный интерфейс, называется _____.
Ответ: _____

6. (ОВ) Объектно-ориентированное программирование (ООП) — это парадигма, основанная на концепции:
a) Последовательного выполнения инструкций
b) **Объектов**, которые содержат данные (поля) и методы для работы с этими данными
c) Исключительно на использовании функций (подпрограмм)
d) Потоков данных

7. (МВ) Какие из следующих утверждений об «инкапсуляции» верны?
a) Это механизм объединения данных и методов, которые работают с этими данными, в одной единице (классе).

b) Это механизм сокрытия внутреннего состояния объекта и деталей реализации от внешнего мира.
c) Это позволяет изменять реализацию класса, не затрагивая код, который его использует.
d) Это синоним наследования.

8. (CO) «Чертеж» или шаблон для создания объектов, который определяет их свойства (атрибуты) и поведение (методы), называется _____.
Ответ: _____

9. (OB) Модификатор доступа (в таких языках, как Java, C++, C#), который делает член класса доступным только внутри этого же класса, — это:
a) public
b) protected
c) private
d) package (default)

10. (CO) Специальный метод класса, который автоматически вызывается при создании нового объекта и обычно используется для инициализации полей объекта, называется _____.
Ответ: _____

11. (MB) Какие утверждения о наследовании в ООП верны?
a) Это механизм, позволяющий одному классу (дочернему, производному) наследовать свойства и методы другого класса (родительского, базового).
b) Он способствует повторному использованию кода.
c) В языках, поддерживающих множественное наследование, класс может иметь несколько прямых родительских классов.
d) Наследование реализует отношение «является» (is-a).

12. (OB) Принцип, согласно которому объекты разных классов могут отвечать на один и тот же вызов метода (сообщение), но делать это по-разному, в зависимости от своей реализации, — это:
a) Инкапсуляция
b) Наследование
c) **Полиморфизм**
d) Абстракция

13. (CO) В ООП полиморфизм, достигаемый за счет переопределения (override) методов в дочерних классах, называется _____ полиморфизмом.
Ответ: _____

14. (OB) Абстрактный класс — это класс, который:
a) Содержит только данные и не имеет методов
b) Нельзя наследовать
c) Нельзя инстанциировать (создавать его объекты напрямую), он предназначен для наследования
d) Автоматически создает свои объекты

15. (MB) Какие из следующих концепций связаны с принципом «абстракции» в ООП?
a) Выделение существенных характеристик объекта, игнорируя несущественные детали.
b) Создание интерфейсов или абстрактных классов, которые определяют контракт, но не реализацию.

c) Объединение данных и методов в одну капсулу.
d) Позволяет работать с объектами, не вдаваясь в детали их внутреннего устройства.

16. (CO) Принцип проектирования, который гласит, что класс должен быть открыт для расширения, но закрыт для модификации, известен как принцип _____ (аббревиатура OCP).
Ответ (полное название на английском): _____

17. (OB) Интерфейс в контексте ООП — это:
a) Графический пользовательский интерфейс (GUI)
b) Контракт, который определяет набор методов (их сигнатуры), которые должен реализовать класс
c) Способ соединения двух физических устройств
d) Внутренняя реализация класса

18. (CO) Отношение между классами, когда один класс (часть) является полем другого класса (целое), описывает принцип «имеет» (has-a) и называется _____.
Ответ: _____

19. (MB) Какие из перечисленных языков программирования в значительной степени поддерживают парадигму ООП?
a) Java
b) C#
c) Python
d) Pascal (в классическом виде)

20. (CO) Конкретный экземпляр класса, обладающий конкретными значениями атрибутов и занимающий место в памяти, называется _____
Ответ: _____

Ключ для проверки

Раздел 1:

1. **b)** Разделение программы на небольшие, независимые и легко управляемые части (модули)
2. **a, b, c** (d — модульность не гарантирует отсутствие ошибок)
3. **архитектура** (software architecture) или **структура**
4. **a)** Сильной связностью (high cohesion) и слабой связанностью (low coupling)
5. **сокрытие информации** (information hiding) или **инкапсуляция** (в общем смысле)

Раздел 2:
6. **b) Объектов**, которые содержат данные (поля) и методы для работы с этими данными
7. **a, b, c** (d — неверно, это разные концепции)
8. **класс** (class)
9. **c)** private
10. **конструктор** (constructor)

Раздел 3:
11. **a, b, c, d** (Все верны, но с верно не для всех языков, например, Java не поддерживает множественное наследование классов)
12. **c) Полиморфизм**
13. **динамическим** (динамический полиморфизм, полиморфизм времени выполнения)

14. **c)** Нельзя инстанциировать (создавать его объекты напрямую), он предназначен для наследования

