

Документ подписан простой электронной подписью

Информация о владельце:

ФИО: Узунов Федор Владимирович

Должность: Ректор

Дата подписания: 30.09.2026 08:54:44

Уникальный программный ключ:

fd935d10451b860e912264c0378b448451b603f94388008e29877a6bcb55

**АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ОРГАНИЗАЦИЯ
«ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ ВЫСШЕГО ОБРАЗОВАНИЯ»**

«УНИВЕРСИТЕТ ЭКОНОМИКИ И УПРАВЛЕНИЯ»

Факультет экономики, управления и юриспруденции

Кафедра управления и бизнес-информатики



УТВЕРЖДАЮ

Проректор по учебно-методической работе

[Signature] / Г.П. Узунова

«02» февраля 2026 г.

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ ПО ДИСЦИПЛИНЕ

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Направление подготовки

10.03.01 «Информационная безопасность»

Профиль

Безопасность компьютерных систем

Квалификация выпускника: бакалавр

Для всех
форм обучения

Симферополь, 2026 г.

1. ОЦЕНОЧНЫЕ СРЕДСТВА ДЛЯ ТЕКУЩЕГО КОНТРОЛЯ

Средства оценивания в ходе текущего контроля:

- устные опросы в ходе семинарских занятий;
- рефераты;
- тестирование;
- практические задания, выполняемые в ходе семинарского (практического) занятия или рекомендуемые для самостоятельной работы.

Коды компетенции	Результаты освоения ОПОП	Перечень планируемых результатов обучения по дисциплине
ОПК-7	Способен использовать языки программирования и технологии разработки программных средств для решения задач профессиональной деятельности	ОПК-7.1. Выбирает структуры данных и разрабатывает алгоритмы решения задач профессиональной деятельности ОПК-7.2. Выполняет отладку и тестирование программ ОПК-7.3. Разрабатывает и реализовывает на языке высокого уровня алгоритмы решения профессиональных задач

1.1 Вопросы к текущему контролю

1. Перечислите основные этапы жизненного цикла программного обеспечения (ЖЦ ПО). В чем заключаются ключевые различия между каскадной моделью и V-моделью разработки?
2. Методология Agile: в чем заключается суть гибкого подхода? Назовите четыре главных ценности «Манифеста гибкой разработки ПО».
3. Сравните фреймворки Scrum и Kanban как инструменты управления процессом разработки. Что такое бэклог продукта и спринт?
4. Модели зрелости процессов CMMI и непрерывная интеграция/непрерывная доставка (CI/CD): как эти концепции влияют на качество выпускаемого программного продукта?
5. Принципы SOLID в объектно-ориентированном проектировании. Раскройте смысл принципа единственной ответственности (*Single Responsibility Principle*) и принципа подстановки Барбары Лисков (*LSP*).
6. Паттерны проектирования (Design Patterns). Приведите примеры одного порождающего, одного структурного и одного поведенческого паттерна из классификации «Банды четырех» (*GoF*).
7. Многоуровневая архитектура информационных систем (*N-tier architecture*). Опишите функции уровней представления (*Presentation*), бизнес-логики (*Business Logic*) и доступа к данным (*Data Access*).
8. Микросервисная архитектура против монолитной. Каковы основные преимущества и недостатки микросервисов при проектировании высоконагруженных систем?
9. Асимптотическая сложность алгоритмов ($\mathcal{O}$ -нотация). Сравните эффективность поиска элемента в неотсортированном массиве, сбалансированном бинарном дереве поиска и хеш-таблице.
10. Рекурсия и итерация. Понятие хвостовой рекурсии. Почему глубокая рекурсия может привести к ошибке переполнения стека (*Stack Overflow*) и как этого избежать?
11. Системы контроля версий (VCS). Объясните разницу между централизованными (SVN, CVS) и распределенными (Git, Mercurial) системами. Что такое слияние (*merge*) и перебазирование (*rebase*) в Git?

12. Интегрированные среды разработки (*IDE*) и редакторы кода. Роль статических анализаторов кода (*Linters*) и инструментов форматирования (*Formatters*) в обеспечении единого стиля кодирования команды.
13. Различия между реляционными (*SQL*) и нереляционными (*NoSQL*) СУБД. Для каких задач целесообразно использовать документные (*MongoDB*) и графовые (*Neo4j*) базы данных?
14. Оптимизация работы с базой данных. Что такое *N+1 query problem*? Как использование *Eager Loading* решает эту проблему в ORM-фреймворках?
15. Пирамида тестирования (*Testing Pyramid*). Охарактеризуйте уровни модульных (*Unit*), интеграционных и сквозных (*End-to-End / UI*) тестов.
16. Практики разработки через тестирование (*TDD — Test Driven Development*) и разработка через поведение (*BDD — Behavior Driven Development*). В чем их принципиальное отличие от традиционного написания тестов после кода?
17. Основные уязвимости веб-приложений согласно списку *OWASP Top 10*. Механизмы защиты от инъекций (*Injection*s) и межсайтового скриптинга (*XSS*).
18. Аутентификация и авторизация. В чем разница между этими понятиями? Опишите принцип работы протокола *OAuth 2.0* для делегированного доступа.
19. Контейнеризация приложений (*Docker*) и оркестрация (*Kubernetes*). Чем контейнер отличается от виртуальной машины и какую роль играет *Dockerfile*?
20. Искусственный интеллект в разработке ПО. Как большие языковые модели (*LLM*, например, *GitHub Copilot* или *GigaChat*) применяются для генерации кода, автодополнения и рефакторинга сегодня?

1.2 Темы рефератов:

1. Эволюция парадигм программирования: от процедурного и структурного подходов к объектно-ориентированному и функциональному программированию.
2. Сравнительный анализ методологий разработки ПО: *Waterfall*, *V-Model*, *Agile (Scrum)* и *DevOps* в контексте современных ИТ-компаний.
3. Принципы *SOLID* в проектировании архитектуры приложений: практическое применение на примере одного из языков высокого уровня (*Java*, *C#*, *Python*).
4. Паттерны проектирования «Банды четырех» (*GoF*): классификация и области применения порождающих паттернов (*Singleton*, *Factory Method*, *Abstract Factory*).
5. Функциональное программирование как инструмент повышения надежности кода: иммутабельность, чистые функции и отсутствие сайд-эффектов.
6. Асинхронное программирование и конкурентность: сравнение моделей многопоточности (*Threading*), корутин (*Async/Await*) и модели акторов (на примере *Akka* или *Orleans*).
7. Языки со строгой и динамической типизацией: влияние систем типов на масштабируемость проекта, производительность и количество ошибок времени выполнения.
8. Управление памятью в языках программирования: сравнительный анализ ручного управления (*C/C++*), сборки мусора (*Garbage Collection — Java, C#*) и владения/борроучека (*Rust*).
9. Роль *LLM* (больших языковых моделей) в современной разработке: использование нейросетей для генерации кода, автодополнения (*GitHub Copilot*) и рефакторинга.
10. Контейнеризация микросервисов с помощью *Docker*: архитектура *Docker Engine*, написание оптимизированных *Dockerfile* и управление образами.
11. Оркестрация контейнеризованных приложений (*Kubernetes*): автоматизация развертывания, масштабирования и самовосстановления распределенных систем.
12. Непрерывная интеграция и доставка (*CI/CD*): построение пайплайнов сборки и тестирования с использованием *Jenkins*, *GitLab CI* или *GitHub Actions*.

13. Тестирование программного обеспечения: стратегия построения пирамиды тестов (Unit, Integration, E2E). Практики TDD (*Test Driven Development*) и BDD (*Behavior Driven Development*).
14. Методы профилирования и оптимизации производительности высоконагруженных сервисов: поиск узких мест (*bottlenecks*) в CPU, памяти и дисковой подсистеме.
15. Безопасность приложения на уровне кода (*Secure Coding*): предотвращение уязвимостей OWASP Top 10 (Injections, XSS, Broken Access Control) средствами самого языка программирования.
16. Проектирование API: архитектурный стиль RESTful против RPC (*gRPC*, *tRPC*). Версионирование, документирование через OpenAPI (Swagger) и вербозность протоколов.
17. Основы реактивного программирования (*Reactive Programming*): потоки данных (*Streams*), наблюдаемые последовательности (*Observables*) и библиотека RxJS.
18. Кросс-платформенная разработка мобильных приложений: фреймворки Flutter и React Native. Сравнение производительности и доступа к нативным функциям по сравнению с native-разработкой.
19. Введение в квантовое программирование: модель кубитов, языки Q# и Qiskit, алгоритм Шора и перспективы взлома классических криптографических алгоритмов.
20. Низкоуровневое программирование и системы реального времени (RTOS): особенности написания кода для микроконтроллеров (Embedded C) и ограничения по времени отклика.

1.3 Тестовые задания

1. Какая модель жизненного цикла ПО предполагает переход на следующий этап только после полного завершения работ на предыдущем без возможности возврата? а) Спиральная б) Инкрементальная в) Каскадная (Waterfall) (*Правильный ответ: в*) г) Agile
2. Что означает буква «S» в аббревиатуре принципов SOLID проектирования? а) Single Responsibility Principle (Принцип единственной ответственности) (*Правильный ответ: а*) б) Secure Code Principle (Принцип безопасного кода) в) Simple Design Principle (Принцип простоты) г) Static Typing Principle (Принцип статической типизации)
3. Какой паттерн проектирования гарантирует, что у класса может быть только один экземпляр и предоставляет к нему глобальную точку доступа? а) Factory Method б) Singleton (*Правильный ответ: б*) в) Observer г) Adapter
4. В чем заключается ключевое преимущество использования системы контроля версий Git по сравнению с централизованными системами (например, SVN)? а) Наличие графического интерфейса б) Возможность полноценной работы локально без подключения к центральному серверу (*Правильный ответ: б*) в) Меньший объем занимаемого места на диске г) Автоматическая генерация документации
5. Сложность алгоритма $O(n^2)$ означает, что при увеличении входных данных в 10 раз время выполнения: а) Увеличится в 10 раз б) Увеличится в 100 раз (*Правильный ответ: б*) в) Увеличится в 2 раза г) Не изменится
6. Чем контейнер Docker принципиально отличается от виртуальной машины (VM)? а) Контейнер эмулирует аппаратное обеспечение, а VM — нет б) Контейнеры используют ядро хостовой операционной системы, не требуя гостевой ОС для каждого экземпляра (*Правильный ответ: б*) в) VM запускаются быстрее контейнеров г) Между ними нет архитектурной разницы