Раздел 4:
15. **a, b, d** (c — это определение инкапсуляции)
16. **Open/Closed Principle** (Принцип открытости/закрытости)
17. **b)** Контракт, который определяет набор методов (их сигнатуры), которые должен реализовать класс
18. **агрегация** или **композиция** (композиция — более строгий вид отношения «часть-целое»)

Раздел 5:
19. **a, b, c** (Pascal — процедурный язык, хотя современные диалекты могут иметь ООП-расширения)
20. **объект** (object) или **экземпляр класса** (instance)

Контрольные задания

Блок 1: Модульное программирование и проектирование

Задание 1. «Рефакторинг монолитного кода в модульную структуру»
Дан код программы для управления библиотекой (на Python, но концепция универсальна). Весь код находится в одном файле library_monolith.py (~200 строк) и содержит функции для работы с книгами, читателями, выдачей книг, а также вывод меню и логику основного цикла.

Исходный код (фрагмент для понимания проблемы):

python
```python
# library_monolith.py
books = []
readers = []
loans = []

def add_book(title, author):
    # ... 20 строк кода, включая проверки и взаимодействие с глобальными переменными
    pass

def find_reader_by_name(name):
    # ... 15 строк кода
    pass

def loan_book(reader_id, book_id):
    # ... 30 строк кода, обращается к books, readers, loans
    pass

def generate_report():
    # ... 25 строк кода, формирует отчет из всех данных
    pass
```

```python
# Основной цикл программы с меню на 50 строк
def main():
    while True:
        print("1. Добавить книгу")
        # ... и т.д.
        choice = input()
        if choice == "1":
            title = input("Введите название: ")
            # ... вызов add_book и т.д.
```

Задача:

1. **Анализ связности и связанности:** Выделите в монолите отдельные функциональные области (сущности). Определите, какие функции сильно связаны между собой (высокая связность) и какие глобальные данные они используют.
2. **Проектирование модулей:** Предложите структуру проекта из 4-5 модулей (файлов .py). Например:
   - book_management.py (функции для работы с книгами)
   - reader_management.py
   - loan_management.py
   - storage.py (работа с данными: загрузка, сохранение в файл/БД)
   - ui.py (меню и ввод/вывод)
   - main.py (главный цикл)
3. **Разработка интерфейсов:** Для каждого модуля определите «публичный интерфейс» — список функций, которые будут импортироваться в другие модули. Опишите, какие данные будут передаваться между модулями (аргументы, возвращаемые значения), чтобы минимизировать связанность (использовать структуры данных, а не глобальные переменные).
4. **План рефакторинга:** Составьте пошаговый план преобразования кода (например: 1) Создать пустые модули; 2) Перенести функции add_book, find_book в book_management.py; 3) Изменить их сигнатуры, чтобы они принимали и возвращали данные, а не использовали глобальный список books; и т.д.).

Задание 2. «Проектирование модульной системы плагинов для текстового редактора»

Требуется спроектировать архитектуру простого текстового редактора, поддерживающего плагины (например, для проверки орфографии, подсветки синтаксиса, статистики).

Требуется:

1. **Определение ядра:** Опишите, какие функции будут входить в **ядро системы** (core module). Что оно должно предоставлять плагинам? (Например, API для доступа к тексту, уведомления об изменениях, регистрацию пунктов меню).
2. **Протокол взаимодействия:** Разработайте **контракт (интерфейс)** для плагинов. Что должен реализовать каждый плагин? (Например, обязательная функция initialize(editor_api), get_name(), опциональная on_text_changed()).
3. **Менеджер плагинов:** Спроектируйте модуль plugin_manager.py. Опишите его функции:
   - Обнаружение плагинов (сканирование папки).
   - Загрузка и инициализация.
   - Управление жизненным циклом (включение/отключение).
   - Маршрутизация событий от ядра к плагинам.

4. **Пример плагина:** Напишите псевдокод (или код на Python) для простого плагина «Статистика», который по команде показывает количество слов в документе, используя предоставленное ядром API.

Блок 2: Объектно-ориентированное программирование (ООП)

Задание 3. «Моделирование предметной области "Банковская система" с помощью ООП»
Необходимо смоделировать следующие
сущности: **Банк**, **Клиент**, **Счет** (Account), **Транзакция**.

Требуется:

1. **Выделение классов и их атрибутов:**
   - Для каждого класса определите поля (атрибуты) и типы данных. Например:
     - Client: id, name, address, list_of_accounts.
     - Account: number, balance, owner (ссылку на Client), transaction_history.
   - Укажите модификаторы доступа для полей (private/protected/public).
2. **Определение методов и инкапсуляции:**
   - Для класса Account разработайте методы: deposit(amount), withdraw(amount), get_balance().
   - Объясните, как инкапсуляция защищает целостность данных (например, запрет прямого изменения баланса, проверка на достаточность средств в withdraw).
3. **Установление отношений между классами:**
   - Изобразите **диаграмму классов UML** (можно схематично), показывающую связи: агрегация (Bank — Client), композиция (Client — Account), ассоциацию (Transaction — Account).
4. **Реализация наследования:** Создайте иерархию для разных типов счетов. Базовый класс Account. Производные классы: SavingsAccount (сберегательный, не может уходить в минус, начисляет процент), CheckingAccount (расчетный, допускает овердрафт до определенного лимита). Покажите, какие методы будут переопределены (override).