7. Как называется практика написания тестов до написания самого производственного кода? а) Unit Testing б) TDD (Test Driven Development) (Правильный ответ: б) в) BDD (Behavior Driven Development) г) Regression Testing
8. Какой HTTP-метод в архитектуре RESTful используется для **полного** обновления существующего ресурса? а) GET б) POST в) PUT (Правильный ответ: в) г) DELETE
9. Что является основной целью процесса непрерывной интеграции (CI)? а) Ручное тестирование продукта перед релизом б) Автоматическое слияние изменений от нескольких разработчиков и проверка сборки несколько раз в день (Правильный ответ: б) в) Написание технической документации г) Развертывание приложения на боевых серверах вручную
10. Проблема взаимоблокировки (*Deadlock*) в многопоточных приложениях возникает, когда: а) Два потока пытаются одновременно изменить одну переменную б) Несколько потоков бесконечно ожидают ресурсы, занятые друг другом (Правильный ответ: б) в) Процессор загружен на 100% г) Закончилось место в оперативной памяти
11. Сборка мусора (*Garbage Collection*) в языках вроде Java или C# предназначена для: а) Оптимизации скорости выполнения циклов б) Автоматического освобождения памяти, на которую больше нет ссылок (Правильный ответ: б) в) Шифрования конфиденциальных данных г) Проверки синтаксиса кода перед компиляцией
12. Какую проблему решает использование ORM-библиотеки (например, Hibernate или Entity Framework)? а) Несоответствие между объектно-ориентированной моделью программы и реляционной моделью базы данных (Правильный ответ: а) б) Низкую скорость работы жесткого диска в) Отсутствие интернет-соединения г) Ошибки в CSS-стилях фронтенда
13. В функциональном программировании функции высшего порядка — это функции, которые: а) Имеют сложность $O(\log n)$ б) Принимают другие функции в качестве аргументов или возвращают их как результат (Правильный ответ: б) в) Работают только с числами высокой точности г) Выполняются исключительно на сервере
14. Уязвимость SQL Injection возникает, если разработчик: а) Использует параметризованные запросы б) Формирует SQL-запрос путем конкатенации строк с пользовательским вводом без фильтрации (Правильный ответ: б) в) Забывает закрыть соединение с БД г) Использует слишком длинные имена таблиц
15. Для чего предназначен файл `package.json` в экосистеме Node.js? а) Для хранения настроек базы данных б) Для описания зависимостей проекта, скриптов и метаданных (Правильный ответ: б) в) Для конфигурации веб-сервера Nginx г) Для написания юнит-тестов
16. Что такое «состояние гонки» (*Race Condition*)? а) Ситуация, когда два процессора соревнуются за мощность блока питания б) Ошибка, возникающая из-за зависимости результата от непредсказуемого порядка переключения потоков (Правильный ответ: б) в) Медленная загрузка веб-страницы г) Переполнение целочисленного типа данных
17. В методологии Scrum собрание команды, которое длится не более 15 минут и проводится ежедневно для синхронизации действий, называется: а) Sprint Review б) Daily Stand-up (Правильный ответ: б) в) Retrospective г) Planning Poker

18. Какова основная задача Webpack, Vite или Rollup в современной веб-разработке? а) Проектирование структуры базы данных б) Модульная сборка (*bundling*), транспиляция и оптимизация ресурсов (*Правильный ответ: б*) в) Управление версиями сервера г) Отправка email-рассылок пользователям

19. Язык TypeScript является строгим надмножеством JavaScript. Что он добавляет поверх стандартного JS? а) Статическую типизацию и интерфейсы (*Правильный ответ: а*) б) Новый движок исполнения в браузере в) Собственную базу данных г) Упрощенный синтаксис HTML

20. Что означает термин «Callback Hell» («Ад коллбэков») в асинхронном программировании? а) Слишком большое количество функций в одном файле б) Глубокая вложенность callback-функций, делающая код трудночитаемым и поддерживаемым (*Правильный ответ: б*) в) Отсутствие обработчиков ошибок г) Рекурсивный вызов функции самой себя

21. Установить соответствие между свойством CSS и его функцией:

1. color	А) Управление фоном элемента
2. background-color	Б) Изменение цвета текста
3. font-size	В) Установка размера шрифта
4. text-align	Г) Выравнивание текста

22. Установить соответствие между графическим форматом и его особенностью:

1. JPEG	А) Поддержка прозрачности и анимации
2. PNG	Б) Поддержка прозрачности без анимации
3. GIF	В) Анимация, ограниченная палитра
4. SVG	Г) Векторный формат, масштабируемость

23. Установить соответствие между этапом разработки сайта и его содержанием:

1. Аналитика	А) Верстка, программирование, тестирование
2. Дизайн	Б) Сбор требований, анализ конкурентов
3. Разработка	В) Создание макетов, прототипов, визуализация
4. Запуск	Г) Публикация сайта, настройка домена и хостинга

24. Установить соответствие между профессией и ее основной задачей:

1. UX-дизайнер	А) Визуальное оформление сайта
2. Front-end разработчик	Б) Программирование серверной части сайта

3. UI-дизайнер	В) Создание удобного пользовательского опыта
4. Back-end разработчик	Г) Верстка и программирование клиентской части сайта

25. Установить соответствие между инструментами и его назначением:

1. Figma	А) Конструктор сайтов без программирования
2. Adobe Photoshop	Б) Графический редактор для растровой графики
3. Visual Studio Code	В) Редактор кода для веб-разработки
4. Tilda	Г) Онлайн-сервис для прототипирования и дизайна интерфейсов

26. Установить соответствие между термином и его определением:

1. Верстка сайта	А) Процесс создания структуры и визуального оформления веб-страницы с помощью HTML и CSS
2. Адаптивный дизайн	Б) Подход, при котором сайт корректно отображается на устройствах с любым размером экрана
3. CMS	В) Система управления контентом сайта
4. Прототипирование	Г) Создание черновой модели сайта для демонстрации структуры и логики работы

27. Установить соответствие между свойством CSS и его описанием:

1. display	А) Управляет внешним видом курсора
2. cursor	Б) Определяет, как элемент отображается на странице
3. box-shadow	В) Добавляет тень к тексту
4. text-shadow	Г) Добавляет тень к блоку элемента

28. Установить соответствие между этапом разработки сайта и его результатом:

1. Проектирование	А) Готовый сайт на хостинге
2. Дизайн	Б) Черно-белая схема структуры страниц
3. Верстка	В) Красивый макет с цветами, шрифтами и изображениями
4. Запуск	Г) Работающий интерактивный сайт в браузере

29. Установить соответствие между тегом HTML и его назначением:

1. <header>	А) Контактная информация, авторство, ссылки
2. <footer>	Б) Заголовок раздела или всей страницы
3. <nav>	В) Основное содержимое страницы
4. <main>	Г) Навигационное меню

30. Установить соответствие между типом позиционирования элемента в CSS и его описанием:

1. static	А) Элемент позиционируется отдельно своего ближайшего позиционированного предка
2. relative	Б) Элемент «вырывается» из потока документа и располагается относительно окна браузера
3. absolute	В) Элемент остается в обычном потоке документа, его положение не меняется
4. fixed	Г) Элемент смещается относительно своего обычного положения, но место в потоке сохраняется