Задание 4. «Применение полиморфизма для системы рендеринга графических фигур»
Создайте программу, которая работает с коллекцией геометрических фигур (круг, прямоугольник, треугольник), не зная их конкретного типа на этапе компиляции.

Задача:

1. **Создание иерархии:** Определите абстрактный базовый класс Shape с абстрактными методами:
   - calculate_area() -> float
   - calculate_perimeter() -> float
   - draw() (выводит условное изображение в консоль)
2. **Реализация полиморфизма:** Создайте классы-наследники Circle, Rectangle, Triangle, которые реализуют эти методы по-своему (используют разные формулы).
3. **Демонстрация:** В функции main создайте список (массив) типа List[Shape]. Добавьте в него объекты разных типов (Circle(5), Rectangle(3,4), Triangle(3,4,5)).
4. **Написание полиморфного кода:** Напишите цикл, который для каждой фигуры в списке вызывает calculate_area() и draw(). Объясните, как компилятор/интерпретатор во время выполнения (runtime) определяет, какую конкретно реализацию метода вызвать.

5. **Расширяемость:** Покажите силу полиморфизма: добавьте новый класс Square, не изменяя существующий цикл обработки списка фигур. Что нужно сделать, чтобы система заработала с новым типом?

Блок 3: Принципы проектирования (SOLID и DRY)

Задание 5. «Рефакторинг кода с нарушением принципов SOLID»
Дан класс OrderProcessor, который нарушает несколько принципов SOLID.

java

```
// Псевдокод на Java-подобном синтаксисе
class OrderProcessor {
    public void process(Order order) {
        // Принцип единственной ответственности нарушен:
        // Класс делает слишком много.
        this.validateOrder(order);
        this.calculateTotal(order);
        this.applyDiscount(order); // Жестко закодирована логика скидок
        this.saveToDatabase(order);
        this.sendEmailNotification(order); // Жестко закодирован email
        this.generateInvoice(order); // Жестко закодирован формат PDF
    }

    private void applyDiscount(Order order) {
        if (order.getCustomer().isVIP()) {
            // Логика скидки для VIP
        } else if (order.getTotal() > 1000) {
            // Логика скидки на большую сумму
        }
        // При добавлении нового типа скидки нужно менять этот класс.
    }

    // ... другие методы
}
```

Анализ и рефакторинг:

1. **Принцип единственной ответственности (SRP):** На какие отдельные классы/сервисы можно разбить функциональность OrderProcessor? Перечислите их (например, OrderValidator, PricingService, OrderRepository, NotificationService, InvoiceGenerator).
2. **Принцип открытости/закрытости (OCP):** Как переработать метод applyDiscount, чтобы система была открыта для добавления новых типов скидок, но закрыта для модификации этого класса? Предложите паттерн (например, Стратегия — DiscountStrategy). Нарисуйте диаграмму классов.
3. **Принцип инверсии зависимостей (DIP):** Класс зависит от конкретных реализаций (база данных, email, PDF). Как его переписать, чтобы он зависел от абстракций? Напишите интерфейсы, которые он должен принимать в конструкторе (например, INotificationService, IInvoiceGenerator).
4. **Итоговый дизайн:** Напишите псевдокод исправленного класса OrderProcessor, который просто координирует работу внедренных через интерфейсы сервисов.

Задание 6. «Проектирование системы с использованием интерфейсов и композиции»
Задача: Спроектировать систему уведомлений (Notification), которая может отправлять сообщения через разные каналы: Email, SMS, Push, Telegram.

Требуется отказаться от наследования в пользу композиции и интерфейсов.

1. **Неправильный подход (наследование):** Покажите, почему плоха иерархия Notification -> EmailNotification -> SMSNotification (проблема «взрывного» роста классов при комбинации свойств).
2. **Правильный подход (композиция):**

- Спроектируйте интерфейс MessageSender с методом send(message, recipient).
- Создайте классы, реализующие этот интерфейс: EmailSender, SMSSender, PushSender.
- Создайте класс Notification, который в своем поле содержит **список** объектов MessageSender.
- У класса Notification будет метод dispatch(message, recipient), который проходит по всем отправителям в списке и вызывает их метод send.

3. **Гибкость:** Продемонстрируйте гибкость:

- Как легко добавить новый способ отправки (например, TelegramSender)?
- Как создать уведомление, которое отправляется и по email, и по SMS (композиция поведений)?

4. **Принцип DRY (Don't Repeat Yourself):** Где в этой архитектуре может возникнуть дублирование кода (например, логика логирования, повтора при ошибке)? Предложите, как это вынести в отдельный компонент (например, декоратор RetrySender, оборачивающий любой MessageSender).