1.4 Практические задания

- Сложность алгоритмов.** Дан массив целых чисел A размером N . Напишите фрагмент кода с асимптотической сложностью $O(N^2)$, который находит все пары индексов (i, j) , для которых сумма элементов $A[i] + A[j]$ является четным числом. (Ожидаемый результат: реализация через два вложенных цикла).
- Рекурсия vs Итерация.** Реализуйте функцию вычисления n -го числа Фибоначчи двумя способами: через классическую рекурсию и через цикл (for или while). Сравните их работу при $n = 40$ и объясните причину колоссальной разницы в производительности. (Ожидаемый результат: демонстрация экспоненциальной сложности $O(2^n)$ у чистой рекурсии из-за повторных вызовов).
- Хеш-таблицы.** У вас есть список из 1 000 000 уникальных телефонных номеров. Вам нужно проверить, содержится ли конкретный номер в этом списке. Почему использование линейного поиска неэффективно? Опишите алгоритм решения этой задачи с использованием хеш-множества (*HashSet*) за время $O(1)$ в среднем случае. (Ожидаемый результат: обоснование выбора структуры данных на основе константного времени доступа).
- Стеки и Очереди.** Дана строка, содержащая только символы $(,), \{, \}, [\text{ и }]$. Напишите алгоритм, определяющий, является ли входная строка корректной последовательностью скобок (каждая открывающая скобка должна закрываться тем же типом в правильном порядке). (Ожидаемый результат: решение с использованием стека; пример: " $\{\{\}\}$ " — валидно, " $([])$ " — невалидно).
- Графы.** В социальной сети пользователи представлены вершинами графа, а дружба между ними — ребрами. Предложите алгоритм (BFS или DFS), который находит кратчайшую цепочку знакомств («степень удаления») между пользователем A и

пользователем Б. (*Ожидаемый результат: описание логики обхода в ширину для поиска минимального количества переходов*).

6. **Фабричный метод.** Разработайте систему отчетов, которая может экспортировать данные в форматах PDF, Excel и JSON. Создайте интерфейс IReportGenerator и реализуйте фабрику, которая возвращает нужный экземпляр класса в зависимости от строки-параметра ("pdf", "excel", "json"), скрывая детали инстанцирования от клиентского кода. (*Ожидаемый результат: код фабрики, избавляющий клиента от использования оператора new для конкретных классов*).

7. **Проблема N+1 запроса.** Имеется сущность «Автор», у которой есть коллекция «Книги». При выводе списка из 10 авторов и названий их книг ORM-фреймворк выполняет 1 запрос к таблице авторов и по 1 запросу на каждого автора для получения книг (всего 11 запросов). Какую технику загрузки (*Eager Loading / JOIN*) нужно применить, чтобы сократить количество запросов до двух? (*Ожидаемый результат: описание механизма включения связанных данных одним SQL-запросом с JOIN*).

8. **Паттерн Наблюдатель (Observer).** Объект «Точка продажи» (Subject) должен уведомлять объекты «Мобильное приложение» и «Дашборд администратора» (Observers) об изменении остатка товара на складе. Опишите структуру взаимодействия этих объектов без жесткой привязки друг к другу. (*Ожидаемый результат: схема интерфейса регистрации/дерегистрации подписчиков и рассылки событий notify*).

9. **Оптимизация транзакций.** Два банковских счета (А и Б) одновременно переводят деньги друг другу (А -> Б и Б -> А). Если операции чтения баланса и записи нового значения не атомарны, возникает race condition. Опишите механизм блокировок (*Pessimistic Locking*), который гарантирует целостность баланса. (*Ожидаемый результат: сценарий выполнения SELECT ... FOR UPDATE*).

10. **Нормализация БД.** В таблице заказов хранятся поля: OrderID, CustomerName, CustomerPhone, ProductID, ProductName, ProductPrice. Нарушены ли здесь правила нормализации? Если да, опишите шаги приведения схемы к третьей нормальной форме (3NF). (*Ожидаемый результат: выделение сущностей Customers и Products в отдельные таблицы для устранения дублирования имен*).

11. **NoSQL документные базы.** Вам нужно хранить профили пользователей, где каждый профиль имеет уникальный набор дополнительных полей (например, у одного есть хобби, у другого — данные о водительских правах). Почему реляционная БД будет неудобна (проблема разреженности таблиц)? Обоснуйте выбор MongoDB для этой задачи. (*Ожидаемый результат: сравнение хранения в виде плоской таблицы со множеством NULL против гибкого JSON/BSON документа*).

12. **Git Merge Conflict.** Два разработчика изменили одну и ту же строку в файле config.js и отправили изменения в разные ветки. При попытке слияния (*merge*) возникла ошибка. Опишите пошаговый алгоритм ручного разрешения этого конфликта и финальной фиксации изменений. (*Ожидаемый результат: последовательность команд git status, редактирование файла, git add, git commit*).

13. **Контейнеризация (Docker).** Приложение на Python зависит от специфической версии библиотеки libc, которая отличается от установленной на сервере разработчика. Составьте минимальный Dockerfile, гарантирующий идентичность среды исполнения на любой машине. (*Ожидаемый результат: базовый образ FROM python:3.9-slim, копирование requirements.txt, установка зависимостей*).

14. **CI/CD Pipeline.** Разработчик отправил код в ветку feature/login-form. Напишите логику простого GitHub Actions workflow, который должен автоматически запуститься, установить зависимости (npm install) и прогнать юнит-тесты (npm test), запретив мердж-реквест при падении тестов. (*Ожидаемый результат: структура YAML-файла с триггером on: [pull_request]*).