Блок 4: Комплексный проект

Задание 7. «Разработка ядра игры "Шахматы" с применением всех принципов ООП»
Цель: Спроектировать и частично реализовать ядро шахматной игры без графического интерфейса.

Этапы:

1. **Модель данных (Классы):**

- Спроектируйте классы Board (доска), Square (клетка), Piece (фигура), Player.
- Используйте наследование для фигур: абстрактный Piece, от него — Pawn, King, Queen и т.д.
- Какая фигура должна содержать информацию о своем цвете? Как связаны Board, Square и Piece? (Агрегация/Композиция).

2. **Поведение (Методы и полиморфизм):**

- В абстрактном классе Piece объявите абстрактный метод get_possible_moves(square, board), возвращающий список клеток, куда фигура может пойти.
- Реализуйте этот метод по-разному в каждом классе-фигуре, учитывая правила ее движения.
- В классе Game создайте метод make_move(player, from_square, to_square), который будет проверять валидность хода, используя полиморфный вызов get_possible_moves.

3. **Принципы проектирования:**

- **SRP:** Разделите ответственности. Кто должен проверять шах/мат? (GameRulesValidator). Кто должен вести историю ходов? (MoveHistory).

- **OCP:** Как спроектировать систему, чтобы легко добавить новую, нестандартную фигуру (например, Chancellor из шахмат Капабланки) или новое правило? (Паттерн Посетитель для правил?).

4. **Модульность:**

- Разбейте проект на модули: core/ (основные классы), rules/ (логика проверок), utils/ (вспомогательные функции). Нарисуйте схему зависимостей между модулями.

5. **Реализация (частичная):** Напишите код для классов Board, Piece, Pawn и метода make_move для простого случая (ход пешкой вперед на пустую клетку).

Критерии оценки:

- **Блок 1 (Модульность):** Оценивается качество анализа монолита, логичность разбиения на модули, четкость определения интерфейсов и практичность плана рефакторинга.
- **Блок 2 (ООП-моделирование):** Оценивается корректность выделения классов и их отношений, глубина понимания инкапсуляции, наследования и демонстрация работы полиморфизма на практических примерах.
- **Блок 3 (Принципы SOLID/DRY):** Оценивается умение выявлять нарушения принципов, предлагать грамотные альтернативы с использованием паттернов, понимание преимуществ композиции над наследованием.
- **Блок 4 (Комплексный проект):** Оценивается системность подхода к проектированию сложной предметной области, интеграция различных принципов ООП и модульности в единое решение.
- **Общее:** Четкость изложения, использование UML-диаграмм или схем там, где это уместно, качество псевдокода или реального кода, понимание компромиссов при выборе того или иного подхода.

Раздел 2. Технологии программирования

Тестовые задания

1. (ОВ) Последовательность фаз, через которые проходит программное обеспечение от возникновения идеи до вывода из эксплуатации, называется:
a) Проектным планом
b) Жизненным циклом программного обеспечения (Software Development Life Cycle - SDLC)
c) Техническим заданием
d) Roadmap продукта

2. (МВ) Какие из перечисленных моделей являются классическими моделями жизненного цикла ПО?
a) Каскадная (водопадная) модель (Waterfall)
b) Гибкая (Agile) методология
c) V-образная модель (V-Model)
d) Спиральная модель (Spiral Model)

3. (СО) Документ, который формально определяет требования к разрабатываемой системе, её функции, ограничения и цели, называется _____ (ТЗ).
Ответ: _____

4. (OB) Основной недостаток классической каскадной модели разработки заключается в:
a) Высокой скорости выпуска версий
b) Сложности и дороговизне внесения изменений на поздних этапах
c) Отсутствии этапа тестирования
d) Необходимости участия заказчика

5. (OB) На этапе сбора и анализа требований определяются:
a) Конкретные функции системы, которые она должна выполнять
b) Язык программирования и фреймворки для реализации
c) Количество разработчиков в команде
d) Цветовая схема интерфейса

6. (MB) Какие техники и инструменты используются для сбора требований?
a) Интервью с заказчиком и будущими пользователями
b) Создание пользовательских историй (User Stories) и сценариев использования (Use Cases)
c) Прототипирование интерфейсов (wireframes, mockups)
d) Написание кода для проверки гипотез

7. (CO) Графическая модель, которая показывает взаимодействие между пользователями (акторами) и системой для достижения конкретной цели, называется _____.
Ответ: _____

8. (OB) Целью технико-экономического обоснования (ТЭО) на раннем этапе является:
a) Написание всей документации проекта
b) Оценка реализуемости, сроков, стоимости и потенциальной выгоды от проекта
c) Создание финальной версии продукта
d) Подбор команды разработчиков