15. **XSS (Cross-Site Scripting).** На сайте есть поле ввода комментария. Если пользователь введет <script>alert('Hacked')</script>, скрипт выполнится в браузере других

посетителей страницы. Какое фундаментальное действие должен выполнить бэкенд перед сохранением и отображением этого текста, чтобы закрыть уязвимость? *(Ожидаемый результат: HTML-экранирование спецсимволов, превращающее теги в <script>).*

16. **Deadlock (Взаимная блокировка).** Поток T1 нужен доступ к ресурсу R1, а затем к R2. Поток T2 нужен доступ к R2, а затем к R1. Оба захватили по одному ресурсу и ждут освобождения второго вечно. Приведите правило нумерации ресурсов, которое гарантированно предотвратит эту ситуацию. *(Ожидаемый результат: требование захватывать блокировки всегда в строго определенном глобальном порядке).*

17. **Async/Await.** Программа должна скачать данные сразу с трех разных API-сервисов. Каждый запрос занимает 2 секунды. Сколько времени займет выполнение при синхронном подходе и сколько при использовании конкурентного ожидания (*Promise.all* или аналог)? *(Ожидаемый результат: ~6 секунд синхронно против ~2 секунд асинхронно, так как запросы идут параллельно).*

18. **RESTful API.** Клиент отправляет DELETE /api/users/123. Сервер успешно удалил пользователя. Что именно должен вернуть сервер в теле ответа согласно спецификации REST и почему? *(Ожидаемый результат: статус 204 No Content, так как ресурс больше не существует и возвращать нечего).*

19. **Логирование.** В высоконагруженном сервисе произошла ошибка деления на ноль. Почему использование конструкции `print(error)` внутри кода недопустимо в production-среде? Какие уровни логирования (*DEBUG, INFO, WARN, ERROR*) следует использовать для трассировки входа в функции и для записи критических сбоев? *(Ожидаемый результат: print блокирует I/O поток процесса; ERROR для сбоев, DEBUG для трассировки).*

20. **Рефакторинг «Божественного объекта» (God Object).** Класс `UserManager` содержит 50 методов: он шлет письма, рисует аватарки, считает налоги и пишет в базу данных. Примените принцип единственной ответственности (SRP) и опишите, на какие 4 отдельных класса/сервиса его следует разбить. *(Ожидаемый результат: UserRepository (БД), EmailService (почта), TaxCalculator (логика), ProfileImageProcessor (графика)).*

2. КРИТЕРИИ ОЦЕНИВАНИЯ ПРИ ПРОВЕДЕНИИ ТЕКУЩЕГО КОНТРОЛЯ

Вид контроля	Наименование работы	Наименование оценочных средств	Шкала оценивания
Текущий контроль	- Вопросы для обсуждения на занятиях; - Устные опросы по ранее изученному материалу; - Письменные работы: рефераты, тестовые задания; - Практические задания; - Рефераты и доклады по темам (вопросам), вынесенным на самостоятельную работу.	Оценка выступлений на практическом (семинарском) занятии, проверка заданий и аудиторных работ, устный опрос, оценивание докладов, рефератов	отлично хорошо удовлетворительно неудовлетворительно

Критерии оценивания устных ответов обучающихся

Шкала оценивания	Характеристика оценивания
отлично	оценивается ответ, который показывает прочные знания основных процессов изучаемой предметной области, отличается глубиной и

	полнотой раскрытия темы; владение терминологическим аппаратом; умение объяснять сущность, явлений, процессов, событий, делать выводы и обобщения, давать аргументированные ответы, приводить примеры; свободное владение монологической речью, логичность и последовательность ответа.
хорошо	оценивается ответ, обнаруживающий прочные знания основных процессов изучаемой предметной области, отличается глубиной и полнотой раскрытия темы; владение терминологическим аппаратом; умение объяснять сущность, явлений, процессов, событий, делать выводы и обобщения, давать аргументированные ответы, приводить примеры; свободное владение монологической речью, логичность и последовательность ответа. Однако допускается одна - две неточности в ответе.
удовлетворительно	оценивается ответ, свидетельствующий в основном о знании процессов изучаемой предметной области, отличающийся недостаточной глубиной и полнотой раскрытия темы; знанием основных вопросов теории; слабо сформированными навыками анализа явлений, процессов, недостаточным умением давать аргументированные ответы и приводить примеры; недостаточно свободным владением монологической речью, логичностью и последовательностью ответа. Допускается несколько ошибок в содержании ответа.
неудовлетворительно	оценивается ответ, обнаруживающий незнание процессов изучаемой предметной области, отличающийся неглубоким раскрытием темы; незнанием основных вопросов теории, несформированными навыками анализа явлений, процессов; неумением давать аргументированные ответы, слабым владением монологической речью, отсутствием логичности и последовательности. Допускаются серьезные ошибки в содержании ответа.

Критерии оценивания работы обучающихся на практических и семинарских занятиях

Шкала оценивания	Показатели	Критерии
Отлично	1. Полнота выполнения практического и тестового задания (полнота ответа); 2. Своевременность выполнения задания; 3. Последовательность и рациональность выполнения практического задания	Задание решено самостоятельно. При этом составлен правильный алгоритм решения задания, в логических рассуждениях, в выборе формул и решении нет ошибок, получен верный ответ, задание решено рациональным способом. Дан правильный и исчерпывающий ответ на поставленные теоретические и тестовые вопросы, в которых обучающийся показал всестороннее системное знание программного материала, усвоение основной и дополнительной литературы, четкое владение понятийным аппаратом.
Хорошо	4. Правильность ответов на вопросы; 5. Самостоятельность решения (владение	Задание решено с помощью преподавателя. При этом составлен правильный алгоритм решения задания, в логическом рассуждении и решении нет существенных ошибок; правильно сделан выбор формул для решения; есть объяснение решения, но задание решено нерациональным способом или допущено не более двух несущественных ошибок, получен верный ответ. На поставленные теоретические и тестовые вопросы,

Шкала оценивания	Показатели	Критерии
	дополнительным материалом); 6. Знание нормативно-законодательной базы и терминологии курса	при которых обучающийся показал достаточный уровень знаний основного программного материала: освоение информации лекционного курса и учебных пособий, овладение понятийным аппаратом, методикой исследований при попытке анализа различных ситуаций.
Удовлетворительно		Задание решено с подсказками преподавателя. Задание решено в общем виде. Обучающийся показал средний уровень знаний основного программного материала, но не мог убедительно аргументировать свой ответ, ошибся в использовании понятийного аппарата, показал недостаточные знания литературных источников.
Неудовлетворительно		Задание не решено. Обучающийся продемонстрировал значительные пробелы в знаниях основного программного материала, не аргументировал свой ответ, показал неудовлетворительные знания понятийного аппарата и специальной литературы.