9. (MB) Что включает в себя этап проектирования (дизайна) системы?
a) Выбор архитектуры приложения (например, клиент-серверная, микросервисная)
b) Проектирование структуры базы данных (ER-диаграммы)
c) Проектирование пользовательского интерфейса (UI) и пользовательского опыта (UX)
d) Непосредственное написание кода всех модулей

10. (CO) Документ или набор диаграмм, описывающих высокоуровневую структуру системы, взаимодействие её компонентов и ключевые технологические решения, называется _____ проектированием.
Ответ: _____

11. (OB) Этап, на котором идеи и проекты превращаются в реальный работающий код, — это этап:
a) Анализа
b) Проектирования
c) **Реализации (кодирования / разработки)**
d) Тестирования

12. (CO) Практика, при которой две или более версии кода сливаются в репозитории несколько раз в день, и каждая интеграция проверяется автоматизированным

сбором и тестами, называется _____ интеграцией.
Ответ: _____

13. (ОВ) Процесс проверки соответствия программного продукта заданным требованиям и ожиданиям — это:
a) Кодирование
b) Документирование
c) **Тестирование**
d) Планирование

14. (МВ) Какие из перечисленных видов тестирования существуют?
a) Модульное (юнит-тестирование)
b) Интеграционное
c) Системное (приёмочное)
d) Тестирование производительности и нагрузки

15. (СО) Тип тестирования, которое имитирует работу реального пользователя для проверки всего приложения в среде, максимально приближенной к боевой, называется _____ тестированием.
Ответ: _____

16. (ОВ) Процесс передачи готового программного обеспечения в эксплуатацию пользователям — это этап:
a) Тестирования
b) **Развертывания (деплоя) и внедрения**
c) Проектирования
d) Сопровождения

17. (МВ) Какие работы обычно выполняются на этапе сопровождения и поддержки ПО?
a) Исправление обнаруженных ошибок (багфиксы)
b) Добавление новой функциональности по запросам пользователей
c) Адаптация к новому оборудованию или операционным системам
d) Полная переработка архитектуры приложения с нуля

18. (СО) Гибкая методология разработки, основанная на коротких итерациях (спринтах), регулярном планировании и демонстрации результатов заказчику, называется _____.
Ответ: _____

19. (ОВ) В гибких методологиях (Agile) список всех функций, которые необходимо реализовать в продукте, упорядоченный по приоритету, называется:
a) Техническим заданием
b) **Бэклогом продукта (Product Backlog)**
c) Дорожной картой (Roadmap)
d) Графиком Ганта

20. (СО) Практика автоматизации процессов сборки, тестирования и развертывания приложения для обеспечения частых и надежных выпусков обновлений называется / (CI/CD).
Ответ (аббревиатура): _____

Ключ для проверки

Раздел 1:

1. **b)** Жизненным циклом программного обеспечения (Software Development Life Cycle - SDLC)
2. **a, c, d** (b — Agile — это семейство гибких методологий, а не единая классическая модель жизненного цикла)
3. **техническое задание**
4. **b)** Сложности и дороговизне внесения изменений на поздних этапах

Раздел 2:
5. **a)** Конкретные функции системы, которые она должна выполнять
6. **a, b, c** (d — относится к этапу реализации)
7. **сценарий использования** (Use Case) или **диаграмма вариантов использования**
8. **b)** Оценка реализуемости, сроков, стоимости и потенциальной выгоды от проекта

Раздел 3:
9. **a, b, c** (d — это следующий этап, реализация)
10. **архитектурным** (или **высокоуровневым**)
11. **c) Реализации (кодирования / разработки)**
12. **непрерывной** (Continuous Integration - CI)

Раздел 4:
13. **c) Тестирование**
14. **a, b, c, d** (Все перечисленные)
15. **приёмочным** (acceptance testing) или **системным** (system testing)
16. **b) Развертывания (деплоя) и внедрения**

Раздел 5:
17. **a, b, c** (d — это, как правило, не сопровождение, а начало нового проекта или крупный рефакторинг)
18. **Scrum** (самый известный фреймворк в рамках Agile)
19. **b) Бэклогом продукта (Product Backlog)**
20. **CI/CD** (Continuous Integration / Continuous Delivery или Deployment)

Контрольные задания

Блок 1: Планирование и анализ требований

Задание 1. «Создание технического задания для мобильного приложения»
Ситуация: Стартап хочет разработать мобильное приложение для поиска наставников (менторов) в IT. Пользователи могут быть двух типов: **Ученики** (ищут ментора) и **Менторы** (предлагают свои услуги).

Требуется:

1. **Анализ стейкхолдеров:** Определите всех заинтересованных лиц (стейкхолдеров) проекта и их ключевые интересы.
2. **Сбор требований:** Составьте список из **10 функциональных требований** (что система должна делать) и **5 нефункциональных требований** (качество системы: производительность, безопасность, удобство).