Критерии оценивания рефератов

Средство контроля	Критерии оценивания	Шкала оценивания
Реферат	Выполнены все требования к написанию и защите реферата: обозначена проблема и обоснована её актуальность, сделан краткий анализ различных точек зрения на рассматриваемую проблему и логично изложена собственная позиция, сформулированы выводы, тема раскрыта полностью, выдержан объём, соблюдены требования к внешнему оформлению, даны правильные ответы на дополнительные вопросы. Реферат раскрывает поднятую проблематику в полном объеме.	отлично
	Основные требования к реферату и его защите выполнены, но при этом допущены недочёты. В частности, имеются неточности в изложении материала; отсутствует логическая последовательность в суждениях; не выдержан объём реферата; имеются упущения в оформлении; на дополнительные вопросы при защите даны неполные ответы. В реферате имеются неточности и предметная область выступления раскрыта не в полной мере.	хорошо
	Имеются существенные отступления от требований к реферированию. В частности: тема освещена лишь частично; допущены фактические ошибки в содержании реферата или при ответе на дополнительные вопросы; во время защиты отсутствует вывод. В реферате не в полной степени раскрыт понятийный аппарат, имеются существенные неточности в процессе формирования выводов.	удовлетворительно
	Тема реферата не раскрыта, обнаруживается существенное непонимание проблемы. Тема реферата не раскрыта или выполнена не по существу ранее	неудовлетворительно

	поставленного вопроса. Реферат не сдан / доклад не сделан.	
--	---	--

Критерии оценивания тестов

Средство контроля	Критерии оценивания – процент положительных ответов	Шкала оценивания
Тестирование	90-100	отлично
	70-89	хорошо
	40-69	удовлетворительно
	< 39	неудовлетворительно

3. ОЦЕНОЧНЫЕ СРЕДСТВА ДЛЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ

Средства оценивания в ходе промежуточной аттестации:

- вопросы для экзамена;
- тестовые задания к экзамену.

Коды компетенции	Результаты освоения ОПОП	Перечень планируемых результатов обучения по дисциплине
ОПК-7	Способен использовать языки программирования и технологии разработки программных средств для решения задач профессиональной деятельности	ОПК-7.1. Выбирает структуры данных и разрабатывает алгоритмы решения задач профессиональной деятельности ОПК-7.2. Выполняет отладку и тестирование программ ОПК-7.3. Разрабатывает и реализовывает на языке высокого уровня алгоритмы решения профессиональных задач

3.1. Вопросы к экзамену:

1. Технология программирования: определение, цели и задачи. Отличие технологии от языка программирования.
2. Жизненный цикл программного обеспечения (ЖЦ ПО). Основные этапы по ГОСТ и международным стандартам.
3. Каскадная модель (Waterfall) разработки ПО: преимущества, недостатки и области применения.
4. Итеративные и инкрементальные модели разработки. Спиральная модель Б. Бозма.
5. Гибкие методологии (*Agile*). Манифест гибкой разработки и его основные принципы.
6. Фреймворк *Scrum*: роли (Владелец продукта, Скрам-мастер, Команда разработки), артефакты и события.
7. Экстремальное программирование (*XP*): практики парного программирования, рефакторинга и коллективного владения кодом.
8. Управление требованиями. Функциональные и нефункциональные требования. Спецификация требований к ПО (*SRS*).
9. Оценка трудозатрат на разработку ПО. Методы LOC и FP (*Function Points*). Модель COSOMO II.

10. Понятие качества программного обеспечения. Характеристики качества по стандарту ISO/IEC 25010.
11. Принципы объектно-ориентированного проектирования. Абстракция, инкапсуляция, наследование, полиморфизм.
12. Принципы SOLID: подробный разбор каждого принципа с примерами нарушений.
13. Паттерны проектирования: назначение и классификация (порождающие, структурные, поведенческие).
14. Порождающие паттерны: Singleton, Factory Method, Abstract Factory — сравнение и сценарии использования.
15. Структурные паттерны: Adapter, Decorator, Facade — решение задач интеграции и упрощения интерфейсов.
16. Поведенческие паттерны: Strategy, Observer, Template Method — управление алгоритмами и взаимодействиями.
17. Многоуровневая архитектура (*N-tier architecture*). Разделение ответственности между уровнями представления, бизнес-логики и данных.
18. Микросервисная архитектура: принципы декомпозиции, преимущества перед монолитом и проблемы распределенных систем.
19. Архитектурный стиль REST: ограничения, унифицированный интерфейс, идемпотентность методов.
20. Гексагональная (порты и адаптеры) и чистая (*Clean*) архитектура: изоляция бизнес-логики от внешних фреймворков и баз данных.
21. Уровни доступа к данным. Проблемы ручного написания SQL-кода (дублирование, сложность тестирования).
22. ORM (*Object-Relational Mapping*). Преимущества и недостатки использования Hibernate (*Java*) или Entity Framework (*C#/NET*).
23. Проблема N+1 запроса в ORM. Стратегии загрузки *Eager Loading* и *Lazy Loading*, использование JOIN FETCH.
24. Миграции схем базы данных. Инструменты управления версиями структуры БД (*Flyway*, *Liquibase*).
25. Работа с транзакциями. Свойства ACID. Уровни изоляции транзакций и их влияние на производительность и согласованность.
26. Оптимистические и пессимистические блокировки данных в высококонкурентной среде.
27. NoSQL базы данных. Классификация (ключ-значение, документные, колоночные, графовые) и критерии выбора.
28. Масштабируемость БД: шардирование (*Sharding*) и репликация. Горизонтальное и вертикальное масштабирование.
29. Системы контроля версий (VCS). Централизованные (*SVN*) и распределенные (*Git*) системы. Основные отличия.
30. Ветвление в Git. Стратегии работы с ветками (*GitFlow*, *Feature Toggles*, *Trunk-Based Development*).
31. Непрерывная интеграция (CI). Цели, задачи и типовой пайплайн сборки проекта.
32. Непрерывная доставка (CD) и непрерывное развертывание. Различия концепций.
33. Автоматизация сборки. Роль Maven/Gradle (*Java*) или MSBuild/NPM (*JS/C#*) в управлении зависимостями и жизненным циклом проекта.
34. Контейнеризация приложений. Docker как стандарт де-факто. Образ, контейнер, слой файловой системы.
35. Оркестрация контейнеров. Задачи Kubernetes: планирование, самовосстановление, масштабирование сервисов.
36. Статический анализ кода (*Linters*, *SonarQube*). Поиск уязвимостей и «запахов» кода до этапа компиляции.