Пример функционального: «Ученик может искать ментора по специализации (Frontend, Data Science) и рейтингу».
Пример нефункционального: «Время отклика при поиске не должно превышать 2 секунд при 1000 одновременных пользователей».

3. **Разработка пользовательских историй:** Напишите **3 пользовательские истории (User Stories)** в формате: «Как [роль], я хочу [возможность], чтобы [ценность/цель]».
Пример: «Как ученик, я хочу записаться на пробный 30-минутный сеанс с ментором, чтобы оценить, подходит ли мне его стиль преподавания».
4. **Создание прототипа:** Нарисуйте **low-fidelity wireframe** (схематичный чертеж) главного экрана приложения для ученика, используя инструмент Figma, Miro или даже на бумаге. Отметьте ключевые элементы: строка поиска, фильтры, список менторов.

Задание 2. «Анализ и выбор модели жизненного цикла»
Ситуации для анализа (выберите одну):

- **А) Разработка ПО для управления ядерным реактором.**
- **Б) Создание мобильной игры-головоломки с частыми обновлениями контента.**
- **В) Разработка внутренней системы учета рабочего времени для компании из 50 человек.**

Требуется для выбранной ситуации:

1. **Сравнительный анализ:** Для вашего проекта сравните **каскадную (Waterfall)** модель и **гибкую (Agile/Scrum)** методологию. Создайте таблицу с плюсами и минусами каждой применительно к вашему случаю.
2. **Обоснованный выбор:** Выберите наиболее подходящую модель/методологию и подробно обоснуйте свой выбор, ссылаясь на специфику проекта (четкость требований, важность документации, необходимость гибкости, критичность к ошибкам).
3. **План действий:** Набросайте **дорожную карту (roadmap)** первых 6 месяцев проекта в выбранной вами методологии. Если выбрали Agile, опишите, что может входить в первые 3 спринта. Если Waterfall — укажите длительность и ключевые результаты каждого этапа.

Блок 2: Проектирование и архитектура

Задание 3. «Проектирование архитектуры системы онлайн-чата»
Задача: Спроектировать backend для простого мессенджера с функциями: регистрация, список контактов, обмен текстовыми сообщениями в реальном времени, история переписки.

Требуется:

1. **Выделение компонентов:** Определите основные компоненты (модули/сервисы) вашей системы (например, Auth Service, User Profile Service, Chat Service, Message Broker, Database).
2. **Диаграмма компонентов:** Нарисуйте **схему взаимодействия компонентов**. Покажите, как они общаются между собой (REST API, WebSocket, очередь сообщений).
3. **Проектирование БД:** Спроектируйте **упрощенную схему базы данных**. Создайте ER-диаграмму с 3-4 основными сущностями (например, User, Conversation, Message) и связями между ними. Укажите ключевые поля.

4. **Выбор технологий:** Обоснуйте выбор технологического стека для одного компонента на выбор (например, почему для Chat Service может подойти Node.js с библиотекой Socket.io, а для хранения истории сообщений — PostgreSQL).

Задание 4. «Разработка технического решения (Technical Design Document)»
Задача: Вам нужно описать техническую реализацию функции «Умный поиск по товарам» для интернет-магазина. Поиск должен учитывать опечатки и синонимы.

Требуется написать разделы TDD:

1. **Контекст и цели:** Краткое описание задачи, которую решает этот документ.
2. **Обзор решения:** Высокоуровневое описание (использовать поисковый движок Elasticsearch для индексации товаров).
3. **Детали реализации:**
   - **Схема данных:** Как будет выглядеть документ товара в Elasticsearch (поля: id, name, description, category).
   - **Алгоритм:** Какие типы запросов Elasticsearch будут использоваться (match, fuzzy match).
   - **Интеграция:** Как backend-сервис будет взаимодействовать с Elasticsearch (REST API клиент).
4. **Альтернативы:** Какие альтернативные решения вы рассматривали (например, LIKE-запросы в SQL) и почему от них отказались.
5. **Оценка:** Грубая оценка трудозатрат в человеко-днях и потенциальные риски (например, сложность настройки релевантности в Elasticsearch).

Блок 3: Реализация, тестирование и CI/CD

Задание 5. «Настройка CI/CD пайплайна для микросервиса»
Исходные данные: У вас есть микросервис на Python (Flask) для управления пользователями. Код хранится в GitLab. Нужно настроить автоматическую сборку, тестирование и деплой.

Требуется:

1. **Создание конфигурации:** Напишите файл .gitlab-ci.yml, который описывает следующие стадии:
   - test: Запуск модульных тестов (pytest) и линтера (flake8).
   - build: Сборка Docker-образа микросервиса и его публикация в GitLab Container Registry.
   - deploy-to-staging: Автоматический деплой образа на тестовый сервер (staging environment) при пуше в ветку develop.
   - deploy-to-production: Ручное подтверждение (manual job) для деплоя в продакшн из ветки main после код-ревью.
2. **Инфраструктура как код:** Напишите простой Dockerfile для этого Python-приложения.
3. **Политика ветвления:** Опишите, какую модель Git-ветвления (GitFlow, GitHub Flow, Trunk-Based) вы бы использовали для этого проекта и почему.