37. Тестирование в пайплайне. Юнит-тесты, интеграционные тесты и мутационное тестирование (*Mutation Testing*).
38. Пирамида тестирования (*Testing Pyramid*). Баланс между модульными, сервисными и UI-тестами.
39. Разработка через тестирование (*TDD — Test Driven Development*). Красный-Зеленый-Рефакторинг.
40. Mock-объекты и стабсы (*Mocks & Stubs*). Использование Mockito (*Java*) или Moq (*.NET*) для изоляции юнит-тестов.
41. Покрытие кода тестами (*Code Coverage*). Виды покрытия (строки, ветвления) и критика метрики как единственного показателя качества.
42. BDD (*Behavior Driven Development*). Написание спецификаций на естественном языке (*Gherkin/Syntax*).
43. Нагрузочное тестирование. Понятия пропускной способности, времени отклика и точки насыщения (*bottleneck*).
44. Регрессионное тестирование. Стратегии минимизации затрат при проверке старого функционала после внесения изменений.
45. Безопасная разработка (*Secure SDLC*). Интеграция проверок безопасности на каждом этапе ЖЦ.
46. Уязвимости OWASP Top 10. Механизмы защиты от SQL-инъекций, XSS и незащищенного прямого обращения к объектам (*IDOR*).
47. Аутентификация и авторизация. Протокол OAuth 2.0 и стандарты JWT (*JSON Web Tokens*) для микросервисов.
48. Хеширование паролей. Алгоритмы bcrypt, scrypt, Argon2. Почему MD5 и SHA-1 недопустимы.
49. Управление секретами. Хранение ключей API и сертификатов вне исходного кода (*HashiCorp Vault*, переменные окружения).
50. Отказоустойчивость распределенных систем. Шаблоны Retry, Circuit Breaker, Bulkhead для обработки частичных отказов сети или сервисов.

3.2. Задания для экзамена:

6. **Сложность алгоритмов.** Дан массив целых чисел A размером N . Напишите фрагмент кода с асимптотической сложностью $O(N^2)$, который находит все пары индексов (i, j) , для которых сумма элементов $A[i] + A[j]$ является четным числом. (*Ожидаемый результат: реализация через два вложенных цикла*).
7. **Рекурсия vs Итерация.** Реализуйте функцию вычисления n -го числа Фибоначчи двумя способами: через классическую рекурсию и через цикл (*for* или *while*). Сравните их работу при $n = 40$ и объясните причину колоссальной разницы в производительности. (*Ожидаемый результат: демонстрация экспоненциальной сложности $O(2^n)$ у чистой рекурсии из-за повторных вызовов*).
8. **Хеш-таблицы.** У вас есть список из 1 000 000 уникальных телефонных номеров. Вам нужно проверить, содержится ли конкретный номер в этом списке. Почему использование линейного поиска неэффективно? Опишите алгоритм решения этой задачи с использованием хеш-множества (*HashSet*) за время $O(1)$ в среднем случае. (*Ожидаемый результат: обоснование выбора структуры данных на основе константного времени доступа*).
9. **Стеки и Очереди.** Дана строка, содержащая только символы $(,), \{, \}, [\text{ и }]$. Напишите алгоритм, определяющий, является ли входная строка корректной последовательностью скобок (каждая открывающая скобка должна закрываться тем же типом в правильном порядке). (*Ожидаемый результат: решение с использованием стека; пример: "{[]}" — валидно, "[]" — невалидно*).

10. **Графы.** В социальной сети пользователи представлены вершинами графа, а дружба между ними — ребрами. Предложите алгоритм (BFS или DFS), который находит кратчайшую цепочку знакомств («степень удаления») между пользователем А и пользователем Б. *(Ожидаемый результат: описание логики обхода в ширину для поиска минимального количества переходов).*

9. **Фабричный метод.** Разработайте систему отчетов, которая может экспортировать данные в форматах PDF, Excel и JSON. Создайте интерфейс IReportGenerator и реализуйте фабрику, которая возвращает нужный экземпляр класса в зависимости от строки-параметра ("pdf", "excel", "json"), скрывая детали инстанцирования от клиентского кода. *(Ожидаемый результат: код фабрики, избавляющий клиента от использования оператора new для конкретных классов).*

10. **Проблема N+1 запроса.** Имеется сущность «Автор», у которой есть коллекция «Книги». При выводе списка из 10 авторов и названий их книг ORM-фреймворк выполняет 1 запрос к таблице авторов и по 1 запросу на каждого автора для получения книг (всего 11 запросов). Какую технику загрузки (*Eager Loading / JOIN*) нужно применить, чтобы сократить количество запросов до двух? *(Ожидаемый результат: описание механизма включения связанных данных одним SQL-запросом с JOIN).*

11. **Паттерн Наблюдатель (Observer).** Объект «Точка продажи» (Subject) должен уведомлять объекты «Мобильное приложение» и «Дашборд администратора» (Observers) об изменении остатка товара на складе. Опишите структуру взаимодействия этих объектов без жесткой привязки друг к другу. *(Ожидаемый результат: схема интерфейса регистрации/дерегистрации подписчиков и рассылки событий notify).*

12. **Оптимизация транзакций.** Два банковских счета (А и Б) одновременно переводят деньги друг другу (А -> Б и Б -> А). Если операции чтения баланса и записи нового значения не атомарны, возникает race condition. Опишите механизм блокировок (*Pessimistic Locking*), который гарантирует целостность баланса. *(Ожидаемый результат: сценарий выполнения SELECT ... FOR UPDATE).*

13. **Нормализация БД.** В таблице заказов хранятся поля: OrderID, CustomerName, CustomerPhone, ProductID, ProductName, ProductPrice. Нарушены ли здесь правила нормализации? Если да, опишите шаги приведения схемы к третьей нормальной форме (3NF). *(Ожидаемый результат: выделение сущностей Customers и Products в отдельные таблицы для устранения дублирования имен).*

14. **NoSQL документные базы.** Вам нужно хранить профили пользователей, где каждый профиль имеет уникальный набор дополнительных полей (например, у одного есть хобби, у другого — данные о водительских правах). Почему реляционная БД будет неудобна (проблема разреженности таблиц)? Обоснуйте выбор MongoDB для этой задачи. *(Ожидаемый результат: сравнение хранения в виде плоской таблицы со множеством NULL против гибкого JSON/BSON документа).*

15. **Git Merge Conflict.** Два разработчика изменили одну и ту же строку в файле config.js и отправили изменения в разные ветки. При попытке слияния (*merge*) возникла ошибка. Опишите пошаговый алгоритм ручного разрешения этого конфликта и финальной фиксации изменений. *(Ожидаемый результат: последовательность команд git status, редактирование файла, git add, git commit).*

16. **Контейнеризация (Docker).** Приложение на Python зависит от специфической версии библиотеки libc, которая отличается от установленной на сервере разработчика. Составьте минимальный Dockerfile, гарантирующий идентичность среды исполнения на любой машине. *(Ожидаемый результат: базовый образ FROM python:3.9-slim, копирование requirements.txt, установка зависимостей).*

17. **CI/CD Pipeline.** Разработчик отправил код в ветку feature/login-form. Напишите логику простого GitHub Actions workflow, который должен автоматически запуститься, установить зависимости (npm install) и прогнать юнит-тесты (npm test), запретив мерж-

реквест при падении тестов. (Ожидаемый результат: структура YAML-файла с триггером on: [pull_request]).

18. **XSS (Cross-Site Scripting).** На сайте есть поле ввода комментария. Если пользователь введет `<script>alert('Hacked')</script>`, скрипт выполнится в браузере других посетителей страницы. Какое фундаментальное действие должен выполнить бэкенд перед сохранением и отображением этого текста, чтобы закрыть уязвимость? (Ожидаемый результат: HTML-экранирование спецсимволов, превращающее теги в `<script>`).

19. **Deadlock (Взаимная блокировка).** Поток T1 нужен доступ к ресурсу R1, а затем к R2. Поток T2 нужен доступ к R2, а затем к R1. Оба захватили по одному ресурсу и ждут освобождения второго вечно. Приведите правило нумерации ресурсов, которое гарантированно предотвратит эту ситуацию. (Ожидаемый результат: требование захватывать блокировки всегда в строго определенном глобальном порядке).

20. **Async/Await.** Программа должна скачать данные сразу с трех разных API-сервисов. Каждый запрос занимает 2 секунды. Сколько времени займет выполнение при синхронном подходе и сколько при использовании конкурентного ожидания (Promise.all или аналог)? (Ожидаемый результат: ~6 секунд синхронно против ~2 секунд асинхронно, так как запросы идут параллельно).

21. **RESTful API.** Клиент отправляет DELETE /api/users/123. Сервер успешно удалил пользователя. Что именно должен вернуть сервер в теле ответа согласно спецификации REST и почему? (Ожидаемый результат: статус 204 No Content, так как ресурс больше не существует и возвращать нечего).

22. **Логирование.** В высоконагруженном сервисе произошла ошибка деления на ноль. Почему использование конструкции print(error) внутри кода недопустимо в production-среде? Какие уровни логирования (DEBUG, INFO, WARN, ERROR) следует использовать для трассировки входа в функции и для записи критических сбоев? (Ожидаемый результат: print блокирует I/O поток процесса; ERROR для сбоев, DEBUG для трассировки).

23. **Рефакторинг «Божественного объекта» (God Object).** Класс UserManager содержит 50 методов: он шлет письма, рисует аватарки, считает налоги и пишет в базу данных. Примените принцип единственной ответственности (SRP) и опишите, на какие 4 отдельных класса/сервиса его следует разбить. (Ожидаемый результат: UserRepository (БД), EmailService (почта), TaxCalculator (логика), ProfileImageProcessor (графика)).

4. ОСНОВНЫЕ КРИТЕРИИ ОЦЕНИВАНИЯ ПРИ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ

Шкала оценивания уровня сформированности универсальной компетенций (экзамен)

Формируемые уровни освоения компетенций	Критерии оценивания	Шкала оценивания
Высокий уровень	Изложено правильное понимание вопроса, четко и самостоятельно дан исчерпывающий ответ, содержание раскрыто полно, профессионально, грамотно. Обучающимся усвоена взаимосвязь основных понятий дисциплины в их значении для приобретаемой профессии. Отражает успешное и систематическое применение навыков и умений по данной дисциплине в соответствии с ФГОС.	отлично

Базовый уровень	Изложено правильное понимание вопроса, дано достаточно подробное описание предмета ответа, приведены и раскрыты в тезисной форме основные понятия, относящиеся к предмету ответа. Ответ отражает полное знание учебно-программного материала, систематический характер знаний по дисциплине, а также наличие базового уровня овладения практическими умениями и навыками по данной дисциплине в соответствии с ФГОС	хорошо
Пороговый уровень	Ответ отражает теоретические знания основного учебно-программного материала в объеме, необходимом для дальнейшей учебы и предстоящей работы по профессии. Данная оценка может быть выставлена обучающемуся, допустившему неточности в ответе, но обладающими необходимыми знаниями для их устранения под руководством преподавателя, отмечен начальный уровень овладения практическими умениями и навыками по данной дисциплине в соответствии с ФГОС	удовлетворительно
Неудовлетворительный уровень	При ответе обучающегося обнаружено отсутствие знаний, умений и навыков и/или фрагментарные знания основного учебно-программного материала.	неудовлетворительно

Текущий контроль и промежуточная аттестация осуществляются в соответствии с «Положением о текущей и промежуточной аттестации обучающихся в Автономной некоммерческой организации «Образовательная организация высшего образования» «Университет экономики и управления».

Вид промежуточной аттестации – экзамен.

Форма проведения промежуточной аттестации – письменный экзамен.