Задание 6. «Планирование и проведение комплексного тестирования»
Объект тестирования: Веб-приложение для проведения онлайн-викторин с таймером и системой рейтинга.

Требуется разработать тест-план:

1. **Матрица тестирования:** Создайте таблицу с видами тестирования, которые необходимо провести. Для каждого вида укажите:

- **Цель** (Что проверяем?)
- **Методы/Инструменты** (Как проверяем?)
- **Критерии завершения** (Когда считаем успешным?).
  Пример строки: Вид: Нагрузочное тестирование. Цель: Определить, выдержит ли система 5000 одновременных участников викторины. Инструменты: JMeter. Критерии: Среднее время ответа < 3 сек, отсутствие ошибок 5xx под нагрузкой.

2. **Чек-лист для регрессии:** Составьте чек-лист из 10 пунктов для быстрого регрессионного тестирования после каждого обновления (например, «Регистрация нового пользователя работает», «Таймер стартует вовремя», «Результаты викторины корректно сохраняются»).
3. **Тест-кейс:** Детально распишите **один тест-кейс** для проверки функционала «Участие в викторине». Включите: Предусловия, Шаги (действия тестировщика), Ожидаемый результат.

Блок 4: Развертывание и сопровождение

Задание 7. «Планирование релиза и отката (Rollback Plan)»
Ситуация: Вы готовитесь выпустить крупное обновление (v2.0) своего SaaS-продукта. Обновление включает миграцию схемы базы данных и изменения в ключевом API.

Требуется:

1. **План проведения релиза (Release Plan):** Распишите пошаговую последовательность действий для дня релиза. Включите:

- Предварительные действия (бэкап БД, уведомление пользователей о техобслуживании).
- Порядок обновления компонентов (сначала бэкап-сервисы, потом основные).
- Процедуры проверки после деплоя (health checks, smoke-тесты).
- Критерии успешного релиза.

2. **Стратегия развертывания:** Какую стратегию деплоя вы выберете и почему? (Синий-зеленый, канареечный, постепенный rollout). Опишите, как она будет реализована технически.
3. **План отката (Rollback Plan):** Разработайте четкий план действий на случай, если после релиза обнаружена критическая ошибка. Что откатывать в первую очередь? Как восстановить данные из бэкапа? Сколько времени это займет? Кто принимает решение об откате?

Задание 8. «Анализ инцидента и улучшение процесса»
Сценарий инцидента (Post-Mortem): В понедельник утром ваше приложение стало недоступно на 45 минут. Причина: разработчик залил в main ветку код, который сломал работу с кэшем. Автоматические тесты прошли успешно, но дежурный инженер заметил проблему только через 15 минут по мониторингу.

Требуется составить «отчет о происшествии»:

1. **Хронология:** Восстановите таймлайн инцидента (когда упало, когда обнаружено, когда начато исправление, когда восстановлено).
2. **Коренная причина (Root Cause Analysis):** Определите не только техническую причину (баг в коде), но и **процессные провалы** (почему баг попал в main? почему тесты не выловили? почему реакция была 15 минут?).

3. **Воздействие:** Оцените влияние (сколько пользователей затронуто, финансовые/репутационные потери).
4. **План предотвращения:** Предложите **5 конкретных действий** (Action Items), которые нужно предпринять, чтобы подобное не повторилось. Разделите их на технические (например, «внедрить тестирование интеграции с кэшем») и процессные (например, «ввести правило: пулл-реквест в main должен получать 2 аппрува»). Назначьте ответственных и сроки.

Критерии оценки:

- **Блок 1 (Планирование):** Оценивается полнота и конкретность требований, качество пользовательских историй, обоснованность выбора модели разработки.
- **Блок 2 (Проектирование):** Оценивается логичность архитектурных решений, качество диаграмм, структурированность технической документации.
- **Блок 3 (Реализация/Тестирование):** Оценивается работоспособность конфигурации CI/CD, системный подход к планированию тестирования, внимание к деталям в тест-кейсах.
- **Блок 4 (Эксплуатация):** Оценивается реалистичность и детализация планов релиза и отката, глубина анализа инцидента и практичность предлагаемых улучшений процессов.
- **Общее:** Умение переводить теорию в практические артефакты (документы, схемы, конфиги), четкость изложения, профессиональный подход к решению комплексных задач.

4.2 Контрольные задания для промежуточной аттестации

Вопросы для зачета

1. Дайте определение алгоритма. Перечислите и охарактеризуйте основные свойства алгоритмов.
2. Что такое структурное программирование? Объясните базовые управляющие конструкции: следование, ветвление, цикл.
3. Объясните понятия «тип данных», «переменная», «константа». Какие простые и структурированные типы данных вы знаете?
4. Что такое функция/подпрограмма? Объясните её назначение, преимущества использования и понятия «формальный» и «фактический» параметр.
5. Опишите основные принципы модульного программирования. Что такое связность (cohesion) и связанность (coupling)?
6. Что такое рекурсия? Приведите пример рекурсивного алгоритма и объясните условия его завершения.
7. Объясните основные алгоритмы сортировки: «пузырьковая» сортировка и «быстрая» сортировка. Сравните их временную сложность.
8. Что такое сложность алгоритма (Big O нотация)? Приведите примеры O(1), O(n), O(n^2).
9. Назовите и дайте определение четырем основным принципам ООП.
10. Что такое класс и объект? Объясните их взаимосвязь на конкретном примере.
11. Раскройте суть принципа инкапсуляции. Для чего используются модификаторы доступа (private, public, protected)?
12. Что такое наследование? Какие виды наследования вы знаете? Приведите пример иерархии классов.

13. Объясните принцип полиморфизма. Что такое перегрузка (overloading) и переопределение (перезапись, overriding) методов?
14. Что такое конструктор и деструктор? Каково их назначение в жизненном цикле объекта?
15. Дайте определение абстрактного класса и интерфейса. В чем их ключевое различие?
16. Объясните понятие «композиция» и «агрегация» в ООП. Чем они отличаются от наследования?

17. Каковы основные функции операционной системы?
18. Объясните, что такое процесс и поток (нить). В чем между ними разница?
19. Что такое виртуальная память? Опишите механизм страничной организации памяти.
20. Опишите классическую архитектуру фон Неймана. Каковы её основные компоненты?
21. Что такое файловая система? Какие типы файловых систем вы знаете (FAT32, NTFS, ext4)?
22. Объясните понятие «прерывание» (interrupt). Какие виды прерываний существуют?
23. Что такое многозадачность? В чем разница между вытесняющей и кооперативной многозадачностью?
24. Опишите основные этапы загрузки операционной системы.

25. Что такое база данных (БД) и СУБД? Назовите основные модели данных.
26. Сформулируйте основные понятия реляционной модели: отношение, атрибут, кортеж, первичный и внешний ключ.
27. Что такое нормализация БД? Какова её основная цель?
28. Какие группы операторов входят в язык SQL? Приведите примеры команд из каждой группы.
29. Опишите базовую модель взаимодействия открытых систем OSI. Назовите её уровни.
30. Что такое IP-адрес, маска подсети и шлюз по умолчанию? Объясните разницу между IPv4 и IPv6.
31. Что такое протокол TCP/IP? Опишите разницу между протоколами TCP и UDP.
32. Дайте определение DNS. Какова его роль в работе сети Интернет?

33. Сформулируйте триаду КЦД (CIA) информационной безопасности. Дайте краткую характеристику каждому компоненту.
34. Что такое вредоносное ПО (malware)? Перечислите и охарактеризуйте основные его типы.
35. Объясните угрозу «Социальная инженерия». Приведите примеры фишинга и фарминга.
36. Что такое брандмауэр (файрвол)? Какие типы брандмауэров вы знаете?
37. Дайте определение облачным вычислениям. Опишите сервисные модели IaaS, PaaS, SaaS.
38. Что такое модель разделенной ответственности (Shared Responsibility Model) в облачной безопасности?
39. Объясните принцип «нулевого доверия» (Zero Trust) в современных подходах к безопасности.
40. Что такое DevOps и DevSecOps? Как эти методологии влияют на жизненный цикл разработки ПО и его безопасность?

Критерии оценки

Оценка «5» - (отлично)

При ответе материал изложен грамотным языком в определенной логической последовательности, точно использована терминология, полно раскрыто содержание материала в объеме, предусмотренном программой, продемонстрировано усвоение ранее изученных сопутствующих вопросов. Возможны одна - две неточности при освещении второстепенных вопросов.

Оценка «4» - (хорошо)

Ответ удовлетворяет в основном требованиям на оценку «5», но при этом имеет один из недостатков: в изложении допущены небольшие пробелы; допущены один – два недочета при освещении основного содержания ответа, исправленные по замечанию преподавателя; допущены ошибка или более двух недочетов при освещении второстепенных вопросов, легко исправленные по замечанию преподавателя.

Оценка «3» - (удовлетворительно)

При ответе неполно или непоследовательно раскрыто содержание материала, но показано общее понимание, имелись затруднения или допущены ошибки в определении понятий.

Оценка «2» - (неудовлетворительно)

При ответе не раскрыто основное содержание учебного материала; обнаружено незнание или непонимание обучающимся большей или наиболее важной части учебного материала; допущены ошибки в определении понятий, допущены существенные ошибки, показавшие, что обучающийся не владеет обязательными умениями по данной теме в полной мере.